Using Colour on The RISC PC

David Spencer explores the exciting new range of colours and screen modes offered by the Risc PC.

It s a fair guess that one of the first things that you are going to want to do after unpacking your brand new Risc PC, is play with all those new screen modes and colours it offers, and show it off to your friends. In this article we will explore how to use the new modes and colours, and give some simple programs that show just what the machine s capable of.

ALL SIZES OF SCREEN

One nice feature of the new VIDC20 video controller is that it is much more relaxed about available screen resolutions than its VIDC1 predecessor. Without getting too technical, this is because it contains circuits to let it generate any speed of video dot clock it cares for (within reason). Couple this feature with the fact that almost all Risc PCs will be attached to a multi-scan monitor, and the world s your oyster.

Unfortunately, the time still hasn t come when you can say I want a display that is 851 by 506 pixels because the attached monitor will still impose restrictions on what can and cannot be done. Instead, there is a definition file for each model of monitor which holds a list of what display modes are available. Because there is a unique file for each model, the definitions can be highly tuned, for example ensuring that the display is centred and flicker free.

Resolution	Frame rates (Hz)	Menu
240 x 352	70	No
320 x 250	70	No
320 x 256	70	No
384 x 288	70	No
480 x 352	70	Yes
640 x 200	70	No
640 x 250	70	No
640 x 256	70	No
640 x 352	70	No
320 x 480	60,72,75	No
360 x 480	60	No
640 x 480	60,72,75	Yes
800 x 600	56,60,72,75	Yes
1024 x 768	60,70*,75*	Yes

The resolutions and frame rates marked with a *

Acorn are supplying two monitors with the Risc PC: the AKF60 and the AKF85. These offer the following resolutions:
As you can see, the table is quite lengthy and includes all the standard Acorn resolutions plus PC VGA, SVGA and XVGA formats. You can also see that some resolutions are offered at a number of different frame rates. This allows software (or the user) to specify a higher refresh rate if needed, to provide a more flicker-free display. The column labelled Menu in the table indicates which resolutions appear by default on the mode selector menu. The others can be selected using the method about to be described.

CHOOSING A MODE

With such a choice of resolutions, and the possibility of them changing between monitor models, the concept of mode numbers falls down somewhat. Instead, modes are better specified using a description string. This was described in the article New modes for future machines a couple of issues back, but we will recap here.

The format of a mode specifier string is, for example:
X640,Y480,C32K,EX1,EY1,F60 The numbers after X and Y specify, as you might guess, the resolution. The C setting selects the number of colours (32K or 32T for 32000, 16M for 16 million). Alternatively, G16 or G256 can be used to specify grey-scale modes. The values of EX and EY are the so-called X and Y Eigen factors. These specify the number of times to double

the actual resolution to give the resolution in OS units as used by your drawing commands. A value of 1 means that there are two OS units to each actual pixel, 0 would be one-to-one, and 2 would indicate four OS units to each pixel. The settings that a program chooses for these are rather arbitrary, provided it remembers what they are! However, the Wimp always draws system text at a size of 8 by 16 OS units. Therefore the Eigen factors can be manipulated to change the apparent size of text. Finally, F is the desired frame rate, in Hertz. Omitting it results in the default rate for the chosen resolution.

As an example of mode strings from Basic,

```
    MODE "X800,Y600,C256,EX1,EY1,F60"
```

would select a 256-colour SVGA mode, whilst:

```
    MODE "X640,Y256,C16,EX1,EY2"
```

would select good old mode 12. (You can still use the command MODE 12, but you should really avoid the use of mode numbers in new programs if they are only going to run on the Risc PC.)

When you change mode in this way, Basic actually makes a call to the command *WimpMode. This means that as well as setting the mode, it also sets up the Wimp palette for modes with fewer than 256 colours. You can use a VDU 20 to restore the default palette if you want. You should also perform a VDU 26 to reset the default screen windows, followed by a CLS. This is because the Wimp sets the display area to match the command window that it opens when running a non-Desktop program.

WHAT MODE AM I IN?

Obviously, there are times when a program might like to find about the current screen mode. This can be done in Basic by using MODE as a function which returns a pointer to a block of memory holding information about the mode. The procedure PROCmodeinfo given in listing 1, which works on all versions of RISC OS, will return the screen resolution in both pixels and OS units, and the number of available colours. The procedure returns the information by way of five variables passed to it, these holding the X and Y pixel resolutions, the X and Y OS Unit resolutions and the number of colours respectively.

