Table Of Contents

NAME

411toppm - convert Sony Mavica .411 image to PPM

SYNOPSIS

411toppm [**-width** *width*] [**-height** *height*] [*411file*]

All options may be abbreviated to the shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

411toppm reads a .411 file, such as from a Sony Mavic camera, and converts it to a PPM image as output.

Output is to Standard Output.

The originator of this program and decipherer of the .411 format, Steve Allen <*sla@alumni.caltech.edu*>, has this to say about the utility of this program: 'There's so little image in a 64x48 thumbnail (especially when you have the full size JPG file) that the only point in doing this was to answer the implicit challenge posed by the manual stating that only the camera can use these files.'

OPTIONS

-width The width (number of columns) of the input image. Default is 64.

-height The height (number of rows) of the input image. Default is 48.

SEE ALSO

ppm(1)

Table Of Contents

NAME

anytopnm - convert an arbitrary type of image file to PBM, PGM, or PPM

SYNOPSIS

anytopnm [*file*]

DESCRIPTION

This program is part of **Netpbm**(1).

anytopnm converts the input image, which may be in any of dozens of graphics formats, to PBM, PGM, or PPM format, depending on that nature of the input image, and outputs it to Standard Output.

To determine the format of the input, **anytopnm** uses the **file** program (possibly assisted by the magic numbers file fragment included with Netpbm). If that fails (very few image formats have magic numbers), **anytopnm** looks at the filename extension. If that fails, **anytopnm** punts.

The type of the output file depends on the input image.

If **file** indicates that the input file is compressed (either via Unix compress, gzip, or bzip compression), **anytopnm** uncompresses it and proceeds as above with the uncompressed result.

If **file** indicates that the input file is encoded by uuencode or btoa, **anytopnm** decodes it and proceeds as above with the decoded result.

If *file* is **-** or not given, **anytopnm** takes its input from Standard Input.

SEE ALSO

pnmfile(1), **pnm**(1), **file** man page

AUTHOR

Table Of Contents

NAME

asciitopgm - convert ASCII graphics into a PGM

SYNOPSIS

asciitopgm [**-d** *divisor*] *height width* [*asciifile*]

DESCRIPTION

This program is part of **Netpbm**(1).

asciitopgm reads ASCII data as input and produces a PGM image with pixel values which are an approximation of the 'brightness' of the ASCII characters, assuming black-on-white printing. In other words, a capital M is very dark, a period is very light, and a space is white.

Obviously, **asciitopgm** assumes a certain font in assigning a brightness value to a character.

asciitopgm considers ASCII control characters to be all white. It assigns special brightnesses to lower case letters which have nothing to do with what they look like printed. **asciitopgm** takes the ASCII character code from the lower 7 bits of each input byte. But it warns you if the most signficant bit of any input byte is not zero.

Input lines which are fewer than *width* characters are automatically padded with spaces.

The *divisor* value is an integer (decimal) by which the blackness of an input character is divided; the default value is 1. You can use this to adjust the brightness of the output: for example, if the image is too bright, increase the divisor.

In keeping with (I believe) Fortran line-printer conventions, input lines beginning with a **+** (plus) character are assumed to 'overstrike' the previous line, allowing a larger range of gray values.

If you're looking for something that creates an image of text, with that text specified in ASCII, that is something quite different. Use **pbmtext** for that.

SEE ALSO

pbmtoascii(1), **pbmtext**(1), **pgm**(1)

AUTHOR

Wilson H. Bent. Jr. (*whb@usc.edu*)

Table Of Contents

NAME

atktopbm - convert Andrew Toolkit raster object to PBM

SYNOPSIS

atktopbm [*atkfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

atktopbm reads an Andrew Toolkit raster object as input. and produces a PBM image as output.

SEE ALSO

pbmtoatk(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

bioradtopgm - convert a Biorad confocal file into a PGM image

SYNOPSIS

bioradtopgm [**-image#**] [*imagedata*]

DESCRIPTION

This program is part of **Netpbm**(1).

bioradtopgm reads a Biorad confocal file as input and produces a PGM image as output. If the resulting image is upside down, run it through **pamflip -tb**.

OPTIONS

-image#

A Biorad image file may contain more than one image. With this flag, you can specify which image to extract (only one at a time). The first image in the file has number zero. If no image number is supplied, only information about the image size and the number of images in the input is printed out. No output is produced.

SEE ALSO

pamflip(1), **pgm**(1)

AUTHORS

Table Of Contents

NAME

bmptopnm - convert a BMP file into a PBM, PGM, or PNM image

SYNOPSIS

bmptopnm [*bmpfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

bmptopnm reads a Microsoft Windows or OS/2 BMP file as input. and produces a PBM, PGM, or PNM image as output. If the input is colormapped and contains only black and white, the output is PBM. If the input is colormapped and contains only black white and gray, the output is PGM. Otherwise, the output is PPM.

This program cannot convert BMP files with compressed (run length encoded, JPEG, PNG) image data. It recognizes the compression and issues an error message.

Before Netpbm 10.18 (September 2003), this program could not convert BMP images with the BI_BITFIELDS format ("compression type"). It would recognize the format and issue an error message.

This program cannot convert OS/2 BMP files with 16 bits per pixel (only because the author did not have a complete specification for them). It recognizes the format and issues an error message. Before Netpbm 10.16 (June 2003), it also could not convert Windows BMP files with 16 bits per pixel.

SEE ALSO

ppmtobmp(1), **ppmtowinicon**(1), **ppm**(1)

AUTHOR

NAME

bmptoppm - replaced by bmptopnm

DESCRIPTION

This program is part of **Netpbm**(1).

bmptoppm was replaced in Netpbm 9.25 (March 2002) by **bmptopnm**(1).

bmptopnm is backward compatible with **bmptoppm** except that it generates PBM and PGM output when it is more appropriate than PPM.

Table Of Contents

NAME

brushtopbm - convert a doodle brush file into a PBM image

SYNOPSIS

brushtopbm [*brushfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

brushtopbm reads a Xerox doodle brush file as input. and produces a portable bitmap as output.

Note that there is currently no pbmtobrush tool.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

cmuwmtopbm - convert a CMU window manager bitmap into a PBM image

SYNOPSIS

cmuwmtopbm [*cmuwmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

cmuwmtopbm reads a CMU window manager bitmap as input. and produces a PBM image as output.

SEE ALSO

pbmtocmuwm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

ddbugtopbm - convert Diddle or DiddleBug sketches to PBM files

SYNOPSIS

ddbugtopbm

EXAMPLES

ddbugtopbm </path/to/palm/backup/dir/DiddleBugDB.pdb

ddbugtopbm </path/to/palm/backup/dir/DiddleDB.pdb

ddbugtopbm </path/to/palm/backup/dir/DiddleIDB.pdb

DESCRIPTION

This program is part of **Netpbm**(1).

ddbugtopbm converts all sketches present in a database used by the PalmOS programs **Diddle** or **DiddleBug** into appropriately-named PBM files. The backup copy of DiddleBug's database you should use as this program's input is usually called **DiddleBugDB.pdb**. Or if you use the original Diddle, it has two separate DBs - **DiddleDB.pdb**, containing unnamed 'scratch' sketches, and **DiddleIDB.pdb**, containing the saved (and named) sketches which are listed by its 'index' option. You can feed this program any of these three on standard input.

USING THE PROGRAM

I recommend you *not* run **ddbugtopbm** from your Palm backup directory, i.e. don't run it from the directory the DB will normally be in. Instead, run it from some other directory (perhaps you could make a directory purely to hold the PBM files, just to keep things simple) and use an absolute or relative path to the DB.

The filenames used for the output PBMs are based on the names given to each sketch; if you have an unnamed sketch, it's given a name along the lines of **sketch-0123.pbm**.

While the named sketches will overwrite any existing PBM file with the same name, the unnamed ones won't - they'll just try using another filename. (I think this is probably the right approach, as you can't really tell the unnamed sketches apart.)

BUGS

The DiddleBug DB reader is only known to work with DBs from DiddleBug version 2.50. But it should probably work on later versions, and I think it'll work on DBs from version 2.15 as well.

It might fall over if fed an empty database, and doesn't do much (if any) checking of the input.

AUTHOR

Russell Marks (*rus@svgalib.org*).

Mitch Blevins's decompression code is directly from DiddleBug itself, which like ddbugtopbm is distributed under the terms of the GNU GPL.

SEE ALSO

palmtopnm(1), **pbm**(1)

Jens-Chr. Heyer's 'didcon' script does something similar.

HISTORY

ddbugtopbm was new in Netpbm 10.18 (August 2003). It was written and independently distributed in August 2002.

Created: 1 August 2003 Table Of Contents

NAME

escp2topbm - convert an ESC/P2 printer file to a PBM image

SYNOPSIS

escp2topbm [*printfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

escp2topbm reads an ESC/P2 printer control stream as input. It produces a PBM image as output.

escp2topbm filters the raster graphic content from an Epson ESC/P2 printer control stream and writes the image it would print as a standard (raw) PBM image.

The input is from the file named by the *printfile* argument, or from Standard Input if you don't specify *printfile*. The output is to Standard Output.

escp2topbm understands compression modes 0 (uncompressed) and 1 (RLE compressed) in the Epson input stream.

OPTIONS

none

HINTS

As **escp2topbm** is a simple program, created mainly to test **pbmtoescp2**, there are some restrictions:

- **escp2topbm** looks only at "ESC." sequences and ignores all data outside these Escape sequences.
- **escp2topbm** assumes that only one raster graphic is in the printer stream. If this isn't true, the result is garbage.
- **escp2topbm** assumes that all "ESC." sequences use the same width value. If this isn't true, the result is garbage.

SEE ALSO

pbmtoescp2(1), **pbm**(1)

AUTHOR

HISTORY

escp2topbm was added to Netpbm in Release 10.18 (August 2003); it was created around the same time.

Table Of Contents

NAME

eyuvtoppm - convert a Berkeley YUV file to a portable pixmap (ppm) file

SYNOPSIS

eyuvtoppm [**--width** *width*] [**--height** *height*] [*eyuvfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

eyuvtoppm reads a Berkeley Encoder YUV (not the same as Abekas YUV) file as input and produces a PPM image on Standard Output.

With no filename argument takes input from Standard Input. Otherwise, *eyuvfile* is the file specification of the input file.

SEE ALSO

ppmtoeyuv(1), **yuvtoppm**(1), **ppm**(1)

Table Of Contents

NAME

fiascotopnm - Convert compressed FIASCO image to PGM, or PPM

SYNOPSIS

fiascotopnm [*option*]... [*filename*]...

DESCRIPTION

This program is part of **Netpbm**(1).

fiascotopnm decompresses the named FIASCO files, or the Standard Input if no file is named, and writes the images as PGM, or PPM files, depending on whether the FIASCO image is black and white or color.

OPTIONS

All option names may be abbreviated; for example, --output may be written --outp or --ou. For all options an one letter short option is provided. Mandatory or optional arguments to long options are mandatory or optional for short options, too. Both short and long options are case sensitive.

-o[*name*], **--output=**[*name*]

Write decompressed image to the file *name*.ppm (if PPM) or *name*.pgm (if PGM). If *name* is -, then produce the image file on the standard output. The optional argument *name* can be omitted, then the input filename is used as basename with the suffix .ppm or .pgm. In case of video streams, the frames are stored in the files *name*.**N**.ppm where **N** is the frame number (of the form 00..0 - 99..9); output on the standard output is not possible with video streams.

If *name* is a relative path and the environment variable **FIASCO_IMAGES** is a (colon-separated) list of directories, then the output file(s) are written to the first (writable) directory of this list. Otherwise, the current directory is used to store the output file(s).

-z, **--fast**

Decompress images in the 4:2:0 format; i.e., each chroma channel is decompressed to an image of halved width and height. Use this option on slow machines when the desired frame rate is not achieved; the output quality is only slightly decreased.

-d, **--double**

Double the size of the X11 window both in width and height; no pixel interpolation is used, each pixel is just replaced by four identical pixels.

-p, **--panel**

Show a panel with play, stop, pause, record and exit buttons to control the display of videos. When pressing the record button, all frames are decompressed and stored in memory. The other buttons work in the usual way.

-m *N*, **--magnify=***N*

Set magnification of the decompressed image. Positive values enlarge and negative values reduce the image width and height by a factor of 2^|*N*|.

-s *N*, **--smooth=***N*

Smooth decompressed image(s) along the partitioning borders by the given amount *N*. *N* is 1 (minimum) to 100 (maximum); default is 70. When *N*=0, then the smoothing amount specified in the FIASCO file is used (defined by the FIASCO coder).

-F *N*, **--fps=***N*
Set number of frames per second to *N*. When using this option, the frame rate specified in the FIASCO file is overridden.

-v, **--version**
Print **fiascotopnm** version number, then exit.

-f *name*, **--config=***name*
Load parameter file *name* to initialize the options of **fiascotopnm**. See file **system.fiascorc** for an example of the syntax. Options of **fiascotopnm** are set by any of the following methods (in the specified order):

- Global ressource file **/etc/system.fiascorc**
- $HOME/**.fiascorc**
- command line
- --config=*name*

-h, **--info**
Print brief help, then exit.

-H, **--help**
Print detailed help, then exit.

EXAMPLES

fiascotopnm foo.wfa >foo.ppm

Decompress the FIASCO file 'foo.wfa' and store it as 'foo.ppm'.

fiascotopnm -o foo1.wfa foo2.wfa

Decompress the FIASCO files 'foo1.wfa' and 'foo2.wfa' and write the frames to the image files 'foo1.wfa.ppm' and 'foo2.wfa.ppm'.

fiascotopnm -oimage foo1.wfa

Decompress the FIASCO file 'foo1.wfa' and write all 15 frames to the image files 'image.00.ppm', ... , 'image.14.ppm'.

fiascotopnm --fast --magnify=-1 --double video.wfa >stream.ppm

Decompress the FIASCO file 'video.wfa'. The decompression speed is as fast as possible: the image is decompressed (in 4:2:0 format) at a quarter of its original size; then the image is enlarged again by pixel doubling.

FILES

/etc/system.fiascorc
The systemwide initialization file.

$HOME/**.fiascorc**
The personal initialization file.

ENVIRONMENT

FIASCO_IMAGES
Save path for image files. Default is ’./’.

FIASCO_DATA
Search path for FIASCO files. Default is ’./’.

SEE ALSO

pnmtofiasco(1), **pnm**(1)

Ullrich Hafner, Juergen Albert, Stefan Frank, and Michael Unger. **Weighted Finite Automata for Video Compression**, IEEE Journal on Selected Areas In Communications, January 1998 Ullrich Hafner. **Low Bit-Rate Image and Video Coding with Weighted Finite Automata**, Ph.D. thesis, Mensch & Buch Verlag, ISBN 3-89820-002-7, October 1999.

AUTHOR

Ullrich Hafner <*hafner@bigfoot.de*>

Table Of Contents

NAME

fitstopnm - convert a FITS file into a PNM image

SYNOPSIS

fitstopnm [**-image** *N*] [**-scanmax**] [**-printmax**] [**-min** *f*] [**-max** *f*] [*FITSfile*]

All options may be abbreviated to their shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

fitstopnm reads a FITS file as input and produces a PPM image if the FITS file consists of 3 image planes (NAXIS = 3 and NAXIS3 = 3), or a PGM image if the FITS file consists of 2 image planes (NAXIS = 2), or if you specify the **-image** option. The results may need to be flipped top for bottom; if so, just pipe the output through **pamflip -tb**.

OPTIONS

The **-image** option is for FITS files with three axes. The assumption is that the third axis is for multiple images, and this option lets you select which one you want.

You can use options **-min** and **-max** to override the min and max values as read from the FITS header or the image data if no DATAMIN and DATAMAX keywords are found.

You can use the **-scanmax** option to force the program to scan the data even when DATAMIN and DATAMAX are found in the header. If you specify **-printmax**, the program will just print the min and max values and quit.

The program tells you what kind of PNM image it is writing.

REFERENCES

FITS stands for Flexible Image Transport System. A full description can be found in Astronomy & Astrophysics Supplement Series 44 (1981), page 363.

SEE ALSO

pnmtofits(1), **pamflip**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

fstopgm - convert a Usenix FaceSaver(tm) file into a PGM image

SYNOPSIS

fstopgm [*fsfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

fstopgm reads a Usenix FaceSaver(tm) file as input and produces a PGM image as output.

FaceSaver(tm) files sometimes have rectangular pixels. While **fstopgm** won't re-scale them into square pixels for you, it will give you the precise **pamscale** command that will do the job. Because of this, reading a FaceSaver(tm) image is a two-step process. First you do:

```
fstopgm > /dev/null
```

This will tell you whether you need to use **pamscale.**

Then use one of the following pipelines:

```
fstopgm | pgmnorm
fstopgm | pamscale -whatever | pgmnorm
```

To go to PBM, you want something more like one of these:

```
fstopgm | pnmenlarge 3 | pgmnorm | pamditherbw
fstopgm | pnmenlarge 3 | pamscale <whatever> | pgmnorm | pamditherbw
```

You want to enlarge when going to a bitmap because otherwise you lose information; but enlarging by more than 3 does not look good.

FaceSaver is a registered trademark of Metron Computerware Ltd. of Oakland, CA.

SEE ALSO

pgmtofs(1), **pgm**(1), **pgmnorm**(1), **pnmenlarge**(1), **pamscale**(1), **pamditherbw**(1)

AUTHOR

Table Of Contents

NAME

g3topbm - convert a Group 3 fax file into a PBM image

SYNOPSIS

g3topbm [**-kludge**] [**-reversebits**] [**-stretch**] [**-stop_error**] [*g3file*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

g3topbm reads a Group 3 fax file as input and produces a PBM image as output.

OPTIONS

-kludge

Tells **g3topbm** to ignore the first few lines of the file; sometimes fax files have some junk at the beginning.

-reversebits

Tells **g3topbm** to interpret bits least-significant first, instead of the default most-significant first. Apparently some fax modems do it one way and others do it the other way. If you get a whole bunch of 'bad code word' messages, try using this flag.

-stretch

Tells **g3topbm** to stretch the image vertically by duplicating each row. This is for the low-quality transmission mode.

-stop_error

Tells **g3topbm** to fail when it finds a problem in the input. 'Fail' means it terminates with a nonzero status code with the contents of the output file undefined.

If you don't specify this option, **g3topbm** does its best to work around input errors and salvage as much of the image as possible in the output image. It first tries to resynchronize to a later line by searching for the next End Of Line marker, skipping any lines or partial lines in between. It saves the beginning of the line in which it encountered the problem. If the input file ends prematurely, **g3topbm** produces output containing the lines up to where it encountered the problem.

g3topbm issues warning messages when it continues in spite of input errors.

This option was new in Netpbm 10.24 (August 2004). Before that, **g3topbm** always failed when it encountered premature EOF and never failed when it encountered other problems.

ABOUT G3

G3 is the near universal format used by fax machines. There is also a newer, more capable G4.

The standard for Group 3 fax is defined in CCITT Recommendation T.4. In the U.S., that is implemented by EIA standards EIA-465 and EIA-466. These standards cover the layers below the image format (which are irrelevant to **g3topbm** as well.

G3 faxes are 204 dots per inch (dpi) horizontally and 98 dpi (196 dpi optionally, in fine-detail mode) vertically. Since G3 neither assumes error free transmission nor retransmits when errors occur, the

encoding scheme used is differential only over small segments never exceeding 2 lines at standard resolution or 4 lines for fine-detail. (The incremental G3 encoding scheme is called two-dimensional and the number of lines so encoded is specified by a parameter called k.)

SEE ALSO

pbmtog3(1), **pbm**(1)

AUTHOR

NAME

gemtopbm - replaced by gemtopnm

DESCRIPTION

This program is part of **Netpbm**(1).

gemtopbm was replaced in Netpbm 9.1 (May 2000) by **gemtopnm**(1).

gemtopnm is backward compatible with **gemtopbm**, but works on color images as well.

Table Of Contents

NAME

gemtopnm - convert a GEM .img file into a PNM image

SYNOPSIS

gemtopnm [**-d**] [*gemfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

gemtopnm reads a GEM .img file, either the one plane (black/white) or four plane (16 color) varieety, as input and produces a PBM or PPM file as output, depending on whether the input is one or four plane.

The input file is *gemfile* if you specify that argument, or Standard Input otherwise. Output is to Standard Output.

OPTIONS

-d Produce output describing the contents of the .img file.

SEE ALSO

pbmtogem(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

giftopnm - convert a GIF file into a PNM image

SYNOPSIS

giftopnm [**--alphaout=**{*alpha-filename*,**-**}] [**-verbose**] [**-comments**] [**-image=**{*N*,**all**}] [*GIFfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

This is a graphics format converter from the GIF format to the PNM (i.e. PBM, PGM, or PPM) format.

If the image contains only black and maximally bright white, the output is PBM. If the image contains more than those two colors, but only grays, the output is PGM. If the image contains other colors, the output is PPM.

A GIF image contains rectangular pixels. They all have the same aspect ratio, but may not be square (it's actually quite unusual for them not to be square, but it could happen). The pixels of a Netpbm image are always square. Because of the engineering complexity to do otherwise, **giftopnm** converts a GIF image to a Netpbm image pixel-for-pixel. This means if the GIF pixels are not square, the Netpbm output image has the wrong aspect ratio. In this case, **giftopnm** issues an informational message telling you to run **pamscale** to correct the output.

OPTIONS

--alphaout=*alpha-filename*

giftopnm creates a PGM (portable graymap) file containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **--alphaout**, **giftopnm** does not generate an alpha file, and if the input image has an alpha channel, **giftopnm** simply discards it.

If you specify **-** as the filename, **giftopnm** writes the alpha output to Standard Output and discards the image.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

-verbose

Produce verbose output about the GIF file input.

-comments

Only output GIF89 comment fields.

-image={*N*,**all**}

This option identifies which image from the GIF stream you want. You can select either one image or all the images. Select al the images with **all**. Select one image by specifying its sequence number in the stream: **1**, **2**, **3**, etc.

The default is just Image 1.

A GIF stream normally contains only one image, so you don't need this option. But some streams, including animated GIFs, have multiple images.

When you select multiple GIF images, the output is a PNM stream with multiple images.

The **all** value was added in Netpbm 10.16 (June 2003). Earlier **giftopnm** can extract only one image.

RESTRICTIONS

This does not correctly handle the Plain Text Extension of the GIF89 standard, since I did not have any example input files containing them.

SEE ALSO

ppmtogif(1), **ppmcolormask**(1), **pamcomp**(1), http://www.lcdf.org/gifsicle , **ppm**(1).

AUTHOR

LICENSE

As a historical note, for a long time if you used **giftopnm**, you were using a patent on the LZW compression method which was owned by Unisys, and in all probability you did not have a license from Unisys to do so. Unisys typically asked $5000 for a license for trivial use of the patent. Unisys never enforced the patent against trivial users, and made statements that it is much less concerned about people using the patent for decompression (which is what **giftopnm** does than for compression. The patent expired in 2003.

Rumor has it that IBM also owns a patent covering **giftopnm**.

A replacement for the GIF format that has never required any patent license to use is the PNG format.

Table Of Contents

NAME

gouldtoppm - convert Gould scanner file into a PPM image

SYNOPSIS

gouldtoppm [*gouldfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

gouldtoppm reads a file produced by the Gould scanner as input and produces a PPM image as output.

SEE ALSO

ppm(1)

AUTHOR

Table Of Contents

NAME

hdifftopam - convert horizontal difference image to original PAM image

SYNOPSIS

hdifftopam[*pamfile*] [**-pnm**] [**-verbose**]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

hdifftopam undoes what **pamtohdiff** does.

OPTIONS

-pnm This option tells **hdifftopam** to create a PNM image (i.e. PGM or PPM). Without it, **hdifftopam** creates a PAM image (with a tuple type of "unhdiff"). If the PAM does not have the proper depth to be a PGM or PPM (i.e. 1 or 3) and you specify **-pnm**, **hdifftopam** fails.

SEE ALSO

pamtohdiff(1)

AUTHOR

Bryan Henderson

Table Of Contents

NAME

hipstopgm - convert a HIPS file into a PGM image

SYNOPSIS

hipstopgm [*hipsfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

hipstopgm reads a HIPS file as input and produces a PGM image as output.

If the HIPS file contains more than one frame in sequence, **hipstopgm** will concatenate all the frames vertically.

HIPS is a format developed at the Human Information Processing Laboratory, NYU.

SEE ALSO

pgm(1)

AUTHOR

Table Of Contents

NAME

hpcdtoppm - convert a Photo-CD image into a PBM image

SYNOPSIS

hpcdtoppm *infile* [**-a**] [{**-C**|**-0**|**-Overview**|**-O**} *file opt*] [**-c0**] [**-c-**] [**-c+**] [**-crop**] [**-d**] [**-dpi** *f*] [**-eps**] [**-epsd**] [**-epsg**] [**-fak** *scale*] [**-hori**] [**-i**] [**-l**] [**-m**] [**-n**] [**-pb** *pos*] [**-pgm**] [**-ph** *height*] [**-pl** *pos*] [**-pos**] [**-ppm**] [**-ps**] [**-psd**] [**-psg**] [**-pw** *width*] [**-r**] [**-rep**] [**-S** *long short*] [**-s**] [**-vert**] [**-x**] [**-ycc**] [**-1**|**-Base/16**|**-128x192**] [**-2**|**-Base/4**|**-256x384**] [**-3**|**-Base**|**-512x768**] [**-4**|**-4Base**|**-1024x1536**] [**-5**|**-16Base**|**-2048x3072**] [**-6**|**-64Base**|**-4096x6144**] [*outfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

This program accepts Photo-CD image or overview file data from the specified input file, *infile* (or, if the resolution is lower than 64Base and the file argument is specified as **-**, from standard input), and writes either PBM Format or PostScript to the specified output file (or to standard output if no file is specified).

On a standard Photo-CD, image files appear in the directory **photo_cd/images**, in files with names of the form img*nnnn*.pcd, where *nnnn* is a 4-digit-number. The overview file appears in **photo_cd/overview.pcd**.

Photo-CD images are stored using as many as 6 different resolutions:

Format	Resolution
64Base	4096x6144 (ProPhotoCD only)
16Base	2048x3072
4Base	1024x1536
Base	512x768
Base/4	256x384
Base/16	128x192

The overview file employs Base/16 format.

OPTIONS

Invoking **hpcdtoppm** without arguments produces a list of default values. Note that you can supply only one size option.

-a Automatically determine image orientation (this option is experimental, and does not work for overview files).

{**-C** | **-0** | **-Overview** | **-O** } *file opt*
Extract all images from an overview file. The mandatory *file* argument is the name of a PPM file; output files are named *filennnn*, where *nnnn* is a 4-digit number. Overview images are extracted in their original Base/16 format. The value of *opt* determines the orientation of the contact sheet image; recognized values are:

n Do not rotate the image.

l Rotate the picture counter-clockwise (portrait mode).

r Rotate the picture clockwise (portrait mode).

-c0 Do not correct (brighten or darken) the image.

-c- Darken the image.

-c+ Brighten the image.

-crop Cut off the black frame which sometimes appears at the image borders.

-d Show only the decompressed difference rather than the complete image (applicable only to 4Base and 16Base images).

-dpi *res*
Set the printer resolution to *res* for dithered Postscript images.

-eps Write a RGB Encapsulated Postscript color image.

-epsd Write a Floyd-Steinberg dithered image in Encapsulated Postscript.

-epsg Write a grayscale image in Encapsulated Postscript.

-fak *scale*
Set the scaling factor for dithered PostScript images to *scale*.

-hori Flip the image horizontally.

-i Send information from an image file header to Standard Error.

-l Rotate the picture counter-clockwise (portrait mode).

-m Write messages about the phases of decoding to standard error.

-n Do not rotate the image.

-pb *pos* Set the bottom position of the Postscript image to *pos*.

-pgm Write a *pgm* (grayscale) image.

-ph height
Set the height of the Postscript image to *height*.

-pl *pos* Set the leftmost position of the Postscript image to *pos*.

-pos Print the relative starting position of the data for the current resolution.

-ppm Write a *ppm* RGB (color) image.

-ps Write a RGB Postscript color image.

-psd Write a Floyd-Steinberg dithered image in Postscript.

-psg Write a Postscript grayscale image.

-pw width
Set the width of the Postscript image to *width*.

-r Rotate the picture clockwise (portrait mode).

-rep Try to jump over reading errors in the Huffman code.

-S *long short*
Cut out a subrectangle with boundaries defined by the values:

long For the longer side of the image.

short For the shorter side of the image.

where *long* and *short* take one of two forms:

a-b Cut from position *a* to position *b*.

a+b Starting at offset *a*, cut a length of *b*.

and where *a* and *b* are either integers representing pixel locations, or floating point values over the range [0.0 ... 1.0], representing the fraction of the length of a side.

-s Apply a simple sharpness operator to the luminosity channel.

-vert Flip the image vertically.

-x Overskip Mode (applicable to Base/16, Base/4, Base and 4Base). In Photo-CD images the luminosity channel is stored in full resolution, the two chromaticity channels are stored in half resolution only and have to be interpolated. In Overskip Mode, the chromaticity channels of the next higher resolution are taken instead of interpolating. To see the difference, generate one PPM with and one PPM without this option. Use **pamarith**(1)**to**generate**the**difference**image** of these two images. Call **ppmhist**(1) for this difference or show it with **xv** (push the **HistEq** button in the color editor).

-ycc Write the image in a variation on PPM format in which the samples are YCC instead of RGB.

-1|**-Base/16**|**-128x192**
Extract the Base/16 image.

-2|**-Base/4**|**-256x384**
Extract the Base/4 image.

-3|**-Base**|**-512x768**
Extract the Base image.

-4|**-4Base**|**-1024x1536**
Extract the 4Base image.

-5|**-16Base**|**-2048x3072**
Extract the 16Base image.

-6|**-64Base**|**-4096x6144**
Extract the 64Base image. This resolution can be extracted from ProPhotoCD images only. The path of the 64Base extension files is derived from the path to the image file. This means that it doesn't work on stdin an the directory structure must be the very same as on the ProPhotoCD.

Postcript Output

For Postscript output (options **-ps**, **-eps**, **-psg**, **-epsg**, **-psd**, **-epsg**) you can define both the resolution and placement of the image. Both size and position are specified in points (1/72 inch).

The position of the image (where the origin is assumed to be at the lower left corner of the page) is controlled by the **-pl** and **-pb** options (applicable at all resolutions).

The size of color and grayscale images is changed with the **-pw** and **-ph** options. Every image pixel is mapped onto one Postscript pixel.

There are three modes of control for dithered Postscript:

Image size
(**-pw** and **-ph**)

Printer resolution
(**-dpi**)

Scaling factor
(**-fak**)

These three factors are interdependent, hence no more then two can be specified simultaneously. Using **-dpi** and the **-pw**/**-ph** options together often yields pleasing results. Even using the default values for these options will produce results differing from those obtained without use of the options.

Limitations

The program ignores read protection.

The **-i** option is not working correctly.

Available information obout the Photo-CD format is vague; this program was developed by trial-and-error after staring at hex-dumps. Please send bugs reports and patches to the author.

SEE ALSO

pcdovtoppm(1), **pamarith**(1), **ppm**(1), **ppmhist**(1), **pnmquant**(1), **ppmtopgm**(1), **ppmtorgb3**(1), **xv**

VERSION

The name **hpcdtoppm** stands for 'Hadmut's pcdtoppm,' to make it distinguishable in the event that someone else is building a similar application and naming it **pcdtoppm**.

This is version 0.6.

AUTHOR

Manual page extensively modified by R. P. C. Rodgers (*rodgers@nlm.nih.gov*).

Table Of Contents

NAME

icontopbm - convert a Sun icon into a PBM image

SYNOPSIS

icontopbm [*iconfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

icontopbm reads a Sun icon as input and produces a PBM image as output.

SEE ALSO

pbmtoicon(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

ilbmtoppm - convert an ILBM file into a PPM image

SYNOPSIS

ilbmtoppm [**-verbose**] [**-ignore**<chunkID>**]** [**-isham**|**-isehb**] [**-adjustcolors**] [*ILBMfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ilbmtoppm reads an IFF ILBM file as input and produces a PPM image as output. **ilbmtoppm** can handle the following ILBM types:

- Normal ILBMs with 1-16 planes.
- Amiga Extra_Halfbrite (EHB)
- Amiga HAM with 3-16 planes.
- 24 bit.
- Multiplatte (normal or HAM) pictures.
- Color map (BMHD + CMAP chunk only, nPlanes = 0).
- Unofficial direct color. 1-16 planes for each color component.

ilbmtoppm uses these ILBM chunks: BMHD, CMAP, CAMG (only HAM & EHB flags used), PCHG, BODY unofficial DCOL chunk to identify direct color ILBM. It ignores these chunks: GRAB, DEST, SPRT, CRNG, CCRT, CLUT, DPPV, DRNG, EPSF. It ignores, but displays in verbose mode, these: NAME, AUTH, (c), ANNO, DPI. It skips chunks whose type it doesn't recognize.

OPTIONS

-verbose

Give some information about the ILBM file.

-ignore *chunkID*

Skip a chunk. *chunkID* is the 4-letter IFF chunk identifier of the chunk to be skipped.

-isham | **-isehb**

Treat the input file as a HAM or Extra_Halfbrite picture, even if these flags or not set in the CAMG chunk (or if there is no CAMG chunk).

-adjustcolors

If all colors in the CMAP have a value of less then 16, ilbmtoppm assumes a 4-bit colormap and gives a warning. With this option the colormap is scaled to 8 bits.

LIMITATIONS

The multipalette PCHG BigLineChanges and Huffman decompression code is untested.

REFERENCES

Amiga ROM Kernel Reference Manual - Devices (3rd Ed.) Addison Wesley, ISBN 0-201-56775-X

SEE ALSO

ppmtoilbm(1), **ppm**(1)

AUTHORS

Modified October 1993 by Ingo Wilken (*Ingo.Wilken@informatik.uni-oldenburg.de*)

Table Of Contents

NAME

imgtoppm - convert an Img-whatnot file into a PPM image

SYNOPSIS

imgtoppm [*imgfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

imgtoppmreads an Img-whatnot file as input and produces a PPM image as output. The Img-whatnot toolkit is available for FTP on venera.isi.edu, along with numerous images in this format.

SEE ALSO

ppm(1)

AUTHOR

Based on a simple conversion program posted to comp.graphics by Ed Falk.

Table Of Contents

NAME

infotopam - convert Amiga .info icons to PAM

SYNOPSIS

infotopam [**-forcecolor**] [**-numcolors** *numcolors*] [**-selected**] [*index color ...*] [*filename*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

EXAMPLES

By default, **infotopam** converts the first icon in a .info file:

```
infotopam amiga.info > amiga.first.pam
```

Use the *-selected* option to convert the second icon in a .info file. Here **infotopam** reads from Standard Input:

```
infotopam -selected < amiga.info > amiga.second.pam
```

Use the *-forcecolor* option to force color conversion for a 1 bit-plane .info file:

```
infotopam -forcecolor bw.info > bw.pam
```

Use *-numcolors* to override colors for indexes 0 and 3. Notice the two ways to specify the color:

```
infotopam -numcolors 2 0 green 3 #FF0000 icon.info > icon.pam
```

Since Amiga monitors do not use square pixels, some icons may appear squished. Filtering the output through **pamscale** can fix this:

```
infotopam squish.info | pamtopnm | pamscale -yscale 1.7 > normal.pnm
```

DESCRIPTION

This program is part of **Netpbm**(1).

infotopam converts an Amiga .info (icon) file to a PAM image. **infotopam** reads a .info file from *filename*, or from Standard Input if you do not specify a file name, and writes the converted PAM image to Standard Output.

infotopam currently handles 1 and 2 bit-plane icons. If the .info icon only has 1 bit-plane, **infotopam** generates a bitmap (black&white) PAM image; otherwise it generates a color PAM image. You can force **infotopam** to convert 1 bit-plane images to color PAM images by using the *-forcecolor* option.

OPTIONS

-forcecolor

Forces **infotopam** to convert 1 bit-plane icons to color PAM images instead of bitmap PAM images. **infotopam** uses the index 2 color for black and the index 1 color for white (more on this below).

-numcolors *numcolors*

Tells **infotopam** how many colors to override. Pixels in the Amiga .info files are assigned an index value rather than a specific color. The standard colors for a 2 bit-plane icon are:

Index 0: Blue (00, 55, AA)
Index 1: White (FF, FF, FF)
Index 2: Black (00, 00, 20)
Index 3: Orange (FF, 8A, 00)

To override the colors, first specify how many colors to override using -*numcolors*, then specify an (*index color*) pair for each color you want to override, where *index* is a value from 0 to 3 and *color* the the new color for that index. Specify *color* as described for the **ppm_parsecolor()** argument .

-selected

Tells **infotopam** to convert the selected (second) icon instead of the normal (first) icon. Each Amiga .info icon file contains two icon images. The first image is the normal, unselected icon, and the second image is the selected icon. By default **infotopam** converts the first icon. You can tell **infotopam** to convert the second icon by using the -*selected* option.

All options can be abbreviated to their shortest unique prefix.

SEE ALSO

pam(1) **pamtopnm**(1) **pamscale**(1)

NOTES

Thanks to the following people on comp.sys.amiga.programmer for tips and pointers on decoding the info file format:

- Ben Hutchings
- Thomas Richter
- Kjetil Svalastog Matheussen
- Anders Melchiorsen
- Dirk Stoecker
- Ronald V.D.

The format of the Amiga .info file is as follows:

DiskObject header 78 bytes
Optional DrawerData header 56 bytes
First icon header 20 bytes
First icon data Varies

Second icon header	20 bytes
Second icon data	Varies

The DiskObject header contains, among other things, the magic number (0xE310), the object width and height (inside the embedded Gadget header), and the version.

Each icon header contains the icon width and height, which can be smaller than the object width and height, and the number of bit-planes.

The icon data has the following format:

BIT-PLANE planes, each with *HEIGHT* rows of (*WIDTH* +15) / 16 * 2 bytes length.

So if you have a 9x3x2 icon, the icon data will look like this:

```
aaaa aaaa a000 0000
aaaa aaaa a000 0000
aaaa aaaa a000 0000
bbbb bbbb b000 0000
bbbb bbbb b000 0000
bbbb bbbb b000 0000
```

where *a* is a bit for the first bit-plane, *b* is a bit for the second bit-plane, and *0* is padding. Thanks again to Ben Hutchings for his very helpful post!

HISTORY

infotopam was new in Netpbm 10.22 (April 2004).

LIMITATIONS

infotopam currently only handles 1 and 2 bit-plane icons.

There is no **pamtoinfo** command, since the .info files contain a lot more than just icon data, and mapping the colors would be difficult.

AUTHOR

Table Of Contents

NAME

jbigtopnm - JBIG to PNM image file converter

SYNOPSIS

jbigtopnm [*options*] [*input-file* [*output-file*]]

DESCRIPTION

This program is part of **Netpbm**(1).

jbigtopnm reads a JBIG bi-level image entity (BIE) from a file or standard input, decompresses it, and outputs a PBM or PGM file. If the input has one plane, or you choose just one plane of it, the output is PBM. Otherwise, the output is PGM.

JBIG is a highly effective lossless compression algorithm for bi-level images (one bit per pixel), which is particularly suitable for scanned document pages.

A JBIG encoded image can be stored in several resolutions in one or several BIEs. All resolution layers except the lowest one are stored efficiently as differences to the next lower resolution layer. You can use options **-x** and **-y** to stop the decompression at a specified maximal output image size. The input file can consist of several concatenated BIEs which contain different increasing resolution layers of the same image.

OPTIONS

-x *number*

Decode only up to the largest resolution layer which is still not more than *number* pixels wide. If no such resolution layer exists, then use the smallest one available.

-y *number*

Decode only up to the largest resolution layer which is still not more than *number* pixels high. If no such resolution layer exists, then use the smallest one available. You can also use options **-x** and **-y** together which selects the largest layer that satisfies both limits.

-b Use binary values instead of Gray code words in order to decode pixel values from multiple bitplanes. This option has effect only if the input has more than one bitplane and you don't select just one of those bitplanes. Note that the decoder has to be used in the same mode as the encoder and cannot determine from the BIE, whether Gray or binary code words were used by the encoder.

-d Diagnose a BIE. With this option, **jbigtopnm** only prints a summary of the header information found in the input file and then exits.

-p *number*

If the input contains multiple bitplanes, then extract only the specified single plane as a PBM file. The first plane has number 0.

STANDARDS

This program implements the JBIG image coding algorithm as specified in ISO/IEC 11544:1993 and ITU-T T.82(1993).

AUTHOR

jbigtopnm is based on the JBIG library by Markus Kuhn, part of his **JBIG-KIT** package . The **jbgtopbm** program is part of the **JBIG-KIT** package.

jbigtopnm is part of the Netpbm package of graphics tools.

SEE ALSO

pnm(1), **pnmtojbig**(1)

LICENSE

If you use **jbigtopnm**, you are using various patents, particularly on its arithmetic encoding method, and in all probability you do not have a license from the patent owners to do so.

Table Of Contents

NAME

jpeg2ktopam - convert JPEG-2000 code stream to PAM/PNM

SYNOPSIS

jpeg2ktopam [**-verbose**] [**-debuglevel=***number*] *filename*

OPTION USAGE

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

jpeg2ktopam converts the named JPEG-2000 code stream file, or Standard Input if no file is named, to a PBM, PGM, PPM, or PAM file on Standard Output.

The JPEG-2000 specification specifies two different formats: JP2 and JPEG-2000 code stream (JPC). JP2 represents a visual image quite specifically, whereas JPC is a more or less arbitrary array of codes. JP2 images are a subset of JPC images. **jpeg2ktopam** converts any JPC image. If the color space identified in the image is grayscale (JAS_IMAGE_CS_GRAY), **jpeg2ktopam** generates a PGM image, unless the image contains only one bit per pixel, in which case **jpeg2ktopam** generates PBM. If the color space is RGB (JAS_IMAGE_CS_RGB), **jpeg2ktopam** generates a PPM image. If the input image is anything else, **jpeg2ktopam** generates a PAM image with no tuple type identified.

In the PGM and PPM cases, **jpeg2ktopam** assumes the intensity values in the input image have the same meaning as for PGM and PPM. One thing that implies is the ITU Rec. 709 color space, which means the assumption is false for JP2 input. JP2 input uses SRGB color encoding. So if you use **jpeg2ktopam** to convert a JP2 image to PPM, it changes the visual image (slightly) -- the colors in the output are not the same as in the input.

In the PAM image, the output samples are numerically identical to the input samples. If the input samples are signed, they are represented in two's complement form in the output (because PAM samples are always unsigned). The PAM plane numbers are identical to the JPC component numbers.

A JPC image has a "precision," which is the number of bits used for each code (in Netpbm lingo, "sample"). Actually, each component has a separate precision. The maxval of a PGM, PPM, or PAM output is the largest number you can represent in the JPC precision of the JPC component with the greatest precision. The samples in all components are scaled to that maxval. So if the red component has a precision of 4 bits and the green component has a precision of 6 bits, the maxval is 63 and the red component codes from the JPC image are multiplied by 63/15 to generate the output samples.

jpeg2ktopam interprets the JPC input with the Jasper JPEG-2000 library . See documentation of the library for details on what **jpeg2ktopam** handles. Note that the Jasper library contains facilities for writing PNM images, but **jpeg2ktopam** does not use those. It uses the Netpbm library instead. Note that the makers of the Jasper library write it "JasPer," but Netpbm documentation follows standard American English typography rules, which don't allow that kind of capitalization.

Use **pamtojpeg2k** to convert in the other direction.

The program **jasper**, which is packaged with the Jasper JPEG-2000 library, also converts between JPEG-2000 and PNM formats. Because it's packaged with the library, it may exploit it better, especially recently added features. However, since it does not use the Netpbm library to read and write the Netpbm formats, it doesn't do as good a job on that side.

OPTIONS

-verbose

This option causes **jpeg2ktopam** to issue informational messages about the conversion process.

-debuglevel=*number*

This option controls debug messages from the Jasper library. **jpeg2ktopam** passes *number* as the debug level to the Jasper JPC decoder.

EXAMPLES

jpeg2ktopam myimg.jpc >myimg.ppm

ABOUT JPEG-2000

See **thepamtojpeg2kmanual**(1) for general information on JPEG-2000 compression and the JPEG-2000 formats.

SEE ALSO

jpctopam(1), **pnmtopeg**(1), **ppm**(1), **pgm**(1), **pbm**(1), **pam**(1),

History

jpeg2ktopam was added to Netpbm in Release 10.12 (November 2002).

Table Of Contents

NAME

jpegtopnm - convert JPEG/JFIF file to PPM or PGM image

SYNOPSIS

jpegtopnm [**-dct** {**int**|**fast**|**float**}] [**-nosmooth**] [**-maxmemory** *N*] [{**-adobe**|**-notadobe**}] [**-comments**] [**-dumpexif**] [**-exif=***filespec*] [**-multiple**] [**-verbose**] [**-tracelevel** *N*] [*filename*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

jpegtopnm converts JFIF images to PPM or PGM images.

By default, **jpegtopnm** expects the input stream to contain one JFIF image and produces one PGM or PPM image as output. It fails if the input stream is empty.

But with the **-multiple** option, **jpegtopnm** reads JFIF images sequentially from the input stream and writes one PPM or PGM image to the output stream for each JFIF input. If the input stream is empty, so is the output.

The input stream is the *filename* you specify or, if you don't specify *filename*, Standard Input. The output stream is Standard Output.

If a JFIF input image is of the grayscale variety, **jpegtopnm** generates a PGM image. Otherwise, it generates a PPM image.

Before Netpbm 10.11 (October 2002), **jpegtopnm** did not have the multiple image stream capability. From 10.11 through 10.22, Netpbm always behaved as if you specified **-multiple**. Starting with Netpbm 10.23 (July 2004), Netpbm's default behavior went back to the pre-10.11 behavior and the new **-multiple** option selected the 10.12 behavior. The reason for the reversion was that there were discovered in the world files that contain JFIF images followed by something other than another JFIF image. The producers of these files expect them to work with any JFIF interpreter because most JFIF interpreters just stop reading the file after the first JFIF image.

jpegtopnm uses the Independent JPEG Group's JPEG library to interpret the input file. See **http://www.ijg.org** for information on the library.

'JFIF' is the correct name for the image format commonly known as 'JPEG.' Strictly speaking, JPEG is a method of compression. The image format using JPEG compression that is by far the most common is JFIF. There is also a subformat of TIFF that uses JPEG compression.

EXIF is an image format that is a subformat of JFIF (to wit, a JFIF file that contains an EXIF header as an APP1 marker). **jpegtopnm** handles EXIF.

JFIF files can have either 8 bits per sample or 12 bits per sample. The 8 bit variety is by far the most common. There are two versions of the IJG JPEG library. One reads only 8 bit files and the other reads only 12 bit files. You must link the appropriate one of these libraries with **jpegtopnm**. Ordinarily, this means the library is in your shared library search path when you run **jpegtopnm**.

jpegtopnm generates output with either one byte or two bytes per sample depending on whether the JFIF input has either 8 bits or 12 bits per sample. You can use **pnmdepth** to reduce a two-byte-per-sample file to a one-byte-per-sample file if you need to.

If the JFIF file uses the CMYK or YCCK color space, the input does not actually contain enough information to know what color each pixel is. To know what color a pixel is, one would have to know the properties of the inks to which the color space refers. **jpegtopnm** interprets the colors using the common transformation which assumes all the inks are simply subtractive and linear.

See the **jpegtopnm**manual**(1)** for information on how images lose quality when you convert to and from JFIF.

OPTIONS

The options are only for advanced users:

-dct int Use integer DCT method (default).

-dct fast

Use fast integer DCT (less accurate).

-dct float

Use floating-point DCT method. The float method is very slightly more accurate than the int method, but is much slower unless your machine has very fast floating-point hardware. Also note that results of the floating-point method may vary slightly across machines, while the integer methods should give the same results everywhere. The fast integer method is much less accurate than the other two.

-nosmooth

Use a faster, lower-quality upsampling routine.

-maxmemory *N*

Set limit on the amount of memory **jpegtopnm** uses in processing large images. Value is in thousands of bytes, or millions of bytes if 'M' is suffixed to the number. For example, **-maxmemory 4m** selects 4000000 bytes. If **jpegtopnm** needs more space, it uses temporary files.

-adobe

-notadobe

There are two variations on the CMYK (and likewise YCCK) color space that may be used in the JFIF input. In the normal one, a zero value for a color components indicates absence of ink. In the other, a zero value means the maximum ink coverage. The latter is used by Adobe Photoshop when it creates a bare JFIF output file (but not when it creates JFIF output as part of Encapsulated Postscript output).

These options tell **jpegtopnm** which version of the CMYK or YCCK color space the image uses. If you specify neither, **jpegtopnm** tries to figure it out on its own. In the present version, it doesn't try very hard at all: It just assumes the Photoshop version, since Photoshop and its emulators seem to be the main source of CMYK and YCCK images. But with experience of use, future versions might be more sophisticated.

If the JFIF image does not indicate that it is CMYK or YCCK, these options have no effect.

If you don't use the right one of these options, the symptom is output that looks like a negative.

-dumpexif

Print the interpreted contents of any Exif header in the input file to the Standard Error file. Similar to the program **jhead** (not part of the Netpbm package).

This option was added in Netpbm 9.19 (September 2001).

-exif=*filespec*

Extract the contents of the EXIF header from the input image and write it to the file *filespec*. *filespec*=**-** means write it to Standard Output. When you write the EXIF header to Standard Output, **jpegtopnm** does not output the converted image (which is what normally would go to Standard Output) at all.

jpegtopnm writes the contents of the EXIF header byte-for-byte, starting with the two byte

length field (which length includes those two bytes).

You can use this file as input to **pnmtojpeg** to insert an identical EXIF header into a new JFIF image.

If there is no EXIF header, **jpegtopnm** writes two bytes of binary zero and nothing else.

An EXIF header takes the form of a JFIF APP1 marker. Only the first such marker within the JFIF header counts.

This option was added in Netpbm 9.19 (September 2001).

-multiple

Read multiple JFIF images sequentially from the input stream. See Description section for details.

This option was new in Netpbm 10.23 (July 2004).

-comments

Print any comments in the input file to the Standard Error file.

-verbose

Print details about the conversion to the Standard Error file.

-tracelevel *n*

Turn on the JPEG library's trace messages to the Standard Error file. A higher value of *n* gets more trace information. **-verbose** implies a trace level of at least 1.

EXAMPLES

This example converts the color JFIF file foo.jpg to a PPM file named foo.ppm:

```
jpegtopnm foo.jpg >foo.ppm
```

HINTS

You can use **pnmquant** to color quantize the result, i.e. to reduce the number of distinct colors in the image. In fact, you may have to if you want to convert the PPM file to certain other formats. **ppmdither** Does a more sophisticated quantization.

Use **pamscale** to change the dimensions of the resulting image.

Use **ppmtopgm** to convert a color JFIF file to a grayscale PGM file.

You can easily use these converters together. E.g.:

```
jpegtopnm foo.jpg | ppmtopgm | pamscale .25 >foo.pgm
```

-dct fast and/or **-nosmooth** gain speed at a small sacrifice in quality.

If you are fortunate enough to have very fast floating point hardware, **-dct float** may be even faster than **-dct fast**. But on most machines **-dct float** is slower than **-dct int**; in this case it is not worth using, because its theoretical accuracy advantage is too small to be significant in practice.

Another program, **djpeg**, is similar. **djpeg** is maintained by the Independent JPEG Group and packaged with the JPEG library which **jpegtopnm** uses for all its JPEG work. Because of that, you may expect it to exploit more current JPEG features. Also, since you have to have the library to run **jpegtopnm**, but not vice versa, **cjpeg** may be more commonly available.

On the other hand, **djpeg** does not use the NetPBM libraries to generate its output, as all the NetPBM tools such as **jpegtopnm** do. This means it is less likely to be consistent with all the other programs that deal with the NetPBM formats. Also, the command syntax of **jpegtopnm** is consistent with that of the other Netpbm tools, unlike **djpeg**.

ENVIRONMENT

JPEGMEM

If this environment variable is set, its value is the default memory limit. The value is specified as described for the **-maxmemory** option. An explicit **-maxmemory** option overrides any **JPEGMEM**.

SEE ALSO

ppm(1), **pgm**(1), **pnmtojpeg**(1), **pnmquant**(1), **pamscale**(1), **ppmtopgm**(1), **ppmdither**(1), **pnmdepth**(1),

djpeg man page, **cjpeg** man page, **jpegtran** man page, **rdjpgcom** man page, **wrjpgcom** man page, **jhead** man page

Wallace, Gregory K. 'The JPEG Still Picture Compression Standard', Communications of the ACM, April 1991 (vol. 34, no. 4), pp. 30-44.

LIMITATIONS

Arithmetic coding is not offered for legal reasons. The program could be much faster.

AUTHOR

jpegtopnm and this manual were derived in large part from **djpeg**, by the Independent JPEG Group. The program is otherwise by Bryan Henderson on March 19, 2000.

Table Of Contents

NAME

leaftoppm - convert Interleaf image format to PPM image

SYNOPSIS

leaftoppm [*leaffile*]

DESCRIPTION

This program is part of **Netpbm**(1).

leaftoppm reads a PPM image as input and generates an Interleaf image file as output.

Interleaf is a now-defunct (actually purchased ca. 2000 by BroadVision) technical publishing software company.

SEE ALSO

ppm(1)

AUTHOR

Table Of Contents

NAME

lispmtopgm - convert a Lisp Machine bitmap file to PGM

SYNOPSIS

lispmtopgm [*lispmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

listpmfile reads a Lisp Machine bitmap as input and produces a PGM image as output.

This is the file format written by the tv:write-bit-array-file function on TI Explorer and Symbolics lisp machines.

Multi-plane bitmaps on lisp machines are color; but the Lispm image file format does not include a color map, so we must treat it as a monochrome instead and produce PGM. This is unfortunate.

SEE ALSO

pgmtolispm(1), **pgm**(1)

LIMITATIONS

The Lispm bitmap file format is a bit quirky; Usually the image in the file has its width rounded up to the next higher multiple of 32, but not always. If the width is not a multiple of 32, we don't deal with it properly, but because of the Lispm microcode, such arrays are probably not image data anyway.

Also, the Lispm code for saving bitmaps has a bug, in that if you are writing a bitmap which is not mod32 across, the file may be up to 7 bits too short! They round down instead of up, and we don't handle this bug gracefully.

AUTHOR

Table Of Contents

NAME

macptopbm - convert a MacPaint file into a PBM image

SYNOPSIS

macptopbm [**-extraskip** *N*] [*macpfile*]

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

macptopbmreads a MacPaint file as input and produces a PBM image as output.

OPTIONS

-extraskip

This flag is to get around a problem with some methods of transferring files from the Mac world to the Unix world. Most of these methods leave the Mac files alone, but a few of them add the 'finderinfo' data onto the front of the Unix file. This means an extra 128 bytes to skip over when reading the file. The symptom to watch for is that the resulting PBM file looks shifted to one side. If you get this, try **-extraskip** 128, and if that still doesn't look right try another value.

SEE ALSO

picttoppm(1), **pbmtomacp**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

mdatopbm - convert a Microdesign .mda or .mdp file into a PBM image

SYNOPSIS

mdatopbm [**-a**] [**-d**] [**-i**] [**--**] [*mdafile*]

DESCRIPTION

This program is part of **Netpbm**(1).

mdatopbm reads a MicroDesign file as input and Produces a PBM image as output.

If you don't specify *mdafile*, **mdatopbm** takes its input from Standard Input.

OPTIONS

-a Generate Plain PBM output instead of Raw PBM

-d Double the height of the output file, to compensate for the aspect ratio used in MicroDesign files.

-i Invert the colors used.

-- End of options (use this if the filename starts with '-')

SEE ALSO

pbmtomda(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

mgrtopbm - convert a MGR bitmap into a PBM image

SYNOPSIS

mgrtopbm [*mgrfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

mgrtopbm reads a MGR bitmap as input and produces a PBM image as output.

SEE ALSO

pbmtomgr(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

MRF - monochrome recursive format (compressed bitmaps)

DESCRIPTION

This program is part of **Netpbm**(1).

MRF is a compressed format for bilevel (1-bit mono) images. It achieves better compression for some such images than either GIF or PNG. (It's also very easy to implement (about the same difficulty as RLE, I'd say) and an MRF reader needs no tables/buffers, which may make it useful for tiny machines).

In case the above hasn't made it sufficiently clear, I'll make this next point explicitly: *MRF cannot represent color at all.* Nor can it represent grayscale. It's a specifically mono format. (If you want to compress a color or grayscale image, my advice is to use JPEG2000).

First, here's what goes where in an MRF file. I'll explain how the compression works afterward.

Offset	Description
0	magic number - 'MRF1' (in ASCII)
4	width (32-bit, MSB first (i.e. big-endian))
8	height (same)
12	reserved (single byte, must be zero)
13	compressed data

Note that there is no end-of-file marker in the file itself - the compressed data carries on right up to EOF.

The way the picture is compressed is essentially very simple, but as they say, the devil is in the detail. So don't be put off if it sounds confusing.

The image is treated as a number of 64x64 squares, forming a grid large enough to encompass it. (If an image is (say) 129x65, it'll be treated in the same way as a 192x128 one. On decompression, the extra area which was encoded (the contents of this area is undefined) should be ignored.) Each of these squares in turn (in left-to-right, top-to-bottom order) is recursively subdivided until the smallest completely black or white squares are found. Some pseudocode (eek!) for the recursive subdivision routine should make things clearer:

```
if square size > 1x1 and square is all one color, output 1 bit
if whole square is black, output a 0 bit and return
if whole square is white, output a 1 bit and return
output a 0 bit
divide the square into four quarters, calling routine for
each in this order: top-left, top-right, bottom-left, bottom-right
```

(Note that the 'output a 0 bit' stage is not reached for squares of size 1x1, which is what stops it recursing infinitely. I mention this as it may not be immediately obvious.)

The whole of the compressed data is made up of the bits output by the above routine. The bits are packed into bytes MSB first, so for example outputting the bits 1,0,0,0,0,0,0,0 would result in a 80h byte being output. Any 'unused' bits in the last byte output are undefined; these are effectively after EOF and their value is unimportant.

If writing that sounds too much like hard work :-), you could always adapt **pbmtomrf** and/or **mrftopbm**. That's the main reason their source code is public domain, after all.

Above, I said the contents of any extra area encoded (when a bitmap smaller than the grid of squares is compressed) is undefined. This is deliberate so that the MRF compressor can make these unseen areas anything it wants so as to maximize compression, rather than simply leaving it blank. **pbmtomrf** does a little in this respect but could definitely be improved upon.

mrftopbm's **-1** option causes it to include the edges, if any, in the output PBM. This may help when debugging a compressor's edge optimization.

Note that the "F" in the name "MRF" comes from "format," which is redundant because it is the name of a format. That sort of makes "MRF format" sound as stupid as "PIN number," but it's not really that bad.

SEE ALSO

mrftopbm(1), **pbmtomrf**(1)

AUTHOR

Russell Marks.

Table Of Contents

NAME

mrftopbm - convert an MRF image to PBM format

SYNOPSIS

mrftopbm [**-a**] [*input.mrf*]

DESCRIPTION

This program is part of **Netpbm**(1).

mrftopbm converts an MRF image to PBM format.

mrftopbm takes the MRF image from the file named by the *input.mrf* argument, or Standard Input if you don't specify *input.mrf*. The output goes to Standard Output.

For more information about mrf, see **the**MRF specification (1).

OPTIONS

-a causes **mrftopbm** to include the edges, if any, in the output PBM. This may help when debugging a compressor's edge optimization.

AUTHOR

Russell Marks.

SEE ALSO

pbmtomrf(1), **pbm**(1), **mrf**(1)

Table Of Contents

NAME

mtvtoppm - convert output from an MTV or PRT ray tracer into a PPM

SYNOPSIS

mtvtoppm [*mtvfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

mtvtoppm reads an input file from Mark VanDeWettering's MTV ray tracer and produces a PPM image as output.

The PRT raytracer also produces this format.

SEE ALSO

ppm(1)

AUTHOR

Table Of Contents

NAME

neotoppm - convert an Atari Neochrome .neo into a PPM image

SYNOPSIS

neotoppm [*neofile*]

DESCRIPTION

This program is part of **Netpbm**(1).

neotoppm reads an Atari Neochrome .neo file as input and produces a portable pixmap as output.

SEE ALSO

ppmtoneo(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

netpbm − netpbm library overview

Overview Of Netpbm

Netpbm is a package of graphics programs and a programming library.

There are over 220 separate programs in the package, most of which have 'pbm', 'pgm', 'ppm', 'pam', or 'pnm' in their names. For example, **pamscale**(1)**and giftopnm**(1).

For example, you might use **pamscale** to shrink an image by 10%. Or use **pamcomp** to overlay one image on top of another. Or use **pbmtext** to create an image of text. Or reduce the number of colors in an image with **pnmquant**.

Netpbm is an open source software package, distributed via the Sourceforge **netpbm** project .

The Netpbm Programs

The Netpbm programs are generally useful run by a person from a command shell, but are also designed to be used by programs. A common characteristic of Netpbm programs is that they are simple, fundamental building blocks. They are most powerful when stacked in pipelines. Netpbm programs do not use graphical user interfaces (in fact, none of them display graphics at all, except for a very simple Linux Svgalib displayer) and do not seek input from a user.

Each of these programs has its own manual, as linked in the directory below.

The Netpbm programs can read and write files greater than 2 GiB wherever the underlying system can. There may be exceptions where the programs use external libraries (The JPEG library, etc.) to access files and the external library does not have large file capability. Before Netpbm 10.15 (April 2003), no Netpbm program could read a file that large.

Common Options

There are a few options that are present on all programs that are based on the Netpbm library, including virtually all Netpbm programs. These are not mentioned in the individual manuals for the programs.

You can use two hyphens instead of one on these options if you like.

-quiet Suppress all informational messages that would otherwise be issued to Standard Error. (To be precise, this only works to the extent that the program in question implements the Netpbm convention of issuing all informational messages via the **pm_message()** service of the Netpbm library).

-version

Instead of doing anything else, report the version of the **libnetpbm** library linked with the program (it may have been linked statically into the program, or dynamically linked at run time). Normally, the Netpbm programs and the library are installed at the same time, so this tells you the version of the program and all the other Netpbm files it uses as well.

-plain If the program generates an image in Netpbm format, generate it in the "plain" (aka "ascii") version of the format, as opposed to the "raw" (aka "binary") version.

This option was introduced in Netpbm 10.10 (October 2002).

Directory

Here is a complete list of all the Netpbm programs (with links to their manuals):

Netpbmprogram**directory**(1)

How To Use The Programs

As a collection of primitive tools, the power of Netpbm is multiplied by the power of all the other unix tools you can use with them. These notes remind you of some of the more useful ways to do this. Often, when people want to add high level functions to the Netpbm tools, they have overlooked some existing tool that, in combination with Netpbm, already does it.

Often, you need to apply some conversion or edit to a whole bunch of files.

As a rule, Netpbm programs take one input file and produce one output file, usually on Standard Output. This is for flexibility, since you so often have to pipeline many tools together.

Here is an example of a shell command to convert all your of PNG files (named *.png) to JPEG files named *.jpg:
for i in *.png; do pngtopnm $i | ppmtojpeg >‘basename $i .png‘.jpg; done

Or you might just generate a stream of individual shell commands, one per file, with awk or perl. Here’s how to brighten 30 YUV images that make up one second of a movie, keeping the images in the same files:

```
ls *.yuv
  | perl -ne 'chomp;
  print yuvtoppm $_ | ppmbrighten -v 100 | ppmtoyuv >tmp$$.yuv;
  mv tmp$$.yuv $_
  '
  | sh
```

The tools **find** (with the **-exec** option) and **xargs** are also useful for simple manipulation of groups of files.

Some shells’ ’process substitution’ facility can help where a non-Netpbm program expects you to identify a disk file for input and you want it to use the result of a Netpbm manipulation. Say the hypothetical program **printcmyk** takes the filename of a Tiff CMYK file as input and what you have is a PNG file **abc.png**.

Try:
printcmyk <({ pngtopnm abc.png | pnmtotiffcmyk ; })

It works in the other direction too, if you have a program that makes you name its output file and you want the output to go through a Netpbm tool.

The Netpbm Formats

All of the programs work with a set of graphics formats called the ’netpbm’ formats. Specifically, these formats are **pbm**(1), **pgm**(1), **ppm**(1), and **pam**(1).

The first three of these are sometimes known generically as ’pnm’.

Many of the Netpbm programs convert from a Netpbm format to another format or vice versa. This is so you can use the Netpbm programs to work on graphics of any format. It is also common to use a combination of Netpbm programs to convert from one non-Netpbm format to another non-Netpbm format. Netpbm has converters for about 100 graphics formats, and as a package Netpbm lets you do more graphics format conversions than any other computer graphics facility.

The Netpbm formats are all raster formats, i.e. they describe an image as a matrix of rows and columns of pixels. In the PBM format, the pixels are black and white. In the PGM format, pixels are shades of gray. In the PPM format, the pixels are in full color. The PAM format is more sophisticated. A replacement for all three of the other formats, it can represent matrices of general data including but not limited to black and white, grayscale, and color images.

Programs designed to work with PBM images have ’pbm’ in their names. Programs designed to work with PGM, PPM, and PAM images similarly have ’pgm’, ’ppm’, and ’pam’ in their names.

All Netpbm programs designed to read PGM images see PBM images as if they were PGM too. All Netpbm programs designed to read PPM images see PGM and PBM images as if they were PPM. See

Implied Format Conversion .

Programs that have 'pnm' in their names read PBM, PGM, and PPM but unlike 'ppm' programs, they distinguish between them and their function depends on the format. For example, **pnmtopng**(1)**creates**a**black**and**white**PNG output image if its input is PBM or PGM, but a color PNG output image if its input is PPM. And **pnmrotate** produces an output image of the same format as the input. A hypothetical **ppmrotate** program would also read all three PNM input formats, but would see them all as PPM and would always generate PPM output.

Programs that have "pam" in their names read all the Netpbm formats: PBM, PGM, PPM, and PAM. They sometimes treat them all as if they are PAM, using an implied conversion, but often they recognize the individual formats and behave accordingly, like a "pnm" program does. See Implied Format Conversion .

If it seems wasteful to you to have three separate PNM formats, be aware that there is a historical reason for it. In the beginning, there were only PBMs. PGMs came later, and then PPMs. Much later came PAM, which realizes the possibility of having just one aggregate format.

The formats are described in the specifications of **pbm**(1), **pgm**(1), **ppm**(1), and **pam**(1).

Implied Format Conversion

A program that uses the PGM library subroutines to read an image can read a PBM image as well as a PGM image. The program sees the PBM image as if it were the equivalent PGM image, with a maxval of 255. **note:** This sometimes confuses people who are looking at the formats at a lower layer than they ought to be because a zero value in a PBM raster means white, while a zero value in a PGM raster means black.

A program that uses the PPM library subroutines to read an image can read a PGM image as well as a PPM image and a PBM image as well as a PGM image. The program sees the PBM or PGM image as if it were the equivalent PPM image, with a maxval of 255 in the PBM case and the same maxval as the PGM in the PGM case.

A program that uses the PAM library subroutines to read an image can read a PBM, PGM, or PPM image as well as a PAM image. The program sees a PBM image as if it were the equivalent PAM image with tuple type **BLACKANDWHITE**. It sees a PGM image as if it were the equivalent PAM image with tuple type **GRAYSCALE**. It sees a PPM image as if it were the equivalent PAM image with tuple type **RGB**. But the program actually can see deeper if it wants to. It can tell exactly which format the input was and may respond accordingly. For example, a PAM program typically produces output in the same format as its input.

Netpbm and Transparency

In many graphics formats, there's a means of indicating that certain parts of the image are wholly or partially transparent, meaning that if it were displayed 'over' another image, the other image would show through there. Netpbm formats deliberately omit that capability, since their purpose is to be extremely simple.

In Netpbm, you handle transparency via a transparency mask in a separate (slightly redefined) PGM image. In this pseudo-PGM, what would normally be a pixel's intensity is instead an opaqueness value. See **pgm**(1). **pamcomp**(1)**is**an**example**of**a**program**that**uses a PGM transparency mask.

Another means of representing transparency information has recently developed in Netpbm, using PAM images. In spite of the argument given above that Netpbm formats should be too simple to have transparency information built in, it turns out to be extremely inconvenient to have to carry the transparency information around separately. This is primarily because Unix shells don't provide easy ways to have networks of pipelines. You get one input and one output from each program in a pipeline. So you'd like to have both the color information and the transparency information for an image in the same pipe at the same time.

For that reason, some new (and recently renovated) Netpbm programs recognize and generate a PAM image with tuple type RGB_ALPHA or GRAYSCALE_ALPHA, which contains a plane for the transparency information. See **the**PAM**specification**(1).

The Netpbm Library

The Netpbm programming library, **libnetpbm**(1),**makes**it**easy**to**write**programs that manipulate graphic images. Its main function is to read and write files in the Netpbm formats, and because the Netpbm package contains converters for all the popular graphics formats, if your program reads and writes the Netpbm formats, you can use it with any formats.

But the library also contain some utility functions, such as character drawing and RGB/YCrCb conversion.

The library has the conventional C linkage. Virtually all programs in the Netpbm package are based on the Netpbm library.

netpbm-config

In a standard installation of Netpbm, there is a program named **netpbm-config** in the regular program search path. We don't consider this a Netpbm program -- it's just an ancillary part of a Netpbm installation. This program tells you information about the Netpbm installation, and is intended to be run by other programs that interface with Netpbm. In fact, **netpbm-config** is really a configuration file, like those you typically see in the */etc/* directory of a Unix system.

Example:

```
$netpbm-config --datadir
/usr/local/netpbm/data
```

If you write a program that needs to access a Netpbm data file, it can use such a shell command to find out where the Netpbm data files are.

netpbm-config is the only file that must be installed in a standard directory (it must be in a directory that is in the default program search path). You can use **netpbm-config** as a bootstrap to find all the other Netpbm files.

There is no detailed documentation of **netpbm-config**. If you're in a position to use it, you should have no trouble reading the file itself to figure out how to use it.

Other Graphics Software

Netpbm contains primitive building blocks. It certainly is not a complete graphics software library.

Graphics Viewers

The first thing you will want to make use of any of these tools is a viewer. (On GNU/Linux, you can use **ppmsvgalib** in a pinch, but it is pretty limiting). **zgv** is a good full service viewer to use on a GNU/Linux system with the SVGALIB graphics display driver library. You can find **zgv** at **ftp://ftp.ibiblio.org/pub/Linux/apps/graphics/viewers/svga .**

zgv even has a feature in it wherein you can visually crop an image and write an output file of the cropped image using **pamcut**(1).

See the **-s** option to **zgv**.

For the X inclined, there is also **xzgv**.

xloadimage and its extension **xli** are also common ways to display a graphic image in X.

gqview is a more modern X-based image viewer.

qiv is a small, very fast viewer for X.

To play mpeg movies, such as produced by **ppmtompeg**, try **xine .**

See **ftp://metalab.unc.edu/pub/Linux/apps/graphics/viewers/X** .

Visual Graphics Software

Visual graphics software is modern point-and-click software that displays an image and lets you work on it and see the results as you go. This is fundamentally different from what Netpbm programs do.

ImageMagick is like a visual version of Netpbm. Using the X/Window system on Unix, you can do basic editing of images and lots of format conversions. The package does include at least some non-visual tools. **convert**, **mogrify**, **montage**, and **animate** are popular programs from the **ImageMagick** package. **ImageMagick** runs on Unix, Windows, Windows NT, Macintosh, and VMS.

xv is a very old and very popular simple image editor in the Unix world. It does not have much in the way of current support, or maintenance, though.

The Gimp is a visual image editor for Unix and X, in the same category as the more famous, less capable, and much more expensive Adobe Photoshop, etc. for Windows. See **http://www.gimp.org** .

Electric Eyes, **kuickshow**, and **gthumb** are also visual editors for the X/Window system, and **KView** and **gwenview** are specifically for KDE.

Programming Tools

If you're writing a program in C to draw and manipulate images, check out gd . Netpbm contains a C library for drawing images, but it is probably not as capable or documented as **gd**. You can easily run any Netpbm program from a C program with the **pm_system** function from the Netpbm programming library, but that is less efficient than **gd** functions that do the same thing.

Ilib is a C subroutine library with functions for adding text to an image (as you might do at a higher level with **pbmtext**, **pamcomp**, etc.). It works with Netpbm input and output. Find it at **http://www.radix.net/~cknudsen/Ilib** . Netpbm also includes character drawing functions in the **libnetpbm**(1)**library,**but**they**do**not**have**as** fancy font capabilities (see **ppmlabel**(1) for an example of use of the Netpbm character drawing functions).

GD is a library of graphics routines that is part of PHP. It has a subset of Netpbm's functions and has been found to resize images more slowly and with less quality.

Tools For Specific Graphics Formats

To create an animated GIF, or extract a frame from one, use **gifsicle**. **gifsicle** converts between animated GIF and still GIF, and you can use **ppmtogif** and **giftopnm** to connect up to all the Netpbm utilities. See **http://www.lcdf.org/gifsicle** .

To convert an image of text to text (optical character recongition - OCR), use **gocr** (think of it as an inverse of **pbmtext**). See **http://altmark.nat.uni-magdeburg.de/~jschulen/ocr/** .

http://schaik.com/pngsuite contains a PNG test suite -- a whole bunch of PNG images exploiting the various features of the PNG format.

Another version of Netpbm's **pnmtopng**/**pngtopnm** is at **http://www.schaik.com/png/pnmtopng.html**(1).

The version in Netpbm was actually based on that package a long time ago, and you can expect to find better exploitation of the PNG format, especially recent enhancements, in that package. It may be a little less consistent with the Netpbm project and less exploitive of recent Netpbm format enhancements, though.

pngwriter is a C++ library for creating PNG images. With it, you plot an image pixel by pixel. You can also render text with the FreeType2 library.

jpegtran Does some of the same transformations as Netpbm is famous for, but does them specifically on JPEG files and does them without loss of information. By contrast, if you were to use Netpbm, you would first decompress the JPEG image to Netpbm format, then transform the image, then compress it back to JPEG format. In that recompression, you lose a little image information because JPEG is a lossy compression. Of course, only a few kinds of lossless transformation are possible. **jpegtran** comes with the Independent Jpeg Group's (http://www.ijg.org) JPEG library.

Some tools to deal with EXIF files (see also Netpbm's **jpegtopnm**(1)**and pnmtojpeg**(1)):

To dump (interpret) an EXIF header: Exifdump ((http://topo.math.u-psud.fr/~bousch/exifdump.py)) or Jhead (http://www.sentex.net/~mwandel/jhead.)

A Python EXIF library and dumper: http://pyexif.sourceforge.net.

Here's some software to work with IOCA (Image Object Content Architecture): ImageToolbox

($2500, demo available). This can convert from TIFF -> IOCA and back again. **Ameri-Imager**(1) ($40 Windows only).

pnm2ppa converts to HP's 'Winprinter' format (for HP 710, 720, 820, 1000, etc). It is a superset of Netpbm's **pbmtoppa** and handles, notably, color. However, it is more of a printer driver than a Netpbm-style primitive graphics building block. See **http://sourceforge.net/project/?group_id=1322** .

Document/Graphics Software

There is a large class of software that does document processing, and that is somewhat related to graphics because documents contain graphics and a page of a document is for many purposes a graphic image. Because of this slight intersection with graphics, I cover document processing software here briefly, but it is for the most part beyond the scope of this document.

First, we look at where Netpbm meets document processing. **pstopnm** converts from Postscript and PDF to PNM. It effectively renders the document into images of printed pages. **pstopnm** is nothing but a convenient wrapper for Ghostscript , and in particular Netpbm-format device drivers that are part of it. **pnmtops** and **pbmtoepsi** convert a PNM image to a Postscript program for printing the image. But to really use PDF and Postscript files, you generally need more complex document processing software.

Adobe invented Postscript and PDF and products from Adobe are for many purposes the quintessential Postscript and PDF tools.

Adobe's free Acrobat Reader displays PDF and converts to Postscript. The Acrobat Reader for unix has a program name of 'acroread' and the -toPostScript option (also see the -level2 option) is useful.

Other software from Adobe, available for purchase, interprets and creates Postscript and PDF files. 'Distill' is a program that converts Postscript to PDF.

xpdf also reads PDF files.

GSview, ghostview, gv, ggv, and kghostview are some other viewers for Postscript and PDF files.

The program **ps2pdf**, part of Ghostscript, converts from Postscript to PDF.

Two packages that produce more kinds of Encapsulated Postscript than the Netpbm programs, including compressed kinds, are bmeps and imgtops .

dvips converts from DVI format to Postscript. DVI is the format that Tex produces. Netpbm can convert from Postscript to PNM. Thus, you can use these in combination to work with Tex/Latex documents graphically.

wvware converts a Microsoft Word document (.doc file) to various other formats. While the web page doesn't seem to mention it, it reportedly can extract an embedded image in a Word document as a PNG.

Latex2html converts Latex document source to HTML document source. Part of that involves graphics, and Latex2html uses Netpbm tools for some of that. But Latex2html through its history has had some rather esoteric codependencies with Netpbm. Older Latex2html doesn't work with current Netpbm. Latex2html-99.2beta8 works, though.

Other

The **file** program looks at a file and tells you what kind of file it is. It recognizes most of the graphics formats with which Netpbm deals, so it is pretty handy for graphics work. Netpbm's **anytopnm**(1)**program**depends**onfile.** See **ftp://ftp.astron.com/pub/file** .

The Utah Raster Toolkit serves a lot of the same purpose as Netpbm, but without the emphasis on format conversions. This package is based on the RLE format, which you can convert to and from the Netpbm formats. **http://www.cs.utah.edu/gdc/projects/urt.html**(1) gives some information on the Utah Raster Toolkit, but does not tell where to get it.

Ivtools is a suite of free X Windows drawing editors for Postscript, Tex, and web graphics production, as well as an embeddable and extendable vector graphic shell. It uses the Netpbm facilities. See **http://www.ivtools.org** .

The program **morph** morphs one image into another. It uses Targa format images, but you can use **tgatoppm** and **ppmtotga** to deal with that format. You have to use the graphical (X/Tk) Xmorph to

create the mesh files that you must feed to **morph**. **morph** is part of the Xmorph package. See **http://www.colorado-research.com/~gourlay/software/Graphics/Xmorph** .

Other Graphics Formats

People never seem to tire of inventing new graphics formats, often completely redundant with pre-existing ones. Netpbm cannot keep up with them. Here is a list of a few that we know Netpbm does *not* handle (yet).

Various commercial Windows software handles dozens of formats that Netpbm does not, especially formats typically used with Windows programs. ImageMagick is probably the most used free image format converter and it also handles lots of formats Netpbm does not.

- VRML (Virtual Reality Modelling Language)
- CAL (originated by US Department Of Defense, favored by architects). **http://www.landfield.com/faqs/graphics/fileformats-faq/part3/section-24.html**(1)
- array formats dx, general, netcdf, CDF, hdf, cm
- CGM+
- Windows Meta File (.WMF). Libwmf converts from WMF to things like Latex, PDF, PNG. Some of these can be input to Netpbm.
- Microsoft Word, RTF. Microsoft keeps a proprietary hold on these formats. Any software you see that can handle them is likely to cost money.
- DXF (AutoCAD)
- IOCA (Image Object Content Architecture) The specification of this format is documented by IBM:
  Data Stream and Object Architectures: Image Object Content Architecture Reference . See above for software that processes this format.
- SVG. Find out about this vector graphics format and software to use with it at this Worldwide Web Consortium web page .
- OpenEXR is an HDR format (like **PFM**(1)). See http://www.openexr.org/about.html (1).
- Xv Visual Schnauzer thumbnail image. This is a rather antiquated format used by the Xv program. In Netpbm circles, it is best known for the fact that it is very similar to Netpbm formats and uses the same signature ('P7') as PAM because it was developed as sort of a fork of the Netpbm format specifications.

History

Netpbm has a long history, starting with Jef Poskanzer's Pbmplus package in 1988. The file *HISTORY* in the Netpbm source code contains a historical overview as well as a detailed history release by release.

Author

Netpbm is based on the Pbmplus package by Jef Poskanzer, first distributed in 1988 and maintained by him until 1991. But the package contains work by countless other authors, added since Jef's original work. In fact, the name is derived from the fact that the work was contributed by people all over the world via the Internet, when such collaboration was still novel enough to merit naming the package after it.

Bryan Henderson has been maintaining Netpbm since 1999. In addition to packaging work by others, Bryan has also written a significant amount of new material for the package.

Table Of Contents

NAME

palmtopnm - convert a Palm Bitmap to a PNM image

SYNOPSIS

palmtopnm

[**-verbose**]

[**-rendition** *N*]

[**-showhist**] [*palmfile*] **palmtopnm**

-transparent

[**-verbose**]

[*palmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

palmtopnm reads a Palm Bitmap as input, from Standard Input or *palmfile* and produces a PPM image as output.

Alternatively (when you specify **-transparent**), **palmtopnm** writes the value of the transparent color in the Palm Bitmap to Standard Output.

Palmtopnm can convert Palm Bitmaps with the following features. This does not mean that it doesn't handle other features. These are just the ones we found worth mentioning.

- Version 0
- Version 1
- Version 2
- Version 3 (new in Netpbm 10.27 (March 2005))
- Scanline compression
- RLE compression
- Packbits compression (new in Netpbm 10.27 (March 2005))

OPTIONS

-verbose

Display various interesting information about the input file and process.

-transparent

If the Palm Bitmap has a transparent color set, **palmtopnm** writes the value for that color to Standard Output in the form #RRGGBB, where RR, GG, and BB are two-digit hexadecimal numbers indicating a value in the range 0 through 255. If no transparent color is set in the Bitmap, **palmtopnm** writes nothing. **palmtopnm** does not generate any output image when you specify **-transparent**.

-rendition N

Palm Bitmaps may contain several different renditions of the same image, with different depths. By default, **palmtopnm** operates on the first rendition (rendition number 1) in the

image. This switch allows you to operate on a different rendition. The value must be between 1 and the number of renditions in the image, inclusive.

-showhist

This option causes **palmtopnm** to write a histogram of colors in the input file to Standard Error.

SEE ALSO

pnmtopalm(1), **pnm**(1), **PalmOS** Reference (1), PalmOS Companion .

LIMITATIONS

You cannot generate an alpha mask if the Palm Bitmap has a transparent color. However, you can still do this with **ppmcolormask** with a Netpbm pipe similar to:

palmtopnm bitmap.palm | ppmcolormask 'palmtopnm -transparent bitmap.palm'

HISTORY

Before Netpbm 10.23 (July 2004), there was a **-forceplain** option. But that had been redundant for a long time, since the Netpbm common option **-plain** does the same thing.

AUTHORS

This program was originally written as Tbmptopnm.c, by Ian Goldberg. It was heavily modified by Bill Janssen to add color, compression, and transparency function.

Table Of Contents

NAME

pamarith - perform arithmetic on two Netpbm images

SYNOPSIS

pamarith -add | **-subtract** | **-multiply** | **-difference** | **-minimum** | **-maximum** | **-mean** | **-compare** | **-and** | **-or** | **-nand** | **-nor** | **-xor** | **-shiftleft** | **-shiftright** *pamfile1 pamfile2*

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamarith reads two PBM, PGM, PPM, or PAM images as input. It performs the specified binary arithmetic operation on their sample values and produces an output of a format which is the more general of the two input formats. The two input images must be of the same width and height. The arithmetic is performed on each pair of identically located tuples to generate the identically located tuple of the output.

For the purpose of the calculation, it assumes any PBM, PGM, or PPM input image is the equivalent PAM image of tuple type **BLACKANDWHITE**, **GRAYSCALE**, or **RGB**, respectively, and if it produces a PBM, PGM, or PPM output, produces the equivalent of the PAM image which is the result of the calculation.

The first *pamfile* argument identifies the 'left' argument image; the second *pamfile* argument identifies the 'right' one.

If the output is PAM, the tuple type is the same as the tuple type of the left input image.

pamarith performs the arithmetic on each pair of identically located tuples in the two input images.

The arithmetic operation is in all cases fundamentally a function from two integers to an integer. The operation is performed on two tuples as follows. The two input images must have the same depth, or one of them must have depth one. **pamarith** fails if one of these is not the case.

If they have the same depth, **pamarith** simply carries out the arithmetic one sample at a time. I.e. if at a particular position the left input image contains the tuple (s1,s2,...,sN) and the right input image contains the tuple (t1,t2,...tN), and the function is f, then the output image contains the tuple (f(s1,t1),f(s2,t2),...,f(sN,tN)).

If one of the images has depth 1, the arithmetic is performed between the one sample in that image and each of the samples in the other. I.e. if at a particular position the left input image contains the tuple (s) and the right input image contains the tuple (t1,t2,...tN), and the function is f, then the output image contains the tuple (f(s,t1),f(s,t2),...,f(s,tN)).

Maxval

The meanings of the samples with respect to the maxval varies according to the function you select.

In PAM images in general, the most usual meaning of a sample (the one that applies when a PAM image represents a visual image), is that it represents a fraction of some maximum. The maxval of the image corresponds to some maximum value (in the case of a visual image, it corresponds to 'full intensity.'), and a sample value divided by the maxval gives the fraction.

For **pamarith**, this interpretation applies to the regular arithmetic functions: **-add**, **-subtract**, **-multiply**, **-difference**, **-minimum**, **-maximum**, **-mean**, and **-compare**. For those, you should think of the arguments and result as numbers in the range [0,1). For example, if the maxval of the left argument image is 100 and the maxval of the right argument image is 200 and the maxval of the output image is 200, and the left sample value in an **-add** calculation is 50 and the right sample is 60, the actual calculation is 50/100 + 60/200 = 160/200, and the output sample value is 160.

For these functions, **pamarith** makes the output image's maxval the maximum of the two input maxvals, except with **-compare**, where **pamarith** uses an output maxval of 2.

If the result of a calculation falls outside the range [0, 1), **pamarith** clips it -- i.e. considers it to be zero or 1-.

In many cases, where both your input maxvals are the same, you can just think of the operation as taking place between the sample values directly, with no consideration of the maxval except for the clipping. E.g. an **-add** of sample value 5 to sample value 8 yields sample value 13.

But with **-multiply**, this doesn't work. Say your two input images have maxval 255, which means the output image also has maxval 255. Consider a location in the image where the input sample values are 5 and 10. You might think the multiplicative product of those would yield 50 in the output. But **pamarith** carries out the arithmetic on the fractions 5/255 and 10/255. It multiplies those together and then rescales to the output maxval, giving a sample value in the output PAM of 50/255 rounded to the nearest integer: 0.

With the bit string operations, the maxval has a whole different meaning. The operations in question are: **-and**, **-or**, **-nand**, **-nor**, **-xor**, and **-shiftleft**, **-shiftright**.

With these, each sample value in one or both input images, and in the output image, represents a bit string, not a number. The maxval tells how wide the bit string is. The maxval must be a full binary count (a power of two minus one, such as 0xff) and the number of ones in it is the width of the bit string. For the dyadic bit string operations (that's everything but the shift functions), the maxvals of the input images must be the same and **pamarith** makes the maxval of the output image the same.

For the bit shift operations, the output maxval is the same as the left input maxval. The right input image (which contains the shift counts) can have any maxval and the maxval is irrelevant to the interpretation of the samples. The sample value is the actual shift count. But it's still required that no sample value exceed the maxval.

The Operations

Most of the operations are obvious from the option name.

-subtract subtracts a value in the right input image from a value in the left input image.

-difference calculates the absolute value of the difference.

-multiply does an ordinary arithmetic multiplication, but tends to produce nonobvious results because of the way **pamarith** interprets sample values. See Maxval .

-compare produces the value **0** when the value in the left input image is less than the value in the right input image, **1** when the values are equal, and **2** when the left is greater than the right.

-and, **-nand**, **-or**, **-nor**, and **-xor** consider the input and output images to contain bit strings; they compute bitwise logic operations.

-shiftleft and **-shiftright** consider the left input image and output image to contain bit strings. They compute a bit shift operation, with bits falling off the left or right end and zeroes shifting in, as opposed to bits off one end to the other. The right input image sample value is the number of bit positions to shift.

Note that the maxval (see Maxval) determines the width of the frame within which you are shifting.

Notes

If you want to apply a unary function, e.g. "halve", to a single image, use **pamfunc**.

SEE ALSO

pamfunc(1), **pnminvert**(1), **ppmbrighten**(1), **ppmdim**(1), **pnmconvol**(1), **pnmdepth**(1), **pnmpsnr**(1), **pnm**(1), **pam**(1)

HISTORY

pamarith replaced **pnmarith** in Netpbm 10.3 (June 2002).

In Netpbm 10.3 through 10.8, though, **pamarith** was not backward compatible because it required the input images to be of the same depth, so you could not multiply a PBM by a PPM as is often done for masking. (It was not intended at the time that **pnmarith** would be removed from Netpbm -- the plan

was just to rewrite it to use **pamarith**; it was removed by mistake).

But starting with Netpbm 10.9 (September 2002), **pamarith** allows the images to have different depths as long as one of them has depth 1, and that made it backward compatible with **pnmarith**.

The original **pnmarith** did not have the **-mean** option.

The **compare** option was added in Netpbm 10.13 (December 2002).

The bit string operations were added in Netpbm 10.27 (March 2005).

Table Of Contents

NAME

pamchannel - extract channels from a PAM image

SYNOPSIS

pamchannel [**-infile** *infile*] [**-tupletype** *tupletype*] [*channum* ...]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamchannel reads a PAM or PNM image as input and produces a PAM image as output, consisting of the indicated channels (planes) of the input.

The output is the same dimensions as the input, except that the depth is the number of *channum* arguments you supply. The tuple type is a null string unless you specify the **-tupletype** option.

pamstack does the opposite of **pamchannel**: It takes multiple PAM or PNM images as input and stacks their planes (channels) on top of one another to yield a single PAM.

OPTIONS

-infile *infile*

This specifies the input file, which defaults to Standard Input. You may specify **-** to select Standard Input explicitly.

This is a little unconventional for Netpbm programs, which usually have the input file specification as an argument. For **pamchannel**, the arguments are channel numbers.

-tupletype *tupletype*

This specified the tuple type name to be recorded in the output. You may use any string up to 255 characters. Some programs recognize some names. If you omit this option, the default tuple type name is null.

SEE ALSO

pam(1) **pamstack**(1)

Table Of Contents

NAME

pamcomp - composite (overlay) two Netpbm images together

SYNOPSIS

pamcomp

[**-align=**{**left**|**center**|**right**| **beyondleft**|**beyondright**}] [**-valign=**{**top**|**middle**|**bottom**| **above**|**below**}] [**-xoff=***X*] [**-yoff=***Y*] [**-alpha=***alpha-pgmfile*] [**-invert**] [**-opacity=***opacity] [***-linear***] overlay_file [underlying_file [output_file]]*

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamcomp reads two images and produces a composite image with one of the images overlayed on top of the other, possible translucently. The images need not be the same size. The input and outputs are Netpbm format image files.

In its simplest use, **pamcomp** simply places the image in the file *overlay_file* on top of the image in the file *underlying_file*, blocking out the part of *underlying_file* beneath it.

If you add the **-alpha** option, then **pamcomp** uses the image in file *alpha-pgmfile* as an alpha mask, which means it determines the level of transparency of each point in the overlay image. The alpha mask must have the same dimensions as the overlay image. In places where the alpha mask defines the overlay image to be opaque, the composite output contains only the contents of the overlay image; the underlying image is totally blocked out. In places where the alpha mask defines the overlay image to be transparent, the composite output contains none of the overlay image; the underlying image shows through completely. In places where the alpha mask shows a value in between opaque and transparent (translucence), the composite image contains a mixture of the overlay image and the underlying image and the level of translucence determines how much of each.

The alpha mask is a PGM file in which a white pixel represents opaqueness and a black pixel transparency. Anything in between is translucent. (Like any Netpbm program, **pamcomp** will see a PBM file as if it is PGM).

If the overlay image is a PAM image of tuple type RGB_ALPHA or GRAYSCALE_ALPHA, then the overlay image contains transparency information itself and **pamcomp** uses it the same way as the alpha mask described above. If you supply both an overlay image that has transparency information and an alpha mask, **pamcomp** multiplies the two opacities to get the opacity of the overlay pixel.

Before Netpbm 10.25 (October 2004), **pamcomp** did not recognize the transparency information in a PAM image -- it just ignored it. So people had to make appropriate alpha masks in order to have a non-opaque overlay. Some Netpbm programs that convert from image formats such as PNG that contain transparency information are not able to create RGB_ALPHA or GRAYSCALE_ALPHA PAM output, so you have to use the old method -- extract the transparency information from the original into a separate alpha mask and use that as input to **pamcomp**.

The output image is always of the same dimensions as the underlying image. **pamcomp** uses only parts of the overlay image that fit within the underlying image.

To specify where on the underlying image to place the overlay image, use the **-align**, **-valign**, **-xoff**, and **-yoff** options. Without these options, the default horizontal position is flush left and the default vertical position is flush top.

The overlay image, in the position you specify, need not fit entirely within the underlying image. **pamcomp** uses only the parts of the overlay image that appear above the underlying image. It is possible to specify positioning such that *none* of the overlay image is over the underlying image -- i.e. the overlay is out of frame. If you do that, **pamcomp** issues a warning.

The overlay and underlying images may be of different formats (e.g. overlaying a PBM text image over a full color PPM image) and have different maxvals. The output image has the more general of the two input formats and a maxval that is the least common multiple the two maxvals (or the maximum maxval allowable by the format, if the LCM is more than that).

OPTIONS

-align=*alignment*

This option selects the basic horizontal position of the overlay image with respect to the underlying image, in syntax reminiscent of HTML. **left** means flush left, **center** means centered, and **right** means flush right.

The **-xoff** option modifies this position.

beyondleft means just out of frame to the left -- the right edge of the overlay is flush with the left edge of the underlying image. **beyondright** means just out of frame to the right. These alignments are useful only if you add a **-xoff** option. These two values were added in Netpbm 10.10 (October 2002).

The default is **left**.

-valign=*alignment*

This option selects the basic vertical position of the overlay image with respect to the underlying image, in syntax reminiscent of HTML. **top** means flush top, **middle** means centered, and **bottom** means flush bottom.

The **-yoff** option modifies this position.

above means just out of frame to the top -- the bottom edge of the overlay is flush with the top edge of the underlying image. **below** means just out of frame to the bottom. These alignments are useful only if you add a **-yoff** option. These two values were added in Netpbm 10.10 (October 2002).

The default is **top**.

-xoff=*x* This option modifies the horizontal positioning of the overlay image with respect to the underlying image as selected by the **-align** option. **pamcomp** shifts the overlay image from that basic position *x* pixels to the right. *x* can be negative to indicate shifting to the left.

The overlay need not fit entirely (or at all) on the underlying image. **pamcomp** uses only the parts that lie over the underlying image.

Before Netpbm 10.10 (October 2002), **-xoff** was mutually exclusive with **-align** and always measured from the left edge.

-yoff=*y* This option modifies the vertical positioning of the overlay image with respect to the underlying image as selected by the **-valign** option. **pamcomp** shifts the overlay image from that basic position *y* pixels downward. *y* can be negative to indicate shifting upward.

The overlay need not fit entirely (or at all) on the underlying image. **pamcomp** uses only the parts that lie over the underlying image.

Before Netpbm 10.10 (October 2002), **-xoff** was mutually exclusive with **-valign** and always measured from the top edge.

-alpha=*alpha-pgmfile*

This option names a file that contains the alpha mask. If you don't specify this option, there is no alpha mask, which is equivalent to having an alpha mask specify total opaqueness

everywhere.

You can specify - as the value of this option and the alpha mask will come from Standard Input. If you do this, don't specify Standard Input as the source of any other input image.

-invert This option inverts the sense of the values in the alpha mask, which effectively switches the roles of the overlay image and the underlying image in places where the two intersect.

-opacity=*opacity*

This option tells how opaque the overlay image is to be, i.e. how much of the composite image should be from the overlay image, as opposed to the underlying image. *opacity* is a floating point number, with 1.0 meaning the overlay image is totally opaque and 0.0 meaning it is totally transparent. The default is 1.0.

If you specify an alpha mask (the **-alpha** option), **pamcomp** uses the product of the opacity indicated by the alpha mask (as modified by the **-invert** option, as a fraction, and this opacity value. The **-invert** option does not apply to this opacity value.

As a simple opacity value, the value makes sense only if it is between 0 and 1, inclusive. However, **pamcomp** accepts all values and performs the same arithmetic computation using whatever value you provide. An opacity value less than zero means the underlay image is intensified and then the overlay image is "subtracted" from it. An opacity value greater than unity means the overlay image is intensified and the underlaying image subtracted from it. In either case, **pamcomp** clips the resulting color component intensities so they are nonnegative and don't exceed the output image's maxval.

This may seem like a strange thing to do, but it has uses. You can use it to brighten or darken or saturate or desaturate areas of the underlaying image. See this description (1) of the technique.

This option was added in Netpbm 10.6 (July 2002). Before Netpbm 10.15 (April 2003), values less than zero or greater than unity were not allowed.

-linear This option indicates that the inputs are not true Netpbm images but rather a non-gamma-adjusted variation. This is relevant only when you mix pixels, using the **-opacity** option or an alpha mask (the **-alpha** option).

The alpha mask and **-opacity** values indicate a fraction of the light intensity of a pixel. But the PNM and PNM-equivalent PAM image formats represent intensities with gamma-adjusted numbers that are not linearly proportional to intensity. So **pamcomp**, by default, performs a calculation on each sample read from its input and each sample written to its output to convert between these gamma-adjusted numbers and internal intensity-proportional numbers.

Sometimes you are not working with true PNM or PAM images, but rather a variation in which the sample values are in fact directly proportional to intensity. If so, use the **-linear** option to tell **pamcomp** this. **pamcomp** then will skip the conversions.

The conversion takes time. And the difference between intensity-proportional values and gamma-adjusted values may be small enough that you would barely see a difference in the result if you just pretended that the gamma-adjusted values were in fact intensity-proportional. So just to save time, at the expense of some image quality, you can specify **-linear** even when you have true PPM input and expect true PPM output.

For the first 13 years of Netpbm's life, until Netpbm 10.20 (January 2004), **pamcomp**'s predecessor **pnmcomp** always treated the PPM samples as intensity-proportional even though they were not, and drew few complaints. So using **-linear** as a lie is a reasonable thing to do if speed is important to you.

Another technique to consider is to convert your PNM image to the linear variation with **pnmgamma**, run **pamcomp** on it and other transformations that like linear PNM, and then convert it back to true PNM with **pnmgamma -ungamma**. **pnmgamma** is often faster than **pamcomp** in doing the conversion.

SEE ALSO

ppmmix(1)**and pnmpaste**(1)**are**simpler,**less**general versions of the same tool.

ppmcolormask(1)**and pbmmask**(1)**can**help**with**generating**an**alpha mask.

pnmcomp(1)**is**an**older**program**that** runs faster, but has less function.

pnm(1)

HISTORY

pamcomp was new in Netpbm 10.21 (March 2004). Its predecessor, **pnmcomp**, was one of the first programs added to Netpbm when the project went global in 1993.

AUTHOR

Table Of Contents

NAME

pamcut - cut a rectangle out of a PAM, PBM, PGM, or PPM image

SYNOPSIS

pamcut

[**-left** *colnum*]

[**-right** *colnum*]

[**-top** *rownum*]

[**-bottom** *rownum*]

[**-width** *cols*]

[**-height** *rows*]

[**-pad**]

[**-verbose**]

[*left top width height*]

[*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamcut reads a PAM, PBM, PGM, or PPM image as input and extracts the specified rectangle, and produces the same kind of image as output.

There are two ways to specify the rectangle to cut: arguments and options. Options are easier to remember and read, more expressive, and allow you to use defaults. Arguments were the only way available before July 2000.

If you use both options and arguments, the two specifications get mixed in an unspecified way.

To use options, just code any mixture of the **-left**, **-right**, **-top**, **-bottom**, **-width**, and **-height** options. What you don't specify defaults. Those defaults are in favor of minimal cutting and in favor of cutting the right and bottom edges off. It is an error to overspecify, i.e. to specify all three of **-left**, **-right**, and **-width** or **-top**, **-bottom**, and **-height**.

To use arguments, specify all four of the *left*, *top*, *width*, and *height* arguments. *left* and *top* have the same effect as specifying them as the argument of a **-left** or **-top** option, respectively. *width* and *height* have the same effect as specifying them as the argument of a **-width** or **-height** option, respectively, where they are positive. Where they are not positive, they have the same effect as specifying one less than the value as the argument to a **-right** or **-bottom** option, respectively. (E.g. *width* = 0 makes the cut go all the way to the right edge). Before July 2000, negative numbers were not allowed for *width* and *height*.

Input is from Standard Input if you don't specify the input file *pnmfile*.

Output is to Standard Output.

If you are splitting a single image into multiple same-size images, **pamdice** is faster than running **pamcut** multiple times.

pamcomp is also useful for cutting and padding an image to a certain size. You create a background image of the desired frame dimensions and overlay the subject image on it.

OPTIONS

-left=*colnum*

The column number of the leftmost column to be in the output. Columns left of this get cut out. If a nonnegative number, it refers to columns numbered from 0 at the left, increasing to the right. If negative, it refers to columns numbered -1 at the right, decreasing to the left.

-right=*colnum*

The column number of the rightmost column to be in the output, numbered the same as for **-left.** Columns to the right of this get cut out.

-top=*rownum*

The row number of the topmost row to be in the output. Rows above this get cut out. If a nonnegative number it refers to rows numbered from 0 at the top, increasing downward. If negative, it refers to columns numbered -1 at the bottom, decreasing upward.

-bottom=*rownum*

The row number of the bottom-most row to be in the output, numbered the same as for **-top**. Rows below this get cut out.

-width=*cols*

The number of columns to be in the output. Must be positive.

-height=*rows*

The number of rows to be in the output. Must be positive.

-pad If the rectangle you specify is not entirely within the input image, **pamcut** fails unless you also specify **-pad**. In that case, it pads the output with black up to the edges you specify. You can use this option if you need to have an image of certain dimensions and have an image of arbitrary dimensions.

pnmpad also adds borders to an image, but you specify their width directly.

pamcomp does a more general form of this padding. Create a background image of the frame dimensions and overlay the subject image on it. You can use options to have the subject image in the center of the frame or against any edge and make the padding any color (the padding color is the color of the background image).

-verbose

Print information about the processing to Standard Error.

SEE ALSO

pnmcrop(1), **pamcomp**(1), **pnmpad**(1), **pnmcat**(1), **pgmslice**(1), **pnm**(1)

HISTORY

pamcut was derived from **pnmcut** in Netpbm 9.20 (May 2001). It was the first Netpbm program adapted to the new PAM format and programming library.

The predecessor **pnmcut** was one of the oldest tools in the Netpbm package.

AUTHOR

Table Of Contents

NAME

pamdeinterlace - remove every other row from a PAM/PNM image

SYNOPSIS

pamdeinterlace

[**-takeodd**]

[**-takeeven**]

N

[*infile*]

You can use the minimum unique abbreviation of the options. You can use two hyphens instead of one. You can separate an option name from its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamdeinterlace removes all the even-numbered or odd-numbered rows from the input PNM or PAM image. Specify which with the **-takeeven** and **-takeodd** options.

This can be useful if the image is a video capture from an interlaced video source. In that case, each row shows the subject 1/60 second before or after the two rows that surround it. If the subject is moving, this can detract from the quality of the image.

Because the resulting image is half the height of the input image, you will then want to use **pamstretch** or **pamscale** to restore it to its normal height:

```
pamdeinterlace myimage.ppm | pamstretch -yscale=2 >newimage.ppm
```

OPTIONS

-takeodd

Take the odd-numbered rows from the input and put them in the output. The rows are numbered starting at zero, so the first row in the output is the second row from the input. You cannot specify both **-takeeven** and **-takeodd**.

-takeeven

Take the even-numbered rows from the input and put them in the output. The rows are numbered starting at zero, so the first row in the output is the first row from the input. This is the default. You cannot specify both **-takeeven** and **-takeodd**.

SEE ALSO

pamstretch(1), **pamscale**(1)

Table Of Contents

NAME

pamdice - slice a Netpbm image into many horizontally and/or vertically

SYNOPSIS

pamdice

-outstem=*filenamestem*

[**-width=***width*]

[**-height=***height*]

[**-hoverlap=***hoverlap*]

[**-voverlap=***voverlap*]

[**-verbose**]

[*filename*]

You can use the minimum unique abbreviation of the options. You can use two hyphens instead of one. You can separate an option name from its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamdice reads a PAM, PBM, PGM, or PPM image as input and splits it horizontally and/or vertically into equal size pieces and writes them into separate files as the same kind of image. You can optionally make the pieces overlap.

See the **-outstem** option for information on naming of the output files.

The **-width** and **-height** options determine the size of the output pieces.

pnmcat can rejoin the images.

One use for this is to make pieces that take less computer resources than the whole image to process. For example, you might have an image so large that an image editor can't read it all into memory or processes it very slowly. With **pamdice**, you can split it into smaller pieces, edit one a time, and then reassemble them.

Another use for this is to print a large image in small printer-sized pieces that you can glue together. **ppmglobe** does a similar thing; it lets you glue the pieces together into a sphere.

OPTIONS

-outstem=filenamestem

This option determines the names of the output files. Each output file is named *filenamestem_y_x.type* where *filenamestem* is the value of the **-outstem** option, *x* and y are the horizontal and vertical locations, respectively, in the input image of the output image, zero being the leftmost and top, and *type* is **.pbm**, **.pgm**, **.ppm**, or **.pam**, depending on the type of image.

-width=*width*

gives the width in pixels of the output images. The rightmost pieces are smaller than this if the input image is not a multiple of *width* pixels wide.

-height=*height*

gives the height in pixels of the output images. The bottom pieces are smaller than this if the input image is not a multiple of *height* pixels high.

-hoverlap=*hoverlap*

gives the horizontal overlap in pixels between output images. Each image in a row will overlap the previous one by *hoverlap* pixels. By default, there is no overlap.

This option was new in Netpbm 10.23 (July 2004).

-voverlap=*voverlap*

gives the vertical overlap in pixels between output images. Each row of images will overlap the previous row by *voverlap* pixels. By default, there is no overlap.

This option was new in Netpbm 10.23 (July 2004).

-verbose

Print information about the processing to Standard Error.

SEE ALSO

pamcut(1), **pnmcat**(1), **pgmslice**(1), **ppmglobe**(1) **pnm**(1) **pnm**(1)

Table Of Contents

NAME

pamditherbw - dither grayscale image to black and white

SYNOPSIS

pamditherbw

[**-floyd** | **-fs** | **-threshold** | **-hilbert** | **-dither8** | **-d8** | **-cluster3** | **-c3** | **-cluster4** | **-c4** | **-cluster8** | **-c8**]

[**-value** *val*]

[**-clump** *size*]

[*pamfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamditherbw dithers a grayscale image. Dithering means turning each shade of gray into a pattern of black and white pixels that, from a distance, look the same as the gray.

The input should be a PGM image or a PAM image of tuple type GRAYSCALE. However, **pamditherbw** doesn't check, so if you feed it e.g. a PPM image, it will produce arbitrary results (actually, it just takes the first channel of whatever you give it and treats it as if it represented gray levels).

The output is a PAM with tuple type BLACKANDWHITE. You can turn this into a PBM (if you need to use it with an older program doesn't understand PAM) with **pamtopnm**.

To do the opposite of dithering, you can usually just scale the image down and then back up again with **pamscale**, possibly smoothing or blurring the result with **pnmsmooth** or **pnmconvol**. Or use the special case program **pbmtopgm**.

To dither a color image (to reduce the number of pixel colors), use **ppmdither**.

OPTIONS

The default quantization method is boustrophedonic Floyd-Steinberg error diffusion (**-floyd** or **-fs**).

Also available are simple thresholding (**-threshold**); Bayer's ordered dither (**-dither8**) with a 16x16 matrix; and three different sizes of 45-degree clustered-dot dither (**-cluster3**, **-cluster4**, **-cluster8**).

A space filling curve halftoning method using the Hilbert curve is also available (**-hilbert**).

Floyd-Steinberg will almost always give the best looking results; however, looking good is not always what you want. For instance, thresholding can be used in a pipeline with the **pnmconvol** tool, for tasks like edge and peak detection. And clustered-dot dithering gives a newspaper-ish look, a useful special effect.

The **-value** option alters the thresholding value for Floyd-Steinberg and simple thresholding. It should be a real number between 0 and 1. Above 0.5 means darker images; below 0.5 means lighter.

The Hilbert curve method is useful for processing images before display on devices that do not render individual pixels distinctly (like laser printers). This dithering method can give better results than the dithering usually done by the laser printers themselves. The **-clump** option alters the number of pixels in a clump. This is usually an integer between 2 and 100 (default 5). Smaller clump sizes smear the image less and are less grainy, but seem to loose some grey scale linearity. Typically a PGM image will have to be scaled to fit on a laser printer page (2400 x 3000 pixels for an A4 300 dpi page), and then dithered to a PBM image before being converted to a postscript file. A printing pipeline might look something like:

```
pamscale -xysize 2400 3000 image.pgm | pamditherbw -hilbert |  pamtopnm | pnmtops -scale 0.25 > image.ps
```

All options can be abbreviated to their shortest unique prefix.

REFERENCES

The only reference you need for this stuff is 'Digital Halftoning' by Robert Ulichney, MIT Press, ISBN 0-262-21009-6.

The Hilbert curve space filling method is taken from 'Digital Halftoning with Space Filling Curves' by Luiz Velho, Computer Graphics Volume 25, Number 4, proceedings of SIGRAPH '91, page 81. ISBN 0-89791-436-8

SEE ALSO

pamtopnm(1), **pgmtopgm**(1), **pbmtopgm**(1), **pbmreduce**(1), **pnmconvol**(1), **pamscale**(1), **pam**(1), **pnm**(1),

HISTORY

pamditherbw was new in Netpbm 10.23 (July 2004), but is essentially the same program as **pgmtopbm** that has existed practically since the beginning. **pamditherbw** differs from its predecessor in that it properly deals with gamma adjustment and that it accepts PAM input in addition to PGM and PBM and produces PAM output.

pamditherbw obsoletes **pgmtopbm**.

AUTHOR

Table Of Contents

NAME

pamedge - edge-detect an image

SYNOPSIS

pamedge [*imagefile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamedgereads a Netpbm image (PNM or PAM) and produces an image that outlines the edges.

The output image is of the same type as the input, except that the maxval of the output is at least 255 and if the input is PBM, the output is PGM.

You can pipe the result through `pamditherbw -threshold` and play with the threshold value to get a PBM (bilevel image) of the edges.

The edge detection technique used is to take the Pythagorean sum of two Sobel gradient operators at 90 degrees to each other. For more details see 'Digital Image Processing' by Gonzalez and Wintz, chapter 7.

The maxval of the output is the same as the maxval of the input The effect is better with larger maxvals, so you may want to increase the maxval of the input by running it through **pnmdepth** first.

SEE ALSO

pgmenhance(1), **pamditherbw**(1), **pnmdepth**(1), **pammasksharpen**(1), **pamsharpness**(1), **pamsharpmap**(1), **pam**(1), **pnm**(1)

HISTORY

pamedge was added to Netpbm in Release 10.14 (March 2003). It replaced **pgmedge**, which was the same thing, but worked only on PGM and PBM images.

AUTHOR

 Adapted to **pnmedge** Peter Kichgessner in 1995, and then to **pamedge** by Bryan Henderson in 2003.

Table Of Contents

NAME

pamendian - reverse endianness of a Netpbm image

SYNOPSIS

pamendian

DESCRIPTION

This program is part of **Netpbm**(1).

All Netpbm formats that have samples in pure binary format with multiple bytes are defined to have them in big endian (most significant byte first) order. However, there exist variations on these formats, primarily developed before official multibyte Netpbm formats existed, that are identical to Netpbm formats in every respect except that samples are in little endian (least signficant byte first) order.

pamendian reverses the byte order of the sample to convert between the two formats. If the input is true PAM, PGM, or PPM, the output is the little endian variation on that format, and vice versa.

Programs that come with the Independent Jpeg Group's JPEG library are known to use the little endian variation of PGM and PPM.

This program takes input only on Standard Input. Its output is always on Standard Output.

You should never have to use this program with images generated by programs in the Netpbm package or programs that use the Netpbm libraries. If you do, that probably means something needs to be fixed in those programs. The Netpbm converter for any graphics format that represents numbers in little endian form should properly reverse the bytes to create correct Netpbm output.

If you create a Netpbm image from a generic stream of samples, using **rawtopgm** or **rawtoppm**, use options on those programs to declare the endianness of your input, thus creating correct endianness in your PGM or PPM output.

SEE ALSO

pnmdepth(1), **rawtopgm**(1), **rawtoppm**(1), **pnm**(1)

NAME

pamenlarge − see http://netpbm.sourceforge.net/doc//pamenlarge.html

DESCRIPTION

pamenlarge is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//pamenlarge.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

pamfile - describe a Netpbm (PAM or PNM) file

SYNOPSIS

pamfile

[**-allimages**]

[*file* ...]

DESCRIPTION

This program is part of **Netpbm**(1).

pamfile reads one or more Netpbm files as input and writes out short descriptions of the image type, size, etc. This is mostly for use in shell scripts, so the format is not particularly pretty.

By default, **pamfile** reads only the header of the input file. If that file is a pipe, that might cause problems for the process that is feeding the pipe. In that case, see the **-allimages** option.

OPTIONS

-allimages

Describe every image in each input file. Without this option, **pamfile** describes only the first image in each input file.

This option also causes **pamfile** to read all the images from the input stream, whereas without it, **pamfile** reads only the header of the first one. If the input stream is from a pipe, the process that is feeding the pipe might require the entire stream to be consumed. In that case, use this option even if the stream contains only one image.

Note that before July 2000, a file could not contain more than one image and many programs ignore all but the first.

SEE ALSO

pam(1), **ppmhist**(1), **file**

AUTHOR

Table Of Contents

NAME

pamflip - flip or rotate a PAM or PNM image

SYNOPSIS

pamflip { **-leftright** | **-lr** | **-topbottom** | **-tb** | **-transpose** | **-xy** | **-rotate90** | **-r90** | **-cw** | **-rotate270** | **-r270** | **-ccw** | **-rotate180** | **-r180** **-null** | **-xform=***xform1***,***xform2*... } [**-memsize=***mebibytes*] [**-pagesize=***bytes*] [*pamfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamflip flips a PAM or PNM image top for bottom or left for right, or transposes it horizontal for vertical, or rotates it 1, 2, or 3 quarter turns.

To rotate at other angles, use **pnmrotate**. It is much slower, though.

The input image is *pamfile*, or Standard Input if *pamfile* is not specified.

To flip/rotate a JFIF (JPEG) image losslessly, use **jpegtran**. **jpegtran** is part of the Independent Jpeg Group's compression library package, not part of Netpbm. The normal Netpbm way to flip a JFIF file would be to convert it to PNM, use **pamflip**, and convert back to JFIF. But since JPEG compression is lossy, the resulting image would have less quality than the original. **jpegtran**, on the other hand, can do this particular transformation directly on the compressed data without loss.

OPTIONS

You must supply exactly one of the following options:

pamflip's predecessor (before Netpbm 10.7 - August 2002) **pnmflip** did not have the **-xform** option and instead allowed you to specify any number of the other options, including zero. It applied all the indicated transformations, in the order given, just like with **pamflip**'s **-xform** option. (Reason for the change: this kind of interpretation of options is inconsistent with the rest of Netpbm and most of the Unix world, and thus hard to understand and to implement).

-leftright

-lr Flip left for right.

-topbottom

-tb Flip top for bottom.

-transpose

-xy Transpose horizontal for vertical. I.e. make the pixel at (x,y) be at (y,x).

-rotate90

-r90

-ccw Rotate counterclockwise 90 degrees.

-rotate180

-r180 Rotate 180 degrees.

-rotate270

-r270

-cw Rotate counterclockwise 270 degrees (clockwise 90 degrees)

-null No change. (The purpose of this option is the convenience of programs that invoke **pamflip** after computing the kind of transformation desired, including none at all).

This option was new in Netpbm 10.13 (December 2002).

-xform=*xform1***,***xform2*...
Apply all the transforms listed, in order. The valid values for the transforms are as follows and have the same meanings as the identically named options above.

- leftright
- topbottom
- transpose

This option was new in Netpbm 10.13 (December 2002).

The following options help **pamflip** use memory efficiently. Some flipping operations on very large images can cause **pamflip** to have a very large working set, which means if you don't have enough real memory, the program can page thrash, which means it takes a ridiculous amount time to run. If your entire image fits in real memory, you don't have a problem. If you're just flipping top for bottom or left for right, you don't have a problem. Otherwise, pay attention. If you're interested in the details of the thrashing problem and how **pamflip** approaches it, you're invited to read a complete explanation in comments in the source code.

-memsize=*mebibytes*
mebibytes is the size in mebibytes (aka megabytes) of *real* memory (not virtual) available for **pamflip**. **pamflip** does nothing special to allocate real memory or control it's allocation -- it gets whatever it gets just by referencing virtual memory normally. This is the maximum amount that **pamflip** can be expected to end up with by doing that. This is just about impossible for you to know, of course, but you can estimate. The total real memory in your system should be a major factor in your estimate.

When you specify **-memsize** and are doing a row for column type of transformation, **pamflip** does the transformation in multiple passes, each one with a working set size less than the specified value.

If your estimate is even slightly too large, it's the same as infinity. If you estimate too small, **pamflip** will use more passes than it needs to, and thus will slow down proportional to the underestimate.

If you do not specify **-memsize**, **pamflip** assumes infinite real memory and does the entire transformation in one pass.

This option did not exist before Netpbm 10.7 (August 2002).

-pagesize=*bytes*

bytes is the size in bytes of a paging unit -- the amount of memory that gets paged in or out as an indivisible unit -- in your system. The default is 4KiB.

This option did not exist before Netpbm 10.7 (August 2002).

Miscellaneous options:

-verbose

This option causes **pamflip** to issue messages to Standard Error about its progress.

SEE ALSO

pnmrotate(1), **pnm**(1), **jpegtran** manual

HISTORY

pamflip replaced **pnmflip** in Netpbm 10.13 (December 2002). **pamflip** is backward compatible, but also works on PAM images.

AUTHOR

Table Of Contents

NAME

pamfunc - Apply simple arithmetic functions to a Netpbm image

SYNOPSIS

pamfunc { **-multiplier**=*realnum* | **-divisor**=*realnum* | **-adder**=*integer* | **-subtractor**=*integer* | **-min**=*wholenum* | **-max**=*wholenum* } [*filespec*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamfunc reads a Netpbm image as input and produces a Netpbm image as output, with the same format, maxval, and dimensions as the input. **pamfunc** applies a simple transfer function to each sample in the input to generate the corresponding sample in the output. The options determine what function.

pamarith is the same thing, except only for PNM images, for binary functions -- it takes two PNM images as input and applies a specified simple arithmetic function (e.g. addition) on pairs of samples from the two to produce the single output image.

OPTIONS

-multiplier=*realnum*

This option makes the transfer function that of multiplying by *realnum*. *realnum* must be nonnegative. If the result is greater than the image maxval, it is clipped to the maxval.

Where the input is a PGM or PPM image, this has the effect of dimming or brightening it. For a different kind of brightening, see **ppmbrighten**(1)**and ppmflash**(1)

Also, see **ppmdim**(1),**which**does**the** same thing as **pamfunc -multiplier** on a PPM image with a multiplier between 0 and 1, except it uses integer arithmetic, so it may be faster.

And **ppmfade**(1)**can**generate**a**whole sequence of images of brightness declining to black or increasing to white, if that's what you want.

-divisor=*realnum*

This option makes the transfer function that of dividing by *realnum*. *realnum* must be nonnegative. If the result is greater than the image maxval, it is clipped to the maxval.

This is the same function as you would get with **-multiplier**, specifying the multiplicative inverse of *realnum*.

-adder=*integer*

This option makes the transfer function that of adding *wholenum*. If the result is greater than the image maxval, it is clipped to the maxval. If it is less than zero, it is

clipped to zero.

Note that in mathematics, this entity is called an 'addend,' and an 'adder' is a snake. We use 'adder' because it makes more sense.

-subtractor=*integer*

This option makes the transfer function that of subtracting *wholenum*. If the result is greater than the image maxval, it is clipped to the maxval. If it is less than zero, it is clipped to zero.

Note that in mathematics, this entity is called a 'subtrahend' rather than a 'subtractor.' We use 'subtractor' because it makes more sense.

This is the same function as you would get with **-multiplier**, specifying the negative of *integer*.

-min=*wholenum*

This option makes the transfer function that of taking the maximum of the argument and *wholenum*. I.e the minimum value in the output will be *wholenum*.

If *wholenum* is greater than the maxval, though, every sample in the output will be maxval.

-max=*wholenum*

This option makes the transfer function that of taking the minimum of the argument and *wholenum*. I.e the maximum value in the output will be *wholenum*.

If *wholenum* is greater than the maxval, the function is idempotent -- the output is identical to the input.

SEE ALSO

ppmdim(1), **ppmbrighten**(1), **pnmdepth**(1), **pamarith**(1), **ppmfade**(1), **pbm**(1), **pbm**(1),

HISTORY

This program was added to Netpbm in Release 10.3 (June 2002).

Table Of Contents

NAME

pamgauss - create a two dimensional gaussian function as a PAM image

SYNOPSIS

pamgauss -*width* -*height* **-sigma=***number* [**-maxval=***number*] [**-tupletype=***string*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

EXAMPLES

```
pamgauss 3 3 -sigma=.5 -tupletype=GRAYSCALE | pamtopnm >gauss.pgm
pnmconvol -nooffset gauss.pgm myimage.ppm >blurred.ppm
```

DESCRIPTION

This program is part of **Netpbm**(1).

pamgauss generates a one-plane PAM image whose samples are a gaussian function of their distance from the center of the image. I.e. the sample value is highest in the center and goes down, in a bell curve shape, as you move away from the center.

The values are scaled so that the area under the surface of the two-dimensional Gaussian function is the maxval of the image.

You can use this image, converted to PGM, as a convolution kernel with **pnmconvol** to blur an image. (This technique is known as Gaussian blurring).

width and *height* are the dimensions of the image that **pamgauss** generates. Mathematically speaking, they are the domain of the two dimensional gaussian function.

The sum of all the samples is equal to the image's maxval (within rounding error). This is true even if you clip the Gaussian function by making the image too small. If you want to be sure you get a whole Gaussian function, make sure that you choose a sigma and image dimensions so that if you made it any larger, the sample values at the edges would be zero.

The output image is PAM. To turn it into a PGM that you can use with **pnmconvol**, specify **-tupletype=GRAYSCALE** and pass the output through **pamtopnm**. You must use the **-nooffset** option on **pnmconvol** because zero means zero in the PAM that **pamgauss** generates.

DESCRIPTION

This program is part of **Netpbm**(1).

-sigma=*number*

This is the sigma parameter of the Gaussian function (if it were a Gaussian probability function, this would be its the standard deviation). The higher the number, the more spread out the function is. Normally, you want to make this number low enough that the function reaches zero value before the edge of your image.

number is in units of pixels.

This option is required. There is no default.

-maxval=*number*

This is the maxval for the output image. It defaults to 255.

-tupletype=*string*

This is the value of the "tuple_type" attribute of the created PAM image. It can be any string up to 255 characters. If you don't specify this, **pamgauss** generates a PAM with unspecified tuple type.

SEE ALSO

pnmconvol(1), **pamtopnm**(1), **pgmkernel**(1), **pamseq**(1), **pam**(1)

HISTORY

pamgauss was new in Netpbm 10.23 (July 2004).

Table Of Contents

NAME

pamlookup - map an image to a new image by using it as indices into a table

SYNOPSIS

pamlookup -lookupfile=*lookupfile* **-missingcolor=***color* [**-fit**] *indexfile*

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamlookup takes a two dimensional array of indices and a lookup table as input. For each position in the index array, it looks up the index in the lookup table and places the result of the lookup in the output image. The output thus has the same width and height as the index image, and tuple types determined by the lookup table.

An index is either a whole number or an ordered pair of whole numbers. If the index image has a depth of one, each index in it is a whole number: the value of the one sample. If the index image has a depth greater than one, each index in it is an ordered pair of the first and second samples in the relevant tuple.

The lookup table is a PAM or PNM image. If the index image contains whole number indices, the lookup image is a single row and the index is a column number. The lookup result is the value of the tuple or pixel at the indicated column in the one row in the lookup table. If the index image contains ordered pair indices, the first element of the ordered pair is a row number and the second element of the ordered pair is a column number. The lookup result is the value of the tuple or pixel at the indicated row and column in the lookup table.

For example: Consider an index image consisting of a 3x2x1 PAM as follows:

0 1 0
2 2 2

and a lookup table consisting of a 3x1 PPM image as follows:

red yellow beige

The lookup table above says Index 0 corresponds to the color red, Index 1 corresponds to yellow, and Index 2 corresponds to beige. The output of **pamlookup** is the following PPM image:

red yellow red
beige beige beige

Now let's look at an example of the more complex case where the indices are ordered pairs of whole numbers instead of whole numbers. Our index image will be this 3x2x2 PAM image:

(0,0) (0,1) (0,0)
(1,1) (1,0) (0,0)

Our lookup table for the example will be this two dimensional PPM:

red yellow
green black

This lookup table says Index (0,0) corresponds to the color red, Index (0,1) corresponds to yellow, Index (1,0) corresponds to green, and Index (1,1) corresponds to black. The output of **pamlookup** is the following PPM image:

red yellow red
black green red

If an index specifies a row or column that exceeds the dimensions of the lookup table image, **pamlookup** uses the value from the top left corner of the lookup image, or the value you specify with the **-missingcolor** option.

The *indexfile* argument identifies the file containing the index PAM or PNM image. **-** means Standard Input. The mandatory **-lookupfile** option identifies the file containing the lookup table image. Again, **-** means Standard Input. It won't work if both the index image file and lookup table file are Standard Input. The output image goes to Standard Output.

You can use **ppmmake** and **pnmcat** to create a lookup table file.

If you want to use two separate 1-plane images as indices (so that your output reflects the combination of both inputs), use **pamstack** to combine the two into one two-plane image (and use a 2-dimensional lookup table image).

OPTIONS

-lookupfile=*lookupfile*

lookupfile names the file that contains the PAM or PNM image that is the lookup table. This option is mandatory.

-missingcolor=*color*

This option is meaningful only if the lookup image (and therefore the output) is a PNM image. *color* specifies the color that is to go in the output wherever the index from the input is not present in the lookup table (not present means the index exceeds the dimensions of the lookup image -- e.g. index is 100 but the lookup image is a 50 x 1 PPM).

If you don't specify this option of **-fit**, **pamlookup** uses the value from the top left corner of the lookup image whenever an index exceeds the dimensions of the lookup image.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

Another way to deal with a too-small lookup image is to use the **-fit** option.

-fit This option says to shrink or expand the lookup image as necessary to fit the indices present in the index image, per the index image's maxval. For example, if your index image has a single plane and a maxval of 255 and your lookup image is 1 row of 10 columns, **pamlookup** stretches your lookup image to 255 columns before doing the lookups. **pamlookup** does the stretching (or shrinking) with the **pamscale**(1) program.

When you use **-fit**, **pamlookup** never fails or warns you due to invalid lookup image dimensions, and the **-missingcolor** option has no effect.

EXAMPLES

Example: rainfall map

Say you have a set of rainfall data in a single plane PAM image. The rows and columns of the PAM indicate lattitude and longitude. The sample values are the annual rainfall in (whole) centimeters. The highest rainfall value in the image is 199 centimeters. The image is in the file rainfall.pam.

You want to produce a PPM rainfall map with green for the wettest places, red for the driest, and other colors in between.

First, compose a lookup table image, probably with a graphical editor and the image blown way up so you can work with individual pixels. The image must have a single row and 200 columns. Make the leftmost pixel red and the rightmost pixel green and choose appropriate colors in between. Call it

colorkey.ppm.

```
pamlookup rainfall.ppm -lookupfile=colorkey.ppm >rainfallmap.ppm
```

Now lets say you're too lazy to type in 200 color values and nobody really cares about the places that have more than 99 centimeters of annual rainfall. In that case, just make colorkey.ppm 100 columns wide and do this:

```
pamlookup rainfall.ppm -lookupfile=colorkey.ppm -missingcolor=black        >rain
```

Now if there are areas that get more than 100 centimeters of rainfall, they will just show up black in the output.

Example: graphical diff

Say you want to compare two PBM (black and white) images visually. Each consists of black foreground pixels on a white background. You want to create an image that contains background where both images contain background and foreground where both images contain foreground. But where Image 1 has a foreground pixel and Image 2 does not, you want red in the output; where Image 2 has a foreground pixel and Image 1 does not, you want green.

First, we create a single image that contains the information from both input PBMs:

```
pamstack image1.pbm image2.pbm >bothimages.pam
```

Note that this image has 1 of 4 possible tuple values at each location: (0,0), (0,1), (1,0), or (1,1).

Now, we create a lookup table that we can index with those 4 values:

```
ppmmake white 1 1 >white.ppm
ppmmake black 1 1 >black.ppm
ppmmake red   1 1 >red.ppm
ppmmake green 1 1 >green.ppm
pnmcat -leftright black.ppm red.ppm   >blackred.ppm
pnmcat -leftright green.ppm white.ppm >greenwhite.ppm
pnmcat -topbottom blackred.ppm greenwhite.ppm >lookup.ppm
```

Finally, we look up the indices from our index in our lookup table and produce the output:

```
pamlookup bothimages.ppm -lookupfile=lookup.ppm >imagediff.ppm
```

SEE ALSO

pnmremap(1), **ppmmake**(1), **pnmcat**(1), **pamstack**(1), **pnm**(1), **pam**(1)

HISTORY

pamlookup was new in Netpbm 10.13 (December 2002).

NAME

pammasksharpen − see http://netpbm.sourceforge.net/doc//pammasksharpen.html

DESCRIPTION

pammasksharpen is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//pammasksharpen.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

pamoil - turn a PAM image into an oil painting

SYNOPSIS

pamoil

[**-n** *N*]

[*pamfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamoil reads a Netpbm image as input and does an 'oil transfer', and writes the same type of Netpbm image as output.

The oil transfer is described in 'Beyond Photography' by Holzmann, chapter 4, photo 7. It's a sort of localized smearing.

The smearing works like this: First, assume a grayscale image. For each pixel in the image, **pamoil** looks at a square neighborhood around it. **pamoil** determines what is the most common pixel intensity in the neighborhood, and puts a pixel of that intensity into the output in the same position as the input pixel.

For color images, or any arbitrary multi-channel image, **pamoil** computes each channel (e.g. red, green, and blue) separately the same way as the grayscale case above.

At the edges of the image, where the regular neighborhood would run off the edge of the image, **pamoil** uses a clipped neighborhood.

OPTIONS

-n *size* This is the size of the neighborhood used in the smearing. The neighborhood is this many pixels in all four directions.

The default is 3.

SEE ALSO

pgmbentley(1), **ppmrelief**(1), **ppm**(1)

AUTHOR

This program is based on pgmoil Copyright (C) 1990 by Wilson Bent (*whb@hoh-2.att.com*)

Modified to ppm by Chris Sheppard, June 25, 2001

Modified to pnm, using pam functions, by Bryan Henderson June 28, 2001.

Table Of Contents

NAME

pamperspective - a reverse scanline renderer for Netpbm images

SYNOPSIS

pamperspective
[**--bottom_margin**=*num*]
[**--detail**=*num*]
[**--frame_include**=*bool*]
[**--height**=*num*]
[**--include=[***x1*,*y1*;*x2*,*y2*; ...]]
[**--input_system**=*spec*]
[**--input_unit**=*spec*]
[**--interpolation**=*spec*]
[**--left_margin**=*num*]
[**--margin**=*num*]
[**--output_system**=*spec*]
[**--proportion**=*spec*]
[**--ratio**=*num*]
[**--right_margin**=*num*]
[**--top_margin**=*num*]
[**--width**=*num*]
{
{
upper_left_x upper_left_y upper_right_x upper_right_y
lower_left_x lower_left_y lower_right_x lower_right_y
}
|
{
{**--upper_left_x**|**--ulx**}=*upper_left_x*
{**--upper_left_y**|**--uly**}=*upper_left_y*
{**--upper_right_x**|**--urx**}=*upper_right_x*
{**--upper_right_y**|**--ury**}=*upper_right_y*
{**--lower_left_x**|**--llx**}=*lower_left_x*
{**--lower_left_y**|**--lly**}=*lower_left_y*
{**--lower_right_x**|**--lrx**}=*lower_right_x*
{**--lower_right_y**|**--lry**}=*lower_right_y*
}
}
[*infile*]

OPTION USAGE

Minimum unique abbreviation of option is acceptable. (But note that shortest unique prefixes might be longer in future versions of the program.) You may use single hyphens instead of double hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value. All options starting with hyphens may be given in any order.

DESCRIPTION

This program is part of **Netpbm**(1).

pamperspective reads a Netpbm image as input and produces a Netpbm image of the same format as output.

The values *upper_left_x ... lower_right_y*, that are required on the command line, specify a quadrilateral in the input image. **pamperspective** interprets the input image as a perspective projection of a

three dimensional scene, for example a photograph. **pamperspective** assumes the specified quadrilateral represents a parallelogram in that scene. The output image consists of this parallelogram, sheared to a rectangle. In this way **pamperspective** undoes the effect of a raytracer or scanline renderer.

The input is from *infile*, or from Standard Input, if *infile* is not specified. The output is to Standard Output.

OPTIONS

For options of the form **--name=***num*, You can specify the value *num* in any of the traditional ways. Additionally, you can specify it as *num1*/*num2*, where *num1* and *num2* are specified traditionally. This is useful for specifying a width/height ratio of 4/3, without having to write infinitely many digits. Where *num* is supposed to be a natural number, **pamperspective** does not allow this format.

Quadrilateral specification options

--upper_left_x=*num*

--ulx=*num*

This specifies the horizontal coordinate of the upper left vertex of the quadrilateral. The meaning of 'upper left' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--upper_left_y=*num*

--uly=*num*

This specifies the vertical coordinate of the upper left vertex of the quadrilateral. The meaning of 'upper left' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--upper_right_x=*num*

--urx=*num*

This specifies the horizontal coordinate of the upper right vertex of the quadrilateral. The meaning of 'upper right' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--upper_right_y=*num*

--ury=*num*

This specifies the vertical coordinate of the upper right vertex of the quadrilateral. The meaning of 'upper right' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--lower_left_x=*num*

--llx=*num*

This specifies the horizontal coordinate of the lower left vertex of the quadrilateral. The meaning of 'lower left' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--lower_left_y=*num*

--lly=*num*

This specifies the vertical coordinate of the lower left vertex of the quadrilateral. The meaning of 'lower left' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--lower_right_x=*num*

--lrx=*num*

This specifies the horizontal coordinate of the lower right vertex of the quadrilateral. The meaning of 'lower right' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--lower_right_y=*num*

--lry=*num*

This specifies the vertical coordinate of the lower right vertex of the quadrilateral. The meaning of 'lower right' is relative to the output image. The interpretation of *num* depends on the values for **--input_system** and **--input_unit**.

--input_system=*system*

--input_unit=*unit*

The input image consists of pixels, which are, from the point of view of a scanline renderer, solid squares. These options specify how the coordinates are interpreted:

system=**lattice**, *unit*=**image**

(0,0) refers to the upper left corner of the upper left pixel and (1,1) refers to the lower right corner of the lower right pixel.

system=**lattice**, *unit*=**pixel**

(0,0) refers to the upper left corner of the upper left pixel and (*width*,*height*) refers to the lower right corner of the lower right pixel. Here *width* and *height* are the width and height of the input image.

system=**pixel**, *unit*=**image**

(0,0) refers to the centre of the upper left pixel and (1,1) refers to the centre of the lower right pixel.

system=**pixel**, *unit*=**pixel**

(0,0) refers to the centre of the upper left pixel and (*width*-1,*height*-1) refers to the centre of the lower right pixel. Here *width* and *height* are the width and height of the input image.

The defaults are **--input_system**=**lattice** and **--input_unit**=**pixel**. Point-and-click front ends should use **--input_system**=**pixel**.

Frame options

By default **pamperspective** outputs exactly the above parallelogram, sheared to a rectangle. With the following options, it is possible to make **pamperspective** output a larger or smaller portion, which we call the 'visible part.' We refer to the default rectangle as the 'frame.' The visible part is always a rectangle the axes of which are parallel to those of the frame.

The frame options are additive. All the parts of the image specified by either margin options, **--include_frame**, or **--include** (or their defaults) are in the visible part. The visible part is the smallest possible rectangle that contains the parts specified those three ways.

The visible part must have nonzero size. That means if you specify **--frame-include=no** (overriding the default), you'll need to specify other frame options in order to have something in the visible part.

[**--margin=***num*]

This specifies an area surrounding the frame that is to be included in the visible part. The units of *num* are the width of the frame for the horizontal extensions and the height of the frame for vertical extensions.

For example, **--margin=1** makes the visible part 9 times as large, because it makes the visible part extend one frame's worth to the left of the frame, one frame's worth to the right, one frame's worth above the frame, and one frame's worth below the frame, for a total of 3 frames' worth in both dimensions.

A negative value has an effect only if you specify **--frame_include=no**. The default is no margin.

The individual margin options below override this common margin setting.

[**--top_margin**=*num*]

[**--left_margin**=*num*]

[**--right_margin**=*num*]

[**--bottom_margin**=*num*]

These are like **--margin**, but they specify only one of the 4 sides. The default value for each is the value (or default) of **--margin**.

[**--frame_include**=*bool*]

Valid values for *bool* are:

yes

true

on

The frame itself is in the visible part.

no

false

off

The frame itself is not necessarily in the visible part (but it could be if other options cause it to be).

The default value is **yes**

--include=[*x1*,*y1*;*x2*,*y2*; ...]

The visible part is made large enough such that every point (*x1*,*y1*), (*x2*,*y2*), of the *input* image is visible. The meaning of *x* and *y* is determined by **--input_system** and **--input_unit**. You can specify any number of semicolon-delimited points, including zero.

If you're supplying these options via a Unix command shell, be sure to use proper quoting, because semicolon (**;**) is usually a shell control character.

The frame options were new in Netpbm 10.25 (October 2004).

Output size options

--width=*width*

--height=*height*

These specify the size of the output image in horizontal and vertical direction. The values are numbers of pixels, so only natural numbers are valid. These values override the default means to determine the output size.

--detail=*num*

If you do not specify **--width**, **pamperspective** determines the width of the output image such that moving *num* output pixels horizontally does not change the corresponding pixel coordinates of the input image by more than 1. **pamperspective** determines the height of the output image analogously. The default value is 1.

--proportion=*prop*

--ratio=*ratio*

Valid values for *prop* are:

free

In this case **--ratio** does not have any effect.

fixed After the width and height are determined according to **--detail**, one of both will be increased, in order to obtain width/height=*ratio*.

The defaults are **--proportion**=**free** and **--ratio**=1.

Output options

--output_system=*spec*

The output image consists of pixels, which are, from the point of view of a scanline renderer, solid squares. This option specifies how the four vertices of the quadrilateral correspond to the pixels of the output image. Valid values for *spec* are:

lattice

The upper left vertex corresponds to the upper left corner of the upper left pixel and The lower right vertex corresponds to the lower right corner of the lower right pixel.

pixel

The upper left vertex corresponds to the centre of the upper left pixel and The lower right vertex corresponds to the centre of the lower right pixel.

The default value is **lattice**. Point-and-click front ends should use **pixel**.

--interpolation=*spec*

Usually (centres of) output pixels do not exactly correspond to (centres of) input pixels. This option determines how the program will choose the new pixels. Valid values for *spec* are:

nearest

The output pixel will be identical to the nearest input pixel.

linear

The output pixel will be a bilinear interpolation of the four surrounding input pixels.

The default value is **nearest**.

HINTS

It might be tempting always to use the options **--include 0,0;0,1;1,0;1,1** (assuming **--input_system=lattice** and **--input_unit=image**), so that no part of the input image is missing in the output. There are problems with that:

- If the threedimensional plane defined by the quadrilateral has a visible horizon in the input image, then the above asks **pamperspective** to include points that cannot ever be part of the output.

- If the horizon is not visible, but close to the border of the input image, this may result in *very* large output files. Consider a picture of a road. If you ask a point close to the horizon to be included, then this point is far away from the viewer. The output will cover many

kilometers of road, while **--detail** perhaps makes a pixel equal to a square centimeter.

When working with large files **pamperspective**'s memory usage might be an issue. In order to keep it small, you should minimize each of the following:

- The vertical range that the top output line consumes in the input image;
- The vertical range that the bottom output line consumes in the input image;
- The vertical range from the topmost (with respect to the input image) quadrilateral point to the top (with respect to the output image) output line.

For this purpose you can use **pamflip** before and/or after **pamperspective**. Example: Instead of

pamperspective 10 0 100 50 0 20 95 100 infile > outfile

you can use

pamflip -rotate90 infile |
pamperspective 50 0 100 5 0 90 20 100 |
pamflip -rotate270 > outfile

SEE ALSO

netpbm(1), **pam**(1), **pnm**(1), **pamcut**(1), **pamflip**(1), **pnmrotate**(1), **pamscale**(1), **pnmshear**(1), **pnmstitch**(1)

HISTORY

Mark Weyer wrote **pamperspective** in March 2004.

It was new in Netpbm 10.22 (April 2004).

AUTHOR

Table Of Contents

Pamperspective User Manual(0)

- HISTORY
- AUTHOR

Table Of Contents

NAME

pampop9 - simulate a multi-lens camera such as the Pop9

SYNOPSIS

pampop9 *pnmfile*|- *xtiles ytiles xdelta ydelta*

DESCRIPTION

This program is part of **Netpbm**(1).

pampop9 tiles your starting image *xtiles* by *ytiles* times. Each of these tiles is taken from a slightly different offset within the source, as determined by the *xdelta* and *ydelta* arguments.

The top line of tiles in the output is taken from the top of the source image, the second starts with the *ydelta* row of the source image, the next with the 2**ydelta* row, and so on. Similarly, the first column of tiles in the output is taken from the left of the source image, the next starts with the *xdelta* column, the next with the 2**xdelta* column, and so on.

EXAMPLES

With a 100 x 100 source image, then the output image from **pampop9** *3 3 10 10* will appear to be a three-by-three grid, each tile being 80 x 80 pixels. If the origin is taken to be the top-left corner, then the top row of tiles will take start at (0, 0) (10, 0) (20, 0) within the source image, the middle row will start at (0, 10) (10, 10) (20, 10), and the bottom row at (0, 20) (10, 20) (20, 20).

HISTORY

pampop9 was new in Netpbm 10.15 (April 2003). It was adapted slightly from **pampup9**, which was distributed by Robert Tinsley. At that time, it was distributed via http://www.thep-oacher.net/code/#pampup9. By October 2004, that URL no longer was valid.

SEE ALSO

pnmtile(1)

AUTHOR

Table Of Contents

NAME

pamscale - scale a Netpbm image

SYNOPSIS

pamscale
[
scale_factor
|
{**-xyfit** | **-xyfill** | **-xysize**} *cols rows*
|
-reduce *reduction_factor*
|
[**-xsize**=*cols* | **-width**=*cols* | **-xscale**=*factor*]
[**-ysize**=*rows* | **-height**=*rows* | **-yscale**=*factor*]
|
-pixels *n*
]
[
[**-verbose**]
[
-nomix
|
-filter=*functionName* [**-window**=functionName]
]
[**-linear**]
}
[*pnmfile*]

OPTION USAGE

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamscale scales a Netpbm image by a specified factor, or scales individually horizontally and vertically by specified factors.

You can either enlarge (scale factor > 1) or reduce (scale factor < 1).

The Scale Factors

When you specify an absolute size or scale factor for both dimensions, **pamscale** scales each dimension independently without consideration of the aspect ratio.

If you specify one dimension as a pixel size and don't specify the other dimension, **pamscale** scales the unspecified dimension to preserve the aspect ratio.

If you specify one dimension as a scale factor and don't specify the other dimension, **pamscale** leaves the unspecified dimension unchanged from the input.

If you specify the *scale_factor* parameter instead of dimension options, that is the scale factor for both dimensions. It is equivalent to **-xscale**=*scale_factor* **-yscale**=*scale_factor*.

Specifying the **-reduce** *reduction_factor* option is equivalent to specifying the *scale_factor* parameter, where *scale_factor* is the reciprocal of *reduction_factor*.

-xyfit specifies a bounding box. **pamscale** scales the input image to the largest size that fits within the box, while preserving its aspect ratio. **-xysize** is a synonym for this. Before Netpbm 10.20 (January 2004), **-xyfit** did not exist, but **-xysize** did.

-xyfill is similar, but **pamscale** scales the input image to the smallest size that completely fills the box, while preserving its aspect ratio. This option has existed since Netpbm 10.20 (January 2004).

-pixels specifies a maximum total number of output pixels. **pamscale** scales the image down to that number of pixels. If the input image is already no more than that many pixels, **pamscale** just copies it as output; **pamscale** does not scale up with **-pixels**.

If you enlarge by a factor of 3 or more, you should probably add a *pnmsmooth* step; otherwise, you can see the original pixels in the resulting image.

Usage Notes

A useful application of **pamscale** is to blur an image. Scale it down (without **-nomix)** to discard some information, then scale it back up using **pamstretch**.

Or scale it back up with **pamscale** and create a 'pixelized' image, which is sort of a computer-age version of blurring.

Transparency

pamscale understands transparency and properly mixes pixels considering the pixels' transparency.

Proper mixing *does not* mean just mixing the transparency value and the color component values separately. In a PAM image, a pixel which is not opaque represents a color that contains light of the foreground color indicated explicitly in the PAM and light of a background color to be named later. But the numerical scale of a color component sample in a PAM is as if the pixel is opaque. So a pixel that is supposed to contain half-strength red light for the foreground plus some light from the background has a red color sample that says *full* red and a transparency sample that says 50% opaque. In order to mix pixels, you have to first convert the color sample values to numbers that represent amount of light directly (i.e. multiply by the opaqueness) and after mixing, convert back (divide by the opaqueness).

Input And Output Image Types

pamscale produces output of the same type (and tuple type if the type is PAM) as the input, except if the input is PBM. In that case, the output is PGM with maxval 255. The purpose of this is to allow meaningful pixel mixing. Note that there is no equivalent exception when the input is PAM. If the PAM input tuple type is BLACKANDWHITE, the PAM output tuple type is also BLACKANDWHITE, and you get no meaningful pixel mixing.

If you want PBM output with PBM input, use **pamditherbw** to convert **pamscale**'s output to PBM. Also consider **pbmreduce**.

pamscale's function is essentially undefined for PAM input images that are not of tuple type RGB, GRAYSCALE, BLACKANDWHITE, or the _ALPHA variations of those. (By standard Netpbm backward compatibility, this includes PBM, PGM, and PPM images).

You might think it would have an obvious effect on other tuple types, but remember that the aforementioned tuple types have gamma-adjusted sample values, and **pamscale** uses that fact in its calculations. And it treats a transparency plane different from any other plane.

pamscale does not simply reject unrecognized tuple types because there's a possibility that just by coincidence you can get useful function out of it with some other tuple type and the right combination of options (consider **-linear** in particular).

Methods Of Scaling

There are numerous ways to scale an image. **pamscale** implements a bunch of them; you select among them with invocation options.

Pixel Mixing

Pamscale's default method is pixel mixing. To understand this, imagine the source image as composed of square tiles. Each tile is a pixel and has uniform color. The tiles are all the same size. Now lay over

that a transparent sheet of the same size, marked off in a square grid. Each cell in the grid stands for a pixel of the target image. For example, if you are scaling a 100x200 image up by 1.5, the source image is 100 x 200 tiles, and the transparent sheet is marked off in 150 x 300 cells.

Each cell covers parts of multiple tiles. To make the target image, just color in each cell with the color which is the average of the colors the cell covers -- weighted by the amount of that color it covers. A cell in our example might cover 4/9 of a blue tile, 2/9 of a red tile, 2/9 of a green tile, and 1/9 of a white tile. So the target pixel would be somewhat unsaturated blue.

When you are scaling up or down by an integer, the results are simple. When scaling up, pixels get duplicated. When scaling down, pixels get thrown away. In either case, the colors in the target image are a subset of those in the source image.

When the scale factor is weirder than that, the target image can have colors that didn't exist in the original. For example, a red pixel next to a white pixel in the source might become a red pixel, a pink pixel, and a white pixel in the target.

This method tends to replicate what the human eye does as it moves closer to or further away from an image. It also tends to replicate what the human eye sees, when far enough away to make the pixelization disappear, if an image is not made of pixels and simply stretches or shrinks.

Discrete Sampling

Discrete sampling is basically the same thing as pixel mixing except that, in the model described above, instead of averaging the colors of the tiles the cell covers, you pick the one color that covers the most area.

The result you see is that when you enlarge an image, pixels get duplicated and when you reduce an image, some pixels get discarded.

The advantage of this is that you end up with an image made from the same color palette as the original. Sometimes that's important.

The disadvantage is that it distorts the picture. If you scale up by 1.5 horizontally, for example, the even numbered input pixels are doubled in the output and the odd numbered ones are copied singly. If you have a bunch of one pixel wide lines in the source, you may find that some of them stretch to 2 pixels, others remain 1 pixel when you enlarge. When you reduce, you may find that some of the lines disappear completely.

You select discrete sampling with **pamscale**'s **-nomix** option.

Actually, **-nomix** doesn't do exactly what I described above. It does the scaling in two passes - first horizontal, then vertical. This can produce slightly different results.

There is one common case in which one often finds it burdensome to have **pamscale** make up colors that weren't there originally: Where one is working with an image format such as GIF that has a limited number of possible colors per image. If you take a GIF with 256 colors, convert it to PPM, scale by .625, and convert back to GIF, you will probably find that the reduced image has way more than 256 colors, and therefore cannot be converted to GIF. One way to solve this problem is to do the reduction with discrete sampling instead of pixel mixing. Probably a better way is to do the pixel mixing, but then color quantize the result with **pnmquant** before converting to GIF.

When the scale factor is an integer (which means you're scaling up), discrete sampling and pixel mixing are identical -- output pixels are always just N copies of the input pixels. In this case, though, consider using **pamstretch** instead of **pamscale** to get the added pixels interpolated instead of just copied and thereby get a smoother enlargement.

pamscale's discrete sampling is faster than pixel mixing, but **pnmenlarge** is faster still. **pnmenlarge** works only on integer enlargements.

discrete sampling (**-nomix**) was new in Netpbm 9.24 (January 2002).

Resampling

Resampling assumes that the source image is a discrete sampling of some original continuous image. That is, it assumes there is some non-pixelized original image and each pixel of the source image is simply the color of that image at a particular point. Those points, naturally, are the intersections of a

square grid.

The idea of resampling is just to compute that original image, then sample it at a different frequency (a grid of a different scale).

The problem, of course, is that sampling necessarily throws away the information you need to rebuild the original image. So we have to make a bunch of assumptions about the makeup of the original image.

You tell **pamscale** to use the resampling method by specifying the **-filter** option. The value of this option is the name of a function, from the set listed below.

To explain resampling, we are going to talk about a simple one dimensional scaling -- scaling a single row of grayscale pixels horizontally. If you can understand that, you can easily understand how to do a whole image: Scale each of the rows of the image, then scale each of the resulting columns. And scale each of the color component planes separately.

As a first step in resampling, **pamscale** converts the source image, which is a set of discrete pixel values, into a continuous step function. A step function is a function whose graph is a staircase-y thing.

Now, we convolve the step function with a proper scaling of the filter function that you identified with **-filter**. If you don't know what the mathematical concept of convolution (convolving) is, you are officially lost. You cannot understand this explanation. The result of this convolution is the imaginary original continuous image we've been talking about.

Finally, we make target pixels by picking values from that function.

To understand what is going on, we use Fourier analysis:

The idea is that the only difference between our step function and the original continuous function (remember that we constructed the step function from the source image, which is itself a sampling of the original continuous function) is that the step function has a bunch of high frequency Fourier components added. If we could chop out all the higher frequency components of the step function, and know that they're all higher than any frequency in the original function, we'd have the original function back.

The resampling method *assumes* that the original function was sampled at a high enough frequency to form a perfect sampling. A perfect sampling is one from which you can recover exactly the original continuous function. The Nyquist theorem says that as long as your sample rate is at least twice the highest frequency in your original function, the sampling is perfect. So we *assume* that the image is a sampling of something whose highest frequency is half the sample rate (pixel resolution) or less. Given that, our filtering does in fact recover the original continuous image from the samples (pixels).

To chop out all the components above a certain frequency, we just multiply the Fourier transform of the step function by a rectangle function.

We could find the Fourier transform of the step function, multiply it by a rectangle function, and then Fourier transform the result back, but there's an easier way. Mathematicians tell us that multiplying in the frequency domain is equivalent to convolving in the time domain. That means multiplying the Fourier transform of F by a rectangle function R is the same as convolving F with the Fourier transform of R. It's a lot better to take the Fourier transform of R, and build it into **pamscale** than to have **pamscale** take the Fourier transform of the input image dynamically.

That leaves only one question: What *is* the Fourier transform of a rectangle function? Answer: sinc. Recall from math that sinc is defined as sinc(x) = sin(PI*x)/PI*x.

Hence, when you specify **-filter=sinc**, you are effectively passing the step function of the source image through a low pass frequency filter and recovering a good approximation of the original continuous image.

<h5>Refiltering</h5>

There's another twist: If you simply sample the reconstructed original continuous image at the new sample rate, and that new sample rate isn't at least twice the highest frequency in the original continuous image, you won't get a perfect sampling. In fact, you'll get something with ugly aliasing in it. Note that this can't be a problem when you're scaling up (increasing the sample rate), because the fact that the old sample rate was above the Nyquist level means so is the new one. But when scaling down, it's a problem. Obviously, you have to give up image quality when scaling down, but aliasing is not the best way to do it. It's better just to remove high frequency components from the original continuous

image before sampling, and then get a perfect sampling of that.

Therefore, **pamscale** filters out frequencies above half the new sample rate before picking the new samples.

<h5>Approximations</h5>

Unfortunately, **pamscale** doesn't do the convolution precisely. Instead of evaluating the filter function at every point, it samples it -- assumes that it doesn't change any more often than the step function does. **pamscale** could actually do the true integration fairly easily. Since the filter functions are built into the program, the integrals of them could be too. Maybe someday it will.

There is one more complication with the Fourier analysis. sinc has nonzero values on out to infinity and minus infinity. That makes it hard to compute a convolution with it. So instead, there are filter functions that approximate sinc but are nonzero only within a manageable range. To get those, you multiply the sinc function by a *window function*, which you select with the **-window** option. The same holds for other filter functions that go on forever like sinc. By default, for a filter that needs a window function, the window function is the Blackman function.

<h5>Filter Functions Besides Sinc</h5>

The math described above works only with sinc as the filter function. **pamscale** offers many other filter functions, though. Some of these approximate sinc and are faster to compute. For most of them, I have no idea of the mathematical explanation for them, but people do find they give pleasing results. They may not be based on resampling at all, but just exploit the fact the convolution that is coincidentally part of a resampling calculation.

For some filter functions, you can tell just by looking at the convolution how they vary the resampling process from the perfect one based on sinc:

The impulse filter assumes that the original continuous image is in fact a step function -- the very one we computed as the first step in the resampling. This is mathematically equivalent to the discrete sampling method.

The box (rectangle) filter assumes the original image is a piecewise linear function. Its graph just looks like straight lines connecting the pixel values. This is mathematically equivalent to the pixel mixing method when scaling down, and interpolation (ala **pamstretch**) when scaling up.

<h5>Gamma</h5>

pamscale assumes the underlying continuous function is a function of brightness (as opposed to light intensity), and therefore does all this math using the gamma-adjusted numbers found in a PNM or PAM image. The **-linear** option is not available with resampling (it causes **pamscale** to fail), because it wouldn't be useful enough to justify the implementation effort.

Resampling (**-filter**) was new in Netpbm 10.20 (January 2004).

<h5>The filter functions</h5>

Here is a list of the function names you can specify for the **-filter** option. For most of them, you're on your own to figure out just what the function is and what kind of scaling it does. These are common functions from mathematics.

point The graph of this is a single point at X=0, Y=1.

box The graph of this is a rectangle sitting on the X axis and centered on the Y axis with height 1 and base 1.

triangle The graph of this is an isosceles triangle sitting on the X axis and centered on the Y axis with height 1 and base 2.

quadratic

cubic

catrom

mitchell

gauss

sinc

bessel

hanning

hamming

blackman

kaiser

normal

hermite

lanczos

Linear vs Gamma-adjusted

The pixel mixing scaling method described above involves intensities of pixels (more precisely, it involves individual intensities of primary color components of pixels). But the PNM and PNM-equivalent PAM image formats represent intensities with gamma-adjusted numbers that are not linearly proportional to intensity. So **pamscale**, by default, performs a calculation on each sample read from its input and each sample written to its output to convert between these gamma-adjusted numbers and internal intensity-proportional numbers.

Sometimes you are not working with true PNM or PAM images, but rather a variation in which the sample values are in fact directly proportional to intensity. If so, use the **-linear** option to tell **pamscale** this. **pamscale** then will skip the conversions.

The conversion takes time. In one experiment, it increased the time required to reduce an image by a factor of 10. And the difference between intensity-proportional values and gamma-adjusted values may be small enough that you would barely see a difference in the result if you just pretended that the gamma-adjusted values were in fact intensity-proportional. So just to save time, at the expense of some image quality, you can specify **-linear** even when you have true PPM input and expect true PPM output.

For the first 13 years of Netpbm's life, until Netpbm 10.20 (January 2004), **pamscale**'s predecessor **pnmscale** always treated the PPM samples as intensity-proportional even though they were not, and drew few complaints. So using **-linear** as a lie is a reasonable thing to do if speed is important to you. But if speed is important, you also should consider the **-nomix** option and **pnmscalefixed**.

Another technique to consider is to convert your PNM image to the linear variation with **pnmgamma**, run **pamscale** on it and other transformations that like linear PNM, and then convert it back to true PNM with **pnmgamma -ungamma**. **pnmgamma** is often faster than **pamscale** in doing the conversion.

With **-nomix**, **-linear** has no effect. That's because **pamscale** does not concern itself with the meaning of the sample values in this method; **pamscale** just copies numbers from its input to its output.

Precision

pamscale uses floating point arithmetic internally. There is a speed cost associated with this. For some images, you can get the acceptable results (in fact, sometimes identical results) faster with **pnmscalefixed**, which uses fixed point arithmetic. **pnmscalefixed** may, however, distort your image a little. See the **pnmscalefixed** user manual for a complete discussion of the difference.

SEE ALSO

pnmscalefixed(1), **pamstretch**(1), **pamditherbw**(1), **pbmreduce**(1), **pbmpscale**(1), **pnmenlarge**(1), **pnmsmooth**(1), **pamcut**(1), **pnmgamma**(1), **pnmscale**(1), **pnm**(1), **pam**(1)

HISTORY

pamscale was new in Netpbm 10.20 (January 2004). It was adapted from, and obsoleted, **pnmscale**. **pamscale**'s primary difference from **pnmscale** is that it handles the PAM format and uses the "pam" facilities of the Netpbm programming library. But it also added the resampling class of scaling method. Furthermore, it properly does its pixel mixing arithmetic (by default) using intensity-proportional values instead of the gamma-adjusted values the **pnmscale** uses. To get the old **pnmscale** arithmetic, you can specify the **-linear** option.

The intensity proportional stuff came out suggestions by *Adam M Costello* in January 2004.

The resampling algorithms are mostly taken from code contributed by *Michael Reinelt* in December 2003.

The version of **pnmscale** from which **pamscale** was derived, itself evolved out of the original Pbmplus version of **pnmscale** by Jef Poskanzer (1989, 1991). But none of that original code remains.

Table Of Contents

NAME

pamseq - generate PAM image of all possible tuple values, in sequence

SYNOPSIS

pamseq [**-tupletype** *tupletype*] *depth maxval*

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamseq generates a PAM image of a specified depth and specified maxval that consists of a single row. The row consists of one tuple of every possible value, in order.

For a depth of one, the order is simple: From 0 to maxval, going from left to right. For higher depths, the highest numbered plane goes from 0 to maxval (going left to right) while all the other planes have value 0. Then the sequence repeats except with the next highest plane set to a value of 1, then 2, etc.

OPTIONS

-tupletype

This is the value of the "tuple_type" attribute of the created PAM image. It can be any string up to 255 characters.

USAGE

To create a simple ramp of the values 0..255, for input to various matrix calculations, try

```
pamseq 1 255
```

(Before **pamseq** existed, **pgmramp** was often pressed into service for this).

To create a PPM color map of all the possible colors representable with a maxval of 5, do

```
pamseq 3 5 -tupletype=RGB | pamtopnm
```

Again, with a modern program based on the Netpbm library, you don't need the **pamtopnm** because a PAM RGB image is equivalent to a PPM image.

You can use such a color map with **pnmremap**(1)**to**quantize**the**colors**in**an image. With the maxval of 5 given in the example, you get a color map of the set of "web safe" colors as defined by Netscape. Most web browsers guarantee that they can produce at least these 216 colors (215 plus black).

SEE ALSO

pnmremap(1), **pamtopnm**(1), **pam**(1)

HISTORY

pamseq was added to Netpbm in June 2002.

Table Of Contents

NAME

pamsharpmap - create map of sharpness in a PNM/PAM image

SYNOPSIS

pamsharpmap [*imagefile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamsharpmapreads a Netpbm image (PNM or PAM) and produces an image that shows how sharp it is at each location.

Sharpness is a measure of how suddenly (in space) colors change in the image. **pamsharpmap** computes the sharpness of each component color (R, G, B) separately and produces a pixel whose intensity of each component color is directly proportional to the sharpness of that component color at the same location in the input image. Thus, at point where the image is very sharp in its red and green components are very sharp, but dull in its blue component, you will see a yellow pixel in the output.

pamsharpmap computes sharpness at a point simply as the average difference in intensity between the pixel at that point and the 8 pixels surrounding it.

At the edges of the image, where there are not 8 pixels surrounding a pixel, **pamsharpmap** assumes the image is extended with a black border.

pamsharpmap assumes that the image is a PNM or PNM equivalent PAM. If it isn't, the results are not necessarily meaningful.

The output image is the same dimensions, depth, and tuple type as the input, and has maxval 255.

SEE ALSO

pamsharpness(1), **pamedge**(1), **pam**(1), **pnm**(1)

HISTORY

pamsharpmap was added to Netpbm in Release 10.21 (March 2004). Bryan Henderson derived it from the program **pnmsharp** by B.W. van Schooten and distributed as part of the Photopnmtools package.

Table Of Contents

NAME

pamsharpness - measure the sharpness of a PNM/PAM image

SYNOPSIS

pamsharpness [*imagefile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamsharpnessreads a Netpbm image (PNM or PAM) and prints a number that tells how sharp it is.

Sharpness is a measure of how suddenly (in space) colors change in the image. **pamsharpness** computes the sharpness of the image as the average difference in intensity between each pixel and its 8 surrounding pixels, in each of the color components (R, G, B).

pamsharpness does not include the edges of the image, where there are not 8 pixels surrounding a pixel, in its computation.

pamsharpness assumes that the image is a PNM or PNM equivalent PAM. If it isn't, the results are not necessarily meaningful.

SEE ALSO

pamsharpmap(1), **pam**(1), **pnm**(1)

HISTORY

pamsharpness was added to Netpbm in Release 10.21 (March 2004). Bryan Henderson derived it from the program **pnmsharp** by B.W. van Schooten and distributed as part of the Photopnmtools package.

Table Of Contents

NAME

pamslice - extract one line of values out of a Netpbm image

SYNOPSIS

pamslice {**-row=***rownumber* | **-column=***columnnumber*} [**-plane=***planenumber*] [*imagefile*]

OPTION USAGE

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamslice extracts one line of tuples (pixels) out of a Netpbm image and prints their values in a table. A line means a row or column. It shows you a one-dimensional cross section of a two-dimensional image. (With the **-plane** option, it can be thought of as a one-dimensional cross-section of a three-dimensional image).

The table has one line per tuple, consisting of blank-separated ASCII decimal numbers. The first number is the column number if you specified a row slice or the row number if you specified a column slice. The rest of the numbers are the sample values in plane number order. For a PBM or PGM input, there is only one plane. For a PPM input, Plane 0 is red, Plane 1 is green, and Plane 2 is blue. See the specifications of the image formats for details on exactly what these numbers mean.

If you want to see all the pixels in a PPM, PGM, or PBM image in ASCII decimal, **pnmtoplainpnm** is a good way to do that.

OPTIONS

-row=*rownumber*

This indicates that the slice is to be horizontal -- i.e. one row of the image -- and indicates which row. Rows are numbered from the top starting with 0.

You cannot specify both **-row** and **-column**.

-column=*colnumber*

This indicates that the slice is to be vertical -- i.e. one column of the image -- and indicates which column. Columns are numbered from the left starting with 0.

You cannot specify both **-row** and **-column**.

-plane=*planenumber*

This specifies that you are interested in only one plane of the image and which one. Planes are numbered from 0 and have meanings that vary on the type of image. In a PPM image, Plane 0 is red, Plane 1 is green, and Plane 2 is blue.

If you don't specify **-plane**, you get all the planes -- each line of output has multiple numbers in addition to the sequence number. If you do specify **-plane**, each line of output contains one number in addition to the sequence number.

-xmgr This option causes **pamslice** to format the output as input for a **xmgr** so you can plot it. The only difference this option makes

is that it adds header information to the beginning of the output.

SEE ALSO

pamcut(1) **pnmtoplainpnm**(1) **pnmtoplainpnm**(1) **pnm**(1)

HISTORY

pamslice replaced **pgmslice** in Netpbm 10.3 (June 2002). It was backward compatible, but worked on Netpbm images other than PGM and PBM and added the **-plane** and **-xmgr** options.

AUTHOR

Jos Dingjan <*jos@tuatha.org*> wrote **pgmslice** after being unable to find the source code to Marco Beijersbergen's program with the same name. Bryan Henderson converted it to **pamslice**.

Table Of Contents

NAME

pamstack - stack planes of multiple PAM images into one PAM image

SYNOPSIS

pamstack [**-tupletype** *tupletype*] [*inputfilespec* ...]

All options may be abbreviated to the shortest unique prefix. You may use two hyphens instead of one. You may separate an option from its value with a space instead of =.

DESCRIPTION

This program is part of **Netpbm**(1).

pamstack reads multiple PAM or PNM images as input and produces a PAM image as output, consisting of all the planes (channels) of the inputs, stacked in the order specified.

The output is the same dimensions as the inputs, except that the depth is the sum of the depths of the inputs. It has the same maxval. **pamstack** fails if the inputs are not all the same width, height, and maxval. The tuple type is a null string unless you specify the **-tupletype** option.

pamchannel does the opposite of **pamstack**: It extracts individual planes from a single PAM.

Use **pamtopnm**(1)**to**convert**a**suitable**PAM** image to a more traditional PNM (PBM, PGM, or PPM) image.

One example of using **pamstack** is that some Netpbm programs accept as input a PAM that represents graphic image with transparency information. Taking a color image for example, this would be a PAM with tupel type "RGB_ALPHA". In Netpbm, such images were traditionally represented as two images - a PPM for the color and a PGM for the transparency. To convert a PPM/PGM pair into PAM(RGB_ALPHA) input that newer programs require, do something like this:

```
$ pamstack -tupletype=RGB_ALPHA myimage.ppm myalpha.pgm |       pamtouil >myimage.ui
```

OPTIONS

-tupletype *tupletype*

This specified the tuple type name to be recorded in the output. You may use any string up to 255 characters. Some programs recognize some names. If you omit this option, the default tuple type name is null.

SEE ALSO

pam(1) **pamchannel**(1)

Table Of Contents

NAME

pamstereogram - create a PAM single-image stereogram from a PAM height map

SYNOPSIS

pamstereogram [**-help**] [**-verbose**] [**-pbm** | **-pgm** | **-ppm**] [**-maxval** *value*] [**-patfile** *pnmfile*] [**-pam**] [**-xshift** *pixels*] [**-yshift** *pixels*] [**-magnifypat** *scale*] [**-guidesize** *pixels*] [**-dpi** *resolution*] [**-crosseyed**] [**-makemask**] [**-eyesep** *inches*] [**-depth** *fraction*] [*infile*]

OPTION USAGE

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamstereogram inputs a height map (a map of the distances from your eye of the points in a scene) and outputs a single-image stereogram (SIS). A SIS is a 2-D image specially designed to appear three dimensional when viewed with relaxed, slightly unfocused eyes. What's exciting about single-image stereograms is that they don't require special glasses to view, although it does require a bit of practice to train your eyes to unfocus properly. **pamstereogram** program provides a wealth of control over how the stereogram is generated, including the following:

- black and white, grayscale, or color output
- single-image random-dot stereograms (SIRDS) or single-image stereograms (SIS) using a tiled image
- images targeting a given device resolution and eye separation
- optional guide boxes to assist in focusing
- the ability to trade off depth levels for easier viewing
- choice of ordinary or cross-eyed stereograms

The output is a PAM image on Standard Output. Options control the exact format of the PAM. If you want a PNM (PBM, PGM, or PPM) image, use **pamtopnm** on the output.

To make a red/green type of stereogram (that you view with 3-D glasses) instead, see **ppm3d**x

OPTIONS

-verbose

Display messages about image sizes and formats and properties of the stereogram being generated.

-blackandwhite

Produce a single-image random-dot black and white stereogram. This is the default.

-grayscale
Produce a single-image random-dot grayscale stereogram.

-color Produce a single-image random-dot color stereogram.

-maxval=*value*
Designate the maximum value of each gray/color component, i.e. the color resolution. Smaller values make the output image have smaller numbers of unique grays/colors. If you don't specify **-maxval**, **pamstereogram** uses the maxval of the input image. This option has no effect with **-blackandwhite**.

-patfile=*pnmfile*
Specify an image to use as a repeated background pattern for the stereogram instead of a random-dot pattern. Intricate images generally produce a crisper 3-D effect that simpler images. The output file will have the same maxval and format (black and white, grayscale or color) as the pattern file. You cannot specify the **-patfile** option along with **-blackandwhite**, **-grayscale**, **-color**, or **-maxval**.

-xshift=*pixels*
Shift the pattern image (designated by **-patfile**) to the right by *pixels* pixels (default: 0). **-xshift** is helpful when creating "true-color" stereograms. This option is valid only along with **-patfile**.

-yshift *pixels*
Shift the pattern image (designated by **-patfile**) downwards by *pixels* pixels (default: 0). This option is valid only along with **-patfile**.

-magnifypat=*scale*
Magnify each pixel in the pattern file or each random dot by integral scaling factor *scale*. Note that **pamstereogram** applies the pattern magnification *after* pattern shifting (**-xshift** and **-yshift**).

-guidesize=*pixels*
Draw a pair of *pixels* by *pixels* black squares on a white background underneath the stereogram proper. These squares help you guide your eyes into proper focus to view the 3-D image. The trick is to focus your eyes some distance behind the image, causing you to see four black squares, then continue altering your focus distance until the middle two black squares fuse into a single black square. At that point, a crisp, 3-D image will appear.

If *pixels* is negative, **pamstereogram** will draw the guide squares above the stereogram instead of below it. If *pixels* is zero (the default), **pamstereogram** will draw no guide squares.

-dpi=*resolution*
Specify the resolution of the output device in dots per inch. The default is 96 DPI, which represents a fairly crisp screen resolution.

-crosseyed
Invert the gray levels in the height map (input image) so that the 3-D image pops out of the page where it would otherwise sink into the page and vice versa. Some people are unable to diverge their eyes and can only cross them. The **-crosseyed** option enables such people to see the 3-D image as intended.

-makemask
Instead of a stereogram, output a PAM mask image showing coloring constraints. New pixels will be taken from the pattern file where the mask is black. Copies of existing pixels will be

taken from the pattern file where the mask is white. The **-makemask** option can be used to help create more sophisticated pattern files (to use with **-patfile**) Note that **-makemask** ignores **-magnifypat**; it always produces masks that assume a pattern magnification of 1.

-eyesep=*inches*

Specify the separation in inches between your eyes. The default, 2.5 inches (6.4 cm), should be sufficient for most people and probably doesn't need to be changed.

-depth=*fraction*

Specify the output image's depth of field. That is, *fraction* represents the fractional distance of the near plane from the far plane. Smaller numbers make the 3-D image easier to perceive but flatter. Larger numbers make the 3-D image more difficult to perceive but deeper. The default, 0.3333, generally works fairly well.

PARAMETERS

The only parameter, *infile*, is the name of an input file that is a height map image. If you don't specify *infile*, the input is from Standard Input.

The input is a PAM image of depth 1. Each sample represents the distance from the eye that the 3-D image at that location should be. Higher numbers mean further from the eye.

pamstereogram pays no attention the the image's tuple type and ignores all planes other than Plane 0.

Like any Netpbm program, **pamstereogram** will accept PNM input as if it were the PAM equivalent.

A good initial test is to input an image consisting of a solid image of distance 0 within a large field of maximum distance.

EXAMPLES

Generate a SIRDS out of small, brightly colored squares and prepare it for display on an 87 DPI monitor:

```
pamstereogram heightmap.pam -dpi 87 -verbose -color -maxval 1 -magnifypat 3 >3d.pam
```

Generate a SIS by tiling a PPM file (a prior run with **-verbose** indicates how wide the pattern file should be for seamless tiling, although any width is acceptable for producing SISes):

```
pamstereogram myheights.pam -patfile mypattern.ppm >mysis.pam
```

SEE ALSO

- **pam**(1)
- **ppm3d**(1)
- Harold W. Thimbleby, Stuart Inglis, and Ian H. Witten. *Displaying 3D Images: Algorithms for Single Image Random Dot Stereograms*. In IEEE Computer, **27**(10):38-48, October 1994.

HISTORY

pamstereogram was new in Netpbm 10.22 (April 2004).

AUTHOR

Table Of Contents

Table Of Contents

NAME

pamstretch-gen - use pamstretch and pamscale to scale by non-integer values

SYNOPSIS

pamstretch-gen

N

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pamstretch-gen is a program which uses **pamstretch**(1), **pnmfile**(1),**and pam-scale**(1)**to**smoothly**scale**up**a**PNM**file** by any ratio; it's like a more general version of pamstretch (hence the name). But other than the 'any ratio' bit, it's much the same as pamstretch. :-)

LIMITATIONS

The program uses **awk** just to make some simple floating-point calculations, which is probably overkill. But using **dc** makes my head hurt.

SEE ALSO

pamstretch(1), **pamscale**(1)

AUTHOR

Russell Marks (*russell.marks@ntlworld.com*).

Table Of Contents

NAME

pamstretch - scale up a PNM or PAM image by interpolating between pixels.

SYNOPSIS

pamstretch

[**-xscale=***X*]

[**-yscale=***Y*] [**-blackedge**]

[**-dropedge**]

N

[*infile*]

You can use the minimum unique abbreviation of the options. You can use two hyphens instead of one. You can separate an option name from its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamstretch scales up pictures by integer values, either vertically, horizontally, or both. **pamstretch** differs from **pamscale** and **pnmenlarge** in that when it inserts the additional rows and columns, instead of making the new row or column a copy of its neighbor, **pamstretch** makes the new row or column an interpolation between its neighbors. In some images, this produces better looking output.

To scale up to non-integer pixel sizes, e.g. 2.5, try **pamstretch-gen**(1)**instead.**

Options let you select alternative methods of dealing with the right/bottom edges of the picture. Since the interpolation is done between the top-left corners of the scaled-up pixels, it's not obvious what to do with the right/bottom edges. The default behaviour is to scale those up without interpolation (more precisely, the right edge is only interpolated vertically, and the bottom edge is only interpolated horizontally), but there are two other possibilities, selected by the **blackedge** and **dropedge** options.

PARAMETERS

The *N* parameter is the scale factor. It is valid only if you *don't* specify **-xscale** or **-yscale**. In that case, **pamstretch** scales in both dimensions and by the scale factor *N*.

OPTIONS

-xscale=*X*

This is the horizontal scale factor. If you don't specify this, but do specify a vertical scale factor, the horizontal scale factor is 1.

-yscale=*Y*

This is the vertical scale factor. If you don't specify this, but do specify a horizontal scale factor, the vertical scale factor is 1.

-blackedge

interpolate to black at right/bottom edges.

-dropedge

drop one (source) pixel at right/bottom edges. This is arguably more logical than the default behaviour, but it means producing output which is a slightly odd size.

BUGS

Usually produces fairly ugly output for PBMs. For most PBM input you'll probably want to reduce the 'noise' first using something like **pnmnlfilt**(1).

SEE ALSO

pamstretch-gen(1), **pnmenlarge**(1), **pamscale**(1), **pnmnlfilt**(1)

AUTHOR

Russell Marks (*russell.marks@ntlworld.com*).

Table Of Contents

NAME

pamsumm - Sum the samples in a Netpbm image

SYNOPSIS

pamsumm { **-sum** | **-mean** | **-min** | **-max** } [**-normalize**] [**-brief**] [*imagefile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamsumm reads a Netpbm image (PNM or PAM) and performs a summary function over all the samples in all the rows, columns, and planes and prints the result to Standard Output.

pamsumm performs the operation on the actual sample values, not on the light intensities represented by them in the case that the image is a PGM or PPM image or PAM equivalent. If you want to do arithmetic on light intensities of such a visual image, you can use **pnmgamma** to convert it to one with samples proportional to light intensity, and then use **pamsumm** on the result.

If you want to summarize by column (e.g. add up the columns separately), use **pamsummcol**. If you want to summarize by row, use a combination of **pamsummcol** and **pamflip**. If you want to summarize a particular plane, use **pamchannel** to extract it and then **pamsumm**.

OPTIONS

You must specify exactly one of **-sum**, **-mean**, **-min**, or **-max**.

-sum

This option makes the summary function addition.

-mean

This option makes the summary function arithmetic mean.

-min

This option makes the summary function arithmetic minimum.

-max

This option makes the summary function arithmetic maximum.

-normalize

This option causes each sample to be normalized to a fraction (in the range 0..1) so the result is independent of the image's maxval. E.g. if you request the mean of an image which has maxval 200 and all the samples have value 50, **pamsumm** will give you 50 as an answer. But **pamsumm -normalize** will give you .25.

If you want a result that is independent of maxval but still in integers, and your input is PNM, you can use **pnmdepth** to

convert to some standard maxval. For example, if you want the mean intensity of a PPM image, on a scale of 0 to 99, do

```
pnmdepth 99 myimage.ppm | pamsumm -mean
```

This option was new in Netpbm 10.22 (April 2004)

-brief

This option causes **pamsumm** to display the answer as a bare number, rather than in a complete sentence.

This option was new in Netpbm 10.22 (April 2004)

SEE ALSO

pamsumm(1), **pam**(1),

HISTORY

pamsumm was added to Netpbm in Release 10.21 (March 2004).

Table Of Contents

NAME

pamsummcol - Sum the columns in a Netpbm image

SYNOPSIS

pamsummcol { **-sum** | **-mean** | **-min** | **-max** } [*imagefile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamsummcolreads a Netpbm image (PNM or PAM) and performs a summary function over all the rows in each column (sum, mean, etc.). It produces an image of the same kind that the same width and depth as the input, and one row high. Its sample values are the result of the summary.

pamsummcol performs the summary operation on each plane independently.

pamsummcol performs the operation on the actual sample values, not on the light intensities represented by them in the case that the image is a PGM or PPM image.

If you want to summarize by row instead of by column, run the input through **pamflip** first (and if you want the output to be a single column instead of a single row, use **pamflip** again).

If you want to summarize over the entire image instead of over columns separately, use **pamsumm**.

pamsummcol performs the operation on the actual sample values, not on the light intensities represented by them in the case that the image is a PGM or PPM image or PAM equivalent. You can use **pnmgamma** to convert such an image to one with samples proportional to light intensity, and then use **pamsummcol** on the result.

You can achieve the same thing as **pamsummcol -mean** with **pamscale**. Just scale vertically to a single row, without scaling horizontally at all. Use the pixel mixing method.

OPTIONS

You must specify exactly one of **-sum**, **-mean**, **-min**, or **-max**.

-sum

This option makes the summary function addition.
In each column and plane of the output row, the sample value is the sum of all the samples values in the same column and plane of the input. If a result is greater than the image maxval, it is clipped to the maxval.

-mean

This option makes the summary function arithmetic mean.
In each column and plane of the output row, the sample value is the mean of all the samples values in the same column and plane of the input.

-min

This option makes the summary function arithmetic minimum.
In each column and plane of the output row, the sample value is the minimum of all the samples values in the same column and plane of the input.

-max

This option makes the summary function arithmetic maximum. In each column and plane of the output row, the sample value is the maximum of all the samples values in the same column and plane of the input.

SEE ALSO

pamsumm(1), **pamscale**(1), **pamfunc**(1), **pamarith**(1), **pamscale**(1), **pam**(1),

HISTORY

pamsummcol was added to Netpbm in Release 10.21 (March 2004).

Table Of Contents

NAME

pamtodjvurle - convert a Netpbm image to DjVu Color RLE format

SYNOPSIS

pamtodjvurle

[**-transparent** *color*] [*netpbmfile* [*rlefile*]]

Mininum unique abbreviation of options in acceptable.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtodjvurle reads a Netpbm image (PNM or PAM equivalent of PNM) as input and produces DjVu Color RLE format as output.

OPTIONS

-transparent *colorname*

This option indicates which color in the image should be considered transparent.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

Default is 'white'.

SEE ALSO

pbmtodjvurle(1) **pam**(1)

HISTORY

pamtodjvurle was new in Netpbm 10.22 (April 2004) but a program that did almost the same thing, called **ppmtodjvurle**, was in Netpbm 10.21 (March 2004). The latter was written and contributed to Netpbm by Scott Pakin <scott+pbm@pakin.org>. **pamtodjvurle** uses techniques taken from **ppmtodjvurle**, but no code is copied between them.

Table Of Contents

NAME

pamtohdiff - convert PAM image to horizontal difference image

SYNOPSIS

pamtohdiff [*pamfile*] [**-verbose**]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphens to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtohdiff takes a PAM (or PNM) image as input and produced a horizontal difference image version of it as output. A horizontal difference image is one where the samples in each row indicate the difference between the sample value in the corresponding sample of the input image and the sample directly above it (in the previous row) in the input image. The horizontal difference image has the property that if a row of the original image is identical to the row above it over a long extent, the corresponding row in the horizontal difference image will contain all zeroes. That makes it compress better than the original image.

Because the horizontal difference samples can be positive or negative, but PAM samples are unsigned integers, the samples in the horizontal difference image PAM are defined to be the difference modulus the range of the input (maxval + 1). This doesn't lose any information, as it might seem, because: of the two differences that could result in the same **pamtohdiff** output value (e.g. if maxval is 99, +20 and -80 would both result in "20" in the output), only one is possible in context and the other would result, when reconstructing the original image, in a value less than 0 or greater than maxval.

Before the modulus operation, the values **pamtohdiff** computes are also biased by half the maxval. This is to make the results easier to inspect visually. Because of the bias, you can display the **pamtohdiff** output as if it were a PNM image. As long as none of your differences are more than half the maxval, large negative differences show up as dark spots, smaller negative differences are lighter, zero differences are medium intensity, and positive differences are light. If you want this to work even for images that have differences that exceed half the maxval, just use **ppmdim 50** on the original image. To avoid losing information, though, do a **pnmdepth** to double the maxval first.

Note that because of the transfer function just described, a difference of zero, which is most common, is represented by a PAM sample value in the output of one half the maxval.

The output PAM has a tuple type of "hdiff".

You can use **hdifftopam** to recover the original image from a horizontal difference image PAM.

SEE ALSO

hdifftopam(1), **pnmdepth**(1),

AUTHOR

Bryan Henderson

Table Of Contents

NAME

pamtohtmltbl - convert pnm/pam visual image to an HTML table

SYNOPSIS

pamtohtmltbl [**-transparent=***color*] [**-verbose**] [*file*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtohtmltbl converts a visual image to an HTML table with one cell per pixel. The cell is empty, but its background color is that of the pixel.

file is a PBM, PGM, PPM, or PAM file. If PAM, it must be a standard visual image of tuple type RGB, GRAYSCALE, or BLACKANDWHITE, or something equivalent with extra higher numbered channels, but **pamtohtmltbl** doesn't check the tuple type; it just assumes.

Note that the more normal way to include a visual image in an HTML document is with a <IMG> tag.

OPTIONS

-transparent=*color*

This option indicates that pixels of the specified color are to be transparent in the HTML table. The table cell for a pixel of this color will have no background color specified.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

-verbose

This option causes **pamtohtmltbl** to display messages about the conversion process.

SEE ALSO

pnm(1) **pam**(1)

HISTORY

pamtohtmltbl was new in Netpbm 10.15 (April 2003).

AUTHORS

Alexander B. Ivanov (SSH) wrote a program he called **pnm2html**. Bryan Henderson adapted it to use the Netpbm libraries and handle PAM images and follow Netpbm conventions, and named it **pamtohtmltbl**.

Table Of Contents

NAME

pamtojpeg2k - convert PAM/PNM image to a JPEG-2000 code stream

SYNOPSIS

pamtojpeg2k [**-imgareatlx**=*column*] [**-imgareatly**=*row*] [**-tilegrdtlx**=*column*] [**-tilegrdtly**=*row*] [**-tilewidth**=*columns*] [**-tileheight**=*rows*] [**-prcwidth**=*columns*] [**-prcheight**=*rows*] [**-cblkwidth**=*columns*] [**-cblkheight**=*rows*] [**-mode**={**integer**|**int**|**real**}] [**-compression**=*ratio*] [**-ilyrrates**=*ratestring*] [**-numrlvls**=*number*] [**-progression**={**lrcp**|**rlcp**|**rpcl**|**pcrl**|**cprl**}] [**-numgbits**=*number*] [**-nomct**] [**-sop**] [**-eph**] [**-lazy**] [**-termall**] [**-segsym**] [**-vcausal**] [**-pterm**] [**-resetprob**] [**-verbose**] [**-debuglevel**=*number*] *filename*

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtojpeg2k converts the named PBM, PGM, PPM, or PAM file, or Standard Input if no file is named, to a JPEG-2000 code stream (JPC) file on Standard Output.

The JPEG-2000 specification specifies two different formats: JP2 and JPEG-2000 code stream (JPC). JP2 represents a visual image quite specifically, whereas JPC is a more or less arbitrary array of codes. **pamtojpeg2k** can't produce a JP2, but the JPC image that **pamtojpeg2k** produces is very similar to a JP2 if the input is a PBM, PGM, or PPM image or equivalent PAM image. One difference is that that RGB intensity values in a JP2 are SRGB values, while **pamtojpeg2k** produces ITU Rec 709 values. Those are very similar, but not identical. Another difference is that a JP2 can contain extra information about an image that JPC cannot.

When the input is a PAM image other than a PBM, PGM, or PPM equivalent, the JPC raster produced contains whatever the PAM raster does. It can have any number of planes with any meanings; the planes are in the same order in the JPC output as in the PAM input.

A JPC image has a "precision," which is the number of bits used for each code (in Netpbm lingo, "sample"). Actually, it has a separate precision for each component. **pamtojpeg2k** uses for the precision of every component the least number of bits that can represent the maxval of the input image. A JPC image does not have an independent concept of maxval; the maxval of a JPC sample is the maximum value that the number of bits specified by the precision can represent in pure binary code. E.g. if the precision is 4, the maxval is 15. **pamtojpeg2k** does of course scale the sample values from the input maxval to the output maxval. Example: The input maxval is 99. This means JPC precision is 7 bits and the JPC maxval is 127. A sample value of 33 in the input becomes a sample value of 43 in the output.

pamtojpeg2k generates the JPC output with the Jasper JPEG-2000 library . See documentation of the library for details on what **pamtojpeg2k** produces. Note that the Jasper library contains facilities for reading PNM images, but **pamtojpeg2k** does not use those. It uses the Netpbm library instead. Note that the makers of the Jasper library write it "JasPer," but Netpbm documentation follows standard American English typography rules, which don't allow that kind of capitalization.

Use **jpeg2ktopam** to convert in the other direction.

The program **jasper**, which is packaged with the Jasper JPEG-2000 library, also converts between JPEG-2000 and PNM formats. Because it's packaged with the library, it may exploit it better, especially recently added features. However, since it does not use the Netpbm library to read and write the Netpbm formats, it doesn't do as good a job on that side.

OPTIONS

Most of the options are identical in name and function to options that the Jasper library JPC encoder subroutine takes. See Jasper documentation for details. Here, we document only options that are not

direct analogs of Jasper options.

-compression=*ratio*

ratio is a floating point number that specifies the compression ratio. **pamtojpeg2k** will adjust quality as necessary to ensure that you get this compression ratio. E.g. 4 means the output will be about one fourth the size in bytes of the input file.

The compression ratio must be at least 1. The default is 1, which means the output has all the quality of the input -- the conversion is lossless.

Note that though Jasper library takes a compression factor, this option specifies a compression ratio. The compression factor is the multiplicative inverse of (1 divided by) the compression ratio.

-verbose

This option causes **pamtojpeg2k** to issue informational messages about the conversion process.

-debuglevel=*number*

This option controls debug messages from the Jasper library. **pamtojpeg2k** passes *number* as the debug level to the Jasper JPC encoder.

EXAMPLES

This example compresses losslessly.

```
pamtojpeg2k myimg.ppm >myimg.jpc
```

jpeg2ktopam will recreate myimg.ppm exactly.

This example compresses the file to one tenth its original size, throwing away information as necessary.

```
pamtojpeg2k -compression=10 myimg.pgm >myimg.jpc
```

ABOUT JPEG-2000

JPEG-2000 is a format that compresses a visual image (or a similar set of data) into a minimal number of bytes for storage or transmission. In that, its goal is similar to JPEG. It has two main differences from JPEG.

One difference is that it does a much better job on most images of throwing out information in order to achieve a smaller output. That means when you reconstruct the image from the resulting compressed file, it looks a lot closer to the image you started with with JPEG-2000 than with JPEG, for the same compressed file size. Or, looked at another way, with JPEG-2000 you get a much smaller file than with JPEG for the same image quality.

The second difference is that with JPEG-2000, you decide how much compression you want and the compressor adjusts the quality to meet your requirement, whereas with JPEG, you decide how much quality you want and the compressor adjusts the size of the output to meet your requirement. I.e. with JPEG-2000, the quality of the result depends on the compressibility of the input, but with JPEG, the *size* of the result depends on the compressibility of the input.

With JPEG-2000, you can specify lossless compression, thus making it compete with GIF and PNG. With standard JPEG, you always lose something. (There are rumored to be variations of JPEG around that are lossless, though).

JPEG is much older than JPEG-2000 and far more popular. JPEG is one of the half dozen most

popular graphics formats and virtually all graphics facilities understand it. JPEG-2000 is virtually unknown.

There is no compatibility between JPEG and JPEG-2000. Programs that read JPEG do not automatically read JPEG-2000 and vice versa.

SEE ALSO

jpeg2ktopam(1), **pnmtopeg**(1), **ppm**(1), **pgm**(1), **pbm**(1), **pam**(1),

History

pamtojpeg2k was added to Netpbm in Release 10.12 (November 2002).

Table Of Contents

NAME

pamtopfm - Convert Netpbm image to PFM (Portable Float Map)

SYNOPSIS

pamtopfm [**-endian=**{**big**|**little**}] [**-scale=***float*] [*imagefile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtopfm reads a Netpbm image (PNM or PAM) and converts it to a PFM (Portable Float Map) image.

The PFM (Portable Float Map) image format is a lot like PPM, but uses floating point numbers with no maxval to achieve a High Dynamic Range (HDR) format. That means it doesn't have a concept of absolute color and it can represent generic light intensity information rather than just visual information like PPM does. For example, two pixels that are so close in intensity that the human eye cannot tell them apart are not visually distinct, so a visual image format such as PPM would have no reason to use different sample values for them. But an HDR format would.

There are details of the PFM format in the **PFM** Format Description (1).

USC's HDRShop program and a program called Lefty use it.

pamtopfm creates a color PFM image if its input is RGB (PPM) and a non-color PFM otherwise.

Use **pfmtopam**(1)**to**convert**a**PFM image to Netpbm format.

OPTIONS

-scale=*float*

This specifies the scale factor of the PFM image.
Scale factor is a component of the PFM format.
Default is 1.0.

-endian={**big**|**little**}

This specifies the endianness of the PFM image. The samples in the raster of a PFM image are 4 byte IEEE floating point numbers. A parameter of the IEEE format, and therefore the PFM format, is endianness, i.e. whether the specified bytes are ordered from low addresses to high addresses or vice versa.

big means big endian -- the natural ordering;
little means little-endian, the Intel-friendly ordering.

Default is whichever endianness the machine on which **pamtopfm** runs uses internally, which results in the faster execution.

SEE ALSO

Netpbm(1), **pfmtopam**(1), **pam**(1)

HISTORY

pamtopfm was added to Netpbm in Release 10.22 (April 2004).

Table Of Contents

NAME

pamtopnm - convert PAM image to PBM, PGM, or PPM

SYNOPSIS

pamtopnm

[**-assume**]

[*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtopnm reads a PAM image as input and produces an equivalent PBM, PGM, or PPM (i.e. PNM) image, whichever is most appropriate, as output.

pamtopnm assumes the PAM image represents the information required for a PBM, PGM, or PPM image if its tuple type is 'BLACKANDWHITE', 'GRAYSCALE', or 'RGB' and its depth and maxval are appropriate. If this is not the case, **pamtopnm** fails.

However, you can override the tuple type requirement with the **-assume** option.

As with any Netpbm program that reads PAM images, **pamtopnm** also reads PNM images as if they were PAM. In that case, **pamtopnm**'s functions reduces to simply copying the input to the output. But this can be useful in a program that doesn't know whether its input is PAM or PNM but needs to feed it to a program that only recognizes PNM.

OPTIONS

-assume

When you specify **-assume**, you tell **pamtopnm** that you personally vouch for the fact that the tuples contain the same data as belongs in the channels of a PBM, PGM, or PPM file. The depth must still conform, though, so to truly force a conversion, you may have to run the input through **pamchannel** first. But be careful with **-assume**. When you -assume, you make an -ass of u and me.

SEE ALSO

pbmtopgm(1), **pamditherbw**(1), **pgmtoppm**(1), **ppmtopgm**(1), **pam**(1), **pnm**(1), **pbm**(1), **pgm**(1), **ppm**(1)

Table Of Contents

NAME

pamtotga - convert a Netpbm image to a TrueVision Targa file

SYNOPSIS

pamtotga [**-mono**|**-cmap**|**-rgb**] [**-norle**] [*pamfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtotga reads a PBM, PGM, PPM, or PAM image as input and produces a TrueVision Targa file as output. The PAM image may be either a BLACKANDWHITE, GRAYSCALE, RGB, or RGB_ALPHA image.

To create a TGA image with transparency (i.e. with an alpha mask), use RGB_ALPHA PAM input. Some Netpbm programs that generate images with alpha masks generate them in that format. For another way to create the proper input stream, see **pamstack**(1).

It is unclear that anything except **pamtotga** knows about TGAs with transparency. The history behind this feature of **pamtotga** is not clear. The format **pamtotga** produces is simply the same as an ordinary RGB TGA image except with a 4th plane added for transparency. The PixelSize field of the TGA header specifies 32 bits instead of 24 and the raster has an extra byte added to each pixel, at the tail end. The value of that byte has the same meaning as in a PAM image with maxval 255.

OPTIONS

-cmap Make output Targa file of type 24 bit colormapped. Input must contain no more than 256 distinct colors.

-mono Make output Targa file of type 8 bit monochrome. Input must be PBM or PGM or a PAM with BLACKANDWHITE or GRAYSCALE tuple type. See **-cmap**.

You may specify at most one of **-mono**, **-cmap**, and **-rgb**. If you specify neither, the default image type is the most highly constrained compatible type is used, where monochrome is more constrained than colormapped which is in turn more constrained than unmapped.

-rgb Make output Targa file of type 24 bit unmapped color. See **-cmap**.

-norle Do not use run-length encoding in the output Targa file. Run-length encoded files are smaller, but Some Targa readers can't read run-length encoded files.

SEE ALSO

tgatoppm(1), **pnmquant**(1), **pamstack**(1), **pam**(1) **pnm**(1)

HISTORY

This program was called **ppmtotga** until Netpbm 10.6 (July 2002). That was always a misnomer, though, because a PPM class program would not be able to tell the difference between PGM and PPM input (it would all look like PPM), and thus could not choose the output Targa image type based on the type of the input. Netpbm 10.6 also added the ability to handle an alpha channel, so it became a PAM class program.

In Netpbm 10.15 (April 2003), the program became the first in the Netpbm package to recognize an

alpha channel in a PAM. It recognized tuple type 'RGBA'. But when this kind of PAM image was later added to the PAM specification, it was specified with tuple type 'RGB_ALPHA'. So in Netpbm 10-26 (January 2005), **pamtotga** changed to recognize 'RGB_ALPHA' instead of 'RGBA'.

AUTHOR

Table Of Contents

NAME

pamtouil - convert a PNM or PNM/alpha image into a Motif UIL icon file

SYNOPSIS

pamtouil [**-name=***uilname*] [*pamfile*]

You may abbreviate any option to its shortest unique prefix. You may use two hyphens instead of one to delimit an option. You may separate an option from its value with whitespace instead of =.

DESCRIPTION

This program is part of **Netpbm**(1).

pamtouil reads a PNM or PAM image as input and produces a Motif UIL icon file as output. If the input is PAM, it may be either a regular grayscale or color image or grayscale+alpha or color+alpha. Where the alpha channel is present, **pamtouil** renders pixels that are more than half transparent as transparent in the output.

In the UIL's colormap, **pamtouil** uses the color names from the RGB database -- the same one **ppmmake**(1) uses. Where a color in the input does not exactly match one of the colors named in the RGB database, **pamtouil** uses the closest color named in the RGB database.

OPTIONS

-name Allows you to specify the prefix string which is printed in the resulting UIL output. If not specified, will default to the filename (without extension) of the ppmfile argument. If **-name** is not specified and no ppmfile is specified (i.e. piped input), the prefix string will default to the string 'noname'.

SEE ALSO

pam(1) **pam**(1) **ppm**(1)

HISTORY

Before May 2002, **pamtouil** was called **ppmtouil** and had no way to specify transparency.

AUTHOR

Converted by Bryan Henderson from ppmtouil.c, which was converted by Jef Poskanzer from ppmtoxpm.c, which is Copyright (C) 1990 by Mark W. Snitily

Table Of Contents

NAME

pbmclean - flip isolated pixels in portable bitmap

SYNOPSIS

pbmclean [**-minneighbors=***N*] [**-black**|**-white**] [*pbmfile*]

OPTION USAGE

You can use the minimum unique abbreviation of the options. You can use two hyphens instead of one. You can separate an option name from its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pbmclean cleans up a PBM image of random specs. It reads a PBM image as input and outputs a PBM that is the same as the input except with isolated pixels inverted. An isolated pixel is one that has very few neighboring pixels of the same color. The **-minneighbors** option gives the number of same-color neighbors are required.

The default is 1 pixel -- only completely isolated pixels are flipped.

(A **-minneighbors** value greater than 8 generates a completely inverted image (but use **pnminvert** to do that) -- or a completely white or completely black image with the **-black** or **-white** option).

pbmclean considers the area beyond the edges of the image to be white. (This matters when you consider pixels right on the edge of the image).

You can use **pbmclean** to clean up 'snow' on bitmap images.

OPTIONS

-black

-white Flip pixels of the specified color. By default, if you specify neither **-black** nor **-white**, **pbmclean** flips both black and white pixels which do not have sufficient identical neighbors. If you specify **-black**, **pbmclean** leaves the white pixels alone and just erases isolated black pixels. Vice versa for **-white**. You may specify both **-black** and **-white** to get the same as the default behavior.

-minneighbors=*N*

This determines how many pixels must be in a cluster in order for **pbmclean** to consider them legitimate and not clean them out of the image. See Description .

Before December 2001, **pbmclean** accepted **-***N* instead of **-minneighbors**. Before Netpbm 10.27 (March 2005), **-minneighbors** was **-minneighbor**.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmlife - apply Conway’s rules of Life to a portable bitmap

SYNOPSIS

pbmlife [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmlife reads a portable bitmap as input, applies the rules of Life to it for one generation, and produces a PBM image as output.

A white pixel in the image is interpreted as a live beastie, and a black pixel as an empty space.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmmake - create a blank bitmap of a specified size

SYNOPSIS

pbmmake [**-white**|**-black**|**-gray**] *width height*

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

pbmmake produces a PBM image of the specified width and height, either all black, all white, or a dithered gray. The default is white.

OPTIONS

In addition to the usual **-white** and **-black**, this program implements **-gray**. This gives a simple 50% gray pattern with 1's and 0's alternating.

SEE ALSO

pbm(1), **ppmmake**(1)

AUTHOR

Table Of Contents

NAME

pbmmask - create a mask bitmap from a regular bitmap

SYNOPSIS

pbmmask [**-expand**] [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmmask reads a PBM image as input and Generates a corresponding mask of the foreground areas as another PBM image.

The color to be interpreted as 'background' is determined automatically. Regardless of which color is background, the mask will be white where the background is and black where the figure is.

This lets you do a masked paste like this, for objects with a black background:

```
pbmmask obj > objmask
pnmpaste < dest -and objmask <x> <y> | pnmpaste -or obj <x> <y>
```

For objects with a white background, you can either invert them or add a step:

```
pbmmask obj > objmask
pnminvert objmask | pnmpaste -and obj 0 0 > blackback
pnmpaste < dest -and objmask <x> <y> | pnmpaste -or blackback <x> <y>
```

Note that this three-step version works for objects with black backgrounds too, if you don't care about the wasted time.

You can also use masks with graymaps and pixmaps, using the *pnmarith* tool. For instance:

```
ppmtopgm obj.ppm | pamditherbw -threshold | pbmmask > objmask.pbm
pnmarith -multiply dest.ppm objmask.pbm > t1.ppm
pnminvert objmask.pbm | pnmarith -multiply obj.ppm - > t2.ppm
pnmarith -add t1.ppm t2.ppm
```

An interesting variation on this is to pipe the mask through *pnmsmooth* before using it. This makes the boundary between the two images less sharp.

OPTIONS

-expand

Expands the mask by one pixel out from the image. This is useful if you want a little white border around your image. (A better solution might be to turn the **pbmlife** program into a general cellular automaton tool...)

SEE ALSO

ppmcolormask(1), **pnmpaste**(1), **pnminvert**(1), **pnmarith**(1), **pnmsmooth**(1) **pbm**(1),

AUTHOR

Table Of Contents

NAME

pbmpage - create a one page test pattern for printing

SYNOPSIS

pbmpage [**-a4**] *test_pattern*

DESCRIPTION

This program is part of **Netpbm**(1).

pbmpage generates a one page test pattern to print on a sheet of paper, for use in calibrating a printer. The test pattern in is PBM format.

pbmpage produces an image intended for 600 dots per inch printer resolution.

If you are printing on an HP PPA printer, you can convert the output of this program to a stream that you can feed to the printer with **pbmtoppa**.

Bear in mind that when you print the test pattern, you are testing not only the printer, but any converter or driver software along the printing path. Any one of these components may adjust margins, crop the image, erase edges, and such.

If, due to addition of margins, the printer refuses to print the image because it is too big, use **pamcut** to cut the right and bottom edges off the test pattern until it is small enough to print.

test_pattern is the number of the test pattern to generate, as follows. The default is **1**.

1 A black on white grid ruled in numbers of pixels. A black one pixel box is at the very edges of the paper.

Before Netpbm 10.18 (August 2003), the perimeter box was not there.

2 A vertical line segment, one pixel wide, extending 1/2’ up from the exact center of the page.

3 Two diagonal line segments, one starting at the upper left corner of the page, the other starting from the lower left corner of the page. Both extend 1/2’ toward the center of the page at 45 degrees.

OPTIONS

-a4 Generate an image for A4 (European) paper. Without this option, **pbmpage** generates an image for US standard paper (8 1/2’ wide x 11’ high).

SEE ALSO

pbmtoppa(1), **pamcut**(1), **pbm**(1)

AUTHOR

Manual page by Bryan Henderson, May 2000. Contributed to the public domain by its author.

Table Of Contents

NAME

pbmpscale - enlarge a PBM image with edge smoothing

SYNOPSIS

pbmpscale *N* [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmpscale reads a PBM image as input, and outputs a PBM image enlarged N times. **pbmpscale** does this enlargement by pixel replication, with some additional smoothing of corners and edges.

SEE ALSO

pnmenlarge(1), **pamscale**(1), **pbm**(1)

AUTHOR

NOTES

pbmpscale works best for enlargements of 2. Enlargements greater than 2 should be done by as many enlargements of 2 as possible, followed by an enlargement by the remaining factor.

Table Of Contents

NAME

pbmreduce - read a PBM image and reduce it N times

SYNOPSIS

pbmreduce [**-floyd**|**-fs**|**-threshold**] [**-value** *val*] *N* [*pbmfile*]

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

pbmreduce reads a PBM image as input and reduces it by a factor of *N*, producing a PBM image as output.

pbmreduce duplicates a lot of the functionality of **pamditherbw**; you could do something like `pamscale | pamditherbw`, but **pbmreduce** is a lot faster.

You can use **pbmreduce** to 're-halftone' an image. Let's say you have a scanner that only produces black&white, not grayscale, and it does a terrible job of halftoning (most b&w scanners fit this description). One way to fix the halftoning is to scan at the highest possible resolution, say 300 dpi, and then reduce by a factor of three or so using **pbmreduce**. You can even correct the brightness of an image, by using the **-value** option.

OPTIONS

By default, **pbmreduce** does the halftoning after the reduction via boustrophedonic Floyd-Steinberg error diffusion; however, you can use the **-threshold** option to specify simple thresholding. This gives better results when reducing line drawings.

The **-value** option alters the thresholding value for all quantizations. It should be a real number between 0 and 1. Above 0.5 means darker images; below 0.5 means lighter.

SEE ALSO

pnmenlarge(1), **pamscale**(1), **pamditherbw**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtext - render text into a PBM image

SYNOPSIS

pbmtext [**-font** *fontfile*] [**-builtin** *fontname*] [**-space** *pixels*] [**-lspace** *pixels*] [**-nomargins**] [**-width** *pixels*] [*text*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtext takes the specified text, either a single line from the command line or multiple lines from standard input, and renders it into a PBM graphical image.

In the image, each line of input is a line of output. Formatting characters such as newline have no effect on the formatting; like any unprintable character, they turn into spaces.

The image is just wide enough for the longest line of text, plus margins, and just high enough to contain the lines of text, plus margins.

The left and right margins are twice the width of the widest character in the font; the top and bottom margins are the height of the tallest character in the font. But if the text is only one line, all the margins are half of this. You can use the **-nomargins** option to eliminate the margins.

pbmtextps does the same thing as **pbmtext**, but uses Ghostscript to generate the characters, which means it's a lot more sophisticated and you can use Postscript fonts. But it also means you have to have Ghostscript installed and it isn't as fast.

OPTIONS

-font

-builtin

-builtin selects a font among those built into Netpbm.

-font selects a font that you supply yourself either as an X Window System BDF file or as a PBM file in a special form.

The default is the built in font 'bdf.'

'bdf' is Times-Roman 15 pixels high. (That's about 14 point type printed at 75 dpi).

'fixed' is a built in fixed with font.

To create a font as a PBM file (to use with the **-font** option), do this: In your window system of choice, display the following text in the desired (fixed-width) font:

```
M ’,/^_[‘jpqy| M

/  !’#$%&’()*+ /
< ,-./01234567 <
> 89:;<=>?@ABC >
@ DEFGHIJKLMNO @
_ PQRSTUVWXYZ[ _
```

```
{ ]^_‘abcdefg {
} hijklmnopqrs }
~ tuvwxyz{|}~ ~

M ’,/^_[‘jpqy| M
```

Do a screen grab or window dump of that text, using for instance **xwd**, **xgrabsc**, or **screendump**. Convert the result into a pbm file. If necessary, use **pamcut** to remove everything except the text. Finally, run it through **pnmcrop**. to make sure the edges are right up against the text. **pbmtext** can figure out the sizes and spacings from that.

-space *pixels*

Add *pixels* pixels of space between characters. This is in addition to whatever space surrounding characters is built into the font, which is usually enough to produce a reasonable string of text.

pixels may be fractional, in which case the number of pixels added varies so as to achieve the specified average. For example **-space=1.5** causes half the spaces to be 1 pixel and half to be 2 pixels.

pixels may be negative to crowd text together, but the author has not put much thought or testing into how this works in every possible case, so it might cause disastrous results.

-lspace *pixels*

Add *pixels* pixels of space between lines. This is in addition to whatever space above and below characters is built into the font, which is usually enough to produce a reasonable line spacing.

pixels must be a whole number.

pixels may be negative to crowd lines together, but the author has not put much thought or testing into how this works in every possible case, so it might cause disastrous results.

-nomargins

By default, **pbmtext** adds margins all around the image as described above. This option causes **pbmtext** not to add any margins.

Note that there may still be space beyond the edges of the type because a character itself may include space at its edges. To eliminate all surrounding background, so the type touches all four edges of the image, use **pnmcrop**.

-width *pixels*

This specifies how much horizontal space the text is supposed to fit into.

If the input is one line, **pbmtext** breaks it into multiple lines as needed to fit the specified width. It breaks it between characters, but does not pay attention to white space; it may break in the middle of a word and a line may begin or end with white space.

If the input is multiple lines, **pbmtext** assumes you already have line breaks where they make sense, and **pbmtext** simply truncates each line as needed to fit the specified width.

USAGE

Often, you want to place text over another image. One way to do this is with **ppmlabel**. **ppmlabel** does not give you the font options that **pbmtext** does, though.

Another way is to use **pbmtext** to create an image containing the text, then use **pamcomp** to overlay the text image onto your base image. To make only the text (and not the entire rectangle containing it) cover the base image, you will need to give **pamcomp** a mask, via its **-alpha** option. You can just use the text image itself as the mask, as long as you also specify the **-invert** option to **pamcomp**.

If you want to overlay colored text instead of black, just use **ppmchange** to change all black pixels to the color of your choice before overlaying the text image. But still use the original black and white image for the alpha mask.

If you want the text at an angle, use **pnmrotate** on the text image (and alpha mask) before overlaying.

SEE ALSO

pbmtextps(1), **pamcut**(1), **pnmcrop**(1), **pamcomp**(1), **ppmchange**(1), **pnmrotate**(1), **ppmlabel**(1), **pstopnm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtextps - render text into a PBM image using a postscript interpreter

SYNOPSIS

pbmtextps [**-font** *fontname*] [**-fontsize** *fontsize*] [**-resolution** *resolution*] [**-stroke** *strokesize*] *text*

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtextps takes a single line of text from the command line and renders it into a PBM image.

The image is cropped at the top and the right. It is not cropped at the left or bottom so that the text begins at the same position relative to the origin. You can use **pnmcrop** to crop it all the way.

OPTIONS

-font By default, **pbmtextps** uses TimesRoman.

You can specify the font to use with the **-font** option. This is the name of any valid postscript font which is installed on your system.

-fontsize

Size of font in points. See the **-resolution** option for information on how to interpret this size.

Default is 24 points.

-resolution

Resolution in dots per inch of distance measurements pertaining to generation of the image. PBM images don't have any inherent resolution, so a distance such as "1 inch" doesn't mean anything unless you separately specify what resolution you're talking about. That's what this option does.

In particular, the meaning of the font size is determined by this resolution. If the font size is 24 points and the resolution is 150 dpi, then the font size is 50 pixels.

Default is 150 dpi.

-stroke Width of line to use for stroke font. There is no default stroke width because the letters are solid by default.

USAGE

You can generate antialiased text by using a larger resolution than the default and scaling the image down using **pamscale**.

See the manual for the similar **pbmtext** for more advice on usage.

HISTORY

pbmtextps was added to Netpbm in Release 10.0 (June 2002).

SEE ALSO

pbmtext(1), **pamcut**(1), **pnmcrop**(1), **pamcomp**(1), **ppmchange**(1), **pnmrotate**(1), **pamscale**(1), **ppmlabel**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmto10x - convert a PBM image into Gemini 10X printer graphics

SYNOPSIS

pbmto10x [**-h**] [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmto10x reads a PBM image as input and produces a file of Gemini 10X printer graphics as output. The 10x's printer codes are alleged to be similar to the Epson codes.

Note that there is no 10xtopbm tool - this transformation is one way.

OPTIONS

The resolution is normally 60H by 72V. If you specify the **-h** option, resolution is 120H by 144V. You may find it useful to rotate landscape images before printing.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmto4425 - Display PBM images on an AT&T 4425 terminal

SYNOPSIS

pbmto4425 [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmto4425 displays PBM format images on an AT&T 4425 ASCII terminal using that terminal's mosaic graphics character set. The program should also work with other VT100-like terminals with mosaic graphics character sets such as the C. Itoh CIT-101, but it has not yet been tested on terminals other than the 4425.

Pbmto4425 puts the terminal into 132 column mode to achieve the maximum resolution of the terminal. In this mode the terminal has a resolution of 264 columns by 69 rows. The pixels have an aspect ratio of 1:2.6, therefore an image should be processed before being displayed in a manner such as this:

% pamscale -xscale 2.6 *pamfile* **| pamscale -xysize 264 69 | ppmtopgm | pamditherbw | pbmto4425**

SEE ALSO

pbmtoascii(1), **ppmtoterm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtoascii - convert a PBM image to ASCII graphics

SYNOPSIS

pbmtoascii

[**-1x2**|**-2x4**]

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoascii reads a PBM image as input and produces a somewhat crude ASCII graphic image as output.

To convert back, use **asciitopgm**(1).

OPTIONS

The **-1x2** and **-2x4** flags give you two alternate ways for the pixels to get mapped to characters. With **1x2**, the default, each character represents a group of 1 pixel across by 2 pixels down. With **-2x4**, each character represents 2 pixels across by 4 pixels down. With the 1x2 mode you can see the individual pixels, so it's useful for previewing small images on a non-graphics terminal. The 2x4 mode lets you display larger images on a standard 80-column display, but it obscures pixel-level details. 2x4 mode is also good for displaying PGM images:

pamscale -width 158 | pnmnorm | pamditherbw -threshold

should give good results.

SEE ALSO

asciitopgm(1) **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtoatk - convert a PBM image to a Andrew Toolkit raster object

SYNOPSIS

pbmtoatk [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoatk reads a PBM image as input and produces a Andrew Toolkit raster object as output.

SEE ALSO

atktopbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtobbnbg - convert a PBM image into BitGraph graphics

SYNOPSIS

pbmtobbng [*rasterop*] [*x y*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtobbnbg reads a portable bitmap as input and produces BBN BitGraph terminal Display Pixel Data (DPD) sequence as output.

The rasterop can be specified on the command line. If this is omitted, 3 (replace) will be used. A position in (x,y) coordinates can also be specified. If both are given, the rasterop comes first. The portable bitmap is always taken from the standard input.

Note that there is no bgtopbm tool.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmtocmuwm - convert a PBM image into a CMU window manager bitmap

SYNOPSIS

pbmtocmuwm [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtocmuwm reads a portable bitmap as input and produces a CMU window manager bitmap as output.

SEE ALSO

cmuwmtopbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtodjvurle - convert a PBM image to DjVu Bitonal RLE format

SYNOPSIS

pbmtodjvurle

[*pbmfile* [*rlefile*]]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtodjvurle reads a PBM image as input and produces DjVu Bitonal RLE format as output.

SEE ALSO

pamtodjvurle(1) **pbm**(1)

HISTORY

pbmtodjvurle was new in Netpbm 10.22 (April 2004).

AUTHOR

Table Of Contents

NAME

pbmtoepsi - convert a PBM image to an encapsulated PostScript style preview bitmap

SYNOPSIS

pbmtoepsi [**-dpi**=*N*[**x***N*]] [**-bbonly**] [*pbmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

Reads a PBM image as input. Produces an encapsulated Postscript style bitmap as output. The output is not a stand alone postscript file, it is only a preview bitmap, which can be included in an encapsulated PostScript file.

pbmtoepsi assumes the PBM input describes a whole output page, with one pixel on the page corresponding to one PBM pixel. It detects white borders in the image and generates Postscript output that contains a Bounding Box statement to describe the location of the principal image (the image excluding the white borders) on the page and thus does not include the borders in the raster part of the Postscript output.

There is no **epsitopbm** tool - this transformation is one way.

OPTIONS

-dpi=*N*[**x***N*]

This option specifies the resolution in dots per inch of the ultimate output device. You must specify this because the Bounding Box statement defines the bounding box in absolute distances, not in pixels. **pbmtoepsi** assumes in calculating the bounding box that each PBM pixel will become one dot on the output device, and applies your **dpi** specification to calculate the size and location on the page of the bounding box.

If you specify *N***x**N, the first number is the horizontal resolution and the second number is the vertical resolution. If you specify just a single number *N*, that is the resolution in both directions.

The default is 72 dots per inch in both directions.

This option was new In Netpbm 10.3 (June 2002). Before that, **pbmtoepsi** always assumed 72 dots per inch in both directions.

-bbonly

Only create a boundary box, don't fill it with the image.

SEE ALSO

pbm(1), **pnmtops**(1), **pstopnm**(1), **psidtopgm**(1), **pbmtolps**(1),

Postscript language documentation

AUTHOR

Table Of Contents

NAME

pbmtoepson - convert a PBM image into Epson printer graphics

SYNOPSIS

pbmtoepson

[**-dpi=***n*] [**-protocol=**{**escp9**|**escp**}] [**-adjacent**] [**-noadjacent**]

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoepson reads a PBM image as input and produces a stream of Epson printer graphics as output.

The input is from the file identified by the *pbmfile* argument or, if you don't specify *pbmfile*, from Standard Input. Output is to Standard Output.

The output is for traditional (ca 1991) Epson 9-wire dot matrix (sometimes called ESC/P 9-wire) printers or newer ESC/P printers. For a more modern Epson ESC/P2 type printer, try **pbmtoescp2**.

Before Netpbm 10.23 (July 2004), **pbmtoepson** could not produce ESC/P streams -- only ESC/P 9-wire.

The Epson printer protocols are described in
Epson's protocol specification .

Note that there is no epsontopbm tool - this transformation is one way.

OPTIONS

-protocol={**escp9**|**escp**}

This determines which Epson printer protocol the output uses. **escp9** is the older ESC/P 9-pin protocol. **escp** is the newer ESC/P protocol. For the even newer **ESC/P2** protocol, you have to use **pbmtoescp2** instead.

This option was new in Netpbm 10.23 (July 2004).

-dpi=*n* This specifies the horizontal print density in dots per inch. The protocol allows only certain values: 60, 72, 80, 90, 120, 144, and 240. Actually, the ESC/P protocol allows a few others, but **pbmtoepson** doesn't know how to generate the command streams that use them.

If you don't specify this, **pbmtoepson** chooses a horizontal print density for you consistent with your other options.

This option was new in Netpbm 10.23 (July 2004).

-adjacent

-noadjacent

These options determine whether the output select 'adjacent dot printing' or not, whatever that is.

If you don't specify this, **pbmtoepson** selects adjacent dot printing unless that is incompatible with your other options.

This option was new in Netpbm 10.23 (July 2004).

SEE ALSO

pbmtoescp2(1), **pbm**(1),

AUTHOR

Table Of Contents

NAME

pbmtoescp2 - convert a PBM image to a ESC/P2 printer file

SYNOPSIS

pbmtoescp2

[**-compress=***compressionmode*] [**-resolution=***dpi*]

[*pbmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

Input is read from file *pbmfile* if specified, otherwise from stdin. Output is written to stdout.

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoescp2 reads a PBM image as input. It produces an ESC/P2 raster graphic printer control stream as output.

This program creates an output that is printable on Epson printers that understand the ESC/P2 printer control language (e.g. the Stylus models). For older Epson 9-pin dot matrix printers, which use the ESC/P protocol, see **pbmtoepson**.

OPTIONS

-compress=*compressionmode*

This determines the compression mode that **pbmtoescp2** uses in its output. Valid values for *compressionmode* are **0** and **1**. **-compress=0** results in a printer control stream with uncompressed raster graphic data. **-compress=1** results in a printer control stream with RLE compressed raster graphic data (RLE means Run Length Encoding). The default is **-compress=1**.

-resolution=*dpi*

This determines the horizontal and the vertical print resolution set in the printer control stream. Another way of looking at it is a declaration of what the resolution of the input image is (PBM images don't have inherent resolution). Valid values for *dpi* are **180** and **360**. See hints for more information on this.

The default is **-resolution=360**.

HINTS

RLE compresses very well bitmaps of line drawings, preferably horizontal oriented contents like texts, sheets of music, etc. However, bitmaps derived from photographs are not ideal for RLE. In extreme cases, when no byte repetitions occur in the input, the result will be even slightly bigger than the input. To avoid this, use compression mode 0 to switch off RLE.

Each pixel in the input PBM image becomes one dot in the printed output. Therefore, you must make sure the width and height of the input are appropriate for the print resolution you choose and the print area you want. E.g. if you print at 180 dpi and want the image to print as 8 inches by 10, you must supply a PBM that is 1440 pixels wide by 1800 pixels high. You can adjust the size of the input with

pamscale, **pamstretch**, **pbmreduce**, or **pnmenlarge**.

SEE ALSO

escp2topbm(1), **pbmtoepson**(1), **pamscale**(1), **pamstretch**(1), **pbmreduce**(1), **pnmenlarge**(1), **pbm**(1)

AUTHOR

HISTORY

pbmtoescp2 was added to Netpbm in Release 10.18 (August 2003); it was created around the same time.

Table Of Contents

NAME

pbmtog3 - convert a PBM image into a Group 3 fax file

SYNOPSIS

pbmtog3 [**-reversebits**] [**-nofixedwidth**] [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtog3 reads a PBM image as input and produces a Group 3 fax file as output.

OPTIONS

-reversebits

This option causes the output to have the bits in every byte reversed so the least signficant bit becomes the most signficant bit. Apparently, there is some ambiguity in transmission protocols so that the bits get reversed on transmission, and this compensates for that. If you get a whole bunch of "bad code word" messages when you try to read the G3 file (e.g. with **g3topbm**, try using this option. Note that the output is not G3 when you use this option.

-nofixedwidth

Most fax machines expect the image to be 1728 columns wide, so **pbmtog3** cuts the output to this width by default. If you want to keep the width of the original image, use this option.

This option was new in Netpbm 10.6 (July 2002). Before that, **pbmtog3** always kept the width of the original image.

REFERENCES

The standard for Group 3 fax is defined in CCITT Recommendation T.4.

SEE ALSO

g3topbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtogem - convert a PBM image into a GEM .img file

SYNOPSIS

pbmtogem [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtogem reads a PBM image as input and produces a compressed GEM .img file as output.

LIMITATIONS

pbmtogem does not support compression of repeated lines

SEE ALSO

gemtopbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtogo - convert a PBM image into compressed GraphOn graphics

SYNOPSIS

pbmtogo [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtogo reads a PBM image as input and produces 2D compressed GraphOn graphics as output.

Be sure to set up your GraphOn with the following modes: 8 bits / no parity; obeys no XON/XOFF; NULs are accepted. These are all on the Comm menu. Also, remember to turn off tty post processing. Note that there is no gotopbm tool.

SEE ALSO

pbm(1)

AUTHOR

NAME

pbmtoibm23xx − see http://netpbm.sourceforge.net/doc//pbmtoibm23xx.html

DESCRIPTION

pbmtoibm23xx is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//pbmtoibm23xx.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

pbmtoicon - convert a PBM image into a Sun icon

SYNOPSIS

pbmtoicon [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoicon reads a PBM image as input and produces a Sun icon as output.

SEE ALSO

icontopbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtolj - convert a PBM image to HP LaserJet format

SYNOPSIS

pbmtolj [**-resolution** *N*] [**-float**] [**-noreset**] [**-packbits**] [**-delta**] [**-compress**] [*pbmfile*] [**-copies** *N*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtolj reads a PBM image as input and produces HP LaserJet data as output. You can send this data to a LaserJet or DeskJet printer (at least some of them).

Each pixel in the input PBM image becomes one dot in the printed output. Therefore, you must make sure the width and height of the input are appropriate for the print resolution you choose and the print area you want. E.g. if you print at 300 dpi and want the image to print as 8 inches by 10, you must supply a PBM that is 2400 pixels wide by 3000 pixels high. You can adjust the size of the input with **pamscale**, **pamstretch**, **pbmreduce**, or **pnmenlarge**.

The input may be a multi-image PBM stream. Each input image becomes a page of output. But before Netpbm 10.28 (June 2005), images after the first one are ignored.

Note that there is no ljtopbm tool.

OPTIONS

-resolution

Specifies the resolution of the output device, in dpi. Another way to look at this is as a declaration of the resolution of the input image (PBM images don't have inherent resolution). Typical values are 75, 100, 150, 300, and 600. The default is 75.

-float Suppresses positioning commands. By default, **pbmtolj** places the sequence *ESC & l 0 E* in the output, which means to force the top margin to zero. With **-float**, **pbmtolj** omits that command.

-noreset

Prevents pbmtolj from writing the reset sequences to the beginning and end of the output file.

-packbits

Enables use of TIFF packbits compression.

-delta Enables use of delta-between-rows compression.

-compress

Enables use of both TIFF packbits, and delta-between-rows compression.

-copies Specifies the the number of copies. The default is 1. This option controls the 'number of copies' printer control; **pbmtolj** generates only one copy of the image.

You can abbreviate any option to its shortest unique prefix.

SEE ALSO

pnmtopclxl.html(1), **ppmtolj.html**(1), **pjtoppm.html**(1), **ppmtopj**(1), **thinkjettopbm**(1), **pbm**(1)

AUTHOR

 -float and **-noreset** options added by Wim Lewis. **-delta, -packbits,** and **-compress** options added by Dave Platt.

Table Of Contents

NAME

pbmtoln03 - convert PBM image to DEC LN03+ Sixel output

SYNOPSIS

pbmtoln03 [**-rltbf**] *pbmfile*

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoln03 reads a PBM image as input and produces a DEC LN03+ Sixel output file.

OPTIONS

-l nn Use 'nn' as value for left margin (default 0).

-r nn Use 'nn' as value for right margin (default 2400).

-t nn Use 'nn' as value for top margin (default 0).

-b nn Use 'nn' as value for bottom margin (default 3400).

-f nn Use 'nn' as value for form length (default 3400).

SEE ALSO

pbm(1)

AUTHOR

Tim Cook, 26 Feb 1992

Table Of Contents

NAME

pbmtolps - convert PBM image to PostScript

SYNOPSIS

pbmtolps [**-dpi** *n*] [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtolps reads a PBM image as input and outputs PostScript. The output Postscript uses lines instead of the image operator to generate a (device dependent) picture which will be imaged much faster.

The Postscript path length is constrained to be less that 1000 points so that no limits are overrun on the Apple Laserwriter and (presumably) no other printers.

SEE ALSO

pnmtops(1), **pstopnm**(1), **pbmtoepsi**(1), **psidtopgm**(1), **gs**, **pbm**(1),

AUTHOR

George Phillips <*phillips@cs.ubc.ca*>

Table Of Contents

NAME

pbmtomacp - convert a PBM image into a MacPaint file

SYNOPSIS

pbmtomacp [**-l** *left*]

[**-r** *right*]

[**-b** *bottom*]

[**-t** *top*]

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtomacp reads a PBM image as input and produces a MacPaint file as output.

If you do not specify *pbmfile*, **pbmtomacp** uses Standard Input.

The generated file is only the data fork of a picture. You will need a program such as **mcvert** to generate a Macbinary or a BinHex file that contains the necessary information to identify the file as a PNTG file to MacOS.

OPTIONS

-l, **-r**, **-b**, and **-t** let you define a square into the pbm file, that must be converted. Default is the whole file. If the file is too large for a MacPaint-file, the bitmap is cut to fit from (left, top).

SEE ALSO

ppmtopict(1), **macptopbm**(1), **pbm**(1), **mcvert** documentation

AUTHOR

NAME

pbmtomatrixorbital − see http://netpbm.sourceforge.net/doc//pbmtomatrixorbital.html

DESCRIPTION

pbmtomatrixorbital is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//pbmtomatrixorbital.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

pbmtomda - convert a PBM image to a Microdesign .mda

SYNOPSIS

pbmtomda

[**-d**] [**-i**] [**--**]

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtomda reads a PBM image as input and produces a MicroDesign 2 area file (.MDA) as output.

If you do not specify *pbmfile*, **pbmtomda** uses Standard Input.

OPTIONS

-d Halve the height of the output file, to compensate for the aspect ratio used in MicroDesign files.

-i Invert the colors used.

-- End of options (use this if the filename starts with '-')

LIMITATIONS

There's no way to produce files in MicroDesign 3 format. MD3 itself and **mdatopbm**(1)**can**read-**files**in**either**format.

SEE ALSO

mdatopbm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtomgr - convert a PBM image into a MGR bitmap

SYNOPSIS

pbmtomgr

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtomgr reads a PBM image as input and produces a MGR bitmap as output.

SEE ALSO

mgrtopbm(1), **pbm**(1)

AUTHOR

NAME

pbmtomrf - convert a PBM format image to MRF

SYNOPSIS

pbmtomrf

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtomrf converts a PBM image to MRF format.

For more information about mrf, see **the**MRF specification (1).

pbmtomrf takes the PBM image from the file named by the *input.pbm* argument, or Standard Input if you don't specify *input.pbm*. The output goes to Standard Output.

The compression of the edges of pictures with width and/or height not an exact multiple of 64 is not optimal in all cases.

OPTIONS

none.

AUTHOR

Russell Marks.

SEE ALSO

pbmtomrf(1), **pbm**(1), **mrf**(1)

Table Of Contents

NAME

pbmtonokia - convert a PBM image to Nokia Smart Messaging Formats

SYNOPSIS

pbmtonokia [**-fmt** {**HEX_NOL**, **HEX_NGG**, **HEX_NMP**, **NOL**, **NGG**}] [**-net** *networkcode*] [**-txt** *text*] [*options*] [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtonokia reads a PBM image as input and produces a Nokia Smart Messaging (hexcode, .nok, .ngg) file as output.

OPTIONS

-fmt Specifies the output format (default is HEX_NOL).

HEX_NOL

Nokia Operator Logo as (uploadable) hexcode. Use option -net to specify network code.

HEX_NGG

Nokia Group Graphic as (uploadable) hexcode.

HEX_NMP

Nokia Picture Message as (uploadable) hexcode. Use option -txt to specify optional text message.

NOL Nokia Operator Logo as .nol format. This is editable by the Group-Graphic Editor from Kessler Wireless Design (www.kessler-design.com)

NGG Nokia Group Graphic as .ngg format. This is editable by the Group-Graphic Editor from Kessler Wireless Design (www.kessler-design.com)

-net Specifies the 6 hex-digit operator network code for Operator Logos (Default is 62F210 = D1,Germany).

-txt Specifies the text message for Picture Messages. Default is no text message.

LIMITATIONS

Currently limited to rows<=255 and columns<=255. Supports only black and white graphics, not animated.

SEE ALSO

pbm(1),

Nokia Smart Messaging Specification (http://forum.nokia.com)

AUTHOR

Table Of Contents

NAME

pbmtopgm - convert PBM image to PGM by averaging areas

SYNOPSIS

pbmtopgm *width height* [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtopgm reads a PBM image as input. It outputs a PGM image in which each pixel's gray level is the average of the surrounding black and white input pixels. The surrounding area is a rectangle of *width* by *height* pixels.

In other words, this is a convolution. **pbmtopgm** is similar to a special case of **pnmconvol**.

You may need a **pnmsmooth** step after **pbmtopgm**.

pbmtopgm has the effect of anti-aliasing bitmaps which contain distinct line features.

pbmtopgm works best with odd sample width and heights.

You don't need **pbmtopgm** just to use a PGM program on a PBM image. Any PGM program (assuming it uses the Netpbm libraries to read the PGM input) takes PBM input as if it were PGM, with only the mininum and maximum gray levels. So unless your convolution rectangle is bigger than one pixel, you're not gaining anything with a **pbmtopgm** step.

The opposite transformation (which would turn a PGM into a PBM) is dithering. See **pamditherbw**.

SEE ALSO

pamditherbw(1), **pnmconvol**(1), **pbm**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pbmtopi3 - convert a PBM image into an Atari Degas .pi3 file

SYNOPSIS

pbmtopi3 [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtopi3 reads a PBM image as input and produces an Atari Degas .pi3 file as output.

SEE ALSO

pi3topbm(1), **ppmtopi1**(1), **pi1toppm**(1) **pbm**(1),

AUTHOR

Table Of Contents

NAME

pbmtopk - convert a PBM image into a packed (PK) format font

SYNOPSIS

pbmtopk *pkfile*[**.pk**] *tfmfile*[**.tfm**] *resolution* [**-s** *designsize*] [**-p** *num param...*] [**-C** *codingscheme*] [**-F** *family*] [**-f** *optfile*] [**-c** *num*] [**-W** *width*] [**-H** *height*] [**-D** *depth*] [**-I** *ital*] [**-h** *horiz*] [**-v** *vert*] [**-x** *xoff*] [**-y** *yoff*] [*pbmfile ...*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtopk reads PBM images as input and produces a packed (PK) font file and a TFM (TeX font metric) file as output. The resolution parameter indicates the resolution of the font, in dots per inch. If the filename '-' is used for any of the filenames, **pbmtopk** uses Standard Input or Standard Output.

OPTIONS

-s *designsize*

Sets the design size of the font, in TeX's points (72.27pt to the inch). The default design size is 1. The TFM parameters are given as multiples of the design size.

-p *num param...*

Sets the first num font parameters for the font. The first seven parameters are the slant, interword spacing, interword space stretchability, interword space shrinkability, x-height, quad width, and post-sentence extra space of the font. Math and symbol fonts may have more parameters; see The TeXbook for a list of these. Reasonable default values are chosen for parameters which are not specified.

-C *codingscheme*

Sets the coding scheme comment in the TFM file.

-F *family*

Sets the font family comment in the TFM file.

-f *optfile*

Reads the file optfile, which should contain a lines of the form:

filename xoff yoff horiz vert width height depth ital

The PBM files specified by the filename parameters are inserted consecutively in the font with the specified attributes. If any of the attributes are omitted, or replaced with '*', a default value will be calculated from the size of the bitmap. The settings of the -W, -H, -D, -I, -h, -v, -x, and -y options do not affected characters created in this way. The character number can be changed by including a line starting with '=', followed by the new number. Lines beginning with '%' or '#' are ignored.

-c *num* Sets the character number of the next bitmap encountered to num.

-W *width*

Sets the TFM width of the next character to width (in design size multiples).

-H *height*

Sets the TFM height of the next character to height (in design size multiples).

-D *depth*
Sets the TFM depth of the next character to depth (in design size multiples).

-I *ital* Sets the italic correction of the next character to ital (in design size multiples).

-h *horiz*
Sets the horizontal escapement of the next character to horiz (in pixels).

-v *vert* Sets the vertical escapement of the next character to vert (in pixels).

-x *xoff* Sets the horizontal offset of the next character to xoff (in pixels).

-y *yoff* Sets the vertical offset of the next character to yoff (in pixels, from the top row).

SEE ALSO

pktopbm(1), **pbm**(1)

AUTHOR

Adapted from Tom Rokicki's pxtopk by Angus Duggan <*ajcd@dcs.ed.ac.uk*>.

Table Of Contents

NAME

pbmtoplot - convert a PBM image into a Unix 'plot' file

SYNOPSIS

pbmtoplot [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoplot reads a PBM image as input and produces a Unix **plot** file as output.

Note that there is no plottopbm tool - this transformation is one-way.

SEE ALSO

pbm(1), **plot**(1)

AUTHOR

Table Of Contents

NAME

pbmtoppa - convert PBM image to HP Printer Performance Architecture (PPA)

SYNOPSIS

pbmtoppa [*pbm_file* [*ppa_file*]]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoppa converts page images in PBM format to Hewlett Packard's PPA (Printer Performance Architecture) format, which is the data stream format expected by some HP 'Windows-only' printers including the HP Deskjet 820C series, the HP DeskJet 720 series, and the HP DeskJet 1000 series.

pbm_file is the file specification of the input file or **-** for Standard Input. The default is Standard Input.

The input file contains one or more PBM images, with each one being a single page. Each image must have the exact dimensions of a page (at 600 pixels per inch in both directions). Significantly, this is the format that Ghostscript produces.

ppa_file is the file specification of the output file or **-** for Standard Output. The default is Standard Output.

To print Postscript on an HP PPA printer, just use Ghostscript with the **pbmraw** (or **pbm**) device driver.

You can generate a test page for use with this program with **pbmpage**.

You can also set up a printer filter so you can submit PBM input directly to your print queue. See the documentation for your print spooler for information on how to do that, or look in hp820install.doc for an example lpd print filter for Postscript and text files.

Sometimes, **pbmtoppa** generates a file which the printer will not print (because **pbmtoppa**'s input is unprintable). When this happens, all three lights blink to signal the error. This is usually because there is material outside of the printer's printable area. To make the file print, increase the margins via **pbmtoppa** options or a configuration file. See the section on calibration below.

OPTIONS

-v *version*
printer version (720, 820, or 1000)

-x *xoff* horizontal offset adjustment in 1/600 inches.

-y *yoff* vertical offset adjustment in 1/600 inches.

-t *topmarg*
top margin in 1/600 inches (default: 150 = 0.25 inch)

-l *leftmarg*
left margin in 1/600 inches (default: 150 = 0.25 inch)

-r *rightmarg*
right margin in 1/600 inches (default: 150 = 0.25 inch)

-b *botmarg*
bottom margin in 1/600 inches (default: 150 = 0.25 inch)

-s *paper*
paper size: **us** or **a4**. Default is **us**.

-d *dpi* Print resolution in dots per inch.

-f *cfgfile*
Read parameters from the configuration file named *cfgfile*. See CONFIGURATION FILES

The offset adjustments you specify with **-x** and **-y** accumulate. I.e. if you specify them multiple times, the total offset adjustment is the sum of the adjustments you specify. **-x 60 -x 120** is the same as **-x 180**.

The **-v** option undoes any preceding **-x** and **-y** options, leaving the horizontal and vertical adjustments their default values.

CONFIGURATION FILES

You can use a configuration file to specify parameters rather than use invocation options. **pbmtoppa** processes the file **/etc/pbmtoppa.conf**, if it exists, before processing any options. It then processes each configuration file named by a **-f** option in order, applying the parameters from the configuration file as if they were invocation options used in the place of the **-f** option.

Configuration files have the following format:

#Comment
key1 value1
key2 value2
[etc.]

Valid *key*s are **version**, **xoffset**, **yoffset**, **topmargin**, **leftmargin**, **rightmargin**, **bottommargin**, **papersize**, or any non-null prefix of these words. Valid values are the same as with the corresponding invocation parameters.

EXAMPLES

Print a test pattern:
pbmpage | pbmppa >/dev/lp1

Print three pages:
cat page1.pbm page2.pbm page3.pbm | pbmppa >/dev/lp1

Print the Postscript file myfile.ps:
gs -sDEVICE=rawpbm -q -dNOPAUSE -r600 -sOutputFile=- myfile.ps ;| pbmtoppa | lpr

CALIBRATION

To be able to print successfully and properly, you need to tell **pbmtoppa** an X and a Y offset appropriate for your printer to use when generating the page. You can specify these offsets with the **-x** and **-y** invocation options or with the **xoff** and **yoff** parameters in a **pbmtoppa** configuration file.

To determine the correct offsets, use the **pbmpage** program.

If while trying to do this calibration, the printer refuses to print a page, but just blinks all three lights, specify large margins (e.g. 600 pixels -- one inch) via **pbmpage** invocation options while doing the calibration.

For example:
pbmpage | pbmtoppa >/dev/lp1
or
pbmpage | pbmtoppa | lpr -l

(if your printer filter recognizes the '-l' (direct output) parameter).

In the test pattern, the grid is marked off in pixel coordinate numbers. Unfortunately, these coordinates are probably cut off before the edge of the paper. You'll have to use a ruler to estimate the pixel coordinate of the left and top edges of the actual sheet of paper (should be within +/- 300, may be negative; there are 600 pixels per inch).

Add these coordinates to the X and Y offsets by either editing the configuration file or using the **-x** and **-y** command-line parameters.

When **pbmtoppa** is properly calibrated, the center mark should be in the center of the paper. Also, the margins should be able to be as small as 1/4 inch without causing the printer to choke with 'blinking lights syndrome'.

REDHAT LINUX INSTALLATION

RedHat users may find the following tip from Panayotis Vryonis <*vrypan@hol.gr*> helpful. The same should work for the 820 and 1000, but it hasn't been tested. Also, use the pbmraw GSDriver if you have it; it's faster.

Here is a tip to intergrate HP720C support in RedHat's printtool:

Install pbmtoppa. Copy pbmtoppa to /usr/bin.

Edit 'printerdb' (in my system it is found in /usr/lib/rhs/rhs-printfilters) and append the following lines:

----------Cut here----------

```
StartEntry: DeskJet720C
 GSDriver: pbm
 Description: {HP DeskJet 720C}
 About: {       This driver supports the HP DeskJet 720C       inkjet printer.       It does does not support color printing.

 Resolution: {600} {600} {}

EndEntry
```

Now you can add an HP720C printer just like any other, using printtool.

SEE ALSO

pbmpage(1), **pstopnm**(1), **pbm**(1)

pnm2ppa is not part of Netpbm, but does the same things as **pbmtoppa** except it also works with color and has lots more features. See http://sourceforge.net/projects/pnm2ppa .

The file INSTALL-MORE in the pbmtoppa directory of the Netpbm source code contains detailed instructions on setting up a system to use pbmtoppa to allow convenient printing on HP PPA printers. It was written by Michael Buehlmann.

For information about the PPA protocol and the separately distributed pbm2ppa program from which **pbmtoppa** was derived, see http://www.httptech.com/ppa .

AUTHOR

Manual page by Bryan Henderson, May 2000.

Table Of Contents

NAME

pbmtopsg3 - convert PBM images to Postscript with G3 fax compression

SYNOPSIS

pbmtopsg3 [**--title=***title*] [**--dpi=***dpi*] [*filespec*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtopsg3 converts the PBM images in the input PBM file to pages in a Postscript file encoded with G3 fax compression.

If you don't specify *filespec*, the input is from Standard Input.

Remember that you can create a multi-image PBM file simply by concatenating single-image PBM files, so if each page is in a different file, you might do:

```
cat faxpage* | pbmtopsg3 >fax.ps
```

OPTIONS

-title The Postscript title value. Default is no title.

-dpi The resolution of the Postscript output. Default is 72 dpi.

SEE ALSO

pnmtops(1), **pstopnm**(1), **gs**(1), **pstopnm**(1), **pbmtolps**(1), **pbmtoepsi**(1), **pbmtog3**(1), **g3topbm**(1), **pbm**(1)

Table Of Contents

NAME

pbmtoptx - convert a PBM image into Printronix printer graphics

SYNOPSIS

pbmtoptx [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoptx reads a PBM image as input and produces a file of Printronix printer graphics as output.

Note that there is no ptxtopbm tool - this transformation is one way.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmtowbmp - convert a PBM image to a wireless bitmap (wbmp) file

SYNOPSIS

pbmtowbmp [*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtowbmp reads a PBM image as input and produces a wbmp file as output.

LIMITATIONS

pbmtowbmp can generate only WBMP type 0. This is the only type specified in the WAP 1.1 specifications.

SEE ALSO

pbm(1), **wbmptopbm**(1),

Wireless Application Environment Specification.

AUTHOR

Table Of Contents

NAME

pbmtox10bm - convert a PBM image into an X10 bitmap

SYNOPSIS

pbmtox10bm

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtox10bm reads a PBM image as input and produces an X10 bitmap as output. This older format is obsoleted by the X11 bitmap format.

Note that there is no x10bmtopbm tool, because *xbmtopbm* can read both X11 and X10 bitmaps.

SEE ALSO

pbmtoxbm(1), **xbmtopbm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pbmtoxbm - convert a PBM image to an X11 bitmap

SYNOPSIS

pbmtoxbm

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoxbm reads a PBM image as input and produces an X11 bitmap as output.

SEE ALSO

pbmtox10bm(1), **xbmtopbm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pgmtoybm - convert a PBM image into a Bennet Yee ’face’ file

SYNOPSIS

pbmtoybm

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtoybm reads a PBM image as input and produces as output a file acceptable to the **face** and **xbm** programs by Bennet Yee (*bsy+@cs.cmu.edu*).

SEE ALSO

ybmtopbm(1), **pbm**(1),

AUTHOR

Table Of Contents

NAME

pbmtozinc - convert a PBM image into a Zinc bitmap

SYNOPSIS

pbmtozinc

[*pbmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pbmtozinc reads a PBM image as input and produces a bitmap in the format used by the Zinc Interface Library (ZIL) Version 1.0 as output.

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pbmupc - create a Universal Product Code PBM image

SYNOPSIS

pbmupc

[**-s1** | **-s2**]

type manufacturer product

DESCRIPTION

This program is part of **Netpbm**(1).

pbmupc generates an image of a Universal Product Code symbol. The three arguments are: a one digit product type, a five digit manufacturer code, and a five digit product code. For example, '0 72890 00011' is the code for Heineken.

As presently configured, **pbmupc** produces an image 230 bits wide and 175 bits high. The size can be altered by changing the defines at the beginning of the program, or by running the output through **pnmenlarge** or **pamscale**.

OPTIONS

The **-s1** and **-s2** options select the style of UPC to generate. The default, **-s1**, looks more or less like this:

```
 ||||||||||||||
 ||||||||||||||
 ||||||||||||||
 ||||||||||||||
0||12345||67890||5
```

The other style, **-s2**, puts the product type digit higher up, and doesn't display the checksum digit:

```
 ||||||||||||||
 ||||||||||||||
0||||||||||||||
 ||||||||||||||
 ||12345||67890||
```

SEE ALSO

pbm(1)

AUTHOR

Table Of Contents

NAME

pc1toppm - convert an Atari Degas .pc1 into a PPM image

SYNOPSIS

pc1toppm

[*pc1file*]

DESCRIPTION

This program is part of **Netpbm**(1).

pc1toppm reads an Atari Degas .pc1 file as input and produces a PPM image as output.

The .pc1 format is a compressed (run length encoded) variation on .pi1.

HISTORY

pc1toppm was new in Netpbm 10.22 (April 2004)

SEE ALSO

ppm(1), **pi1toppm**(1), **pi3topbm**(1), **pbmtopi3**(1)

Table Of Contents

NAME

pcdovtoppm - create index image for a photo CD

SYNOPSIS

pcdovtoppm [**-m** *width*] [**-s** *size*] [**-a** *across*] [**-c** *colors*] [**-f** *font*] [**-b**|**-w**] [*pcdfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

This program generates an index image in PPM format for a photo CD, based on the photo CD overview file.

You can achieve a similar result with **hpcdtoppm -Overview** followed by **pnmindex -black** on the generated PPM images.

OPTIONS

-w*width*
: Maximum width of the result image (default: 1152).

-s*size* Maximum size of each of the images (default: 192).

-a*across*
: Maximum number of images across (default: 6).

-c*colors*
: Maximum number of colors, or **n**

to mean no quantization

-f*font* Font to be used for annotation (default: internal font).

-b Black background color (default).

-w White background color.

EXAMPLES

pcdovtoppm -m 768 -s 96 -f smallfont.pbm overview.pcd > overview.ppm

pcdovtoppm /cdrom/photo_cd/overview.pcd | ppmtojpeg > overview.jpg

HISTORY

This program was formerly called **pcdindex**, which did not fit Netpbm naming conventions.

SEE ALSO

hpcdtoppm(1), **pnmindex**(1), **ppmtojpeg**(1), **ppm**(1)

Table Of Contents

NAME

pcxtoppm - convert a PCX file into a PPM image

SYNOPSIS

pcxtoppm [**-stdpalette**] [**-verbose**] [*pcxfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pcxtoppm reads a PCX file as input and produces a PPM image as output.

pcxtoppm recognizes the following PCX types:

- Colormapped files with 2-16 colors.

  'Packed pixel' format (1, 2 or 4 bits/pixel, 1 plane) or bitplane format (1 bit/pixel, 1-4 planes). The program uses a predefined standard palette if the image does not provide one. 'Does not provide one' means the palette in the PCX header is completely black.

- Colormapped files with 256 colors.

  8 bits/pixel, 1 plane, colormap at the end of the file.

- 24bit truecolor files.

  24bit RGB: 8 bits/pixel, 3 planes.

- 32bit truecolor files.

  24bit RGB + 8bit intensity: 8 bits/pixel, 4 planes.

OPTIONS

-stdpalette

This option causes **pcxtoppm** to use its predefined standard palette even if the PCX image provides its own. This is meaningful only for an image in the 16 color paletted PCX format.

The image may appear to provide its own palette but in fact be created by a program too primitive to understand palettes that created a random palette by accident.

SEE ALSO

ppmtopcx(1), **ppm**(1)

AUTHORS

Modified 1994 by Ingo Wilken (*Ingo.Wilken@informatik.uni-oldenburg.de*)

Table Of Contents

NAME

pamtopfm - Convert PFM (Portable Float Map) image to Netpbm format

SYNOPSIS

pfmtopam [**-maxval=***n*] [**-verbose**] [*imagefile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pfmtopam reads a PFM (Portable Float Map) image and converts it to PAM.

See **pamtopfm**(1)**for**a**description**of PFM.

If you want one of the older, more portable Netpbm formats, run the output through **pamtopnm**.

pamtopfm creates a PAM with tuple type 'RGB' or 'GRAYSCALE' depending on whether or not the PFM is in the color subformat.

Use **pamtopfm**(1)**to**convert**a**PFM image to Netpbm format.

OPTIONS

-maxval=*n*

This specifies the maxval for the PAM. Default is 255.

-verbose

This causes **pfmtopam** to display messages describing the PFM input file.

SEE ALSO

pamtopfm(1), **pam**(1),

HISTORY

pfmtopam was added to Netpbm in Release 10.22 (April 2004).

Table Of Contents

NAME

pgmabel - create cross section using Abel Integration for Deconvolution

SYNOPSIS

pgmabel [**-help**] [**-axis** *axis*] [**-factor** *factor*] [**-pixsize** *pixsize*] [**-left** | **-right**] [**-verbose**] [*filespec*]

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

pgmabel reads as input a PGM image, which it assumes to be an image of a rotational symmetric transparent object. The image must have a vertical symmetry axis. **pgmabel** produces as output an image of a cross-section of the image.

pgmabel does the calculation by performing the Abel Integration for Deconvolution of an axial-symmetrical image by solving the system of linear equations.

After integration, **pgmabel** weights all gray-values of one side by the surface area of the calculated ring in square pixels divided by 4**factor* multiplied by the size of one pixel (*pixsize*). With the **-verbose** option, **pgmabel** prints the weighting factors.

Where the caculation generates a negative result, the output is black.

The computation is unstable against periodic structures with size 2 in the vertical direction.

OPTIONS

-help Prints a help message.

-axis *axis*

Position of the axis of symmetry in the image in pixels from the left edge of the image. Default is the center of the image.

-factor *factor*

User defined factor for enhancement of the output. Use a *factor* less than 1 for decreasing gary values. Default is 1.0.

-pixsize *pixsize*

The size of a pixel for getting scale invariant. Default is 0.1.

-left Calculate only the left side of the image. You cannot specify both **left** and **right**.

-right Analogous to **-left**.

-verbose

print information about the calculation.

EXAMPLE

Rotate a PGM image to get an image with a vertical axis of symmetry, then calculate the cross section:

```
pnmrotate 90 file.pgm | pgmabel -axis 140 >cross_section.pgm
```

SEE ALSO

pnmrotate(1), **pgm**(1),

HISTORY

This program was added to Netpbm in Release 10.3 (June 2002).

AUTHOR

Volker Schmidt (lefti@voyager.boerde.de)

Table Of Contents

NAME

pgmbentley - Bentleyize a PGM image

SYNOPSIS

pgmbentley [*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmbentleyreads a PTM image as input and performs the Bentley Effect, and writes a PGM image as output.

The Bentley Effect is described in 'Beyond Photography' by Holzmann, chapter 4, photo 4. It's a vertical smearing based on brightness.

SEE ALSO

pgmoil(1), **ppmrelief**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pgmcrater - create cratered terrain by fractal forgery

SYNOPSIS

pgmcrater

[**-number** *n*]

[**-height**|**-ysize** *s*]

[**-width**|**-xsize** *s*]

[**-gamma** *g*]

All options can be abbreviated to their shortest unique prefix.

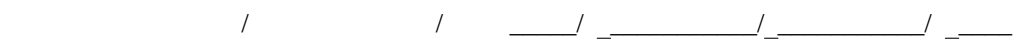
DESCRIPTION

This program is part of **Netpbm**(1).

pgmcrater creates a PGM image which mimics cratered terrain. The PGM image is created by simulating the impact of a given number of craters with random position and size, then rendering the resulting terrain elevations based on a light source shining from one side of the screen. The size distribution of the craters is based on a power law which results in many more small craters than large ones. The number of craters of a given size varies as the reciprocal of the area as described on pages 31 and 32 of Peitgen and Saupe[1]; cratered bodies in the Solar System are observed to obey this relationship. The formula used to obtain crater radii governed by this law from a uniformly distributed pseudorandom sequence was developed by Rudy Rucker.

High resolution images with large numbers of craters often benefit from being piped through **pnmsmooth**. The averaging performed by this process eliminates some of the jagged pixels and lends a mellow ''telescopic image'' feel to the overall picture.

pgmcrater simulates only small craters, which are hemispherical in shape (regardless of the incidence angle of the impacting body, as long as the velocity is sufficiently high). Large craters, such as Copernicus and Tycho on the Moon, have a ''walled plain'' shape with a cross-section more like:



Larger craters should really use this profile, including the central peak, and totally obliterate the pre-existing terrain.

OPTIONS

-number *n*

Causes *n* craters to be generated. If no **-number** specification is given, 50000 craters will be generated. Don't expect to see them all! For every large crater there are many, many more tiny ones which tend simply to erode the landscape. In general, the more craters you specify the more realistic the result; ideally you want the entire terrain to have been extensively turned over again and again by cratering. High resolution images containing five to ten million craters are stunning but take quite a while to create.

-height *height*

Sets the height of the generated image to *height* pixels. The default height is 256 pixels.

-width *width*

Sets the width of the generated image to *width* pixels. The default width is 256 pixels.

-xsize *width*
Sets the width of the generated image to *width* pixels. The default width is 256 pixels.

-ysize *height*
Sets the height of the generated image to *height* pixels. The default height is 256 pixels.

-gamma *factor*
The specified *factor* is used to gamma adjust the image in the same manner as performed by **pnmgamma**. The default value is 1.0, which results in a medium contrast image. Values larger than 1 lighten the image and reduce contrast, while values less than 1 darken the image, increasing contrast.

Note that this is separate from the gamma correction that is part of the definition of the PGM format. The image **pnmgamma** generates is a genuine, gamma-corrected PGM image in any case. This option simply changes the contrast and may compensate for a display device that does not correctly render PGM images.

DESIGN NOTES

The-**gamma** option isn't really necessary since you can achieve the same effect by piping the output from **pgmcrater** through **pnmgamma**. However, **pgmcrater** performs an internal gamma map anyway in the process of rendering the elevation array into the PGM format, so there's no additional overhead in allowing an additional gamma adjustment.

Real craters have two distinct morphologies.

SEE ALSO

pnmgamma(1), **pnmsmooth**(1) **pgm**(1),

[1] Peitgen, H.-O., and Saupe, D. eds., The Science Of Fractal Images, New York: Springer Verlag, 1988.

AUTHOR

John Walker
Autodesk SA
Avenue des Champs-Montants 14b
CH-2074 MARIN
Suisse/Schweiz/Svizzera/Svizra/Switzerland
Usenet:*kelvin@Autodesk.com*
Fax:038/33 88 15
Voice:038/33 76 33

HISTORY

The original 1991 version of this manual contains the following:

PLUGWARE!

If you like this kind of stuff, you may also enjoy 'James Gleick's Chaos--The Software' for MS-DOS, available for $59.95 from your local software store or directly from Autodesk, Inc., Attn: Science Series, 2320 Marinship Way, Sausalito, CA 94965, USA. Telephone: (800) 688-2344 toll-free or,

outside the U.S. (415) 332-2344 Ext 4886. Fax: (415) 289-4718. 'Chaos--The Software' includes a more comprehensive fractal forgery generator which creates three-dimensional landscapes as well as clouds and planets, plus five more modules which explore other aspects of Chaos. The user guide of more than 200 pages includes an introduction by James Gleick and detailed explanations by Rudy Rucker of the mathematics and algorithms used by each program.

NAME

pgmedge - replaced by pamedge

DESCRIPTION

This program is part of **Netpbm**(1).

pgmedge was replaced in Netpbm 10.14 (March 2002) by **pamedge**(1).

pamedge is backward compatible with **pgmedge**, but works on color images too.

Table Of Contents

NAME

pgmenhance - edge-enhance a PGM image

SYNOPSIS

pgmenhance

[-*N*]

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmenhance reads a PGM image as input, enhances the edges, and writes a PGM image as output.

The edge enhancing technique is taken from Philip R. Thompson's 'xim' program, which in turn took it from section 6 of 'Digital Halftones by Dot Diffusion', D. E. Knuth, ACM Transaction on Graphics Vol. 6, No. 4, October 1987, which in turn got it from two 1976 papers by J. F. Jarvis et. al.

OPTIONS

The optional **-***N* option should be a digit from 1 to 9. 1 is the lowest level of enhancement; 9 is the highest. The default is 9.

SEE ALSO

pgmedge(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pgmhist - print a histogram of the values in a PGM image

SYNOPSIS

pgmhist

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmhist reads a PGM image as input and prints a histogram of the gray values.

SEE ALSO

pgmnorm(1), **ppmhist**(1) **pgm**(1),

AUTHOR

Table Contents

NAME

pgmkernel - generate a convolution kernel

SYNOPSIS

pgmkernel

[**-weight** *w*]

width

[*height*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmkernel generates a convolution kernel that you can use with **pnmconvol**. The kernel is one where the weight of each location is inversely proportional to its distance from the center of the kernel.

pgmkernel generates a PGM image of size *width* by *height* (or *width* by *width* if you don't specify *height*).

pgmkernel computes the convolution function K as follows.

K(i,j) = 1 / (1 + w * sqrt(i^2 + j^2))

where *w* is a coefficient specified via the -*weight* flag. *i* and *j* are measured in pixels. K is zero everywhere beyond the specified kernel width and height.

pgmkernel generates the output PGM file in the Plain (text) variation of PGM.

OPTIONS

The optional -*weight* flag should be a real number greater than -1. The default value is 6.0.

LIMITATIONS

The computation time is proportional to *width***height*. This increases rapidly with the increase of the kernel size. A better approach could be using a FFT in these cases.

SEE ALSO

pnmconvol(1), **pnmsmooth**(1) **pamgauss**(1) **pgm**(1)

AUTHOR

Alberto Accomazzi (*alberto@cfa.harvard.edu*).

Table Of Contents

NAME

pgmminkowski - compute Minkowski integral

SYNOPSIS

pgmminkowski *pgmfile*

DESCRIPTION

This program is part of **Netpbm**(1).

pgmminkowski computes the 3 Minkowski integrals of a PGM image.

The Minkowski integrals mathematically characterize the shapes in the image and hence are the basis of "morphological image analysis."

Hadwiger's theorem has it that these integrals are the only motion-invariant, additive and conditionally continuous functions of a two-dimensional image, which means that they are preserved under certain kinds of deformations of the image. On top of that, they are very easy and quickly calculated. This makes them of interest for certain kinds of pattern recognition.

Basically, the Minkowski integrals are the area, total perimeter length, and the Euler characteristic of the image, where these metrics apply to the foreground image, not the rectangular PGM image itself. The foreground image consists of all the pixels in the image that are white. For a grayscale image, there is some threshold of intensity applied to categorize pixels into black and white, and the Minkowski integrals are calculated as a function of this threshold value. The total surface area refers to the number of white pixels in the PGM and the perimeter is the sum of perimeters of each closed white region in the PGM.

For a grayscale image, these numbers are a function of the threshold of what you want to call black or white. **pgmminkowski** reports these numbers as a function of the threshold for all possible threshold values. Since the total surface area can increase only as a function of the threshold, it is a reparameterization of the threshold. It turns out that if you consider the other two functions, the boundary length and the Euler characteristic, as a function of the first one, the surface, you get two functions that are a fingerprint of the picture. This fingerprint is e.g. sufficient to recognize the difference between pictures of different crystal lattices under a scanning tunnelling electron microscope.

For more info, see e.g.
K. Michielsen and H. De Raedt, "Integral-Geometry Morphological Image Analysis", Phys. Rep. 347, 461-538 (2001).

The output is suitable for direct use as a datafile in **gnuplot**.

In addition to the three Minkowski integrals, **pgmminkowski** also lists the horizontal and vertical edge counts.

SEE ALSO

pgmmorph(1) **pgm**(1)

AUTHORS

Luuk van Dijk, 2001.

Table Of Contents

NAME

pgmmorphconv - perform morphological convolutions: dilation, erosion

SYNOPSIS

pgmmorphconv [**-erode** | **-dilate** | **-open** | **-close**] *templatefile* [*pgmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pgmmorphconv performs morphological convolutions on a PGM image: dilation and erosion.

pgmmorphconv performs a "topological" convolution. For each pixel of the input, **pgmmorphconv** generates an output pixel in the same position. To determine the intensity of the output pixel, **pgmmorphconv** lays the template image over the input image such that the middle pixel of the template is over the input pixel in question. **pgmmorphconv** looks at the input pixels underneath each white pixel in the template. For a dilation, the maximum intensity of all those pixels is the intensity of the output pixel. For an erosion, it is the minimum.

Thus, the dilation effect is that bright areas of the input get bigger and dark areas smaller. The erosion effect is the opposite. The simplest template image would be one with a white pixel in the middle and the rest black. This would produce an output image identical to the input. Another simple template image is a fully white square. This causes bright or dark areas to expand in all directions. A template image that is white on the left side and black on the right would smear the image to the right.

The template file named by *templatefile* contains the template image as a PBM image. It must have an odd number of rows and an odd number of columns, so there is a definite middle pixel. It must contain at least one white pixel.

This is similar to the continuous convolution done by **pnmconvol**, except that with **pnmconvol** the output intensity is a weighted average of nearby input pixels instead of a minimum or maximum.

This convolution changes the three Minkowski integrals in a predefined way, an can be used to filter an image to enhance certain features, to ease their automatic recognition.

The options **-erode** and **-dilate** obviously produce an erosion or dilation, respectively.

The **-open** option causes **pgmmorphconv** to perform first an erode and then a dilate operation. The **-close** option causes a dilate first and then an erode. If you specify none of these options, it is the same as **-dilate**.

SEE ALSO

pgmminkowski(1) **pnmconvolz**(1) **pgm**(1)

AUTHORS

Luuk van Dijk, 2001.

Table Of Contents

NAME

pgmnoise - create a graymap made up of white noise

SYNOPSIS

pgmnoise

width height

DESCRIPTION

This program is part of **Netpbm**(1).

pgmnoise creates a portable graymap that is made up of random pixels with gray values in the range of 0 to PGM_MAXMAXVAL (depends on the compilation, either 255 or 65535). The graymap has a size of width * height pixels.

SEE ALSO

pgm(1)

AUTHOR

NAME

pgmnorm - replaced by pnmnorm

DESCRIPTION

This program is part of **Netpbm**(1).

pgmnorm was replaced in Netpbm 9.25 (March 2002) by **pnmnorm**(1).

pnmnorm is backward compatible with **pgmnorm**, but it also handles PPM images.

NAME

pgmoil - replaced by pamoil

DESCRIPTION

This program is part of **Netpbm**(1).

pgmoil was replaced in Netpbm 9.16 (July 2001) by **pamoil**(1).

pamoil is backward compatible with **pgmoil**, but works on color images too.

Table Of Contents

NAME

pgmramp - generate a grayscale ramp

SYNOPSIS

pgmramp -lr|-tb|-rectangle|-ellipse -maxval=*maxval width height*

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

Generates a graymap of the specified size containing a black-to-white ramp. These ramps are useful for multiplying with other images, using the **pnmarith** tool.

The ramp is linear in brightness, not intensity. I.e. the gamma-corrected sample values in the PGM rise linearly with distance from the corner of the image. If you want a ramp that is linear in light intensity, use **pnmgamma** with **pgmramp**.

To generate a simple ramp of all the values from 0 to maxval, and not necessarily a graphic image, use **pamseq**(1).

OPTIONS

You must specify exactly one of the ramp type options.

-lr A left to right ramp.

-tb A top to bottom ramp.

-rectangle
An outside-in rectangular ramp. It is black around the edges and white in the center.

-ellipse An outside-in elliptical ramp. It is black around the edge and white in the center.

-maxval=*maxval*
The maxval for the generated image. Default is 255.

This option did not exist before June 2002. Before, the maxval was always 255.

SEE ALSO

pnmarith(1), **pnmgamma**(1), **pamseq**(1), **ppmrainbow**(1), **pgm**(1)

AUTHOR

NAME

pgmslice - extract one line of pixel values out of a PGM

DESCRIPTION

This program is part of **Netpbm**(1).

Starting with Netpbm 10.3, **pgmslice** is obsolete. Use **pamslice**(1)**instead.**It**is**backward compatible.

Table Of Contents

NAME

pgmtexture - calculate textural features on a PGM image

SYNOPSIS

pgmtexture

[**-d** *d*]

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmtexture reads a PGM image as input and calculates textural features based on spatial dependence matrices at 0, 45, 90, and 135 degrees for a distance *d* (default = 1).

Textural features include:

- Angular Second Moment
- Contrast
- Correlation
- Variance
- Inverse Difference Moment
- Sum Average
- Sum Variance
- Sum Entropy
- Entropy
- Difference Variance
- Difference Entropy
- Information Measures of Correlation
- Maximal Correlation Coefficient

Algorithm taken from: *Haralick, R.M., K. Shanmugam, and I. Dinstein. 1973. Textural features for image classification. IEEE Transactions on Systems, Man, and Cybertinetics, SMC-3(6):610-621.*

LIMITATIONS

The program can run incredibly slow for large images (larger than 64 x 64) and command line options are limited. The method for finding the Maximal Correlation Coefficient, which requires finding the second largest eigenvalue of a matrix Q, does not always converge.

SEE ALSO

pgm(1), **pamsharpness**(1), **pamsharpmap**(1), **pgmedge**(1), **pgmminkowski**(1)

AUTHOR

Table Of Contents

NAME

pgmtofs - convert PGM image to Usenix FaceSaver(tm) format

SYNOPSIS

pgmtofs

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmtofs reads a PGM image as input and produces Usenix FaceSaver(tm) format as output.

FaceSaver is a registered trademark of Metron Computerware Ltd. of Oakland, CA.

SEE ALSO

fstopgm(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pgmtolispm - convert a PGM image to Lisp Machine format

SYNOPSIS

pgmtolispm

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmtolispm reads a PGM image as input and produces a Lisp Machine bitmap as output.

This is the file format read by the tv:read-bit-array-file function on TI Explorer and Symbolics lisp machines.

Given a PGM (instead of a PBM), **pgmtolispm** outputs a multi-plane image. This is probably not useful unless you have a color lisp machine.

Multi-plane bitmaps on lisp machines are color; but the lispm image file format does not include a color map, so we must treat it as a graymap instead. This is unfortunate.

SEE ALSO

lispmtopgm(1), **pgm**(1)

LIMITATIONS

Output width is always rounded up to the nearest multiple of 32; this might not always be what you want, but it probably is (arrays which are not modulo 32 cannot be passed to the Lispm BITBLT function, and thus cannot easily be displayed on the screen).

No color.

AUTHOR

Table Of Contents

NAME

pgmtopbm - convert a PGM image to PBM

SYNOPSIS

pgmtopbm

[**-floyd** | **-fs** | **-threshold** | **-hilbert** | **-dither8** | **-d8** | **-cluster3** | **-c3** | **-cluster4** | **-c4** | **-cluster8** | **-c8**]

[**-value** *val*]

[**-clump** *size*]

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

This program is obsolete since Netpbm 10.23 (July 2004). Use **pamditherbw**(1)**to**do**what**this**program** used to do.

pgmtopbm never was the simple converter it appeared to be. It was a dithering program. Unfortunately, it didn't do the dithering properly because it treated the PGM input samples as if they were directly proportional to light intensity, but they are actually gamma-adjusted.

pamditherbw is backward compatible with **pgmtopbm** except that it does the correct gamma adjustments.

Table Of Contents

NAME

pgmtopgm - copy PGM image

SYNOPSIS

pgmtopgm

DESCRIPTION

This program is part of **Netpbm**(1).

pgmtopgm simply copies a PGM image from Standard Input to Standard Output. This may seem an unnecessary duplication of **cat**, but remember that a PGM program can read a PBM image as if it were PGM. So **pgmtopgm** can read either a PBM or PGM image and produce a PGM image as output.

Even that is of limited usefulness because of the fact that almost any program to which you would feed the resulting PGM image could also just take the original image directly. However, sometimes you really need a true PGM image.

When you know you have a PBM image and want a PGM image, **pbmtopgm** is a more general way to do that conversion.

SEE ALSO

ppmtoppm(1), **pnmtopnm**(1), **pamtopnm**(1), **pbmtopgm**(1), **pgm**(1), **pgm**(1),

HISTORY

This program was added to Netpbm in Release 10.9 (September 2002).

Table Of Contents

NAME

pgmtoppm - colorize a PGM (grayscale) image into a PPM (color) image

SYNOPSIS

pgmtoppm

colorspec

[*pgmfile*] **pgmtoppm**

colorspec1-colorspec2

[*pgmfile*] **pgmtoppm -map**

mapfile

[*pgmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pgmtoppm reads a PGM as input and produces a PPM file as output with a specific color assigned to each gray value in the input.

If you specify one color argument, black in the pgm file stays black and white in the pgm file turns into the specified color in the ppm file. Gray values in between are linearly mapped to differing intensities of the specified color.

If you specify two color arguments (separated by a dash), then black gets mapped to the first color and white gets mapped to the second and gray values in between get mapped linearly (across a three dimensional space) to colors in between.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

Also, you can specify an entire colormap with the **-map** option. The mapfile is just a **ppm** file; it can be any shape, all that matters is the colors in it and their order. In this case, black gets mapped into the first color in the map file, and white gets mapped to the last and gray values in between are mapped linearly onto the sequence of colors in between.

A more direct way to specify a particular color to replace each particular gray level is to use **pamlookup**. You make an index file that explicitly associates a color with each possible gray level.

NOTE - MAXVAL

The 'maxval,' or depth, of the output image is the same as that of the input image. The maxval affects the color resolution, which may cause quantization errors you don't anticipate in your output. For example, you have a simple black and white image as a PGM with maxval 1. Run this image through **pgmtoppm 0f/00/00** to try to make the image black and faint red. Because the output image will also have maxval 1, there is no such thing as faint red. It has to be either full-on red or black. **pgmtoppm** rounds the color 0f/00/00 down to black, and you get an output image that is nothing but black.

The fix is easy: Pass the input through **pnmdepth** on the way into **pgmtoppm** to increase its depth to something that would give you the resolution you need to get your desired color. In this case, **pnmdepth 16** would do it. Or spare yourself the unnecessary thinking and just say **pnmdepth 255 .**

PBM input is a special case. While you might think this would be equivalent to a PGM with maxval 1 since only two gray levels are necessary to represent a PBM image, **pgmtoppm**, like all Netpbm programs, in fact treats it as a maxval of 255.

SEE ALSO

pnmdepth(1), **rgb3toppm**(1), **ppmtopgm**(1), **ppmtorgb3**(1), **ppm**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pi1toppm - convert an Atari Degas .pi1 into a PPM image

SYNOPSIS

pi1toppm

[*pi1file*]

DESCRIPTION

This program is part of **Netpbm**(1).

pi1toppm reads an Atari Degas .pi1 file as input and produces a PPM image as output.

SEE ALSO

ppmtopi1(1), **pc1toppm**(1), **ppm**(1), **pi3topbm**(1), **pbmtopi3**(1)

AUTHOR

Table Of Contents

NAME

pi3topbm - convert an Atari Degas .pi3 file into a PBM image

SYNOPSIS

pi3topbm

[*pi3file*]

DESCRIPTION

This program is part of **Netpbm**(1).

pi3topbm reads an Atari Degas .pi3 file as input and produces a PBM image as output.

SEE ALSO

pbmtopi3(1), **pbm**(1), **pi1toppm**(1), **ppmtopi1**(1)

AUTHOR

Table Of Contents

NAME

picttoppm - convert a Macintosh PICT file into a portable pixmap

SYNOPSIS

picttoppm

[**-verbose**]

[**-fullres**]

[**-noheader**]

[**-quickdraw**]

[**-fontdir**file**]**

[*pictfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

picttoppm reads a PICT file (version 1 or 2) and outputs a PPM image.

This is useful as the first step in converting a scanned image to something that can be displayed on Unix.

OPTIONS

-fontdir *file*

Make the list of BDF fonts in ''file'' available for use by *picttoppm* when drawing text. See below for the format of the fontdir file.

-fullres Force any images in the PICT file to be output with at least their full resolution. A PICT file may indicate that a contained image is to be scaled down before output. This option forces images to retain their sizes and prevent information loss. Use of this option disables all PICT operations except images.

-noheader

Do not skip the 512 byte header that is present on all PICT files. This is useful when you have PICT data that was not stored in the data fork of a PICT file.

-quickdraw

Execute only pure quickdraw operations. In particular, turn off the interpretation of special PostScript printer operations.

-verbose

Turns on verbose mode which prints a a whole bunch of information that only *picttoppm* hackers really care about.

LIMITATIONS

The PICT file format is a general drawing format. *picttoppm* does not recognize all the drawing commands, but it does fully implement all image commands and mostly implement line, rectangle, polgon and text drawing. It is useful for converting scanned images and some drawing conversion.

Memory is used very liberally with at least 6 bytes needed for every pixel. Large bitmap PICT files will likely run your computer out of memory.

FONT DIR FILE FORMAT

picttoppm has a built in default font and your local installer probably provided adequate extra fonts. You can point *picttoppm* at more fonts which you specify in a font directory file. Each line in the file is either a comment line which must begin with ‘‘#’’ or font information. The font information consists of 4 whitespace spearated fields. The first is the font number, the second is the font size in pixels, the third is the font style and the fourth is the name of a BDF file containing the font. The BDF format is defined by the X window system and is not described here.

The font number indicates the type face. Here is a list of known font numbers and their faces.

0 Chicago
1 application font
2 New York
3 Geneva
4 Monaco
5 Venice
6 London
7 Athens
8 San Franciso
9 Toronto
11 Cairo
12 Los Angeles
20 Times Roman
21 Helvetica
22 Courier
23 Symbol
24 Taliesin

The font style indicates a variation on the font. Multiple variations may apply to a font and the font style is the sum of the variation numbers which are:

1 Boldface
2 Italic
4 Underlined
8 Outlined
16 Shadow
32 Condensed
64 Extended

Obviously the font defintions are strongly related to the Macintosh. More font numbers and information about fonts can be found in Macintosh documentation.

SEE ALSO

Inside Macintosh volumes 1 and 5, **ppmtopict**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

pjtoppm - convert an HP PaintJet file to a PPM image

SYNOPSIS

pjtoppm

[*paintjet*]

DESCRIPTION

This program is part of **Netpbm**(1).

pjtoppm reads an HP PaintJet file as input and converts it into a PPM image. This was a quick hack to save some trees, and it only handles a small subset of the paintjet commands. In particular, it will only handle enough commands to convert most raster image files.

REFERENCES

HP PaintJet XL Color Graphics Printer User's Guide

SEE ALSO

ppmtopj(1)

AUTHOR

Table Of Contents

NAME

pktopbm - convert packed (PK) format font into PBM

SYNOPSIS

pktopbm pkfile[.pk] [-x width] [-y height] [-c num] pbmfile ...

DESCRIPTION

This program is part of **Netpbm**(1).

pktopbm reads a packed (PK) font file as input, and produces PBM images as output. If the filename '-' is used for any of the filenames, the standard input stream (or standard output where appropriate) will be used. If either the width or height is specified, this value will be used for all bitmaps produced. Also if one or both values are specified, the bitmap will be relocated with the hoffset and voffset given in the pkfile. The basepoint will be placed in the lower left corner of the bitmap if the bitmap is bigger than the specified size it will be truncated at the top or right.

OPTIONS

-c *num* Sets the character number of the next bitmap written to *num*.

-x *width*

Sets the width of the image in columns.

-y *width*

Sets the height of the image in rows.

SEE ALSO

pbmtopk(1), **pbm**(1)

AUTHOR

Adapted from Tom Rokicki's pxtopk by Angus Duggan <*ajcd@dcs.ed.ac.uk*>. <*bartel@informatik.tu-muenchen.de*> in March 1995.

Table Of Contents

NAME

pngtopnm - convert a PNG image into a PNM image

SYNOPSIS

pngtopnm [**-verbose**] [**-alpha** | **-mix**] [**-background**=*color*] [**-gamma**=*value*] [**-text**=*filename*] [**-time**] [*pngfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pngtopnm reads a PNG image (Portable Network Graphics) as input and produces a PPM image as output. The type of the output file depends on the input file - if it's black & white, **pngtopnm** creates a PBM file. If it's grayscale, **pngtopnm** creates a PGM file. Otherwise, it creates a PPM file.

OPTIONS

-verbose

Display various information about the input PNG image and the conversion process.

If you want even more information about the PNG image, use **pngcheck** (not part of Netpbm).

-alpha Output the alpha channel or transparency mask of the image. The result is either a PBM file or a PGM file, depending on whether different levels of transparency appear.

-mix Compose the image with the transparency or alpha mask against a background. The background color is determined by the bKGD chunk in the PNG, except that you can override it with **-background**. If the PNG has no bKGD chunk and you don't specify **-background**, the background color is white.

-background=*color*

This option specifies the background color with which to mix the image when you specify **-mix**.

color is as described for the argument of the **ppm_parsecolor()** library routine .

Examples:

- `-background=rgb:01/ff/80`
- `-background=rgbi:1/255/128`

If you don't specify **-background**, the background color is what is specified in the PNG image, and if the PNG doesn't specify anything, white.

You cannot specify **-background** unless you also specify **-mix**. Before Netpbm 10.27 (March 2005), you could specify **-background** with **-mix** and it was just ignored. (This caused a usability problem).

-gamma= *value*

Converts the image to a new display-gamma value. If a gAMA chunk is present in the *png-file*, **pngtopnm** uses the specified image-gamma value. If not, **pngtopnm** considers the image-gamma to be 1.0. Based on the image-gamma and the display-gamma given with this option, **pngtopnm** adjusts the colors written to the *pnm-file*.

Because the gammas of uncompensated monitors are around 2.6, which results in an image-gamma of 0.45, some typical situations are: when the image-gamma is 0.45 (use -verbose to check) and the picture is too light, your system is gamma-corrected, so convert with '-gamma 1.0'. When no gAMA chunk is present or the image-gamma is 1.0, use 2.2 to make the picture lighter and 0.45 to make the picture darker.

-text=*file*

Writes the tEXt and zTXt chunks to a file, in a format as described in the **pnmtopng** user manual. These chunks contain text comments or annotations.

-time Prints the tIME chunk to stderr.

SEE ALSO

pnmtopng(1), **ptot**, **pnmgamma**(1), **pnm**(1)

For information on the PNG format, see http://schaik.com/png .

NOTE

A PNG image contains a lot of information that can't be represented in Netpbm formats. Therefore, you lose information when you convert to another format with "pngtopnm | pnmtoxxx". If there is a specialized converter that converts directly to the other format, e.g. **ptot** to convert from PNG to TIFF, you'll get better results using that.

LIMITATIONS

There could be an option to read the comment text from pnm comments instead of a separate file.

The program could be much faster, with a bit of code optimizing. As with any Netpbm program, speed always takes a back seat to quick present and future development.

AUTHORS

Table Of Contents

NAME

pnmalias - antialias a PNM image

SYNOPSIS

pnmalias

[**-bgcolor** *color*]

[**-fgcolor** *color*]

[**-bonly**]

[**-fonly**]

[**-balias**]

[**-falias**]

[**-weight** *w*]

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmalias reads a PNM image as input, and applies anti-aliasing to background and foreground pixels. If the input file is a PBM, **pnmalias** promots the output anti-aliased image to a PGM, and prints a message informing the user of the change in format.

OPTIONS

-bgcolor *colorb* sets the background color the *colorb*.

-fgcolor *colorf* sets the foreground color to *colorf*.

Pixels with these values will be anti-aliased. By default, **pnmalias** takes the background color to be black, and foreground color to be white.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

Note that even when dealing with PGMs, background and foreground colors need to be specified in the fashion described above. In this case, **pnmalias** takes the background and foreground pixel values to be the value of the red component for the specified color.

-bonly says to apply anti-aliasing only to the background pixels.

-fonly says to apply anti-aliasing only to the foreground pixels.

-balias says to apply anti-aliasing to all pixels surrounding background pixels.

-falias says to apply anti-aliasing to all pixels surrounding foreground pixels.

If you specify neither **-balias** nor **-falias**, **pnmalias** applies anti-aliasing only among neighboring background and foreground pixels.

-weight *w* says to use *w* as the central weight for the aliasing filter. *w* must be a real number in the range $0 < w < 1$. The lower the value of *w* is, the 'blurrier' the output image is. The default is w = 1/3.

SEE ALSO

pbmtext(1), **pnmsmooth**(1), **pnm**(1)

AUTHOR

NAME

pnmarith - perform arithmetic on two PNM images

DESCRIPTION

This program is part of **Netpbm**(1).

Starting with Netpbm 10.3, **pnmarith** is obsolete. Use **pamarith**(1)**instead.**

pamarith is backward compatible with **pnmarith** except where your input images have different depths. **pnmarith** would convert the one with the lesser depth to have the higher depth before doing the arithmetic. With **pamarith**, you must do that step separately, using **pgmtoppm**.

Table Of Contents

NAME

pnmcat - concatenate Netpbm images

SYNOPSIS

pnmcat

{**-leftright** | **-lr** | **-topbottom** | **-tb**}

[**-white**|**-black**]

[**-jtop**|**-jbottom**|**-jcenter**]

pnmfile ...

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmcat reads one or more PNM images as input, concatenates them either left to right or top to bottom, and produces a single PNM image as output.

To do the reverse, you might use **pamdice** to split an image up into smaller ones of equal size or **pamcut** to chop off part of an image or extract part of an image.

pnmtile concatenates a single input image to itself repeatedly.

OPTIONS

If the PNM images are not all the same height (left-right) or width (top-bottom), the smaller ones have to be justified with the largest. By default, **pnmcat** centers them, but you can specify justification to one side or the other with one of the **-j***xxx* options. So, **-topbottom -jleft** would stack the PNMs on top of each other, flush with the left edge.

The **-white** and **-black** options specify what color to use to fill in the extra space when doing this justification. If neither is specified, **pnmcat** chooses whichever seems to be right for the images.

SEE ALSO

pamdice(1), **pnmtile**(1), **pamcut**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmcolormap - create quantization color map for a Netpbm image

SYNOPSIS

pnmcolormap

[**-center**|**-meancolor**|**-meanpixel**]

[**-spreadbrightness**|**-spreadluminosity**]

[**-sort**]

[**-square**]

ncolors|**all**

[*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmcolormap reads a PNM image as input, chooses *ncolors* colors to best represent the image and writes a PNM color map defining them as output.

You can use this map as input to **pnmremap** on the same input image to quantize the colors in that image, I.e. produce a similar image with fewer colors. **pnmquant** does both the **pnmcolormap** and **pnmremap** steps for you.

A PNM colormap is a PNM image of any dimensions that contains at least one pixel of each color in the set of colors it represents. The ones **pnmcolormap** generates have exactly one pixel of each color, except where padding is necessary with the **-square** option.

The quantization method is Heckbert's 'median cut'. See QUANTIZATION METHOD .

If the input image is a PPM, the output image is a PPM. If the input image is a PBM or PGM, the output colormap is a PGM. Note that a colormap of a PBM image is not very interesting.

The colormap generally has the same maxval as the input image, but **pnmcolormap** may reduce it if there are too many colors in the input, as part of its quantization algorithm.

If you want to create a colormap without basing it on the colors in an input image, see **ppmcolors**.

PARAMETERS

The single parameter, which is required, is the number of colors you want in the output colormap. **pnmcolormap** may produce a color map with slightly fewer colors than that. You may specify **all** to get a colormap of every color in the input image (no quantization).

OPTIONS

-sort This option causes the output colormap to be sorted by the red component intensity, then the green, then the blue in ascending order. This is an insertion sort, so it is not very fast on large colormaps. Sorting is useful because it allows you to compare two sets of colors.

-square

By default, **pnmcolormap** produces as the color map a PPM image with one row and one column for each color in the colormap. This option causes **pnmcolormap** instead to produce a

PPM image that is within one row or column of being square, with the last pixel duplicated as necessary to create a number of pixels which is such an almost-perfect square.

-verbose

This option causes **pnmcolormap** to display messages to Standard Error about the quantization..TP **-center**

-meancolor

-meanpixel

-spreadbrightness

-spreadluminosity

These options control the quantization algorithm. See QUANTIZATION METHOD .

QUANTIZATION METHOD

A quantization method is a way to choose which colors, being fewer in number than in the input, you want in the output. **pnmcolormap** uses Heckbert's 'median cut' quantization method.

This method involves separating all the colors into 'boxes,' each holding colors that represent about the same number of pixels. You start with one box and split boxes in two until the number of boxes is the same as the number of colors you want in the output, and choose one color to represent each box.

When you split a box, you do it so that all the colors in one sub-box are 'greater' than all the colors in the other. 'Greater,' for a particular box, means it is brighter in the color component (red, green, blue) which has the largest spread in that box. **pnmcolormap** gives you two ways to define 'largest spread.': 1) largest spread of brightness; 2) largest spread of contribution to the luminosity of the color. E.g. red is weighted much more than blue. Select among these with the **-spreadbrightness** and **-spreadluminosity** options. The default is **-spreadbrightness**.

pnmcolormap provides three ways of choosing a color to represent a box: 1) the center color - the color halfway between the greatest and least colors in the box, using the above definition of 'greater'; 2) the mean of the colors (each component averaged separately by brightness) in the box; 3) the mean weighted by the number of pixels of a color in the image.

Note that in all three methods, there may be colors in the output which do not appear in the input at all.

Select among these with the **-center**, **-meancolor**, and **-meanpixel** options. The default is **-center**.

REFERENCES

'Color Image Quantization for Frame Buffer Display' by Paul Heckbert, SIGGRAPH '82 Proceedings, page 297.

SEE ALSO

pnmremap(1), **pnmquant**(1), **ppmquantall**(1), **pnmdepth**(1), **ppmdither**(1), **ppmquant**(1), **ppm**(1)

HISTORY

Before Netpbm 10.15 (April 2003), **pnmcolormap** used a lot more memory for large images because it kept the entire input image in memory. Now, it processes it a row at a time, but because it sometimes must make multiple passes through the image, it first copies the input into a temporary seekable file if it is not already in a seekable file.

pnmcolormap first appeared in Netpbm 9.23 (January 2002). Before that, its function was available only as part of the function of **pnmquant** (which was derived from the much older **ppmquant**). Color quantization really has two main subfunctions, so Netpbm 9.23 split it out into two separate programs:

pnmcolormap and **pnmremap** and then Netpbm 9.24 replaced **pnmquant** with a program that simply calls **pnmcolormap** and **pnmremap**.

AUTHOR

Table Of Contents

NAME

pnmcomp - composite (overlay) two PNM images together

SYNOPSIS

pnmcomp

[**-align=**{**left**|**center**|**right**| **beyondleft**|**beyondright**}] [**-valign=**{**top**|**middle**|**bottom**| **above**|**below**}] [**-xoff=***X*] [**-yoff=***Y*] [**-alpha=***alpha-pgmfile*] [**-invert**] [**-opacity=***opacity] overlay_file [underlying_file [output_file]]*

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmcomp was obsoleted by **pamcomp**(1),**introduced**with**Netpbm**10.21 (March 2004). **pamcomp** is backward compatible with **pnmcomp**, plus adds many additional functions, including the ability to process PAM images, and tends to produce better transparency results.

pnmcomp remains in the Netpbm package because it probably has fewer bugs for now than **pamcomp**, and is faster. Some day, **pnmcomp** will probably become an alias for **pamcomp**.

You can use the **pamcomp** documentation for **pnmcomp**, considering the following differences:

- **pnmcomp** options are a subset of **pamscale**'s, as documented above.
- **pnmcomp** always assumes the input is linear, as **pamcomp** does when you specify its **-linear** option.
- **pnmcomp** cannot process PAM images.

Table Of Contents

NAME

pnmconvol - general MxN convolution on a PNM image

SYNOPSIS

pnmconvol

convolution_matrix_file [-nooffset] [*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmconvol reads two PNM images as input, convolves the second using the first, and writes a PNM image as output.

Convolution means replacing each pixel with a weighted average of the nearby pixels. The weights and the area to average are determined by the convolution matrix (sometimes called a convolution kernel), which you supply by way of the PNM image in the file you identify with the *convolution_matrix_file* argument. There are two ways **pnmconvol** interprets the PNM convolution matrix image pixels as weights: with offsets, and without offsets.

The simpler of the two is without offsets. That is what happens when you specify the **-nooffset** option. In that case, **pnmconvol** simply normalizes the sample values in the PNM image by dividing by the maxval.

For example, here is a sample convolution file that causes an output pixel to be a simple average of its corresponding input pixel and its 8 neighbors, resulting in a smoothed image:

```
P2
3 3
18
2 2 2
2 2 2
2 2 2
```

pnmconvol divides each of the sample values (2) by the maxval (18) so the weight of each of the 9 input pixels gets is 1/9, which is exactly what you want to keep the overall brightness of the image the same. **pnmconvol** creates an output pixel by multiplying the values of each of 9 pixels by 1/9 and adding.

Note that with maxval 18, the range of possible values is 0 to 18. After scaling, the range is 0 to 1.

For a normal convolution, where you're neither adding nor subtracting total value from the image, but merely moving it around, you'll want to make sure that all the scaled values in (each plane of) your convolution PNM add up to 1, which means all the actual sample values add up to the maxval.

When you *don't* specify **-nooffset**, **pnmconvol** applies an offset, the purpose of which is to allow you to indicate negative weights even though PNM sample values are never negative. In this case, **pnmconvol** subtracts half the maxval from each sample and then normalizes by dividing by half the maxval. So to get the same result as we did above with **-nooffset**, the convolution matrix PNM image would have to look like this:

```
P2
3 3
18
10 10 10
```

10 10 10
10 10 10

To see how this works, do the above-mentioned offset: 10 - 18/2 gives 1. The normalization step divides by 18/2 = 9, which makes it 1/9 - exactly what you want. The equivalent matrix for 5x5 smoothing would have maxval 50 and be filled with 26.

Note that with maxval 18, the range of possible values is 0 to 18. After offset, that's -9 to 9, and after normalizing, the range is -1 to 1.

For a normal convolution, where you're neither adding nor subtracting total value from the image, but merely moving it around, you'll want to make sure that all the offset, scaled values in (each plane of) your convolution PNM add up to 1. That means the actual sample values, less half the maxval, add up to half the maxval as in the example above.

The convolution file will usually be a PGM, so that the same convolution gets applied to each color component. However, if you want to use a PPM and do a different convolution to different colors, you can certainly do that.

At the edges of the convolved image, where the convolution matrix would extend over the edge of the image, **pnmconvol** just copies the input pixels directly to the output.

The convolution computation can result in a value which is outside the range representable in the output. When that happens, **pnmconvol** just clips the output, which means brightness is not conserved.

HISTORY

The **-nooffset** option was new in Netpbm 10.23 (July 2004).

SEE ALSO

pnmsmooth(1), **pgmmorphconv**(1), **pnmnlfilt**(1), **pgmkernel**(1), **pamgauss**(1), **pnm**(1)

AUTHORS

 Modified 26 November 1994 by Mike Burns, *burns@chem.psu.edu*

Table Of Contents

NAME

pnmcrop - crop a PNM image

SYNOPSIS

pnmcrop

[**-white**|**-black**|**-sides**]

[**-left**]

[**-right**]

[**-top**]

[**-bottom**]

[*pnmfile*]

All options may be abbreviated to their shortest unique prefix or specified with double hyphens.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmcrop reads a PBM, PGM, or PPM image as input, removes borders that are the background color, and produces the same type of image as output.

If you don't specify otherwise, **pnmcrop** assumes the background color is whatever color the top left and right corners of the image are and if they are different colors, something midway between them. You can specify that the background is white or black with the **-white** and **-black** options or make **pnmcrop** base its guess on all four corners instead of just two with **-sides**.

By default, **pnmcrop** chops off any stripe of background color it finds, on all four sides. You can tell **pnmcrop** to remove only specific borders with the **-left**, **-right**, **-top**, and **-bottom** options.

If you want to chop a specific amount off the side of an image, use **pamcut**.

If you want to add different borders after removing the existing ones, use **pnmcat** or **pamcomp**.

OPTIONS

-white Take white to be the background color. **pnmcrop** removes borders which are white.

-black Take black to be the background color. **pnmcrop** removes borders which are black.

-sides Determine the background color from the colors of the four corners of the input image. **pnmcrop** removes borders which are of the background color.

If at least three of the four corners are the same color, **pnmcrop** takes that as the background color. If not, **pnmcrop** looks for two corners of the same color in the following order, taking the first found as the background color: top, left, right, bottom. If all four corners are different colors, **pnmcrop** assumes an average of the four colors as the background color.

The **-sides** option slows **pnmcrop** down, as it reads the entire image to determine the background color in addition to the up to three times that it would read it without **-sides**.

-left Remove any left border.

-right Remove any right border.

-top Remove any top border.

-bottom
Remove any bottom border.

-verbose
Print on Standard Error information about the processing, including exactly how much is being cropped off of which sides.

SEE ALSO

pamcut(1), **pamfile**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmcut - cut a rectangle out of a PBM, PGM, or PPM image

SYNOPSIS

pnmcut

[**-left** *leftcol*]

[**-right** *rightcol*]

[**-top** *toprow*]

[**-bottom** *bottomrow*]

[**-width** *width*]

[**-height** *height*]

[**-pad**]

[**-verbose**]

[*left top width height*]

[*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmcut was obsoleted by **pamcut**(1),**introduced**with**Netpbm**9.20**(May** 2001). **pamcut** is backward compatible with **pnmcut**, plus adds many additional functions, including the ability to process PAM images.

pnmcut remains in the Netpbm package because it probably has fewer bugs for now than **pamcut**. Some day, **pnmcut** will probably become an alias for **pamcut**.

You can use the **pamcut** documentation for **pnmcut**, as long as you avoid any function which it says was added after Netpbm 9.20.

Table Of Contents

NAME

pnmdepth - change the maxval in a PNM image

SYNOPSIS

pnmdepth

newmaxval

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmdepth reads a PNM image as input, scales all the pixel values, and writes out the image with the new maxval. Scaling the colors down to a smaller maxval will result in some loss of information.

Be careful of off-by-one errors when choosing the new maxval. For instance, if you want the color values to be five bits wide, use a maxval of 31, not 32.

One important use of **pnmdepth** is to convert a new format 2-byte-per-sample PNM file to the older 1-byte-per-sample format. Before April 2000, essentially all raw (binary) format PNM files had a maxval less than 256 and one byte per sample, and many programs may rely on that. If you specify a *newmaxval* less than 256, the resulting file should be readable by any program that worked with PNM files before April 2000.

SEE ALSO

pnm(1), **pnmquant**(1), **ppmdither**(1)

AUTHOR

NAME

pnmenlarge - replaced by pamenlarge

DESCRIPTION

This program is part of **Netpbm**(1).

pnmenlarge was replaced in Netpbm 10.25 (October 2004) by **pamenlarge**(1).

pamenlarge is backward compatible with **pnmenlarge**, but works on PAM images too.

NAME

pnmfile - replaced by pamfile

DESCRIPTION

This program is part of **Netpbm**(1).

pnmfile was replaced in Netpbm 10.9 (September 2002) by **pamfile**(1).

pamfile is backward compatible with **pnmfile**, but works on PAM images too.

NAME

pnmflip − see http://netpbm.sourceforge.net/doc//pnmflip.html

DESCRIPTION

pnmflip is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//pnmflip.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

pnmgamma - perform gamma correction on a PNM image

SYNOPSIS

pnmgamma

[**-ungamma**]

[**-cieramp**|**-srgbramp**]

[*value*

[*pnmfile*]] **pnmgamma**

[**-ungamma**]

[**-cieramp**|**-srgbramp**]

redgamma greengamma bluegamma

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

Pnmgamma performs gamma correction on pseudo-PNM images.

The PPM format specification specifies that certain sample values in a file represent certain light intensities in an image. In particular, they specify that the sample values are directly proportional to gamma-corrected intensity values. The gamma correction they specify is CIE Rec. 709.

However, people sometimes work with approximations of PPM and PGM where the relationship between the image intensities and the sample values are something else. For example, the sample value might be directly proportional to the intensity with no gamma correction (often called 'linear intensity'). Or a different gamma transfer function may be used.

pnmgamma allows you to manipulate the transfer function, thus working with and/or creating pseudo-PPM files that are useful for various things.

For example, if you feed a true PPM to **pnmgamma -cieramp -ungamma**, you get as output a file which is PPM in every respect except that the sample values are directly proportional to the light intensities in the image. If you feed such a file to **pnmgamma -cieramp**, you get out a true PPM.

The situation for PGM images is analogous. And **pnmgamma** treats PBM images as PGM images.

When you feed a linear PPM image to a display program that expects a true PPM, the display appears darker than it should, so **pnmgamma** has the effect of lightening the image. When you feed a true PPM to a display program that expects linear sample values, and therefore does a gamma correction of its own on them, the display appears lighter than it should, so **pnmgamma** with a gamma value less than one (the multiplicative inverse of whatever gamma value the display program uses) has the effect of darkening the image.

PARAMETERS

The only parameters are the specification of the input image file and the gamma values. Every gamma transfer function **pnmgamma** uses contains an exponent, which is the gamma value, and you can choose that value.

Furthermore, you can choose different values for each of the three RGB components. If you specify only one gamma value, **pnmgamma** uses that value for all three RGB components.

If you don't specify any gamma parameters, **pnmgamma** chooses a default. For the transfer functions defined by standards, the default is the value defined by the standard. If you specify anything else, you will be varying from the standard. For the simple power function transfer function, the default gamma is 1/.45.

OPTIONS

-ungamma

Apply the inverse of the specified transfer function (i.e. go from gamma-corrected nonlinear intensities to linear intensities).

-cieramp

Use the CIE Rec. 709 gamma transfer function. Note that it is true CIE Rec. 709 only if you use the default gamma value (i.e. don't specify any gamma parameters). This transfer function is a power function modified with a linear ramp near black.

If you specify neither **-cieramp** nor **-srgbramp**, the transfer function defaults to a simple power function.

-srgbramp

Use the Internation Electrotechnical Commission (IEC) SRGB gamma transfer function (as specified in the standard IEC 61966-2-1). Note that it is true SRGB only if you use the default gamma value (i.e. don't specify any gamma parameters). This transfer function is like the one selected by **-cieramp**, but with different constants in it.

Note that SRGB is often spelled 'sRGB'. In this document, we use standard English typography, though, which doesn't allow for that kind of capitalization.

If you specify neither **-cieramp** nor **-srgbramp**, the transfer function defaults to a simple power function.

WHAT IS GAMMA?

A good explanation of gamma is in Charles Poynton's Gamma FAQ at http://www.poynton.com/GammaFAQ.html (1) and Color FAQ at http://www.poynton.com/ColorFAQ.html (1).

In brief: The simplest way to code an image is by using sample values that are directly proportional to the intensity of the color components. But that wastes the sample space because the human eye can't discern differences between low-intensity colors as well as it can between high-intensity colors. So instead, we pass the light intensity values through a transfer function that makes it so that changing a sample value by 1 causes the same level of perceived color change anywhere in the sample range. We store those resulting values in the image file. That transfer function is called the gamma transfer function and the transformation is called gamma correcting.

Virtually all image formats, either specified or de facto, use gamma-corrected values for their sample values.

What's really nice about gamma is that by coincidence, the inverse function that you have to do to convert the gamma-corrected values back to real light intensities is done automatically by CRTs. You just apply a voltage to the CRT's electron gun that is proportional to the gamma-corrected sample value, and the intensity of light that comes out of the screen is close to the intensity value you had before you applied the gamma transfer function!

And when you consider that computer video devices usually want you to store in video memory a value proportional to the signal voltage you want to go to the monitor, which the monitor turns into a proportional drive voltage on the electron gun, it is really convenient to work with gamma-corrected sample values.

SEE ALSO

pnm(1)

AUTHOR

Table Of Contents

NAME

pnmhisteq - histogram equalise a PNM image

SYNOPSIS

pnmhisteq

[**-gray**]

[**-rmap** *pgmfile*]

[**-wmap** *pgmfile*]

[**-verbose**]

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmhisteq increases the contrast of a portable graymap or pixmap through the technique of *histogram equalisation*[1].

A histogram of the luminance of pixels in the map is computed, from which a transfer function is calculated which spreads out intensity levels around histogram peaks and compresses them at troughs. This has the effect of using the available levels of intensity more efficiently and thereby increases the detail visible in the image.

Mathematically, if *N[i]* is the number of pixels of luminosity *i* in the image and *T* is the total number of pixels, luminosity *j* is replaced by:

```
 j
---
    > N[i] / T
/
---
i=0
```

If you're processing a related set of images, for example frames of an animation, it's generally best to apply the same intensity map to every frame, since otherwise you'll get distracting frame-to-frame changes in the brightness of objects. **pnmhisteq**'s **-wmap** option allows you to save, as a portable graymap, the luminosity map computed from an image (usually a composite of the images you intend to process created with **pnmcat**). Then, you can subsequently process each of the individual images using the luminosity map saved in the file, supplied with the **-rmap** option.

OPTIONS

-gray When processing a pixmap, only gray pixels (those with identical red, green, and blue values) are included in the histogram and modified in the output image. This is a special purpose option intended for images where the actual data are gray scale, with color annotations you don't want modified. Weather satellite images that show continent outlines in color are best processed using this option. The option has no effect when the input is a graymap.

-rmap *mapfile*

Process the image using the luminosity map specified by the PGM file *mapfile.*

The PGM image, usually created by an earlier run of **pnmhisteq** with the **-wmap** option,

contains a single row with number of columns equal to the maxval (greatest intensity value) of the image. Each pixel in the image is transformed by looking up its luminosity in the corresponding column in the map file and changing it to the value given by that column.

-wmap *mapfile*
Creates a PGM file *mapfile*, containing the luminosity map computed from the histogram of the input image. This map file can be read on subsequent runs of **pnmhisteq** with the **-rmap** option, allowing a group of images to be processed with an identical map.

-verbose
Prints the histogram and luminosity map on standard error.

You can abbreviate any option to its shortest unique prefix.

LIMITATIONS

Histogram equalisation is effective for increasing the visible detail in scientific imagery and in some continuous-tone pictures. It is often too drastic, however, for scanned halftone images, where it does an excellent job of making halftone artifacts apparent. You might want to experiment with **pgnnorm**, **ppmnorm**, and **pnmgamma** for more subtle contrast enhancement.

The luminosity map file supplied by the **-rmap** option must have the same maxval as the input image. This is always the case when the map file was created by the **-wmap** option of **pnmhisteq**. If this restriction causes a problem, simply adjust the maxval of the map with **pnmdepth** to agree with the input image.

If the input is a PBM file (on which histogram equalisation is an identity operation), the only effect of passing the file through **pnmhisteq** will be the passage of time.

SEE ALSO

pgmnorm(1), **pnm**(1), **pnmcat**(1), **pnmdepth**(1), **pnmgamma**(1), **pnmnorm**(1)

[1] Russ, John C. The Image Processing Handbook. Boca Raton: CRC Press, 1992. Pages 105-110.

AUTHOR

Table Of Contents

NAME

pnmhistmap - draw a histogram for a PGM or PPM file

SYNOPSIS

pnmhistmap

[**-black**]

[**-white**]

[**-max** *N*]

[**-verbose**]

[*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmhistmap reads a PNM image as input and produces an image showing a histogram of the color (or gray) values in the input. A PGM input results in a PBM output. A PPM input results in a PPM output with three overlaid histograms: a red one for the red input, a green one for the green input, and a blue one for the blue input.

For example, from the following image produces the following histogram:

image histogram from image

If the input is PBM, **pnmhistmap** produces an error message and no output image.

OPTIONS

-red

-green

-blue Include the indicated color component in the output. If you specify none of these, **pnmhistmap** include all three.

These options are meaningless if the input is PGM.

These options were new in Netpbm 10.26 (January 2005). Before that, **pnmhistmap** always included all three color components.

-dots Plot the histogram as dots. By default, **pnmhistmap** plots bars.

Example of dots:.B -dots example

This option was new in Netpbm 10.26 (January 2005). Before that, **pnmhistmap** always plotted bars.

-lval *minpixval*

-rval *maxpixval*

These options specify the range of intensity values to include. **pnmhistmap** ignores intensities less than *minpixval* and greater than *maxpixval*. So the left side of the histogram corresponds to *minpixval* and the right side corresponds to *maxpixval*.

By default, **pnmhistmap** plots the entire possible range: zero to the maxval.

These options were new in Netpbm 10.26 (January 2005). Before that, **pnmhistmap** always plotted from zero to the maxval.

-height

-width These options specify the dimensions, in pixels of the histogram image.

The default height is 200 pixels.

The default width is one pixel for each plotted intensity value (so it's controlled by the maxval of the image and the **-lval** and **-rval** options). The 'count buckets' in the histogram are always one pixel wide. If you specify a width less than the number of plotted intensity values, a bucket represents more than one intensity value. If you specify a width greater that the number of plotted intensity values, some buckets represent no color (the count is zero).

This option was new in Netpbm 10.26 (January 2005). Before that, the dimensions were always what the default is today.

-black Ignore the count of black pixels when scaling the histogram.

-white Ignore the count of white pixels when scaling the histogram.

The -black and -white options, which can be used separately or together, are useful for images with a large percentage of pixels whose value is zero or 255, which can cause the remaining histogram data to become unreadbaly small. Note that, for pixmap inputs, these options apply to all colors; if, for example, the input has a large number of bright-red areas, you will probably want to use the -white option.

-max N

Force the scaling of the histogram to use N as the largest-count value. This is useful for inputs with a large percentage of single-color pixels which are not black or white.

-verbose

Report the progress of making the histogram, including the largest-count value used to scale the output.

LIMITATIONS

pnmhistmap assumes maxval is always 255. Images with a smaller maxval will only use the lower-value side of the histogram. You can overcome this either by piping the input through **pnmdepth** or by cutting and scaling the lower-value side of the histogram. Neither is a particularly elegant solution to the problem.

The program does not allow you to specify the output size.

SEE ALSO

pgmhist(1), **ppmhist**(1), **pgm**(1), **ppm**(1)

AUTHOR

Wilson H. Bent. Jr. (*whb@usc.edu*).

Table Of Contents

NAME

pnmindex - build a visual index of a bunch of PNM images

SYNOPSIS

pnmindex

[**-size** *N*]

[**-across** *N*]

[**-colors** *N*]

[**-black**]

[**-title** *title*]

[**-quant**|**-noquant**]

pnmfile...

DESCRIPTION

This program is part of **Netpbm**(1).

pnmindex creates an index image containing thumbnail (small) versions of a bunch of PNM files you supply.

pnmindex labels each thumbnail and, optionally, contains a title.

If you just want to concatenate some images together, use **pnmcat** for that. If you want to make a grid of the same image repeated over and over, that's **pnmtile**.

If you want to take apart the image you generated with **pnmindex**, use **pamdice** or **pamcut**.

OPTIONS

-size *N* The size of each thumbnail. The image is scaled to fit maximally inside a *N* x *N* pixel box without changing its aspect ratio. Default is 100.

-across *N*

The number of thumbnails in each row. Default is 6.

-colors *N*

The maximum number of colors allowed in the overall image. If it would otherwise have more colors than these, **pnmindex** quantizes the result. The default is 256.

However, this value is meaningless if you specify the **-noquant** option.

-black This controls the color of the padding between the images; normally it's white and the labels are black lettering on white background, but the **-black** option reverses this.

-title *title*

Specifies a title top place at the top of the image. Default is no title.

-quant Enables quanization (to the number of colors specified by **-colors**). Quantization is on by default but you can disable it with **-noquant.**

-noquant
See **-quant**.

SEE ALSO

pamscale(1), **pnmcat**(1), **pbmtext**(1), **pnmquant**(1), **pnm**(1)

AUTHOR

-title and **-noquant** added 2000 by John Heidemann.

NAME

pnminterp - replaced by pamstretch

DESCRIPTION

This program is part of **Netpbm**(1).

pnminterp was replaced in Netpbm 9.21 (December 2001) by **pamstretch**(1).

pamstretch is backward compatible with **pnminterp**, but also recognizes PAM input, including that with an alpha channel.

Table Of Contents

NAME

pnminvert - invert a PNM image

SYNOPSIS

pnminvert

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnminvert reads a PNM image as input, inverts it black for white, and produces a PNM image as output.

If the image is grayscale, **pnminvert** replaces a pixel with one of complementary brightness, i.e. if the original pixel has gamma-adjusted gray value G, the output pixel has gray value maxval - G.

If the image is color, **pnminvert** inverts each individual RGB component the same as for a grayscale image.

SEE ALSO

pnm(1)

AUTHOR

Table Of Contents

NAME

pnmmargin - add a border to a PNM image

SYNOPSIS

pnmmargin

[**-white**|**-black**|**-color**

colorspec]

size

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmmargin adds a border around a PNM image.

OPTIONS

You can specify the border color with the **-white**, **-black**, and **-color** flags. If no color is specified, the program makes a guess.

To remove a border of a specified size from an image, use **pamcut**. **pnmcrop** also removes borders, but determines by itself what is border and what is subject.

For lower level control, including to add different size borders to different sides of the image, look at **pnmcat**.

If all you're trying to do is get the image up to a certain required size, **pamcut** may be what you want.

SEE ALSO

pamcut(1) **pnmcrop**(1) **pnmcat**(1) **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmmontage - create a montage of PNM images

SYNOPSIS

pnmmontage

[**-header=***headerfile*]

[**-quality=***n*]

[**-prefix=***prefix*]

[**-0**|**-1**|**-2**|**...**|**-9**]

pnmfile...

DESCRIPTION

This program is part of **Netpbm**(1).

pnmmontage packs images of differing sizes into a minimum-area composite image, optionally producing a C header file with the locations of the subimages within the composite image.

OPTIONS

-header

Tells **pnmmontage** to write a C header file of the locations of the original images within the packed image. Each original image generates four #defines within the packed file: xxxX, xxxY, xxxSZX, and xxxSZY, where xxx is the name of the file, converted to all uppercase. The ouput also includes #defines OVERALLX and OVERALLY, which specifies the total size of the montage image.

-prefix Tells **pnmmontage** to use the specified prefix on all of the #defines it generates.

-quality

Before attempting to place the subimages, **pnmmontage** will calculate a minimum possible area for the montage; this is either the total of the areas of all the subimages, or the width of the widest subimage times the height of the tallest subimage, whichever is greater. **pnmmontage** then initiates a problem-space search to find the best packing; if it finds a solution that is (at least) as good as the minimum area times the quality as a percent, it will break out of the search. Thus, **-q 100** will find the best possible solution; however, it may take a very long time to do so. The default is **-q 200.**

-0, **-1**, ... **-9**

These options control the quality at a higher level than **-q**; **-0** is the worst quality (literally pick the first solution found), while **-9** is the best quality (perform an exhaustive search of problem space for the absolute best packing). The higher the number, the slower the computation. The default is **-5**.

NOTES

Using **-9** is excessively slow on all but the smallest image sets. If the anymaps differ in maxvals, then pnmmontage will pick the smallest maxval which is evenly divisible by each of the maxvals of the original images.

SEE ALSO

pnmcat(1), **pnmindex**(1), **pnm**(1), **pam**(1), **pbm**(1), **pgm**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

pnmnlfilt - non-linear filters: smooth, alpha trim mean, optimal estimation smoothing, edge enhancement.

SYNOPSIS

pnmnlfilt *alpha radius* [*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmnlfilt produces an output image where the pixels are a summary of multiple pixels near the corresponding location in an input image.

This program works on multi-image streams.

This is something of a swiss army knife filter. It has 3 distinct operating modes. In all of the modes each pixel in the image is examined and processed according to it and its surrounding pixels values. Rather than using the 9 pixels in a 3x3 block, 7 hexagonal area samples are taken, the size of the hexagons being controlled by the radius parameter. A radius value of 0.3333 means that the 7 hexagons exactly fit into the center pixel (ie. there will be no filtering effect). A radius value of 1.0 means that the 7 hexagons exactly fit a 3x3 pixel array.

Alpha trimmed mean filter (0.0 <= alpha <= 0.5)

The value of the center pixel will be replaced by the mean of the 7 hexagon values, but the 7 values are sorted by size and the top and bottom alpha portion of the 7 are excluded from the mean. This implies that an alpha value of 0.0 gives the same sort of output as a normal convolution (ie. averaging or smoothing filter), where radius will determine the 'strength' of the filter. A good value to start from for subtle filtering is alpha = 0.0, radius = 0.55 For a more blatant effect, try alpha 0.0 and radius 1.0

An alpha value of 0.5 will cause the median value of the 7 hexagons to be used to replace the center pixel value. This sort of filter is good for eliminating 'pop' or single pixel noise from an image without spreading the noise out or smudging features on the image. Judicious use of the radius parameter will fine tune the filtering. Intermediate values of alpha give effects somewhere between smoothing and 'pop' noise reduction. For subtle filtering try starting with values of alpha = 0.4, radius = 0.6 For a more blatant effect try alpha = 0.5, radius = 1.0

Optimal estimation smoothing. (1.0 <= alpha <= 2.0)

This type of filter applies a smoothing filter adaptively over the image. For each pixel the variance of the surrounding hexagon values is calculated, and the amount of smoothing is made inversely proportional to it. The idea is that if the variance is small then it is due to noise in the image, while if the variance is large, it is because of 'wanted' image features. As usual the radius parameter controls the effective radius, but it probably advisable to leave the radius between 0.8 and 1.0 for the variance calculation to be meaningful. The alpha parameter sets the noise threshold, over which less smoothing will be done. This means that small values of alpha will give the most subtle filtering effect, while large values will tend to smooth all parts of the image. You could start with values like alpha = 1.2, radius = 1.0 and try increasing or decreasing the alpha parameter to get the desired effect. This type of filter is best for filtering out dithering noise in both bitmap and color images.

Edge enhancement. (-0.1 >= alpha >= -0.9)

This is the opposite type of filter to the smoothing filter. It enhances edges. The alpha parameter controls the amount of edge enhancement, from subtle (-0.1) to blatant (-0.9). The radius parameter controls the effective radius as usual, but useful values are between 0.5 and 0.9. Try starting with values of alpha = 0.3, radius = 0.8

Combination use.

The various modes of **pnmnlfilt** can be used one after the other to get the desired result. For instance to turn a monochrome dithered image into a grayscale image you could try one or two passes of the smoothing filter, followed by a pass of the optimal estimation filter, then some subtle edge enhancement. Note that using edge enhancement is only likely to be useful after one of the non-linear filters (alpha trimmed mean or optimal estimation filter), as edge enhancement is the direct opposite of smoothing.

For reducing color quantization noise in images (ie. turning .gif files back into 24 bit files) you could try a pass of the optimal estimation filter (alpha 1.2, radius 1.0), a pass of the median filter (alpha 0.5, radius 0.55), and possibly a pass of the edge enhancement filter. Several passes of the optimal estimation filter with declining alpha values are more effective than a single pass with a large alpha value. As usual, there is a tradeoff between filtering effectiveness and loosing detail. Experimentation is encouraged.

References:

The alpha-trimmed mean filter is based on the description in IEEE CG&A May 1990 Page 23 by Mark E. Lee and Richard A. Redner, and has been enhanced to allow continuous alpha adjustment.

The optimal estimation filter is taken from an article 'Converting Dithered Images Back to Gray Scale' by Allen Stenger, Dr Dobb's Journal, November 1992, and this article references 'Digital Image Enhancement and Noise Filtering by Use of Local Statistics', Jong-Sen Lee, IEEE Transactions on Pattern Analysis and Machine Intelligence, March 1980.

The edge enhancement details are from **pgmenhance**(1),**which**is**taken**from**Philip** R. Thompson's 'xim' program, which in turn took it from section 6 of 'Digital Halftones by Dot Diffusion', D. E. Knuth, ACM Transaction on Graphics Vol. 6, No. 4, October 1987, which in turn got it from two 1976 papers by J. F. Jarvis et. al.

SEE ALSO

pgmenhance(1), **pnmconvol**(1), **pnm**(1)

AUTHOR

Graeme W. Gill *graeme@labtam.oz.au*

NAME

pnmnoraw - replaced by pnmtoplainpnm

DESCRIPTION

This program is part of **Netpbm**(1).

pnmnoraw was replaced in Netpbm 8.2 (March 2000) by **pnmtoplainpnm**(1),**which**was**obsoleted**by **pnmtopnm** in Netpbm 10.23 (July 2004).

pnmtoplainpnm was actually the same program; it was just renamed to make it clear that is just a format converter.

pnmtopnm is more general, in that it can go both directions. **pnmtopnm -plain** is the same as **pnmnoraw**.

Table Of Contents

NAME

pnmnorm - normalize the contrast in a Netbpm image

SYNOPSIS

pnmnorm

[**-bpercent** *N* | **-bvalue** *N*]

[**-wpercent** *N* | **-wvalue** *N*]

[**-keephues**]

[**-brightmax**]

[*ppmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmnorm reads a PNM image (PBM, PGM, or PPM). Normalizes the contrast by forcing the lightest pixels to white, the darkest pixels to black, and linearly rescaling the ones in between; and produces the same kind of file as output. This is pretty useless for a PBM image.

The program first determines a mapping of old brightness to new brightness. For each possible brightness of a pixel, the program determines a corresponding brightness for the output image.

Then for each pixel in the image, the program computes a color which has the desired output brightness and puts that in the output. With a color image, it is not always possible to compute such a color and retain any semblance of the original hue, so the brightest and dimmest pixels may only approximate the desired brightness.

Note that for a PPM image, this is different from separately normalizing the individual color components.

OPTIONS

By default, the darkest 2 percent of all pixels are mapped to black, and the lightest 1 percent are mapped to white. You can override these percentages by using the **-bpercent** and **-wpercent** flags, or you can specify the exact pixel values to be mapped by using the **-bvalue** and **-wvalue** flags. You can get appropriate numbers for the flags from **ppmhist**. If you just want to enhance the contrast, then choose values at elbows in the histogram; e.g. if value 29 represents 3% of the image but value 30 represents 20%, choose 30 for *bvalue*. If you want to lighten the image, then set *bvalue* to 0 and just fiddle with *wvalue*; similarly, to darken the image, set *wvalue* to maxval and play with *bvalue*.

If you specify both **-bvalue** and **-bpercent**, **pnmnorm** uses the one that produces the minimal change. The same goes for **-wvalue** and **-wpercent**.

If you want to maximize the change instead of minimizing it, just cascade two runs of **pnmnorm**, specifying values for the first and percentages for the second.

Before Netpbm 10.26 (December 2004), it was not valid to specify both **-bvalue** and **-bpercent** or **-wvalue** and **-wpercent**.

The **-keephues** option says to keep each pixel the same hue as it is in the input; just adjust its intensity. By default, **pnmnorm** normalizes contrast in each component independently (except that the meaning of the **-wpercent** and **-bpercent** options are based on the overall intensities of the colors, not each component taken separately). So if you have a color which is intensely red but dimly green, **pnmnorm**

would make the red more intense and the green less intense, so you end up with a different hue than you started with.

If you specify **-keephues**, **pnmnorm** would likely leave this pixel alone, since its overall intensity is medium.

-keephues can cause clipping, because a certain color may be below a target intensity while one of its components is saturated. Where that's the case, **pnmnorm** uses the maximum representable intensity for the saturated component and the pixel ends up with less overall intensity, and a different hue, than it is supposed to have.

This option is meaningless on grayscale images.

Before March 2002, there was no **-keephues** option.

The **-brightmax** option says to use the intensity of the most intense RGB component of a pixel as the pixel's brightness. By default, **pnmnorm** uses the luminosity of the color as its brightness.

This option is meaningless on grayscale images.

Before March 2002, there was no **-brightmax** option.

SEE ALSO

ppmhist(1), **pgmhist**(1), **pnmgamma**(1), **ppmbrighten**(1), **ppmdim**(1), **pnm**(1)

Table Of Contents

NAME

pnmpad - add borders to a PNM image

SYNOPSIS

pnmpad [**-verbose**] [**-white**|**-black**] [**-width=***pixels*] [**-halign=***ratio*] [**-left=***pixels*] [**-right=***pixels*] [**-height=***pixels*] [**-valign=***ratio*] [**-top=***pixels*] [**-bottom=***pixels*] [*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmpad reads a PNM image as input and outputs a PNM image that is the input image plus black or white borders of the sizes specified.

If you just need to convert an image to a certain size regardless of the original dimensions, **pamcut** with the **-pad** option may be a better choice.

OPTIONS

-verbose

Verbose output.

-white

-black Set pad color. Default is **-black**.

-left=*pixels*

-right=*pixels*

-width=*width*

-halign=*ratio*

Specify amount of left and right padding in pixels.

-left and **-right** directly specify the amount of padding added to the left and right sides, respectively, of the image.

Alternatively, you can specify **-width** and just one of **-left** and **-right** and **pnmpad** calculates the required padding on the other side based on the width of the input image. If the **-width** value is less than the width of the image plus the specified padding, the **-width** values is ignored.

If you specify all three of **-width**, **-left**, and **-right**, you must ensure that the **-left** and **-right** padding are sufficient to make the image at least as wide as **-width** specifies. Otherwise, **pnmpad** fails.

When you specify **-width** without **-left** or **-right**, and **-width** is larger than the input image, **pnmpad** chooses left and right padding amounts in a certain ratio. That ratio defaults to half, but you can set it to anything (from 0 to 1) with the **-halign** option. If the input image is already at least as wide as **-width** specifies, **pnmpad** adds no padding.

Common values for **-halign** are:

0.0 left aligned

0.5 center aligned (default)

1.0 right aligned

Before Netpbm 10.23 (July 2004), **pnmpad** did not allow the **-left** or **-right** option together with **-width**.

-top=*pixels*

-bottom=*pixels*

-height=*height*

-valign=*ratio*

These options determine the vertical padding. They are analogous to the horizontal padding options above.

HISTORY

Before February 2002, **pnmpad** had a different option syntax which was less expressive and not like conventional Netpbm programs. That syntax is still understood by **pnmpad** for backward compatibility, but not documented or supported for future use.

SEE ALSO

pbmmake(1), **pnmpaste**(1), **pamcut**(1), **pnmcrop**(1), **pamcomp**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

pnmpaste - paste a rectangle into a PNM image

SYNOPSIS

pnmpaste

[**-replace**|**-or**|**-and**|**-xor**]

frompnmfile x y

[*intopnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmpaste reads two PNM images as input and inserts the first image (the "pasted image") into the second (the "base image") at the specified location, and produces a PNM image the same size and type as the base image as output. If you don't specify the second file, **pnmpaste** reads the base image from Standard Input.

x and *y* specify the location in the base image at which to put the top left corner of the pasted image, *x* giving the horizontal position and *x* giving the vertical position. A nonnegative value indicates the number of pixels right of the right edge or below the top edge of the base image, while a negative value indicates the number of pixels right of the right edge or below the bottom edge (so x = -5 means 5 pixels left of the right edge).

If any part of the pasted image does not fit within the base image, **pnmpaste** fails.

This tool is most useful in combination with *pamcut*. For instance, if you want to edit a small segment of a large image, and your image editor cannot edit the large image, you can cut out the segment you are interested in, edit it, and then paste it back in.

Another useful companion tool is **pbmmask**.

pamcomp is a more general tool, except that it lacks the 'or,' 'and,' and 'xor' functions. **pamcomp** allows you to specify an alpha mask in order to have only part of the inserted image get inserted. So the inserted pixels need not be a rectangle. You can also have the inserted image be translucent, so the resulting image is a mixture of the inserted image and the base image.

OPTIONS

The optional flag specifies the operation to use when doing the paste. The default is **-replace**. The other logical operations are only allowed if both input images are PBM images. These operations act as if white is TRUE and black is FALSE.

You can abbreviate all options to their shortest unique prefix.

SEE ALSO

pamcomp(1), **pamcut**(1), **pnminvert**(1), **pnmarith**(1), **pbmmask**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmpsnr - compute the difference between two images (the PSNR)

SYNOPSIS

pnmpsnr

[*pnmfile1*]

[*pnmfile2*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmpsnr reads two PBM, PGM, or PPM files, or PAM equivalents, as input and prints the peak signal-to-noise ratio (PSNR) difference between the two images. This metric is typically used in image compression papers to rate the distortion between original and decoded image.

If the inputs are PBM or PGM, **pnmpsnr** prints the PSNR of the luminance only. Otherwise, it prints the separate PSNRs of the luminance, and chrominance (Cb and Cr) components of the colors.

The PSNR of a given component is the ratio of the mean square difference of the component for the two images to the maximum mean square difference that can exist between any two images. It is expressed as a decibel value.

The mean square difference of a component for two images is the mean square difference of the component value, comparing each pixel with the pixel in the same position of the other image. For the purposes of this computation, components are normalized to the scale [0..1].

The maximum mean square difference is identically 1.

So the higher the PSNR, the closer the images are. A luminance PSNR of 20 means the mean square difference of the luminances of the pixels is 100 times less than the maximum possible difference, i.e. 0.01.

SEE ALSO

pnm(1)

Table Of Contents

NAME

pnmquant - quantize the colors in a Netpbm image to a smaller set

SYNOPSIS

pnmquant [**-center**|**-meancolor**|**-meanpixel**] [**-floyd**|**-fs**] [**-nofloyd**|**-nofs**] [**-spreadbrightness**|**-spreadluminosity**] *ncolors* [*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmquant reads a PNM image as input. It chooses *ncolors* colors to best represent the image, maps the existing colors to the new ones, and writes a PNM image as output.

This program is simply a combination of **pnmcolormap** and **pnmremap**, where the colors of the input are remapped using a color map which is generated from the colors in that same input. The options have the same meaning as in those programs. See their documentation to understand **pnmquant**.

It is much faster to call **pnmcolormap** and **pnmremap** directly than to run **pnmquant**. You save the overhead of the Perl interpreter and creating two extra processes. **pnmquant** is just a convenience.

pnmquant did not exist before Netpbm 9.21 (January 2001). Before that, **ppmquant** did the same thing, but only on PPM images. **ppmquant** continues to exist, but is only a front end (for name compatibility) to **pnmquant**.

SEE ALSO

pnmcolormap(1), **pnmremap**(1), **ppmquantall**(1), **pnmdepth**(1), **ppmdither**(1), **ppmquant**(1), **pnm**(1)

Table Of Contents

NAME

pnmremap - replace colors in a PNM image with colors from another set

SYNOPSIS

pnmremap

[**-floyd**|**-fs**|**-nfloyd**|**-nofs**]

[**-firstisdefault**]

[**-verbose**]

[**-mapfile=***palettefile*]

[**-missingcolor=***color*]

[*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmremap replaces the colors in an input image with those from a palette you specify. Where colors in the input are present in the palette, they just stay the same in the output. But where the input contains a color that is not in the palette, **pnmremap** gives you three choices: 1) choose the closest color from the palette; 2) choose the first color from the palette; 3) use a color specified by a command option.

You can also dither, which means rather than mapping pixel by pixel, **pnmremap** uses colors from the palette to try to make multi-pixel regions of the output have the same average color as the input.

Two reasons to do this are: 1) you want to reduce the number of colors in the input image; and 2) you need to feed the image to something that can handle only certain colors.

To reduce colors, you can generate the palette with **pnmcolormap**.

By default, **pnmremap** maps an input color that is not in the palette to the closest color that *is* in the palette. Closest means with the smallest cartesian distance in the red, green, blue brightness space (smallest sum of the squares of the differences in red, gree, and blue ITU Rec 709 gamma-adjusted intensities).

You can instead specify a single default color for **pnmremap** to use for any color in the input image that is not in the palette. Use the **-missing** option for this.

You can also specify that the first color in the palette image is the default. Use the **-firstisdefault** option for this.

The palette is simply a PNM image. The colors of the pixels in the image are the colors in the palette. Where the pixels appear in the image, and the dimensions of the image, are irrelevant. Multiple pixels of the same color are fine. However, a palette image is typically a single row with one pixel per color.

If you specify **-missing**, the color you so specify is in the palette in addition to whatever is in the palette image.

For historical reasons, Netpbm sometimes calls the palette a 'colormap.' But it doesn't really map anything. **pnmremap** creates its own map, based on the palette, to map colors from the input image to output colors.

Palette/Image Type Mismatch

In the simple case, the palette image is of the same depth (number of planes, i.e. number of components in each tuple (pixel)) as the input image and **pnmremap** just does a straightforward search of the palette for each input tuple (pixel). In fact, **pnmremap** doesn't even care if the image is a visual image.

But what about when the depths differ? In that case, **pnmremap** converts the input image (in its own memory) to match the palette and then proceeds as above.

There are only two such cases in which **pnmremap** knows how to do the conversion when one of them is tuple type RGB, depth 3, and the other is tuple type GRAYSCALE or BLACKANDWHITE, depth 1; and vice versa.

In any other case, **pnmremap** fails.

Note that as long as your input and palette images are PNM, they'll always fall into one of the cases **pnmremap** can handle. There's an issue only if you're using some exotic PAM image.

Before Netpbm 10.27 (March 2005), **pnmremap** could not handle the case of a palette of greater depth than the input image.

In any case, the output image has the same tuple type and depth as the palette image.

Examples

pnmcolormap testimg.ppm 256 >palette.ppm

pnmremap -map=palette.ppm testimg.ppm >reduced_testimg.ppm

To limit colors to a certain set, a typical example is to create an image for posting on the World Wide Web, where different browsers know different colors. But all browsers are supposed to know the 216 'web safe' colors which are essentially all the colors you can represent in a PPM image with a maxval of 5. So you can do this:

pamseq 3 5 >websafe.pam

pnmremap -map=websafe.pam testimg.ppm >websafe_testimg.ppm

Another useful palette is one for the 8 color IBM TTL color set, which you can create with
pamseq 3 1 >ibmttl.pam

If you want to quantize one image to use the colors in another one, just use the second one as the palette. You don't have to reduce it down to only one pixel of each color, just use it as is.

The output image has the same type and maxval as the palette image.

PARAMETERS

There is one parameter, which is required: The file specification of the input PNM file.

OPTIONS

-mapfile=*palettefilename*

This names the file that contains the palette image.

This option is mandatory.

-floyd

-fs

-nofloyd

-nofs These options determine whether Floyd-Steinberg dithering is done. Without Floyd-Steinberg, the selection of output color of a pixel is based on the color of only the corresponding input pixel. With Floyd-Steinberg, multiple input pixels are considered so that the average color of an area tends to stay more the same than without Floyd-Steinberg. For example, if

you map an image with a black, gray, gray, and white pixel adjacent, to a palette that contains only black and white, it might result in an output of black, black, white, white. Pixel-by-pixel mapping would instead map both the gray pixels to the same color.

Floyd-Steinberg gives vastly better results on images where unmodified quantization has banding or other artifacts, especially when going to a small number of colors such as the above IBM set. However, it does take substantially more CPU time.

-fs is a synomym for **-floyd**. **-nofs** is a synonym for **-nofloyd**.

The default is **-nofloyd**.

-firstisdefault

This tells **pnmremap** to map any input color that is not in the palette to the first color in the palette (the color of the pixel in the top left corner of the palette image)

See DESCRIPTION .

If you specify **-firstisdefault**, the maxval of your input must match the maxval of your palette image.

-missingcolor=*color*

This specifies the default color for **pnmremap** to map to a color in the input image that isn't in the palette. *color* may or may not be in the palette image; it is part of the palette regardless.

If you specify **-missingcolor**, the maxval of your input must match the maxval of your palette image.

-verbose

Display helpful messages about the mapping process.

SEE ALSO

pnmcolormap(1), **pamseq**(1), **pnmquant**(1), **ppmquantall**(1), **pnmdepth**(1), **ppmdither**(1), **ppmquant**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

pnmrotate - rotate a PNM image by some angle

SYNOPSIS

pnmrotate [**-noantialias**] [**-background=***color*] *angle* [*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmrotate reads a PNM image as input. It rotates it by the specified angle and produces the same kind of PNM image as output.

The input is the file named by *pnmfile* or Standard Input if you don't specify *pnmfile*. The output goes to Standard Output.

The resulting image is a rectangle that contains the (rectangular) input image within it, rotated with respect to its bottom edge. The containing rectangle is as small as possible to contain the rotated image. The background of the containing image is a single color that **pnmrotate** determines to be the background color of the original image, or that you specify explicitly.

angle is in decimal degrees (floating point), measured counter-clockwise. It can be negative, but it should be between -90 and 90.

You should use **pamflip** instead for rotations that are a multiple of a quarter turn. It is faster and more accurate.

For rotations greater than 45 degrees you may get better results if you first use *pamflip* to do a 90 degree rotation and then *pnmrotate* less than 45 degrees back the other direction.

The rotation algorithm is Alan Paeth's three-shear method. Each shear is implemented by looping over the source pixels and distributing fractions to each of the destination pixels. This has an 'anti-aliasing' effect - it avoids jagged edges and similar artifacts. However, it also means that the original colors or gray levels in the image are modified. If you need to keep precisely the same set of colors, you can use the **-noantialias** option.

The program runs faster and uses less real memory with the **-noantialias** option. It uses a large amount of virtual memory either way, as it keeps a copy of the input image and a copy of the output image in memory, using 12 bytes per pixel for each. But with **-noantialias**, it accesses this memory sequentially in half a dozen passes, with only a few pages of memory at a time required in real memory.

In contrast, without **-noantialias**, the program's real memory working set size is one page per input image row plus one page per output image row. Before Netpbm 10.16 (June 2003), **-noantialias** had the same memory requirement.

OPTIONS

-background=*color*

This determines the color of the background on which the rotated image sits.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

By default, if you don't specify this option, **pnmrotate** selects what appears to it to be the background color of the original image. It determines this color rather simplisticly, by taking an average of the colors of the two top corners of the image.

This option was new in Netpbm 10.l5. Before that, **pnmrotate** always behaved as is the

default now.

-noantialias

This option forces **pnmrotate** to simply move pixels around instead of synthesizing output pixels from multiple input pixels. The latter could cause the output to contain colors that are not in the input, which may not be desirable. It also probably makes the output contain a large number of colors. If you need a small number of colors, but it doesn't matter if they are the exact ones from the input, consider using **pnmquant** on the output instead of using **-noantialias**.

Note that to ensure the output does not contain colors that are not in the input, you also must consider the background color. See the **-background** option.

REFERENCES

'A Fast Algorithm for General Raster Rotation' by Alan Paeth, Graphics Interface '86, pp. 77-81.

SEE ALSO

pnmshear(1), **pamflip**(1), **pnmquant**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmscale - scale a PNM image

SYNOPSIS

pnmscale
[
scale_factor
|
-xysize *cols rows*
|
-reduce *reduction_factor*
|
[**-xsize**=*cols* | **-width**=*cols* | **-xscale**=*factor*]
[**-ysize**=*rows* | **-height**=*rows* | **-yscale**=*factor*]
|
-pixels *n*
]
[**-verbose**]
[**-nomix**]
[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmscale was obsoleted by **pamscale**(1),**introduced**with**Netpbm**10.20 (January 2004). **pamscale** is backward compatible with **pnmscale**, plus adds many additional functions, including the ability to process PAM images, and tends to produce better results.

pnmscale remains in the current Netpbm package because it probably has fewer bugs for now than **pamscale**, and may have superior performance characteristics in some cases. Some day, **pnmscale** will probably become an alias for **pamscale**.

You can use the **pamscale** documentation for **pnmscale**, considering the following differences:

- **pnmscale** options are a subset of **pamscale**'s, as documented above.
- **pnmscale** always assumes the input is linear, as **pamscale** does when you specify its **-linear** option.
- **pnmscale** cannot process PAM images.

Table Of Contents

NAME

pnmscale - scale a PNM file quickly

DESCRIPTION

This program is part of **Netpbm**(1).

pnmscalefixed is the same thing as **pnmscale** except that it uses fixed point arithmetic internally instead of floating point, which makes it run faster. In turn, it is less accurate and may distort the image.

Use the **pnmscale** user manual with **pnmscalefixed**. This document only describes the difference.

pnmscalefixed uses fixed point 12 bit arithmetic. By contrast, **pnmscale** uses floating point arithmetic which on most machines is probably 24 bit precision. This makes **pnmscalefixed** run faster (30% faster in one experiment), but the imprecision can cause distortions at the right and bottom edges.

The distortion takes the following form: One pixel from the edge of the input is rendered larger in the output than the scaling factor requires. Consequently, the rest of the image is smaller than the scaling factor requires, because the overall dimensions of the image are always as requested. This distortion will usually be very hard to see.

pnmscalefixed with the **-verbose** option tells you how much distortion there is.

The amount of distortion depends on the size of the input image and how close the scaling factor is to an integral 1/4096th.

If the scaling factor is an exact multiple of 1/4096, there is no distortion. So, for example doubling or halving an image causes no distortion. But reducing it or enlarging it by a third would cause some distortion. To consider an extreme case, scaling a 100,000 row image down to 50,022 rows would create an output image with all of the input squeezed into the top 50,000 rows, and the last row of the input copied into the bottom 22 rows of output.

pnmscalefixed could probably be modified to use 16 bit or better arithmetic without losing anything. The modification would consist of a single constant in the source code. Until there is a demonstrated need for that, though, the Netpbm maintainer wants to keep the safety cushion afforded by the original 12 bit precision.

pnmscalefixed does not have **pnmscale**'s **-nomix** option.

Table Of Contents

NAME

pnmshear - shear a PNM image by a specified angle

SYNOPSIS

pnmshear

[**-noantialias**]

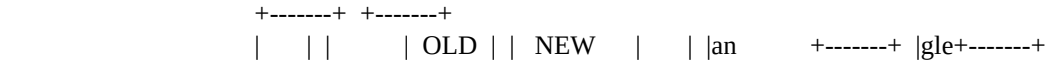
angle

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmshear reads a PNM image as input and shears it by the specified angle and produce a PNM image as output. If the input file is in color, the output will be too, otherwise it will be grayscale. The angle is in degrees (floating point), and measures this:



If the angle is negative, it shears the other way:

The angle should not get too close to 90 or -90, or the resulting anymap will be unreasonably wide.

pnmshear does the shearing by looping over the source pixels and distributing fractions to each of the destination pixels. This has an 'anti-aliasing' effect - it avoids jagged edges and similar artifacts. However, it also means that the original colors or gray levels in the image are modified. If you need to keep precisely the same set of colors, you can use the **-noantialias** flag. This does the shearing by moving pixels without changing their values. If you want anti-aliasing and don't care about the precise colors, but still need a limited *number* of colors, you can run the result through **pnmquant**.

All options can be abbreviated to their shortest unique prefix.

SEE ALSO

pnmrotate(1), **pamflip**(1), **pnmquant**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmsmooth - smooth out an image

SYNOPSIS

pnmsmooth

[**-size** *width height*]

[**-dump**

dumpfile]

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmsmooth smoothes out an image by replacing each pixel with the average of its width X height neighbors. It is implemented as a progam that generates a PGM convolution matrix and then invokes **pnmconvol** with it.

OPTIONS

-size *width height*

Specifies the size of the convolution matrix. Default size is a 3x3 matrix. Width and height sizes must be odd. Maximum size of convolution matrix is limited by the maximum value for a pixel such that (width * height * 2) must not exceed the maximum pixel value.

-dump *dumpfile*

This options makes **pnmsmooth** only generate and save the convolution file. It does not invoke **pnmconvol** and does not produce an output image.

SEE ALSO

pnmconvol(1), **pnm**(1)

Table Of Contents

NAME

pnmsplit - split a multi-image PNM file into multiple single-image files

SYNOPSIS

pnmsplit

[*pnmfile*[*output_file_pattern*]]

[**-padname=***n*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmsplit reads a PNM stream as input. It copies each image in the input into a separate file, in the same format.

pnmfile is the file specification of the input file, or **-** to indicate Standard Input. The default is Standard Input.

output_file_pattern tells how to name the output files. It is the file specification of the output file, except that the first occurence of '%d' in it is replaced by the image sequence number in unpadded ASCII decimal, with the sequence starting at 0. If there is no '%d' in the pattern, **pnmsplit** fails.

The default output file pattern is 'image%d'.

The **-padname** option specifies to how many characters you want the image sequence number in the output file name padded with zeroes. **pnmpslit** adds leading zeroes to the image sequence number to get up to at least that number of characters. This is just the number of characters in the sequence number part of the name. For example, **pnmsplit - outputfile%d.ppm -padname=3** would yield output filenames **outputfile000.ppm**, **outputfile001.ppm**, etc.

The default is no padding (equivalent to **-padname=0**.

The **-padname** option was new in Netpbm 10.23 (July 2004). Before that, there was never any padding.

Note that to do the reverse operation (combining multiple single-image PNM files into a multi-image one), there is no special Netpbm program. Just use **cat**.

SEE ALSO

pnm(1), **cat** man page

Table Of Contents

NAME

pnmstitch - stitch together two panoramic (side-by-side) photographs

SYNOPSIS

pnmstitch [[*left_filespec*] *right_filespec* | *left_filespec right_filespec output_filespec*] [**-width=***width*] [**-height=***height*] [**-xrightpos=***column*] [**-yrightpos=***row*] [**-stitcher=**{**RotateSliver**, **BiLinearSliver**,**LinearSliver**}] [**-filter=**{**LineAtATime**,**HorizontalCrop**}] [**-output=***output_filespec*] [**-verbose**]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmstitch stitches together two panoramic photographs. This means if you have photographs of the left and right side of something that is too big for a single camera frame, **pnmstitch** can join them into one wide picture.

pnmstitch works only on side-by-side images, not top and bottom (though you could certainly use **pnmrotate** in combination with **pnmstitch** to achieve this). It stitches together two images, but you can use it repeatedly to stitch together as many as you need to.

Your photographs must overlap in order for **pnmstitch** to work, and the overlap should be substantial. **pnmstitch** shifts and stretches the right hand image to match it up the left hand image. You probably want to crop the result with **pamcut** to make a nice rectangular image.

If you're just trying to join (concatenate) two images at their edges, use **pnmcat**.

The *left_filespec* and *right_filespec* arguments are the specifications (names) of the PNM files containing the left hand and right hand images. If you specify only *right_filespec*, the left hand image comes from Standard Input. If you specify neither, both images come from Standard Input as a multi-image file containing first the left and then the right image.

output_filespec is the specification (name) of the output PNM file. The **-output** option also specifies the output file. You cannot specify both the argument and the option. If you specify neither, the output goes to Standard Output.

OPTIONS

-width=*width*

-height=*height*

-xrightpos=*column*

-yrightpos=*row*

These are constraints on where **pnmstitch** stitches the images together. For the **LinearSliver** method, *column* and *row* tell what location in the right hand image matches up to the top right corner of the left hand image.

-stitcher={**RotateSliver**,**BiLinearSliver**,
LinearSliver} The default is **RotateSliver**.

-filter={**LineAtATime**,**HorizontalCrop**}

No details available.

-output=*output_filespec*

Name of output file. If you don't specify this option, the output image goes to Standard Output.

-verbose

This option causes **pnmstitch** to issue messages to Standard Error about the stitching process.

SEE ALSO

pamcut(1), **pnmcat**(1), **pnmrotate**(1), **pnm**(1),

HISTORY

This program was added to Netpbm in Release 10.7 (August 2002).

Table Of Contents

NAME

pnmtile - replicate an image to fill a specified region

SYNOPSIS

pnmtile *width height* [*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtile reads a PNM image as input. Replicates it to fill an area of the specified dimensions and produces an image in the same format as output.

You can do pretty much the reverse with **pamdice**.

You can explicitly concatenate an image to itself (or anything else) with **pnmcat**.

If you're trying to tile multiple images into a superimage (such as a thumbnail sheet), see **pnmindex**.

SEE ALSO

pnmcat(1), **pamdice**(1), **pampop9**(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmtoddif - Convert a PNM image to DDIF format

SYNOPSIS

pnmtoddif [**-resolution** *x y*] [*pnmfile* [*ddiffile*]]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtoddif takes a PNM image and converts it into a DDIF image file.

pnmtoddif writes PBM format (bitmap) data as a 1 bit DDIF, PGM format data (grayscale) as an 8 bit grayscale DDIF, and PPM format (color) data as an 8,8,8 bit color DDIF. **pnmtoddif** writes any DDIF image file uncompressed. The data plane organization is interleaved by pixel.

In addition to the number of pixels in the width and height dimension, DDIF images also carry information about the size that the image should have, that is, the physical space that a pixel occupies. Netpbm images do not carry this information, hence you have to supply it externally. The default of 78 dpi has the beneficial property of not causing a resize on most Digital Equipment Corporation color monitors.

OPTIONS

resolution *x y*

The horizontal and vertical resolution of the output image in dots per inch. Default is 78 dpi.

ARGUMENTS

pnmfile The filename for the image file in pnm format. Default is to read from Standard Input.

ddiffile The filename for the image file to be created in DDIF format. By default, **pnmtoddif** writes to Standard Output.

AUTHOR

Burkhard Neidecker-Lutz, Digital Equipment Corporation, CEC Karlsruhe *neideck@nestvx.enet.dec.com*

Table Of Contents

NAME

pnmtofiasco - Convert PNM file to FIASCO compressed file

SYNOPSIS

pnmtofiasco [*option*]... [*filename*]...

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtofiasco compresses the named pbm, pgm, or ppm image files, or Standard Input if no file is named, and produces a FIASCO file on Standard Output.

OPTIONS

All option names may be abbreviated; for example, --optimize may be written --optim or --opt. For most options a one letter short option is provided. Mandatory or optional arguments to long options are mandatory or optional for short options, too. Both short and long options are case sensitive.

The basic options are:

-i *name*, **--input-name**=*name*

Compress the named images, not Standard Input. If *name* is **-**, read Standard Input. *name* has to be either an image filename or a template of the form:

prefix[start-end{+,-}step]suffix

Templates are useful when compressing video streams: e.g., if you specify the template **img0[12-01-2].pgm**, then **pnmtofiasco** compresses the images img012.pgm, img010.pgm, ..., img002.pgm.

If *name* is a relative path, **pnmtofiasco** searches for the image files in the current directory and in the (colon-separated) list of directories given by the environment variable **FIASCO_IMAGES**.

-o *output-file*, **--output-name**=*name*

Write FIASCO output to the named file, not to Standard Output.

If *name* is a relative path and the environment variable **FIASCO_DATA** is a (colon-separated) list of directories, then **pnmtofiasco** writes the output file to the first (writable) directory of this list. Otherwise, **pnmtofiasco** write it to the current directory.

-q *N*, **--quality**=*N*

Set quality of compression to *N*. Quality is 1 (worst) to 100 (best); default is 20.

-v, **--version**

Print **pnmtofiasco** version number, then exit.

-V *N*, **--verbose** *N*

Set level of verbosity to *N*. Level is 0 (no output at all), 1 (show progress meter), or 2 (show detailed compression statistics); default is 1.

-B *N*, **--progress-meter** *N*
Set type of progress-meter to *N*. The following types are available; default is 1:

0 no progress meter

1 RPM style progress bar using 50 hash marks

2 percentage meter

-f *name*, **--config=***name*
Load parameter file *name* to initialize the options of **pnmtofiasco**. See file **system.fiascorc** for an example of the syntax. Options of **pnmtofiasco** are set by any of the following methods (in the specified order):

- Global resource file **/etc/system.fiascorc**
- $HOME/.fiascorc
- command line
- --config=*name*

-h, **--info**
Print brief help, then exit.

-H, **--help**
Print detailed help, then exit.

The options for advanced users are:

-b *name*, **--basis-name=***name*
Preload compression basis *name* into FIASCO. The basis *name* provides the initial compression dictionary. Either use one of the files 'small.fco', 'medium.fco', or 'large.fco' that come with **pnmtofiasco** or create a new ASCII basis file.

-z *N*, **--optimize=***N*
Set optimization level to *N*. Level is 0 (fastest) to 3 (slowest); default is 1. Be warned, the encoding time dramatically increased when *N*=**2** or *N*=**3** while the compression performance only slightly improves.

-P, **--prediction**
Use additional predictive coding. If this optimization is enabled then the image is compressed in two steps. In the first step, a coarse approximation of the image is computed using large unichrome blocks. Finally, the delta image is computed and the prediction error is approximated using the standard FIASCO algorithm.

-D *N*, **--dictionary-size=***N*
Set size of dictionary that is used when coding the luminance band to *N*; default is 10000, i.e., the dictionary is not restricted.

-C *N*, **--chroma-dictionary=***N*
Set size of dictionary that is used when coding chroma bands to *N*; default is 40.

-Q *N*, **--chroma-qfactor=***N*
Reduce the quality of chroma band compression *N*-times with respect to the user defined quality *q* of the luminance band compression (**--quality**=*q*); default is 2.

-t *N*, **--tiling-exponent=***N*
Subdivide the image into 2^*N* tiles prior coding; default is 4, i.e. the image is subdivided into 16 tiles. The processing order of the individual tiles is defined by the option **--tiling-method=***name*.

-T *name*, **--tiling-method=***name*
Order the individual image tiles (the image is subdivided into; see option **--tiling-exponent=***N*) by method *name*; default is **desc-variance**.

desc-variance
Tiles with small variances are processed first.

asc-variance
Tiles with large variances are processed first.

desc-spiral
Tiles are process in spiral order starting in the middle.

asc-spiral
Tiles are process in spiral order starting at the border.

--rpf-mantissa=*N*
Use *N* mantissa bits for quantized coefficients.

--dc-rpf-mantissa=*N*
Use *N* mantissa bits for quantized DC coefficients.

--rpf-range=*N*
Coefficients outside the quantization interval [-*N*,+*N*] are set to zero.

--dc-rpf-range=*N*
DC coefficients outside the quantization interval [-*N*,+*N*] are set to zero.

Additional options for video compression are

-s *N*, **--smooth=***N*
Smooth decompressed reference frames along the partitioning borders by the given amount *N*. *N* is 0 (no smoothing) to 100; default is 70. This factor is stored in the FIASCO file.

-m *N*, **--min-level=***N*
Start prediction (motion compensated prediction or additional prediction) on block level *N*; default is level 6. I.e., motion compensation is applied to all image blocks of at least 8x8 pixels (binary tree level *N*=6), 16x8 (*N*=7), 16x16 (*N*=8), etc.

-M *N*, **--max-level=***N*
Stop prediction (motion compensated prediction or additional prediction) on block level *N*; default is level 10. I.e., motion compensation is applied to all image blocks of at most 16x16 pixels (*N*=8), 32x16 (*N*=9), 32x32 (*N*=10), etc.

-2, **--half-pixel**
Use half pixel precise motion compensation.

-F *N*, **--fps=***N*
Set number of frames per second to *N*. This value is stored in the FIASCO output file and is used in the decoder **fiascotopnm**(1)**to**control**the**framerate.

-p *type*, **--pattern=***type*
Defines the type of inter frame compression which should be applied to individual frames of a video stream. *type* is a sequence of characters; default is 'IPPPPPPPPP'. Element **N** defines the type of predicting which should be used for frame **N**; the frame type pattern is periodically extended. Valid characters are:

I intra frame, i.e., no motion compensated prediction is used at all.

P predicted frame, i.e., a previously encoded frame is used for prediction (forward prediction).

B bidirectional predicted frame, i.e., not only a previously shown frame but also a frame of the future is used for prediction (forward, backward or interpolated prediction).

--cross-B-search
Instead of using exhaustive search the 'Cross-B-Search' algorithm is used to find the best interpolated prediction of B-frames.

--B-as-past-ref
Also use previously encoded B-frames when prediction the current frame. If this option is not set, only I- and P-frames are used to predict the current frame.

EXAMPLES

Compress the still image 'foo.ppm' to the FIASCO file 'foo.wfa' using the default options:

```
pnmtofiasco < foo.ppm >foo.wfa
```

Compress the video frames 'foo0*.ppm' to the FIASCO file 'video.wfa' using half pixel precise motion compensation at a frame rate of 15 frames per second. Intra frame 1 is used to predict P-frame 4, frames 1 and 4 are used to predict B-frames 2 and 3, and so on. Frame 10 is again an intra-frame.

```
pnmtofiasco -2 -p 'IBBPBBPBB' -fps 15 -o video.wfa foo0*.ppm
```

FILES

/etc/system.fiascorc
The systemwide initialization file.

$HOME**/.fiascorc**
The personal initialization file.

ENVIRONMENT

FIASCO_IMAGES
Search path for image files. Default is ’./’.

FIASCO_DATA
Search and save path for FIASCO files. Default is ’./’.

SEE ALSO

fiascotopnm(1), **ppmtojpeg**(1), **pnmtojbig**(1), **ppmtogif**(1), **pnm**(1)

Ullrich Hafner, Juergen Albert, Stefan Frank, and Michael Unger. **Weighted Finite Automata for Video Compression**, IEEE Journal on Selected Areas In Communications, January 1998

Ullrich Hafner. **Low Bit-Rate Image and Video Coding with Weighted Finite Automata**, Ph.D. thesis, Mensch & Buch Verlag, ISBN 3-89820-002-7, October 1999.

AUTHOR

Ullrich Hafner <*hafner@bigfoot.de*>

Table Of Contents

NAME

pnmtofits - convert a PNM image into FITS format

SYNOPSIS

pnmtofits [**-max** *f*] [**-min** *f*] [*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtofits reads a PNM image as input and produces a FITS (Flexible Image Transport System) file as output. The resolution of the output file is either 8 bits/pixel, or 16 bits/pixel, depending on the value of maxval in the input file. If the input file is a portable bitmap or a portable graymap, the output file consists of a single plane image (NAXIS = 2). If instead the input file is a portable pixmap, the output file will consist of a three-plane image (NAXIS = 3, NAXIS3 = 3). You can find a full description of the FITS format in Astronomy & Astrophysics Supplement Series 44 (1981), page 363.

OPTIONS

You can use options **-min** and **-max** to set DATAMAX, DATAMIN, BSCALE and BZERO in the FITS header, but not cause the data to be rescaled.

SEE ALSO

fitstopnm(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmtojbig - PNM to JBIG file converter

SYNOPSIS

pnmtojbig

[*options*] [*input-file* [*output-file*]]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtojbig reads a PBM or PGM image, compresses it, and outputs the image as a JBIG bi-level image entity (BIE) file.

JBIG is a highly effective lossless compression algorithm for bi-level images (one bit per pixel), which is particularly suitable for scanned document pages.

A JBIG encoded image can be stored in several resolutions (progressive mode). These resolution layers can be stored all in one single BIE or they can be stored in several separate BIE files. All resolution layers except the lowest one are stored merely as differences to the next lower resolution layer, because this requires less space than encoding the full image completely every time. Each resolution layer has twice the number of horizontal and vertical pixels than the next lower layer. JBIG files can also store several bits per pixel as separate bitmap planes, and **pnmtojbig** can read a PGM file and transform it into a multi-bitplane BIE.

OPTIONS

-q Encode the image in one single resolution layer (sequential mode). This is usually the most efficient compression method. By default, the number of resolution layers is chosen automatically such that the lowest layer image is not larger than 640×480 pixels.

-x *number*

Specify the maximal horizontal size of the lowest resolution layer. The default is 640 pixels.

-y *number*

Specify the maximal vertical size of the lowest resolution layer. The default is 480 pixels.

-l *number*

Select the lowest resolution layer that will be written to the BIE. It is possible to store the various resolution layers of a JBIG image in progressive mode into different BIEs. Options **-l** and **-h** allow you to select the resolution-layer interval that will appear in the created BIE. The lowest resolution layer has number 0 and this is also the default value. By default, **pnmtojbig** writes all layers.

-h *number*

Select the highest resolution layer that will be written to the BIE. By default, **pnmtojbig** writes all layers. See also option **-l**.

-b Use binary values instead of Gray code words in order to encode pixel values in multiple bitplanes. This option has only an effect if the input is a PGM file and if more than one bitplane is produced. Note that the decoder has to make the same selection but cannot determine from the BIE, whether Gray or binary code words were used by the encoder.

-d *number*

Specify the total number of differential resolution layers into which the input image will be split in addition to the lowest layer. Each additional layer reduces the size of layer 0 by 50 %. This option overrides options **-x** and **-y**, which are usually a more comfortable way of selecting the number of resolution layers.

-s *number*

The JBIG algorithm splits each image into a number of horizontal stripes. This option specifies that each stripe shall have *number* lines in layer 0. The default value is selected so that approximately 35 stripes will be used for the whole image.

-m *number*

Select the maximum horizontal offset of the adaptive template pixel. The JBIG encoder uses a number of neighbour pixels in order to get statistical a priori knowledge of the probability, whether the next pixel will be black or white. One single pixel out of this template of context neighbor pixels can be moved around. Especially for dithered images it can be a significant advantage to have one neighbor pixel which has a distance large enough to cover the period of a dither function. By default, the adaptive template pixel can be moved up to 8 pixels away. This encoder supports up to 23 pixels, however as decoders are only required to support at least a distance of 16 pixels by the standard, no higher value than 16 for *number* is recommended in order to maintain interoperability with other JBIG implementations. The maximal vertical offset of the adaptive template pixel is always zero.

-t *number*

Encode only the specified number of most significant bit planes. This option allows to reduce the depth of an input PGM file if not all bits per pixel are needed in the output.

-o *number*

JBIG separates an image into several horizontal stripes, resolution layers and planes, were each plane contains one bit per pixel. One single stripe in one plane and layer is encoded as a data unit called stripe data entity (SDE) inside the BIE. There are 12 different possible orders in which the SDEs can be stored inside the BIE and *number* selects which one shall be used. The order of the SDEs is only relevant for applications that want to decode a JBIG file which has not yet completely arrived from e.g. a slow network connection. For instance some applications prefer that the outermost of the three loops (stripes, layers, planes) is over all layers so that all data of the lowest resolution layer is transmitted first.

The following values for *number* select these loop arrangements for writing the SDEs (outermost loop first):

0 planes, layers, stripes

2 layers, planes, stripes

3 layers, stripes, planes

4 stripes, planes, layers

5 planes, stripes, layers

6 stripes, layers, planes

All loops count starting with zero, however by adding 8 to the above order code, the layer loop can be reversed so that it counts down to zero and then higher resolution layers will be stored before lower layers. Default order is 3 which writes at first all planes of the first stripe and then completes layer 0 before continuing with the next layer and so on.

-p *number*
This option allows you to activate or deactivate various optional algorithms defined in the JBIG standard. Just add the numbers of the following options which you want to activate in order to get the *number* value:

4	deterministic prediction (DPON)
8	typical prediction (TPBON)
16	diff. layer typical prediction (TPDON)
64	layer 0 two-line template (LRLTWO)

Except for special applications (like communication with JBIG subset implementations) and for debugging purposes you will normally not want to change anything here. The default is 28, which provides the best compression result.

-c The adaptive template pixel movement is determined as suggested in annex C of the standard. By default the template change takes place directly in the next line which is most effective. However a few conformance test examples in the standard require the adaptive template change to be delayed until the first line of the next stripe. This option selects this special behavior, which is normally not required except in order to pass some conformance test suite.

-v After **pnmtojbig** creates the BIE, it lists a few technical details of the created file (verbose mode).

FORMATS

Most of the format **pnmtojbig** creates is defined by the JBIG standard.

The standard, however, does not specify which values in the BIE mean white and which mean black. It contains a recommendation that for a single plane image zero mean background and one mean foreground, but the Netpbm formats have no concept of foreground and background. And the standard says nothing about values for multiple plane BIEs.

pnmtojbig follows Markus Kuhn's implementation of the standard in the **pbmtojbg** program that comes with his JBIG library: If the BIE is a single plane BIE, zero means white and one means black. If it is a multiple plane BIE, zero means black and the maximal value is white.

STANDARDS

This program implements the JBIG image coding algorithm as specified in ISO/IEC 11544:1993 and ITU-T T.82(1993).

AUTHOR

pnmtojbig is based on the JBIG library by Markus Kuhn, part of his **JBIG-KIT** package. The **pbmtojbg** program is part of the *JBIG-KIT* package. The most recent version of that library and tools set is

freely available on the Internet from anonymous ftp server ftp.informatik.uni-erlangen.de in directory pub/doc/ISO/JBIG/.

pnmtojbig is part of the Netpbm package of graphics tools.

SEE ALSO

pnm(1), **jbigtopnm**(1)

LICENSE

If you use **pnmtojbig**, you are using various patents, particularly on its arithmetic encoding method, and in all probability you do not have a license from the patent owners to do so.

Table Of Contents

NAME

pnmtojpeg - convert PNM image to a JFIF ('JPEG') image

SYNOPSIS

pnmtojpeg [**-exif=***filespec*] [**-quality=***n*] [{**-grayscale**|**-greyscale**}] [**-density=***n***x***n*[**dpi**,**dpcm**]] [**-optimize**|**-optimise**] [**-rgb**] [**-progressive**] [**-comment=***text*] [**-dct=**{**int**|**fast**|**float**}] [**-arithmetic**] [**-restart=***n*] [**-smooth=***n*] [**-maxmemory=***n*] [**-verbose**] [**-baseline**] [**-qtables=***filespec*] [**-qslots=n[,...]**] [**-sample=***H***x***V*[,...]] [**-scans=***filespec*] [**-tracelevel=***N*]

filename

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtojpeg converts the named PBM, PGM, or PPM image file, or the standard input if no file is named, to a JFIF file on the standard output.

pnmtojpeg uses the Independent JPEG Group's JPEG library to create the output file. See **http://www.ijg.org** for information on the library.

'JFIF' is the correct name for the image format commonly known as 'JPEG.' Strictly speaking, JPEG is a method of compression. The image format using JPEG compression that is by far the most common is JFIF. There is also a subformat of TIFF that uses JPEG compression.

EXIF is an image format that is a subformat of JFIF (to wit, a JFIF file that contains an EXIF header as an APP1 marker). **pnmtojpeg** creates an EXIF image when you specify the **-exif** option.

OPTIONS

The basic options are:

-exif=*filespec*

This option specifies that the output image is to be EXIF (a subformat of JFIF), i.e. it will have an EXIF header as a JFIF APP1 marker. The contents of that marker are the contents of the specified file. The special value **-** means to read the EXIF header contents from standard input. It is invalid to specify standard input for both the EXIF header and the input image.

The EXIF file starts with a two byte field which is the length of the file, including the length field, in pure binary, most significant byte first. The special value of zero for the length field means there is to be no EXIF header, i.e. the same as no **-exif** option. This is useful for when you convert a file from JFIF to PNM using **jpegtopnm**, then transform it, then convert it back to JFIF with **pnmtojpeg**, and you don't know whether or not it includes an EXIF header. **jpegtopnm** creates an EXIF file containing nothing but two bytes of zero when the input JFIF file has no EXIF header. Thus, you can transfer any EXIF header from the input JFIF to the output JFIF without worrying about whether an EXIF header actually exists.

The contents of the EXIF file after the length field are the exact byte for byte contents of the APP1 marker, not counting the length field, that constitutes the EXIF header.

-quality=*n*

Scale quantization tables to adjust image quality. *n* is 0 (worst) to 100 (best); default is 75. Below about 25 can produce images some interpreters won't be able to interpret. See below

for more info.

-grayscale

-greyscale

-rgb These options determine the color space used in the JFIF output. **-grayscale** (or **-greyscale**) means to create a gray scale JFIF, converting from color PPM input if necessary. **-rgb** means to create an RGB JFIF, and the program fails if the input is not PPM.

If you specify neither, The output file is in YCbCr format if the input is PPM, and grayscale format if the input is PBM or PGM.

YCbCr format (a color is represented by an intensity value and two chrominance values) usually compresses much better than RGB (a color is represented by one red, one green, and one blue value). RGB is rare. But you may be able to convert between JFIF and PPM faster with RGB, since it's the same color space PPM uses.

The **testimg.ppm** file that comes with Netpbm is 2.3 times larger with the **-rgb** option that with the YCbCr default, and in one experiment **pnmtojpeg** took 16% more CPU time to convert it. The extra CPU time probably indicates that processing of all the extra compressed data consumed all the CPU time saved by not having to convert the RGB inputs to YCbCr.

Grayscale format takes up a lot less space and takes less time to create and process than the color formats, even if the image contains nothing but black, white, and gray.

The **-rgb** option was added in Netpbm 10.11 in October 2002.

-density=*density*

This option determines the density (aka resolution) information recorded in the JFIF output image. It does not affect the raster in any way; it just tells whoever reads the JFIF how to interpret the raster.

The density value takes the form *x***x***y* followed by an optional unit specifier of **dpi** or **dpcm**. Examples: **1x1**, **3x2**, **300x300dpi**, **100x200dpcm**. The first number is the horizontal density; the 2nd number is the vertical density. Each may be any integer from 1 to 65535. The unit specifier is **dpi** for dots per inch or **dpcm** for dots per centimeter. If you don't specify the units, the density information goes into the JFIF explicitly stating "density unspecified" (also interpreted as "unknown"). This may seem pointless, but note that even without specifying the units, the density numbers tell the aspect ratio of the pixels. E.g. **1x1** tells you the pixels are square. **3x2** tells you the pixels are horizontal rectangles.

Note that if you specify different horizontal and vertical densities, the resulting JFIF image is *not* a true representation of the input PNM image, because **pnmtojpeg** converts the raster pixel-for-pixel and the pixels of a PNM image are defined to be square. Thus, if you start with a square PNM image and specify **-density=3x2**, the resulting JFIF image is a vertically squashed version of the original. However, it is common to use an input image which is a slight variation on PNM rather than true PNM such that the pixels are not square. In that case, the appropriate -density option yields a faithful reproduction of the input pseudo-PNM image.

The default is 1x1 in unspecified units.

Before Netpbm 10.15 (April 2003), this option did not exist and the **pnmtojpeg** always created a JFIF with a density of 1x1 in unspecified units.

-optimize

Perform optimization of entropy encoding parameters. Without this, **pnmtojpeg** uses default encoding parameters. **-optimize** usually makes the JFIF file a little smaller, but **pnmtojpeg** runs somewhat slower and needs much more memory. Image quality and speed of

decompression are unaffected by **-optimize**.

-progressive
Create a progressive JPEG file (see below).

-comment=*text*
Include a comment marker in the JFIF output, with comment text *text*.

Without this option, there are no comment markers in the output.

The **-quality** option lets you trade off compressed file size against quality of the reconstructed image: the higher the quality setting, the larger the JFIF file, and the closer the output image will be to the original input. Normally you want to use the lowest quality setting (smallest file) that decompresses into something visually indistinguishable from the original image. For this purpose the quality setting should be between 50 and 95 for reasonable results; the default of 75 is often about right. If you see defects at **-quality=75**, then go up 5 or 10 counts at a time until you are happy with the output image. (The optimal setting will vary from one image to another.)

-quality=100 generates a quantization table of all 1's, minimizing loss in the quantization step (but there is still information loss in subsampling, as well as roundoff error). This setting is mainly of interest for experimental purposes. Quality values above about 95 are *not* recommended for normal use; the compressed file size goes up dramatically for hardly any gain in output image quality.

In the other direction, quality values below 50 will produce very small files of low image quality. Settings around 5 to 10 might be useful in preparing an index of a large image library, for example. Try **-quality=2** (or so) for some amusing Cubist effects. (Note: quality values below about 25 generate 2-byte quantization tables, which are considered optional in the JFIF standard. **pnmtojpeg** emits a warning message when you give such a quality value, because some other JFIF programs may be unable to decode the resulting file. Use **-baseline** if you need to ensure compatibility at low quality values.)

The **-progressive** option creates a 'progressive JPEG' file. In this type of JFIF file, the data is stored in multiple scans of increasing quality. If the file is being transmitted over a slow communications link, the decoder can use the first scan to display a low-quality image very quickly, and can then improve the display with each subsequent scan. The final image is exactly equivalent to a standard JFIF file of the same quality setting, and the total file size is about the same -- often a little smaller.

Caution: progressive JPEG is not yet widely implemented, so many decoders will be unable to view a progressive JPEG file at all.

If you're trying to control the quality/file size tradeoff, you might consider the JPEG2000 format instead. See **pamtojpeg2k**(1).

Options for advanced users:

-dct=int
Use integer DCT method (default).

-dct=fast
Use fast integer DCT (less accurate).

-dct=float
Use floating-point DCT method. The float method is very slightly more accurate than the int method, but is much slower unless your machine has very fast floating-point hardware. Also note that results of the floating-point method may vary slightly across machines, while the integer methods should give the same results everywhere. The fast integer method is much less accurate than the other two.

-arithmetic

Use arithmetic coding. Default is Huffman encoding. Arithmetic coding tends to get you a smaller result.

You may need patent licenses to use this option. According to the JPEG FAQ , This method is covered by patents owned by IBM, AT&T, and Mitsubishi.

The author of the FAQ recommends against using arithmetic coding (and therefore this option) because the space savings is not great enough to justify the legal hassles.

-restart=*n*

Emit a JPEG restart marker every *n* MCU rows, or every *n* MCU blocks if you append **B** to the number. **-restart 0** (the default) means no restart markers.

-smooth=*n*

Smooth the input image to eliminate dithering noise. *n*, ranging from 1 to 100, indicates the strength of smoothing. 0 (the default) means no smoothing.

-maxmemory=*n*

Set a limit for amount of memory to use in processing large images. Value is in thousands of bytes, or millions of bytes if you append **M** to the number. For example, **-max=4m** selects 4,000,000 bytes. If **pnmtojpeg** needs more space, it will use temporary files.

-verbose

Print to the Standard Error file messages about the conversion process. This can be helpful in debugging problems.

The **-restart** option tells **pnmtojpeg** to insert extra markers that allow a JPEG decoder to resynchronize after a transmission error. Without restart markers, any damage to a compressed file will usually ruin the image from the point of the error to the end of the image; with restart markers, the damage is usually confined to the portion of the image up to the next restart marker. Of course, the restart markers occupy extra space. We recommend **-restart=1** for images that will be transmitted across unreliable networks such as Usenet.

The **-smooth** option filters the input to eliminate fine-scale noise. This is often useful when converting dithered images to JFIF: a moderate smoothing factor of 10 to 50 gets rid of dithering patterns in the input file, resulting in a smaller JFIF file and a better-looking image. Too large a smoothing factor will visibly blur the image, however.

Options for wizards:

-baseline

Force baseline-compatible quantization tables to be generated. This clamps quantization values to 8 bits even at low quality settings. (This switch is poorly named, since it does not ensure that the output is actually baseline JPEG. For example, you can use **-baseline** and **-progressive** together.)

-qtables=*filespec*

Use the quantization tables given in the specified text file.

-qslots=n[,...]

Select which quantization table to use for each color component.

-sample=*H***x***V*[,...]
Set JPEG sampling factors for each color component.

-scans=*filespec*
Use the scan script given in the specified text file. See below for information on scan scripts.

-tracelevel=*N*
This sets the level of debug tracing the program outputs as it runs. 0 means none, and is the default. This level primarily controls tracing of the JPEG library, and you can get some pretty interesting information about the compression process.

The 'wizard' options are intended for experimentation with JPEG. If you don't know what you are doing, **don't use them**. These switches are documented further in the file wizard.doc that comes with the Independent JPEG Group's JPEG library.

EXAMPLES

This example compresses the PPM file foo.ppm with a quality factor of 60 and saves the output as foo.jpg:

```
pnmtojpeg -quality=60 foo.ppm > foo.jpg
```

Here's a more typical example. It converts from BMP to JFIF:

```
cat foo.bmp | bmptoppm | pnmtojpeg > foo.jpg
```

JPEG Loss

When you compress with JPEG, you lose information -- i.e. the resulting image has somewhat lower quality than the original. This is a characteristic of JPEG itself, not any particular program. So if you do the usual Netpbm thing and convert from JFIF to PNM, manipulate, then convert back to JFIF, you will lose quality. The more you do it, the more you lose. Drawings (charts, cartoons, line drawings, and such with few colors and sharp edges) suffer the most.

To avoid this, you can use a compressed image format other than JPEG. PNG and JPEG2000 are good choices, and Netpbm contains converters for those.

If you need to use JFIF on a drawing, you should experiment with **pnmtojpeg**'s **-quality** and **-smooth** options to get a satisfactory conversion. **-smooth 10** or so is often helpful.

Because of the loss, you should do all the manipulation you have to do on the image in some other format and convert to JFIF as the last step. And if you can keep a copy in the original format, so much the better.

The **-optimize** option to **pnmtojpeg** is worth using when you are making a 'final' version for posting or archiving. It's also a win when you are using low quality settings to make very small JFIF files; the percentage improvement is often a lot more than it is on larger files. (At present, **-optimize** mode is automatically in effect when you generate a progressive JPEG file).

You can do flipping and rotating transformations losslessly with the program **jpegtran**, which is packaged with the Independent Jpeg Group's JPEG library. **jpegtran** exercises its intimate knowledge of the way JPEG works to do the transformation without ever actually decompressing the image.

Another program, **cjpeg**, is similar. **cjpeg** is
maintained by the Independent JPEG Group and packaged with the JPEG library which **pnmtojpeg** uses for all its JPEG work. Because of that, you may expect it to exploit more current JPEG features. Also, since you have to have the library to run **pnmtojpeg**, but not vice versa, **cjpeg** may be more commonly available.

On the other hand, **cjpeg** does not use the NetPBM libraries to process its input, as all the NetPBM

tools such as **pnmtojpeg** do. This means it is less likely to be consistent with all the other programs that deal with the NetPBM formats. Also, the command syntax of **pnmtojpeg** is consistent with that of the other Netpbm tools, unlike **cjpeg**.

SCAN SCRIPTS

Use the **-scan** option to specify a scan script. Or use the **-progressive** option to specify a particular built-in scan script.

Just what a scan script is, and the basic format of the scan script file, is covered in the **wizard.doc** file that comes with the Independent JPEG Group's JPEG library. Scan scripts are same for **pnmtojpeg** as the are for **cjpeg**.

This section contains additional information that isn't, but probably should be, in that document.

First, there are many restrictions on what is a valid scan script. The JPEG library, and thus **pnmtojpeg**, checks thoroughly for any lack of compliance with these restrictions, but does little to tell you how the script fails to comply. The messages are very general and sometimes untrue.

To start with, the entries for the DC coefficient must come before any entries for the AC coefficients. The DC coefficient is Coefficient 0; all the other coefficients are AC coefficients. So in an entry for the DC coefficient, the two numbers after the colon must be 0 and 0. In an entry for AC coefficients, the first number after the colon must not be 0.

In a DC entry, the color components must be in increasing order. E.g. '0,2,1' before the colon is wrong. So is '0,0,0'.

In an entry for an AC coeffient, you must specify only one color component. I.e. there can be only one number before the colon.

In the first entry for a particular coefficient for a particular color component, the 'Ah' value must be zero, but the Al value can be any valid bit number. In subsequent entries, Ah must be the Al value from the previous entry (for that coefficient for that color component), and the Al value must be one less than the Ah value.

The script must ultimately specify at least some of the DC coefficent for every color component. Otherwise, you get the error message 'Script does not transmit all the data.' You need not specify all of the bits of the DC coefficient, or any of the AC coefficients.

There is a standard option in building the JPEG library to omit scan script capability. If for some reason your library was built with this option, you get the message 'Requested feature was omitted at compile time.'

ENVIRONMENT

JPEGMEM

If this environment variable is set, its value is the default memory limit. The value is specified as described for the **-maxmemory** option. An explicit **-maxmemory** option overrides any **JPEGMEM**.

SEE ALSO

jpegtopnm(1), **pnm**(1), **cjpeg** man page, **djpeg** man page, **jpegtran** man page, **rdjpgcom** man page, **wrjpgcom** man page

Wallace, Gregory K. 'The JPEG Still Picture Compression Standard', Communications of the ACM, April 1991 (vol. 34, no. 4), pp. 30-44.

AUTHOR

pnmtojpeg and this manual were derived in large part from **cjpeg**, by the Independent JPEG Group. The program is otherwise by Bryan Henderson on March 07, 2000.

Table Of Contents

NAME

pnmtopalm - convert a PNM image to a Palm Bitmap

SYNOPSIS

pnmtopalm

[**-verbose**]

[**-depth=***N*]

[**-maxdepth=***N*]

[**-colormap**]

[**-transparent=***color*]

[**-density=***N*]

[**-offset**]

[**-withdummy**] [**-scanline-compression** | **-rle-compression** | **-packbits-compression**]

[*pnmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtopalm reads a PNM image as input, from Standard Input or *pnmfile* and produces a Palm Bitmap as output.

Palm Bitmap files are either grayscale files 1, 2, or 4 bits wide, or color files 8 bits wide, so **pnmtopalm** automatically scales colors to have an appropriate maxval, unless you specify a depth or max depth. Input files must have an appropriate number and set of colors for the selected output constraints.

This often means that you should run the PNM image through **pnmquant** or **pnmremap** before you pass it to **pnmtopalm**. Netpbm comes with several colormap files you can use with **pnmremap** for this purpose. They are *palmgray2.map* (4 shades of gray for a depth of 2), *palmgray4.map* (16 shades of gray for a depth of 4), and *palmcolor8.map* (232 colors in default Palm colormap). In a standard Netpbm installation, these are in the Netpbm data directory, and you can find the Netpbm data directory with a **netpbm-config --datadir** shell command.

Example:

```
pnmremap myimage.ppm          -mapfile=$(netpbm-config --datadir)/palmgray2.map   | pnmtopalm -depth=2 >myimage.pal
```

Palm Bitmap Version

pnmtopalm generates a Version 0, 1, 2, or 3 Palm Bitmap. It generates the oldest (lowest) version it can for the given image and the options you specify.

- If you specify a density (**-density** option) higher than 'low,' the version is at least 3.

- If you specify transparency (**-transparent** option) or any compression, the version is at least 2.

- If you specify a custom colormap (**-colormap** option), the verison is at least 1.

- If the image has more than one bit per pixel, the version is at least 1. The image has more than one bit per pixel if you specify it with **-depth** or if you let it default and the image has more than two colors (or shades of gray).

All releases of Palm OS can read a Version 0 bitmap. Palm OS 3.0 and later can read a Version 1 bitmap. Palm OS 3.5 and later can read a Version 2 bitmap. To read a Version 3 bitmap, you need Palm OS Garnet or a handheld running the High Density Display Feature Set.

OPTIONS

-verbose

Display the format of the output file.

-depth=*N*

Produce a file of depth *N*, where *N* must be either 1, 2, 4, 8, or 16. Because the default Palm 8-bit colormap is not grayscale, if the input is a grayscale or monochrome image, the output will never be more than 4 bits deep, regardless of the specified depth. Note that 8-bit color works only in PalmOS 3.5 (and higher), and 16-bit direct color works only in PalmOS 4.0 (and higher). However, the 16-bit direct color format is also compatible with the various PalmOS 3.x versions used in the Handspring Visor, so these images may also work in that device.

-maxdepth=*N*

Produce a file of minimal depth, but in any case less than *N* bits wide. If you specify 16-bit, the output will always be 16-bit direct color.

-offset Set the **nextDepthOffset** field in the palm file header to indicate the end of the file (and pad the end of the file to 4 bytes, since **nextDepthOffset** can point only to 4 byte boundaries).

A palm image file can contain multiple renditions of the same image, with different color depths, so a viewer can choose one appropriate for the display. The **nextDepthOffset** field tells where in the stream the next rendition begins.

pnmtopalm creates a file that contains only one image, but you can separately concatenate multiple one-image files to create a multi-image file. If you do that, you'll need to use **-offset** so that the resulting concatenation is a correct stream.

By default (if you don't specify **-offset**), **pnmtopalm** generates a **nextDepthOffset** field that says there is no following image (and does not add any padding after the image).

Version 3 Palm Bitmaps actually have a **nextBitmapOffset** field instead of the **nextDepthOffset**. The foregoing applies to whichever is relevant.

The **-offset** option was new in Netpbm 10.26 (January 2005). Before that, **pnmtopalm** always set the **nextDepthOffset** field to 'none.'

Before Netpbm 10.27 (March 2005), you cannot use **-offset** if you create a compressed raster (because **pnmtopalm** isn't smart enough to be able to know the size of the image at the time it writes the header). You also cannot use it with 16 bit color depth or with the **-colormap**

option, for much the same reason.

-withdummy

This option tells **pnmtopalm** to put in the stream, after after the image, a dummy image header to introduce subsequent high density images.

This dummy image header is a special sequence specified in Palm Bitmap specifications. It looks to an older Palm Bitmap interpreter like an invalid image header, so such an intepreter will stop reading the stream there. But a new Palm Bitmap interpreter recognizes it for what it is (just something to choke an old interpreter) and skips over it. Presumably, you will add to the stream after this high density images which would confuse an older interpreter.

If you specify **-withdummy**, you must also specify **-offset**, since it doesn't make any sense otherwise.

-withdummy was new in Netpbm 10.27 (March 2005).

-colormap

Build a custom colormap and include it in the output file. This is not recommended by Palm, for efficiency reasons. Otherwise, **pnmtopalm** uses the default Palm colormap for color output.

-transparent=*color*

Marks *one* particular color as fully transparent. The format to specify the color is either (when for example orange) '1.0,0.5,0.0', where the values are floats between zero and one, or with the syntax '#RGB', '#RRGGBB' or '#RRRRGGGGBBBB' where R, G and B are hexadecimal numbers. Transparency works only on Palm OS 3.5 and higher.

-scanline-compression

Specifies that the output Palm bitmap will use the Palm scanline compression scheme. Scanline compression works only in Palm OS 2.0 and higher.

-rle-compression

Specifies that the output Palm bitmap will use the Palm RLE compression scheme. RLE compression works only with Palm OS 3.5 and higher.

-packbits-compression

Specifies that the output Palm bitmap will use the Palm packbits compression scheme. Packbits compression works only with Palm OS 4.0 and higher.

This option was new in Netpbm 10.27 (March 2005).

-density=*N*

This specifies the Palm Bitmap density. The density is a number that is proportional to the resolution the image should have when displayed. The proportionality factor is up to whatever is doing the displaying, but it's helpful to think of these numbers as being pixels per inch. The allowable values are:

- 72
- 108
- 144
- 216

- 288

  This option was new in Netpbm 10.27 (March 2005). Earlier Netpbm could not generate Version 3 Palm Bitmaps, so there was no such thing as density.

SEE ALSO

palmtopnm(1), **pnmquant**(1), **pnmremap**(1), **pnm**(1), **PalmOS** Reference (1), PalmOS Companion .

NOTES

Palm Bitmaps may contains multiple renditions of the same bitmap, in different depths. To construct an N-multiple-rendition Palm Bitmap with **pnmtopalm**, first construct renditions 1 through N-1 using the **-offset** option, then construct the Nth image without the **-offset** option. Then concatenate the individual renditions together in a single file using **cat**.

If you will include both high density and low density renditions, put the high density images last and when you create the last of the low density images, use the **-withdummy** option.

LIMITATIONS

You cannot generate an alpha mask if the Palm pixmap has a transparent color. However, you can still do this with **ppmcolormask** with a Netpbm pipe similar to:

palmtopnm pixmap.palm | ppmcolormask ‘palmtopnm -transparent pixmap.palm‘

AUTHORS

This program was originally written as ppmtoTbmp.c, by Ian Goldberg and George Caswell. It was completely re-written by Bill Janssen to add color, compression, and transparency function.

Table Of Contents

NAME

pnmtopclxl - convert a PNM image to an HP LaserJet PCL XL printer stream

SYNOPSIS

pnmtopclxl [**-o** *outfile*] [**-dpi** *N*] [**-xoffs** *N*] [**-yoffs** *N*] [**-center**] [**-duplex** {**vertical**|**horizontal**}] [**-format** *paperformat*] [**-feeder** *N*] [**-copies** *N*] [**-colorok**] *pnmfile1 pnmfile2 ...*

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtopclxl reads one or more PNM input streams, each containing one or more PNM images, and generates a sequence of output pages in the HP PCL-XL (formerly named PCL 6) printer control language. You can send this stream to a PCL-XL printer to print the images.

If the input is PPM, the output is a color printer stream (the PCL color space is RGB). Otherwise, the output is grayscale (the PCL color space is grayscale). If you want a grayscale output from a color input, run your input through **ppmtopgm**(1).**See**the **-colorok** option for more information about choosing between color and grayscale.

OPTIONS

-o *outfile*

This option specifies the name of the PCL-XL output file. If you don't specify this, the output goes to Standard Output. All of the pages go to one file, concatenated in the same order as the input images.

-dpi *N* This option selects the resolution of the image (not the printer!). *N* is the resolution in dots per inch, from 1 to 65535. The default is 300.

-xoffs *N*

This option and **-yoffs** determine the location on the page of the upper left corner of each image. Note that the image may have built in borders tool, which would make the main image within more left and down that what you specify here.

-xoffs and **-yoffs** specify the distance from the left of the page and from the top of the page, respectively, in inches, of the upper left corner of the image. The default for each is zero.

These options are meaningless if you specify **-center**.

-yoffs *N*

See **-xoffs**.

-center This option tells **pnmtopclxl** to center each image on the page. If you don't specify this option, the position of an image on the page is determined by **-xoffs** and **-yoffs** (or their defaults).

-duplex {**vertical**|**horizontal**}

This option causes **pnmtopclxl** to create a printer stream that prints pages on boths sides of the sheet of paper. **vertical** means to print them so that the left edge of both pages is on the same edge of the sheet, while **horizontal** means the more usual duplexing where the top of both pages is on the same edge of the sheet.

-format *paperformat*

This option selects the media (e.g. paper size) that the printer control stream specifies. *paperformat* is one of the following self-explanatory keywords:

- letter
- legal
- a3
- a4
- a5
- a6
- jb4
- jb5
- jb6
- exec
- ledger
- b5envelope
- c5envelope
- com10envelope
- monarchenvelope
- dlenvelope
- jpostcard
- jdoublepostcard

The default is "letter".

-feeder *N*

This options selects the media source (e.g. paper tray) that the printer control stream specifies.

-copies *N*

This option specifies the number of copies that the printer control stream tells the printer to print.

-colorok

This option simply tells **pnmtopclxl** not to warn you if you supply a color input and therefore get color output. By default, **pnmtopclxl** issues a warning any time it produces a color printer stream because it is usually a mistake. It's a mistake because PCL XL is mainly used for laser printers, and laser printers are mainly black and white. If you send a color print stream to a black and white printer, it typically refuses to print anything, and even if it manages to convert it to black and white and print it, it takes 3 times as long to transmit a color stream to the printer than to transmit the grayscale image that gives the same result.

SEE ALSO

ppmtolj(1), **pbmtolj**(1), **ppmtopj**(1), **ppmtopjxl**(1), **thinkjettopbm**(1), **ppm**(1)

HISTORY

pnmtopclxl was added to Netpbm in Release 10.6 (July 2002). It was contributed by *Jochen Karrer*.

NAME

pnmtoplainpnm - replaced by pnmtopnm

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtoplainpnm was obsoleted in Netpbm 10.23 (July 2004) by **pnmtopnm**(1).**Just**use**the**Netpbm-**common**option **-plain**.

pnmtoplainpnm exists today for backward compatibility; all it does is call **pnmtopnm -plain**.

pnmtoplainpnm was new in Netpbm 8.2 (March 2000) as a renaming of **pnmnoraw**, which was new in Pbmplus in November 1989.

Table Of Contents

NAME

pnmtopng - convert a PNM image to PNG

SYNOPSIS

pnmtopng [**-verbose**] [**-downscale**] [**-interlace**] [**-alpha** *file*] [**-transparent** [=]*color*] [**-background** *color*] [**-palette** *palettefile*] [**-gamma** *value*] [**-hist**] [**-chroma** *wx wy rx ry gx gy bx by*] [**-phys** *x y unit*] [**-text** *file*] [**-ztxt** *file*] [**-time** [*yy*]*yy-mm-dd hh*:*mm*:*ss*] [**-nofilter**] [**-sub**] [**-up**] [**-avg**] [**-paeth**] [**-compression** level] [**-force**] [**-version**] [*pnmfile*]

OPTION USAGE

Obsolete options:

[**-filter** type]

You may abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtopng reads a PNM image as input and produces a PNG image as output.

Color values in PNG files are either eight or sixteen bits wide, so *pnmtopng* will automatically scale colors to have a maxval of 255 or 65535. Grayscale files will be produced with bit depths 1, 2, 4, 8 or 16. An extra **pnmdepth** step is not necessary.

OPTIONS

-verbose

Display the format of the output file.

-downscale

Enables scaling of maxvalues of more then 65535 to 16 bit. Since this means loss of image data, the step is not performed by default.

-interlace

Creates an interlaced PNG file (Adam7).

-alpha *file*

This specifies the transparency (alpha channel) of the image. You supply the alpha channel as a standard PGM alpha mask (see the **PGM**(1)**specification.pnmtopngdoes**not necessarily represents the transparency information as an alpha channel in the PNG format. If it can represent the transparency information through a palette, it will do so in order to make a smaller PNG file. **pnmtopng** even sorts the palette so it can omit the opaque colors from the transparency part of the palette and save space for the palette.

-transparent *color*

pnmtopng marks the specified color as transparent in the PNG image.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine . E.g. **red** or **rgb:ff/00/0d**. If the color you specify is not present in the image, **pnmtopng** selects instead the color in the image that is closest to the one you specify. Closeness is measured as a cartesian distance between colors in RGB space. If multiple colors are equidistant, **pnmtopng** chooses one of them arbitrarily.

However, if you prefix your color specification with '=', e.g.

-transparent =red

only the exact color you specify will be transparent. If that color does not appear in the

image, there will be no transparency. **pnmtopng** issues an information message when this is the case.

-background *color*
Causes **pnmtopng** to create a background color chunk in the PNG output which can be used for subsequent alpha channel or transparent color conversions. Specify *color* the same as for **-transparent**.

-palette *palettefile*
This option specifies a palette to use in the PNG. It forces **pnmtopng** to create the paletted (colormapped) variety of PNG -- if that isn't possible, **pnmtopng** fails. If the palette you specify doesn't contain exactly the colors in the image, **pnmtopng** fails. Since **pnmtopng** will automatically generate a paletted PNG, with a correct palette, when appropriate, the only reason you would specify the **-palette** option is if you care in what order the colors appear in the palette. The PNG palette has colors in the same order as the palette you specify.

You specify the palette by naming a PPM file that has one pixel for each color in the palette.

Alternatively, consider the case that have a palette and you want to make sure your PNG contains only colors from the palette, approximating if necessary. You don't care what indexes the PNG uses internally for the colors (i.e. the order of the PNG palette). In this case, you don't need **-palette**. Pass the Netpbm input image and your palette PPM through **pnmremap**. Though you might think it would, using **-palette** in this case wouldn't even save **pnmtopng** any work.

-gamma *value*
Causes **pnmtopng** to create an gAMA chunk. This information helps describe how the color values in the PNG must be interpreted. Without the gAMA chunk, whatever interprets the PNG must get this information separately (or just assume something standard). If your input is a true PPM or PGM image, you should specify **-gamma .45**. But sometimes people generate images which are ostensibly PPM except the image uses a different gamma transfer function than the one specified for PPM. A common case of this is when the image is created by simple hardware that doesn't have digital computational ability. Also, some simple programs that generate images from scratch do it with a gamma transfer in which the gamma value is 1.0.

-hist Use this parameter to create a chunk that specifies the frequency (or histogram) of the colors in the image.

-chroma white point X and Y, red X and Y, green X and Y,
and blue X and Y This option specifies the white point and rgb values following the CIE-1931 spec.

-phys *x y unit*
This option determines the aspect ratio of the individual pixels of your image as well as the physical resolution of it.

unit is either **0** or *1*. When it is *1*, the option specifies the physical resolution of the image in pixels per meter. For example, **-phys 10000 15000 1** means that when someone displays the image, he should make it so that 10,000 pixels horizontally occupy 1 meter and 15,000 pixels vertically occupy one meter. And even if he doesn't take this advice on the overall size of the displayed image, he should at least make it so that each pixel displays as 1.5 times as high as wide.

When *unit* is **0**, that means there is no advice on the absolute physical resolution; just on the ratio of horizontal to vertical physical resolution.

This information goes into the PNG's pHYS chunk.

When you don't specify **-phys**, **pnmtopng** creates the image with no pHYS chunk, which means square pixels of no absolute resolution.

-text *file*

This option lets you include comments in the text chunk of the PNG output. *file* is the name of a file that contains your text comments.

Here is an example of a comment file:

```
Title          PNG file

Author         Bryan Henderson

Description    how to include a text chunk
               PNG file
"Creation date" 3-feb-1987

Software       pnmtopng
```

The format of the file is as follows: The file is divided into lines, delimited by newline characters. The last line need not end with a newline character. A group of consecutive lines represents a comment.

A "delimiter character" is a blank or tab or null character. The first line representing a comment must not start with a delimiter character. Every other line in the group is a "continuation line" and must start with a delimiter character.

The first line representing a comment consists of a keyword and the first line of comment text. The keyword begins in Column 1 of the file line and continues up to, but not including, the first delimiter character, or the end of the line, whichever is first. Exception: you can enclose the keyword in double quotes and spaces and tabs within the double quotes are part of the keyword. The quotes are not part of the keyword. A NUL character is not allowed in a keyword.

The first line of the comment text is all the text in the file line beginning after the keyword and any delimiter characters after it. immediately after the delimiter character that marks the end of the keyword.

A continuation line defines a subsequent line of the comment. The comment line is all the text on the continuation line starting with the first non-delimiter character.

There is one newline character between every two comment lines. There is no newline character after the last line of comment text.

There is no limit on the length of a file line or keyword or comment text line or comment text. There is no limit on the number of comments or size of or number of lines in the file.

-ztxt *file*

The same as -text, except **pnmtopng** considers the text compressed.

-time *yy-mm-dd hh:mm:ss*

or **-time** *yyyy-mm-dd hh:mm:ss* This option allows you to specify the modification time value to be placed in the PNG output. You can specify the year parameter either as a two digit or four digit value.

-filter *type*

This option is obsolete. Before Netpbm 10.22 (April 2004), this was the only way to specify a row filter. It specifies a single type of row filter, by number, that **pnmtopng** must use on each row.

Use **-nofilter**, **-sub**, **-up**, **-avg**, and **-paeth** in current Netpbm.

-nofilter

-sub

-up

-avg

-paeth Each of these options permits **pnmtopng** to use one type of row filter. **pnmtopng** chooses whichever of the permitted filters it finds to be optimal. If you specify none of these options, it is the same as specifying all of them -- **pnmtopng** uses any row filter type it finds optimal.

These options were new with Netpbm 10.22 (April 2004). Before that, you could use the **-filter** option to specify one permitted row filter type. The default, when you specify no filter options, was the same.

-compression *level*

To explicitly set the compression level of zlib use this parameter. Select a level between 0 for no compression (maximum speed) and 9 for maximum compression (minimum speed).

-force When you specify this, **pnmtopng** limits its optimizations. The resulting PNG output is as similar to the Netpbm input as possible. For example, the PNG output will not be paletted and the alpha channel will be represented as a full alpha channel even if the information could be represented more succinctly with a transparency chunk.

-libversion

This option causes **pnmtopng** to do nothing but display version information about the libraries it uses.

SEE ALSO

pngtopnm(1), **pnmremap**(1), **pnmgamma**(1), **pnm**(1)

For information on the PNG format, see http://schaik.com/png .

AUTHORS

Table Of Contents

NAME

pnmtopnm - copy a PNM image

SYNOPSIS

pnmtopnm

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtopnm simply copies a PNM image to Standard Output. The output has the same major PNM format (PBM, PGM, or PPM) and maxval as the input. This may seem an unnecessary duplication of **cat**, but it lets you convert between the plain (ASCII) and raw (binary) subformats of PNM. Use the **-plain** Netpbm common option to ensure the output is plain PNM, and don't use **-plain** to ensure the output is raw PNM. See
Common Options .

You don't normally need to convert between the PNM subformats, because any program that uses the Netpbm library to read a PNM image will read all of them directly. But there are a lot of programs that don't use the Netpbm library and understand only the raw format. Plain format is nice because it is human readable; people often use it to debug programs that process PNM images.

pnmtopnm is really just another name for the program **pamtopnm**. The latter does the job because like any Netpbm program that takes PAM input via the Netpbm programming library facilities, it also takes PNM input.

HISTORY

pnmtopnm was new in Netpbm 10.23 (July 2004). It obsoleted **pnmtoplainpnm**, which specifically did the conversion to plain PNM. There was no program to explicitly convert to raw PNM, but many Netpbm programs can be made, with the right options, to be idempotent (i.e. to do the same thing as **pnmtopnm**).

Then David Jones realized that the existing **pamtopnm** already did everything that **pnmtopnm** did and more, so in Netpbm 10.27 (March 2005), **pnmtopnm** became simply an alternate name for **pamtopnm**.

SEE ALSO

ppmtoppm(1) **pgmtopgm**(1) **pamtopnm**(1) **pnm**(1)

Table Of Contents

NAME

pnmtops - convert PNM image to PostScript

SYNOPSIS

pnmtops [**-scale**=*s*] [**-dpi**=*N*[**x***N*]] [**-imagewidth**=*n*] [**-imageheight**=*n*] [**-width**=*N*] [**-height**=*N*] [**-equalpixels**] [**-turn**|**-noturn**] [**-rle**|**-runlength**] [**-flate**] [**-ascii85**] [**-nocenter**] [**-nosetpage**] [**-level**=*N*] [**-psfilter**] [**-noshowpage**] [*pnmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtops reads a Netpbm image stream as input and produces Encapsulated PostScript (EPSF) as output.

If the input file is in color (PPM), **pnmtops** generates a color PostScript file. Some PostScript interpreters can't handle color PostScript. If you have one of these you will need to run your image through **ppmtopgm** first.

pnmotps produces Level 2 Postscript. (i.e. the line it places at the top of the file to indicate the version is '%%!PS-Adobe-2.0 EPSF-2.0').

If you specify no output dimensioning options, the output image is dimensioned as if you had specified **-scale=1.0**, which means aproximately 72 pixels of the input image generate one inch of output (if that fits the page).

Use **-imagewidth**, **-imageheight**, **-equalpixels**, **-width**, **-height**, and **-scale** to adjust that.

Each image in the input stream becomes one complete one-page Postscript program in the output. (This may not be the best way to create a multi-page Postscript stream; someone who knows Postscript should work on this).

What is Encapsulated Postscript?

Encapsulated Postscript (EPSF) is a subset of Postscript (i.e. the set of streams that conform to EPSF is a subset of those that conform to Postscript). It is designed so that an EPSF stream can be embedded in another Postscript stream. A typical reason to do that is where an EPSF stream describes a picture you want in a larger document.

An Encapsulated Postscript document conforms to the DSC (Document Structuring Convention). The DSC defines some Postscript comments (they're comments from a Postscript point of view, but have semantic value from a DSC point of view).

More information about Encapsulated Postscript is at http://www.cs.indiana.edu/docproject/programming/postscript/eps.html (1).

Many of the ideas in **pnmtops** come from Dirk Krause's **bmeps**. See SEE ALSO

OPTIONS

-imagewidth, **-imageheight**

Tells how wide and high you want the image on the page, in inches. The aspect ratio of the image is preserved, so if you specify both of these, the image on the page will be the largest image that will fit within the box of those dimensions.

If these dimensions are greater than the page size, you get Postscript output that runs off the page.

You cannot use **imagewidth** or **imageheight** with **-scale** or **-equalpixels**.

-equalpixels

This option causes the output image to have the same number of pixels as the input image. So if the output device is 600 dpi and your image is 3000 pixels wide, the output image would be 5 inches wide.

You cannot use **-equalpixels** with **-imagewidth**, **-imageheight**, or **-scale**.

-scale tells how big you want the image on the page. The value is the number of inches of output image that you want 72 pixels of the input to generate.

But **pnmtops** rounds the number to something that is an integral number of output device pixels. E.g. if the output device is 300 dpi and you specify **-scale=1.0**, then 75 (not 72) pixels of input becomes one inch of output (4 output pixels for each input pixel). Note that the **-dpi** option tells **pnmtops** how many pixels per inch the output device generates.

If the size so specified does not fit on the page (as measured either by the **-width** and **-height** options or the default page size of 8.5 inches by 11 inches), **pnmtops** ignores the **-scale** option, issues a warning, and scales the image to fit on the page.

-dpi=*N*[**x***N*]

This option specifies the dots per inch resolution of your output device. The default is 300 dpi. In theory PostScript is device-independent and you don't have to worry about this, but in practice its raster rendering can have unsightly bands if the device pixels and the image pixels aren't in sync.

Also this option is crucial to the working of the **equalpixels** option.

If you specify *N***x**N, the first number is the horizontal resolution and the second number is the vertical resolution. If you specify just a single number *N*, that is the resolution in both directions.

-width, **-height**

These options specify the dimensions, in inches, of the page on which the output is to be printed. This can affect the size of the output image.

The page size has no effect, however, when you specify the **-imagewidth**, **-imageheight**, or **-equalpixels** options.

These options may also affect positioning of the image on the page and even the paper selected (or cut) by the printer/plotter when the output is printed. See the **-nosetpage** option.

The default is 8.5 inches by 11 inches.

-turn

-noturn

These options control whether the image gets turned 90 degrees. Normally, if an image fits the page better when turned (e.g. the image is wider than it is tall, but the page is taller than it is wide), it gets turned automatically to better fit the page. If you specify the **-turn** option, **pnmtops** turns the image no matter what its shape; If you specify **-noturn**, **pnmtops** does *not* turn it no matter what its shape.

-rle

-runlength

These identical options tell **pnmtops** to use run length compression in encoding the image in the Postscript program. This may save time if the host-to-printer link is slow; but normally the

printer's processing time dominates, so **-rle** has no effect (and in the absence of buffering, may make things slower.

This may, however, make the Postscript program considerable smaller.

This usually doesn't help at all with a color image and **-psfilter**, because in that case, the Postscript program **pnmtops** creates has the red, green, and blue values for each pixel together, which means you would see long runs of identical bytes only in the unlikely event that the red, green, and blue values for a bunch of adjacent pixels are all the same. But without **-psfilter**, the Postscript program has all the red values, then all the green values, then all the blue values, so long runs appear wherever there are long stretches of the same color.

-flate This option tells **pnmtops** to use 'flate' compression (i.e. compression via the 'Z' library -- the same as PNG).

See the **-rle** option for information about compression in general.

You may not use this option together with **-rle**. You must specify **-psfilter** if you specify **-flate**.

This option was new in Netbpm 10.27 (March 2005).

-ascii85

By default, **pnmtops** uses 'asciihex' encoding of the sample values in the image raster. E.g. it would encode the number twenty as the two characters '14'. (Note that a Postscript program is composed of text, so the numbers that make up the raster have to be encoded into text somehow).

With the **-ascii85** option, **pnmtops** uses 'ascii85' encoding instead. I don't know what that is, but it appears to be a code in which a byte is represented by 5 odd characters. I can't see how this could be preferable to asciihex.

This option was new in Netbpm 10.27 (March 2005).

-psfilter

pnmtops can generate two different kinds of Encapsulated Postscript programs to represent an image. By default, it generates a program that redefines **readstring** in a custom manner and doesn't rely on any built-in Postscript filters. But with the **-psfilter** option, **pnmtops** leaves **readstring** alone and uses the built-in Postscript filters **/ASCII85Decode**, **/ASCIIHexDecode**, **/RunLengthDecode**, and **/FlateDecode**.

This option was new in Netbpm 10.27 (March 2005). Before that, **pnmtops** always used the custom **readstring**.

The custom code can't do flate or ascii85 encoding, so you must use **-psfilter** if you want those (see **-flate**, **-ascii85**).

-level This option determines the level (version number) of Postscript that **pnmtops** uses. By default, **pnmtops** uses Level 2. Some features of **pnmtops** are available only in higher Postscript levels, so if you specify too low a level for your image and your options, **pnmtops** fails. For example, **pnmtops** cannot do a color image in Level 1.

This option was new in Netpbm 10.27 (March 2005). Before that, **pnmtops** always used Level 2.

-dict This causes the Postscript program create a separated dictionary for its local variables and remove it from the stack as it exits.

This option was new in Netbpm 10.27 (March 2005).

-vmreclaim

This option causes the Postscript program to force a memory garbage collection as it exits.

This option was new in Netbpm 10.27 (March 2005).

-nocenter

By default, **pnmtops** centers the image on the output page. You can cause **pnmtops** to instead put the image against the upper left corner of the page with the **-nocenter** option. This is useful for programs which can include PostScript files, but can't cope with pictures which are not positioned in the upper left corner.

For backward compatibility, **pnmtops** accepts the option **-center**, but it has no effect.

-setpage

This causes **pnmtops** to include a 'setpagedevice' directive in the output. This causes the output to violate specifications of EPSF encapsulated Postscript, but if you're not using it in an encapsulated way, may be what you need. The directive tells the printer/plotter what size paper to use (or cut). The dimensions it specifies on this directive are those selected by the **-width** and **-height** options or defaulted.

From January through May 2002, the default was to include 'setpagedevice' and this option did not exist. Before January 2002, there was no way to include 'setpagedevice' and neither the **-setpage** nor **-nosetpage** option existed.

-nosetpage

This tells **pnmtops** not to include a 'setpagedevice' directive in the output. This is the default, so the option has no effect.

See the **-setpage** option for the history of this option.

-noshowpage

This tells **pnmtops** not to include a 'showpage' directive in the output. By default, **pnmtops** includes a 'showpage' at the end of the EPSF program According to EPSF specs, this is OK, and the program that includes the EPSF is supposed to redefine showpage so this doesn't cause undesirable behavior. But it's often easier just not to have the showpage.

This options was new in Netpbm 10.27 (March 2005). Earlier versions of **pnmtops** always include the showpage.

-showpage

This tells **pnmtops** to include a 'showpage' directive at the end of the EPSF output. This is the default, so the option has no effect.

This option was new in Netpbm 10.27 (March 2005).

SEE ALSO

bmeps converts from Netpbm and other formats to Encapsulated Postscript. It is suitable for hooking up to **dvips** so you can include an image in a Latex document just with an includegraphics directive.

bmeps has a few functions **pnmtops** does not, such as the ability to include a transparency mask in the Postscript program (but not from PAM input -- only from PNG input).

pnm(1), **gs**, **psidtopgm**(1), **pstopnm**(1), **pbmtolps**(1), **pbmtoepsi**(1), **pbmtopsg3**(1), **ppmtopgm**(1),

AUTHOR

Modified November 1993 by Wolfgang Stuerzlinger, *wrzl@gup.uni-linz.ac.at*

Table Of Contents

NAME

pnmtorast - convert a PPM into a Sun rasterfile

SYNOPSIS

pnmtorast [**-standard**|**-rle**] [*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtorastreads a portable pixmap as input and produces a Sun rasterfile as output.

Color values in Sun rasterfiles are eight bits wide, so **pnmtorast** will automatically scale colors to have a maxval of 255. An extra **pnmdepth** step is not necessary.

OPTIONS

The **-standard** option forces the result to be in RT_STANDARD form; the **-rle** option, RT_BYTE_ENCODED, which is smaller but, well, less standard. The default is **-rle**.

All options can be abbreviated to their shortest unique prefix.

SEE ALSO

rasttopnm(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmtorle - convert a Netpbm image file into an RLE image file.

SYNOPSIS

pnmtorle

[**-h**] [**-v**] [**-a**] [**-o** *outfile*] [*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

This program converts Netpbm image files into Utah RLE image files. You can include an alpha mask. If the input is a multiple image file, the output consists of several concatenated RLE images.

The RLE file will contain either a three channel color image (24 bits) or a single channel grayscale image (8 bits) depending upon the pnm file depth. If a converted ppm is 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** or **rlequant** utilities. We show an example of this later.

OPTIONS

-v This option will cause **pnmtorle** to operate in verbose mode. The header information is written to 'stderr'. Actually, there is not much header information stored in a Netpbm file, so this information is minimal.

-h This option allows the header of the Netpbm 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 causes **pnmtorle** to include an alpha channel in the output image. The alpha channel is based on the image: Wherever a pixel is black, the corresponding alpha value is transparent. Everywhere else, the alpha value is fully opaque.

-o *outfile*

If you specify this option, **pnmtorle** writes the output to this file. If *outfile* is **-** or you don't specify **-o**, **pnmtorle** writes the output to Standard Output.

pnmfile The name of the Netpbm image data file to be converted. If not specified, standard input is assumed.

EXAMPLES

pnmtorle -v file.ppm -o file.rle

While running in verbose mode, convert file.ppm to RLE format and store resulting data in file.rle.

pnmtorle -h file.pgm

Dump the header information of the Netpbm file called file.pgm.

SEE ALSO

rletopnm(1)

AUTHOR

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

Table Of Contents

NAME

pnmtosgi - convert a PNM image to a SGI image file

SYNOPSIS

pnmtosgi

[**-verbatim**|**-rle**]

[**-imagename** *Name*]

[**pnmfile**]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtosgi reads a PNM image as input and produces an SGI image file as output. The SGI image will be 2-dimensional (1 channel) for PBM and PGM input, and 3-dimensional (3 channels) for PPM.

OPTIONS

-verbatim

Write an uncompressed file.

-rle (default)

Write a compressed (run length encoded) file.

-imagename *name*

Write the string 'name' into the imagename field of the header. The name string is limited to 79 characters. If no name is given, pnmtosgi writes 'no name' into this field.

REFERENCES

SGI Image File Format documentation (draft v0.95) by Paul Haeberli (*paul@sgi.com*). Available via ftp at sgi.com:graphics/SGIIMAGESPEC.

SEE ALSO

pnm(1), **sgitopnm**(1)

AUTHOR

Table Of Contents

NAME

pnmtosir - convert a PNM image into a Solitaire format

SYNOPSIS

pnmtosir

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtosir reads a PNM image as input and produces a Solitaire Image Recorder format image.

pnmtosir produces an MGI TYPE 17 file for PBM and PGM files. For PPM, it writes a MGI TYPE 11 file.

SEE ALSO

sirtopnm(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

pnmtotiff - convert a PNM image to a TIFF file

SYNOPSIS

pnmtotiff

[**-none** | **-packbits** | **-lzw** | **-g3** | **-g4** | **-flate** | **-adobeflate**]

[**-2d**]

[**-fill**]

[**-predictor=***n*]

[**-msb2lsb**|**-lsb2msb**]

[**-rowsperstrip=***n*]

[**-minisblack**|**-miniswhite**|**mb**|**mw**]

[**-truecolor**]

[**-color**]

[**-indexbits=***bitwidthlist*]

[**-xresolution=***xres*]

[**-yresolution=***yres*]

[**-resolutionunit=**{**inch** | **centimeter** | **none** | **in** | **cm** | **no**}]

[**-indexbits=**[**1**[**2**[**4**[**8**]]]]]

[**-append**]

[*pnmfile*]

You can use the minimum unique abbreviation of the options. You can use two hyphens instead of one. You can separate an option name from its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtotiff reads a PNM image as input and produces a TIFF file as output.

Actually, it handles multi-image PNM streams, producing multi-image TIFF streams (i.e. a TIFF stream with multiple 'directories'). But before Netpbm 10.27 (March 2005), it ignored all but the first PNM image in the input stream.

The Output File

The output goes to Standard Output. **pnmtotiff** approaches this output file differently from Unix and Netpbm convention. This is entirely due to **pnmtotiff**'s use of the TIFF library to do all TIFF output.

- The output file must be seekable. **pnmtotiff** does not write it sequentially. Therefore, you can't use a pipe; you can't pipe the output of **pnmtotiff** to some other program. But any regular file should work.

- If the output file descriptor is readable, you must either specify **-append** so as to add to the existing file, or make sure the file is empty. Otherwise, **pnmtotiff** will fail with an unhelpful message telling you that a TIFF library function failed to open the TIFF output stream.

- If you are converting multiple images (your input stream contains multiple images), the output file must be both readable and writable.

If you're using a Unix command shell to run **pnmtotiff**, you use facilities of your shell to set up Standard Output. In Bash, for example, you would set up a write-only Standard Output to the file /tmp/myimage.tiff like this:

```
$ pnmtotiff myimage.pnm >/tmp/myimage.tiff
```

In Bash, you would set up a read/write Standard Output to the file /tmp/myimage.tiff like this:

```
$ pnmtotiff myimage.pnm 1<>/tmp/myimage.tiff
```

OPTIONS

Compression

By default, **pnmtotiff** creates a TIFF file with no compression. This is your best bet most of the time. If you want to try another compression scheme or tweak some of the other even more obscure output options, there are a number of flags to play with.

Before Netpbm 8.4 (April 2000), the default was to use LZW compression. But then new releases of the TIFF library started omitting the LZW compression capability due to concern about patents on LZW. So since then, the default has been no compression. The LZW patents have now expired and new TIFF libraries do LZW, but the **pnmtotiff** behavior remains the same for compatibility with older TIFF libraries and applications of **pnmtotiff**.

The **-none**, **-packbits**, **-lzw**, **-g3**, **-g4**, **-flate**, and **-adobeflate** options are used to override the default and set the compression scheme used in creating the output file. The CCITT Group 3 and Group 4 compression algorithms can be used only with bilevel data. The **-2d** and **-fill** options are meaningful only with Group 3 compression: **-2d** requests 2-dimensional encoding, while **-fill** requests that each encoded scanline be zero-filled to a byte boundary. The **-predictor** option is meaningful only with LZW compression: a predictor value of 2 causes each scanline of the output image to undergo horizontal differencing before it is encoded; a value of 1 forces each scanline to be encoded without differencing. By default, **pnmtotiff** creates a TIFF file with msb-to-lsb fill order. The **-msb2lsb** and **-lsb2msb** options are used to override the default and set the fill order used in creating the file.

With some older TIFF libraries, **-lzw** doesn't work because the TIFF library doesn't do LZW compression. This is because of concerns about Unisys's patent on LZW which was then in force. Actually, with very old TIFF libraries, **-lzw** works because no distributors of the TIFF library were sensitive yet to the patent issue.

-flate chooses 'flate' compression, which is the patent-free compression common in the Unix world implemented by the 'Z' library. It is what the PNG format uses.

Fill Order

The **-msb2lsb** and **lsb2msb** options controll the fill order.

The fill order is the order in which pixels are packed into a byte in the Tiff raster, in the case that there are multiple pixels per byte. msb-to-lsb means that the leftmost columns go into the most significant bits of the byte in the Tiff image. However, there is considerable confusion about the meaning of fill order. Some believe it means whether 16 bit sample values in the Tiff image are little-endian or big-

endian. This is totally erroneous (The endianness of integers in a Tiff image is designated by the image's magic number). However, ImageMagick and older Netpbm both have been known to implement that interpretation. 2001.09.06.

If the image does not have sub-byte pixels, these options have no effect other than to set the value of the FILLORDER tag in the Tiff image (which may be useful for those programs that misinterpret the tag with reference to 16 bit samples).

Color Space

-color tells **pnmtotiff** to produce a color, as opposed to grayscale, TIFF image if the input is PPM, even if it contains only shades of gray. Without this option, **pnmtotiff** produces a grayscale TIFF image if the input is PPM and contains only shades of gray, and at most 256 shades. Otherwise, it produces a color TIFF output. For PBM and PGM input, **pnmtotiff** always produces grayscale TIFF output and this option has no effect.

The **-color** option can prevent **pnmtotiff** from making two passes through the input file, thus improving speed and memory usage. See Multiple Passes .

-truecolor tells **pnmtotiff** to produce the 24-bit RGB form of TIFF output if it is producing a color TIFF image. Without this option, **pnmtotiff** produces a colormapped (paletted) TIFF image unless there are more than 256 colors (and in the latter case, issues a warning).

The **-truecolor** option can prevent **pnmtotiff** from making two passes through the input file, thus improving speed and memory usage. See Multiple Passes .

The **-color** and **-truecolor** options did not exist before Netpbm 9.21 (December 2001).

If **pnmtotiff** produces a grayscale TIFF image, this option has no effect.

The **-minisblack** and **-miniswhite** options force the output image to have a 'minimum is black' or 'minimum is white' photometric, respectively. If you don't specify either, **pnmtotiff** uses minimum is black except when using Group 3 or Group 4 compression, in which case **pnmtotiff** follows CCITT fax standards and uses 'minimum is white.' This usually results in better compression and is generally preferred for bilevel coding.

Before February 2001, **pnmtotiff** always produced 'minimum is black,' due to a bug. In either case, **pnmtotiff** sets the photometric interpretation tag in the TIFF output according to which photometric is actually used.

The **-indexbits** option is meaningful only for a colormapped (paletted) image. In this kind of image, the raster contains values which are indexes into a table of colors, with the indexes normally taking less space that the color description in the table. **pnmtotiff** can generate indexes of 1, 2, 4, or 8 bits. By default, it will use 8, because many programs that interpret TIFF images can't handle any other width.

But if you have a small number of colors, you can make your image considerably smaller by allowing fewer than 8 bits per index, using the **-indexbits** option. The value is a comma-separated list of the bit widths you allow. **pnmtotiff** chooses the smallest width you allow that allows it to index the entire color table. If you don't allow any such width, **pnmtotiff** fails. Normally, the only useful value for this option is **1,2,4,8**, because a program either understands the 8 bit width (default) or understands them all.

In a Baseline TIFF image, according to the 1992 TIFF 6.0 specification, 4 and 8 are the only valid widths. There are no formal standards that allow any other values.

This option was added in June 2002. Before that, only 8 bit indices were possible.

Resolution

A Tiff image may contain information about the resolution of the image, which means how big in real dimensions (centimeters, etc.) each pixel in the raster is. You control that with the **-xresolution**, **-yresolution**, and **-resolutionunit** options.

These options do not control how many pixels **pnmtotiff** generates or how much information is in the pixels. They control only the value of tags that may or may not be used by whatever reads the image.

The value of the **-xresolution** option is a floating point decimal number that tells how many pixels there are per unit of distance in the horizontal direction. **-yresolution** is analogous for the vertical

direction.

The unit of distance is given by the value of the **-resolutionunit** option. That value is either **inch**, **centimeter** or **none** (or abbreviations **in**, **cm**, or **no**). **none** means the unit is arbitrary or unspecified. This could mean that the creator and user of the image have a separate agreement as to what the unit is. But usually, it just means that the horizontal and vertical resolution values cannot be used for anything except to determine aspect ratio (because even though the unit is arbitrary or unspecified, it has to be the same for both resolution numbers).

If you don't specify **-xresolution**, the Tiff image does not contain horizontal resolution information. Likewise for **-yresolution**. If you don't specify **-resolutionunit**, the default is inches.

Before Netpbm 10.16 (June 2003), **-resolutionunit** did not exist and the resolution unit was always inches.

Other

You can use the **-rowsperstrip** option to set the number of rows (scanlines) in each strip of data in the output file. By default, the output file has the number of rows per strip set to a value that will ensure each strip is no more than 8 kilobytes long.

The **-append** option tells **pnmtotiff** to add images to the existing output file (a TIFF file may contain multiple images) instead of the default, which is to replace the output file.

-append was new in Netpbm 10.27 (March 2005).

NOTES

There are myriad variations of the TIFF format, and this program generates only a few of them. **pnmtotiff** creates a grayscale TIFF file if its input is a PBM (monochrome) or PGM (grayscale) file. **pnmtotiff** also creates a grayscale file if it input is PPM (color), but there is only one color in the image. If the input is a PPM (color) file and there are 256 colors or fewer, but more than 1, **pnmtotiff** generates a color palette TIFF file. If there are more colors than that, **pnmtotiff** generates an RGB (not RGBA) single plane TIFF file. Use **pnmtotiffcmyk** to generate the cyan-magenta-yellow-black ink color separation TIFF format.

The number of bits per sample in the TIFF output is determined by the maxval of the PNM input. If the maxval is less than 256, the bits per sample in the output is the smallest number that can encode the maxval. If the maxval is greater than or equal to 256, there are 16 bits per sample in the output.

Multiple Passes

pnmtotiff reads the input image once if it can, and otherwise twice. It needs that second pass (which happens before the main pass, of course) to analyze the colors in the image and generate a color map (palette) and determine if the image is grayscale. So the second pass happens only when the input is PPM. And you can avoid it then by specifying both the **-truecolor** and **-color** options.

If the input image is small enough to fit in your system's file cache, the second pass is very fast. If not, it requires reading from disk twice, which can be slow.

When the input is from a file that cannot be rewound and reread, **pnmtotiff** reads the entire input image into a temporary file which can, and works from that. Even if it needs only one pass.

Before Netpbm 9.21 (December 2001), **pnmtotiff** always read the entire image into virtual memory and then did one, two, or three passes through the memory copy. The **-truecolor** and **-color** options did not exist. The passes through memory would involve page faults if the entire image did not fit into real memory. The image in memory required considerably more memory (12 bytes per pixel) than the cached file version of the image would.

SEE ALSO

tifftopnm(1), **pnmtotiffcmyk**(1), **pnmdepth**(1), **pnm**(1)

AUTHOR

Derived by Jef Poskanzer from ras2tiff.c, which is Copyright (c) 1990 by Sun Microsystems, Inc. Author: Patrick J. Naughton (*naughton@wind.sun.com*).

Table Of Contents

NAME

pnmtotiffcmyk - convert a Netpbm image into a CMYK encoded TIFF file

SYNOPSIS

pnmtotiffcmyk

[*Compargs*][*Tiffargs*][*Convargs*][*pnmfile*]

Compargs:

[**-none**|**-packbits**|**-lzw**

[**-predictor** *n*]]

Tiffargs:

[**-msb2lsb**|**-lsb2msb**]

[**-rowsperstrip** *n*] [**-lowdotrange** *n*]

[**-highdotrange** *n*] [**-knormal**|**-konly**|**-kremove**]

Convargs:

[[**-default**][**Defargs**]|**-negative**]

Defargs:

[**-theta** *deg*]

[**-gamma** *n*]

[**-gammap** *-1* | **-gammap** *n*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtotiffcmykreads a PNM image as input and produces a CMYK encoded TIFF file as output. It optionally modifies the color balance and black level, and modifies removal of CMY from under K.

OPTIONS

The order of most options is not important, but options for particular conversion algorithms must appear after the algorithm is selected (**-default**,**-negative**). If you don't select an algorithm, **pnmtotiffcmyk** assumes **-default** and the appropriate options (**-theta**,**-gamma**,**-gammap**) can appear anywhere.

-none,-packbits,-lzw,-predictor

Tiff files can be compressed. By default, **pnmtotiffcmyk** uses LZW decompression, but (apparently) some readers cannot read this, so you may want to select a different algorithm (**-none**,**-packbits**). For LZW compression, a **-predictor** value of 2 forces horizontal differencing of scanlines before encoding; a value of 1 forces no differencing.

-msb2lsb,-lsb2msb

These options control fill order (default is **-msb2lsb**).

-rowsperstrip

This sets the number of rows in an image strip (data in the Tiff files generated by this program is stored in strips - each strip is compressed individually). The default gives a strip size of no more than 8 kb.

-lowdotrange,-highdotrange

These options set tag values that may be useful for printers.

-knormal,-kremove,-konly

These options control the calculation of the CMYK ink levels. They are useful only for testing and debugging the code.

-kremove sets the black (K) levels to zero while leaving the other ink levels as they would be if the black level were normal.

-konly sets all inks to the normal black value.

-default,-negative

These options control what ink levels **pnmtotiffcmyk** uses to represent each input color.

-negative selects a simple algorithm that generates a color negative. None of the following options apply to this algorithm. The algorithm is included as an example in the source code to help implementors of other conversions.

-default is not necessary, unless you have to countermand a **-negative** on the same command line.

The default conversion from RGB to CMYK is as follows: The basic values of the 3 pigments are C = 1-R, M = 1-G, Y = 1-B. From this, **pnmtotiffcmyk** chooses a black (K) level which is the minimum of those three. It then replaces that much of the 3 pigments with the black. I.e. it substracts K from each of the basic C, M, and Y values.

The options below modify this conversion.

-theta *deg*

-theta provides a simple correction for any color bias that may occur in the printed image because, in practice, inks do not exactly complement the primary colors. It rotates the colors (before black replacement) by *deg* degrees in the color wheel. Unless you are trying to produce unusual effects you will need to use small values. Try generating three images at -10, 0 (the default) and 10 degrees and see which has the best color balance.

-gamma *n*

-gamma applies a gamma correction to the black (K) value described above. Specifically, instead of calculating the K value as min(C,M,Y), **pnmtotiffcmyk** raises that value (normalised to the range 0 to 1) to the *n*th power. In practice, this means that a value greater than 1 makes the image lighter and a value less than 1 makes the image darker. The range of allowed values is 0.1 to 10.

-gammap *n*

This option controls the black replacement.

If you specify **-gammap**, **pnmtotiffcmyk** uses the specified gamma value in computing how much ink to remove from the 3 pigments, but still uses the regular gamma value (**-gamma** option) to generate the actual amount of black ink with which to replace it.

Values of *n* from 0.01 to 10 are valid.

For example, it may be best to only subtract black from the colored inks in the very darkest regions. In that case, *n* should be a large value, such as 5.

As a special case, if *n* is -1, **pnmtotiffcmyk** does not remove any pigment (but still adds the black ink). This means dark areas are even darker. Furthermore, when printed, dark areas contain a lot of ink which can make high contrast areas, like lettering, appear fuzzy. It's hard to see what the utility of this is.

SEE ALSO

pnmtotiff(1), **tifftopnm**(1), **pnm**(1)

AUTHOR

Much of the code uses ideas from other Netpbm programs, written by Jef Poskanzer (thanks go to him and libtiff maintainer Sam Leffler). A small section of the code - some of the tiff tag settings - is derived directly from pnmtotiff, by Jef Poskanzer, which, in turn, acknowledges Patrick Naughton with the following text:

Table Of Contents

NAME

pnmtoxwd - convert a PNM into an X11 window dump

SYNOPSIS

pnmtoxwd

[**-pseudodepth** *n*]

[**-directcolor**]

[*pnmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

pnmtoxwd reads a PNM image as input and produces an X11 window dump as output. You can display this output with **xwdu**.

Normally, pnmtoxwd produces a StaticGray dump file for PBM and PGM files. For PPM, it writes a PseudoColor dump file if there are up to 256 colors in the input, and a DirectColor dump file otherwise. The **-directcolor** flag can be used to force a DirectColor dump. And the **-pseudodepth** flag can be used to change the depth of PseudoColor dumps from the default of 8 bits / 256 colors.

In an X11 window dump file, various integers can be represented in either big endian (most significant byte first) or little endian code. Those generated by **pnmtoxwd** are always big endian.

SEE ALSO

xwdtopnm(1), **pnm**(1), **xwud**

AUTHOR

Table Of Contents

NAME

ppm3d - convert two PPM images into a red/blue 3d glasses PPM

SYNOPSIS

ppm3d *leftppmfile rightppmfile* [*horizontal_offset*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppm3d reads two PPM images as input and produces a PPM as output, with the images overlapping by *horizontal_offset* pixels in blue/red format. The idea is that if you look at the image with 3-D glasses (glasses that admit only red through one eye and only green through the other), you see an image with depth. This is called a stereogram.

horizontal_offset defaults to 30 pixels. The input PPMs must be the same dimensions.

To make a different kind of stereogram, use **pamstereogram**. That makes a steregram that you view without special glasses, just by letting your eyes unfocus so that each eye sees different parts of the image.

SEE ALSO

pamstereogram(1) **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmbrighten - change a PPM image's Saturation and Value

SYNOPSIS

ppmbrighten [**-normalize**] [**-saturation** [+|-*saturation_percent*]] [**-value** [+|-*value_percent*]] *ppmfile*

OPTION USAGE

All options can be abbreviated to their shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmbrighten increases or decreases the Saturation and Value (from the HSV color space) of each pixel of a PPM image. You specify the per centage change for each of those parameters.

You can also remap the colors of the pixels so their Values cover the full range of possible Values.

Hue-Saturation-Value, or HSV, is one way to represent a color, like the more well-known RGB. Hue, Saturation, and Value are numbers in the range from 0 to 1. We always capitalize them in this document when we mean the number from the HSV color space, especially since "value" as a conventional English word has a much more abstract meaning.

Value is a measure of how much total light intensity is in the color, relative to some specified maximum (the PPM format is also defined in terms of a specified maximum intensity -- For the purposes of this program, they are the same). In particular, it is the intensity of the most intense primary color component of the color divided by the maximum intensity possible for a component. Zero Value means black. White has full Value.

Hue is an indication of the secondary color with the same intensity that most closely approximates the color. A secondary color is made of a combination of at most two of the primary colors.

Saturation is a measure of how close the color is to the color indicated by the Hue and Value. A lower number means more light of the third primary color must be added to get the exact color. Full Saturation means the color is a secondary color. Zero Saturation means the color is gray (or black or white). Decreasing the saturation of a color tends to make it washed out.

If it is impossible to increase the Value of a pixel by the amount you specify (e.g. the Value is .5 and you specify +200%), **ppmbrighten** increases it to full Value instead.

If it is impossible to increase the Saturation of a pixel by the amount you specify (e.g. it is already half saturated and you specify +200%), **ppmbrighten** increases it to full Saturation instead.

For a simpler kind of brightening, you can use **pamfunc -multiplier** simply to increase the intensity of each pixel by a specified per centage, clipping each RGB component where the calculated intensity would exceed full intensity. Thus, the brightest colors in the image would change chromaticity in addition to not getting the specified intensity boost. For *decreasing* brightness, **pamfunc** should do the same thing as **ppmbrighten**.

ppmflash does another kind of brightening. It changes the color of each pixel to bring it a specified per centage closer to white. This increases the value and saturation.

EXAMPLES

To double the Value of each pixel:
ppmbrighten -v 100

To double the Saturation and halve the value of each pixel:
ppmbrighten -s 100 -v -50

OPTIONS

-value *value_percent*

This option specifies the amount, as a per centage, by which you want to change the Value of each pixel. It may be negative.

-saturation *value_percent*

This option specifies the amount, as a per centage, by which you want to change the Saturation of each pixel. It may be negative.

-normalize

This option causes **ppmbrighten** to linearly remap the Values of the pixels to cover the range 0 to 1. The option name is wrong -- this operation is not normalization (it was named in error and the name has been kept for backward compatibility).

ppmbrighten applies the brightening that you specify with the **-value** option *after* the remapping.

Before Netpbm 10.14 (March 2003), your input must be from a seekable file (not a pipe) to use **-normalize**. If it isn't, the program fails with a bogus error message.

SEE ALSO

pgmnorm(1), **ppmdim**(1), **pamfunc**(1), **ppmflash**(1), **pnmdepth**(1), **pnmgamma**(1), **ppmhist**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmchange - change all pixels of one color to another in a PPM image

SYNOPSIS

ppmchange

[**-closeness=***closeness_percent*] [**-remainder=***remainder_color*] [**-closeok**] [*oldcolor newcolor*] ... [*ppmfile*]

EXAMPLES

ppmchange red blue redimage.ppm >blueimage.ppm

ppmchange red red -remainder=black myimage.ppm >redblack.ppm

ppmchange -closeness=90 white white black black

DESCRIPTION

This program is part of **Netpbm**(1).

ppmchange reads a PPM image as input and changes all pixels of colr *oldcolor* to color *newcolor*.

You may specify up to 256 oldcolor/newcolor pairs on the command line. **ppmchange** leaves all colors not mentioned unchanged, unless you specify the **-remainder** option, in which case they are all changed to the single specified color.

You can specify that colors similar, but not identical, to the ones you specify get replaced by specifying a 'closeness' factor.

Specify the colors as described for the argument of the **ppm_parsecolor()** library routine .

If a pixel matches two different *oldcolor*s, **ppmchange** replaces it with the *newcolor* of the leftmost specified one.

The maxval of the output image is the same as that of the input image. If a *newcolor* you specify cannot be exactly represented in that maxval, **ppmchange** fails unless you specify the **-closeok** option, in which case **ppmchange** assumes a color that is as close as possible to what you specified but can be represented with your maxval.

A common way that you can have this maxval problem, where the color you specify cannot be represented with your maxval, is that your input is a PBM (black and white) image that you are colorizing. The maxval in this case is 1, which severely limits the colors to which you can change. To avoid this problem, use **pnmdepth** to make the maxval of your input something consistent with your colors. 255 is usually a good choice.

Before Netpbm 10.22 (April 2004), **ppmchange** always behaved as if the user specified **-closeok** and there was no **-closeok** option.

OPTIONS

-closeness *closeness_percent*

closeness is an integer per centage indicating how close to the color you specified a pixel must be to get replaced. By default, it is 0, which means the pixel must be the exact color you specified.

A pixel gets replaced if the distance in color between it and the color you specified is less than or equal to *closeness* per cent of the maxval.

The 'distance' in color is defined as the cartesian sum of the individual differences in red,

green, and blue intensities between the two pixels, normalized so that the difference between black and white is 100%.

This is probably simpler than what you want most the time. You probably would like to change colors that have similar chrominance, regardless of their intensity. So if there's a red barn that is variously shadowed, you want the entire barn changed. But because the shadowing significantly changes the color according to **ppmchange**'s distance formula, parts of the barn are probably about as distant in color from other parts of the barn as they are from green grass next to the barn.

Maybe **ppmchange** will be enhanced some day to do chrominance analysis.

-closeok

This option affects how **ppmchange** interprets a color you specify in the arguments. When you specify this option, **ppmchange** may use a color close to, but not the same as what you specify. See the description section for details.

This option was new in Netpbm 10.22 (April 2004). Before that, **ppmchange** always behaved as if you specified this option.

-remainder *color*

ppmchange changes all pixels which are not of a color for which you specify an explicit replacement color on the command line to color *color*.

An example application of this is

ppmchange -remainder=black red red

to lift only the red portions from an image, or
ppmchange -remainder=black red white | ppmtopgm

to create a mask file for the red portions of the image.

SEE ALSO

pgmtoppm(1), **ppmcolormask**(1), **ppm**(1)

AUTHOR

Wilson H. Bent. Jr. (*whb@usc.edu*) with modifications by Alberto Accomazzi (*alberto@cfa.harvard.edu*)

Table Of Contents

NAME

ppmcie - draw a CIE color chart as a PPM image

SYNOPSIS

ppmcie

[**-rec709**|**-cie**|**-ebu**|**-hdtv**|**-ntsc**|**-smpte**] [**-xy**|**-upvp**]

[**-red** *rx ry*]

[**-green** *gx gy*]

[**-blue** *bx by*]

[**-white** *wx wy*]

[**-size** *edge*]

[{**-xsize**|**-width**} *width*]

[{**-ysize**|**-height**} *height*]

[**-noblack**] [**-nowpoint**] [**-nolabel**] [**-noaxes**] [**-full**]

You may abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmcie creates a PPM file containing a plot of the CIE 'tongue' color chart -- to the extent possible in a PPM image. Alternatively, creates a pseudo-PPM image of the color tongue using RGB values from a color system of your choice.

The CIE color tongue is an image of all the hues that can be described by CIE X-Y chromaticity coordinates. They are arranged on a two dimensional coordinate plane with the X chromaticity on the horizontal axis and the Y chromaticity on the vertical scale. (You can choose alternatively to use CIE u'-v' chromaticity coordinates, but the general idea of the color tongue is the same).

Note that the PPM format specifies that the RGB values in the file are from CIE Rec. 709 color system, gamma-corrected. And positive. See **ppm**(1)**for**details.**If**you**use**one**of**the color system options on **ppmcie**, what you get is not a true PPM image, but is very similar. If you display such **ppmcie** output using a device that expects PPM input (which includes just about any computer graphics display program), it will display the wrong colors.

However, you may have a device that expects one of these variations on PPM.

In every RGB color system you can specify, including the default (which produces a true PPM image) there are hues in the color tongue that can't be represented. For example, monochromatic blue-green with a wavelength of 500nm cannot be represented in a PPM image.

For these hues, **ppmcie** substitutes a similar hue as follows: They are desaturated and rendered as the shade where the edge of the Maxwell triangle intersects a line drawn from the requested shade to the white point defined by the color system's white point. Furthermore, unless you specify the **-full** option, **ppmcie** reduces their intensity by 25% compared to the true hues in the image.

ppmcie draws and labels the CIE X-Y coordinate axes unless you choose otherwise with options.

ppmcie draws the Maxwell triangle for the color system in use on the color tongue. The Maxwell triangle is the triangle whose vertices are the primary illuminant hues for the color system. The hues inside the triangle show the color gamut for the color system. They are also the only ones that are

correct for the CIE X-Y chromaticity coordinates shown. (See explanation above).

ppmcie also places a mark at the color system's white point and displays in text the CIE X-Y chromaticities of the primary illuminants and white point for the color system. You can turn this off with options, though.

ppmcie annotates the periphery of the color tongue with the wavelength, in nanometers of the monochromatic hues which appear there.

Finally, **ppmcie** displays the black body chromaticity curve for Planckian radiators from 1000 to 30000 kelvins on the image.

You can choose from several standard color systems, or specify one of your own numerically.

CIE charts, by their very nature, contain a very large number of colors. If you're encoding the chart for a color mapped device or file format, you'll need to use **pnmquant** or **ppmdither** to reduce the number of colors in the image.

OPTIONS

-rec709

-cie

-ebu

-hdtv

-ntsc

-smpte Select a standard color system whose gamut to plot. The default is **-rec709**, which chooses CIE Rec. 709, gamma-corrected. This is the only color system for which **ppmcie**'s output is a true PPM image. See explanation above. **-ebu** chooses the primaries used in the PAL and SECAM broadcasting standards. **-ntsc** chooses the primaries specified by the NTSC broadcasting system (few modern monitors actually cover this range). **-smpte** selects the primaries recommended by the Society of Motion Picture and Television Engineers (SMPTE) in standards RP-37 and RP-145, and **-hdtv** uses the much broader *HDTV ideal* primaries. **-cie** chooses a color system that has the largest possible gamut within the spectrum of the chart. This is the same color system as you get with the **-cie** option to John Walker's **cietoppm** program.

-xy plot CIE 1931 x y chromaticities. This is the default.

-upvp plot u' v' 1976 chromaticities rather than CIE 1931 x y chromaticities. The advantage of u' v' coordinates is that equal intervals of distance on the u' v' plane correspond roughly to the eye's ability to discriminate colors.

-red *rx ry*
specifies the CIE *x* and *y* co-ordinates of the red illuminant of a custom color system and selects the custom system.

-green *gx gy*
specifies the CIE *x* and *y* co-ordinates of the green illuminant of the color system and selects the custom system.

-blue *bx by*
specifies the CIE *x* and *y* co-ordinates of the blue illuminant of the color system and selects the custom system.

-white *wx wy*
specifies the CIE *x* and *y* co-ordinates of the white point of the color system and selects the custom system.

-size *edge*
Create a pixmap of *edge* by *edge* pixels. The default is 512x512.

-xsize|-width *width*
Sets the width of the generated image to *width* pixels. The default width is 512 pixels. If the height and width of the image are not the same, the CIE diagram will be stretched in the longer dimension.

-ysize|-height *height*
Sets the height of the generated image to *height* pixels. The default height is 512 pixels. If the height and width of the image are not the same, the CIE diagram will be stretched in the longer dimension.

-noblack
Don't plot the black body chromaticity curve.

-nowpoint
Don't plot the color system's white point.

-nolabel
Omit the label.

-noaxes
Don't plot axes.

-full Plot the entire CIE tongue in full intensity; don't enhance the gamut of the specified color system.

SEE ALSO

ppmdither(1), **pnmquant**(1), **ppm**(1)

AUTHOR

WWW home page: http://www.fourmilab.ch/

Table Of Contents

NAME

ppmcolormask - produce mask of areas of a certain color in a PPM file

SYNOPSIS

ppmcolormask

color

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmcolormask reads a PPM file as input and produces a PBM (bitmap) file as output. The output file is the same dimensions as the input file and is black in all places where the input file is the color *color*, and white everywhere else.

The output of **ppmcolormask** is useful as an alpha mask input to **pamcomp**. Note that you can generate such an alpha mask automatically as you convert to PNG format with **pnmtopng**(1).**Use**its-**transparent** option.

ppmfile is the input file. If you don't specify *ppmfile*, the input is from Standard Input.

The output goes to Standard Output.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

SEE ALSO

pgmtoppm(1), **pamcomp**(1), **pbmmask**(1), **ppm**(1)

NAME

ppmcolors − see http://netpbm.sourceforge.net/doc//ppmcolors.html

DESCRIPTION

ppmcolors is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//ppmcolors.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

ppmdim - dim a PPM image

SYNOPSIS

ppmdim *dimfactor* [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

This program is largely obsoleted by the more general **pamfunc**(1)**(use**the-**multiplier** option). **ppmdim** remains for backward compatibility and also because its use of integer arithmetic may make it faster.

ppmdim reads a PPM image input. Diminishes its brightness by the specified dimfactor. The dimfactor may be in the range from 0.0 (total blackness, deep night, nada, null, nothing) to 1.0 (original picture's brightness).

SEE ALSO

ppm(1), **pamfunc**(1),

AUTHOR

Table Of Contents

NAME

ppmdist - simplistic grayscale assignment for machine generated, color images

SYNOPSIS

ppmdist

[**-intensity**|**-frequency**]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmdist reads a PPM image as input and performs a simplistic grayscale assignment intended for use with grayscale or bitmap printers.

Often conversion from ppm to pgm will yield an image with contrast too low for good printer output. The program maximizes contrast between the gray levels output.

A ppm input of n colors is read, and a pgm of n gray levels is written. The gray levels take on the values 0..n-1, while maxval takes on n-1.

The mapping from color to stepped grayscale can be performed in order of input pixel intensity, or input pixel frequency (number of repetitions).

This program is helpful only for images with a very small number of colors.

OPTIONS

-frequency

Sort input colors by the number of times a color appears in the input, before mapping to evenly distributed graylevels of output.

-intensity

Sort input colors by their grayscale intensity, before mapping to evenly distributed graylevels of output. This is the default.

SEE ALSO

ppmtopgm(1), **ppmhist**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmdither - ordered dither for color images

SYNOPSIS

ppmdither

[**-dim** *power*]

[**-red** *shades*]

[**-green** *shades*]

[**-blue** *shades*]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmdither reads a PPM image as input, and applies dithering to it to reduce the number of colors used down to the specified number of shades for each primary. The default number of shades is red=5, green=9, blue=5, for a total of 225 colors. To convert the image to a binary rgb format suitable for color printers, use -red 2 -green 2 -blue 2.

OPTIONS

-dim *power*

The size of the dithering matrix. The dithering matrix is a square whose dimension is a power of 2. *power* is that power of 2. The default is 4, for a 16 by 16 matrix.

-red *shades*

The number of red shades to be used, including black; minimum of 2.

-green *shades*

The number of green shades to be used, including black; minimum of 2.

-blue *shades*

The number of blue shades to be used, including black; minimum of 2.

SEE ALSO

pnmdepth(1), **pnmquant**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmfade - generate a transition between two image files using special effects

SYNOPSIS

ppmfade [**-f** *first.ppm*] [**-l** *last.ppm*] [**-mix**|**-spread**|**-shift**| **-relief**|**-oil**|**-edge**|**-bentley**|**-block**] [**-base** *name*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmfadegenerates a transition between either two input images or between one input image and black. You can use the 30 intermediate images generated to show a smooth transition between segments of a movie. The input and output images are in the PPM format. If you specify both input images, they should both be the same size. If you want to fade from black to an image, specify only the last image. If you want to fade from an image to black, specify only the first image. **ppmfade** names the resulting image files *base***.***nnnn***.ppm**, where *nnnn* is a number varying between 0001 and 0030 and *base* is what you specify with via the **-base** option (default **fade**).

Another way to convert by steps from one image to another is morphing. You can use **xmorph** to do that.

OPTIONS

-f *first.ppm*

This is the image file (PPM format) to be used at the beginning of the transition. If you don't specify this, the fade will start from black.

-l *last.ppm*

This is the image file (PPM format) to be used at the ending of the transition. If you don't specify this, the fade will end with black.

-mix The two images are superimposed with the brightness of the first image decreasing from full to none and the brightness of the final image increasing from none to full. The transition is quadratic in brightness with faster transition in the beginning and slower at the end.

-spread

The pixels in the first image will be moved (spread) further and further from their original location and then moved into the proper location in the final image. This is the default transition.

-shift The pixels in the first image will be shifted further and further horizontally from their original location and then moved into the proper location in the final image.

-relief The first image is faded to a Laplacian relief filtered version of the first image. This is then faded to a Laplacian relief filtered version of the second image and finally faded to the final image.

-oil The first image is faded to an 'oil transfer' version of the first image. This is then faded to an 'oil transfer' version of the second image and finally faded to the final image.

-edge The first image is faded to an edge detected version of the first image. This is then faded to an edge detected version of the second image and finally faded to the final image.

-bentley

The first image is faded to a 'Bentley Effect' version of the first image. This is then faded to a 'Bentley Effect' version of the second image and finally faded to the final image.

-block The first image is defocused to small blocks. The small blocks are converted to match a defocused version of the last image. The block version of the last image is then focused to the final image.

-base *name*

This forms part of the output filenames, as described above.

EXAMPLES

ppmfade -f teapot.ppm -l pyr.ppm

Fade from teapot.ppm to pyr.ppm generating fade.0001.ppm to fade.0030.ppm using the 'spread' transition.

ppmfade -l teapot.ppm

Fade from black to teapot.ppm generating fade.0001.ppm to fade.0030.ppm.

ppmfade -f teapot.ppm -base end

Fade from teapot.ppm to black generating end.0001.ppm to end.0030.ppm.

SEE ALSO

tontsc manual, **sgifade** manual, **smart_vfr** manual, **xmorph** manual, **ppm**(1)

AUTHOR

Bryan Henderson, Olympia WA; April 2000

Inspired by and intended as a replacement for **pbmfade** (not a Netpbm program) by Wesley C. Barris.

Table Of Contents

NAME

ppmflash - brighten a picture to approach white

SYNOPSIS

ppmflash *flashfactor* [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmflash reads a PPM image as input. It changes the color of each pixel to bring it a specified amount closer to white. It generates a PPM image of the result.

flashfactor is a real number between 0 and 1, inclusive. **ppmflash** increases the intensity of each RGB component by the fraction *flashfactor* of the difference between the current value and full intensity. So if a pixel contains 60% full red, 10% full green, and no blue and you specify 0.5 (half), **ppmflash** increases the red to 80% (because it was 40% from full intensity, so it adds half of 40% to the original 60%), the green to 55%, and the blue to 50%.

If *flashfactor* is zero, the output is identical to the input. If *flashfactor* is one, the output is all white.

ppmbrighten does a more normal kind of brightening. **pamfunc** does a very simple brightening. Both **ppmbrighten** and **pamfunc** can reduce brightness as well.

pnmgamma is another way people do a similar brightening, though it isn't really intended for that.

SEE ALSO

ppmbrighten(1) **pamfunc**(1), **pnmgamma**(1), **ppm**(1),

AUTHOR

Table Of Contents

NAME

ppmforge - fractal forgeries of clouds, planets, and starry skies

SYNOPSIS

ppmforge

[**-clouds**] [**-night**] [**-dimension** *dimen*] [**-hour** *hour*] [**-inclination**|**-tilt** *angle*] [**-mesh** *size*] [**-power** *factor*] [**-glaciers** *level*] [**-ice** *level*] [**-saturation** *sat*] [**-seed** *seed*] [**-stars** *fraction*] [{**-xsize**|**-width**} *width*] [{**-ysize**|**-height**} *height*]

You can abbreviate any options to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmforge generates three kinds of ''random fractal forgeries,'' the term coined by Richard F. Voss of the IBM Thomas J. Watson Research Center for seemingly realistic pictures of natural objects generated by simple algorithms embodying randomness and fractal self-similarity. The techniques used by **ppmforge** are essentially those given by Voss[1], particularly the technique of spectral synthesis explained in more detail by Dietmar Saupe[2].

The program generates two varieties of pictures: planets and clouds, which are just different renderings of data generated in an identical manner, illustrating the unity of the fractal structure of these very different objects. A third type of picture, a starry sky, is synthesised directly from pseudorandom numbers.

The generation of planets or clouds begins with the preparation of an array of random data in the frequency domain. The size of this array, the ''mesh size,'' can be set with the **-mesh** option; the larger the mesh the more realistic the pictures but the calculation time and memory requirement increases as the square of the mesh size. The fractal dimension, which you can specify with the **-dimension** option, determines the roughness of the terrain on the planet or the scale of detail in the clouds. As the fractal dimension is increased, more high frequency components are added into the random mesh.

Once the mesh is generated, an inverse two dimensional Fourier transform is performed upon it. This converts the original random frequency domain data into spatial amplitudes. We scale the real components that result from the Fourier transform into numbers from 0 to 1 associated with each point on the mesh. You can further modify this number by applying a ''power law scale'' to it with the **-power** option. Unity scale leaves the numbers unmodified; a power scale of 0.5 takes the square root of the numbers in the mesh, while a power scale of 3 replaces the numbers in the mesh with their cubes. Power law scaling is best envisioned by thinking of the data as representing the elevation of terrain; powers less than 1 yield landscapes with vertical scarps that look like glacially-carved valleys; powers greater than one make fairy-castle spires (which require large mesh sizes and high resolution for best results).

After these calculations, we have a array of the specified size containing numbers that range from 0 to 1. The pixmaps are generated as follows:

Clouds A color map is created that ranges from pure blue to white by increasing admixture (desaturation) of blue with white. Numbers less than 0.5 are colored blue, numbers between 0.5 and 1.0 are colored with corresponding levels of white, with 1.0 being pure white.

Planet The mesh is projected onto a sphere. Values less than 0.5 are treated as water and values between 0.5 and 1.0 as land. The water areas are colored based upon the water depth, and land based on its elevation. The random depth data are used to create clouds over the oceans. An atmosphere approximately like the Earth's is simulated; its light absorption is calculated to create a blue cast around the limb of the planet. A function that rises from 0 to 1 based on latitude is modulated by the local elevation to generate polar ice caps--high altitude terrain

carries glaciers farther from the pole. Based on the position of the star with respect to the observer, the apparent color of each pixel of the planet is calculated by ray-tracing from the star to the planet to the observer and applying a lighting model that sums ambient light and diffuse reflection (for most planets ambient light is zero, as their primary star is the only source of illumination). Additional random data are used to generate stars around the planet.

Night A sequence of pseudorandom numbers is used to generate stars with a user specified density.

Cloud pictures always contain 256 or fewer colors and may be displayed on most color mapped devices without further processing. Planet pictures often contain tens of thousands of colors which must be compressed with **pnmquant** or **ppmdither** before encoding in a color mapped format. If the display resolution is high enough, **ppmdither** generally produces better looking planets. **pnmquant** tends to create discrete color bands, particularly in the oceans, which are unrealistic and distracting. The number of colors in starry sky pictures generated with the **-night** option depends on the value specified for **-saturation**. Small values limit the color temperature distribution of the stars and reduce the number of colors in the image. If the **-saturation** is set to 0, none of the stars will be colored and the resulting image will never contain more than 256 colors. Night sky pictures with many different star colors often look best when color compressed by **pnmdepth** rather than **pnmquant** or **ppmdither**. Try *newmaxval* settings of 63, 31, or 15 with **pnmdepth** to reduce the number of colors in the picture to 256 or fewer.

OPTIONS

-clouds Generate clouds. A pixmap of fractal clouds is generated. Selecting clouds sets the default for fractal dimension to 2.15 and power scale factor to 0.75.

-dimension *dimen*

Sets the fractal dimension to the specified *dimen*, which may be any floating point value between 0 and 3. Higher fractal dimensions create more ‘‘chaotic’’ images, which require higher resolution output and a larger FFT mesh size to look good. If no dimension is specified, 2.4 is used when generating planets and 2.15 for clouds.

-glaciers *level*

The floating point *level* setting controls the extent to which terrain elevation causes ice to appear at lower latitudes. The default value of 0.75 makes the polar caps extend toward the equator across high terrain and forms glaciers in the highest mountains, as on Earth. Higher values make ice sheets that cover more and more of the land surface, simulating planets in the midst of an ice age. Lower values tend to be boring, resulting in unrealistic geometrically-precise ice cap boundaries.

-hour *hour*

When generating a planet, **ppmforge** uses *hour* as the ’hour angle at the central meridian.’ If you specify **-hour 12**, for example, the planet will be fully illuminated, corresponding to high noon at the longitude at the center of the screen. You can specify any floating point value between 0 and 24 for *hour*, but values which place most of the planet in darkness (0 to 4 and 20 to 24) result in crescents which, while pretty, don’t give you many illuminated pixels for the amount of computing that’s required. If no **-hour** option is specified, a random hour angle is chosen, biased so that only 25% of the images generated will be crescents.

-ice *level*

Sets the extent of the polar ice caps to the given floating point *level*. The default level of 0.4 produces ice caps similar to those of the Earth. Smaller values reduce the amount of ice, while larger **-ice** settings create more prominent ice caps. Sufficiently large values, such as 100 or more, in conjunction with small settings for **-glaciers** (try 0.1) create ’ice balls’ like Europa.

-inclination|-tilt *angle*

The inclination angle of the planet with regard to its primary star is set to *angle*, which can be any floating point value from -90 to 90. The inclination angle can be thought of as specifying, in degrees, the ‘‘season’’ the planet is presently experiencing or, more precisely, the latitude at which the star transits the zenith at local noon. If 0, the planet is at equinox; the star is directly overhead at the equator. Positive values represent summer in the northern hemisphere, negative values summer in the southern hemisphere. The Earth’s inclination angle, for example, is about 23.5 at the June solstice, 0 at the equinoxes in March and September, and -23.5 at the December solstice. If no inclination angle is specified, a random value between -21.6 and 21.6 degrees is chosen.

-mesh *size*

A mesh of *size* by *size* will be used for the fast Fourier transform (FFT). Note that memory requirements and computation speed increase as the square of *size*; if you double the mesh size, the program will use four times the memory and run four times as long. The default mesh is 256x256, which produces reasonably good looking pictures while using half a megabyte for the 256x256 array of single precision complex numbers required by the FFT. On machines with limited memory capacity, you may have to reduce the mesh size to avoid running out of RAM. Increasing the mesh size produces better looking pictures; the difference becomes particularly noticeable when generating high resolution images with relatively high fractal dimensions (between 2.2 and 3).

-night A starry sky is generated. The stars are created by the same algorithm used for the stars that surround planet pictures, but the output consists exclusively of stars.

-power *factor*

Sets the ’power factor’ used to scale elevations synthesised from the FFT to *factor*, which can be any floating point number greater than zero. If no factor is specified a default of 1.2 is used if a planet is being generated, or 0.75 if clouds are selected by the **-clouds** option. The result of the FFT image synthesis is an array of elevation values between 0 and 1. A non-unity power factor exponentiates each of these elevations to the specified power. For example, a power factor of 2 squares each value, while a power factor of 0.5 replaces each with its square root. (Note that exponentiating values between 0 and 1 yields values that remain within that range.) Power factors less than 1 emphasise large-scale elevation changes at the expense of small variations. Power factors greater than 1 increase the roughness of the terrain and, like high fractal dimensions, may require a larger FFT mesh size and/or higher screen resolution to look good.

-saturation *sat*

Controls the degree of color saturation of the stars that surround planet pictures and fill starry skies created with the **-night** option. The default value of 125 creates stars which resemble the sky as seen by the human eye from Earth’s surface. Stars are dim; only the brightest activate the cones in the human retina, causing color to be perceived. Higher values of *sat* approximate the appearance of stars from Earth orbit, where better dark adaptation, absence of skyglow, and the concentration of light from a given star onto a smaller area of the retina thanks to the lack of atmospheric turbulence enhances the perception of color. Values greater than 250 create ‘‘science fiction’’ skies that, while pretty, don’t occur in this universe.

Thanks to the inverse square law combined with Nature’s love of mediocrity, there are many, many dim stars for every bright one. This population relationship is accurately reflected in the skies created by **ppmforge**. Dim, low mass stars live much longer than bright massive stars, consequently there are many reddish stars for every blue giant. This relationship is preserved by **ppmforge**. You can reverse the proportion, simulating the sky as seen in a starburst galaxy, by specifying a negative *sat* value.

-seed *num*

Sets the seed for the random number generator to the integer *num*. The seed used to create each picture is displayed on standard output (unless suppressed with the **-quiet** option). Pictures generated with the same seed will be identical. If no **-seed** is specified, a random seed derived from the date and time will be chosen. Specifying an explicit seed allows you to re-render a picture you particularly like at a higher resolution or with different viewing parameters.

-stars *fraction*

Specifies the percentage of pixels, in tenths of a percent, which will appear as stars, either surrounding a planet or filling the entire frame if **-night** is specified. The default *fraction* is 100.

-xsize|-width *width*

Sets the width of the generated image to *width* pixels. The default width is 256 pixels. Images must be at least as wide as they are high; if a width less than the height is specified, it will be increased to equal the height. If you must have a long skinny pixmap, make a square one with **ppmforge**, then use **pamcut** to extract a portion of the shape and size you require.

-ysize|-height *height*

Sets the height of the generated image to *height* pixels. The default height is 256 pixels. If the height specified exceeds the width, the width will be increased to equal the height.

LIMITATIONS

The algorithms require the output pixmap to be at least as wide as it is high, and the width to be an even number of pixels. These constraints are enforced by increasing the size of the requested pixmap if necessary.

You may have to reduce the FFT mesh size on machines with 16 bit integers and segmented pointer architectures.

SEE ALSO

pamcut(1), **pnmdepth**(1), **ppmdither**(1), **pnmquant**(1), **ppm**(1)

[1] Voss, Richard F., ''Random Fractal Forgeries,'' in Earnshaw et. al., Fundamental Algorithms for Computer Graphics, Berlin: Springer-Verlag, 1985.

[2] Peitgen, H.-O., and Saupe, D. eds., The Science Of Fractal Images, New York: Springer Verlag, 1988.

AUTHOR

John Walker
Autodesk SA
Avenue des Champs-Montants 14b
CH-2074 MARIN
Suisse/Schweiz/Svizzera/Svizra/Switzerland
Usenet:*kelvin@Autodesk.com*
Fax:038/33 88 15
Voice:038/33 76 33

PLUGWARE!

If you like this kind of stuff, you may also enjoy ‘‘James Gleick’s Chaos--The Software’’ for MS-DOS, available for $59.95 from your local software store or directly from Autodesk, Inc., Attn: Science Series, 2320 Marinship Way, Sausalito, CA 94965, USA. Telephone: (800) 688-2344 toll-free or, outside the U.S. (415) 332-2344 Ext 4886. Fax: (415) 289-4718. ‘‘Chaos--The Software’’ includes a more comprehensive fractal forgery generator which creates three-dimensional landscapes as well as clouds and planets, plus five more modules which explore other aspects of Chaos. The user guide of more than 200 pages includes an introduction by James Gleick and detailed explanations by Rudy Rucker of the mathematics and algorithms used by each program.

Table Of Contents

NAME

ppmglobe - generate strips to glue onto a sphere

SYNOPSIS

ppmglobe *stripcount*

DESCRIPTION

This program is part of **Netpbm**(1).

ppmglobe does the inverse of a cylindrical projection of a sphere. Starting with a cylindrical projection, it produces an image you can cut up and glue onto a sphere to obtain the spherical image of which it is the cylindrical projection.

What is a cylindrical projection? Imagine a map of the Earth on flat paper. There are lots of different ways cartographers show the three dimensional information in such a two dimensional map. The cylindrical projection is one. You could make a cylindrical projection by putting a light inside a globe and wrapping a rectangular sheet of paper around the globe, touching the globe at the Equator. Then trace the image that the light projects onto the paper. Lay the paper out flat and you have a cylindrical projection.

Here's where **ppmglobe** comes in: Pass the image on that paper through **ppmglobe** and what comes out the other side looks something like this:

Example of map of the earth run through ppmglobe

You could cut out the strips and glue it onto a sphere and you'd have a copy of the original globe.

Note that cylindrical projections are not what you normally see as maps of the Earth. You're more likely to see a Mercator projection. In the Mercator projection, the Earth gets stretched North-South as well as East-West as you move away from the Equator. It was invented for use in navigation, because you can draw straight compass courses on it, but is used to day because it is pretty.

You can find maps of planets at maps.jpl.nasa.gov .

PARAMETERS

stripcount is the number of strips **ppmglobe** is to generate in the output. More strips makes it easier to fit onto a sphere (less stretching, tearing, and crumpling of paper), but makes you do more cutting out of the strips.

SEE ALSO

ppm(1)

HISTORY

ppmglobe was new in Netpbm 10.16 (June 2003).

It is derived from
Max Gensthaler's **ppmglobemap** .

AUTHORS

Max Gensthaler wrote a program he called **ppmglobemap** in June 2003 and suggested it for inclusion in Netpbm. Bryan Henderson modified the code slightly and included it in Netpbm as **ppmglobe**.

Table Of Contents

NAME

ppmhist - print a histogram of the colors in a PPM image

SYNOPSIS

ppmhist [**-hexcolor** | **-float** | **-colorname** | **-map**] [**-nomap**] [**-noheader**] [**-sort=**{**frequency**,**rgb**}] [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmhistreads a PPM image as input and generates a histogram of the colors in the image, i.e. a list of all the colors and how many pixels of each color are in the image.

Output Format

The output is one line for each color in the input image.

By default, there are two lines of column header at the top. Use the **-noheader** option to suppress those lines.

In each line, **ppmhist** identifies the color by red, green, and blue components. By default, it lists each of these in decimal, using the exact values that are in the PPM input. So if the image has a maxval of 255, the numbers in the listing range from 0 to 255. With the **-hexcolor** option, you can change these numbers to hexadecimal. With the **-float** option, the numbers are fractional, adjusted to a maxval of 1.

Each line lists the luminosity of the color. It is in decimal on the same scale as the rgb values (see above).

Each line lists the number of pixels in the image that have the color. This is in decimal.

OPTIONS

-sort={**frequency**,**rgb**}

The **-sort** option determines the order in which the colors are listed in the output. **frequency** means to list them in order of how pixels in the input image have the color, with the most represented colors first. **rgb** means to sort them first by the intensity of the red component of the color, the of the green, then of the blue, with the least intense first.

The default is **frequency**.

-hexcolor

Print the color components in hexadecimal. See output format .

You may not specify this option along with **-float** or **map**.

-float Print the color components and the luminosity as floating point numbers in the range [0,1]. See output format .

You may not specify this option along with **-hexcolor** or **map**.

This option was added in Netpbm 10.19 (November 2003).

-map Generates a PPM file of the colormap for the image, with the color histogram as comments. See output format .

You may not specify this option along with **-float** or **hexcolor**.

-nomap

Generates the histogram for human reading. This is the default.

-colorname

Add the color name to the output. This is the name from the system color dictionary . If the exact color is not in the color dictionary, it is the closest color that is in the dictionary and is preceded by a '*'. If you don't have a system color dictionary, the program fails.

This option was added in Netpbm 10.10 (October 2002).

-noheader

Do not print the column headings.

SEE ALSO

ppm(1), **pgmhist**(1), **pnmcolormmap**(1), **pnmhistmap**(1), **ppmchange**(1)

AUTHOR

Table Of Contents

NAME

ppmlabel - add text to a PPM image

SYNOPSIS

ppmlabel

[**-angle** *angle*]

[**-background** { **transparent** | *color* }]

[**-color** *color*]

[**-file** *filename*]

[**-size**

textsize]

[**-text** *text_string*]

[**-x** *column*]

[**-y** *row*]

...

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmlabel uses the text drawing facilities of **ppmdraw** to add text to a PBM image. You control the location, size, baseline angle, color of the text, and background color (if any) with command line arguments. You can specify the text on the command line or supply it in files.

You can add any number of separate labels in a single invocation of **ppmlabel**, limited only by any restrictions your environment has on the number and size of program arguments (e.g. a shell's command size limit).

If you don't specify *ppmfile*, **ppmdraw** reads its input PPM image from Standard Input.

The output image goes to Standard Output.

A more sophisticated way to add a label to an image is to use **pbmtext** or **pbmtextps** to create an image of the text, then **pamcomp** to overlay it onto the base image.

OPTIONS

The arguments on the **ppmlabel** command line are not options in the strict sense; they are commands which control the placement and appearance of the text being added to the input pixmap. They are executed left to right, and any number of arguments may appear.

You can abbreviate any "option" to its shortest unique prefix.

-angle *angle*

This option sets the angle of the baseline of subsequent text. *angle* is an integral number of degrees, measured counterclockwise from the row axis of the image.

-background { **transparent** | *color* }

If the argument is **transparent**, **ppmlabel** draws the text over the existing pixels in the pixmap. If you specify a *color* (see the **-color** option below for information on how to specify colors), **ppmlabel** generates background rectangles enclosing subsequent text, and those rectangles are filled with that color.

-color *color*

This option sets the color for subsequent text.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

-colour is an acceptable alternate spelling.

-file *filename*

This option causes **ppmlabel** to read lines of text from the file named *filename* and draw it on successive lines.

-size *textsize*

This option sets the height of the tallest characters above the baseline to *textsize* pixels.

-text *text_string*

This option causes **ppmlabel** to draw the specified text string. It advances the location for subsequent text by 1.75 times the current *textsize*, which allows drawing multiple lines of text in a reasonable manner without specifying the position of each line.

Note that if you invoke **ppmlabel** via a shell command and your text string contains spaces, you'll have to quote it so the shell treats the whole string as a single token. E.g.

```
$ ppmlabel -text "this is my text" baseimage.ppm >annotatedimage.ppm
```

-x *column*

This option sets the pixel column at which subsequent text will be left justified. Depending on the shape of the first character, the actual text may begin a few pixels to the right of this point.

-y *row* This option sets the pixel row which will form the baseline of subsequent text. Characters with descenders, such as "y," will extend below this line.

LIMITATIONS

Text strings are restricted to 7 bit ASCII. The text font used by **ppmdraw** doesn't include definitions for 8 bit ISO 8859/1 characters.

When drawing multiple lines of text with a non-transparent background, it should probably fill the space between the lines with the background color. This is tricky to get right when the text is rotated to a non-orthogonal angle.

SEE ALSO

ppmmake(1), **pbmtext**(1), **pbmtextps**(1), **pamcomp**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmmake - create a PPM image of a specified color and dimensions

SYNOPSIS

ppmmake -maxval=*maxval color width height*

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmmake produces a PPM image of the specified color, width, height, and maxval.

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

EXAMPLES

```
ppmmake red 50 50
ppmmake rgb:ff/80/80 50 100 -maxval=5
```

OPTIONS

-maxval=*maxval*

The maxval for the generated image. Default is 255.

This option did not exist before June 2002. Before, the maxval was always 255.

SEE ALSO

pbmmake(1), **ppmpat**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmmix - blend together two PPM images

SYNOPSIS

ppmmix *fadefactor*

ppmfile1

ppmfile2

DESCRIPTION

This program is part of **Netpbm**(1).

ppmmix reads two PPM images as input and mixes them together using the specified fade factor. The fade factor may be in the range from 0.0 (only ppmfile1's image data) to 1.0 (only ppmfile2's image data). Anything in between specifies a smooth blend between the two images.

The two pixmaps must have the same size.

pamcomp is a more general alternative. It allows you to mix images of different size and to have the fade factor vary throughout the image (through the use of an alpha mask).

SEE ALSO

pamcomp(1), **ppm**(1)

AUTHOR

NAME

ppmnorm - replaced by pnmnorm

DESCRIPTION

This program is part of **Netpbm**(1).

ppmnorm was replaced in Netpbm 9.25 (March 2002) by **pnmnorm**(1).

pnmnorm is backward compatible with **ppmnorm** except that for PBM and PGM input, it produced PBM and PGM output.

Table Of Contents

NAME

ppmntsc - Make RGB colors legal for NTSC or PAL color systems.

SYNOPSIS

ppmntsc

[**--pal**] [**--legalonly**] [**--illegalonly**] [**--correctedonly**] [**--verbose**] [**--debug**] [*infile*]

Minimum unique abbreviations of options are acceptable.

DESCRIPTION

This program is part of **Netpbm**(1).

This program makes colors legal in the NTSC (or PAL) color systems. Often, images generated on the computer are made for use in movies which ultimately end up on video tape. However, the range of colors (as specified by their RGB values) on a computer does not match the range of colors that can be represented using the NTSC (or PAL) systems. If an image with 'illegal' colors is sent directly to an NTSC (or PAL) video system for recording, the 'illegal' colors will be clipped. This may result in an undesirable looking picture.

This utility tests each pixel in an image to see if it falls within the legal NTSC (or PAL) range. If not, it raises or lowers the pixel's saturation in the output so that it does fall within legal limits. Pixels that are already OK just go unmodified into the output.

Input is from the file named *input*. If *input* is -, input is from Standard Input. If you don't specify *input*, input is from Standard Input.

Output is always to Standard Output.

This program handles multi-image PPM input, producing multi-image PPM output.

OPTIONS

--pal Use the PAL transform instead of the default NTSC.

--verbose

Print a grand total of the number of illegal pixels.

--debug

Produce a humongous listing of illegal colors and their legal counterparts. NOTE: This option may produce a great deal of output.

--legalonly

Output only pixels that are already legal. Output black in place of pixels that are not.

--illegalonly

Output only pixels that are illegal (and output them uncorrected). Output black in place of pixels that are already legal.

--correctedonly

Output only pixels that are corrected versions of illegal pixels. Output black in place of pixels that are already legal.

SEE ALSO

pnmdepth(1), **ppmdim**(1), **ppmbrighten**(1), **ppm**(1)

AUTHOR

Wes Barris, Minnesota Supercomputer Center, Inc., Bryan Henderson

Table Of Contents

NAME

ppmpat - make a pretty PPM image

SYNOPSIS

ppmpat {**-gingham2**|**-g2**} | {**-gingham3**|**-g3**} | **-madras** | **-tartan** | **-poles** | **-squig** | **-camo** | **-anticamo** *width height*

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmpat produces a PPM of the specified width and height, with a pattern in it.

This program is mainly to demonstrate use of the ppmdraw routines, a simple but powerful drawing library. See the ppmdraw.h include file for more info on using these routines. Still, some of the patterns can be rather pretty. If you have a color workstation, something like **ppmpat -squig 300 300 | pnmquant 128** should generate a nice background.

Some of these patterns have large numbers of colors, so if you want a simpler pattern, use **pnmquant** on the output.

OPTIONS

The options specify various pattern types:

-gingham2
A gingham check pattern. Can be tiled.

-gingham3
A slightly more complicated gingham. Can be tiled.

-madras
A madras plaid. Can be tiled.

-tartan A tartan plaid. Can be tiled.

-poles Color gradients centered on randomly-placed poles.

-squig Squiggley tubular pattern. Can be tiled.

-camo Camouflage pattern.

-anticamo
Anti-camouflage pattern - like -camo, but ultra-bright colors.

REFERENCES

Some of the patterns are from ’Designer’s Guide to Color 3’ by Jeanne Allen.

SEE ALSO

pnmtile(1), **pnmquant**(1), **ppmmake**(1), **ppmrainbow**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmquant - quantize the colors in a PPM image down to a specified number

SYNOPSIS

ppmquant [**-floyd**|**-fs**] *ncolors* [*ppmfile*] **ppmquant** [**-floyd**|**-fs**] [**-nofloyd**|**-nofs**] **-mapfile** *mapfile* [*ppmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmquant is obsolete. All it does now is invoke **pnmquant** or **pnmremap**. You should use one of those programs in any new program, or if you are modifying an old program, and your program does not have to work with a version of Netpbm before 9.21 (January 2001). **ppmquant** exists only for name compatibility.

pnmquant is fully backward compatible with **ppmquant** *without* the **-mapfile** option; **pnmremap** is fully backward compatible with **ppmquant** *with* the **-mapfile** option.

Except with differences suggested by the syntax synopsis above, **ppmquant**'s function is the same as **pnmquant** and **pnmremap**.

Before Netpbm 10.19 (November 2003), **ppmquant** was a completely separate program from **pnmquant**, and was a bona fide PPM program. That means if you gave it a PGM or PBM image as input, it would process it as if it were PPM and generate a PPM output. Now, since it is really a PNM program, it processes PBM and PGM inputs as what they are and produces the same kind of output.

Note: The reason **ppmquant** was changed in Netpbm 10.19 is that for some time before that, **ppmquant** had a serious bug that would have been difficult to fix -- it chose the wrong color set. Maintaining two versions of the same code did not make sense.

SEE ALSO

pnmquant(1), **pnmremap**(1), **pnmcolormap**(1), **pamseq**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmquantall - run ppmquant on a bunch of files all at once, so they share a common colormap

SYNOPSIS

ppmquantall

[**-ext** *extension*]

ncolors

ppmfile ...

DESCRIPTION

This program is part of **Netpbm**(1).

ppmquantall takes a bunch of PPM images as input, chooses *ncolors* colors to best represent all of the images, maps the existing colors to the new ones, and **overwrites the input files** with the new quantized versions.

If you don't want to overwrite your input files, use the **-ext** option. The output files are then named the same as the input files, plus a period and the extension text you specify.

Verbose explanation: Let's say you've got a dozen PPMs that you want to display on the screen all at the same time. Your screen can only display 256 different colors, but the PPMs have a total of a thousand or so different colors. For a single PPM you solve this problem with **pnmquant**; **ppmquantall** solves it for multiple PPMs. All it does is concatenate them together into one big PPM, run **pnmquant** on that, and then split it up into little PPMs again.

(Note that another way to solve this problem is to pre-select a set of colors and then use **pnmremap** to separately quantize each PPM to that set.)

SEE ALSO

pnmquant(1), **pnmremap**(1), **pnmcolormap**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmrainbow - Generate a rainbow

SYNOPSIS

ppmrainbow

[**-width=***number*]

[**-height=***number*] [**-tmpdir=***directory*]

[**-norepeat**]

[**-verbose**]

color ...

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmrainbow generates a PPM image that fades from one color to another to another from left to right, like a rainbow. The colors are those you specify on the command line, in that order. The first color is added again on the right end of the image unless you specify the **-norepeat** option.

If you want a vertical or other non-horizontal rainbow, run the output through **pnmrotate** or **pamflip**.

One use for such a rainbow is to compose it with another image under an alpha mask in order to add a rainbow area to another image. In fact, you can make rainbow-colored text by using **pbmtext**, **pamcomp**, and **ppmrainbow**.

OPTIONS

-width *number*

The width in pixels of the output image.

Default is 600.

-height *number*

The height in pixels of the output image.

Default is 8.

-norepeat

This option makes **ppmrainbow** end the rainbow with the last color you specify. Without this option, **ppmrainbow** adds the first color you specify to the right end of the rainbow as if you had repeated it. *(I don't understand the point of this default behavior; it exists today just for backward compatibility).*

-tmpdir

The directory specification of the directory **ppmrainbow** is to use for temporary files.

Default is the value of the **TMPDIR** environment variable, or **/tmp** if **TMPDIR** is not set.

-verbose

Print the 'commands' (invocations of other Netpbm programs) that **ppmrainbow** uses to create the image.

SEE ALSO

ppmmake(1), **pamcomp**(1), **pbmtext**(1), **ppmfade**(1), **pnmrotate**(1), **pamflip**(1), **ppm**(1).

AUTHOR

Arjen Bax wrote **ppmrainbow** in June 2001 and contributed it to the Netpbm package. Bryan Henderson wrote this manual in July 2001.

Table Of Contents

NAME

ppmrelief - run a Laplacian relief filter on a PPM image

SYNOPSIS

ppmrelief

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmrelief reads a PPM image as input, does a Laplacian relief filter, and writes a PPM image as output.

The Laplacian relief filter is described in 'Beyond Photography' by Holzmann, equation 3.19. It's a sort of edge-detection.

SEE ALSO

pgmbentley(1), **pgmoil**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmrough - create PPM image of two colors with a ragged border between them

SYNOPSIS

ppmrough

[**-left** *pixels*]

[**-right** *pixels*]

[**-top** *pixels*]

[**-bottom** *pixels*]

[**-width** *pixels*]

[**-height** *pixels*]

[**-bg** *rgb:##/##/##*]

[**-fg** *rgb:##/##/##*]

[**-var** *pixels*]

[**-init** *seed*]

[**-verbose**]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmrough generates a PPM image of the specified width, height, and colors. **ppmrough** tiles the image into semi-rectangular regions with a ragged borders between them. It calculates the fluctuations with the **rand()** standard C library function.

ppmrough writes the PPM image to Standard Output.

The maxval of the output image is 255 (You can change this with **pnmdepth**).

Use the options **-left** or **-right**, respectively, to make vertical borders, and **-top** or **-bottom**, respectively, to generate horizontal borders inside the image. Each of these options needs an integer value *pixels* that determines the average distance of the interior border to the related edge of the image. You may combine the **-left**, **-right**, **-top**, and **-bottom** options to generate an image with more than one border. The algorithm ensures that you can concatenate two images produced with the same (i.e. **-left**) value without dislocations.

You specify the dimensions of the generated image with the **-width** and **-height** options.

Use the **-bg** and **-fg** options to set the background (margin) color and the foreground (interior) color, respectively. If you don't specify any of the **-left**, **-right**, **-top**, and **-bottom** options, all pixels are set to foreground color. The defaults are white foreground and black background.

Use the **-var** option to control the 'raggedness' of the border. The less its value is the smoother the border is. You can initialize the pseudo-random generator with the **-init** option.

You could use **ppmrough** with **ppmtopgm** to create a PGM alpha mask and use it to roughen up the edges of another image.

OPTIONS

-left *pixels*

Specifies the mean distance of the border from the left margin (default: no border).

-right *pixels*

Specifies the mean distance of the border from the right margin (default: no border).

-top *pixels*

Specifies the mean distance of the border from the top margin (default: no border).

-bottom *pixels*

Specifies the mean distance of the border from the bottom margin (default: no border).

-width *pixels*

Specifies the width of the image (default: 100).

-height *pixels*

Specifies the height of the image (default: 100).

-bg *color*

Background color. Specify this the same way you specify a color with **ppmmake**. Default is black.

-fg *color*

Foreground color. Specify this the same way you specify a color with **ppmmake**. Default is white.

-var *pixels*

Specifies the variance of the ragged border (default: 10). Must be a positive integer. Set *pixels* to 1 to get a straight border.

-init *seed*

Use this option to initialize the pseudo-random number generator (the Standard C library **rand()** function) with *seed*.

-verbose

Run **ppmrough** in verbose mode. It reports all parameters on Standard Error.

SEE ALSO

ppmmake(1), **pnmcat**(1), **ppmtopgm**(1), **ppm**(1),

HISTORY

This program was added to Netpbm in Release 10.9 (September 2002).

AUTHOR

 Table Of Contents

NAME

ppmshadow - add simulated shadows to a PPM image

SYNOPSIS

ppmshadow [**-b** *blur_size*] [**-k**] [**-t**] [**-x** *xoffset*] [**-y** *yoffset*] [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmshadow adds a simulated shadow to an image, giving the appearance that the contents of the image float above the page, casting a diffuse shadow on the background. Shadows can either be black, as cast by opaque objects, or translucent, where the shadow takes on the color of the object which casts it. You can specify the crispness of the shadow and its displacement from the image with command line options.

ppmshadow sees your image as a foreground on a background. The background color is whatever color the top left pixel of your image is. The background is all the pixels that are that color and the foreground is everything else. The shadow that **ppmshadow** generates is a shadow of the foreground, cast on the background.

The shadow is the same size as the foreground, plus some fringes as determined by the **-b** option. It is truncated to fit in your image. The output image is the same dimensions as the input image.

You can use **pamcomp** to place a foreground image over a background before running **ppmshadow** on it. You can use **ppmmake** to make the background image (just an image of a solid color).

OPTIONS

-b *blur_size*

Sets the distance of the light source from the image. Larger values move the light source closer, casting a more diffuse shadow, while smaller settings move the light further away, yielding a sharper shadow. *blur_size* is the number of pixels of fringe there is on the shadow, beyond where the shadow would be if there were no blurring.

The default is 11 pixels.

Note that this option controls only the fringing effect of moving the light source closer to the object. It does not make the shadow grow or shrink as would happpen in the real world if you moved a point light source closer to and further from an object.

-k Keep the intermediate temporary image files. When debugging, these intermediate files provide many clues as to the source of an error. See below for a list of the contents of each file.

-t Consider the non-background material in the image translucent -- it casts shadows of its own color rather than a black shadow, which is default. This often results in fuzzy, difficult-to-read images but in some circumstances may look better.

-x *xoffset*

Specifies the displacement of the light source to the left of the image. Larger settings of **xoffset** displace the shadow to the right, as would be cast by a light further to the left. If not specified, the horizontal offset is half of *blur_size* (above), to the left.

-y *yoffset*

Specifies the displacement of the light source above the top of the image. Larger settings displace the shadow downward, corresponding to moving the light further above the top of the image. If you don't specify **-y**, the vertical offset defaults to the same as the horizontal offset

(above), upward.

FILES

Input is a PPM file named by the *ppmfile* command line argument; if you don't specify *ppmfile*, the input is Standard Input.

The output is a PPM file, written to Standard Output.

ppmshadow creates a number of temporary files as it executes. It creates a new directory for them, **/tmp/ppmshadow***pid*, where *pid* is the process ID of the **ppmshadow** process. If the **TMPDIR** environment variable is set, **ppmshadow** creates the directory there instead of **/tmp**.

In normal operation, **ppmshadow** deletes each temporary file as soon as it is done with it and leaves no debris around after it completes. To preserve the intermediate files for debugging, use the **-k** command line option.

The temporary files are:

infile.ppm
A copy of the input.

bgmask.ppm
Positive binary mask

convkernel.ppm
Convolution kernel for blurring shadow

blurred.ppm
Blurred, colored shadow image

blurred2.ppm
Blurred shadow image before coloring

shadow.ppm
Clipped shadow image, offset as requested

background.ppm
Blank image with background of source image

shadow.ppm
Offset shadow

fgmask.ppm
Inverse mask file

justfg.ppm
Just the foreground. Rest is black. Original image times inverse mask.

shadback.ppm
Generated shadow times positive mask

allbutfg.ppm
Everything but the foreground (foreground area is black).

LIMITATIONS

The source image must contain sufficient space on the edges in the direction in which the shadow is cast to contain the shadow -- if it doesn't some of the internal steps may fail. You can usually expand the border of a too-tightly-cropped image with **pnmmargin** before processing it with **ppmshadow**.

Black pixels and pixels with the same color as the image background don't cast a shadow. If this causes unintentional 'holes' in the shadow, fill the offending areas with a color which differs from black or the background by RGB values of 1, which will be imperceptible to the viewer. Since the comparison is exact, the modified areas will now cast shadows.

The background color of the source image (which is preserved in the output) is deemed to be the color of the pixel at the top left of the input image. If that pixel isn't part of the background, simply add a one-pixel border at the top of the image, generate the shadow image, then delete the border from it.

If something goes wrong along the way, the error messages from the various Netpbm programs **ppmshadow** calls will, in general, provide little or no clue as to where **ppmshadow** went astray. In this case, Specify the **-k** option and examine the intermediate results in the temporary files (which this option causes to be preserved). If you manually run the commands that **ppmshadow** runs on these files, you can figure out where the problem is. In problem cases where you want to manually tweak the image generation process along the way, you can keep the intermediate files with the **-k** option, modify them appropriately with an image editor, then recombine them with the steps used by the code in **ppmshadow**.

See the **ppmshadow.doc** file in the Netpbm source tree for additional details and examples of the intermediate files and debugging **ppmshadow**.

Shadows are by default black, as cast by opaque material in the image occluding white light. Use the **-t** option to simulate translucent material, where the shadow takes on the color of the object that casts it. If the contrast between the image and background is insufficient, the **-t** option may yield unattractive results which resemble simple blurring of the original image.

Because Netpbm used to have a maximum maxval of 255, which meant that the largest convolution kernel **pnmconvol** could use was 11 by 11, **ppmshadow** includes a horrid, CPU-time-burning kludge which, if a blur of greater than 11 is requested, performs an initial convolution with an 11 x 11 kernel, then calls **pnmsmooth** (which is itself a program that calls **pnmconvol** with a 3 x 3 kernel) as many times as the requested blur exceeds 11. It's ugly, but it gets the job done on those rare occasions where you need a blur greater than 11.

If you wish to generate an image at high resolution, then scale it to publication size with **pamscale** in order to eliminate jagged edges by resampling, it's best to generate the shadow in the original high resolution image, prior to scaling it down in size. If you scale first and then add the shadow, you'll get an unsightly jagged stripe between the edge of material and its shadow, due to resampled pixels intermediate between the image and background obscuring the shadow.

EXIT STATUS

ppmshadow returns status 0 if processing was completed without errors, and a nonzero Unix error code if an error prevented generation of output. Some errors may result in the script aborting, usually displaying error messages from various Netpbm components it uses, without returning a nonzero error code. When this happens, the output file will be empty, so be sure to test this if you need to know if the program succeeded.

SEE ALSO

pnm(1), **pnmmargin**(1), **pnmconvol**(1), **pamscale**(1), **pnmsmooth**(1), **ppm**(1)

AUTHOR

John Walker http://www.fourmilab.ch August 8, 1997

COPYRIGHT

Table Of Contents

NAME

ppmshift - shift lines of a PPM image left or right by a random amount

SYNOPSIS

ppmshift *shift* [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmshift reads a PPM image as input. Shifts every row of image data to the left or right by a certain amount. The *shift* parameter determines by how many pixels a row is to be shifted at most.

This is another one of those effects I intended to use for MPEG tests. Unfortunately, this program will not help me here - it creates too random patterns to be used for animations. Still, it might give interesting results on still images.

EXAMPLE

Check this out: Save your favourite model's picture from something like alt.binaries.pictures.supermodels (ok, or from any other picture source), convert it to ppm, and process it e.g. like this, assuming the picture is 800x600 pixels:

```
#take the upper half, and leave it like it is
pamcut -top=0 -width=800 -height=300 cs.ppm >upper.ppm

#take the lower half, flip it upside down, dim it and distort it a little
pamcut -top=300 -width=800 -height=300 cs.ppm |  pamflip -topbottom |  ppmdim 0.7 |  ppmshift 10 >lower

#and concatenate the two pieces
pnmcat -topbottom upper.ppm lower.ppm >newpic.ppm
```

The resulting picture looks like the image being reflected on a water surface with slight ripples.

SEE ALSO

ppm(1), **pamcut**(1), **pamflip**(1), **ppmdim**(1), **pnmcat**(1)

AUTHOR

Table Of Contents

NAME

ppmspread - displace a PPM image's pixels by a random amount

SYNOPSIS

ppmspread

amount

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmspread reads a PPM image as input and moves every pixel around a bit relative to its original position. *amount* determines by how many pixels a pixel is to be moved around at most.

Pictures processed with this filter will seem to be somewhat dissolved or unfocussed (although they appear more coarse than images processed by something like *pnmconvol*).

SEE ALSO

ppm(1), **pnmconvol**(1)

AUTHOR

Table Of Contents

NAME

ppmsvgalib - display PPM image on Linux console using Svgalib

SYNOPSIS

ppmsvgalib

[**-mode=***mode*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or an equals sign between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmsvgalib displays a PPM image on a Linux virtual console using the Svgalib facility. Svgalib is a popular means of displaying Graphics in Linux without the use of the X Window System.

If you run **ppmsvgalib** with a version of Svgalib earlier than 1.9, you must run it with CAP_SYS_RAWIO capability (on most Linux systems, that means you run it as superuser), because Svgalib uses the **ioperm()** system call to access the console hardware. Newer Svgalib has its own device driver, so you need only properly permissions on a device special file to access the console.

ppmsvgalib is not capable of using color mapped video modes. These are the old video modes that are usually called '8 bit' color modes.

ppmsvgalib is a bare displayer. It won't do any manipulation of the image and is not interactive in any way. If you want a regular interactive graphics viewer that uses Svgalib, try **zgv** (not part of Netpbm).

To exit **ppmsvgalib** while it is displaying your image, send it a SIGINTR signal (normally, this means 'hit control C').

ppmsvgalib draws a white border around the edges of the screen. It does this to help you isolate problems between the image you're displaying and the facilities you're using to display it.

(Note: if the image you're displaying reaches the edges of the screen, it will replace the white border).

ppmsvgalib places the image in the center of the screen.

If your image is too big to display in the video mode you selected, **ppmsvgalib** fails. You can use **pamcut** to cut out a part of the image to display or use **pamscale** to shrink the image to fit.

If you want to play with **ppmsvgalib**, **ppmcie** is a good way to generate a test image.

To be pedantic, we must observe that **ppmsvgalib** displays a PPM image in the correct colors only if the display has a transfer function which is the exact inverse of the gamma function that is specified in the PPM specification. Happily, most CRT displays are pretty close.

Running the PPM image through **pnmgamma** can help cause **ppmsvgalib** to display the correct colors.

OPTIONS

-mode=*mode*

This tells **ppmsvgalib** what video mode to use. *mode* is the Svgalib video mode number. You can get a list of all the video modes and their Svgalib video mode numbers with the program **vgatest** that is packaged with Svgalib. (Unfortunately, the various interesting programs that are packaged with Svgalib are typically not installed on systems that have the Svgalib library installed).

In practice, there are probably only two modes you'll ever care about: 25 is the standard SVGA direct color mode, which is 1024 columns by 768 rows with 8 bit red, green, and blue components for each pixel and no fancy options. 28 is the same, but with the popular higher resolution of 1280 x 1024.

But if you have an older video controller (with less than 4MB of memory), those modes aren't available and you might like mode 19, which is 640 x 480 and takes less than a megabyte of video memory. This is a standard VGA video mode.

SEE ALSO

pamcut(1), **pamscale**(1), **ppmcie**(1), **ppm**(1), **zgv**, **Svgalib**, **vgatest**

AUTHOR

By Bryan Henderson, January 2002.

Table Of Contents

NAME

ppmtoacad - convert PPM to Autocad database or slide

SYNOPSIS

ppmtoacad

[**-dxb**]

[**-poly**]

[**-background** *color*]

[**-white**]

[**-aspect** *ratio*]

[**-8**]

[*ppmfile*]

You may abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoacad reads a PPM image as input and produces an Autocad® slide file or binary database import (.dxb) file as output. If you don't specify *ppmfile*, **ppmtoacad** takes the input from Standard Input.

(Typographical note: the name of Autocad is often rendered as AutoCAD. Netpbm documentation uses standard American typography, wherein that is not a valid form of capitalization).

OPTIONS

-dxb **ppmtoacad** writes an Autocad binary database import (.dxb) file. You read this file with the DXBIN command and, once loaded, it becomes part of the Autocad geometrical database, so you can view and edit it like any other object. Each sequence of identical pixels becomes a separate object in the database; this can result in very large Autocad drawing files. However, if you want to trace over a bitmap, it lets you zoom and pan around the bitmap as you wish.

-poly If you don't specify the **-dxb** option, **ppmtoacad** generates an Autocad slide file. Normally each row of pixels is represented by an Autocad line entity. If you specify **-poly**, **ppmtoacad** renders the pixels as filled polygons. If you view the slide on a display with higher resolution than the source image, this will cause the pixels to expand instead of appearing as discrete lines against the screen background color. Regrettably, this representation yields slide files which occupy more storage space and take longer to display.

-background *color*

Most Autocad display drivers can be configured to use any available color as the screen background. Some users prefer a black screen background, others white, while splinter groups advocate burnt ocher, tawny puce, and shocking gray. Discarding pixels whose closest Autocad color representation is equal to the background color can substantially reduce the size of the Autocad database or slide file needed to represent a bitmap. If you don't specify **-background**, **ppmtoacad** assumes the screen background color to be black. You may specify any Autocad color number as the screen background; **ppmtoacad** assumes color numbers to specify the hues defined in the standard Autocad 256 color palette.

-white Since many Autocad users choose a white screen background, this option is provided as a short-cut. Specifying **-white** is identical in effect to **-background 7**.

-aspect *ratio*
If the source image had non-square pixels (which means it is not standard PPM), specify the ratio of the pixel width to pixel height as *ratio*. **ppmtoacad** will correct the resulting slide or .dxb file so that pixels on the Autocad screen will be square. For example, to correct an image made for a 320x200 VGA/MCGA screen, specify **-aspect 0.8333**.

-8 Restricts the colors in the output file to the 8 RGB shades.

RESTRICTIONS

Autocad has a fixed palette of 256 colors, distributed along the hue, lightness, and saturation axes. So it may poorly render images which contain many nearly-identical colors, or colors not closely approximated by Autocad's palette.

ppmtoacad works best if the system displaying its output can display the full 256 color Autocad palette. Monochrome, 8 color, and 16 color configurations will produce less than optimal results.

When creating a .dxb file or a slide file with the **-poly** option, **ppmtoacad** finds both vertical and horizontal runs of identical pixels and consolidates them into rectangular regions to reduce the size of the output file. This is effective for images with large areas of constant color but it's no substitute for true raster to vector conversion. In particular, this process does not optimize thin diagonal lines at all.

Output files can be huge.

SEE ALSO

Autocad Reference Manual: *Slide File Format* and *Binary Drawing Interchange (DXB) Files*, **ppm**(1)

AUTHOR

John Walker
Autodesk SA
Avenue des Champs-Montants 14b
CH-2074 MARIN
Suisse/Schweiz/Svizzera/Svizra/Switzerland
Usenet:*kelvin@Autodesk.com*
Fax:038/33 88 15
Voice:038/33 76 33

Table Of Contents

NAME

ppmtoarbtxt - generate image in arbitrary text format from PPM image

SYNOPSIS

ppmtoarbtxt *bodyskl* [**-hd** *headskl*] [**-tl** *tailskl*] [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoarbtxt generates simple text-based graphics formats based on format descriptions given as input. A text-based graphics format is one in which an image is represented by text (like PNM plain format, but unlike PNM raw format).

ppmtoarbtxtreads a PPM image as input. For each pixel in the image, **ppmtoarbtxt** writes the contents of the skeleton file *bodyskl*, with certain substitutions based on the value of the pixel, to stdout. The substitutions are as follows:

#(ired *format blackref whiteref***)**
: generates an integer in the range *blackref* to *whiteref* using *format* representing the red intensity of the pixel. A red intensity of 0 becomes *blackref*; a red intensity of maxval becomes *whiteref*.

  #(ired) is equivalent to **#(ired %d 0 255)**.

#(igreen *format blackref whiteref***)**
: Same as **#(ired...**, but for green.

#(iblue *format blackref whiteref***)**
: Same as **#(ired...**, but for blue.

#(ilum *format blackref whiteref***)**
: Same as **#(ired...**, but representing the luminance value (0.299*red + 0.587*green + 0.114*blue) of the pixel.

#(fred *format blackref whiteref***)**
: Same as **#(ired...**, but generates a floating point number instead of an integer.

  #(fred) is equivalent to **#(fred %f 0.0 1.0)**.

#(fgreen *format blackref whiteref***)**
: Same as **#(fred...**, but for green.

#(fblue *format blackref whiteref***)**
: Same as **#(fred...**, but for blue.

#(flum *format blackref whiteref***)**
: Same as **#(fred...**, but representing the luminance value (0.299*red + 0.587*green + 0.114*blue) of the pixel.

#(width)
Generates the width in pixels of the image.

#(height)
Generates the height in pixels of the image.

#(posx)
Generates the horizontal position of the pixel, in pixels from the left edge of the image.

#(posy)
Generates the vertical position of the pixel, in pixels from the top edge of the image.

If the skeleton file ends with a LF-character, **ppmtoarbtxt** ignores it -- it does not include it in the output.

OPTIONS

-hd *headskl*
This option causes **ppmtoarbtxt** to place the contents of the file named *headskl* at the beginning of the output, before the first pixel. It does the same substitutions as for *bodyskl*, except substitutions based on a pixel value are undefined.

-tl *tailskl*
This option causes **ppmtoarbtxt** to place the contents of the file named *tailskl* at the end of the output, after the last pixel. It is analogous to **-hd**.

EXAMPLES

gray inversion

Here we generate a PGM plain-format image with gray inversion (like **ppmtopgm | pnminvert**).

Contents of our head skeleton file:

```
P2
#(width) #(height)
255
```

Contents of our body skeleton file:

```
#(ilum %d 255 0)
```

povray file

Here we generate a povray file where each pixel is represented by a sphere at location (x,y,z) = (posx,height-posy,luminance). The color of the sphere is the color of the pixel.

Contents of our head skeleton:

```
#include 'colors.inc'
#include 'textures.inc'
camera {
  location  <#(width) * 0.6, #(height) * 0.7, 80>
  look_at   <#(width) * 0.5, #(height) * 0.5, 0>
}

light_source { <#(width) * 0.5, #(height) * 0.5, 25> color White
}
```

Contents of our body skeleton:

```
sphere { <#(posx),#(height)-#(posy),#(ilum %d 0 10)>, 0.5
  texture {
    pigment {
      color rgb <#(fred),#(fgreen),#(fblue)>
    }
    finish {
      phong 1
    }
  }
}
```

SEE ALSO

pnmtoplainpnm(1) **ppm**(1)

HISTORY

ppmtoarbtxt was added to Netpbm in Release 10.14 (March 2003). It existed under the name **ppm-totxt** since 1995.

AUTHOR

Table Of Contents

NAME

ppmtobmp - convert a PPM image into a BMP file

SYNOPSIS

ppmtobmp

[**-windows**]

[**-os2**]

[**-bpp=***bits_per_pixel*]

[*ppmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtobmp reads a PPM image as input and produces a Microsoft Windows or OS/2 BMP file as output.

OPTIONS

-windows

Tells the program to produce a Microsoft Windows BMP file. (This is the default.)

-os2 Tells the program to produce an OS/2 BMP file. (Before August 2000, this was the default).

-bpp This tells how many bits per pixel you want the BMP file to contain. Only 1, 4, 8, and 24 are possible. By default, **ppmtobmp** chooses the smallest number with which it can represent all the colors in the input image. If you specify a number too small to represent all the colors in the input image, **ppmtobmp** tells you and terminates. You can use **pnmquant** or **ppmdither** to reduce the number of colors in the image.

NOTES

To get a faithful reproduction of the input image, the maxval of the input image must be 255. If it is something else, the colors in the BMP file may be slightly different from the colors in the input.

Windows icons are not BMP files. Use **ppmtowinicon** to create those.

SEE ALSO

bmptoppm(1), **ppmtowinicon**(1), **pnmquant**(1), **ppmdither**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmtoeyuv - convert a PPM image into a Berkeley YUV file

SYNOPSIS

ppmtoeyuv

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoeyuv reads a PPM image as input and produces a Berkeley Encoder YUV (not the same as Abekas YUV) file on the Standard Output file.

With no argument, **ppmtoeyuv**takes input from Standard Input. Otherwise, *ppmfile* is the file specification of the input file.

ppmtoeyuv handles multi-image PPM input streams, outputting consecutive eyuv images. There must be at least one image, though.

SEE ALSO

eyuvtoppm(1), **ppmtoyuv**(1), **ppm**(1)

Table Of Contents

NAME

ppmtogif - convert a PPM image to a GIF image

SYNOPSIS

ppmtogif

[**-interlace**]

[**-sort**]

[**-mapfile** *mapfile*] [**-transparent**=[=]*color*]

[**-alpha**=*pgmfile*]

[**-comment**=*text*]

[**-nolzw**] [*ppmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one to designate an option. You may use either white space or equals signs between an option name and its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtogif reads a PPM iamge as input and produces a GIF file as output.

This program creates only individual GIF images. To combine multiple GIF images into an animated GIF, use **gifsicle** (not part of the Netpbm package).

ppmtogif creates either an original GIF87 format GIF file or the newer GIF89 format. It creates GIF89 when you request features that were new with GIF89, to wit the **-transparent** or **-comment** options. Otherwise, it creates GIF87. Really old GIF readers conceivably could not recognize GIF89.

OPTIONS

-interlace

Produce an interlaced GIF file.

-sort Produce a GIF file with a sorted color map.

-mapfile=*mapfile*

Use the colors found in the file *mapfile* to create the colormap in the GIF file, instead of the colors from *ppmfile*. *mapfile* can be any PPM file; all that matters is the colors in it. If the colors in *ppmfile* do not match those in *mapfile*, **ppmtogif** matches them to a 'best match.' You can obtain a much better result by using **pnmremap** to change the colors in the input to those in the map file.

The *mapfile* file is not a palette file, just an image whose colors you want to use. The order of colors in the GIF palette have nothing to do with where they appear in the *mapfile* image, and duplication of colors in the image is irrelevant.

-transparent=*color*

ppmtogif marks the specified color as transparent in the GIF image.

If you don't specify **-transparent**, **ppmtogif** does not mark any color transparent (except as

indicated by the **-alpha** option).

Specify the color (*color*) as described for the argument of the **ppm_parsecolor()** library routine .

If the color you specify is not present in the image, **ppmtogif** selects instead the color in the image that is closest to the one you specify. Closeness is measured as a cartesian distance between colors in RGB space. If multiple colors are equidistant, **ppmtogif** chooses one of them arbitrarily.

However, if you prefix your color specification with '=', e.g. **-transparent==red**, only the exact color you specify will be transparent. If that color does not appear in the image, there will be no transparency. **ppmtogif** issues an information message when this is the case.

You cannot specify both **-transparent** and **-alpha**.

-alpha=*pgmfile*

This option names a PGM file that contains an alpha mask for the image. **ppmtogif** creates fully transparent pixels wherever the alpha mask indicates transparency greater than 50%. The color of those pixels is that specified by the **-alphacolor** option, or black by default.

To do this, **ppmtogif** creates an entry in the GIF colormap in addition to the entries for colors that are actually in the image. It marks that colormap entry as transparent and uses that colormap index in the output image to create a transparent pixel.

The alpha image must be the same dimensions as the input image, but may have any maxval. White means opaque and black means transparent.

You cannot specify both **-transparent** and **-alpha**.

-alphacolor

See **-alpha**.

-comment=*text*

Include a comment in the GIF output with comment text *text*.

Without this option, there are no comments in the output.

Note that in a command shell, you'll have to use quotation marks around *text* if it contains characters (e.g. space) that would make the shell think it is multiple arguments:
$ ppmtogif -comment "this is a comment" <xxx.ppm >xxx.gif

-nolzw

This option is mainly of historical interest -- it involves use of a patent that is now expired.

This option causes the GIF output, and thus **ppmtogif**, not to use LZW (Lempel-Ziv) compression. As a result, the image file is larger and, before the patent expired, no royalties would be owed to the holder of the patent on LZW. See the section LICENSE below.

LZW is a method for combining the information from multiple pixels into a single GIF code. With the **-nolzw** option, **ppmtogif** creates one GIF code per pixel, so it is not doing any compression and not using LZW. However, any GIF decoder, whether it uses an LZW decompressor or not, will correctly decode this uncompressed format. An LZW decompressor would see this as a particular case of LZW compression.

Note that if someone uses an LZW decompressor such as the one in **giftopnm** or pretty much

any graphics display program to process the output of **ppmtogif -nolzw** , he is then using the LZW patent. But the patent holder expressed far less interest in enforcing the patent on decoding than on encoding.

SEE ALSO

giftopnm(1), **ppmquant**(1), **pngtopnm**(1),

gifsicle http://www.lcdf.org/gifsicle , **ppm**(1).

AUTHOR

Based on GIFENCOD by David Rowley <*mgardi@watdcsu.waterloo.edu*>. