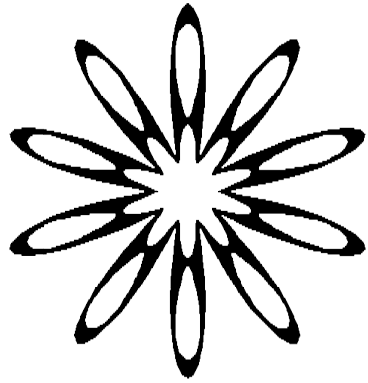
CHAPTER EIGHT

FILL ROUTINES

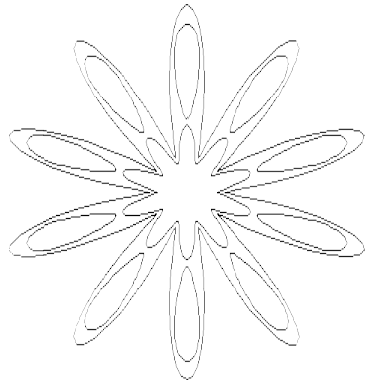
Although the operating system contains a very good selection of graphics routines it lacks several useful sets of commands such as a sprite routine. Another thing the operating system lacks is an efficient method of filling shapes with colour. In fact the only command the operating system provides for producing blocks of colour is the PLOT85 triangle fill routine. This command can be used to very good effect for producing solid circles, etcetera, by approximating the shape in question by a series of overlapping triangles. A time comes, however, when there is a need for a routine that will fill all the area within a boundary. Take the following shape for example:



8.1 A sample shape

This could be drawn using triangles but that would be very complicated. What we want to be able to do is to draw the outline and then call a machine code routine, first specifying a starting point inside the outline, that will fill the shape for us. The first thing we need is a program that will produce the outline of the shape we want to fill.

```
 10 REM Program to draw outline
 20 MODE0
 30 VDU23,1,0;0;0;0;
 40 P%=4
 50 FORA=0TOPI*2STEPPI/100
 60 R%=COS(A*10)*200+300
 70 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
 80 P%=5
 90 NEXT
100 P%=4
110 FORA=0TOPI*2STEPPI/80
120 R%=COS(A*10)*50+130
130 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
140 P%=5
150 NEXT
160 FORA=0TOPI*2STEPPI/5
170 P%=4
180 FORB=0TOPI*2STEPPI/30
190 C=A+SIN(B)*PI/32
200 R%=325+125*COS(B)
210 PLOTP%,SIN(C)*R%+640,COS(C)*R%+512
220 P%=5
230 NEXT,
```



8.2 A sample outline

A BASIC fill

Next we need a fill routine. Instead of writing the machine code program straightaway we will write an experimental BASIC one first. This is often a good idea with complicated machine code programs as it is easier to debug a BASIC program than a machine code on. Once the BASIC program works

it is relatively easy to convert it to machine code. For our BASIC fill program we will use a proccedure. The parameters we will need are the X and Y coordinates of a point within the outline which we've already drawn, and a definition of what the routine is to consider as the outline. The easiest way to do this is to specify a colour and say that any point of a different colour is part of the outline. The colour we specify, then, is the background colour of the shape. Here we can use the X and Y coordinates of 640 and 32 for the point within the outline; the background colour is zero. So our example will need the lines:

```
500 PROCfill(640,32,0)
510 END
```

We are going to use X% and Y% for the coordinates of the point we are looking at any one time − the fill 'cursor'. For convenience, then, we can define procedures for moving this cursor up, down, left and right one pixel at a time, which is necessary as we are exploring the area just around the current point. So that we can use these procedures in different graphics modes we need to specify a variable M%, which will differ for different modes, which is the amount by which we have to alter X% to move one pixel left or right. This variable should be defined at the beginning of the program (line 20) when we change mode. We replace the old line 20 by:

```
20 MODE0:M%=2
```

Our up, down, left and right procedures are:

```
2000 DEFPROCup:Y%=Y%+4:ENDPROC
2010 DEFPROCdown:Y%=Y%-4:ENDPROC
2020 DEFPROCleft:X%=X%-M%:ENDPROC
2030 DEFPROCright:X%=X%+M%:ENDPROC
```

We must also define a function that will return 'true' only if X% and Y% point to a pixel whose colour is the background colour (C%).

```
2040 DEFFNbackground=(POINT(X%,Y%)=C%)
```

Here, (POINT(X%,Y%)=C%) is true only if both sides of the = sign agree.

Now we can start to try and write the fill procedure itself. For this purpose let's take a simple example shape which we can use to work out the program. The cross represents the cursor position (though in the real program this will be invisible). The cursor is initially positioned at the point we have used to define the inside of the outline.

See diagram 8.3.

We are going to fill the shape with a series of horizontal lines, each one pixel thick. We will start from the bottom of the outline and work up. For this reason, we must first find a bottom point to start at. Imagine the outline as holding water: we need a place where water will collect − a hollow of some kind.

