Pico Tutorial

Learn how to use pico in 32 easy steps.

Gerard J. Holzmann

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Pico is a small expression language for picture composition. In all its simplicity, it is a remarkably powerful tool for making transformations to pictures. This memo explains how to use it.

[Draft for an AT&T Technical Memorandum, 22 January 1985, 4 pgs.]

January 22, 1985

Pico Tutorial

Learn how to use pico in 32 easy steps.

Gerard J. Holzmann

Bell Laboratories
Murray Hill, New Jersey 07974

Black&White

So you want to edit a picture, maybe combine it in interesting ways with one or more other pictures that you have stored in files. Let us assume that you want to make the average of two pictures named ‘rob’ and ‘pjw’, just to see what it looks like. Assuming that you have access to the framebuffer (that is: you have a login on ‘kwee’), this is what you type:

```
$ pico rob pjw
1: bw ($rob+$pjw)/2
2:
```

The number followed by a colon and a space is the prompt for commands we will assume here. (Pico’s normal prompt is an arrow: ‘−> ’.) The first command consists of a prefix ‘bw’ (black&white) followed by an expression. In this expression, names preceded by a dollar sign are taken to be picture files, eg as specified on the command line. A long name such as ‘/n/kwee/t0/face/512x512x8/rob’ is abbreviated in the dollar argument to its basename ‘rob’ (the part following the last slash). Not all file names have to be provided on the command line. We can, for instance, append a new file ‘doug’ by typing:

```
2: a doug
```

and we can check which files are currently open by typing ‘f’:

```
3: f
$0 old         color  resident
$1 rob  black/white resident
$2 pjw  black/white resident
$3 doug black/white absent
```

The numbers serve as a shorthand for the full filenames. Typing ‘$1’ therefore is equivalent to typing ‘$rob’. We will use both notations ‘$1’ and ‘$rob’ below. The first line ‘$0’ is just a placeholder that refers to the screen. Ignore it for the moment. We have a black and white image on the screen that is an average of the two files ‘rob’ and ‘pjw’. To see ‘rob’ separately we could type:

```
4: bw $1
```

but that is hardly an inspiring procedure. Let’s just take the left half or rob’s face combined with the right half of pjw’s:

```
5: bw (x < 256) ? $rob : $pjw
```

The variable ‘x’ is predefined and gives the x-coordinate of the pixels on the screen. Since 512 pixels will fit on one scanline a pixel in the middle of the screen has an x-coordinate of 256. The expression above is a conditional of the form:

```
condition ? iftrue : iffalse .
```

For every screen position where the condition holds the ‘iftrue’ part of the expression applies and everywhere else the ‘iffalse’ part applies. Two other predefined variables are ‘y’ (the y-coordinate of the destination) and ‘i’ (512*y+x). Since $0 refers to the screen we can now turn the picture on the screen upside

down by typing:

```
6: bw $0[x, 511−y]
```

or equivalently:

```
7: bw $0[512*512−i]
```

or, since ‘old’ is a synonym for $0:

```
8: bw old[x, 511−y]
```

All pixels in the black&white picture are internally represented by a value in the range 0..255, where 0 means black and 255 means white. To reverse an image, therefore it would suffice to subtract the current value of each pixel from its maximum value 255. Getting very bold we can turn the picture on its side, and make it negative by saying:

```
9: bw 255 − old[y,x]
```

Note that we swapped x and y to turn the picture on its side. Nothing can stop us now:

```
10: bw (x<512/3)?$1:(x>512*2/3)?$2:3*((x−512/3)*$2+(512*2/3−x)*$1)/512
```

fades ‘rob’ slowly into ‘pjw’.

Of course, nothing tells us that we should only type things that make sense. All normal arithmetic operators from C have been implemented in pico. The ‘^’ operator, for instance, makes an ‘exclusive or’ of its operands. Thus,

```
11: bw x^y^$rob
```

is a particularly striking effect,

```
12: bw ($rob > $pjw) ? ($rob*$pjw)/255 : $doug
```

looks promising but is rather disappointing. But then,

```
13: bw $rob +(255 − $rob[x+2, y+2])
```

is an attempt to make a relief that again works relatively well.†

Color

A complete picture specifies pixel values for each of three separate color channels: red, green, and blue. For a black&white picture the three channels are equal, but there is of course no need to restrict ourselves to that here. The prefix ‘bw’ simply means that the expression will apply equally to all three color channels. By replacing ‘bw’ with one of ‘red’, ‘grn’ or ‘blu’ we can write each channel separately. So:

```
14: red $rob
15: grn $rob
16: blu 255−$rob
```

will write ‘rob’ on the red and green channels, and its negative on the blue channel. We could have used a shorthand for the first two lines:

```
17: rg $rob .
```

