All Things Sprite and Beautiful

Liven up your programs with some exciting sprite displays. Alan Wrigley

If your programs need a little extra something to make them look exciting, why not add a touch of gratuitous graphic artistry? Displaying and manipulating sprites is really quite simple, and the routines presented here will enable you to do it in style. In this article we will be describing procedures to plot sprites on the screen in various ways including scaling. A second article in the near future will introduce other transformation processes including rotation and shearing. These procedures can easily be incorporated into your own single-tasking programs. A future article in the Wimp Topics series will cover the use of sprites in multi-tasking windows.

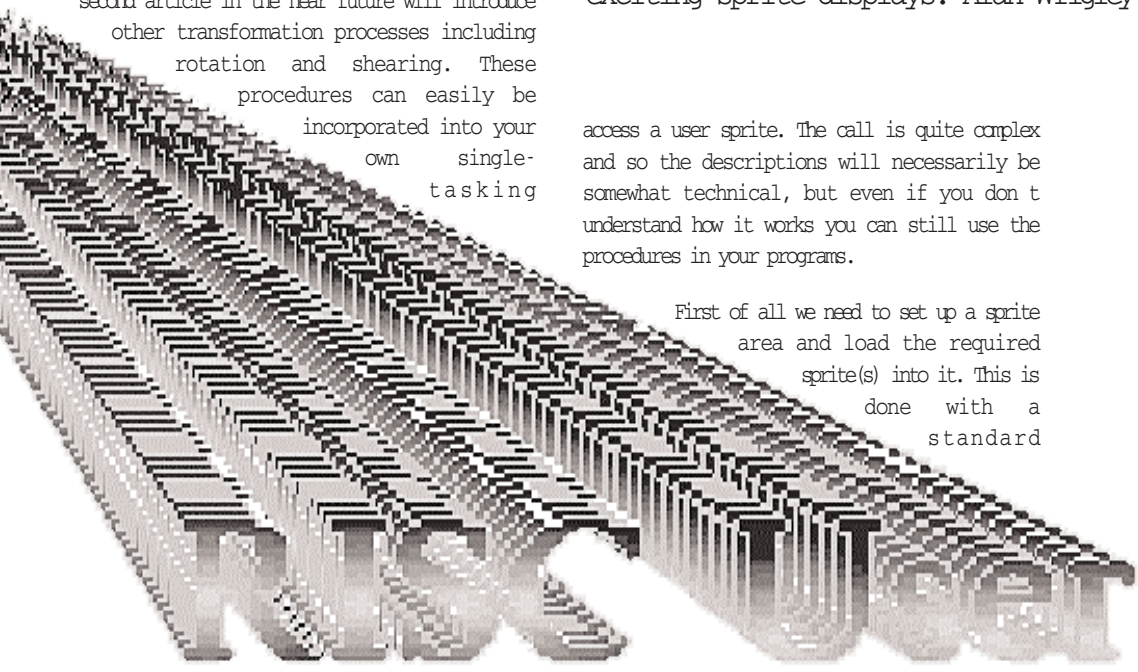

Figure 1. A simple sprite can make an attractive display

The heart of all these routines is the SWI OS_SpriteOp (SWI &2E) which can perform a multitude of operations on a chosen sprite, depending on the reason code supplied in R0. The particular reason code we will be making most use of is 56, which plots a sprite according to certain transformation processes specified in the other parameters to the call. We will assume that the sprite to be manipulated is in a user sprite area kept within the program's memory, and so when we refer in the article to OS_SpriteOp 56, we really mean 256+56, since 256 must be added to the reason code to access a user sprite. The call is quite complex and so the descriptions will necessarily be somewhat technical, but even if you don't understand how it works you can still use the procedures in your programs.

First of all we need to set up a sprite area and load the required sprite(s) into it. This is done with a standard procedure as follows:

```
DEF PROCload_sprites(spfile$)
SYS "OS_File",5,spfile$ TO ,,,,size%
DIM sp% size%+4:!sp%=size%+4:sp%!8=16
SYS "OS_SpriteOp",256+10,sp%,spfile$
ENDPROC
```

This assumes that spfile$ holds the pathname of the sprite file. OS_File 5 finds the length of the file, and a block of memory to hold it (plus 4 bytes for the first word of the header) is dimensioned at sp%. The file is then loaded with OS_SpriteOp 10.

