NAME

aliastorle − Convert Alias™ raster files to RLE format.

SYNOPSIS

aliastorle [**−v**] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program converts an image in Alias™ "pix" format to *RLE*(5) format. Since "pix" and *RLE* differ on the origin location, the program flips the image top to bottom.

OPTIONS

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream. **−v** Verbose output.

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

rletoalias(1), *urt*(1), *RLE*(5).

AUTHOR

Raul Rivero, Mathematics Department, University of Oviedo.

NAME

applymap − Apply the color map in an RLE file to the pixel data

SYNOPSIS

applymap [**−l**] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program takes the color map in an *RLE*(5) file and modifies the pixel values by applying the color map to them. If there is more than one color channel in the input file, the color map in the input file should have the same number of channels. If the input file has a single color channel, the output file will have the same number of color channels as the color map.

Each pixel in the input file is mapped as follows: For a multi-channel input file, a pixel in channel *i* is mapped as *map[i][pixel] >> 8*, producing a pixel in output channel *i*. The right shift takes the 16 bit color map value to an 8 bit pixel value. For a single channel input file, to produce a pixel in output channel *i* is produced from the corresponding input pixel value as *map[i][pixel] >> 8*.

OPTIONS

−l This option will cause a linear (identity) color map to be loaded into the output file. Otherwise, the output file will have no color map.

infile The input will be read from this file, otherwise, input will be taken from stdin.

−o *outfile*
If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

rleldmap(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

BUGS

If the image data and color map channels in the input file do not conform to the restriction stated above (N→N or 1→N) the program will most likely core dump.

NAME

avg4 − Downfilter an image by simple averaging.

SYNOPSIS

avg4 [**−o** *outfile*] [*infile*]

DESCRIPTION

Avg4 downfilters an RLE image into a resulting image of 1/4th the size, by simply averaging four pixel values in the input image to produce a single pixel in the output. If the original image does not contain an alpha channel, avg4 creates one by counting the number of non-zero pixels in each group of four input pixels and using the count to produce a coverage value. While the alpha channel produced this way is crude (only four levels of coverage) it is enough to make a noticeable improvement in the edges of composited images.

OPTIONS

infile The input will be read from this file, otherwise, input will be taken from stdin.

−o *outfile*

If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

fant(1), *rlecomp*(1), *smush*(1), *urt*(1), *RLE*(5).

AUTHOR

Rod Bogart, John W. Peterson

BUGS

Very simple minded − more elaborate filters could be implemented.

NAME

crop − Change the size of an RLE image

SYNOPSIS

crop [**−b**] [*xmin ymin xmax ymax*] [**−o** *outfile*] [*infile*]

DESCRIPTION

Crop changes the size of an RLE image. The command line numbers *xmin ymin xmax ymax* specify the bounds of the resulting image. If the resulting image is larger than the original, *crop* supplies blank pixels, otherwise pixels are thrown away.

OPTIONS

−b The input image is cropped to the enclosing box. Extra rows and columns of black pixels are removed. The *infile* must be a file; no piped input is allowed for this option.

−o *outfile*
If specified, output will be written to this file, otherwise it will go to stdout.

infile The input will be read from this file, otherwise, input will be taken from stdin.

SEE ALSO

repos(1), *urt*(1), *RLE*(5).

AUTHOR

Rod Bogart

BUGS

Could be combined with *repos*. Does not check to see if the input and output regions are disjoint.

NAME

cubitorle − Convert cubicomp image to an RLE format file.

SYNOPSIS

cubitorle [**−o** *outfile*] *inprefix*

DESCRIPTION

Cubitorle converts a set of files in the Cubicomp image format to a raster file in the Utah Raster Toolkit RLE format. *Cubitorle* expects as input a set of files of the form "*inprefix*.r8", "*inprefix*.g8", and "*inprefix*.b8". These files are combined to form a single *RLE*(5) file. The output is written to *stdout* unless an output file name is given using the **−o** option.

OPTIONS

−o Allows specification of an output file name.

SEE ALSO

rleflip(1), *urt*(1), *RLE*(5).

AUTHOR

Rod Bogart

NAME

dvirle − convert dvi version 2 files, produced by TeX82, to RLE images

SYNOPSIS

dvirle [**−m** *number*] [**−h**] [**−s**] [**−d** *number*] [**−x** *xfilter*] [**−y** *yfilter*] *infile.dvi*

DESCRIPTION

Dvirle converts .dvi files produced by *TeX*(1) to *RLE*(5) format. The basic process involves two passes. In the first pass, the .dvi file is converted into a list of characters. The second pass takes this list and converts it to *RLE*. The image is filtered to produce gray-scale letters. 300dpi fonts are used, producing an unfiltered page size of approximately 2500×3500 pixels. The default is to average this by 5 pixels in the X direction and 5 in the Y, producing a 510×708 image. The filtering parameters can be altered with the **−x** and **−y** flags.

The **−m** *number* option is used to change the device magnification (which is in addition to any magnification defined in the TeX source file). *Number* should be replaced by an integer which is 1000 times the magnification you want. for example, **−m 1315** would produce output magnified to 131.5% of true size. The default is no magnification (1000). Note, however, that a site will only support particular magnifications. If you get error messages indicating that fonts are missing when using this option, you probably have picked an unsupported magnification.

The **−h** flag, when supplied, causes the image to be converted "on its side" (rotated by 90 degrees).

Normally the first pass prints the page numbers from the .dvi file. The **−s** flag suppresses these.

The default *maxdrift* parameter is 2 pixels (1/100th of an inch); the **−d** option may be used to alter this. The *maxdrift* parameter determines just how much font spacing is allowed to influence character positioning. The default value 2 allows a small amount of variation within words without allowing any letters to become too far out of position.

The output file contains a number of separate *RLE* images concatenated, one for each page in the input. These can be separated with *rlesplit*(1). The output images have a single image channel and an identical "alpha" channel. For compositing with a colored background, it will be necessary to use *rleswap*(1) to expand it to 3 color channels.

The shell script *topcrop* will crop off the top 384 lines of the output image (assuming the default *LaTeX* page size and *dvirle* filtering parameters), making it suitable for viewing on a (384×512) frame buffer.

```
topcrop <file.rle >cropfile.rle
```

A better solution is to use something like the following *LaTeX* macros to set the page size so that, with the default filter parameters, the output images will be 510×384.

```
\newcommand{\maxpage}{%% Make page as large as possible
                \setlength{\topmargin}{0in}
                \setlength{\oddsidemargin}{0pt}
                \setlength{\evensidemargin}{0pt}
                \setlength{\marginparwidth}{0pt}
                \setlength{\marginparsep}{0pt}
                \setlength{\headheight}{0pt}
                \setlength{\headsep}{0pt}
                \setlength{\textwidth}{6.5in}}
\newcommand{\plainpage}{%% Page with space for headers
                \pagestyle{plain}
                \setlength{\textheight}{4.0667in}
                \setlength{\footheight}{12pt}
                \setlength{\footskip}{24pt}
                \maxpage}

\newcommand{\headingspage}{%% Page with headers
```

```
                \pagestyle{headings}
                \setlength{\textheight}{4.0667in}
                \setlength{\footheight}{12pt}
                \setlength{\footskip}{24pt}
                \maxpage}
\newcommand{\emptypage}{%% Page with no headers
                \pagestyle{empty}
                \setlength{\textheight}{4.4in}
                \setlength{\footheight}{0pt}
                \setlength{\footskip}{0pt}
                \maxpage}
```

FILES

dvirle1 first pass
dvirle2 second pass

SEE ALSO

rleflip(1), *rlesplit*(1), *rleswap*(1), *urt*(1), *RLE*(5).

AUTHOR

The original (Versatec) version was written by Janet Incerpi of Brown University. Richard Furuta and Carl Binding of the University of Washington modified the programs for DVI version 2 files. Chris Torek of the University of Maryland rewrote both passes in order to make them run at reasonable speeds. Spencer W. Thomas of the University of Utah converted it to produce RLE images as output.

BUGS

The **−h** option doesn't work properly. Use *rleflip*(1) instead.

Truncates pages wider than 2550 pixels (8.5 inches).

Doesn't handle missing fonts gracefully.

Should be a single program, instead of a shell script and two programs. Doesn't use the usual RLE argument and file name conventions. Should output the TeX page numbers as picture comments.

NAME

fant − perform simple spatial transforms on an image

SYNOPSIS

fant [**−a** *angle*] [**−b blurfactor**] [**−o** *outfile*] [**−p** *xoff yoff*] [**−s** *xscale yscale*] [**−S** *xsize ysize*] [**−v**] [*infile*]

DESCRIPTION

Fant rotates or scales an image by an arbitrary amount. It does this by using pixel integration (if the image size is reduced) or pixel interpolation if the image size is increased. Because it works with sub-pixel precision, aliasing artifacts are not introduced (hah! see BUGS). *Fant* uses a two-pass sampling technique to perform the transformation. If *infile* is "−" or absent, input is read from the standard input.

OPTIONS

−a *angle*

Amount to rotate image by, a real number from 0 to 45 degrees (positive numbers rotate clockwise). Use rleflip(1) first to rotate an image by larger amounts.

−b *blur_factor*

Control the amount of blurring in the output image. If the blur factor is greater than one, image blurring will increase. If the blur factor is smaller than one, image blurring will decrease but aliasing artifacts may be visible.

−o *outfile*

Specifies where to place the resulting image. The default is to write to stdout. If *outfile* is "−", the output will be written to the standard output stream.

−p *xoff yoff*

Specifies where the origin of the image is − the image is rotated or scaled about this point. If no origin is specified, the center of the image is used.

−s *xscale yscale*

The amount (in real numbers) to scale an image by. This is often useful for correcting the aspect of an image for display on a frame buffer with non square pixels. For this use, the origin should be specified as 0, 0 (see **−p** above). If an image is only scaled in Y and no rotation is performed, *fant* only uses one sampling pass over the image, cutting the computation time in half.

−S *xsize ysize*

An alternate method of specifying the scale factors. *xsize* and *ysize* give the desired output image size.

The **−S** option can not be used in combination with **−a**, **−p**, or **−s**.

−v Verbose output. Primarily for debugging.

SEE ALSO

avg4(1), *rleflip*(1), *rlezoom*(1), *urt*(1), *RLE*(5),
Fant, Karl M. "A Nonaliasing, Real-Time, Spatial Transform Technique", *IEEE CG&A*, January, 1986, p. 71.

AUTHORS

John W. Peterson, James S. Painter

BUGS

Fant uses a rather poor anti-aliasing filter (a triangle filter). This is usually good enough but will exhibit noticeable aliasing artifacts on nasty input images.

NAME

giftorle − Convert GIF images to RLE format

SYNOPSIS

giftorle [**−c**] [**−o** *outfile.rle*] [*infile.gif ...*]

DESCRIPTION

Giftorle converts a file from Graphics Interchange Format (GIF) format into RLE format. Multiple input images may be converted, these will be written sequentially to the output RLE file. The origin of a GIF image is at the upper left, while the origin of an RLE image is at the lower left. This program automatically flips the image to preserve its orientation.

OPTIONS

−c Preserve the colormap that the GIF image contains, otherwise the colormap is applied to input image.

−o *outfile.rle*
If specified, the output will be written to this file. If *outfile.rle* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.gif ...
The input will be read from these files. If *infile.gif* is "−" or is not specified, the input will be read from the standard input stream.

MISC

GIF and Graphics Interchange Format are both trademarks of CompuServe Incorporated.

SEE ALSO

rletogif (1), *urt*(1), *RLE*(5).

AUTHOR

David Koblas (koblas@mips.com or koblas@cs.uoregon.edu)

NAME

graytorle − Merges gray scale images into an RLE format file.

SYNOPSIS

graytorle [**−a**] [**−h** *hdrsize*] [**−o** *outfile*] *xsize ysize infiles*

DESCRIPTION

Graytorle reads a list of 8-bit gray scale images in unencoded binary format and converts them to an *RLE*(5) image with the number of channels corresponding to the number of input files. A command line option allows specifying one of the files as an alpha channel.

OPTIONS

−a Designates the first file in the input list as being information for the alpha channel of the image.

−h *hdrsize*
Often gray scale image files include some sort of header information. This option allows specification of a count of bytes to discard at the beginning of each input file.

xsize Specifies the horizontal resolution of the input files.

ysize Specifies the vertical resolution of the input files.

−o *outfile*
This option is used to name the output file. Otherwise, output is written to *stdout.*

infiles List of input files.

SEE ALSO

rletogray(1), *urt*(1), *RLE*(5).

AUTHOR

Michael J. Banks, University of Utah.

NAME

into − copy into a file without destroying it

SYNOPSIS

into [**−f**] *outfile*

DESCRIPTION

Into copies its standard input into the specified *outfile*, but doesn't actually modify the file until it gets EOF. This is useful in a pipeline for putting stuff back in the "same place." The *outfile* is not overwritten if that would make it zero length, unless the **−f** option is given. That option also forces overwriting of the *outfile* even if it is not directly writable (as long as the directory is writable).

SEE ALSO

pipe(2)

BUGS

For efficiency reasons, the directory containing the *outfile* must be writable by the invoker.
The original *outfile's* owner is not preserved.

NAME

mcut − Quantize colors in an image using the median cut algorithm

SYNOPSIS

mcut [**−n** *colors*] [**−d**] [**−o** *outfile*] *infile*

DESCRIPTION

Mcut reads an RLE file and tries to choose the "best" subset of colors to represent the colors present in the original image. A common use for this is to display a 24 bit image on a frame buffer with only eight bits per pixel using a 24 bit color map. *Mcut* first quantizes intensity values from eight bits to five bits, and then chooses the colors from this space.

Mcut runs in two passes; the first pass scans the image to find the color distributions, and the second pass maps the original colors into color map indices. The output file has a color map containing the colors *mcut* has chosen. *Mcut* also sets the picture comment "color_map_length" equal to the number of colors it has chosen. The *getx11* program (among others) will use this color map instead of dithering.

OPTIONS

−n *ncolors*

Limit the number of colors chosen to ncolors. The default is 200.

−d Uses Floyd/Steinberg dither to hide contouring. Greatly improves images that have a small number of colors.

infile The input will be read from this file. If it is a multi-image file, each image will be quantized to its own colormap. Piped input is not allowed.

−o *outfile*

If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

getx11(1), *rlequant*(1), *urt*(1), *RLE*(5),
"Color Image Quantization for Frame Buffer Display", by Paul Heckbert, Procedings of SIGGRAPH ’82, July 1982, p. 297.

AUTHOR

Robert Mecklenburg, John W. Peterson, University of Utah.

BUGS

The initial quantization is hardwired to five bits. This should be an option.

NAME

mergechan − merge channels from several RLE files into a single output stream

SYNOPSIS

mergechan [**−a**] [**−o** *outfile*] *infiles* ...

DESCRIPTION

Mergechan takes input from several RLE files and combines them into a single output stream. Each channel in the output stream comes from the respective filename specified on the input (i.e., channel zero is taken from the first file, channel one from the next, etc). The same file can be specified more than once. If the **−a** flag is given, the channels are numbered from -1 (the alpha channel) instead of zero. All of the input channels must have exactly the same dimensions (use *crop*(1) to adjust files to fit each other).

Mergechan is typically used to introduce an alpha mask from another source into an image, or combine color channels digitized independently.

If **−o** is specified, the output will be written to *outfile*.

SEE ALSO

crop(1), *rleswap*(1), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson, University of Utah.

BUGS

Mergechan is totally ignorant of the color maps of the input files.

The restriction that all input files must be the same size could probably be removed.

NAME

painttorle − Convert MacPaint images to RLE format.

SYNOPSIS

painttorle [**−c** [*red*] [*green*] [*blue*] [*alpha*]] [**−r**] [**−o** *outfile.rle*] [*infile.paint*]

DESCRIPTION

Painttorle converts a file from MacPaint format into RLE format. Because MacPaint and RLE disagree on which end is up, the output should be sent through *rleflip* to preserve orientation.

OPTIONS

−c*[red] [green] [blue] [alpha]*
Allows the color values to be specified (the default is 255).

−r Invert the color of the MacPaint pixels (reverse video).

infile.paint
The input paint data will be read from this file, otherwise, input will be taken from stdin.

−o *outfile.rle*
If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

rletopaint(1), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson

NAME

pgmtorle − convert a pbmplus/pgm image file into an RLE image file.

SYNOPSIS

pgmtorle [**−h**] [**−v**] [**−a**] [**−o** *outfile*] [*filename*]

DESCRIPTION

This program converts PBMPLUS grayscale (pgm) image files into Utah *RLE*(5) image files. PBMPLUS/pgm image files contain the image dimensions and 8-bit pixels with no matte or alpha data. When converting to an RLE file, the alpha channel may optionally be computed. The RLE file will contain a "grayscale" image (8 bits) with no colormap. The origins of PBMPLUS and Utah RLE files are in the upper left and lower left corners respectively, so this program automatically "flips" the image. These RLE files may then be viewed using any RLE image viewer.

OPTIONS

−v This option will cause pgmtorle to operate in verbose mode. The header information is written to "stderr". Actually, there is not much header information stored in a PBMPLUS file so this information is minimal.

−h This option allows the header of the PBMPLUS image to be dumped to "stderr" without converting the file. It is equivalent to using the −v option except that no file conversion takes place.

−a This option will cause pgmtorle to use the grayscale data to compute an alpha channel in the resulting RLE file. For any non-zero grayscale data, the alpha channel will contain a value of 255. The resulting RLE image file will contain one color channel and one alpha channel.

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.pgm
The name of the PBMPLUS image data file to be converted. This file must end in ".pgm". However, it is not necessary to supply the ".pgm" extension as it will be added to the supplied name if not already there.

EXAMPLES

pgmtorle −v test.pgm −o test.rle
While running in verbose mode, convert test.pgm to RLE format and store resulting data in test.rle.

pgmtorle −h test
Dump the header information of the PBMPLUS file called test.pgm.

SEE ALSO

ppmtorle(1), *rletoppm*(1), *urt*(1), *RLE*(5).

AUTHOR

Wesley C. Barris
Army High Performance Computing Research Center (AHPCRC)
Minnesota Supercomputer Center, Inc.

NAME

ppmtorle − convert a PBMPLUS/ppm image file into an RLE image file.

SYNOPSIS

ppmtorle [**−h**] [**−v**] [**−a**] [**−o** *outfile*] [*infile.ppm*]

DESCRIPTION

This program converts PBMPLUS full-color (ppm) image files into Utah *RLE*(5) image files. PBMPLUS/ppm image files contain the image dimensions and image data in the form of RGB triplets. When converting to an RLE file, the alpha channel may be optionally computed. The origins of PBMPLUS and Utah RLE files are in the upper left and lower left corners respectively, so this program automatically "flips" the image. The input can consist of several concatenated ppm images, in which case, the output will consist of several concatenated RLE images.

The RLE file will contain a "true color" image (24 bits). These RLE files may then be viewed using any RLE image viewer. When they are displayed on an 8 bit display, the image must be dithered. In order to produce a better looking image (on 8 bit displays), it is recommended that the image be quantizing (to 8 bit mapped color) prior to its display. This may be done by piping the output of this program into the Utah *mcut*(1) or *rlequant*(1) utilities. An example of this is shown later.

OPTIONS

−v This option will cause ppmtorle to operate in verbose mode. The header information is written to "stderr". Actually, there is not much header information stored in a PBMPLUS file, so this information is minimal.

−h This option allows the header of the PBMPLUS image to be dumped to "stderr" without converting the file. It is equivalent to using the −v option except that no file conversion takes place.

−a This option will cause ppmtorle to use the RGB data to compute an alpha channel in the resulting RLE file. For any non-zero RGB data, the alpha channel will contain a value of 255. The resulting RLE image file will contain three color channels and an alpha channel.

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.ppm
The name of the PBMPLUS image data file to be converted. This file must end in ".ppm". However, it is not necessary to supply the ".ppm" extension as it will be added to the supplied name if not already there.

EXAMPLES

ppmtorle −v test.ppm −o test.rle
While running in verbose mode, convert test.ppm to RLE format and store resulting data in test.rle.

ppmtorle test | mcut >test.rle
Convert test.ppm to RLE format and convert to 8 bit mapped color before storing data in test.rle

ppmtorle −h test
Dump the header information of the PBMPLUS file called test.ppm.

SEE ALSO

mcut(1), *pgmtorle*(1), *rlequant*(1), *rletoppm*(1), *urt*(1), *RLE*(5).

AUTHOR

Wesley C. Barris
Army High Performance Computing Research Center (AHPCRC)
Minnesota Supercomputer Center, Inc.

NAME

pyrmask − Blend two images together using Gaussian pyramids.

SYNOPSIS

pyrmask [**−l** *levels*] [**−o** *outfile*] *inmask outmask maskfile*

DESCRIPTION

Pyrmask blends two images together by first breaking the images down into separate bandpass images, combining these separate images, and then adding the new bandpass images back into a single output image. This can produce very seamless blends of digital images. The two images are combined on the basis of a third "mask" image. The resulting image will contain the *inmask* image where the mask contains a maximum value (255) and the *outmask* image where the mask contains zeros. This is done on a channel by channel basis, i.e. the maskfile should have data in each channel describing how to combine each channel of the *inmask* and *outmask* images. All three images must have exactly the same dimensions (both image size and number of channels). For best results, it's often useful to filter the mask image a little with *smush*(1) first.

OPTIONS

−l *levels*

How many pyramid levels to use (maximum is log(2) of image size).

−o *outfile*

If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

smush(1), *rleswap*(1), *urt*(1), *RLE*(5),

Burt and Adelson, "A Multiresolution Spline With Applications to Image Mosaics", *ACM Transactions on Graphics*, October 1983, V2 #4, p. 217.

Ogden, Adelson, Bergen and Burt, "Pyramid-based Computer Graphics", *RCA Engineer*, Sept/Oct 1985, p. 4.

AUTHOR

Rod Bogart

BUGS

The current implementation has very strict requirements for image sizes and dimensions. The extensive use of floating point computation makes it very slow for normal sized images. It also keeps all of the bandpass images in core at once, which requires considerable amounts of memory.

Pyrmask is built on top of a library of functions for working with Gaussian pyramids. This library has yet to be documented or fully tested.

NAME

rawtorle − Convert raw image data to RLE.

SYNOPSIS

rawtorle [**−N**] [**−s**] [**−r**] [**−w** *width*] [**−h** *height*] [**−f** *header-size*] [**−t** *trailer-size*] [**−n** *nchannels*] [**−a** [*alpha-value*]] [**−p** *scanline-pad*] [**−l** *left-scanline-pad*] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program is used to convert image data in any of a number of "raw" forms to the *RLE*(5) format. The expected input size is computed from the arguments, so that several images may be concatenated together and will be processed in sequence. In this case, the output file will contain several RLE images.

OPTIONS

−N The input is in non-interleaved order, as might be generated by the command
cat pic.r pic.g pic.b

−s The input is in scanline interleaved order.

−r Reverse the channel order. (E.g., data will be interpreted as ABGR instead of RGBA.)

−w *width*
Specify the width of the input image.

−h *height*
Specify the height of the input image.

−f *header-size*
This many bytes will be skipped before starting to read image data.

−t *trailer-size*
This many bytes will be skipped at the end of the image data.

−n *nchannels*
The input data has this many color channels.

−a [*alpha-value*]
Generate a constant alpha channel. The default value for *alpha-value* is 255.

−p *scanline-pad*
This many bytes will be skipped at the end of each scanline.

−l *left-scanline-pad*
This many bytes will be skipped at the beginning of each scanline.

−o *outfile*
If specified, output will be written to this file, otherwise it will go to stdout.

infile The input will be read from this file, otherwise, input will be taken from stdin.

The input data is assumed to have an alpha channel if there are 2 or 4 channels. The alpha channel is the last input channel unless **−r** is specified, in which case it is the first.

EXAMPLES

512x512 grayscale
rawtorle −w 512 −h 512 −n 1

640x512 raw RGB
rawtorle −w 640 −h 512 −n 3

picture.[rgb]
cat picture.[rgb] | rawtorle −w 640 −h 512 −n 3 −N −r
(I.e., separate red, green, blue image files. This subsumes *graytorle*(1).)

JPL ODL Voyager pics
rawtorle −w 800 −h 800 −f 2508 −t 1672 −n 1 −p 36

24bit Sun raster file
rawtorle −f 32 −w ... −h ... −n 3
(But *rastorle*(1) is easier.)

pic.{000-100}.[rgb]
cat pic.* | rawtorle −w ... −h ... −n 3 −s −r
(I.e., each color of each scanline is in a separate file.)

SEE ALSO

graytorle(1), *rastorle*(1), *urt*(1), *RLE*(5).

AUTHOR

Martin Friedmann

NAME

repos − reposition an RLE image

SYNOPSIS

repos [**−p** *xpos ypos*] [**−P** *xinc yinc*] [**−o** *outfile*] [*infile*]

DESCRIPTION

repos repositions an RLE image. Repos just changes the coordinates stored in the RLE header (see *RLE(5))*, no modification is made to the image itself.

OPTIONS

If neither of the following flags are specified, **−p** 0 0 is assumed.

−p *xpos ypos*
: Reposition the image to the absolute coordinates *xpos ypos.*

−P *xinc yinc*
: Move the image by *xinc yinc* pixels from where it currently is (relative movement).

infile The input will be read from this file, otherwise, input will be taken from stdin.

−o *outfile*
: If specified, output will be written to this file, otherwise it will go to stdout.

DIAGNOSTICS

Repos does not allow the image origin to have negative coordinates.

SEE ALSO

rlesetbg(1), *urt*(1), *RLE*(5).

AUTHORS

Rod Bogart, John W. Peterson

NAME

rlatorle − convert a Wavefront "rla" or "rlb" image file into an RLE image file.

SYNOPSIS

rlatorle [**−b**] [**−h**] [**−v**] [**−m**] [**−o** *outfile*] [*infile.rla*]

DESCRIPTION

This program converts Wavefront image files (rla or rlb formats) into Utah *RLE*(5) image files. Wavefront image files store RGB data as well as a matte channel. They also define a "bounding box" containing non-background pixels which is in many cases smaller than the total image area. Only this non-background area is run length encoded. When converting to an RLE file, the matte channel is stored as an alpha channel and the "bounding box" dimensions are ignored. It is for this reason that in general the RLE version of the file will be larger than its Wavefront counterpart.

The RLE file will contain a "true color" image (24 bits). These RLE files may then be viewed using any RLE image viewer. When they are displayed on an 8 bit display, the image will be dithered. In order to produce a better looking image (on 8 bit displays), it is recommended that the image be quantizing (to 8 bit mapped color) prior to its display. This may be done by piping the output of this program into the Utah *mcut*(1) or *rlequant*(1) utilities. An example of this is shown later.

OPTIONS

−b This option will cause rlatorle to convert from a Wavefront "rlb" image rather than use the default "rla" conversion.

−v This option will cause rlatorle to operate in verbose mode. The header information is written to "stderr".

−h This option allows the header of the wavefront image to be dumped to "stderr" without converting the file. It is equivalent to using the −v option except that no file conversion takes place.

−m This option will cause rlatorle to ignore the RGB data and use the matte channel information to produce a monochrome image. The resulting RLE image file will contain only one color channel instead of the usual four (RGB + alpha).

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.rla
The name of the Wavefront image data file to be converted. It is not necessary to supply the ".rla" or ".rlb" extension as it will be added to the supplied name if not already there.

EXAMPLES

rlatorle −v test.0001.rla −o test.rle
While running in verbose mode, convert test.0001.rla to RLE format and store resulting data in test.rle.

rlatorle test.0001.rla | mcut >test.rle
Convert test.0001.rla to RLE format and convert to 8 bit mapped color before storing data in test.rle

rlatorle −h test.0001.rla
Dump the header information of the Wavefront file called test.0001.rla.

rlatorle -b test.0001 | get4d
Convert test.0001.rlb to RLE format and display the resulting image.

SEE ALSO

mcut(1), *rlequant*(1), *rletorla*(1), *urt*(1), *RLE*(5).

AUTHOR

Wesley C. Barris
Army High Performance Computing Research Center (AHPCRC)
Minnesota Supercomputer Center, Inc.

NAME

rleaddcom − add picture comments to an RLE file.

SYNOPSIS

rleaddcom [**−d**] [**−i**] [**−o** *outfile*] *infile comments*

DESCRIPTION

The *rleaddcom* program will add one or more comments to an *RLE*(5) file. If *infile* is "−", it will read from the standard input. The modified *RLE*(5) file is written to the standard output if the **−o** *outfile* option is not given. All remaining arguments on the command line are taken as comments. Comments are nominally of the form *name=value* or *name*. Any comment already in the file with the same *name* will be replaced.

OPTIONS

−d Will cause matching comments to be deleted, no comments will be added in this case.

−i "In place." The input file will be rewritten with the added comments. This argument requires write permission to the directory containing *infile*, but does not require write permission for *infile*. Of the special file name cases described in *urt*(1), only compressed files may be updated in place. (It doesn't make sense to update the output of a pipe "in place", does it?)

If **−o** *outfile* is specified together with **−i**, then *outfile* will not be modified until *rleaddcom* has finished (this is similar to the way that *into*(1) works).

SEE ALSO

into(1), *rlehdr*(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

rleaddeof − Put an end of image marker on an RLE file.

SYNOPSIS

Superseded by *rlecat*(1).

NAME

rlebg − generate simple backgrounds

SYNOPSIS

rlebg [**−l**] [**−v** [*top* [*bottom*]]] [**−s** *xsize ysize*] [**−o** *outfile*] *red green blue* [*alpha*]

DESCRIPTION

rlebg generates a simple background. These are typically used for compositing below other images. The values *red green blue* specify the pixel values (between 0 and 255) the background will have. If *alpha* is not specified, it defaults to 255 (full coverage). *rlebg* generates both constant backgrounds and backgrounds with continuous ramps.

OPTIONS

−s *xsize ysize*

This is the size of the background image. The default is 512×480.

−l Generate a linear ramp of pixel values. If no ramp flag is given, *rlebg* generates a constant background.

−v *top bottom*

Generate a variable ramp, using a quadratic function (this looks best with gamma corrected images). *top* and *bottom* are the fractions of the full color values at the top and bottom of the image. The defaults are 1.0 0.1, respectively. If both **−v** and **−l** are given, then a linear ramp function is used instead of a quadratic ramp.

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

SEE ALSO

rlesetbg(1), *urt*(1), *RLE*(5).

AUTHOR

Rod Bogart

NAME

rlebox − print bounding box for image in an RLE file.

SYNOPSIS

rlebox [**−c**] [**−m** *margin*] [**−v**] [*infile*]

DESCRIPTION

This program prints the bounding box for the image portion of an *RLE*(5) file. This is distinct from the bounds in the file header, since it is computed solely on the basis of the actual image. All background pixels are ignored.

OPTIONS

−c Print the numbers in the order that crop wants them on its command line. The default order is *xmin xmax ymin ymax*. If this option is specified, the bounds are printed in the order *xmin ymin xmax ymax*. Thus, a file *foo.rle* could be trimmed to the smallest possible image by the command

crop ‘rlebox −c foo.rle‘ foo.rle

−m *margin*

Pads the output values by the margin given.

−v Verbose mode: label the numbers for human consumption.

infile Name of the *RLE* file (defaults to standard input).

SEE ALSO

crop(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

rlecat − concatenate and repeat images.

SYNOPSIS

rlecat [**−c**] [**−n** *repeat-count*] [**−o** *outfile*] [*files ...*]

DESCRIPTION

This program will concatenate all the input *RLE*(5) images, adding titles, and optionally repeating the images a specified number of times. For each input file, it copies all images to the output file. If an image does not have a *title* or *TITLE* comment, and the input is not coming from the standard input, then the file name (and an image number, if it is not the first image in the file) is added as a *TITLE* comment. If the input file were named 'images.rle', the first image would be given a comment *TITLE=images.rle*, the second would get a comment *TITLE=images.rle(2)*, and so on.

OPTIONS

−c With **−n**, specifies that the output images should be "collated". In other words, the repeat sequence will be 1 2 3 ... 1 2 3 ... instead of the default of 1 1 ... 2 2 ... 3 3 ...

−n *repeat-count*
Specifies that each input image should be repeated *repeat-count* times. The "repeat unit" (if **−c** is specified, this is the entire concatenated sequence of input images, otherwise it is just each image, separately) is written to a temporary file, and then copied to the output the requisite number of times.

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

files The input will be read from these files. If a file name is "−", or none are specified, the input will be read from the standard input stream.

EXAMPLES

rlebg 128 128 128 | rlecat -n 25
Generates 25 copies of a gray background; useful for using *rlecomp*(1) to put background on an animation sequence (with 25 or fewer frames).

rlecat *.rle | <some processing> | getx11
Adds *TITLE* comments so the individual images are correctly identified by *getx11*(1).

rlecat -c -r 3 anim*.rle
Generates an animation with 3 repeats of the action.

rlecat -r 3 anim*.rle
Generates a "triple-framed" animation − each frame is repeated 3 times.

FILES

/tmp/rlecatXXXXXXXX

SEE ALSO

rleaddcom(1), *rlehdr*(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Michigan

BUGS

If the /tmp directory is not writable, or if there is not sufficient space on /tmp to hold a repeat unit, the program will not work correctly.

NAME

rleccube − Make a picture of a color cube.

SYNOPSIS

rleccube [**−w** *squares-wide*] [**−o** *outfile*] [*cube-side*]

DESCRIPTION

This program computes an *RLE*(5) image of slices through the RGB color cube. The arguments control the size of the cube and the arrangement of the slices into an image. Slices are taken in planes of constant red, with green varying along the "x" axis and blue along the "y" axis within a slice. The slice for red=0 is placed in the lower left corner of the image; red increases along the bottom row, then to the left of the next row, and so on. The *rleswap*(1) program can be used to get an image with slices of constant green or blue.

OPTIONS

−w *squares-wide*

The number of slices in a row will be *squares-wide*. The default is the smallest divisor of *cube-side* larger than *sqrt(cube-side)*. If *squares-wide* is not an exact divisor of *cube-side*, the top row will be filled in with slices starting from red near 0.

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

cube-side

The number of samples on each side of the cube. Each slice will be *cube-sidexcube-side*, and there will be *cube-side* slices. The default value is 64.

SEE ALSO

rleswap(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas

BUGS

It really should fill in the excess space in the last row with black.

NAME

rleClock − Generate a clock face in RLE format

SYNOPSIS

rleClock [*options*] [**−o** *outfile*]

DESCRIPTION

This program generates an analog clock face in *RLE*(5) file format and writes it to *outfile* or standard output. The picture is a standard clock face with optional digital representation above. The user has control over the colors of the portions of the clock face, the text, and the text background. The user also has control over the clock configuration: number of ticks, scale of the big and little hands, the values of the big and little hands, and the format used to generate the digital portion.

By default, **rleClock** generates a standard analog clock face displaying the current time and with no digital portion. This default face is transparent, that is, the alpha channel is only defined for the clock outline, tick marks, and the hands.

On those options that expect colors, three numbers must be given after the option switch. These are values for red, green, and blue on a scale of zero through 255. Those color options that are capitalized indicate the colors for the filled regions (optional for the clock face and text but default for the hands). Those that are not capitalized are for lines that either outline or constitute the feature (the clock face is default, but they're optional for the hands).

OPTIONS

−help Prints a synopsis of the options.

The options that control the value displayed by the clock are

−ls *FLOAT*
: This specifies the full scale (360 degrees) of the little hand. Default is 12.

−lv *FLOAT*
: This specifies the value of the little hand, expressed in units of the little hand full scale. Default is the current hour time on a 12-hour scale.

−bs *FLOAT*
: This specifies the full scale (360 degrees) of the big hand. Default is 60.

−bv *FLOAT*
: This specifies the value of the big hand, expressed in units of the big hand full scale. Default is the current minute time.

The following options manage the display configuration of the clock:

−x *INT* The INT specifies the width of the clock in pixels. Default is 128.

−cy *INT*
: The INT specifies the height of the clock face (minus text portion) in pixels. The default is 128.

−ty *INT*
: The INT specifies the height in pixels of the text portion of the display. If it is zero (the default), no text portion is displayed.

−t *INT* This specifies the number of tick marks to place around the clock. The default is 12.

−lw *INT*
: This specifies the line width in pixels of the clock face, the tick marks, the optional hand borders, and the text. The default is one, but two or three give better looking clocks.

−tf *STR*
: The string describes how to show the digital portion of the clock. The rules for forming STR are the same as for *printf* format strings, that is, a percent sign, optionally followed by field width values, followed by a key letter. In this case, the key letter may be **b, l, B,** or **L.** Lower case **b** means to insert the integer value of the big hand and upper case **B** means to insert the floating point value of the big hand. Lower case **l** means to insert the integer value of the little hand and upper case **L** means to insert the floating point value of the little hand.

−fc *R G B*
This specifies the color in red, green, and blue, of the clock face.

−Fc *R G B*
This specifies the color to fill in inside the clock face, under the hands. If this option is not supplied, the clock is generated with no inside-face background (by use of the alpha channel).

−Hc *R G B*
This specifies the color to draw in the hands with.

−hc *R G B*
This specifies the color to draw the outlines of the hands. If it is not given, no outlines are drawn on the edges of the hands.

−tc *R G B*
This specifies the color of the text above the clock. It only has effect if a text height (-ty) is supplied.

−Tc *R G B*
This specifies the color of a background field to place behind the text. If omitted, no background (zero alpha channel) is drawn.

EXAMPLES

rleClock
Generates a transparent clock face showing the current time and no digital representation.

rleClock −ty 32
Generates a current-time clock with digital representation above.

rleClock −Fc 255 0 0 −Hc 0 0 255 −lw 3 −ty 96 −tc 0 255 0 −Tc 128 128 128
Generates a clock with a red inside, white face, blue hands, wide lines, tall text field, green test, and grey text background.

rleClock −ty 32 −bs 10 −bv 4.51 −ls 100 −lv 45.1 −tf "%2l.%2.2B"
Generates a clock with the scale of the big hand set to 10 and it's value at 4.51, the scale and value of the little hand as 100 and 45.1, and the format for the digital portion formatted as **%2d.%2.2f** to print the integer little hand value (two spaces) and the floating point big hand value.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Robert L. Brown, RIACS, NASA Ames Research Center

BUGS

Not thoroughly checked when the line width is cranked up. May dump core.

NAME

rlecomp − Digital image compositor

SYNOPSIS

rlecomp [**−o** *outfile*] *Afile operator Bfile*

DESCRIPTION

rlecomp implements an image compositor based on presence of an alpha, or matte channel the image. This extra channel usually defines a mask which represents a sort of a cookie-cutter for the image. This is the case when alpha is 255 (full coverage) for pixels inside the shape, zero outside, and between zero and 255 on the boundary. If *Afile* or *Bfile* is just a single −, then *rlecomp* reads that file from the standard input.

The operations behave as follows (assuming the operation is "*A operator B*"):

over The result will be the union of the two image shapes, with *A* obscuring *B* in the region of overlap.

in The result is simply the image *A* cut by the shape of *B*. None of the image data of *B* will be in the result.

atop The result is the same shape as image *B*, with *A* obscuring *B* where the image shapes overlap. Note this differs from **over** because the portion of *A* outside *B*'s shape does not appear in the result.

out The result image is image *A* with the shape of *B* cut out.

xor The result is the image data from both images that is outside the overlap region. The overlap region will be blank.

plus The result is just the sum of the image data. Output values are clipped to 255 (no overflow). This operation is actually independent of the alpha channels.

minus The result of *A* − *B*, with underflow clipped to zero. The alpha channel is ignored (set to 255, full coverage).

diff The result of abs(*A* − *B*). This is useful for comparing two very similar images.

add The result of *A* + *B*, with overflow wrapping around (*mod* 256).

subtract
The result of *A* - *B*, with underflow wrapping around (*mod* 256). The **add** and **subtract** operators can be used to perform reversible transformations.

SEE ALSO

urt(1), *RLE*(5),
"Compositing Digital Images", Porter and Duff, *Proceedings of SIGGRAPH '84* p.255

AUTHORS

Rod Bogart and John W. Peterson

BUGS

The other operations could be optimized as much as **over** is.

Rlecomp assumes both input files have the same number of channels.

NAME

rledither − Floyd Steinberg dither an image to the given colors.

SYNOPSIS

rledither [**−e** *edge_factor*] [**−l** *nchan length*] **−{tf}** *mapfile* [**−o** *outfile*] [*infile*]

DESCRIPTION

This program accepts an *RLE*(5) file and a file of colormap entries, and dithers the image to those colors. Edge enhancement is also performed, if specified.

OPTIONS

−e *edge_factor*

An *edge_factor* of zero means no edge enhancement (the default). A value of 1.0 looks pretty good for most images.

−l *nchan length*

Specifies the number of channels in the colormap, and the number of entries in each channel. The default is 3 channels of 256 entries, which is appropriate for an eight bit color display.

−{tf} *mapfile*

The *mapfile* must contain at least *nchan***length* values in the range 0 to 255. The **−t** flag causes *mapfile* to be read as R G B R G B R G B... The **-f** flag implies the entries are listed as R R R... G G G... B B B...

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

mcut(1), *rlehdr*(1), *rlequant*(1), *urt*(1), *RLE*(5).

AUTHOR

Rod G. Bogart, University of Michigan

BUGS

It should read colormaps from RLE files, too. For the moment, edit the output from *rlehdr -m*.

NAME

rleflip − Invert, reflect or rotate an image.

SYNOPSIS

rleflip −{rlhv} [**−o** *outfile*] [*infile*]

DESCRIPTION

Rleflip inverts, reflects an image; or rotates left or right by 90 degrees. The picture's origin remains the same. If no input file is specified, the image is read from standard input. For rotations of other than 90 degrees, use *fant*(1).

OPTIONS

Exactly one of the following flags must be given:

−r Rotate the image 90 degrees to the right

−l Rotate the image 90 degrees to the left

−h Reflect the image horizontally

−v Flip the image vertically

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

SEE ALSO

fant(1), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson

NAME

rlegrid − create grids and checkerboards in rle format

SYNOPSIS

rlegrid [**−b** *bg_color*] [**−c**] [**−f** *fg_color*] [**−o** *outfile*] [**−s** *xsize ysize*] [**−w** *width*]

DESCRIPTION

rlegrid generates simple grid and checkboard patterns.

OPTIONS

−b *bg_color*

Specifies the background color value. Should between 0 and 255. Default is 0.

−c Generate checkboards. With the **-c** option, rlegrid will generate a checkboard with squares of size *width* on a side. Squares will alternate between the foreground and background colors.

Without the **-c** option, rlegrid will generate a grid. Grid lines will be *width* apart and will be in the foreground color. The remainder of the image will be in the background color.

−f *fg_color*

Specifies the background color value. Should between 0 and 255. Default is 255.

−o *outfile*

Specifies where to place the resulting image. The default is to write to stdout. If *outfile* is "−", the output will be written to the standard output stream.

−s *xsize yssize*

This is the size of the resulting image. Default is 512x512.

−w *width*

The spacing between grid lines or checkboard squares. The default is 16.

SEE ALSO

rlebg(1),

AUTHOR

James S. Painter

NAME

rlehdr − Prints the header of an RLE file

SYNOPSIS

rlehdr [**−b**] [**−c***comment-names*] [**−d**] [**−m**] [**−v**] [*files* ...]

DESCRIPTION

This program prints the header of *RLE*(5) files in a human readable form. If the optional *files* argument is not supplied, input is read from standard input.

OPTIONS

−b Print the information in a "brief" one-line form. The form of the output line is
name: [*l*,*b*]+[*xs*,*ys*]x*nc*+A, BG=*color*, map=*N*x*L*, (C)
Where [*l*,*b*] is the position of the lower-left corner of the image, [*xs*,*ys*] is the size of the image in pixels, *nc* is the number of channels saved, +A is present if an alpha channel is saved. BG= or OV= indicate that a background color was saved; OV= means that the existing background is not cleared to the background color before the image is read (this was used for a cheap form of compositing, but is basically obsolete now). *color* is the saved background color. The map= entry will be present only if a color map was saved; *N* is the number of channels in the color map and *L* is the length of the map. Finally (C) is appended if there are comments present.

−c *comment-names*
If a comment identified by any of the words in the comma-separated list *comment-names* is present in the input file, its first line will be printed. Each name is tried, in turn, and only the first match is printed. If no match is found, but comments are present, (C) will be printed. The **−c** flag implies **−b**.

−d Dump a very verbose version of the image contents as text to the standard error output stream.

−m Print out the color map information. **−v** Prints the raster toolkit version and patch level. No input files will be processed if this option is given.

EXAMPLES

rlehdr image.rle
Print the header information for all images in the file image.rle.

rlehdr -m image.rle
Also print the color map contents, if one is present.

rlehdr -b *.rle
Print one line summaries of all the images in the directory.

rlehdr -c title,TITLE *.rle
Print one line summaries of all the images, and print the title of any that have a title comment.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

rlehisto − generate histogram of RLE image.

SYNOPSIS

rlehisto [**−b**] [**−c**] [**−t**] [**−h** *height*] [**−o** *outfile*] [*infile*]

DESCRIPTION

Rlehisto counts the pixel values in an RLE file, producing an RLE file graphing frequency of occurrence. The horizontal axis runs from pixel value 0 on the left to pixel value 255 on the right. The height indicates the number of pixels seen for each pixel value. Histograms are computed independently for each channel, scaled identically, and then overlaid.

The following options are available:

−b Don't count the background pixel values when scaling the histogram. This is useful if most pixels are colored the background color, so that the interesting part of the histogram would be too small. This option is ignored if **−c** is specified.

−c Output cumulative values instead of discrete values.

−t Print the totals instead of generating the histogram as an RLE file.

−h *height* Scale the output image to the specified height. The default is 256.

−o *outfile*
Direct the output to *outfile.*

SEE ALSO

urt(1), *RLE*(5).

AUTHORS

Gregg Townsend, University of Arizona; Rod Bogart, University of Utah.

NAME

rleintrp − Interpolate between 2 RLE images .

SYNOPSIS

rleintrp [**−o** *prefixe-out*] [**−1** *file1*] [**−2** *file2*] [**−n** *nbimages*]

DESCRIPTION

This program create nbimages files of RLE images resulting of linear interpolati *RLE*(5) images must have the same caracteristics (Number of colors, channel alpha, backg

OPTIONS

−o *prefixe-out*
specify the prefix of the names of output files. The names are composed of this *file1* (or of *file2)* is used as prefix. If a name cannot be create by the above methods it is arbitra

−1 *file1* Specify the name of the file containing the initial image for interpolate. If op *-2* exist we interpolate between a black image and the existing file (’fondu au noir *file1* is ’-’ standard input is used.

−2 *file2* Specify the name of the file containing the initial image for interpolate. If op *-1* exist we interpolate between a black image and the existing file (’fondu au noir *file2* is ’-’ standard input is used.

−n *nbimages*
Specify The number of images to create. By default nbimages = 1. The value is li

EXAMPLES

rleintrp −1 image1 −o fondu.rle −n 5

Interpolate 1 image between image1 and image2.
rleintrp −1 image1 −2 image2

Interpolate 3 images with names of files inter_XXX.rle :
rleintrp −1 image1 −2 image2 -o inter.rle −n 3

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Michel Gaudet SLX Onera CHATILLON (France)

NAME

rleldmap − Load a new color map into an RLE file

SYNOPSIS

rleldmap [**−{ab}**] [**−n** *nchan length*] [**−s** *bits*] [**−l** [*factor*]] [**−g** *gamma*] [**−{tf}** *file*] [**−m** *files ...*] [**−r** *rlefile*] [**−o** *outfile*] [*infile*]

DESCRIPTION

The program will load a specified color map into an *RLE*(5) file. The color map may be computed by *rleldmap* or loaded from a file in one of several formats. The input is read from *infile* or stdin if no file is given, and the result is written to *outfile* or stdout.

The following terms are used in the description of the program and its options:

input map:
: A color map already in the input RLE file.

applied map:
: The color map specified by the arguments to *rleldmap*. This map will be applied to or will replace the input map to produce the output map.

output map:
: Unless **−a** or **−b** is specified, this is equal to the applied map. Otherwise it will be the composition of the input and applied maps.

map composition:
: If the applied map is composed *after* the input map, then the output map will be *applied map*[*input map*]. Composing the applied map before the input map produces an output map equal to *input map*[*applied map*]. The maps being composed must either have the same number of channels, or one of them must have only one channel. If an entry in the map being used as a subscript is larger than the length of the map being subscripted, the output value is equal to the subscript value. The output map will be the same length as the subscript map and will have the number of channels that is the larger of the two. If the input map is used as a subscript, it will be downshifted the correct number of bits to serve as a subscript for the applied map (since the color map in an *RLE*(5) file is always stored left justified in 16 bit words). This also applies to the applied map if it is taken from an *RLE*(5) file (**−r** option below). Note that if there is no input map, that the result of composition will be exactly the applied map.

nchan: The number of separate lookup tables (channels) making up the color map. This defaults to 3.

length: The number of entries in each channel of the color map. The default is 256.

bits: The size of each color map entry in bits. The default value is the log base 2 of the length.

range: The maximum value of a color map entry, equal to 2**bits − 1.

OPTIONS

−a Compose the applied map *after* the input map.

−b Compose the applied map *before* the input map. Only one of **−a** or **−b** may be specified.

−n *nchan length*
: Specify the size of the applied map if it is not 3x256. The *length* should be a power of two, and will be rounded up if necessary. If applying the map *nchan* must be either 1 or equal to the number of channels in the input map. It may have any value if the input map has one channel or is not present.

−s *bits* Specify the size in bits of the color map entries. I.e., only the top *bits* bits of each color map entry will be set.

Exactly one of the options **−l**, **−g**, **−t**, **−f**, **−m**, or **−r**, must be specified.

−l *factor*
: Generate a linear applied map with the *nth* entry equal to
*range * min(1.0, factor*(n/(length−1)))*.
Factor defaults to 1.0 if not specified. Negative values of *factor* will generate a map with

values equal to

*range * max(0.0, 1.0 − factor*(n/(length−1)))*.

−g *gamma*

Generate an applied map to compensate for a display with the given gamma. The *nth* entry is equal to

*range * (n/(length−1))**(1/gamma)*.

−t *file* Read color map entries from a table in a text file. The values for each channel of a particular entry follow each other in the file. Thus, for an RGB color map, the file would look like:

red0 green0 blue0
red1 green1 blue1
... ...

Line breaks in the input file are irrelevant.

−f *file* Reads the applied map from a text file, with all the entries for each channel following each other. Thus, the input file above would appear as

red0 red1 red2 ... (*length* values)
green0 green1 green2 ... (*length* values)
blue0 blue1 blue2 ... (*length* values)

As above, line breaks are irrelevant.

−m *files ...*

Read the color map for each channel from a separate file. The number of files specified must equal the number of channels in the applied map. (Note: the list of files must be followed by another flag argument or by the null flag **−−** to separate it from the *infile* specification.

−o *outfile*

The output will be written to the file *outfile* if this option is specified. Otherwise the output will go to *stdout*.

infile The input will be taken from this file if specified. Otherwise, the input will be read from *stdin*.

SEE ALSO

applymap(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

rlemandl − Compute images of the Mandelbrot set.

SYNOPSIS

rlemandl [**−o** *outfile*] [**−s** *xsize ysize*] [**−v**] *real imaginary width*

DESCRIPTION

Rlemandl computes images of the Mandelbrot set as an eight bit gray scale image. The *real* and *imaginary* arguments specify the center of the area in the complex plane to be examined. *Width* specifies the width area to be examined.

OPTIONS

−o *outfile*

If specified, output will be written to this file, otherwise it will go to stdout.

−s *xsize ysize* Specify the resolution of the image (in pixels).

−v Print a message after every 50 lines are generated.

SEE ALSO

urt(1),
"Computer Recreations," *Scientific American*, August 1985.

AUTHOR

John W. Peterson, University of Utah.

BUGS

What a frob. Gratuitous features are left as exercise to the reader. The command name is spelled incorrectly.

NAME

rlenoise − Add random noise to an image

SYNOPSIS

rlenoise [**−n** *amount*] [**−o** *outfile*] [*infile*]

DESCRIPTION

Rlenoise adds uniform random noise to an image. The peak-to-peak amplitude of the noise can be specified with the **−n** flag, the default value is 4. This program may be useful for trying to deal with quantization in an output device, if you are able to trade spatial resolution for color resolution, and you don't have a good characterization of the quantization function.

OPTIONS

infile The input will be read from this file, otherwise, input will be taken from stdin.

−o *outfile*

If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Michigan.

BUGS

Of limited utility.

NAME

rlepatch − patch smaller RLE files over a larger image.

SYNOPSIS

rlepatch [**−o** *outfile*] *infile patchfiles...*

DESCRIPTION

Rlepatch puts smaller RLE files on top of a larger RLE image. One use for rlepatch is to place small "fix" images on top of a larger image that took a long time to compute. Along with *repos*(1), *rlepatch* can also be used as a simple way to build image mosaics.

Unlike *rlecomp*(1), *rlepatch* does not perform any arithmetic on the pixels. If the patch images overlap, the patches specified last cover those specified first.

If the input files each contain multiple images, they are treated as streams of images merging to form a stream of output images. I.e., the *n*th image in each input file becomes part of the *n*th image in the output file. The process ceases as soon as any input file reaches its end.

OPTIONS

infile The background image will be read from this file. If input is to be taken from stdin, "−" must be specified here.

−o *outfile*
If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

rlecomp(1), *crop*(1), *repos*(1), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson, University of Utah.

BUGS

Rlepatch uses the "row" interface to the RLE library. It would run much faster using the "raw" interface, particularly for placing small patches over a large image. Even fixing it to work like *rlecomp* (which uses "raw" mode only for non-overlapping images) would make a major improvement.

NAME

rleprint − Print the values of all the pixels in the file.

SYNOPSIS

rleprint [**−a**] [*infile*]

DESCRIPTION

This program reads an *RLE*(5) image and prints the values of all the pixels to the standard output. Each pixel is printed on a single line. For example, a count of all the unique pixel values in the file could be obtained by

rleprint pic.rle | sort −u | wc

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

OPTIONS

−a Print the alpha value (if available) as the last entry on the line.

SEE ALSO

rlehdr(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas

BUGS

This program is of limited utility because of the sheer volume of output it generates.

NAME

rlequant − variance based color quantization for RLE images

SYNOPSIS

rlequant [**−b** *bits*] [**−c**] [**−d**] [**−f**] [**−i** *cubeside*] [**−m**] [**−n** *colors*] [**−r** *mapfile*] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program quantizes the colors in an RLE image using a variance-based method. See *colorquant*(3) for more details on the method.

−b *bits* The colors in the input image will be "prequantized" to this many bits before applying the variance-based method. Two internal tables of size **2^(3*bits)** are allocated, so values of *bits* greater than 6 are likely to cause thrashing or may prevent the program from running at all. The default value of *bits* is 5. It must be less than or equal to 8 and greater than 0.

−c Only the color map will be output; the image will not be digitized. The output file will be a 0x0 *RLE* file with a color map, suitable for input to *rleldmap*(1), *rledither*(1), or *rlequant -r*.

−d Floyd Steinberg dithering is performed on the output. This is very helpful for images being quantized to a small number of colors.