The first thing we can do is go vertically down until we hit the boundary.

```
1000 DEFPROCfill(X%,Y%,C%)
1010 PROCdown:IFFNbackground THEN1010
     ELSE PROCup
```

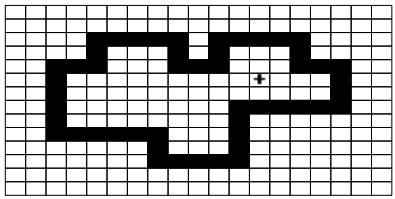
This moves the cursor down until the boundary is reached and then moves it up on pixel so that the cursor is on the bottom background pixel.

See diagram 8.4.

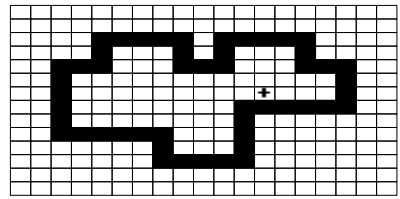
Next we can go left in a horizontal line until we hit the boundary. We then need to move right and, for reasons we shall see later, we need to make a copy of these coordinates in X1% and Y1%.

```
1020 PROCleft:IFFNbackground THEN1020
1030 PROCright:X1%=X%:Y1%=Y%
```

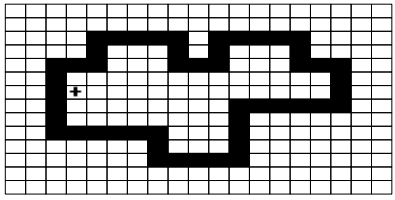
However, at this point (diagram 8.5) we can still go down. We need to check whether there is still



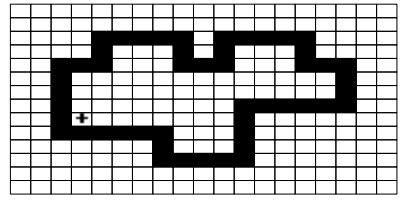
8.3



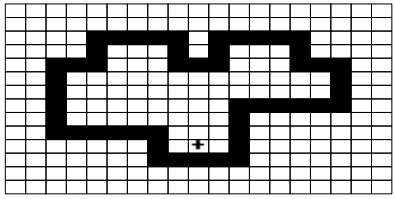
8.4



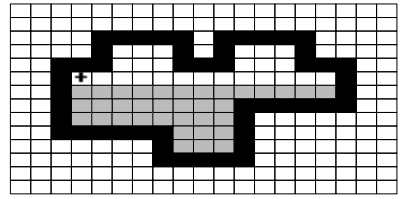
8.5



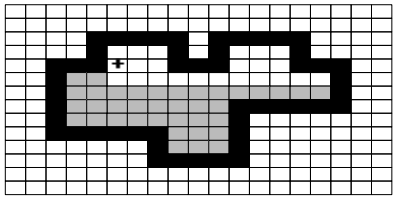
8.5



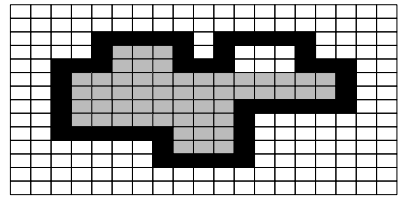
8.7



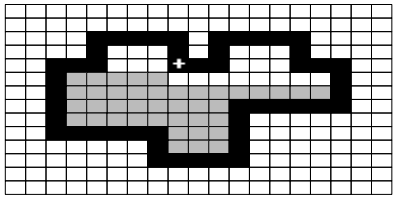
8.8



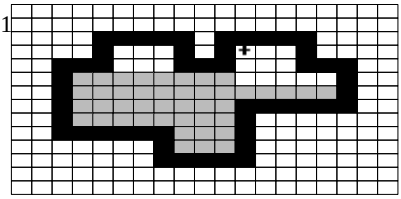
8.9



8.10



8.11



8.12

background below the cursor, and, if so, go back to line 1010 to find the bottom of the section.

```
1040 PROCdown:IFFNbackground THEN1010
```

By the time we get back to line 1020 again we have reached a new position.

See diagram 8.6.

At line 1020 we cannot go further left. At line 1040 there is no background below. We now have to go right in a horizontal line until we reach the boundary on the other side. As we go, we can check below for any background. If this is found then we can go back to line 1010 yet again to find the bottom.

```
1050 PROCup:PROCright:IFFNbackground THEN1040
```

By the time we get back to this point again we have found the bottom of the shape. The coordinates of the leftmost point of the bottom line are stored in X1% and Y1%. We must place the cursor back at this position.

```
1060 X%=X1%:Y%=Y1%
```

See diagram 8.7.