Next we need to find out the size of the sprite and relate it to the screen size in the current mode. This is done as follows (assuming that spr$ contains the name of the sprite):

```
DEF PROCread_sprite_info
LOCAL x%,y%
SYS "OS_SpriteOp",256+40,sp%,spr$ TO ,
,,xpix%,ypix%
SYS "OS_ReadModeVariable",-1,4 TO ,,x%
```

```
SYS "OS_ReadModeVariable",-1,5 TO ,,y%
spx%=xpix%<<x%:spy%=ypix%<<y%
ENDPROC
```

Here we have used OS_SpriteOp 40 to read the sprite size in pixels, and converted these to OS units (in spx% and spy%) by reading the xeig and yeig factors for the current mode (these factors indicate how many bits to shift the co-ordinate to the right to get the number of OS units). Note that if you are displaying more than one sprite, this process must be done for each sprite, and separate values of spx% and spy% held for each.

IT S ALL IN THE PLOT

Once this is done we can think about putting the sprite on the screen. Find a suitable sprite of a reasonable size - a piece of text as shown in Figure 1 can make an attractive display. The particular one shown here is a mode 15 sprite with a mask, 326x26 pixels in size. To start with, let s simply display the sprite at whatever scale we choose. This can be done with the following procedure:

```
DEF PROCscale(spr$,x%,y%,scx,scy)
dest%!0=x%*256
dest%!4=(y%+spy%*scy)*256
dest%!8=(x%+spx%*scx)*256
dest%!12=(y%+spy%*scy)*256
dest%!16=(x%+spx%*scx)*256
dest%!20=y%*256
dest%!24=x%*256:dest%!28=y%*256
SYS "OS_SpriteOp",256+56,sp%,spr$,1,,8
,dest%
ENDPROC
```

The procedure is called with five parameters, the first of which is the name of the sprite. After this come the x and y co-ordinates of the bottom left-hand corner of the plot position, and finally the x and y scale factors.

At this point some explanation of OS_SpriteOp 56 is required. The parameters we have used here are as follows:

- R0 = 56
- R1 = pointer to sprite area
- R2 = pointer to sprite name
- R3 = flags (see text)
- R4 = only used if R3 bit 1 is set
- R5 = GCOL action (+8 if mask to be used)
- R6 = pointer to destination co-ordinate block
- R7 = pixel translation table (0 if none)

Some of the registers can hold different parameters depending on the state of the flags in R3. Here we have set bit 0, meaning treat R6 as a destination co-ordinate block pointer . We will make use of other settings in a subsequent article.

The destination co-ordinate block is 32 bytes long, and contains the co-ordinates of the area in which the sprite will be plotted, in 1/256th OS units. This doesn t have to be a rectangle, though it must be a parallelogram. This enables you to plot the sprite so that it appears slanted. Because of this, four sets of co-ordinates are needed rather than two, and these are placed into the destination block in the following order: top left (x,y), top right (x,y), bottom right (x,y), bottom left (x,y). The co-ordinates required are calculated by multiplying the width or height of the sprite by the appropriate scale factor, and again by 256 to get the correct values in 1/256th OS units as required by the call.

The following program will therefore display a sprite called MySprite , contained in a file adfs::0.$.Sprites, at OS co-ordinates 100,300, with both its width and height 50% greater than the original:

```
MODE n:CLG
DIM dest% 31
PROCload_sprites("adfs::0.$.Sprites")
spr$="MySprite"
PROCread_sprite_info
PROCscale(spr$,100,300,1.5,1.5)
```

Note that we have not performed any colour translation in this simple example, so the mode number in the first line (the value of n) must have the same number of colours as the sprite.

Try experimenting with this simple example to achieve some different effects. For example, you could alter either or both of the scale factors. If you alter one and not the other, the proportions of the sprite will change. You could also try slanting the text by adding an offset to two of the co-ordinates. For example, the bottom left and bottom right x co-ordinates could be shifted 50 units to the right by altering lines 6 and 8 of PROCscale as follows:

```
dest%!16=(x%+50+spx%*scx)*256
dest%!24=(x%+50)*256:dest%!28=y%*256
```

Note that if the resulting co-ordinates do not form a parallelogram the SWI will generate an error.

