The 10th Edition Raster Graphics System

Tom Duff

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The current (late 1989) state of generating and displaying raster graphics in Research UNIX® is described.

1. Introduction

The Research UNIX system contains a number of commands to capture, manipulate, display and record monochrome and full-color raster images. Three groups of commands may be identified: interactive programs that operate on a frame buffer, commands that operate on images stored in picture files (see *picfile*(5)), and programs that interface to various graphical I/O devices: video cameras, scanners, paper plotters, film cameras and video tape recorders.

2. Video Facilities

No discussion of our raster graphics software can ignore the hardware on which it runs. The hardware available at different sites will, of course, vary. For definiteness, and to provide help for the local audience, this section will discuss the hardware available in Center 1127's graphics and image processing laboratory (MH 2C-524) and its neighborhood. Most other environments will have hardware that is similar in spirit if different in detail.

There are seven work stations in 2C-528. On the day this was written, four of them had TTY 5620 terminals, two had Gnot terminals and one had a SUN-3 workstation computer. Eventually, most of the 5620's will be replaced with Gnots. Each work station also has a Sony GDM-1901-12 video monitor that displays high-resolution video signals.

The room contains other video displays and recorders, including a Barco video projector in the ceiling, a 35-inch Mitsubishi monitor at the front of the room, a 19-inch Barco monitor at work station 6, two small Sony monitors in the video rack next to the audio console, two Panasonic Super-VHS recorders, a Sony 3/4-inch (U-MATIC) video player, a multi-standard (SECAM, NTSC, PAL) VHS player, a Sony BVH-2500 1-inch (SMPTE-C) video tape recorder and a Sony video camera.

The video equipment supports at least three incompatible video formats. High-resolution RGB video has 1024 scan lines, a 60 hz non-interlaced vertical scan rate, and transmits red, green and blue information on separate cables with synchronization pulses superimposed on the green channel. Low-resolution RGB video has between 480 and 488 scan lines, 30hz interlaced vertical scan, and separate RGB with sync on green. NTSC (National Television Standards Committee) video has the same timing characteristics as low-resolution RGB video, but encodes red, green, blue and sync into a single signal. NTSC is the encoding used by American, Canadian and Japanese television broadcasters, and by almost all video recording and playback equipment in those countries.

Various computer terminals generate video in other formats that our equipment handles with only limited success. Gnots, 630s, 5620s, Sun terminals, IBM-compatible PCs and Macintoshes all generate mutually incompatible video. Their vertical and horizontal scan-rates differ. The voltages and impedances of the signals they produce differ. Their color encodings differ. Monitors that can display video from all of these sources are rare, let alone hardware to convert from one format to another. For example, the only reliable way to record a signal from any of these sources is to place a camera in front of a monitor. The quality of the resulting recordings is often bad. It is a black art to adjust our Barco video projector to handle non-standard signals, but with a few days notice it can often be done. Again, the results are not often as good as one might like – the projector does not focus as tightly as a monitor and its brightness is limited. As

Equipment	High-res RGB	Low-res RGB	NTSC	Gnot	IBM	PAL/SECAM
workstation monitors	•					
Barco projector	•	•	•	•	maybe	maybe
35-inch Mitsubishi		•	•		•	
Barco at work station 6		•	•			
Sony rack monitors			•			
Super-VHS recorders			•			
U-MATIC player			•			
1-inch recorder			•			
multi-standard player			•			•
camera		•	•			
Metheus frame buffers	•					
ITI frame buffer		•				
Pixel Machine	•	•				

Table 1. Video devices

better video displays become available our situation will improve. Table 1 summarizes the equipment available and the video formats that each supports.

We have several Metheus 3610 frame buffers (seven on `pipe`, one on `arend`, one on `encke`) and an Imaging Technology, Inc. (ITI) RGB-512. All of our frame buffers store 32 bits at each pixel, one byte each for red, green blue and alpha. The 3610's generate high-resolution (1280×1024) video. The ITI generates low-resolution (512×480) video that may be recorded on video tape after conversion to NTSC. Connected to `pyxis`, a four CPU SGI 4D-240, is an AT&T Pixel Machine with 58 processors. It can generate either high- or low-resolution video under software control; the Pixel Machine documentation can tell you how.

Each piece of video equipment may be connected to any other via a video patch bay in 2C-538 (the `alice` room.) Alternating rows of the patch bay present video outputs and inputs. If an output and the input immediately below it are not plugged into anything, an internal connection routes one to the other. The patch bay has been layed out so that the most useful configurations require no patch cords. The patch bay is carefully labelled so that its proper use ought to be obvious.

The Sony BVH-2500 video recorder produces very high quality recordings on 1" video tape. Since it can overwrite arbitrary single frames of the tape, it is an ideal machine on which to record animation.

Before using a new tape, you must record (‘‘grind’’) time-code on it, numbering each frame of the tape. Time-code values are usually denoted by values of the form *hh.mm.ss.ff* (like `02.43.17.15`). The 2500's monitor output, available on the video patch bay, displays time-code superimposed on the 2500's output signal.

To grind time-code, use the patch bay to connect the color bar generator to the 2500's input, thread up a tape, and manually set the 2500 to record by pushing its `REC` and `PLAY` buttons simultaneously. Let it go until the tape runs out.

The *2500* command operates the recorder, reading instructions from its standard input. Its instruction set is moderately complicated; for most uses the following subset is adequate:

`cue` *hh.mm.ss.ff*
: Cue the tape to the given time code. The time-code displayed on the 2500's monitor output may be a few frames off, but the recorder will be cued to the correct point.

`still mode on`
: Put the recorder in single-frame record mode.

`still mode off`
: Put the recorder out of single-frame record mode.

`snap [`*n*`]`
: Record *n* frames (default 1) at the current cue point, and advance the cue point by *n* frames. The recorder must be in single frame mode.

`play`
: Start playing back from the current cue point.

`stop` Stop the recorder.

`!`*unix-command*
: Run the given *unix-command* using `/bin/sh`.

We currently have only two sources of digital video that may be recorded on video tape. These are the Pixel Machine and the ITI frame buffer attached to `kwee`. To use either one, you must patch its output to the NTSC color encoder, and patch the encoder's output to the video recorder. The ITI frame buffer is also useful as a frame-grabber, capturing its video input in its memory whence it may be saved in a picture file or otherwise manipulated.

The ITI is served by an ancient software regime whose commands all begin with the letters `iti`.

`itifbinit [-x]`
: Re-initialize the ITI to the state expected by the rest of the software. The ITI is often unused for days at at time, during which its health often decays. `itifbinit` is its restorative. The `-x` flag causes its output signal to be synchronized to the sync pulses of its input, instead of running from its internal clock. This is always a good idea.

`itigamma`
: Load the ITI's color map to correct intensities for display on CRT monitors.

`itigrab [-gs]`
: Run the frame-grabber. The `-g` flag starts the frame-grabber running. The displayed image will track the ITI's input video. `-s` stops the frame-grabber, freezing the image. Unadorned by flags, `grab` starts the frame-grabber and stops it one frame later.

`itigit` *picture-file*
: Copy the image stored in *picture-file* into the ITI.

`itisiv` *picture-file*
: Save the image in the ITI in *picture-file*.

3. Other output devices

Many modern laser printers and typesetters read data in the PostScript format.

`pic2ps [-h` *height*`] [`*picture*`]`
: converts a picture file into encapsulated PostScript, suitable for inclusion in any PostScript document. The `-h` option specifies the height, in inches, of the output image. It is not often required, as document processors usually insert PostScript illustrations in a scale-independent manner.

The `alice` room contains an Imagitex scanner that can be used to convert photographs to digital form. To use it, place the image to be scanned under the hold-down leaves, slide the leaves to make a window around the section you wish to scan, and use the *imscan* command.

`imscan [-s`*scale*`] [-l`*lens*`] file`
: The `-l` option causes the scanner to use a lens of focal length *lens* inches. The possibilities are 5 (754 dots per inch) and 8 (480 dots per inch); 8 is the default. The `-s` option sets the sub-sampling *scale*, which can vary from 1 to 9. One pixel in each *scale* by *scale* square will be stored. The default is 4. In conjunction with the default 8-inch lens, this causes scans to be stored at 120 dot-per-inch resolution.

There is a high-resolution one-bit-per-pixel Canon document scanner at the back of the graphics lab accessed through the *cscan* command.

`cscan [-f`*x*`,`*y*`] [-fL] [-s`*seconds*`] [-v] [`*file ...*`]`
: scans pages into the given files (default, one page onto standard output.) The `-f` option sets the size of the scan in pixels (400 to the inch); `-fL` sets double-letter size (11 by 17 inches, the largest possible.) The `-s` option sets the number of seconds to wait before scanning each page after the first.

In the Alice room is a Matrix Instruments QCR digital film recorder. It will record color or black-and-white images in a variety of photographic formats, include 8x10 Polaroid, 4x5 and 35mm. The *qsnap*(1) command will output an image to film.

4. Frame buffer commands

A frame buffer is a large memory organized as a two-dimensional array of pixels. Our Metheus 3610 frame buffers have 1024 scan lines of 1280 pixels each. The ITI frame buffer has 480 lines of 512 pixels. The coordinate system has (0,0) in the upper left-hand corner, with x increasing to the right, and y increasing down. This apparent weirdness is fairly standard, since it makes video output happen in row-major order.

Here we will mostly discuss commands for the Metheus displays. The corresponding ITI commands have the same names, but prefixed with the string `iti`.

There are seven Metheus frame buffers attached to pipe, named `/dev/om[0-6]`. All of the commands discussed below determine which one to use by examining the environment variable `FB`. It is often hard to tell what frame buffer is displayed on which monitor because of connections in the patch bay. The `fbi` (frame buffer identification) command displays each frame buffer's name in it.

Our frame buffers all have 32 bits per pixel, divided into four 8-bit channels. The channel values are normally thought of as fractions ranging from 0 to 1, although frame buffer commands perversely refer to them as integers between 0 and 255. Three of the channels specify the red, green and blue color components of the image. The fourth channel, called *alpha*, is used to indicate whether or not the image covers the pixel, and is not normally displayed. *Alpha* is used to control image compositing operations [2]. Fractional values of *alpha* describe pixels that the image partly or translucently covers, and facilitate anti-aliased compositing.

Each frame buffer contains three 256 entry look-up tables that specify mappings from the values stored in the red, green and blue channels to the voltages supplied at the frame buffers' video outputs. A couple of commands manipulate these mappings.

`gamma [`*power*`]`
: command loads these tables with a function that inverts the power-law relation between voltage and luminous flux normally encountered in CRT displays. Thus, pixel values normally correspond directly to displayed intensities. *Power* is the exponent of the power-law. The default of 2.3 is adequate for all our displays.

`getmap` *file* `[...]`
: command, whose arguments are a list of files containing color maps. On the ITI, the argument '`%`' refers to the current content of the frame buffer's color map. (The Metheuses' color maps are write-only.) The functional composition of the specified color maps is loaded into the frame buffer's color map. *Getmap* searches for files in `.`, then `/fb/cmap`, then `/usr/td/2d/cmap/lib`. A color map file contains 256 records of 3 bytes each, specifying the output values for the corresponding red, green and blue input values.

`ranmap`
: command loads random values into the color map.

The *zoom* and *movie* commands support magnification and animation of images.

`zoom [`*amount* `[`*x y*`]]`
: magnifies part of the image. With three arguments, *zoom* magnifies by *amount*, mapping the point (*x*,*y*) (default (0,0)) to the upper left-hand corner of the screen. With no arguments, *amount* defaults to 1. The Metheuses can magnify by any integral factor from 1 to 16. The ITI can magnify only by 1 or 2.

`movie` *xsize ysize nx ny* `[`*delay*`]`
: views an array of images in sequence by zooming and panning. The arguments are the size of the individual frames, the number of frames in the array in each direction, and optionally the number of 60ths of a second to delay between frames. The frames must be arranged boustrophedonically, with alternate rows proceeding from left to right and right to left. (This is because neither Metheus nor ITI frame buffers can pan in x and y simultaneously without glitching.)

There are a number of commands to load simple patterns into the frame buffer:

`clr [-w` *x0 y0 x1 y1*`] [`*r* `[`*g b* `[`*alpha*`]]]`
: sets all pixels to the given value. If only *r* is given, *g* and *b* are set to *r*. If *alpha* is not given, it is set to 255 (completely opaque.) The `-w` flag restricts attention to pixels inside the window whose upper-left corner is (*x0*,*y0*) and with (*x1*,*y1*) just diagonally outside the lower-right corner.

`cbars`
: displays a color-bars test pattern. The 8 bars at the top exercise all combinations of the 3 primary colors. The 9 patches at the bottom are a logarithmic (perceptually uniform) grey scale.

`ramp [-w` *x0 y0 x1 y1*`] [-v] [[`*c0*`]` *c1*`]`
: displays a horizontal ramp whose color is *c0* at the left and *c1* at the right. Colors are specified as for `clr` (green and blue

default equal to red, alpha defaults to 255). *C0* defaults to `0 0 0 255`. `-w` restricts `ramp` to the given window. `-v` gives a vertical ramp with *c0* at the top and *c1* at the bottom.

`colors [-gfr]`
: displays a 16 by 16 array of grey-colored (equal red, green and blue) squares in the middle of the screen with red, green and blue ramps at the top. This is mostly useful for examining color maps. The flags modify the display in small ways. `-r` suppresses the ramps. `-g` suppresses the gaps between the squares. `-f` expands the display to fill the full screen, making the patches non-square and suppressing the ramps.

The *xhair* command can be used to examine the contents of the frame buffer. It is named after the cross-hair that it draws on the screen. Single character commands manipulate the cross-hair, magnify the video and print pixel values. The commands are

`h`	print the help message
`lrud`	move left, right, up or down 1 pixel
`LRUD`	move left, right, up or down 16 pixels
`0`	move to center of screen (x=256, y=240)
`1-8`	magnify ×1–8
`9`	magnify ×16
`p`	print current coordinates and pixel value
`P`	print pixel after each command (toggle)
`m`	type coordinates to move to
`x`	type x coordinate to move to
`y`	type y coordinate to move to
`c`	change the crosshair display to a rectangle
`s`	manipulate other corner of rectangle
`^D,q`	exit xhair and run command
`Q`	exit xhair, don't demagnify or run command
`X`	exit and don't run command

If *xhair* is given arguments, they represent a command to be executed before exiting, after making substitutions for any argument whose first character is `%`. The substitutions made are:

`%r`	the current rectangle
`%w`	the current rectangle
`%p`	the upper-left corner of the rectangle
`%o`	the upper-left corner of the rectangle
`%c`	the lower-right corner of the rectangle
`%x`	the x coordinate of the upper-left corner
`%y`	the y coordinate of the upper-left corner
`%X`	the x coordinate of the lower-right corner
`%Y`	the y coordinate of the lower-right corner

The *mplot* command is a version of the standard UNIX *plot*(1) filter that produces output in a Metheus frame buffer.

5. Picture file commands

Most of our raster graphics commands require no special hardware. They synthesize images in picture files from textual or other descriptions, they modify images in picture files, producing results in picture files, or they combine the contents of several picture files to produce composite images, again storing the result in a picture file.

The *pcp* command takes two names of picture files or frame buffers and copies the first onto the second. As with all picture file commands, the special names `IN` and `OUT` refer to standard input and standard output. Frame buffers are designated by names that begin with `%`:

`%0`	Metheus frame buffer #0.
...	
`%9`	Metheus frame buffer #9.

Pcp has a number of options that alter the copied picture:

`-o` *x y*
: Add (*x*,*y*) to the picture's window coordinates.

`-w` *x0 y0 x1 y1*
: Clip the input picture's window to the given coordinates. If `-o` and `-w` are both given, the window is clipped before being offset.

`-t` *type*
: The output picture will have `TYPE=` *type*.

`-c` *channels*
: The output picture will be assembled from the given channels of the input picture. In many cases, a request for a channel not found in the input picture will be satisfied by standard conversions. For example, if *channels* includes `m`, but the input picture has only `rgb`, a monochrome channel is synthesized by computing NTSC luminance (`m=.299r+.587g+.114b`). Conversely, `rgb` will be synthesized from `m` by lookup in the input's color map, if it has one, or by `r=g=b` otherwise. If *channels* mentions `a` and the input has none, 255 is used. If *channels* mentions `z...` and the input has none, 1.0 (floating point) is used. Any other channel missing in the input is set to zero.

`-C` *channels*
: Put `CHAN=`*channels* in the output's header.

Without this option, the output's `CHAN` attribute is taken from the `-c` option, or failing that from the input's `CHAN` attribute. `-C` is useful, for example, to create a monochrome (`CHAN=m`) image from the red channel of a color image using `pcp -cr -Cm`.

The *lam* command combines any number of images, writing a picture file whose window is large enough to contain all the windows of its inputs. The input files are combined with pixels of later images overwriting earlier ones. This is only really useful if the windows of the input images differ. `-o` *file* specifies the output file name (standard output by default). All input images must have the same `NCHAN`.

The *posit* and *3matte* commands combine images using the two- and three-dimensional compositing operations described in [2] and [1]. Each takes a list of picture file names as arguments, producing a composite on standard output. The `-a` option will cause either program to output only the `rgb` channels, suppressing `a` (and `z...` in the case of *3matte*).

There is an army of commands to read an image and, under the control of a few parameters, write a modified image on standard output. Those that read a single picture file by default use standard input, so they are usable in a pipeline. They include:

`lum [`*picture*`]`
: File *picture* (default standard input) contains a color image or a monochrome image with a color map. A gray-level image is written on standard output, using the NTSC luminance formula.

`clip [-o` *x y*`]` *x0 y0 x1 y1* `[`*picture*`]`
: Clip an image to have `WINDOW=`*x0 y0 x1 y1*. A picture that does not fill out the window is filled with black pixels.

`xpand [-s] [`*picture*`] [`*lo hi* `[`*inlo inhi*`]]`
: The input picture has its dynamic range adjusted so that pixels in the range *inlo* to inhi are mapped to the range *lo* to *hi* (default 0 to 255). The default values for *inlo* and *inhi* are determined per channel by examining the input picture. The `-s` option causes all channels to be examined together. *Lo*, *hi*, *inlo* and *inhi* may have any values whatsoever. If *hi* is smaller than *lo*, pixel values will be inverted, producing a negative image. Any output pixel that would be mapped outside the range 0–255 is set to 0 or 255.

`dither [`*picture*`]`
: Convert a full-color (3 channel) picture to one channel with a color map by dithering.

`floyd [`*picture*`]`
: Convert an 8-bit gray-scale picture to one bit per pixel using a version of the Floyd-Steinberg error-diffusion method.

`halftone` *screen* `[`*picture*`]`
: Convert an 8-bit gray-scale picture to one bit using a given half-tone *screen*. A description of the screen is read from a file in `/usr/td/lib/screens`. The available screens include (among others)

`ALLEBACH`	Allebach's ordered-dither
`BAYER`	Standard ordered-dither
`BLUENOISE`	A pebble-screen pattern
`CLASSIC`	A 3-pixel-wide dot screen
`CLASSIC2`	Another 3-pixel-wide dot screen
`CLASSIC3`	A 4-pixel-wide dot screen
`CLASSIC4`	An 8-pixel-wide dot screen
`DIAMOND`	Rao and Arce's ordered-dither
`LINE`	Ulichney's line screen
`RING`	A concentric ring screen
`TILT18`	A tilted dot screen

`he [`*picture*`]`
: Histogram equalization: the intensity histogram of the input image is measured. The output image has its contrast altered for maximum use of the output range, equalizing the histogram as much as possible.

`hysteresis` *low high* `[`*picture*`]`
: Pixel values of *picture* below *low* are mapped to zero. Those above *high* are mapped to 255. If *low* and *high* are not equal, any region below *high* that has any 8-connected neighbors below *low* is mapped to zero.

`picaverage` *weight picture1 picture2*
: The output picture is a weighted average of *picture1* and *picture2*. *Weight* determines the fraction of the average contributed by *picture1*.

`piccat` *picture ...*
: The input *picture*s are concatenated one atop another. The output has the width of the widest input.

`picjoin` *picture ...*
: The input *picture*s are concatenated side by

side. The output has the height of the highest input.

`adapt` [*picture*]
: Adaptive contrast enhancement: a 7 by 7 neighborhood around each pixel is examined for its minimum and maximum values. The center pixel is remapped linearly in a way that would send the neighborhood's maximum to 255 and its minimum to 0. That is, `cen=255*(cen-min)/(max-min)`.

`ahe` [*picture*]
: Adaptive histogram equalization: each pixel of the output image is the histogram-equalized value of the center of a 17×17 pixel window surrounding it in the input image.

`clean` [*picture*]
: Bayer-Powell noise removal filter. If the center pixel of each 3×3 window in the input differs from the average of the other 8 pixels by more than 64, it is replaced by the periphery-average. This has the effect of flattening isolated noise pixels.

`crispen` [*picture*]
: 3×3 linear crispening filter. Convolves the input image with the kernel

```
-1 -1 -1
-1  9 -1
-1 -1 -1
```

This is a mild high-pass filter.

`edge` [*picture*]
: 3×3 linear edge-detection filter. Convolves the input image with the kernel

```
-1 -1 -1
-1  8 -1
-1 -1 -1
```

This is just the difference between the original image and the output of `crispen`.

`edge2` [*picture*]
: 3×3 non-linear edge-detection (Sobel operator) filter.

`extremum` [*picture*]
: 3×3 extremum filter. Replaces the center pixel of each by the value in the 3×3 window surrounding it that most differs from it.

`laplace` [*picture*]
: 3×3 Laplacian filter. Convolves the input image with the kernel

```
 0 -1  0
-1  5 -1
 0 -1  0
```

This is a fairly extreme high-pass filter.

`median` [*picture*]
: 3×3 median filter. Each pixel is replaced by the median of the 3×3 window surrounding it.

`smooth` [*picture*]
: 3×3 Bartlett filter. Convolves the input image with the kernel

```
1/16 2/16 1/16
2/16 4/16 2/16
1/16 2/16 1/16
```

This is a moderately strong low-pass filter.

`3to1` [`-e`] *colormap* [*picture*]
: Converts the input picture from full-color (`rgb`) to a single channel mapping each pixel to the closest entry of *colormap*.

`mcut` [*picture*]
: Reads a picture, and writes a color map on standard output suitable for use by *3to1*. *Mcut* uses Heckbert's median-cut algorithm to pick a color map that matches *picture*'s colors pretty well.

`improve` *colormap* [*picture*]
: Given a color map and a picture file, this outputs a new color map that better represents the colors of the picture. The algorithm is to output the centroid of those pixel values that are closest to each input color map entry. Running *improve* several times may produce better and better color maps.

`quantize` [*picture*]
: Convert a full-color picture to an 8-bit picture with color map. This is just a command file that calls *mcut*, *improve* and *3to1*. It does a much better job than *dither*.

`remap` *colormap* [*picture*]
: The input picture should be full color (`CHAN=rgb`). The output will have its pixel values will be altered so that when mapped through the given *colormap* they will be as close as possible to the input's pixel values.

`resample` *width* [*picture*] [*B C*]
: Resample the input image to be *width* pixels wide. The default filter used in resampling minimizes both pre- and post-

aliasing. Numeric parameters *B* and *C* (both default to 1/3) pick the resampling kernel from a Mitchell and Netravali's two-parameter family of piecewise cubic kernels.

`transpose [-vhadrlui] [-o` *x y*`] [`*picture*`]`
Transpose the input picture. This is useful in conjunction with commands that operate on scan-lines, like *resample*, to perform operations on columns instead of rows. Under control of its options, *transpose* can perform any symmetry operation of the integer lattice. The `-v` option reflects through a vertical line. The `-h` option reflects through a horizontal line. The `-a` option reflects through an ascending diagonal line. The `-d` option reflects through a descending diagonal line (the default). The `-r` option rotates right (clockwise 90 degrees). The `-l` option rotates left (counterclockwise 90 degrees). The `-u` option flips the image upside down (180 degree rotation.) For completeness, the `-i` option does the identity transformation. The `-o` option translates the picture, adding (*x*,*y*) to all coordinates. Without this option, the upper-left corner of the image's window does not change.

`shear` *angle* `[`*picture*`]`
Rotate the input image by the given *angle* (in degrees). It's called *shear* because it operates by shearing the image 3 times (horizontally, then vertically, then horizontally).

`lx [-o`*file*`] [-A`*aspect*`] [-a] [-s`*scale*`] [-r`*rot*`] [-x`*xscale*`] [-y`*yscale*`] [`*picture*`]` Perform a linear transformation on the input image. The `-o` option specifies the output file name. The default is standard output. The `-A` option specifies the aspect ratio of the pixels. The default is 1. The ITI frame-grabber produces images whose pixel aspect-ratio is 1.25. The `-a` option suppresses the writing of an alpha channel. Normally an alpha channel is computed even for input images that don't have one, since the output picture is often rotated and thus doesn't completely cover its window.

The transformation is specified by a sequence of options. The specified transformations are combined in the order given to yield a composite transformation. The relevant options are:

`-s`*scale* scale by *scale*.
`-r`*rot* rotate by *rot* degrees clockwise.
`-x`*xscale* scale in x by *xscale*.
`-y`*yscale* scale in y by *yscale*.

There are several commands to generate images from three-dimensional geometric descriptions of various sorts. Most of these produce `CHAN=rgbaz...` images that may be combined using *3matte*. In their output files, points at the near clipping plane will be mapped to points having *z=0*, and points at the far clipping plane will have *z=1*.

`ncpr [-a` *aspect*`] [-w` *x0 y0 x1 y1*`] [-c` *rgbaz*`]` *input* `[`*output*`]`
New Cheezy Polygon Renderer. *Output* (default standard output) is the name of the picture file that will contain the rendered version of the scene described in *input*, a text file specifying a polygonal scene. The `-a` option sets the pixel aspect-ratio (default 1.) The `-w` option sets the window of the output picture. The `-c` option specifies which channels should be written to the output picture.

The input file contains a sequence of single-letter commands, each with several numeric parameters. The commands are:

`v` *fov near far ex ey ez lx ly lz ux uy uz*
Set viewing parameters. *Fov* is the angle subtended vertically by the screen at the eye point. Points whose distance from the eye is not between *near* and *far* will be clipped away before drawing. However tempted, do not set *near* to zero, lest underflow or divide-check occur. (*ex*,*ey*,*ez*) is the coordinate of the eye, the point from which the scene is viewed and the center of perspective. (*lx*,*ly*,*lz*) is a vector pointing from the eye toward the center of the scene. The point (*lx+ex*,*ly+ey*,*lz+ez*) is mapped into the center of the screen. (*ux*,*uy*,*uz*) is the up vector, the direction of the zenith. The point (*lx+ux*,*ly+uy*,*lz+uz*) is mapped into a point somewhere above the center of the screen.

`l` *x y z*
Set the direction of the light source to (*x*,*y*,*z*). The light source is ''at infinity'' in the given direction.

`b` *red green blue alpha* Clear the screen to the given color. *Red*, *green*, *blue* and *alpha* should all be between 0 and 255.

`c` *index red green blue alpha* Set a color table entry. Indices into the color table are used to specify the colors of polygons (see below.) The table has 500 entries. Unless reloaded by the `c` command, the first 256 entries contain the 256 shades of gray, the following 12 entries (256-267) are set to 12 logarithmically spaced (perceptually equal) gray shades, and the next 20 entries (268-287) to 20 logarithmically spaced gray shades.

`t` *x0 y0 z0 x1 y1 z1 x2 y2 z2 c0 c1*
Render a triangle with vertices (*x0,y0,z0*), (*x1,y1,z1*) and (*x2,y2,z2*). The side the normal (calculated using the right hand rule) out of has color *c0*, on the other it is *c1*. If *c0* or *c1* is positive, the polygon's color is found in the corresponding color table entry. If negative, the color is found by modifying the color table entry as though the surface were illuminated by a light source whose direction was specified by the `l` command.

`p` *c0 c1 x0 y0 z0 x1 y1 z1 ... xn yn zn* `;`
Render a polygon whose color is *c0* on one side and *c1* on the other. The polygon's vertices are (*x0,y0,z0*), (*x1,y1,z1*), ..., (*xn,yn,zn*).

`quad [-a] [-z] [-w` *x0 y0 x1 y1*`]` *in out*
Compute an image of a quadric surface. The `-a` option suppresses writing out the alpha channel. The `-z` option suppresses writing out the z channel. The `-w` option specifies the output window. The input file should contain 34 floating point numbers. The first ten numbers are the upper triangle of the symmetric matrix describing the quadratic form (in screen coordinates.) The next 16 numbers are a matrix that converts screen-space coordinates into world-space normals for illumination computations. The next three numbers are the direction of the light source. The next four numbers are the red, green, blue and alpha of the surface's color. The last number is the amount of ambient light in the environment.

`terrain` *in out ex ey ez lx ly fov near far*
Render a terrain image. The input file is a 2-channel picture file containing 16-bit elevation data on a regular grid. (*ex,ey,ez*) is the eye position. (*lx,ly,0*) is a vector pointing from the eye to the center of the scene. The up direction is (*0,0,1*). *Fov* is the vertical field-of-view angle. *Near* and *far* are the distances from the eye to the near and far clipping planes.

`bg` *r0 g0 b0 r1 g1 b1 out*
Generate a background card whose color varies smoothly from (*r0,g0,b0*) at the top to (*r1,g1,b1*) at the top. Its z coordinate is set to 2, which is beyond the far clipping plane.

`aplot [-t` *type*`] [-r` *range*`] [-w` *x0 y0 x1 y1*`]` *input* Produces an anti-aliased isometric plot of a square array of binary data, read from its input file. The `-r` option specifies the maximum absolute value of the data. This may be adjusted to affect the height of the highest peaks in the plot. By default, the input is examined to find its range. The `-w` option specifies the window in which the plot will be drawn. The data file is just a binary dump of a square array. It has no header, and in particular is not a picture file. The `-t` option (default `-tf)` specifies the type of data in the array.

option	type
`-tf`	float
`-ts`	short int
`-ti`	int
`-tl`	long int
`-td`	double
`-tc`	char
`-tu`	unsigned char

6. Animation

To use a command-based raster graphics system as described here to for animation requires writing command files to create and record long sequences of images. Typical command files contain long sequences of repeated commands with slowly changing numeric parameters. Several sequences starting and ending at different times may be interleaved to describe overlapping motion. They are at best tedious and at worst tricky to generate by hand or using the usual tools.

Moto is a command generator tailored for an animator's needs. Its input is a concise description of the animation to be performed; its output is a command file suitable for input to *sh*, *rc* or some other command interpreter. Its arguments are an optional file name containing a *moto* program (default standard input) and list of numeric parameters that are made available to the program.

A *moto* program consists of a list of groups of commands. Each block is guarded by a range of frames. Here is an example:

```
1,5:    pcp this %0
        pcp %0 that
```

This generates

```
pcp this %0
pcp %0 that
pcp this %0
pcp %0 that
pcp this %0
pcp %0 that
pcp this %0
pcp %0 that
pcp this %0
pcp %0 that
```

The command group is repeated for each of frames 1 to 5.

Groups may contain parameter ranges enclosed in brackets `[]`:

```
1,5:    pcp frame.[1,5] %0
        echo snap|2500
```

This generates:

```
pcp frame.1 %0
echo snap|2500
pcp frame.2 %0
echo snap|2500
pcp frame.3 %0
echo snap|2500
pcp frame.4 %0
echo snap|2500
pcp frame.5 %0
echo snap|2500
```

Programs may have multiple groups, each guarded by a separate range of frames. For each frame, *moto* checks each group and processes those whose guards include the current frame number.

Two special guards, `BEGIN` and `END`, specify actions to be taken before an after processing frames:

```
BEGIN:  clr
1,5:    pcp section[1,5] %0
END:    pcp %0 composite
```

This generates

```
clr
pcp section1 %0
pcp section2 %0
pcp section3 %0
pcp section4 %0
pcp section5 %0
pcp %0 composite
```

Moto allows complex computations inside parameter brackets:

```
1,10:   clr [127.5*(1-cos([0,360]))]
```

This generates

```
clr 0
clr 29.82933350233
clr 105.35985734747
clr 191.25
clr 247.3108091502
clr 247.3108091502
clr 191.25
clr 105.35985734747
clr 29.82933350233
clr 0
```

Expressions may include constants and variables. All values are double-precision floating point numbers. The operators =, `/`, `+`, `-` (both unary and binary), <, >, <=, >=, ==, `!=`, `? :` and `!`, all with their meanings as in C, except that all results are coerced to `double`. The result of `a%b` is `a-b*(int)(a/b)`. The result of `a && b` is `a?b:a` . The result of `a || b` is `a?a:b` . The exponentiation operator is `^`, also written `**`. The expression `[a,b]` varies from `a` to `b`, linearly as the frame number varies between the guards of the group containing the expression. The expression `a[b,c]` has the value `a*b+(1-a)*c`. Its value varies from `b` to `c` as `a` varies from 0 to 1. The expression `$i` has the value of the `i`'th parameter following the file name on *moto*'s command line.

The precedence of operators is, from lowest to highest:

```
=
? :
||
&&
< <= == != > >=
+ -
* / %
[ ]
^ **
- (unary) ! $
```

Expressions may be parenthesized to alter precedence.

The following math functions are available:

```
acos    besy0   exp     log10
asin    besy1   fabs    sin
atan    besyn   floor   sinh
besj0   ceil    gamma   sqrt
besj1   cos     hypot   tan
besjn   cosh    log     tanh
```

All math functions are as described in the C library, except that angles are measured in degrees rather than radians for the trig and inverse trig functions. In addition *hypot* may have two or three arguments, *atan* may take two arguments instead of one, and may also be spelled *atan2*.

For parameterization, and to allow even more complex computations, *moto* has variables, assignment and computation groups. A computation group is distinguished from a command group by having a double colon separating its guard from the expressions to be computed:

```
BEGIN:: n=5
1,n::   x=512*sin([0,90])
1,n:    pcp -w 0 0 [x] 488 pic.[1,n] %0
```

This generates

```
pcp -w 0 0 0 488 pic.1 %0
pcp -w 0 0 195.93391737093 488 pic.2 %0
pcp -w 0 0 362.03867196751 488 pic.3 %0
pcp -w 0 0 473.02632064578 488 pic.4 %0
pcp -w 0 0 512 488 pic.5 %0
```

Upon occasion it is useful to split *moto*'s output into several files, under program control. A group that is separated from its guards by an at-sign `@` instead of a colon names a file into which subsequent output is to be written. For example,

```
1,5@    file.[1,5]
1,5:    This is file.[1,5].
```

creates 5 files, with names `file.1`,...,`file.5`. Each file's contents will announce its name.

As is true for all sufficiently large programs, *moto* has a shell escape. A group separated from its guards by an exclamation point `!` instead of a colon has its result text interpreted by a subshell.

Finally, Figure 1 shows an example taken from a real application. This *moto* program composites the frames of a short movie showing two flying saucers, flying in formation, chased by a third, racing over New Jersey. The flying saucer images (files `run.*` and `chase.*`) and the background (file `bg`) have been computed in advance. In the composite, the `run.*` images are re-used, staggered in time, to do the first two saucers.

7. References

1. Duff, T. Compositing 3D rendered images. *Computer Graphics 19*, 3 (1985), 41-44. (1985 Siggraph Proceedings).
2. Porter, T. and Duff, T. Compositing digital images. *Computer Graphics 18*, 3 (1984), 253-258. (1984 Siggraph Proceedings).


```
BEGIN::  nchase=108
         nrun=195
         d1=12
         d2=32
         end=nrun+d2
         chase=end-nchase+1
1,end:  inputs=                        # empty the input list
1,nrun: inputs="$inputs run.[1,nrun]"   # add the first saucer to the input list
1+d1,nrun+d1:
        inp="$inputs run.[1,nrun]"      # add the second saucer
chase,end:
        inp="$inputs chase.[1,nchase]"  # add the chasing saucer
1,end:
        3matte -a $inp bg frame.[1,end] # create the composite
```

Figure 1. Flying saucer script

photo page

divider with title

Implementation and Maintenance