The obvious thing to do now is to fill this bottom line. However, as we do this we must check to see if we have finished filling the outline. As we fill each pixel of this line we must check the pixel above to see if it is background. If no background is found above then we have finished filling the shape. If some background is found then we must make a note in X1% and Y1% of the first pixel of background we find and set a flag to tell us that there is more to be filled above. What we will do is use the variable F% as a flag. At the start of filling the line, we set this to 0.

```
1060 X%=X1%:Y%=Y1%:F%=0
```

Then, when the first background pixel is found above, we set F% to I and store the coordinates in X1% and Y1%. Any further background points we find, F% will already be set to 1 and we will know not to alter X1% and Y1%. Then when the line is filled, if F% is still zero, no background has been found on the next line up and we have finished. Otherwise, we can move the cursor to the coordinates in X1% and Y1% and start to fill the next line.

So the first job to do, before we fill each pixel on the current line, is to moved up and check whether there is background above.

```
1070 PROCup:IFNOTFNbackground THEN1090
```

If background has been found and F% is still 0, then we must set F% to 1 and copy the cursor coordinates into X1% and Y1%.

```
1080 IFF%=0 F%=1:X1%=X%:Y1%=Y%
```

Next we can fill the pixel on the line we are currently working on.

```
1090 PROCdown:PLOT69,X%,Y%
```

Next we move right a pixel and, unless the boundary has been reached, go back and fill the next pixel.

```
1100 PROCright:IFFNbackground THEN1070
```

If F% is now 1 then we can go back and fill the next line up. We need to go back to line 1020 for this, to find the left-hand end of the line.

```
1110 IFF%=1 X%=X1%:Y%=Y1%:GOTO1020
```

Otherwise, we have finished and can return from the procedure.

```
1120 ENDPROC
```

This would appear to be perfectly good fill routine. However, it will only work until it reaches this point.

See diagram 8.8.

At line 1080 onwards, it will have reached this point.

See diagram 8.9.

It will keep a copy of this point in X1% and Y1% and set F% to 1. It will then ignore any other background that it finds on the line above. Thus when it fills the next line up, it will only fill the left-hand are of the shape. The final result, when the computer thinks it has finished, will be this:

See diagram 8.10.

To solve this problem we need to re-write lines 1070 to 1120 leaving gaps for lines we need to add:

```
1070 PROCup:IFFNbackground THEN1100
1090 GOTO1120
1100 IFF%=0 F%=1:X1%=X%:Y1%=Y%
1120 PROCdown:PLOT69,X%,Y%
1130 PROCright:IFFNbackground THEN1070
```

See diagram 8.11.

When the cursor reaches this point we need to set F% to 2. This shows that we have found the entire width of the first area. Thus if boundary is detected and F% is already 1, then we need to set F% to 2.

```
1080 IFF%=1 F%=2
```

See diagram 8.12.

When the cursor reaches this point, F% will be 2. Thus we know that the program must make a decision about which area to fill first. We will say that the program will keep a note of the position of the first area (already stored in X1% and Y1%) and carry on filling from where the cursor is, as if it is

starting a new horizontal line.

```
1110 IFF%=2THEN1170
```

However, in filling the second area, the program might come across another decision point and have to keep a note of a second set of coordinates.

What we need is a buffer in which we can place a whole list of coordinates. We can use an array for this. We are unlikely to need to keep a note of more than 64 sets of coordinates so we can dimension an array 64 by 2. Because we can use entry zero in an array, the following command will be sufficient.

```
5 DIM S%(63,1)
```

We can then say that the first point saved on this buffer will have X and Y coordinates stored in S%(0,0) and S$(0,1) and so on.

We also need a pointer to keep track of how many points we have kept a note of at any time. For this we can use the variable P%. This needs to be set to zero on entry to the procedure.

```
1000 DEFPROCfill(X%,Y%,C%):P%=0
```

When we reach a decision point, we must place the coordinates of the first area (currently in X1% and Y1%) in the first clear entry of the buffer and add one to the pointer P%. Then we can move down a pixel, back to the line we were originally filling. We need to set F% to 0, so that the program carries on looking for areas above the current line that need filling, and then go back to line 1070.

```
1170 S%(P%,0)=X1%:S%(P%,1)=Y1%:P%=P%+1
1180 PROCdown:F%=0:GOTO1070
```

If the program does not reach a decision point then it will finish filling the line it is on. At this stage, if there is more to be filled on the next line up, F% will be either 1 or 2 and the coordinates of the area to be filled will be in X1% and Y1%

```
1140 IFF%>0 X%=X1%:Y%=Y1%:GOTO1020
```

If note then the program has reached a dead-end. However, there may still be some points stored in the buffer that need to be explored. If P% is 0 then we have finished the entire job and can return from the procedure.

```
1150 IFP%=0 ENDPROC
```

If P% is not 0, then we need to subtract one from the pointer, P%; remove the most recent point from the stack; and start filling from this point.

```
1160 P%=P%-1:X%=S%(P%,0):Y%=S%(P%,1):GOTO1020
```