−f If this option is specified, a faster approximation will be used. In most cases, the error so introduced will be barely noticeable.

−i *cubeside*

Initializes the output color map with a "color cube" of size *cubeside*^3. I.e., if *-i 2* were specified, the 8 corners of the color cube (black, red, green, blue, yellow, cyan, magenta, white) would be added to the output colormap. This reduces the number of colors available for quantization. The color cube will be used to quantize the output image, but will not otherwise affect the choice of representative colors.

−m Computes a single color map suitable for quantizing all the input images. This is useful when the quantized images will be used as a "movie" (e.g., with the **−m** flag of *getx11*(1)). The input may not come from a pipe when this option is specified, unless **−c** is also specified.

−n *colors*

The output image will be quantized to at most *colors* colors. It might have fewer if the input image has only a few colors itself. The default value of *colors* is 256 − *cubesize*^3 − *mapsize*. It must be less than or equal to 256. If a color cube (**−c**) or an input map (**−r**) is given, *colors* may be 0; otherwise it must be greater than 0.

−r *mapfile*

The color map from the RLE file *mapfile* will be added to the output color map. The number of colors in the input color map, *mapsize* is calculated as follows: If a *color_map_length* comment is present in *mapfile*, its value is used. If not, the size of the color map (usually 256) is used (the *rlehdr*(1) program will display the color map size and the comment, if present). The input color map will be used to quantize the output image, but will not otherwise affect the choice of representative colors. If the combination *-n 0 -r mapfile* is specified, then *rlequant* will just quantize (and dither, if requested) the input images to the given colormap. This is usually faster than using *rledither*.

−o *outfile*

The output will be written to the file *outfile*. If not specified, or if *outfile* is "−", the output will be written to the standard output stream.

infile This file contains one or more concatenated RLE images. Each will be processed in turn. A separate quantization map will be constructed for each image. If not specified, or if *infile* is "−", the image(s) will be read from the standard input stream.

EXAMPLES

rlequant file.rle

Quantizes *file.rle* to 256 colors using a 5-bit pre-quantization. If *file.rle* has multiple images, each will get its own (different) colormap.

rlequant −m file.rle

Quantizes *file.rle* to 256 colors using a 5-bit pre-quantization. If *file.rle* has multiple images, they will all be used to choose the color map, and will all be quantized to the same color map.

cat *.rle | rlequant −m −c >map.rle ;
cat *.rle | rlequant −n 0 −r map.rle

Computes a single colormap based on all the images in the files **.rle*, then quantizes each image to that color map. The output is the stream of quantized images.

rlequant -i 4 -d file.rle

Compute 192 representative colors for each image in *file.rle*, add a 4x4x4 color cube to the resulting color map, and then quantize the image to the resulting set of colors with dithering.

rlequant -b 6 file.rle

Quantize file.rle to 256 colors using a 6-bit prequantization. This provides slightly more precision in color matching than does a 5-bit prequantization. It also runs significantly slower and requires approximately 8 times the memory for its intermediate storage.

SEE ALSO

mcut(1), *rledither*(1), *rlehdr*(1), *urt*(1), *colorquant*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
Craig Kolb (Yale University) wrote the color quantization code.
Rod Bogart wrote the dithering code.

NAME

rlescale − produce gray scale images.

SYNOPSIS

rlescale [**−c**] [**−n** *nsteps*] [**−o** *outfile*] [*xsize*] [*ysize*]

DESCRIPTION

Rlescale produces an RLE image containing a (more-or-less) standard gray scale image. Along the bottom are 8 colored patches (in the standard primary and secondary colors). Above these are a sequences of logarithmically scaled gray patches. By default, a 16 step scale is produced. The size of the output file (default 512 by 480) can be set with the *xsize* and *ysize* arguments.

OPTIONS

−c Produce red, green, blue, and gray scales.

−n *nsteps*

Specify the number of steps to be produced.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Michigan.

BUGS

Can’t make an image narrower than 3 * *nsteps* pixels wide.

NAME

rleselect − Select images from an RLE file.

SYNOPSIS

rleselect [**−i** *infile*] [**−o** *outfile*] [**−v**] [*image-numbers ...*]

DESCRIPTION

This program selects images from an *RLE*(5) file containing multiple concatenated images. The selected images are specified by number; the first image in the file is number 1. A negative number in the *image-numbers* list means that all images from the previous number in the list to the absolute value of this number should be included. A zero in the list is taken as '−infinity', so that all images from the previous number to the last image in the file will be included. To try to clarify this, some examples are included below.

OPTIONS

−i *infile* The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

−v Verbose output.

EXAMPLES

rleselect 1 4 5
Selects image 1, 4, and 5.

rleselect 4 1 5
Also selects image 1, 4, and 5.

rleselect 1 −4 5
Selects images 1 through 4 and 5 (i.e., 1 through 5).

rleselect 3 0
Selects images 3 through the last.

rleselect −4
Selects images 1 through 4.

SEE ALSO

rlesplit(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas

NAME

rlesetbg − Set the background value in the RLE header.

SYNOPSIS

rlesetbg [**−{DO}**] [**−c** *bgcolor ...*] [**−o** *outfile*] [*infile*]

DESCRIPTION

rlesetbg sets the background color field in the image header of an *RLE*(5) image (none of the actual pixels are changed). If *infile* isn't specified, the image is read from stdin.

The background color in the header is used to save space in the run-length encoded file. Runs of background-colored pixels longer than 2 pixels are simply not saved. (Doing this for runs of 1 or 2 background pixels can make the saved image larger than if no encoding were done.) Therefore, changing the background color with *rlesetbg* may still leave some pixels saved in the original background color. The **−D** option will delete the background color altogether from the header; this can be useful in certain circumstances, but can also lead to very strange results.

OPTIONS

−D Delete any background specification that might be present.

−O Specifies that the image has no background, it overlays existing images.

−c *bgcolor ...*

Specifies the color values to set the background to. There should be at least as many values as there are color channels in the image. Use −− or another option to separate the list of colors from *infile*.

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

AUTHORS

John W. Peterson and Rod Bogart

SEE ALSO

repos(1), *urt*(1), *RLE*(5).

BUGS

This should really be part of a single program that does all header munging...

NAME

rlespiff − Use simple contrast enhancement to "spiff up" an image.

SYNOPSIS

rlespiff [**−b** *blacklevel*] [**−s**] [**−t** *threshold*] [**−w** *whitelevel*] [**−o** *outfile*] [*infile*]

DESCRIPTION

Rlespiff "spiffs up" an image by stretching the contrast range so that the darkest pixel maps to black and the lightest to white. If the **−s** flag is given, the color channels will be treated separately. This will likely cause some drastic color shifts.

OPTIONS

−b *blacklevel*
: The darkest input pixel will map to this pixel value in the output image. The default is 0.

−s
: If specified, each color channel will be mapped separately.

−t *threshold*
: This argument controls the number of samples of a pixel value that should be considered insignificant (and will therefore be ignored). It is specified in pixels/million. A threshold of 4 applied to a 512x512 image would mean that any value that existed at only one pixel would be ignored. The default value is 10.

−w *whitelevel*
: The lightest input pixel will map to this pixel value in the output image. The default is 255.

−o *outfile*
: If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile
: The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Spencer W. Thomas

NAME

rlesplice − Splice two RLE files together horizontally or vertically.

SYNOPSIS

rlesplice −{hv} [**−c**] [**−o** *outfile*] *infile1 infile2*

DESCRIPTION

rlesplice splices two RLE images together either vertically or horizontally. If one image is smaller, then its background value or black is used to pad that image to equal the larger dimension in the other image. The **−c** flag is used to specify whether the smaller image should be centered when put next to the larger. Presently the two images must have the same number of color channels, the same presence of an alpha channel, and the same colormap size and length. The colormap from the first image is used for the resultant image.

SEE ALSO

rlecomp(1), *rlepatch*(1), *unslice*(1), *urt*(1), *RLE*(5).

AUTHOR

Martin R. Friedmann

NAME

rlesplit − split a file of concatenated RLE images into separate image files

SYNOPSIS

rlesplit [**−n** *number* [*digits*]] [**−o** *prefix*] [*infile*]

DESCRIPTION

This program will split a file containing a concatenated sequence of *RLE*(5) images into separate files, each containing a single image. The output file names will be constructed from the input file name or a specified prefix, and a sequence number. If an input *infile* is specified, then the output file names will be in the form "*rlefileroot*.#.rle", where *rlefileroot* is *infile* with any ".rle" suffix stripped off. If the option **−o** *prefix* is specified, then the output file names will be of the form "*prefix*.#.rle". If neither option is given, then the output file names will be in the form "#.rle". Input will be read from *infile* if specified, from standard input, otherwise. File names will be printed on the standard error output as they are generated.

The option **−n** allows specification of an initial sequence number, and optionally the number of digits used for the sequence number. By default, numbering starts at 1, and numbers are printed with 3 digits (and leading zeros).

SEE ALSO

rleselect(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas

NAME

rlestereo − produce anaglyph from stereo pair

SYNOPSIS

rlestereo [**−l** *leftscale*] [**−r** *rightscale*] *leftimage rightimage*

DESCRIPTION

Rlestereo reads the two named RLE files and produces a single image suitable for viewing with red-blue or red-green glasses.

The 'left' image is converted to greyscale and written on the red channel. The 'right' image is converted to greyscale and written on the blue or green channel. The intensity of the two channels may be scaled in order to compensate for the relative intensities of the two base colors as viewed through the glasses.

OPTIONS

−g The right-eye image is written to the green channel rather than the blue.

−l *leftscale*

Scale the intensities of the left-eye greyscale image by the given amount. The default value is 0.7.

−r rightscale

Scale the intensities of the right-eye greyscale image by the given amount. The default value is 1.0.

SEE ALSO

urt(1), *RLE*(5).

Cardboard glasses are available at many comic book stores.

AUTHOR

Craig Kolb, Yale University

NAME

rleswap − swap the channels in an RLE file.

SYNOPSIS

rleswap [**−v**] [**−f** *from-channels,...*] [**−t** *to-channels,...*] [**−d** *delete-channels,...*] [**−p** *channel-pairs,...*] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program can be used to select or swap the color channels in a *RLE*(5) file. The major options provide four different ways of specifying a mapping between the channels in the input file and the output file. Only one of the options **−f**, **−t**, **−d**, or **−p** may be specified. If the optional *infile* is not given, input will be read from standard input. A new *RLE*(5) file will be written to the standard output or to *outfile*, if specified. The output image will be similar to the input, except for the specified channel remappings.

OPTIONS

−v Print the channel mappings that will be performed on the standard error output.

−f Following this option is a comma separated list of numbers indicating the input channel that maps to each output channel in sequence. I.e., the first number indicates the input channel mapping to output channel 0. The alpha channel will be passed through unchanged if present. Any input channels not mentioned in the list will not appear in the output.

−t Following this option is a comma separated list of numbers indicating the output channel to which each input channel, in sequence, will map. I.e., the first number gives the output channel to which the first input channel will map. No number may be repeated in this list. The alpha channel will be passed through unchanged if present. Any output channel not mentioned in the list will not receive image data. If there are fewer numbers in the list than there are input channels, the excess input channels will be ignored. If there are more numbers than input channels, it is an error.

−d Following this option is a comma separated list of numbers indicating channels to be deleted from the input file. All other channels will be passed through unchanged. The alpha channel may be specified as −1.

−p Following this option is a comma separated list of pairs of channel numbers. The first channel of each pair indicates a channel in the input file that will be mapped to the the channel in the output file indicated by the second number in the pair. No output channel number may appear more than once. Any input channel not mentioned will not appear in the output file. Any output channel not mentioned will not receive image data. The alpha channel may be specified as −1.

SEE ALSO

mergechan(1), *urt*(1), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

rletoabA60 − convert RLE images to Abekas yuv format

SYNOPSIS

rletoabA60 [**−c**] [**−{pP}** *x y*] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program converts an *RLE*(5) file to a yuv byte file suitable for display on an Abekas A60. Typically the yuv file is then rcp'd to the Abekas for display. By default *rletoabA60* will attempt to place the image according to the placement values in the image header. If the image is to large to fit in the Abekas format (720x486), the portion of the image extending off the edge will be cropped.

OPTIONS

−c Center the image on a black background.

−p *x y* Position the lower left corner of the image at (x y).

−P *x y* Increment the position of the image by (x y).

At most one of **−c**, **−p**, or **−P** can be specified.

infile The input will be read from this file, otherwise, input will be taken from stdin.

−o *outfile*
If specified, output will be written to this file, otherwise it will go to stdout.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Thomas Todd Elvins, University of Utah

NAME

rletoabA62 − Convert from RLE Format to Abekas A62 Dump Format

SYNOPSIS

rletoabA62 [**−N**] [**−f** *n*] [**−n** *n*] [*infile*]

DESCRIPTION

RletoabA62 converts a raster file in the Utah Raster Toolkit RLE format into a format suitable for writing to an Abekas A62 dump tape and subsequent loading onto the Abekas disk. The generated image is 768 pixels wide and 512 pixels high. If the input is larger, it is truncated. If it is smaller, it is padded on the top and right with black. The output is written to *stdout*, and should be written to a tape in 24K byte blocks with *dd* as in the following:

```
dd of=/dev/rmt8 obs=24k
```

Normally, the output is processed with a simple digital filter; this feature may be turned off with an option. *RletoabA62* normally writes two consecutive frames, normally starting at frame 1.

Input is taken from *stdin* unless a file name is given on the command line. Only a single file may be given, and so if multiple invocations of *rletoabA62* are performed in a script, care must be taken to tell the program to convert the data for the proper Abekas frame number (1-4). Otherwise, the colors will appear wrong; they will be rotated on a vector scope diagram.

EXAMPLE

The following example converts all files ending in *.rle* in the current directory and writes them to a tape. Two frames are written per image and the frame number is incremented accordingly.

```
frame=1
number=2
for file in *.rle
do
        rletoabA62 −f $frame $file
        frame=‘expr \( \( $frame − 1 \) + $number \) % 4 + 1‘
done |
dd of=/dev/rmt8 obs=24k
```

OPTIONS

Options are parsed by getopt(3).

−N Do not apply digital filtering.

−f *n* Create the first frame as Abekas frame number *n*, having a value from one to four. Consecutive frames increment this number modulo four. The default is one.

−n *n* Write *n* frames of output, incrementing the frame number each time. The default is two.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Bob Brown, RIACS.

BUGS

This program does not preserve the aspect ratio of the input.

NAME

rletoalias − Convert RLE image to Alias™ pix format.

SYNOPSIS

rletoalias [**−v**] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program converts an image in *RLE*(5) format to Alias™ "pix" format. Since "pix" and *RLE* differ on the origin location, the program flips the image top to bottom.

OPTIONS

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream. **−v** Verbose output.

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

aliastorle(1), *urt*(1), *RLE*(5).

AUTHOR

Raul Rivero, Mathematics Department, University of Oviedo.

NAME

rletoascii − Print an RLE image as ASCII chars.

SYNOPSIS

rletoascii [**−S** *asciistr*] [**−r**] [**−o** *outfile*] [*infile*]

DESCRIPTION

Rletoascii reads a file in *RLE*(5) format, converts it to black and white, then dumps it as ASCII characters. The 0 to 255 range of pixel values in the image is scaled to the length of *asciistr* and a the character at that position in the string is printed for each pixel. Input will be read from *infile* if specified, from standard input, otherwise. Output dumps to standard output, or *outfile*, if specified.

Usually, the input will need to be resized by *fant*(1) or *rlezoom*(1) to make it small enough to fit on the screen and to adjust the pixel aspect ratio to the "character aspect ratio" of the terminal. To get it "right side up", use *rleflip*(1) with the **−v** option. Finally, it may be helpful to maximize the dynamic range with *rlespiff* (1).

OPTIONS

−S *asciistr*

Specifies the range of ascii characters for conversion. The default string (**@BR*#$PX0woIcv:+!˜"., **) was designed to look good with the X 6x13 font.

−r Reverse video. This causes the 0 to 255 range to be mapped to the reverse of the ascii string.

SEE ALSO

fant(1), *rleflip*(1), *rlespiff* (1), *rlezoom*(1), *urt*(1), *RLE*(5).

AUTHOR

Rod G. Bogart, University of Michigan.

DEFICIENCIES

Could be rewritten to use overprinting for output to a real printer.

NAME

rletogif − Convert RLE files to GIF format.

SYNOPSIS

rletogif [**−o** *outfile.gif*] [*infile.rle*]

DESCRIPTION

This program converts an *RLE*(5) image file to *GIF* format. The input file must be a single channel (8 bit) image. Three channel (24 bit) images can be converted to single channel images using the programs *tobw*(1), *to8*(1), *mcut*(1), or *rlequant*(1). The input image will be flipped vertically, since the *GIF* origin is in the upper left, and the *RLE* origin is in the lower left. Only a single image will be converted.

OPTIONS

−o *outfile.gif*

If specified, the output will be written to this file. If *outfile.gif* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.rle

The input will be read from this file. If *infile.rle* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

to8(1), *mcut*(1), *rlequant*(1), *giftorle*(1), *urt*(1), *RLE*(5).

AUTHOR

Bailey Brown, University of Michigan

NAME

rletogray − Splits an RLE format file into gray scale images.

SYNOPSIS

rletogray [**−o** *prefix*] [*infile*]

DESCRIPTION

Rletogray reads a file in *RLE*(5) format and splits the file into unencoded binary files, one for each channel in the RLE file. The output file names will be constructed from the input file name or a specified prefix.

If an input *infile* is specified, then the output file names will be in the form "*rlefileroot*.{alpha, red, green, blue}", where *rlefileroot* is *infile* with any ".rle" suffix stripped off. If the option **−o** *prefix* is specified, then the output file names will be of the form "*prefix*.{alpha, red, green, blue}". If neither option is given, then the output file names will be "out.{alpha, red, green, blue}". Input will be read from *infile* if specified, from standard input, otherwise. If more channels than just red, green, blue, and alpha are present in the input, numeric suffixes will be used for the others.

OPTIONS

−o *prefix*
: Specifies the output file name prefix to be used.

infile
: This option is used to name the input file. If not present, input is taken from *stdin.*

SEE ALSO

rletoraw(1), *urt*(1), *RLE*(5).

AUTHOR

Michael J. Banks, University of Utah.

NAME

rletopaint − convert an RLE file to MacPaint format using dithering

SYNOPSIS

rletopaint [**−l**] [**−r**] [**−g** [*gamma*]] [**−o** *outfile.paint*] [*infile*]

DESCRIPTION

Rletopaint converts a file from *RLE*(5) format to MacPaint format. The program uses dithering to convert from a full 24 bit color image to a bitmapped image. If the RLE file is larger than a MacPaint image (576×720) it is cropped to fit.

Because MacPaint files have their coordinate origin in the upper left instead of the lower left, the RLE file should be piped through *rleflip*(1) −v before *rletopaint.*

The resulting file can be downloaded to a Macintosh in binary mode, and should be given a type of *PNTG* and a creator of *MPNT*, so it will be recognized as a MacPaint file.

OPTIONS

−l Use a linear map in the conversion from 24 bits to bitmapped output.

−g [*gamma*]
Use a gamma map of *gamma* (gamma is 2.0 if not specified).

−r Invert the sense of the output pixels (white on black instead of black on white). For normal images, you probably want this flag.

SEE ALSO

painttorle(1), urt(1), *RLE*(5).

AUTHOR

John W. Peterson. Byte compression routine by Jim Schimpf.

BUGS

Should use a color map in the file, if present.

NAME

rletoppm − convert a Utah RLE image file into a PBMPLUS/ppm image file.

SYNOPSIS

rletoppm [**−h**] [**−v**] [**−p**] [*infile*]

DESCRIPTION

This program converts Utah *RLE*(5) image files into PBMPLUS full-color (ppm) image files. Rletoppm will handle four types of RLE files: Grayscale (8 bit data, no color map), Pseudocolor (8 bit data with a color map), Truecolor (24 bit data with color map), and Directcolor (24 bit data, no color map). Since the orgins for the RLE and PBMPLUS image file formats are in different locations, this program automatically "flips" the image when converting.

OPTIONS

−v This option will cause rletoppm to operate in verbose mode. Header information is printed to "stderr".

−h This option allows the header of the RLE file to be dumped to "stderr" without converting the file. It is equivalent to using the −v option except that no file conversion takes place.

−p This option will output the ppm data in the "plain" format (P3), instead of the default "raw bits" format (P6). The plain format is more readable, but takes up more space.

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream. The resulting PBMPLUS/ppm data will be sent to "stdout".

EXAMPLES

rletoppm −v lenna.rle >lenna.ppm
: While running in verbose mode, convert lenna.rle to PBMPLUS/ppm format and store resulting data in lenna.ppm.

rletoppm −h test.rle
: Dump the header information of the RLE file called test.rle.

rletoppm −p test.rle >lenna.ppm
: Convert lenna.rle to PBMPLUS/ppm plain (P3) format and store in lenna.ppm.

SEE ALSO

ppmtorle(1), *pgmtorle*(1), *urt*(1), *RLE*(5)

AUTHOR

Wesley C. Barris
Army High Performance Computing Research Center (AHPCRC)
Minnesota Supercomputer Center, Inc.
Modifications by Eric Haines to support raw and plain formats.

NAME

rletops − Convert RLE images to PostScript

SYNOPSIS

rletops [**−C**] [**−a** *aspect*] [**−c** *center*] [**−h** *height*] [**−o** *outfile.ps*] [**−s**] [*infile*]

DESCRIPTION

Rletops converts *RLE*(5) images into *PostScript*. The conversion uses the *PostScript* **image** operator, instructing the device to reproduce the image to the best of its abilities. If *infile* isn't specified, the RLE image is read from stdin. The PostScript output is dumped to stdout, or to *outfile.ps*, if specified.

OPTIONS

−a *aspect*
: Specify aspect ratio of image. Default is 1.0 (note PostScript uses square pixels).

−C
: Causes a color PostScript image to be generated. This creates larger files and uses the PostScript **colorimage** operator, which is not recognized by all devices. The default is monochrome.

−c *center*
: Centers the images about a point *center* inches from the left edge of the page (or left margin if **−s** is specified). Default is 4.25 inches.

−h *height*
: Specifies the height (in inches) the image is to appear on the page. The default is three inches. The width of the image is calculated from the image height, aspect ratio, and pixel dimensions.

−s
: Specifies image is to be generated in "Scribe Mode." The image is generated without a PostScript *showpage* operator at the end, and the default image center is changed to 3.25 inches from the margin (which usually is 1 inch). This is to generate PostScript files that can be included in Scribe documents with the @Picture command. Images may also be included in LaTex documents with local conventions like the \special{psfile=image.ps} command.

NOTES

On devices like the Apple LaserWriter, *rletops* generates large PostScript files that take a non-trivial amount of time to download and print. A 512x512 image takes about ten minutes. For including images in documents at the default sizes, 256x256 is usually sufficient resolution.

SEE ALSO

avg4(1), *urt*(1), *RLE*(5).

AUTHORS

Rod Bogart, John W. Peterson, Gregg Townsend.

Portions are based on a program by Marc Majka.

BUGS

Due to a mis-understanding with the PostScript interpreter, *rletops* always rounds the image size up to an even number of scanlines.

NAME

rletoraw − Convert RLE file to raw RGB form.

SYNOPSIS

rletoraw [**−a**] [**−[Ns]**] [**−r**] [**−f** *header-size*] [**−t** *trailer-size*] [**−l** *left-scanline-pad*] [**−p** *scanline-pad* [**−o** *outfile*] [*infile*]

DESCRIPTION

This program converts an *RLE*(5) image to a raw RGB form. The output file is normally a stream of pixels (RGBRGB...), in left-to-right, bottom-to-top order (this can be changed with the **−N** or **−s** flags). The width and height of the input image will be printed on the standard error stream.

OPTIONS

−a If specified, an alpha channel will be written to the output file. This is the last output channel, unless **−r** is specified, in which case it will be the first.

−N If specified, the output will be written in a non-interleaved order. I.e., all the red pixels will be written first, then all the green pixels, etc.

−s If specified, the output will be written in a scanline-interleaved order. I.e., all the red pixels for a scanline will be written, followed by all the green pixels for the scanline, etc. The options **−N** and **−s** are mutually exclusive.

−r Reverse the order of the channels in the output. I.e., output will be written ABGR instead of RGBA.

−f *header-size*
A header of this many zero bytes will be written to the output file.

−t *trailer-size*
A trailer of this many zero bytes will be written after the output file.

−l *left-scanline-pad*
The left (beginning) of each scanline will be padded with this many zero bytes.

−p *left-scanline-pad*
The right (end) of each scanline will be padded with this many zero bytes.

−o *outfile*
If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

infile The input will be read from this file. If *infile* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

rawtorle(1), *urt*(1), *RLE*(5).

AUTHOR

Martin Friedmann

BUGS

Basically handles input files with 1 or 3 channels (plus alpha). Only the first channel of a 2 channel image will be written.

The header, trailer, and pad options are of dubious utility.

NAME

rletorla − convert a Utah RLE image file into a Wavefront "rla" or "rlb" image file.

SYNOPSIS

rletorla [**−b**] [**−h**] [**−v**] [**−o outfile**] [*infile*]

DESCRIPTION

This program converts Utah *RLE*(5) image files into Wavefront "rla" or "rlb" image files. Rletorla will handle four types of RLE files: Grayscale (8 bit data, no color map), Pseudocolor (8 bit data with a color map), Truecolor (24 bit data with color map), and Directcolor (24 bit data, no color map). In each case the resulting Wavefront image file will contain RGB data as well as a matte channel. If no alpha channel is found in the RLE file, the Wavefront matte channel will be computed using the RGB or mapped data. The entire area of the Wavefront image will be run length encoded. The size of the Wavefront "bounding box" data structure will be set to that of the total image area.

NOTE: Even though images of any size can be converted, Wavefront is very fussy about image dimensions. Normally, the converted image must be one of the following sizes or Wavefront will complain with "ERROR, cannot open image file filename, error -8":

646x485 (0-645x0-484) ntsc_4d
720x486 (0-719x0-485) qtl_ntsc
636x484 (0-635x0-483) iris_ntsc
1024x1024 (0-1023x0-1023) 1k_square

To get around this problem, the aspect ratio field in the Wavefront "rla" file will be "faked" with "ntsc_4d" for all formats that do not match one of those shown above. This way, Wavefront will find a valid format string, and any image size will be readable. "rlb" image file do not have this limitation.

OPTIONS

−b This option will cause rletorla to create a Wavefront "rlb" image file instead of using the default "rla" conversion.

−v This option will cause rletorla to operate in verbose mode. Header information is printed to "stderr".

−h This option allows the header of the RLE file to be dumped to "stderr" without converting the file. It is equivalent to using the −v option except that no file conversion takes place.

−o outfile
This option allows the name of the output file to be specified. Re-directing standard output as is done with most all other toolkit utilities is not permitted here because the resulting "rla" or "rlb" file is not written sequentially.

infile The name of the RLE image data file to be converted. The name of the resulting Wavefront file will be derived from the name of the input file (unless the −o option is used) -- the extension will be changed from "rle" to "rla" or "rlb". (Note: if you use the extended input file names described in *urt*(1), this will result in a very strange filename for the Wavefront file.)

EXAMPLES

rletorla −v lenna.rle
While running in verbose mode, convert lenna.rle to Wavefront rla format and store resulting data in lenna.rla.

rletorla −h test.0001.rle
Dump the header information of the RLE file called test.0001.rle.

rletorla −b -o junk.rlb test.rle
Convert test.rle into a Wavefront "rlb" file called junk.rlb.

SEE ALSO

rlatorle(1), *urt*(1), *RLE*(5).

AUTHOR

Wesley C. Barris
Army High Performance Computing Research Center (AHPCRC)
Minnesota Supercomputer Center, Inc.

NAME

rletotarga − Convert an RLE(5) image file to Truevision TARGA format.

SYNOPSIS

rletotarga [*infile*] *outfile*

DESCRIPTION

Rletotarga reads a file in RLE(5) format and converts it to Truevision's TARGA format. If no input file is specified, the data is read from stdin. The output TARGA file will be in one of three formats, depending on the contents of the RLE file: 8−bit B/W (format #3), 24− or 32−bit true color (format #2). Only the first image in the RLE file is read.

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Andrew C. Hadenfeldt, University of Nebraska−Lincoln

NAME

rletotiff − Convert 24 bit RLE image files to TIFF.

SYNOPSIS

rletotiff [**−{cC}**] **−o** *outfile.tif* [**−v**] [*infile.rle*]

DESCRIPTION

This program converts a 24 bit image in *RLE*(5) format into *TIFF* form. Only a single image will be converted.

OPTIONS

−{cC} Sets the type of compression used in the output file. **−c** (the default) will cause the output file to be compressed using the Lempel-Ziv-Welch (LZW) algorithm. **−C** will suppress any compression.

−o *outfile.tif*

The output will be written to this file. *outfile.tif* must be a real file, the special cases described in *urt (1)* do not apply. Note also that this "option" is not optional. The **−o** flag is required for consistency with the other tools.

−v Flip image vertically.

infile.rle

The input will be read from this file. If *infile.rle* is "−" or is not specified, the input will be read from the standard input stream.

SEE ALSO

tifftorle(1), *urt*(1), *RLE*(5).

AUTHOR

Bailey Brown, University of Michigan.

NAME

rlezoom − Magnify an RLE file by pixel replication.

SYNOPSIS

rlezoom *factor* [*y-factor*] [**−f**] [**−o** *outfile*] [*infile*]

DESCRIPTION

This program magnifies (zooms) an *RLE*(5) file by a floating point factor. Each pixel in the original image becomes a block of pixels in the output image. If no *y-factor* is specified, then the image will be magnified by *factor* equally in both directions. If *y-factor* is given, then each input pixel becomes a block of *factor* × *y-factor* pixels in the output. If *factor* or *y-factor* is less than 1.0, pixels will be dropped from the image. There is no pixel blending performed. Input is taken from *infile*, or from the standard input if not specified. The magnified image is written to the standard output, or *outfile*, if specified.

You should use *rlezoom* over *fant*(1) if you just want a quick magnification of an image with the pixel boundaries showing. It is significantly faster than *fant* because it does no arithmetic on the pixel values. If you need blending between pixels in the magnified image, then *fant* is the correct program to use. Use *rlezoom −f factor y-factor* to produce an image the same size as *fant −p 0 0 −s factor y-factor* for previewing purposes.

Note: due to the way that *scanargs*(3) parses the arguments from the command line, if the name of *infile* is a number, and it is in the current directory, you should prefix it with "./" so that it will not be confused with *factor* or *y-factor*.

SEE ALSO

fant(1), *urt*(1), *scanargs*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Gerald A. Winters.

NAME

smush − defocus an RLE image.

SYNOPSIS

smush [**−m** *maskfile*] [**−n**] [**−o** *outfile*] [*levels*] [*infile*]

DESCRIPTION

Smush convolves an image with a 5x5 Gaussian mask, blurring the image. One may also provide a mask in a text file. The file must contain an integer to specify the size of the square mask, followed by size*size floats. The mask will be normalized (forced to sum to 1.0) unless the **−n** flag is given.

The resulting image is the same size as the input image, no sub-sampling takes place. The levels option, which defaults to one, signifies the number of times which the image will be blurred. Each successive blurring is done with a more spread out mask, so a *smush* of level 2 is blurrier than piping two level one *smush* calls. If no input file is specified, *smush* reads from stdin. If no output file is specified with **−o** it writes the result to stdout.

SEE ALSO

avg4(1), *urt*(1), *RLE*(5).

AUTHOR

Rod G. Bogart

BUGS

Smush should probably automatically generate different sized gaussians and other common filters.

NAME

targatorle − Convert Truevision TARGA images to RLE format.

SYNOPSIS

targatorle [**−h** *headerfile*] [**−n** *nchannels*] [**−o** *outfile.rle*] [*infile.tga*]

DESCRIPTION

Targatorle converts a file from Truevision's TARGA format into RLE format. If no input file is specified, the data is read from stdin. *Targatorle* recognizes (but cannot necessarily process) all of the image subtypes defined by the 1989 TARGA 2.0 specification:

0 − Header Only, No Image Data
1 − Uncompressed, Color-mapped Image
2 − Uncompressed, True-color Image
3 − Uncompressed, B/W (gray scale) Image
9 − Run-length encoded, Color-mapped Image
10 − Run-length encoded, True-color Image
11 − Run-length encoded, B/W Image

Targatorle should correctly process images in formats 0, 2, 3, 10, and 11. No support is currently available for color mapped images.

OPTIONS

−h Allow the program to write TARGA header information to *headerfile*

−n where *nchannels* is 3 or 4. If input is a color image, copy only *nchannels* of the TARGA file; this allows the alpha channel to be stripped. By default, the alpha channel will be copied if present.

−o Use *outfile* as output instead of *stdout.*

LIMITATIONS

The TARGA image descriptor byte is ignored; therefore, the image origin is assumed to be that of RLE(5) (bottom left). None of the color-mapped TARGA formats (types 1 and 9) are supported. Finally, no attempt has been made to support extensions to the TARGA File Format introduced by Truevision in 1989 (new support for time stamps, comments, user−defined data fields, etc.).

SEE ALSO

urt(1), *RLE*(5).

AUTHOR

Hann-Bin Chuang
Andrew C. Hadenfeldt, Univ. of Nebraska−Lincoln (modifications)

NAME

tifftorle − Convert TIFF image files to RLE.

SYNOPSIS

tifftorle [**−o** *outfile.rle*] *infile.tif*

DESCRIPTION

This program converts a TIFF image to *RLE*(5) format.

OPTIONS

−o *outfile.rle*
: If specified, the output will be written to this file. If *outfile.rle* is "−", or if it is not specified, the output will be written to the standard output stream.

infile.tif
: The input will be read from this file. *infile.tif* must be a real file, the special cases described in *urt*(1) do not apply here.

LIMITATIONS

Can’t handle RGB TIFF files with a separate planar configuration.
Can’t handle tiled TIFF files.

SEE ALSO

tifftorle(1), *urt*(1), *libtiff*, *RLE*(5).

AUTHOR

Bailey Brown, University of Michigan.
Extended by David R. L. Worthington, SRI International to single channel TIFF files.
Extended by Spencer W. Thomas, University of Michigan to TIFF files with fewer than 8 bits/sample.
Requires libtiff, by Sam Leffler.

BUGS

Doesn’t copy alpha channel when present.

NAME

to8 − Convert a 24 bit RLE file to eight bits using dithering.

SYNOPSIS

to8 [**−g** *display_gamma*] [**−{iI}** *image_gamma*] [**−o** *outfile*] [*infile*]

DESCRIPTION

To8 Converts an image with 24 bit pixel values (eight bits each of red, green and blue) to eight bits of color using a dithered color map (the special color map is automatically written into the output file). If no input file is specified, *to8* reads from stdin. If no output file is specified with **−o** it writes the result to the standard output.

Other options allow control over the gamma, or contrast, of the image. The dithering process assumes that the incoming image has a gamma of 1.0 (i.e., a 200 in the input represents an intensity twice that of a 100.) If this is not the case, the input values must be adjusted before dithering via the **−i** or **−I** option. The input file may also specify the gamma of the image via a picture comment (see below). The output display is assumed to have a gamma of 2.5 (standard for color TV monitors). This may be modified via the **−g** option if a display with a different gamma is used.

To8 will put a picture comment into the output file indicating the display gamma assumed in constructing the dithering color map.

OPTIONS

−i *image_gamma*

Specify the gamma (contrast) of the image. A low contrast image, suited for direct display without compensation on a high contrast monitor (as most monitors are) will have a gamma of less than one. The default image gamma is 1.0. Image gamma may also be specified by a picture comment in the *RLE (5)* file of the form **image_gamma=***gamma*. The command line argument will override the value in the file if specified.

−I *image_gamma*

An alternate method of specifying the image gamma, the number following **−I** is the gamma of the display for which the image was originally computed (and is therefore 1.0 divided by the actual gamma of the image). Image display gamma may also be specified by a picture comment in the *RLE (5)* file of the form **display_gamma=***gamma*. The command line argument will override the value in the file if specified.

−g *display_gamma*

Specify the gamma of the *X* display monitor. The default value is 2.5, suitable for most color TV monitors (this is the gamma value assumed by the NTSC video standard).

−o *outfile*

If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

SEE ALSO

tobw(1), *getx11*(1), *mcut*(1), *rlequant*(1), *urt*(1), *dither*(3), *RLE*(5).

AUTHOR

Spencer Thomas

NAME

tobw − Convert a 24 bit RLE file to eight bits of gray scale value.

SYNOPSIS

tobw [**−t**] [**−o** *outfile*] [*infile*]

DESCRIPTION

Tobw converts an image with 24 bit pixel values (eight bits each of red, green and blue) to eight bits of grayscale information. The *NTSC Y* transform is used. If the **−t** flag is given, then the monochrome pixel values are replicated on all three output channels (otherwise, just one channel of eight bit data is produced). If no input file is specified, *tobw* reads from stdin. If no output file is specified with **−o,** it writes the result to stdout.

SEE ALSO

to8(1), *urt*(1), *rgb_to_bw*(3), *RLE*(5).

AUTHOR

Spencer Thomas

NAME

unexp − Convert "exponential" files into normal files.

SYNOPSIS

unexp [**−m** *maxval*] [**−o** *outfile*] [**−p**] [**−s**] [**−v**] *infile*

DESCRIPTION

Unexp Converts a file of "exponential" floating point values into an *RLE*(5) file containing integer valued bytes. Exponential files have N-1 channels of eight bit data, with the Nth channel containing a common exponent for the other channels. This allows the values represented by the pixels to have a wider dynamic range.

If no maximum value is specified, *unexp* first reads the RLE file to find the dynamic range of the whole file. It then rewinds the file and scales the output to fit within that dynamic range. If a maximum value is specified, *unexp* runs in one pass, and clamps any values exceeding the maximum.

Files containing exponential data are expected to have a "exponential_data" comment; *unexp* prints a warning if such a comment doesn't exist. An exponential file should be *unexp*'ed before attempting to use any tools that perform arithmetic on pixels (e.g., *rlecomp*(1), *avg4*(1), *fant*(1), or *applymap*(1)) or displaying the image.

Unexp does not allow piped input. The *infile* must be a real file; the special filenames described in *urt*(1) are not allowed. ("−" does work, as long as the input is coming from a real file; this is of minimal utility, therefore, as typing *unexp - <foo.rle* is harder than typing *unexp foo.rle*.)

OPTIONS

−m *maxval*
: Specify the maximum value (i.e., the data in the file is assumed to be in the range 0..maxval). Only the conversion pass is executed, and values found exceeding the maximum are clamped.

−o *outfile*
: If specified, the output will be written to this file. If *outfile* is "−", or if it is not specified, the output will be written to the standard output stream.

−p Print the maximum value found during the scanning phase

−s Just scan the file to find the maximum, don't generate any output.

−v Verbose mode, print a message to stderr after scanning or converting every hundred scanlines.

SEE ALSO

float_to_exp(3), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson

BUGS

Unexp is provided because of the lack of floating point or extended precision RLE files.

The **−v** flag is a historical relict from the slow CPU days.

NAME

unslice − Quickly assemble image slices

SYNOPSIS

unslice [**−f** *ctlfile*] [**−y** *ymax*] [**−o** *outfile*] *infiles ...*

DESCRIPTION

Unslice quickly assembles a number of horizontal image strips into a single output image. A typical use for *unslice* is to put together portions of an image ("slices") computed independently into a single output picture. Because *unslice* uses the "raw" RLE library calls to read and write the images, it runs much faster than doing the equivalent operations with crop and comp.

unslice has two modes of operation. If given the **−f** flag, *unslice* reads a control file telling it how to assemble the images. This is a text file with two decimal numbers on each line, one line for each slice to be assembled into the output image. Each line gives the starting and stopping scanlines (inclusive) for each slice. These must be in ascending order. This is useful if the slices have excess image area that should be cropped away.

If no control file is given, the **−y** flag is used. This tells *unslice* what the maximum Y value of the output image is. *Unslice* reads the files in order, using the RLE headers to determine where to place the slices. If two slices overlap, the first scanlines from the second slice are thrown away. In both cases, the slices must be in ascending order, and are expected to be of uniform width.

SEE ALSO

crop(1), *rlecomp*(1), *rlepatch*(1), *repos*(1), *urt*(1), *RLE*(5).

AUTHOR

John W. Peterson

BUGS

Unslice has really been superceded by *rlepatch*(1).

NAME

urt − overview of the Utah Raster Toolkit

SYNOPSIS

applymap	Apply color map to image data.
avg4	Simple 2x2 downsizing filter.
crop	Crop image.
cubitorle	Convert Cubicomp format to RLE.
dvirle	Typeset TeX ".dvi" files as RLE images.
fant	Image scale/rotate with anti-aliasing.
get4d	Display on SGI Iris/4D display.
get_orion	Display on "Orion" display.
getap	Display on Apollo.
getbob	Display under HP window system.
getcx3d	Display RLE on Chromatics CX3D.
getfb	Display using BRL generic fb library.
getgmr	Display on Grinnell GMR-27 frame buffer.
getiris	Display on SGI 2400/3000 w/o window manager.
getmac	Display on Mac under MPW.
getmex	Display on SGI under the window manager.
getqcr	Display on Matrix QCR camera.
getren	Display on HP SRX.
getsun	Display using SunTools.
getx10	Display on X10 display.
getx11	Display using X11.
giftorle	Convert GIF files to RLE.
graytorle	Convert separate rrr ggg bbb files to RLE.
mcut	Median cut color quantization.
mergechan	Merge colors from multiple images.
painttorle	Convert MacPaint to RLE.
pgmtorle	Convert PBMPLUS pgm format to RLE.
ppmtorle	Convert PBMPLUS ppm format to RLE.
pyrmask	Generate "pyramid" filter mask.
rastorle	Convert Sun Raster to RLE.
rawtorle	Convert various raw formats to RLE.
read98721	Read the screen of an HP 98721 "Renaissance" to an RLE file.
repos	Reposition an image.
rlatorle	Convert Wavefront RLA format to RLE.
rleClock	Draws a clock face.
rleaddcom	Add comments to an RLE file.
rleaddeof	Add an EOF code to an RLE file.
rlebg	Generate a "background".
rlebox	Find bounding box of an image.
rlecomp	Image composition.
rledither	Floyd-Steinberg dither an image to a given colormap.
rleflip	Flip an image or rotate it 90.
rlehdr	Print info about an RLE file.
rlehisto	Make a histogram of an image.
rleldmap	Load a new colormap into a file.
rlemandl	Make a Mandelbrot image.
rlenoise	Add noise to an image.
rlepatch	Patch smaller images on a big one.
rleprint	Print all pixel values in image.
rlequant	Variance based color quantization.
rlescale	Generate a "gray scale".
rleselect	Select images from an RLE file.
rlesetbg	Set the background color of an image file.
rleskel	Skeleton tool. Programming example.

rlespiff	Simple contrast enhancement.
rlesplice	Splice two images horizontally or vertically.
rlesplit	Split concatenated images into files.
rlestereo	Combine two images into a "red-green" stereo pair.
rleswap	Swap or select color channels.
rletoabA60	Convert RLE to Abekas A60 format.
rletoabA62	Convert to Abekas A62 format.
rletoascii	Make a line-printer/CRT version of an RLE image.
rletogif	Convert RLE images to GIF format.
rletogray	Convert RLE to separate rrr ggg bbb files.
rletopaint	Convert RLE to MacPaint.
rletoppm	Convert RLE to PBMPLUS ppm format.
rletops	Convert RLE to (B&W) PostScript.
rletorast	Convert RLE to Sun Raster.
rletoraw	Convert RLE to rgbrgb raw format.
rletorla	Convert RLE to Wavefront RLA format.
rletotiff	Convert RLE to TIFF 24 bit format.
rlezoom	Scale image by sub- or super-sampling.
smush	Generic filtering.
targatorle	Convert TARGA to RLE.
tifftorle	Convert TIFF 24 bit images to RLE.
to8	24 to 8 bit ordered dither color conversion.
tobw	Color→B&W conversion.
unexp	Convert "exp" format to normal colors.
unslice	Paste together "slices" into a full image.
wasatchrle	Convert Wasatch paint system to RLE.

DESCRIPTION

The *Utah Raster Toolkit* is a collection of programs and C routines for dealing with raster images commonly encountered in computer graphics. A device and system independent image format stores images and information about them. Called the *RLE*(5) format, it uses run length encoding to reduce storage space for most images.

The programs (tools) currently included in the toolkit are listed above, together with a short description of each one. Most of the tools read one or more input RLE files and produce an output RLE file. Some generate RLE files from other information, and some read RLE files and produce output of a different form.

An input file is almost always specified by mentioning its name on the command line. Some commands, usually those which take an indefinite number of non-file arguments (e.g., *rleaddcom*) require a **−i** flag to introduce the input file name. If the input file name is absent the tool will usually read from the standard input. An input file name of "−" also signals that the input should be taken from the standard input.

On Unix systems, there are two other specially treated file name forms. A file name starting with the character '|' will be passed to *sh*(1) to run as a command. The output from the command will be read by the tool. A file whose name ends in ".Z" (and which does not begin with a '|') will be decompressed by the *compress*(1) program. Both of these options supply input to the tool through a pipe. Consequently, certain programs (those that must read their input twice) cannot take advantage of these features. This is noted in the manual pages for the affected commands.

An output file is almost always specified using the option **−o** *outfile*. If the option is missing, or if *outfile* is "−", then the output will be written to the standard output.

On Unix systems, the special file name forms above may also be used for output files. File names starting with '|' are taken as a command to which the tool output will be sent. If the file name ends in ".Z", then *compress* will be used to produce a compressed output file.

Several images may be concatenated together into a single file, and most of the tools will properly process all the images. Those that will not are noted in their respective man pages.

Picture comments. Images stored in *RLE* form may have attached comments. There are some comments that are interpreted, created or manipulated by certain of the tools. In the list below, a word enclosed in <> is a place-holder for a value. The <> do not appear in the actual comment.

image_gamma=<float number>

Images are sometimes computed with a particular ‘‘gamma’’ value -- that is, the pixel values in the image are related to the actual intensity by a power law, *pixel_value=intensity^image_gamma*. Some of the display programs, and the *buildmap*(3) function will look for this comment and automatically build a "compensation table" to transform the pixel values back to true intensity values.

display_gamma=<float number>

The *display_gamma* is just *1/image_gamma*. That is, it is the ‘‘gamma’’ of the display for which the image was computed. If an *image_gamma* comment is not present, but a *display_gamma* is, the displayed image will be gamma corrected as above. The *to8* program produces a *display_gamma* comment.

colormap_length=<integer>

The length of the colormap stored in the *RLE* header must be a power of two. However, the number of useful entries in the colormap may be smaller than this. This comment can be used to tell some of the display programs (*getx11*, in particular) how many of the colormap entries are used. The assumption is that entries 0 − *colormap_length−1* are used. This comment is produced by *mcut*, *rlequant*, and *rledither*.

image_title=<string>

This comment is used by *getx11* to set the window title. If present, the comment is used instead of the file name. (No other programs currently pay attention to this comment.) The comments *IMAGE_TITLE*, *title*, and *TITLE* are also recognized, in that order. No programs produce this comment.

HISTORY=<string>

All toolkit programs (with the exception of rleaddcom) create or add to a *HISTORY* comment. Each tool appends a line to this comment that contains its command line arguments and the time it was run. Thus, the image contains a history of all the things that were done to it. No programs interpret this comment.

exponential_data

This comment should be present in a file stored in ‘‘exponential’’ form. See *unexp*(1) and *float_to_exp*(3) for more information. The *unexp* program expects to see this comment.

SEE ALSO

compress(1), *sh*(1), *RLE*(5).

AUTHOR

Many people contributed to the Utah Raster Toolkit. This manual page was written by Spencer W. Thomas, University of Michigan.

NAME

wasatchrle − Convert Wasatch Systems image files to RLE format

SYNOPSIS

wasatchrle [**−o** *outfile*] *basename*

DESCRIPTION

Wasatchrle converts image files generated by the Wasatch Systems Paint program to RLE format. It expects to find two files, "*basename*.lut" (the color look-up table) and "*basename*.rlc" (the run-length encoded data).

Wasatchrle generates as output a single channel RLE image with a full color map. Since the Wasatch Paint program's origin is the top left of the image, the results should be passed through *rleflip −v* to correctly orient the image. If the image is to be used with other toolkit operations (e.g., compositing), it should first be run through *applymap*(1) to convert the image to a full color (three channel) RLE file.

SEE ALSO

rleflip(1), applymap(1), urt(1), *RLE*(5),
Wasatch Systems, "Wasatch Raster Image File Definition for Wasatch Illustration Software (Version 1.2 and Later)"

AUTHOR

John W. Peterson

NAME

buildmap − create a color map array from an RLE file header.

SYNOPSIS

#include <rle.h>

rle_pixel ** buildmap(the_hdr, minmap, orig_gamma, new_gamma)
rle_hdr * the_hdr;
int minmap;
double orig_gamma, new_gamma;

DESCRIPTION

The color map in the *rle_hdr*(3) structure is not in the most easily used form. The function *buildmap* returns a pointer to a colormap array with certain minimum dimensions, making it a little easier to implement color mapping in a program. The color map from first argument, *the_hdr*, is used to build the result. If no map is present in *the_hdr*, then an identity map of the minimum size will be returned.

The returned color map will have at least *minmap* rows or channels, each of which is at least 256 entries long (so that indexing into the color map with an 8 bit *rle_pixel* value will always succeed.)

The color map from *the_hdr* will be composed with a gamma compensation curve to account for the gamma of the display for which the input color map was presumably computed. The argument *orig_gamma* specifies the gamma of the compensation curve. It would typically be the gamma of the original display.

If *gamma* is 0, then if a picture comment *image_gamma=i_gamma* is present, *gamma* will be set to *1.0/i_gamma*. Otherwise, if a comment *display_gamma=d_gamma* is present, *gamma* will be set to *d_gamma*. The gamma compensation value for pixel *i* is *255*(i/255)^gamma*.

If this color map will be used directly for another display, the gamma of this new display should be passed in *new_gamma*.

The returned value is a pointer to an array of pointers to arrays of *rle_pixel* values. It may be doubly indexed in C code, so that if *cmap* is the return value, the RGB color mapping for a pixel *pixval* is *(cmap[0][pixval], cmap[1][pixval], cmap[2][pixval])*.

NOTES

Generally, unless the user explicitly specifies the image or original display gamma (e.g., as with the **−i** or **−I** flags of *getx11*(1), you should pass 0 for *orig_gamma*. This lets *buildmap* use the value from *the_hdr*, if it is present.

If you are going to use the result of buildmap to generate values to be dithered, *new_gamma* should always be 1.0, and the display gamma (**-g** in *getx11*) should be passed to *dithermap*(3). If you are not planning to dither, then pass the user supplied display gamma as *new_gamma*.

The color map storage allocated by *buildmap* can be released by calling *free(map[0])*.

SEE ALSO

dithermap(3), *rle_hdr*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, University of Utah

NAME

colorquant − variance-based color quantization

SYNOPSIS

#include <colorquant.h>
int colorquant(red, green, blue, npix, colormap, colors, bits, rgbmap, flags, accum_hist)
unsigned char *red, *green, *blue;
unsigned long npix;
unsigned char *colormap[3];
int colors, bits;
unsigned char *rgbmap;
int flags;
int accum_hist;

DESCRIPTION

Colorquant performs variance-based color quantization on a given image. A representative colormap and a table for performing RGB to colormap index mapping are computed. The number of colors to which the image was quantized (the total number of colormap entries computed) is returned. The arguments to *colorquant* are:

red, green, blue
The red, green and blue channels of the image. The ith pixel is represented as the RGB triple (*red*[i], *green*[i], *blue*[i]). These arrays usually contain values that have been 'prequantized' (see below).

npix The length, in bytes, of the *red*, *green* and *blue* arrays. Equal to the total number of pixels in the image.

colormap
Points to a pre-allocated, three-channel colormap. These arrays will be filled with the colormap values computed by the variance-based color quantization algorithm. *colormap*[0][i], *colormap*[1][i], and *colormap*[2][i] are, respectively, the red, green and blue components of the ith colormap entry.

colors The number of pre-allocated colormap entries. The image will be quantized to at most this many colors.

bits The number of significant bits in each entry of the *red*, *green* and *blue* arrays. Normally, the red, green and blue arrays contain values that have been prequantized to fewer than eight significant bits (see *flags* below). Five significant bits usually represents a good tradeoff between image quality and running time. Anything above six significant bits will likely lead to excessive paging, as the size of *rgbmap* and the internal histogram are proportional to (2^*bits*)^3.

rgbmap A pointer to an array of unsigned chars of size (2^*bits*)^3. This array is used to map from pixels to colormap entries. The prequantized red, green and blue components of a pixel are used as an index into this array to retrieve the colormap index that should be used to represent the pixel. The array is indexed as:

colorindex = *rgbmap*[(((r << *bits*) | g) << *bits*) | b];

where r, g, and b are the prequantized red, green and blue components of the pixel in question.

flags A collection of bit-flags that modify the operation of *colorquant*. Currently defined values are **CQ_FAST**, **CQ_QUANTIZE**, and **CQ_NO_RGBMAP**.
If **CQ_FAST** is set, the construction of *rgbmap* will be relatively fast. If not, *rgbmap* will be built slowly but more accurately. In most cases, the error introduced by the 'fast' approximation is barely noticeable.
If **CQ_QUANTIZE** is set, the values in *red*, *green*, and *blue* are taken as 8-bit values and will be quantized to *bits* significant bits by *colorquant*. If not set, these values are assumed to be prequantized.
If **CQ_NO_RGBMAP** is set, *rgbmap* will not be built.

accum_hist
This argument provides a facility to accumulate multiple images into a single colormap. If *accum_hist* is zero, the routine works normally. To build a colormap for several images,

accum_hist should have the value 1 for the first image, and 2 for subsequent images. Finally, after all the images have been processed, a value of 3 for *accum_hist* will compute the *colormap* and *rgbmap*. The values of *colors* and *bits* should not change during this process. The arguments *colormap*, *rgbmap*, and *fast* are ignored if *accum_hist* is 1 or 2, and *red*, *green*, *blue*, and *npix* are ignored if *accum_hist* is 3.

AUTHOR

Craig Kolb, Yale University.
Martin Friedmann, MIT Media Lab did the *accum_hist* changes.

REFERENCE

Wan, Wong, and Prusinkiewicz, *An Algorithm for Multidimensional Data Clustering*, Transactions on Mathematical Software, Vol. 14 #2 (June, 1988), pp. 153-162.

SEE ALSO

rlequant(1), *inv_cmap*(3).

NAME

dithermap, bwdithermap, make_square, dithergb, ditherbw − functions for dithering color or black and white images.

SYNOPSIS

dithermap(levels, gamma, rgbmap, divN, modN, magic)
int levels;
double gamma;
int rgbmap[][3], divN[256], modN[256], magic[16][16];

bwdithermap(levels, gamma, bwmap, divN, modN, magic)
int levels;
double gamma;
int bwmap[], int divN[256], modN[256], magic[16][16];

make_square(N, divN, modN, magic)
double N;
int divN[256], modN[256], magic[16][16];

dithergb(x, y, r, g, b, levels, divN, modN, magic)
int x, y, r, g, b, levels;
int divN[256], modN[256], magic[16][16];

ditherbw(x, y, val, divN, modN, magic)
int x, y, val, divN[256], modN[256], magic[16][16];

DESCRIPTION

These functions provide a common set of routines for dithering a full color or gray scale image into a lower resolution color map.