Now try calling PROCscale in a loop as follows:

```
FOR n=1 TO 60
 PROCscale(spr$,100+6*n,800-6*n,0.2+n/
70,0.2+n/70)
NEXT
```

This plots the sprite repeatedly, each time shifting the starting co-ordinates and the size by a small amount. The result is shown in Figure 1.

FLIPPING SPRITES

Now let's rotate the sprite. To do this we use another couple of reason codes for OS_SpriteOp. Reason code 33 flips a sprite about its x axis, while 47 does the same for the y axis. So the following two procedures will flip the sprite horizontally (PROCflipx) or vertically (PROCflipy). Note that the alteration is made to the sprite as held in the sprite area, so it will remain flipped whenever it is displayed in future unless you carry out a reverse operation before you want to use it the right way round again.

```
DEF PROCflipx(spr$)
SYS "OS_SpriteOp",256+33,sp%,spr$
ENDPROC

DEF PROCflipy(spr$)
SYS "OS_SpriteOp",256+47,sp%,spr$
ENDPROC
```

You could now plot three copies of the sprite, with the second flipped vertically and the third flipped on both axes, as follows:

```
PROCscale(spr$,200,600,1.2,1.2)
PROCflipx(spr$)
PROCscale(spr$,200,400,1.2,1.2)
PROCflipy(spr$)
PROCscale(spr$,200,200,1.2,1.2)
```

Don't forget to call PROCflipx and PROCflipy again afterwards if you want the sprite the right way round in future.

SINE HERE

Finally for this month we will introduce three routines that create patterns from multiple plottings of a sprite. PROCswirl creates a whorl, PROCsinewave does exactly what it suggests, while PROCsnake produces a serpentine object. Figure 2 shows the results.

```
DEF PROCswirl(spr$,xcent%,ycent%,sangl
e%,eangle%,stepang%,swrad%)
IF stepang%=0 ERROR 1,"Angle step size
cannot be zero"
FOR omega%=sangle% TO eangle% STEP ste
pang%
 swrad%-=2
 x%=xcent%+swrad%*COSRADomega%
 y%=ycent%+swrad%*SINRADomega%
 swf=5000/swrad%/swrad%
 PROCscale(spr$,x%,y%,swf,swf)
NEXT
ENDPROC
```

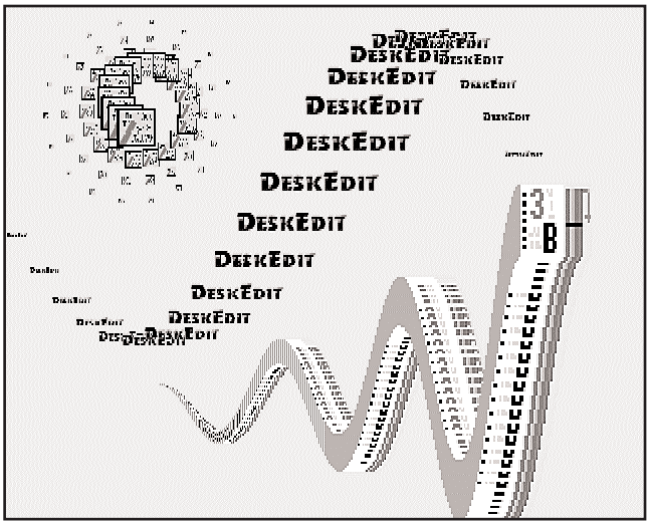

Figure 2.
A combination of swirl, sinewave and snake effects

```
DEF PROCsinewave(spr$,magnif,ycent%,am
p%,freq,step%)
mag=0.1
FOR sangle%=-180 TO 0 STEP step%
 x%=sangle%*freq+640
 y%=ycent%+amp%*SINRADsangle%
 PROCscale(spr$,x%,y%,mag,mag)
 mag+=magnif
NEXT
FOR sangle%=step% TO 180 STEP step%
 x%=sangle%*freq+640
 y%=ycent%+amp%*SINRADsangle%
 PROCscale(spr$,x%,y%,mag,mag)
 mag-=magnif
NEXT
ENDPROC

DEF PROCsnake(spr$)
x%=100
```