This completes the fill program. To make it easier to type in, here is the complete program.

```
  5 DIM S%(63,1)
 10 REM Program to draw outline.
 20 MODE0:M%=2
 30 VDU23,1,0;0;0;0;
 40 P%=4
 50 FORA=0TOPI*2STEPPI/100
 60 R%=COS(A*10)*200+300
 70 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
 80 P%=5
 90 NEXT
100 P%=4
110 FORA=0TOPI*2STEPPI/80
120 R%=COS(A*10)*50+130
130 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
140 P%=5
150 NEXT
160 FORA=0TOPI*2STEPPI/5
170 P%=4
180 FORB=0TOPI*2STEPPI/30
190 C=A+SIN(B)*PI/32
200 R%=325+125*COS(B)
210 PLOTP%,SIN(C)*R%+640,COS(C)*R%+512
220 P%=5
230 NEXT,
```

```
  500 PROCfill(640,32,0)
  510 END
 1000 DEFPROCfill(X%,Y%,C%):P%=0
 1010 PROCdown:IFFNbackground THEN1010 ELSE
PROCup
 1020 PROCleft:IFFNbackground THEN1020
 1030 PROCright:X1%=X%:Y1%=Y%
 1040 PROCdown:IFFNbackground THEN1020
 1050 PROCup:PROCright:IFFNbackground THEN1040
 1060 X%=X1%:Y%=Y1%:F%=0
 1070 PROCup:IFFNbackground THEN1100
 1080 IFF%=1 F%=2
 1090 GOTO1120
 1100 IFF%=0 F%=1:X1%=X%:Y1%=Y%
 1110 IFF%=2THEN1170
 1120 PROCdown:PLOT69,X%,Y%
 1130 PROCright:IFFNbackground THEN1070
 1140 IFF%>0 X%=X1%:Y%=Y1%:GOTO1020
 1150 IFP%=0 ENDPROC
 1160 P%=P%-1:X%=S%(P%,0):Y%=S%(P%,1):GOTO1020
 1170 S%(P%,0)=X1%:S%(P%,1)=Y1%:P%=P%+1
 1180 PROCdown:F%=0:GOTO1070
 2000 DEFPROCup:Y%=Y%+4:ENDPROC
 2010 DEFPROCdown:Y%=Y%-4:ENDPROC
 2020 DEFPROCleft:X%=X%-M%:ENDPROC
 2030 DEFPROCright:X%=X%+M%:ENDPROC
```

A machine code fill

If you run the BASIC program you will find that it takes about eight minutes to fill our outline. This is clearly impracticable for serious use so we need to re-write the routine in machine code. The quickest way to do this is to simply covert the program, line for line, into machine code.

First we must define the zero page locations we will use to replace each of the BASIC variables.

&70 and &71	X% low and high
&72 and &73	Y% low and high
&74	colour of current pixel
&75	C% − background colour
&76	P% − stack pointer
&77	F% − decision flag
&78 and &79	X1% low and high
&7A and &7B	Y1% low and high

We need a procedure to assemble the machine code.

```
1000 DEFPROCass
1010 DIMmc%350
1020 FORpass%=0TO2STEP2
1030 P%=mc%
1040 [OPTpass%
```

Now we can start writing the routine. We can improve the speed by disabling interrupts during the fill. Then we can look at the first line of the BASIC which defined the procedure and set P% to zero. This becomes:

```
1050 .fill    SEI
1060          LDA #0
1070          STA &76
```

Where we used procedures in the BASIC examples for left, right, up and down, we can use subroutines in machine code with the same labels. For the function BACKGROUND we can write a subroutine called BACKGR which sets the zero flag only if the pixel under the cursor is of the background colour.

The next line of the BASIC was:

```
1010 PROCdown:IFFNbackground THEN1010
     ELSE PROCup
```

This, then, becomes the following piece of machine code:

```
1080 .loop1   JSR down
1090          JSR backgr
1100          BEQ loop1
1110          JSR up
```

Next we had the lines:

```
1020 PROCleft:IFFNbackground THEN1020
1030 PROCright:X1%=X%:Y1%=Y%
```

This becomes:

```
1120 .loop2   JSR left
1130          JSR backgr
1140          BEQ loop2
1150          JSR right
1160          LDA &70
1170          STA &78
1180          LDA &71
1190          STA &79
1200          LDA &72
1210          STA &7A
1220          LDA &73
1230          STA &7B
```

Then the lines:

```
1040 PROCdown:IFFNbackground THEN1010
1050 PROCup:PROCright:IFFNbackground THEN1040
1060 X%=X1%:Y%=Y1%:F%=0
```

This becomes:

```
1240 .loop3   JSR down
1250          JSR backgr
1260          BEQ loop1
1270          JSR up
1280          JSR right
1290          JSR backgr
1300          BEQ loop3
1310          LDA &78
1320          STA &70
1330          LDA &79
1340          STA &71
1350          LDA &7A
1360          STA &72
1370          LDA &7B
1380          STA &73
1390 .loop4   LDA #0
1400          STA &77
```

Note the label LOOP4. By jumping to here when we reach the equivalent of line 1180 of the BASIC, we can save two commands.