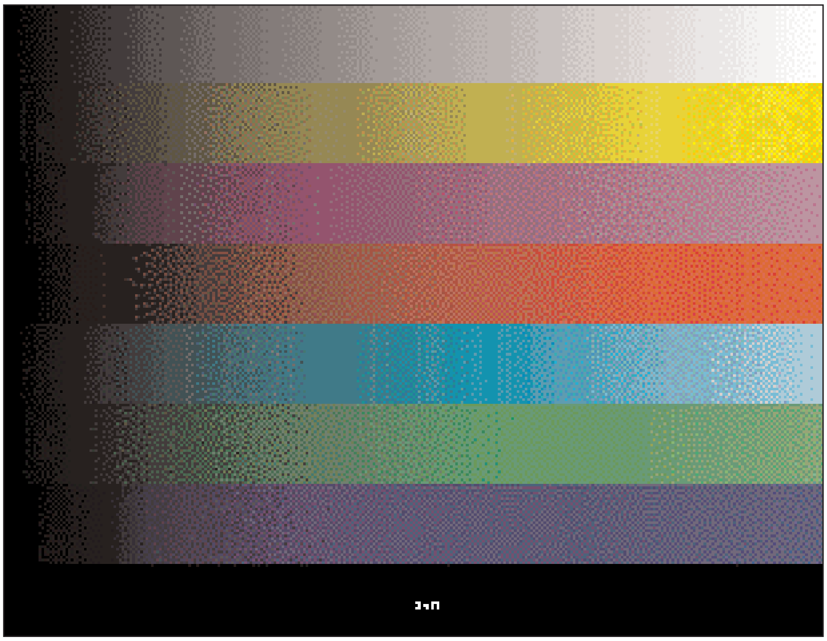
Listing 1

```
    DEF PROCmodeinfo(RETURN x, RETURN y, RETURN
osx, RETURN osy, RETURN c)
    LOCAL M%,ex%,ey%
    SYS "OS_ReadModeVariable",-1,11 TO ,,x:x+=1
    SYS "OS_ReadModeVariable",-1,12 TO ,,y:y+=1
    SYS "OS_ReadModeVariable",-1,4 TO ,,ex%
    SYS "OS_ReadModeVariable",-1,5 TO ,,ey%
    SYS "OS_ReadModeVariable",-1,3 TO ,,c:c+=1
    IF c=64 THEN c=256
    IF c=65536 THEN c=32768
    IF c=0 THEN c=2^24
    osx=x<<ex%:osy=y<<ey%
    ENDPROC
```

A FINAL NOTE ON MODES

Before moving on, there is one further point about the new mode system that you must know about. Because of the way that Basic works, using MODE with a mode specifier string will change the Wimp mode as well. This means that if you run a non-Desktop program from within the Desktop, then when it completes and returns to the Desktop, it will not automatically restore the correct mode. It is therefore up to your program to make a note of the mode when it is run, and restore it afterwards. This is good practice even when not running from the Desktop, as it returns the machine to the state it was in before running your program.

Unfortunately, the new mode system doesn t look favourably on sticking M%=MODE at the start of the program and MODE M% at the end. This is because MODE as a function will return a pointer to a block of data detailing the mode, and not a simple number. You can t be sure that this data will still be there by the time you quit, and even if it was, it contains no information on the palette settings. Instead, you should use the functions given in listing 2. PROCsavemode should be called before changing mode for



The demo program in action

the first time, and PROCrestoremode should be called before the program quits.

Listing 2

```
     DEF PROCsavemode
     LOCAL p%,g%,i%,x%,A%,B%,C%,D%,E%,F%
     DIM _myblk% 16
     SYS "OS_ReadModeVariable",-1,11 TO ,,!_myblk%
     SYS "OS_ReadModeVariable",-1,12 TO ,,_myblk%!4
     SYS "OS_ReadModeVariable",-1,4 TO ,,_myblk%?8
     SYS "OS_ReadModeVariable",-1,5 TO ,,_myblk%?9
     SYS "OS_ReadModeVariable",-1,3 TO ,,_myblk%!12
     IF _myblk%!12=255 OR (_myblk%!12>0 AND _myblk
%!12<4)
     THEN_myblk%?10=ASC"G" ELSE
_myblk%?10=ASC"C"
     IF _myblk%!12=15 THEN
     DIM p% 80
     !p%=&E1A09005:p%!4=&EF000034:p%!8=&E1B0F00E

A%=0:B%=&11000010:C%=p%+16:D%=0:E%=7:F%=&23
     CALLp%
     g%=1:FOR i%=p%+16 TO p%+79 STEP 4
     IF (i%?1<>i%?2) OR (i%?2<>i%?3) g%=0
     NEXT
     IF g% THEN _myblk%?10=ASC"G"
     ENDIF
     ENDPROC
     :
     DEF PROCrestoremode
     LOCAL a$
     a$="X"+STR$(!_myblk%+1)+",Y"+STR$(_myblk
%!4+1)+","+CHR$_myblk%?10
     CASE _myblk%!12 OF
     WHEN 1,3,15,255: a$+=STR$(_myblk%!12+1)
     WHEN 63: a$+="256" WHEN 65535: a$+="32K"
     WHEN &FFFFFFFF: a$+="16M"
     ENDCASE
     a$+=",EX"+STR$_myblk%?8+",EY"+STR$_mybl k%?9
     MODE a$
     ENDPROC
```

An alternative to this is to bypass the Wimp and set the mode at a lower level. This can be done by using MODE <block>, where <block> is the address of a list of data called a mode specifier block. The format of this was described in the article New modes for future machines already mentioned.

USING ALL THEM COLOURS

OK, we ve selected a screen mode with n million colours, but how do we use them.

Well, for all but 32000 and 16 million colour modes, you can if you wish still use the same forms of GCOL and COLOUR as on older machines. Not only that, but you also have the bonus that if you use the command COLOUR p,r,g,b to set the RGB values of a palette entry, they can now be set to any value between 0 and 255. In other words, there is always a palette of 16 million colours to choose from even if you can only have a handful of them on screen at once.

To handle 32000 and 16 million colour modes, new forms of GCOL and COLOUR have been added to Basic. These take the forms:

```
    COLOUR r,g,b
    GCOL [<action>,]r,g,b
```

where <action> is the graphics plotting mode (OR, AND, EOR etc.)

These are analogous to the corresponding forms that specify a colour number, but instead specify an actual colour by its RGB value. So, for example,

```
    COLOUR 200,200,50
```

would set the text colour to a pale yellow, whilst

```
    GCOL 200,200,50
```

would do the same for the graphics colour.

A rather nice feature of these new forms of COLOUR and GCOL is that they use the ColourTrans module to set the colour. This means that they can be used in lower colour-depth modes (for example 16-colour modes) and they will pick the closest colour in the palette and use that.

The astute among you will probably have noticed that as each RGB component can be any number between 0 and 256, there is nowhere to add 128 to set the background colour. And indeed, you re partly right. For GCOL, the background can be set by adding 128 to the plotting action, so for example:

```
    GCOL 131,255,0,0
```

would set the background to red with Exclusive-OR plotting. However, the story for COLOUR is not so simple. If you want to set an absolute RGB background colour here, you have to use the following call:

```
    SYS "ColourTrans_SetTextColour",
    (B%<<24)+(G%<<16)+(R%<<8),,,128
```

where R%, B% and G% are the RGB values in the range 0-255. (COLOUR r,g,b uses this call with the final parameter set to 0 and not 128 to adjust the foreground colour.)

I guess the reason that this situation has arisen is that there is already a four-parameter version of COLOUR used to define the palette, and there was therefore no room to include a foreground/background toggle without confusing the poor interpreter.

The operation of the POINT function which reads the colour of a pixel at a given point on the screen has also been forced to change to allow for the new wider range of colours. In modes of up to 256 colours, it behaves as before, while in 32000 and 16 million colour modes, it returns 32 bits indicating the RGB value of the colour at that position. The meaning of these bits differs for the two colour ranges:

32000 colours

Bits 0-4	Red
Bits 5-9	Green
Bits 10-14	Blue
Bits 15-31	Reserved (set to 0)

16 million colours

Bits 0-7	Red
Bits 8-15	Green
Bits 16-23	Blue
Bits 24-31	Reserved (set to 0)

To show what all these colours can do, the program in listing 3 draws horizontal bars for the primary and secondary colours; red, green, blue, magenta, cyan, yellow and white, in 256 graduations from black to full intensity. This is only actually achievable in 16 million colour modes, although the use of ColourTrans by Basic ensures that in other modes the results are the best they can be. The program runs through 16, 256, 32000 and 16 million colour modes to demonstrate this point. Before running this, you should change into a 16 million colour 800 x 600 mode in the Desktop to ensure that there is enough free memory, because MODE will default to a lower colour depth and spoil the effect if it can t claim enough RAM. For those without a Risc PC, the result is shown in figure 1.

Listing 3

```
   10 REPEAT
   20 READ C$
   30 MODE "X800,Y600,C"+C$+",EX1,EY1"
   40 VDU 26:CLS:OFF
   50 FOR s=0 TO 255
   60 PROCblock(0,0,1,s)
   70 PROCblock(0,1,0,s)
   80 PROCblock(1,0,0,s)
   90 PROCblock(0,1,1,s)
  100 PROCblock(1,0,1,s)
  110 PROCblock(1,1,0,s)
  120 PROCblock(1,1,1,s)
  130 NEXT
  140 PRINT TAB(48,70)C$
  150 *FX15 1
  160 A=GET
  170 UNTIL C$="16M"
  180 END
  190 DATA 16,256,32K,16M
  200 :
  210 DEF PROCblock(r,g,b,s)
  220 GCOL r*s,g*s,b*s
  230 RECTANGLE FILL s*6,150*(r*4+g*2+b),6,150
  240 ENDPROC
```

caption

Finally, if you want to be really impressive, it is relatively easy to include the rather nice colour selector used by the likes of Draw in your own Wimp programs. You will however need to wait until next month to find out how to do this.