Raster Graphics in Plan 9

Tom Duff
td@plan9.att.com

1. Introduction

Section 9 of the Plan 9 Programmer's Manual describes a file format for storing raster images (*picfile* (9.6)) and a suite of commands and libraries for manipulating them.

Binaries of the picture manipulation programs are located in the directory /$cputype/bin/fb. Examples in this discussion assume that you have run

```
bind -b /$cputype/bin/fb /bin
```

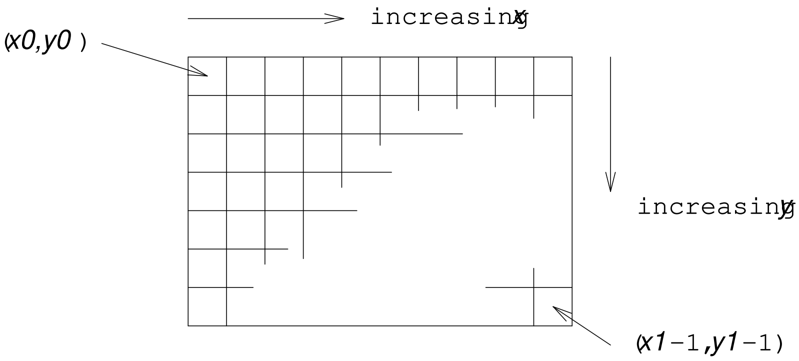
to merge the fb commands with the regular contents of /bin. For example, you might put this command in $home/lib/profile. Alternatively, you can prefix any *program* name from Section 9 with the string fb/, running it as fb/*program*.

2. Pictures and picfiles

For present purposes, a picture is a rectangular array of *n*-byte pixels. A pixel of a full-color picture has three bytes, encoding the brightness of a CRT's red, green and blue phosphors at a particular spot on the screen. A monochrome picture needs only one byte per pixel to describe its brightness. In either case, a byte containing 0 is black and 255 is full intensity.

As a storage-saving compromise, some pictures are represented by one-byte pixels with an associated *color map*, a table of 256 3-byte entries. In this case the pixel values are used as indices to look up 3-byte colors in the color map.

The bounding rectangle ($x0$, $y0$, $x1$, $y1$) of a picture is represented as in the *graphics* (2) section of the Programmer's Manual. (See the figure.) That is, ($x0$, $y0$) is the coordinate of the upper-leftmost pixel of the picture and ($x1-1$, $y1-1$) is the coordinate of the lower-rightmost pixel. X increases from left to right, and Y increases from top to bottom. Thus $x0 \le x1$ and $y0 \le y1$. Pixels in the order they are scanned out on the CRT face are in row-major order in the pixel array.



The picture file format described in *picfile* (9.6) is fairly simple. Such a *picfile* encodes a rectangular array of *n*-channel pixel records, each channel being a single byte. The file contains a textual header that describes the dimensions and encoding of the picfile. At the end of the header is an empty line two newline characters in succession. Following the header is a binary encoding (possibly compressed) of the pixel data in scan-line order. The header may indicate that the picfile has a colormap, in which case a burst of 256 3-byte records separates the header from the pixel data.

The lines of the picfile header are *attribute* = *value* pairs. The value of the WINDOW= attribute specifies the dimensions of the picfile, and the CHAN= attribute names the channels. Most often you will see CHAN=rgb for full-color pictures or CHAN=m for monochrome or colormapped pictures. Some pictures have CHAN=rgba. These are full color with an ± channel. Most of the objects that we make pictures of are not rectangular, contrary to the WINDOW= attribute. The ± channel gives us a way of describing a picture's shape. Think of ± as a fraction between 0 and 1 (represented by 255) that indicates whether the picture covers the pixel or not. Fractional ± values indicate pixels in which the background should shine through, because the foreground is translucent or only partly covers the pixel. The paper Compositing digital images, by Thomas Porter and Tom Duff (1984 SIGGRAPH Proceedings, pp. 253-258), describes how ± is used to control anti-aliased picture compositing operations.

Here is a sample image, called pjw:

This is its header:

```
TYPE=runcode
WINDOW=0   0  320  240
NCHAN=1
CHAN=m
COMMAND=resample     320  pjw
COMMAND=transpose
COMMAND=resample     240
COMMAND=transpose
COMMAND=xpand    pjw  0  255  0  251
```

The COMMAND= lines were inserted by the various programs used to create the file. It was resampled to 320×240 resolution by the pipeline

```
resample   320  pjw  |  transpose   |  resample   240  |  transpose
```

and had its contrast adjusted by xpand. Resample, transpose, and xpand are discussed below.

The picinfo command allows you to examine a picfile header, but

```
sed '/^$/q' file
```

works just as well.

The hist and bbox commands extract other interesting information from picfiles. Hist prints a histogram of a picfile's pixel values, and bbox prints the bounding rectangle of the non-zero (or, when given an appropriate flag, any other value) pixels of a picfile.

All commands that read or write picture files understand the names IN and OUT to be synonyms for standard input and output.

3. Displays

Plan 9 terminals have a CRT that displays the contents of a *frame buffer*, a large memory containing a rectangular array of pixels. Most Plan 9 terminals have one, two or eight-bit pixels. In Plan 9, one-bit pixels are white when zero and black when one. Two-bit pixels may take on four shades of grey, with white again corresponding to zero. Eight-bit frame buffers generally have a color map.

The colors [-gfr] command creates a small window and fills it with a 16 by 16 array of squares, each of a different color in the color map. Pixel value zero is in the upper left-hand corner and colors increase by ones across each row.

The color map can be loaded from a file by the command getmap *file*. Getmap searches for files in the current directory and /lib/fb/cmap. A color map file contains 256 3-byte records. The command

```
getmap bw
```

loads a color map containing 256 shades of grey, effectively converting the terminal from color to monochrome, while

```
getmap 9
```

gets the Plan 9 default color map, loaded by the kernel at boot time and expected by many programs.

The 9v command creates a new window and displays a picture file in it. If the picture contains a color map, 9v loads it into the display. If the picture is full color (24 bits per pixel), 9v converts it to monochrome before displaying it.

Non-Plan 9 systems support innumerable other picture file formats. 9v can decipher pictures in many foreign formats it uses their file names as a clue to the format, as in this table:

Name	Format
*.gif	Compuserve GIF format
*.ega	IBM-PC EGA dump
*.face	USENIX facesaver format
*.jpeg	*jfif*-format *jpeg*
*.jpg	*jfif*-format *jpeg*
*.pcx	Paintbrush *pcx* format
*.rle	University of Utah Run-Length Encoded
*.sgi	Silicon Graphics image file
*.tga	Truevision TARGA format
*.tif	TIFF (Tagged Image File Format)
*.tiff	TIFF (Tagged Image File Format)
*.xbm	X Window System bitmap

Depending on the name of a file as a guide to its image format can be unreliable. The cvt2pic command recognizes foreign image files by their contents and converts them to picture files, so

```
cvt2pic file | 9v
```

may succeed where a naked 9v command would fail. There s much more to know about converting image file formats. Look at *cvt2pic* (9.1) for more detail and *pic2ps* (9.1) for a token nod at converting pic files into formats that may be useful elsewhere.

9v s conversion of full-color pictures to monochrome is often inappropriate. In that case, the 3to1 command is a useful intermediary. Its two arguments are the name of a colormap and a picfile (default standard input). It outputs an approximation of its input picture using the given colormap. So, a full-featured thunderclap to view a color picture in a foreign format is

```
cvt2pic file | 3to1 9 | 9v
```

Beware there s a limit to how well 3to1 can approximate its input. In particular, if the original file was a GIF or EGA-format picture, it already had 8-bit pixels and a color map, so running it through 3to1 wouldn t improve things and may very well make them much worse.

4. Picfile manipulation

Often you will need to convert a picture to have a particular size, or with a particular set of channels. The pcp (picfile copy) command will often do the job. For example,

```
pcp -w 0 70 320 125 pjw eyes
```

copies pjw to eyes. It copies only pixels in the window (0,70,320,125). This is the result:

Generally, pcp lets you select any window and any set of channels (in any order) from the input picture and rename the channels arbitrarily. It will use the NTSC luminance formula, $m=.299\times r+.587\times g+.114\times b$, to synthesize CHAN=m from CHAN=rgb and vice versa. Read *pcp* (9.1) for more details.

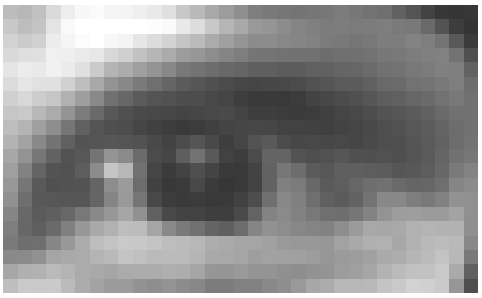
Several commands can alter a picfile s geometry in more complicated ways. The resample command will arbitrarily adjust the width of a picture by optimal anti-aliased resampling. It does a very good job (the best possible, in some precise sense), but it only works horizontally. To resample vertically, you can use it in conjunction with transpose, which, unsurprisingly, computes a picfile s transpose. This pipeline rescales a picfile in both directions:

```
resample xsize file | transpose | resample ysize | transpose
```