Next we had:

```
1070 PROCup:IFFNbackground THEN1100
1080 IFF%=1 F%=2
1090 GOTO1120
1100 IFF%=0 F%=1:X1%=X%:Y1%=Y%
1110 IFF%=2THEN1170
```

This becomes

```
1410 .loop5   JSR up
1420          JSR backgr
1430          BEQ skip1
1440          LDA &77
1450          CMP #1
1460          BNE point
1470          LDA #2
1480          STA &77
1490          BNE point
1500 .skip1   LDA &77
1510          BNE skip2
1520          LDA #1
1530          STA &77
1540          LDA &70
1550          STA &78
1560          LDA &71
1570          STA &79
1580          LDA &72
1590          STA &7A
1600          LDA &73
1610          STA &78
1620 .skip2   LDA &77
1630          CMP #2
1640          BEQ skip5
```

We then had the line:

```
1120 PROCdown:PLOT69,X%,Y%
```

This codes into machine code using the VDU25 equivalent of the PLOT command.

```
1650 .point   JSR down
```

```
1660          LDA #25
1670          JSR &FFEE
1680          LDA #69
1690          JSR &FFEE
1700          LDA &70
1710          JSR &FFEE
1720          LDA &71
1730          JSR &FFEE
1740          LDA &72
1750          JSR &FFEE
1760          LDA &73
1770          JSR &FFEE
```

```
1130 PROCright:IFFNbackground THEN1070
```

Becomes:

```
1780          JSR right
1790          JSR backgr
1800          BEQ loop5
```

Then we had the lines:

```
1140 IFF%>0 X%=X1%:Y%=Y1%:GOTO1020
1150 IFP%=0 ENDPROC
```

This becomes:

```
1810          LDA &77
1820          BEQ skip3
1830          LDA &78
1840          STA &70
1850          LDA &79
1860          STA &71
1870          LDA &7A
1880          STA &72
1890          LDA &7B
1900          STA &73
1910          JMP loop2
1920 .skip3   LDA &76
1930          BNE skip4
1940          CLI
1950          RTS
```

Next we have the first mention of the buffer. For this we can use the stack. The points can be pushed on the stack, four bytes each, and pulled off again in the reverse order. Location &76 can be used to keep track of how many points are on the stack, as before.

The next line from the BASIC was:

```
1160 P%=P%-1:X%=S%(P%,0):Y%=S%(P%,1):GOTO1020
```

This, then, becomes:

```
1960 .skip4   DEC &76
1970          PLA
1980          STA &73
1990          PLA
2000          STA &72
2010          PLA
2020          STA &71
2030          PLA
2040          STA &70
2050          JMP loop2
```

The next two lines from the BASIC were:

```
1170 S%(P%,0)=X1%:S%(P%,1)=Y1%:P%=P%+1
1180 PROCdown:F%=0:GOTO1070
```

These become:

```
2060 .skip5   LDA &78
2070          PHA
2080          LDA &79
2090          PHA
2100          LDA &7A
2110          PHA
2120          LDA &7B
2130          PHA
2140          INC &76
2150          JSR down
2160          JMP loop4
```

Finally we have the five subroutines. The four move routines code rather obviously.

```
2000 DEFPROCup:Y%=Y%+4:ENDPROC
2010 DEFPROCdown:Y%=Y%-4:ENDPROC
2020 DEFPROCleft:X%=X%-M%:ENDPROC
2030 DEFPROCright:X%=X%+M%:ENDPROC
```

These become:

```
2170 .up      LDA &72
2180          CLC
2190          ADC #4
2200          STA &72
2210          BCC uskip
2220          INC &73
2230 .uskip   RTS

2240 .down    LDA &72
2250          SEC
2260          SBC #4
2270          STA &72
2280          BCS dskip
2290          DEC &73
2300 .dskip   RTS

2310 .left    LDA &70
2320          SEC
2330          SBC #M%
2340          STA &70
2350          BCS lskip
2360          DEC &71
2370 .lskip   RTS

2380 .right   LDA &70
2390          CLC
2400          ADC #M%
2410          STA &70
2420          BCC rskip
2430          INC &71
2440 .rskip   RTS
```

For the last routine, we need to use OSWORD 9 (the machine code equivalent of POINT). For this, a parameter block must be set up with the X coordinate in the first two bytes (low then high) and

the Y coordinate in the next two bytes (low then high). The OSWORD routine then places the colour of the pixel in the fifth byte of the parameter block.
We already have the X coordinate in &70 and &71, and the Y coordinate in &72 and &73; we also want the colour returned in &74. So, all we need to do is set the X and Y registers (low, high) to point to location &70, set the accumulator to 9 and call OSWORD. We can then compare the colour returned in &74 with the background colour in &75. If they are the same then the zero flag will be set as required.

```
2450 .backgr  LDA #9
2460          LDX #&70
2470          LDY #0
2480          JSR &FFF1
2490          LDA &74
2500          CMP &75
2510          RTS
2520 ]
2530 NEXT
2450 ENDPROC
```

This finishes the machine code but we still need a BASIC procedure that calls the routine.

```
950 DEFPROCfill(X%,Y%,C%)
960 !&70=X%+Y%*&10000
970 ?&75=C%
980 CALLfill
990 ENDPROC
```

As an example try adding the BASIC example section (lines 10 to 510) from the previous program. We don't now need line 5 as we are using the machine stack; but we need to call the assembly procedure once M% has been set.

```
25 PROCass
```

IF you run this you will find that it is about eight times faster than the BASIC version. This routine will work in any mode so long as M% is set correctly

before the assembler procedure is called.

A faster fill

For most purposes this routine is fast enough. However, there will be occasions when a still faster routine is needed. The things which slow down this routine are the operating system calls. The routines we have used are all written so as to be as flexible as possible. However, this slows them down considerably. If we are prepared to limit the fill routine to working in only one mode then we can write our own versions of the operating system routines which will be much faster.

To do this we need to use a different method of storing the coordinates of a point.

As an example of what is possible we can write a Mode 0 fill routine that will fill our example shape. As Mode 0 is a two-colour mode this shouldn't too difficult. To simplify the program further still, we will assume that the background colour is zero.

Each pixel in Mode 0 is represented by one bit in the memory. So instead of X% and Y% from our BASIC example, we need a different method of determining which pixel we are looking at. The best way to do this is to use &70 and &71 to store the address of the byte in memory; and another byte, at &72, to describe which bit of that byte we are interested in. This byte will contain a one in the bit corresponding to the bit we are interested in and zeroes in all the other bits. The reason for this is that we can then load the byte from memory into the accumulator and AND it with the contents of &72 to leave either zero if the pixel was black or not zero if the pixel was white. This technique is sometimes called 'bit masking'.

For convenience and speed we will also keep the character column position (that is, the number of pixels along divided by eight) in &73. This will make it easier to check whether we have moved off the edge of the screen.

The first change we need to make to our machine code program is to add a routine at the beginning to convert the X and Y coordinates into an address, a bit mask, and a column number. On entry to the routine we must specify that &78 and &79 contain

the X coordinate, and &7A and &7B contain the Y coordinate, in pixels, where the top left-hand corner of the screen is (0,0). This means that the BASIC procedure for calling the fill routine now becomes:

```
960 DEFPROCfill(X%,Y%)
970 !&78=X%DIV2+&10000*(255-Y%DIV4)
980 CALLfill
990 ENDPROC
```

The first job for the machine code (after disabling interrupts and setting P% to zero) is to work out the address of the byte. This will be:
&3000 + 640*Y%DIV8 + X%AND&FFF8 + Y%MOD8

So the first section of the routine looks like this:

```
.fill    SEI          \ Disable interrupts.
         LDA #0       \ Set P% to 0
         STA &76      \
         LDA &7A      \ Find
         AND #&F8     \ Y% DIV 8
         LSR A        \ times 2
         LSR A        \ for look O.S.
         TAX          \ Y% MOD 8
         LDA &7A      \
         AND #7       \
         CLC          \
         ADC &C376,X  \ look up 640
         STA &70      \ times table
         LDA &C375,X  \
         CLC          \
         ADC #&30     \ add &3000
         STA &71      \
         LDA &78      \ add X% AND
         AND #&F8     \ &FFF8
         CLC          \
         ADC &70      \
         STA &70      \
         LDA &71      \
         ADC &79      \
         STA &71      \

(1050-1290)
```

If you are not sure how this works, look at chapter 10 on sprites, as this routine uses a similar section.

Next we must calculate the 'bit mask'. This must have the value &80 if X%AND7 is zero and &01 if it is seven. To calculate it we can place the value &80 in location &72 and then shift it right the number of times specified by X%AND7.

```
        LDA #&80
        STA &72
        LDA &78
        AND #7
        TAX
        BEQ skipA
.loopA  LSR &72
        DEX
        BNE loopA
.skipA  ...
```

(1300-1380)

Lastly we need to calculate the column number. This will be the X coordinate divided by eight. As the X coordinate cannot be larger than 639 we can divide it two, twice, to get a number that cannot be larger than 159. Then we only need to shift the low byte for the last division by two. This saves one command.

```
.skipA  LSR &79
        ROR &78
        LSR &79
        ROR &78
        LSR &78
        LDA &78
        STA &73
```

(1390-1450)

From here on the program is similar to the previous machine code version. As the X and Y coordinates are stored as four bytes from &70 to &73 the com-

mands X1%=X%:Y%=Y% etcetera, will still be the same. The next thing we need to change is where a point is actually plotted. As we have assumed that the background colour is zero, the foreground colour must be one. This means that to plot a point we need to set the relevant bit in the memory to one. To do this we can simply load the byte into the memory, OR it with the bit mask and store it back in the memory. So the code after the label POINT becomes:

```
.point   JSR down
         LDY #0
         LDA (&70),Y
         ORA &72
         STA (&72),Y
         JSR right
         ...
```

Finally we need to rewrite the routines UP, DOWN, LEFT, RIGHT and BACKGR. Here, UP must first check if the point is off the top of the screen. If so, then it must check the least significant three bits of the address. If these are zero then we need to go up a whole character line. Otherwise we need to take one away from the address.

```
.up      LDA &71
         CMP #&30
         BCC up2
         LDA &70
         AND #7
         BNE up1
         LDA &70
         SEC
         SBC #&79
         STA &70
         LDA &71
         SBC #2
         STA &71
         RTS
.up1     DEC &70
.up2     RTS
```

```
(2480-2630)
```

DOWN is similar.

```
.down    LDA &71
         BMI down2
         LDA &70
         AND #7
         CMP #7
         BNE down1
         LDA &70
         CLC
         ADC #&79
         STA &70
         LDA &71
         ADC #2
         STA &71
         RTS
.down1   INC &70
.down2   RTS
```

```
(2650-2800)
```

LEFT needs to check that the pixel is on the screen and, if so, shift the bit mask left a bit; if the carry is then set, it needs to subtract eight from the address, decrement the column number and set the bit mask to one; RIGHT is similar.

```
.left    LDA &73
         BMI left1
         ASL &72
         BCC left1
         LDA #1
         STA &72
         DEC &73
         LDA &70
         SEC
         SBC #8
         STA &70
         LDA &71
         SBC #0
         STA &71
.left1   RTS
```

```
.right   LDA &73
         CMP #80
         BEQ right1
         LSR &72
         BCC right1
         LDA #128
         STA &72
         INC &73
         LDA &70
         CLC
         ADC #8
         STA &70
         LDA &71
         ADC #0
         STA &71
.right1  RTS

(2820-3130)
```

Lastly we need to deal with BACKGR. This must first check that the point is on the screen. If so, then it can load the byte from the memory, AND it with the bit mask and return to the main program with zero in the accumulator only if the pixel is background.

```
.backgr  LDA &73
         BMI off
         CMP #80
         BEQ off
         LDA &71
         BMI off
         CMP #&30
         BCC off
         LDY #0
         LDA (&70),Y
         AND &72
         BNE off
         LDA #0
         RTS
.off     LDA #1
         RTS
```

(3150-3300)

We now have an effective routine. Here is a complete listing of it:

```
  5 PROCass
 10 REM Program to draw outline
 20 MODE0
 30 VDU23,1,0;0;0;0;
 40 P%=4
 50 FORA=0TOPI*2STEPPI/100
 60 R%=COS(A*10)*200+300
 70 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
 80 P%=5
 90 NEXT
100 P%=4
110 FORA=0TOPI*2STEPPI/80
120 R%=COS(A*10)*50+130
130 PLOTP%,640+SIN(A)*R%,512+COS(A)*R%
140 P%=5
150 NEXT
160 FORA=0TOPI*2STEPPI/5
170 P%=4
180 FORB=0TOPI*2STEPPI/30
190 C=A+SIN(B)*PI/32
200 R%=325+125*COS(B)
210 PLOTP%,SIN(C)*R%+640,COS(C)*R%+512
220 P%=5
230 NEXT,
500 PROCfill(640,32)
510 END
960 DEFPROCfill(X%,Y%)
970 !&78=X%DIV2+&10000*(255-Y%DIV4)
980 CALLfill
990 ENDPROC
1000 DEFPROCass
1010 DIMmc%450
1020 FORpass%=0TO2STEP2
1030 P%=mc%
1040 [OPTpass%
1050 .fill    SEI
1060          LDA #0
1070          STA &76
1080          LDA &7A
```

```
1090          AND #&F8
1100          LSR A
1110          LSR A
1120          TAX
1130          LDA &7A
1140          AND #7
1150          CLC
1160          ADC &C376,X
1170          STA &70
1180          LDA &C375,X
1190          CLC
1200          ADC #&30
1210          STA &71
1220          LDA &78
1230          AND #&F8
1240          CLC
1250          ADC &70
1260          STA &70
1270          LDA &71
1280          ADC &79
1290          STA &71
1300          LDA #&80
1310          STA &72
1320          LDA &78
1330          AND #7
1340          TAX
1350          BEQ skipA
1360 .loopA   LSR &72
1370          DEX
1380          BNE loopA
1390 .skipA   LSR &79
1400          ROR &78
1410          LSR &79
1420          ROR &78
1430          LSR &78
1440          LDA &78
1450          STA &73
1460 .loop1   JSR down
1470          JSR backgr
1480          BEQ loop1
1490          JSR up
1500 .loop2   JSR left
1510          JSR backgr
1520          BEQ loop2
```

```
1530          JSR right
1540          LDA &70
1550          STA &78
1560          LDA &71
1570          STA &79
1580          LDA &72
1590          STA &7A
1600          LDA &73
1610          STA &7B
1620 .loop3   JSR down
1630          JSR backgr
1640          BEQ loop1
1650          JSR up
1660          JSR right
1670          JSR backgr
1680          BEQ loop3
1690          LDA &78
1700          STA &70
1710          LDA &79
1720          STA &71
1730          LDA &7A
1740          STA &72
1750          LDA &7B
1760          STA &73
1770 .loop4   LDA #0
1780          STA &77
1790 .loop5   JSR up
1800          JSR backgr
1810          BEQ skip1
1820          LDA &77
1830          CMP #1
1840          BNE point
1850          LDA #2
1860          STA &77
1870          BNE point
1880 .skip1   LDA &77
1890          BNE skip2
1900          LDA #1
1910          STA &77
1920          LDA &70
1930          STA &78
1940          LDA &71
1950          STA &79
1960          LDA &72
```

```
1970          STA &7A
1980          LDA &73
1990          STA &7B
2000 .skip2   LDA &77
2010          CMP #2
2020          BEQ skip5
2030 .point   JSR down
2040          LDY #0
2050          LDA (&70),Y
2060          ORA &72
2070          STA (&70),Y
2080          JSR right
2090          JSR backgr
2100          BEQ loop5
2110          LDA &77
2120          BEQ skip3
2130          LDA &78
2140          STA &70
2150          LDA &79
2160          STA &71
2170          LDA &7A
2180          STA &72
2190          LDA &7B
2200          STA &73
2210          JMP loop2
2220 .skip3   LDA &76
2230          BNE skip4
2240          CLI
2250          RTS
2260 .skip4   DEC &76
2270          PLA
2280          STA &73
2290          PLA
2300          STA &72
2310          PLA
2320          STA &71
2330          PLA
2340          STA &70
2350          JMP loop2
2360 .skip5   LDA &78
2370          PHA
2380          LDA &79
2390          PHA
2400          LDA &7A
```

```
2410          PHA
2420          LDA &7B
2430          PHA
2440          INC &76
2450          JSR down
2460          JMP loop4
2480 .up      LDA &71
2490          CMP #&30
2500          BCC up2
2510          LDA &70
2520          AND #7
2530          BNE up1
2540          LDA &70
2550          SEC
2560          SBC #&79
2570          STA &70
2580          LDA &71
2590          SBC #2
2600          STA &71
2610          RTS
2620 .up1     DEC &70
2630 .up2     RTS
2640
2650 .down    LDA &71
2660          BMI down2
2670          LDA &70
2680          AND #7
2690          CMP #7
2700          BNE down1
2710          LDA &70
2720          CLC
2730          ADC #&79
2740          STA &70
2750          LDA &71
2760          ADC #2
2770          STA &71
2780          RTS
2790 .down1   INC &70
2800 .down2   RTS
2810
2820 .left    LDA &73
2830          BMI left1
2840          ASL &72
2850          BCC left1
```

```
2860          LDA #1
2870          STA &72
2880          DEC &73
2890          LDA &70
2900          SEC
2910          SBC #8
2920          STA &70
2930          LDA &71
2940          SBC #0
2950          STA &71
2960 .left1   RTS
2970
2980 .right   LDA &73
2990          CMP #80
3000          BEQ right1
3010          LSR &72
3020          BCC right1
3030          LDA #128
3040          STA &72
3050          INC &73
3060          LDA &70
3070          CLC
3080          ADC #8
3090          STA &70
3100          LDA &71
3110          ADC #0
3120          STA &71
3130 .right1  RTS
3140
3150 .backgr  LDA &73
3160          BMI off
3170          CMP #80
3180          BEQ off
3190          LDA &71
3200          BMI off
3210          CMP #&30
3220          BCC off
3230          LDY #0
3240          LDA (&70),Y
3250          AND &72
3260          BNE off
3270          LDA #0
3280          RTS
3290 .off     LDA #1
```

```
3300          RTS
3310 ]
3320 NEXT
3330 ENDPROC
```

Notice that in this form this program takes up nearly 5K of memory. Disc users, in particular, may have trouble using this routine (though if it is typed in exactly as above it should just fit). If you need a shorter version you can just take out all the spaces and put more than one command per line and this should approximately have the length of the assembly code.

Fill routines are not the most efficient way of filling large areas with colour. Where possible use the triangle plot command for large areas and keep the fill routine for filling small areas. If used to fill the whole screen, the routine above will take just over 30 seconds. However, for our example shape it only takes about four seconds.