Dithermap computes a color map and some auxiliary parameters for dithering a full color (24 bit) image to fewer bits. The argument *levels* tells how many different intensity levels per primary color should be computed. To get maximum use of a 256 entry color map, use *levels*=6. The computed map uses *levelsˆ3* entries. The *gamma* argument provides for gamma compensation of the generated color map (that is, the values in the map will be adjusted to give a linear intensity variation on a display with the given gamma). The computed color map will be returned in the array *rgbmap*. *divN* and *modN* are auxiliary arrays for computing the dithering pattern (see below), and *magic* is the magic square dither pattern.

To compute a color map for dithering a black and white image to fewer intensity levels, use *bwdithermap*. The arguments are as for *dithermap*, but only a single channel color map is computed. The value of *levels* can be larger than for *dithermap*, as the computed map only has *levels* entries.

To just build the magic square and other parameters, use *make_square*. The argument *N* should be equal to 255.0 divided by the desired number of intensity levels less one (i.e., *N = 255.0 / (levels - 1)*). The other arguments are filled in as above.

The color map index for a dithered full color pixel is computed by *dithergb*. Since the pattern depends on the screen location, the first two arguments *x* and *y*, specify that location. The true color of the pixel at that location is given by the triple *r*, *g*, and *b*. The number of intensity *levels* and the dithering parameter matrices computed by *dithermap* are also passed to *dithergb*.

The color map index for a dithered gray scale pixel is computed by *ditherbw*. Again, the screen position is specified, and the intensity value of the pixel is supplied in *val*. The dithering parameters must also be supplied.

Alternatively, the dithering may be done in line instead of incurring the extra overhead of a function call, which can be significant when repeated a million times. The computation is as follows:

```
row = y % 16;
col = x % 16;
#define DMAP(v,col,row) (divN[v] + (modN[v]>magic[col][row] ? 1 : 0))
```

```
pix = DMAP(r,col,row) + DMAP(g,col,row)*levels +
        DMAP(b,col,row)*levels*levels;
```

For a gray scale image, it is a little simpler:

```
pix = DMAP(val,row,col);
```

And on a single bit display (assuming a 1 means white):

```
pix = divN[val] > magic[col][row] ? 1 : 0
```

SEE ALSO

rgb_to_bw(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

float_to_exp − Convert floating point values into "exponential" pixels.

SYNOPSIS

#include <rle.h>

float_to_exp(count, floats, pixels)
int count;
float * floats;
rle_pixel * pixels;

DESCRIPTION

The function *float_to_exp* converts *count* floating point numbers (pointed to by *floats*) into *count+1* bytes (pointed to by *pixels)* using an "exponential" format. This format generates *count* pixels as eight bit "mantissa" values, and another byte containing a common exponent for all of the data values. This format has a wider dynamic range of values with little extra overhead. The inverse mapping is

```
float expnt, flt_val;
rle_pixel exponent, val;
expnt = ldexp( 1/256.0, (int)exponent - 127 );
flt_val = (float)val * expnt;
```

Files containing exponential data may be converted into displayable images using the *unexp*(1) command. *Unexp* should be used before using any tools that perform arithmetic on pixel values, or displaying the image. *Unexp* expects files containing exponential data to have an "exponential_data" picture comment.

SEE ALSO

unexp(1), *rle_putcom*(3), *librle*(3), *RLE*(5).

AUTHOR

John W. Peterson, based on code by Spencer Thomas.
University of Utah

NAME

hilbert_i2c, hilbert_c2i − Compute points on a Hilbert curve.

SYNOPSIS

void hilbert_i2c(dim, bits, idx, coords)
int dim, bits;
long int idx;
int coords[];

void hilbert_c2i(dim, bits, coords, idx)
int dim, bits;
int coords[];
long int *idx;

DESCRIPTION

These procedures map the real line onto a Hilbert curve and vice versa. (A Hilbert curve is a space filling curve similar to the Peano curve, except it is not closed.) The procedure *hilbert_i2c* returns the coordinates of a point on the Hilbert curve, given an index value representing its sequential position on the curve. The procedure *hilbert_c2i* reverses the process. The arguments are:

dim The dimensionality of the Hilbert curve. For the usual planar curve case, this would be 2.

bits The resolution to which the Hilbert curve will be computed. The space is quantized to 2^*bits* values on each axis, so there are 2^(3**bits*) points on the curve. The product of *dim***bits* should be less than or equal to the number of bits in a long integer (typically 32), and *bits* should be less than or equal to the number of bits in an integer.

idx The sequential position of the point along the curve (starting from 0). This is a 3**bits* bit integer.

coords The spatial coordinates of the point on the curve. The array should hold *dim* values. Each is a *bits* bit integer.

REFERENCE

A. R. Butz, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Trans. Comput.*, vol C-20, pp. 424-426, Apr. 1971.

AUTHOR

Spencer W. Thomas

NAME

inv_cmap − efficiently compute an inverse colormap

SYNOPSIS

void inv_cmap(colors, colormap, bits, dist_buf, rgbmap)

int colors, bits;
unsigned char *colormap[3], *rgbmap;
unsigned long *dist_buf;

DESCRIPTION

Inv_cmap computes an inverse colormap to translate an RGB color to the nearest color in the given *colormap*. The arguments are

colors The number of colors in the input colormap. Must be 256.

colormap
The input colormap. The *i*th color is (*Colormap[0][i]*, *Colormap[1][i]*, *Colormap[2][i]*).

bits Controls the size and precision of the inverse colormap. The resulting colormap will be a cube *2^bits* on a side, and will therefore contain *2^(3*bits)* entries. RGB colors must be quantized to *bits* bits before using the inverse colormap.

dist_buf
Temporary storage used by *inv_cmap*. It should contain at least *2^(3*bits)* elements.

rgbmap The inverse colormap. Should be allocated with at least *2^(3*bits)* elements. After calling *inv_cmap*, an RGB color (r,g,b) can be mapped to its closest representative in *colormap* by evaluating
#define quantize(p) ((p)>>(8-bits))
rgbmap[(((quantize(r) << bits) | quantize(g)) << bits) | quantize(b)]

Predicted performance is *O(2^(3*bits)*log(colors))*. The measured performance is sublinear (but not as good as *log*) in the number of input colors and also in the size of the output inverse colormap. (I.e., it goes up more slowly than *2^(3*bits)*.)

SEE ALSO

colorquant(3).

AUTHOR

Spencer W. Thomas

NAME

librle − Functions to create and read Run Length Encoded image files.

SYNOPSIS

#include <rle.h>

cc ... -lrle

DESCRIPTION

The *RLE*(5) image file format provides a method for saving and restoring images in a device independent form. A number of subroutines are available to facilitate writing and reading *RLE*(5) files. They are described separately in their own manual pages (listed below).

SEE ALSO

buildmap(3), *bwdithermap*(3), *colorquant*(3), *ditherbw*(3), *dithergb*(3), *dithermap*(3), *float_to_exp*(3), *make_square*(3), *rgb_to_bw*(3), *rle_addhist*(3), *rle_cp*(3), *rle_debug*(3), *rle_delcom*(3), *rle_freeraw*(3), *rle_get_error*(3), *rle_get_setup*(3), *rle_get_setup_ok*(3), *rle_getcom*(3), *rle_getraw*(3), *rle_getrow*(3), *rle_getskip*(3), *rle_open_f*(3), *rle_open_f_noexit*(3), *rle_put_init*(3), *rle_put_setup*(3), *rle_putcom*(3), *rle_puteof*(3), *rle_putraw*(3), *rle_raw_alloc*(3), *rle_raw_free*(3), *rle_rawtorow*(3), *rle_row_alloc*(3), *rle_row_free*(3), *rle_skiprow*(3), *scanargs*(3), *rle_hdr*(3), *rle_op*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua, and others.

NAME

rgb_to_bw − convert a color scanline to black and white.

SYNOPSIS

#include <rle.h>

void rgb_to_bw(red, green, blue, bw, length);
rle_pixel * red, * green, * blue, *bw;
int length;

DESCRIPTION

rgb_to_bw converts red/green/blue color information to black and white using the *NTSC Y* transform: *Y = 0.30 * R + 0.59 * G + 0.11 * B* . The arguments point to scanlines with *length* bytes in each. *bw* may be identical to one of *red*, *green*, or *blue*.

SEE ALSO

tobw(1), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_addhist − add a history comment to an RLE file.

SYNOPSIS

#include <rle.h>

void rle_addhist(argv, in_hdr, out_hdr)
char **argv;
rle_hdr *in_hdr, *out_hdr;

DESCRIPTION

rle_addhist is used to add history comments to the *RLE*(5) file in the form:

HISTORY=cmd arg1 arg2 on Tue Sep 13 01:06:49 WST 1988

where cmd, arg1, etc. are the command line arguments which have been used to generate or filter this *RLE* file. The HISTORY comment is always appended to so that an accumulated history is kept along with a timestamp. Programs which generate *RLE* files should call *rle_addhist* as follows:

```
rle_addhist(argv,(rle_hdr *)0,&out_hdr);
```

Programs which filter *RLE* files should call *rle_addhist* as:

```
rle_addhist(argv,&in_hdr,&out_hdr);
```

SEE ALSO

rle_hdr(3), *rle_putcom*(3), *rle_getcom*(3), *librle*(3), *RLE*(5).

AUTHOR

Andrew Marriott,
Curtin University of Technology (Australia)

NAME

rle_cp − Copy the rest of an image to the output.

SYNOPSIS

#include <rle.h>

rle_cp(in_hdr, out_hdr)
rle_hdr *in_hdr, *out_hdr;

DESCRIPTION

This routine copies the image contents of one *RLE*(5) file to another. The image described by *in_hdr* will be copied to the image file described by *out_hdr*. If any rows have been read with *rle_getrow*(3) or *rle_getraw*(3), those rows must have also been written with *rle_putrow*(3) or *rle_putraw*(3), respectively, in order for the input and output files to be "in sync". In any case, the header should have been written to the output file with *rle_put_setup*(3). When *rle_cp* returns, the input image file will be positioned at the end of the image, and an end of image code will have been written to the output image file.

SEE ALSO

rle_hdr(3), *rle_getrow*(3), *rle_getraw*(3), *rle_putrow*(3), *rle_putraw*(3), *rle_put_setup*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Michigan

NAME

rle_get_setup − Read the header from an RLE file.
rle_get_setup_ok − Print error message and exit if rle_get_setup fails.
rle_get_error − Print error message for rle_get_setup failure.
rle_debug − Turn on or off debugging messages.

SYNOPSIS

#include <rle.h>

rle_get_setup(the_hdr);
rle_hdr * the_hdr;

rle_get_setup_ok(the_hdr, prog_name, file_name);
rle_hdr * the_hdr;
char * prog_name, * file_name;

rle_get_error(code, prog_name, file_name)
int code;
char *prog_name, *file_name;

rle_debug(on_off)
int on_off;

DESCRIPTION

Rle_get_setup is called to initialize the process of reading an *RLE*(5) file. It will fill in *the_hdr* with the header information from the *RLE* file, and will initialize state for *rle_getrow*(3) and *rle_getraw*(3). The *rle_file* field of *the_hdr* should be initialized to the input stream before calling *rle_get_setup*. The *bits* field is initialized by *rle_get_setup* to enable reading of all the channels present in the input file. To prevent *rle_getrow* or *rle_getraw* from reading certain channels (e.g., the alpha channel), the appropriate bits should be cleared before calling them. The error codes returned by *rle_get_setup* are defined in *rle.h*.

Rle_get_setup_ok invokes *rle_get_setup* and checks the return code. If an error occurred, it calls *rle_get_error(err_code, prog_name, file_name)* to print the appropriate error message on *stderr*, and the program exits with the status code set.

Rle_get_error can be called to print an appropriate error message on the standard error output for the failure code returned by *rle_get_setup*. The *prog_name* and *file_name* parameters are used for the error message. If *file_name* is NULL or "−", the string "Standard input" is substituted.

The function *rle_debug* is used to enable or disable debug printing for the *rle_get* functions. If *on_off* is non-zero, all input read from any *RLE* file will be printed in a readable form on the standard error output. Calling *rle_debug(0)* will turn off this activity.

SEE ALSO

rle_hdr(3), *rle_getrow*(3), *rle_getraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua
University of Utah

NAME

rle_getraw − Read run length encoded data from an RLE file.
rle_freeraw − Free pixel storage allocated by rle_getraw.

SYNOPSIS

```
#include <rle.h>
#include <rle_raw.h>

unsigned int rle_getraw( the_hdr, scanraw, nraw )
rle_hdr * the_hdr;
rle_op ** scanraw;
int * nraw;

void rle_freeraw( the_hdr, scanraw, nraw );
rle_hdr * the_hdr;
rle_op ** scanraw;
int * nraw;
```

DESCRIPTION

Rle_getraw can be used to read information from an RLE file in the "raw" form.

The *scanraw* argument is an array of pointers to arrays of *rle_op*(3) structures. Each *rle_op* structure specifies a run or sequence of pixel values. The array *nraw* gives the number of *rle_op* structures for each channel. I.e., *nraw[i]* is the length of the array pointed to by *scanraw[i]*.

Return value is the current scanline number. Returns 32768 at EOF.

Sufficient space must be allocated in the arrays of *rle_op* structures to hold the data read from the file. A function, *rle_raw_alloc*(3), is provided to make this easier. The storage required by any pixel sequences in the input will be dynamically allocated by *rle_getraw*.

The pixel storage allocated dynamically by *rle_getraw*(3) must be freed to avoid memory leaks. This is most easily accomplished by calling *rle_freeraw*. The argument *scanraw* points to an array of *rle_op* structures, with *nraw* indicating the number of structures in each channel. All pixel data arrays will be freed by the call to *rle_freeraw*.

EXAMPLE

The usual code looks something like

```
rle_hdr in_hdr, out_hdr;
rle_op **raw;
int *nraw;
while ( rle_getraw( &in_hdr, raw, nraw ) != 32768 )
{
        /* Process data. */
        rle_putraw( &out_hdr, raw, nraw );
        rle_freeraw( &in_hdr, raw, nraw );
}
```

SEE ALSO

rle_hdr(3), *rle_op*(3), *rle_putraw*(3), *rle_raw_alloc*(3), *rle_raw_free*(3), *rle_getrow*(3), *rle_getskip*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_getrow − Read a scanline of pixels from an RLE file.

SYNOPSIS

#include <rle.h>

rle_getrow(the_hdr, rows);
rle_hdr * the_hdr;
rle_pixel ** rows;

DESCRIPTION

Rle_getrow reads information for a single scanline from the input file each time it is called. *The_hdr* should point to the structure initialized by *rle_get_setup*(3). The array *rows* should contain pointers to arrays of characters, into which the scanline data will be written. There should be as many elements in *rows* as there are primary colors in the input file (typically 1 or 3), and the scanline arrays must be indexable up to the maximum X coordinate, as specified by *the_hdr→xmax*. *rle_getrow* returns the y value of the scanline just read. This will always be 1 greater than the y value from the scanline previously read, and starts at *the_hdr→ymin*. Only those channels enabled by *the_hdr→bits* will be returned.

NOTES

If an alpha channel is present in the input and enabled (by *RLE_SET_BIT*, see *rle_hdr*(3)), then *rows* should include a −1 entry. (I.e., *rows[−1]* should point to a valid scanline array.) The easiest way to ensure this is to use *rle_row_alloc*(3) to allocate *rows*.

Rle_getrow will continue to return scanlines even after the end of the input file has been reached, incrementing the return scanline number each time it is called. The calling program should use some other termination criterion (such as the scanline number reaching *the_hdr→ymax*, or explicitly testing testing for end of file on the input with *feof(infile)*. The second test may fail if *rle_getrow* has encountered a logical EOF in the file. The first will always work eventually.)

EXAMPLE

The code below reads the first two 3 color scanlines of 512 pixels from an RLE file on the standard input.

```
char scanline[2][3][512], *rows[3];
int row, i;
rle_dflt_hdr.rle_file = stdin;
rle_get_setup( &rle_dflt_hdr );
for ( row = 0; row < 2; row++ )
{
        for ( i = 0; i < 3; i++ )
                rows[i] = scanline[row][i];
        rle_getrow( &rle_dflt_hdr, rows );
}
```

SEE ALSO

rle_hdr(3), *rle_row_alloc*(3), *rle_row_free*(3), *rle_get_setup*(3), *rle_getraw*(3), *rle_getskip*(3), *rle_putrow*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua
University of Utah

NAME

rle_getskip − Skip the rest of an input image.

SYNOPSIS

#include <rle.h>

unsigned int rle_getskip(in_hdr)
rle_hdr *in_hdr;

DESCRIPTION

This routine skips the unread part of an *RLE*(5) image. Each time *rle_getskip* is called, a scanline in the image described by *in_hdr* will be skipped. *rle_getskip* returns the scanline number of the next scanline that would be read by *rle_getrow*(3) or *rle_getraw*(3). When the end of the image is reached, *rle_getskip* returns 32768.

SEE ALSO

rle_hdr(3), *rle_getrow*(3), *rle_getraw*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_hdr − Structure for communication with RLE functions.

SYNOPSIS

#include <rle.h>

rle_hdr rle_dflt_hdr;

RLE_SET_BIT(the_hdr,bit)
RLE_CLR_BIT(the_hdr,bit)
RLE_BIT(the_hdr,bit)
rle_hdr the_hdr;

DESCRIPTION

This data structure provides communication to and between all the *RLE*(5) file routines. It describes the parameters of the image being saved or read, and contains some variables describing file state that are private to the routines. The public components are described below.

```
typedef unsigned char rle_pixel;
typedef unsigned short rle_map;

rle_hdr {
        int         ncolors,        /* Number of colors being saved */
                    *bg_color,      /* Background color array */
                    alpha,          /* if ≠ 0, save alpha channel (color -1) */
                                    /* alpha channel background is always 0 */
                    background,     /* if = 0, no background processing */
                                    /* if = 1 or 2, save only non-bg pixels */
                                    /* If 2, set clear-to-bg flag in file */
                    xmin,           /* Min X bound of saved raster */
                    xmax,           /* Max X bound */
                    ymin,           /* Min Y bound */
                    ymax,           /* Max Y bound */
                    ncmap,          /* number of color channels in color map */
                                    /* if = 0, color map is not saved */
                    cmaplen;        /* Log2 of the number of entries in */
                                    /* each channel of the color map */
        rle_map     *cmap;          /* pointer to color map, stored as 16-bit */
                                    /* words, with values left justified */
        char        **comments;     /* Pointer to array of pointers */
                                    /* to comment strings. */
        FILE *      rle_file;       /* I/O to this file */
        /*
         * Bit map of channels to read/save.  Indexed by (channel mod 256).
         */
        char    bits[256/8];
};
```

A global variable, *rle_dflt_hdr*, is available, conveniently initialized with default values.

FIELDS

ncolors The number of colors (exclusive of the alpha channel) in the image. This is one greater than the largest channel index (i.e., *ncolors* would be 3 if channels 0, 1, and 2 were saved, or if only channel 2 were saved.)

bg_color

A pointer to an array of *ncolors* integers, defines the background color (if used). The background alpha value is always 0, so is not included in the *bg_color* array.

alpha If non-zero, an alpha channel is present as channel −1. This should always be 0 or 1. *Rle_get_setup* and *rle_put_setup* enforce this constraint. The alpha channel will only be actually read or written if the corresponding bit in *bits* is also set.

background
Controls whether background color processing is done. If 0, no background processing is done at all (and *bg_color* is ignored). If 1 or 2, then runs of 3 or more pixels in the background color are not saved at all. If 2, then these runs will be restored by *rle_getrow*; if 1, they will not (this can lead to some strange images).

xmin, xmax, ymin, ymax
The bounds of the image. All pixels from *xmin* to *xmax*, inclusive, in rows numbered from *ymin* to *ymax*, inclusive, will be saved. Thus the dimensions of the image are

(xmax - xmin + 1) × *(ymax - ymin + 1)*

ncmap, cmaplen
The size of the saved colormap (if any). The color map will have *ncmap* channels, each *2^cmaplen* long. If *ncmap* is zero, no color map is present.

cmap A pointer to colormap data, if present. The data is stored in "channel major" order, so that all the values for channel 0 precede all the values for channel 1, etc. Each individual value is left-justified in 16 bits (i.e., the range of values is 0−65535).

comments
A pointer to picture comment data, if present. Use the functions *rle_putcom*(3), *rle_getcom*(3), and *rle_delcom*(3) to manipulate this field.

rle_file The standard I/O *FILE* pointer for the file containing this image.

bits A bitmap that selects the channels that are actually written to/read from the file. The macros below are used to modify this bitmap.

MACROS

The macro *RLE_BIT* will retrieve the state of one of the bits in the *bits* map. *RLE_SET_BIT*, and *RLE_CLR_BIT* set and clear bits in the *bits* map. The predefined symbols *RLE_RED*, *RLE_GREEN*, *RLE_BLUE*, and *RLE_ALPHA*, or an integer value from −1 to 254 may be used in these macros.

SEE ALSO

librle(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua

NAME

rle_op − Data structure for raw run-length encoded image data.

SYNOPSIS

#include <rle.h>
#include <rle_raw.h>
typedef struct rle_op rle_op;

DESCRIPTION

The *rle_op* data structure is used to describe a single run of data in a *RLE*(5) run-length encoded image. It is filled by the function *rle_getraw*(3), and is used by the functions *rle_putraw*(3) and *rle_rawtorow*(3).

The structure is

```
struct rle_op {
        int         opcode;     /* One of RByteDataOp or RRunDataOp. */
        int         xloc;       /* X starting location of this data. */
        int         length;     /* Length of run or data array. */
        union {
                    rle_pixel   *pixels;    /* ByteData case. */
                    int         run_val;    /* RunData case. */
        } u;
};
```

If the *opcode* has the value *RByteDataOp*, then the *u.pixels* component points to an array of *length* pixel values. If the *opcode* has the value *RRunDataOp*, then the *u.run_val* component contains a pixel value that is to be repeated *length* times.

SEE ALSO

rle_hdr(3), *rle_getraw*(3), *rle_putraw*(3), *rle_rawtorow*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas

NAME

rle_open_f − Open a binary file for input or output with defaults.
rle_open_f_noexit − Returns error code instead of exiting.

SYNOPSIS

FILE *rle_open_f(prog_name, file_name, mode)
char *prog_name, *file_name, *mode;

FILE *rle_open_f_noexit(prog_name, file_name, mode)
char *prog_name, *file_name, *mode;

DESCRIPTION

The function *rle_open_f* is provided to simplify the task of opening files in toolkit programs. It works similarly to *fopen*(3), but it also provides error checking and messages, and default values for input and output. If the specified *file_name* cannot be opened, an error message is printed and the program exits. A variant *rle_open_f_noexit* is provided which will return NULL if the file cannot be opened. An error message is still printed.

On those systems which require it, a ’b’ will be appended to the mode string so that the file will be opened in binary mode.

If the *file_name* is NULL or "−", then *stdin* will be returned for input (*mode* "r") files and *stdout* will be returned for output (*mode* "w" or "a") files.

The following two options are available only on systems supporting pipes. If the *file_name* starts with a "|" character, then the rest of the file name will be taken as a *sh*(1) command. If *mode* is "r", a pipe from the output of the *sh* command will be returned. If *mode* is "w" or "a", a pipe to the input of the *sh* command will be returned.

If the *file_name* ends with the suffix ".Z" (and does not start with "|"), then the *compress*(1) program will be invoked to uncompress (*mode* "r") or compress (*mode* "w" or "a") the file. The file descriptor returned by *rle_open_f* will be a pipe from or to the compress program.

SEE ALSO

fopen(3), *popen*(3), *compress*(1).

AUTHOR

Gerald Winter
Spencer W. Thomas
University of Michigan

BUGS

If the command invoked via *popen* does not exist, the *popen* still returns successfully, and the underlying *sh* prints an error message.

There is no way of telling that a particular *FILE* pointer has been created by *popen*, so it isn’t possible to cleanly close the pipe with *pclose*. In fact, the eventual output file may not even exist by the time the program exits.

NAME

rle_put_setup − setup to create an RLE file.
rle_put_init − setup for writing to an RLE file.

SYNOPSIS

#include <rle.h>

void rle_put_setup(the_hdr);
rle_hdr * the_hdr;

void rle_put_init(the_hdr);
rle_hdr * the_hdr;

DESCRIPTION

Rle_put_setup is called to initialize the output and write the image header of an *RLE*(5) image. The argument is a pointer to a *rle_hdr*(3) structure, which has been filled in with appropriate values for the image being saved.

Rle_put_init is called to initialize the header data structure for writing output to an *RLE* file. The argument is a pointer to a *rle_hdr*(3) structure, which has been filled in with appropriate values for the image being saved. The "private" elements of the header will be initialized. The header is not written to the file. This function could be useful for appending image data to an existing file. The new data should have the same number channels, the same width, etc. as the existing image.

SEE ALSO

rle_hdr(3), *rle_putrow*(3), *rle_putraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua

NAME

rle_putcom − set the value of a picture comment in an RLE file.
rle_getcom − get a picture comment from an RLE file.
rle_delcom − delete a picture comment from an RLE file.

SYNOPSIS

#include <rle.h>

char * rle_putcom(value, the_hdr)
char * value;
rle_hdr * the_hdr;

char * rle_getcom(name, the_hdr)
char * name;
rle_hdr * the_hdr;

char * rle_delcom(name, the_hdr)
char * name;
rle_hdr * the_hdr;

DESCRIPTION

Rle_putcom can be used to add a picture comment or change the value of a picture comment in a *rle_hdr*(3) structure. The argument *value* is the string value of the comment, and is generally of the form *name=value*. It may also be of the form *name*. If there is another comment with the same *name*, it will be replaced with the new *value*, and the previous comment will be returned as the value of *rle_putcom*.

Rle_getcom returns a pointer to the data portion of a picture comment from an RLE file. The comment is assumed to be in the form *name=value*; a pointer to *value* is returned. If the comment is of the form *name*, a pointer to the null character at the end of the string is returned. If there is no comment of the above forms, a *NULL* pointer is returned. The *the_hdr* structure contains the picture comments in question.

Rle_delcom is used to delete a picture comment from a *rle_hdr*(3) structure. It is called with the *name* of the comment and the *the_hdr* structure to be modified. The first comment in the *rle_hdr* structure of the form *name=value* or *name* will be deleted. The deleted comment will be returned as the function value.

SEE ALSO

rle_addhist(3), *rle_hdr*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_puteof − write an end of image to an RLE file.

SYNOPSIS

#include <rle.h>

rle_puteof(the_hdr);
rle_hdr * the_hdr;

DESCRIPTION

Call *rle_puteof* to write an end of image opcode into an *RLE*(5) file. *Rle_puteof* also frees some storage allocated by *rle_putrow*(3), "flushes" the output file, and generally cleans up.

SEE ALSO

rle_hdr(3), *rle_put_setup*(3), *rle_putrow*(3), *rle_putraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_putraw − write run length encoded data to an RLE file.

SYNOPSIS

#include <rle.h>
#include <rle_raw.h>

rle_putraw(scanraw, nraw, the_hdr);
rle_op ** scanraw;
int * nraw;
rle_hdr * the_hdr;

DESCRIPTION

The function *rle_putraw* provides a structured method for creating run length encoded output. It is passed an array, *scanraw*, of pointers to arrays of *rle_op*(3) structures, and an array of lengths. Each *rle_op* structure specifies a run or sequence of pixel values. The array *nraw* gives the number of *rle_op* structures for each channel. I.e., *nraw[i]* is the length of the array pointed to by *scanraw[i]*.

SEE ALSO

rle_hdr(3), *rle_op*(3), *rle_put_setup*(3), *rle_puteof*(3), *rle_skiprow*(3), *rle_raw_alloc*(3), *rle_raw_free*(3), *rle_getraw*(3), *rle_freeraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_putrow − Write a row (scanline) of data to an RLE file.

SYNOPSIS

#include <rle.h>

void rle_putrow(rows, length, the_hdr);
rle_pixel ** rows;
int length;
rle_hdr * the_hdr;

DESCRIPTION

Rle_putrow is called for each output scanline when creating an *RLE*(5) image. *Rows* is an array of pointers to the pixel data for the color components of the scanline. Rows should have *the_hdr→ncolors* elements. If an alpha channel is being saved, rows[-1] should point to the alpha channel data. *Length* is the number of pixels in the scanline. *Rows[i]* should point to the *the_hdr→xmin* element of the scanline.

The function *rle_row_alloc*(3) will properly allocate memory for use by *rle_putrow*.

SEE ALSO

rle_hdr(3), *rle_skiprow*(3), *rle_putraw*(3), *rle_puteof*(3), *rle_row_alloc*(3), *rle_row_free*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, Todd Fuqua

BUGS

Having the scanline indexed from *xmin* is an incredible botch. Its origin lies in the deep dark history of the raster toolkit, and it seems it’s too late to change it now.

NAME

rle_raw_alloc − Allocate memory for rle_getraw or rle_putraw.
rle_raw_free − free memory allocated by rle_raw_alloc.

SYNOPSIS

#include <rle.h>
#include <rle_raw.h>

rle_raw_alloc(the_hdr, scanp, nrawp)
rle_hdr * the_hdr;
rle_op * scanp;**
int ** nrawp;

rle_raw_free(the_hdr, scanp, nrawp)
rle_hdr * the_hdr;
rle_op ** scanp;
int * nrawp;

DESCRIPTION

The function *rle_raw_alloc* is provided to make it easier to allocate storage for use by the RLE "raw" functions. It examines the *the_hdr* structure provided and return (via its other arguments) newly allocated space suitable for reading from or writing to an RLE file described by the *the_hdr* structure. *Rle_raw_alloc* allocates (*the_hdr→xmax* − *the_hdr→xmin* + 1) elements per channel, which is more than should ever be needed for a valid *RLE* file.

Rle_raw_free should be used to free memory allocated by *rle_raw_alloc*(3). The arguments are pointers to the allocated storage. This is distinct from *rle_freeraw*(3), which only frees pixel arrays referenced by individual *rle_op* structures, while *rle_raw_free* frees the storage consumed by the arrays of pointers and *rle_op* structures. In fact, *rle_freeraw* should be called before calling *rle_raw_free*.

SEE ALSO

rle_hdr(3), *rle_op*(3), *rle_putraw*(3), *rle_getraw*(3), *rle_freeraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

BUGS

The naming confusion between *rle_freeraw* and *rle_raw_free* is unfortunate.

NAME

rle_rawtorow − Convert "raw" RLE data to scanline form.

SYNOPSIS

#include <rle.h>
#include <rle_raw.h>

rle_rawtorow(the_hdr, raw, nraw, outrows)
rle_hdr *the_hdr;
rle_op **raw;
int *nraw;
rle_pixel **outrows;

DESCRIPTION

Rle_rawtorow interprets the "raw" run-length encoded data in *raw*, such as might be returned by *rle_getraw*(3), and produces the corresponding scanline data in *outrows*, such as would have been returned by *rle_getrow*(3).

SEE ALSO

rle_hdr(3), *rle_op*(3), *rle_getraw*(3), *rle_getrow*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas, after code by Rod G. Bogart and John W. Peterson.

NAME

rle_row_alloc − Allocate scanline memory for rle_putrow or rle_getrow.
rle_row_free − Free scanline memory allocated by rle_row_alloc.

SYNOPSIS

#include <rle.h>

rle_row_alloc(the_hdr, scanp)
rle_hdr * the_hdr;
rle_pixel * scanp;**

rle_row_free(the_hdr, scanp)
rle_hdr * the_hdr;
rle_pixel ** scanp;

DESCRIPTION

The function *rle_row_alloc* is provided to make it easier to allocate storage for use by the RLE functions. It examines the *the_hdr* structure provided and returns (via its other argument) newly allocated space suitable for reading from or writing to an RLE file described by the *the_hdr* structure. *rle_row_alloc* allocates (*the_hdr→xmax* + 1) bytes for each scanline, to allow for *rle_getrow* usage. Only those rows enabled by the bit-map in *the_hdr* will have memory allocated.

To free memory allocated by *rle_row_alloc*(3), call *rle_row_free* with the pointer to the allocated storage.

SEE ALSO

rle_hdr(3), *rle_getrow*(3), *rle_putrow*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

NAME

rle_skiprow − Skip output scanlines in an RLE file.

SYNOPSIS

#include <rle.h>

rle_skiprow(the_hdr, nrow)
rle_hdr * the_hdr;
int nrow;

DESCRIPTION

This routine is used to output blank (background) scanlines to an *RLE*(5) file. It is used in conjunction with *rle_putrow*(3) or *rle_putraw*(3). The number of scanlines indicated by *nrow* will be blank in the output file.

SEE ALSO

rle_hdr(3), *rle_put_setup*(3), *rle_putrow*(3), *rle_putraw*(3), *librle*(3), *RLE*(5).

AUTHOR

Spencer W. Thomas
University of Utah

BUGS

Rle_skiprow should not be called when creating an *RLE* file with *the_hdr→background* set to zero. The specified number of rows will indeed be skipped, but they will not be filled with background when the file is read.

NAME

rle − Run length encoded file format produced by the rle library

DESCRIPTION

The output file format is (note: all words are 16 bits, and in PDP-11 byte order):

Word 0

A "magic" number 0xcc52. (Byte order 0x52, 0xcc.)

Words 1-4

The structure (chars saved in PDP-11 order)

```
{
   short   xpos,                  /* Lower left corner
           ypos,
           xsize,                 /* Size of saved box
           ysize;
}
```

Byte 10

(flags) The following flags are defined:

H_CLEARFIRST

(0x1) If set, clear the frame buffer to background color before restoring.

H_NO_BACKGROUND

(0x2) If set, no background color is supplied. If *H_CLEARFIRST* is also set, it should be ignored (or alternatively, a clear-to-black operation could be performed).

H_ALPHA

(0x4) If set, an alpha channel is saved as color channel -1. The alpha channel does not contribute to the count of colors in *ncolors*.

H_COMMENT

(0x8) If set, comments will follow the color map in the header.

Byte 11

(ncolors) Number of color channels present. 0 means load only the color map (if present), 1 means a B&W image, 3 means a normal color image.

Byte 12

(pixelbits) Number of bits per pixel, per color channel. Values greater than 8 currently will not work.

Byte 13

(ncmap) Number of color map channels present. Need not be identical to *ncolors*. If this is non-zero, the color map follows immediately after the background colors.

Byte 14

(cmaplen) Log base 2 of the number of entries in the color map for each color channel. I.e., would be 8 for a color map with 256 entries.

Bytes 15−...

The background color. There are *ncolors* bytes of background color. If *ncolors* is even, an extra padding byte is inserted to end on a 16 bit boundary. The background color is only present if *H_NO_BACKGROUND* is not set in *flags*. IF *H_NO BACKGROUND* is set, there is a single filler byte. Background color is ignored, but present, if *H_CLEARFIRST* is not set in *flags*.

If *ncmap* is non-zero, then the color map will follow as *ncmap**2^*cmaplen* 16 bit words. The color map data is left justified in each word.

If the *H_COMMENT* flag is set, a set of comments will follow. The first 16 bit word gives the length of the comments in bytes. If this is odd, a filler byte will be appended to the comments. The comments are interpreted as a sequence of null terminated strings which should be, by

convention, of the form *name=value*, or just *name*.

Following the setup information is the Run Length Encoded image. Each instruction consists of an opcode, a datum and possibly one or more following words (all words are 16 bits). The opcode is encoded in the first byte of the instruction word. Instructions come in either a short or long form. In the short form, the datum is in the second byte of the instruction word; in the long form, the datum is a 16 bit value in the word following the instruction word. Long form instructions are distinguished by having the 0x40 bit set in the opcode byte. The instruction opcodes are:

SkipLines (1)
The datum is an unsigned number to be added to the current Y position.

SetColor (2)
The datum indicates which color is to be loaded with the data described by the following ByteData and RunData instructions. Typically, 0→red, 1→green, 2→blue. The operation also resets the X position to the initial X (i.e. a carriage return operation is performed).

SkipPixels (3)
The datum is an unsigned number to be added to the current X position.

ByteData (5)
The datum is one less than the number of bytes of color data following. If the number of bytes is odd, a filler byte will be appended to the end of the byte string to make an integral number of 16-bit words. The X position is incremented to follow the last byte of data.

RunData (6)
The datum is one less than the run length. The following word contains (in its lower 8 bits) the color of the run. The X position is incremented to follow the last byte in the run.

EOF (7)
This opcode indicates the logical end of image data. A physical end-of-file will also serve as well. The **EOF** opcode may be used to concatenate several images in a single file.

SEE ALSO

librle(3)

AUTHOR

Spencer W. Thomas, Todd Fuqua