The transpose command has flags that will make it rotate multiples of 90° or mirror-reflect a picture vertically or horizontally the *transpose* (9.1) page in the Programmer s Manual gives more details, and also describes rotate, which rotates pictures by arbitrary angles. *Pdup* (9,1) describes a simpler and faster program that magnifies by duplicating pixels, giving a blocky appearance to its output. For example,

```
pcp -w 120 80 153 100 pjw | pdup 8 8
```

produces this output:



5. Creating new images

The commands described so far view pictures, convert between formats, and adjust their shapes and configurations in simple ways. The following commands create new pictures, either from whole cloth or by modifying and combining existing pictures.

The card command outputs a picture file all of whose pixels are the same color. The ramp command makes slightly more exciting pictures that blend between two colors from edge to edge. For example,

```
ramp -v -w 0 0 320 240
```

creates a 320×240 vertical ramp:

There are three categories of program that transform a single image: those that operate pointwise (that is, the output pixel values depend only on the corresponding input pixel), neighborhood operations, for which the output is some combination of several pixels surrounding the corresponding input pixel, and half-toning or color-quantization programs.

Pointwise operations tweak the intensity or color of the input picture. The xpand command can expand or compress the range of pixel values, adjusting the picture's contrast. By default, it expands the range of pixel values in its input, mapping the lowest input pixel value to zero and the highest to 255. Optionally, the range of output and/or input pixel values can be specified on the command line. Thus,

```
xpand pjw 255 0 0 255
```

interchanges black and white, producing a negative image:

He (an abbreviation of Histogram Equalization) evens out its input's distribution of pixel values, making it as even as possible. The cmap command maps its input's pixel values through a colormap. It works on an original that has CHAN=rgb, and uses the input pixel's red, green and blue channels to index the red, green and blue of the colormap. For example,

```
pcp -crgb pjw OUT | cmap 5.oclock.shado
```

produces this output:

Since pjw is a CHAN=m picture, we must use pcp to convert it to CHAN=rgb before passing it through cmap .

Remap tries to be the inverse of cmap . It may not succeed because the cmap mapping may not be invertible. Its output has rgb set to indices of the colormap entries that best approximate the rgb input.

The *filters* (9.1) manual page describes a group of programs that operate on neighborhoods. These include operations for blurring or sharpening images, adaptively adjusting their contrast, detecting or enhancing edges, and removing or exaggerating noise.

Adapt does adaptive contrast enhancement. It finds the minimum and maximum values in a 7×7 window around each pixel and remaps the center pixel using the linear function that sends the minimum to zero and the maximum to 255. The result of adapt pjw is

Ahe does adaptive histogram equalization. In 17×17 windows it counts the number of pixels whose value is less than the center pixel, counting ½ for each pixel equal to the center value. The output is just the count scaled to be between 0 and 255. For example, ahe pjw produces this result:

Crispen and laplace are high-pass filters that sharpen edges. Crispen is more extreme than laplace. Here is the output of crispen pjw:

Edge, edge2, and edge3 are all edge-detecting filters. Edge2 usually produces the best results. The output of edge2 pjw is

Combining the output of an edge detection operator with the original image enhances the edges.

```
edge2 pjw | lerp IN .3 pjw
```

produces this result:

In this example the lerp command interpolates pixel values linearly between the two images. The first image's pixels are multiplied by the coefficient .3 and added to .7 (that is 1-.3) times the second.

Median and nonoise are noise-reduction filters. They try to reduce the amplitude of random signals without affecting the underlying image. Smooth low-pass filters the image, blurring all the detail. For example,

```
smooth pjw | smooth | smooth
```

produces this result:

The limited color resolution of many displays, and particularly of hard-copy output devices, makes reducing the number of colors used in an image a popular and complicated topic. The *floyd* (9.1) page of the Programmer s Manual is devoted to programs that reduce grey-scale images to one bit per pixel. The best of these is probably floyd . The output of floyd pjw is

The *quantize* (9.1) page describes programs that try to reduce full-color (24 bit-per-pixel) pictures to 8 bits per pixel.

6. Composite images

The *lam* (9.1) page describes four commands that read multiple picture files and paste them together in different ways. The piccat and picjoin commands conjoin their inputs top-to-bottom and left-to-right. The lam command overlays its inputs so that their coordinate systems match; posit does likewise, but uses its inputs ± channels to let background images show through. For example

```
pcp -crgba pjw OUT | posit 9ball IN
```

produces this result

Pjw must first be passed through pcp because posit expects CHAN= attributes of all its inputs to match.