There are three such shorthands: ‘rg’, ‘gb’, and ‘br’. We could also have written a separate value to each channel with the ‘rgb’ prefix and a ‘composite’ in square brackets:

```
18: rgb [ $rob, $rob, 255−$rob ]
```

The channels are addressed by the three fields of the composite in the order: [red, green, blue]. Omitting to specify a composite with an ‘rgb’ prefix typically results in only the red channel being written. In fact, you

† Be warned that an out of range pixel array index produces an undefined pixel value.

probably don't want to know but the result modulo 256 (E%256) will be mapped on the red channel, the same result E shifted right 8 places ((E/256)%256) will be mapped on the green channel and the remaining bits ((E/(256*256))%256) are written to the blue channel. A nice color picture can be created therefore by making up a range of big numbers:

```
19: rgb x*y*64
```

but for normal picture processing the color composite shown on line 18 is probably more useful.

There is only one more thing to know: to select the pixel values of the different channels as values in the expressions the suffixes '.red', '.grn' and '.blu' can be used:

```
20: rgb [ old.grn, old.blu, old.red ]
```

rotates the colors of the picture. And, of course, you can freely combine the color suffixes with array indexing: $rob.blu is just a shorthand for either '$rob[x, y].blu' or '$rob[i].blu' and the variables 'x', 'y', and 'i' in these can be replaced by just any monstrous C-style expression.

The Color Maps

If you must, the color map can be set with one of the commands cmap (all channels), cmapr (red channel), cmapg (green), or cmapb (blue). The color map is a mapping table that assigns a pixel brightness in the range 0..255 to a pixel value as stored in the pixel array, also in the range 0..255. The variable 'i' is used to index the color map. For instance:

```
21: cmap 255−i
```

will very quickly make a negative, and

```
22: cmap i
```

turns the picture back to normal. To fake a color picture in a black and white image you can try:

```
23: cmapr (i <= 85) ? i : 0
24: cmapg (i > 85 && i < 170) ? i : 0
25: cmapb (i >= 170) ? i : 0
```

Note however that changing the color map only changes the appearance of the picture on the monitor, not its definition in the picture array. Alas, you cannot refer to the pictures themselves to set the color map.

Undo, Read, Write, Windows

There are two flavors of 'undo': one to undo the effect of the last command, either per channel ('ur', 'ug', or 'ub') or for all three channels simultaneously (plain 'u'). Then there is one command to undo the effect of the whole session and restore the display to the state in which it was when you started: 'U'.

The 'append' command, to add files to the list of dollar arguments was discussed before. 'r' instead of 'a' will put the file into $0, that is on the screen. To save the current state of the display in a file, use:

```
26: w filename
```

To close a no longer used file and free up some memory for others, say:

```
27: c $doug
```

giving the dollar argument, instead of the filename. You can check with 'f' which is which when in doubt.

The framebuffer will only show 480 out of a maximum of 512 scanlines. To see the bottom 32 line you can say:

```
28: up
```

and to restore the display:

```
29: down
```

To restrict the updates to a window on the screen you can set:

```
30: window 10 100 200 300
```

which makes a window with origin at (x,y) = (10,100), 200 pixels wide and 300 pixels deep. To check the effect you can use ‘box’ instead of window, which will outline the window until you hit return.

```
31: box 10 100 200 300
```

To exit pico, finally, use the ‘quit’ command:

```
32: q
```

Some frequently occuring constants have been predefined: X is the coordinate of the leftmost pixel on a scanline (511), Y the coordinate of the lowest scanline (also 511, even though only 480 scanlines are visible at any one time), and the constant Z gives the value of the brightest pixel (255).

Credits

The expressions submitted to pico are, in the worst case, evaluated 262,144 times per screen update. Pico could hardly be called an interactive tool if these updates would take more than a few seconds. To make this possible Ken Thompson wrote an on-the-fly compiler that converts each expression into highly optimized machine code. Clearly, this compiler is easily the most important part of pico. The expression language itself is still the subject of much debate and it is likely to grow substantially.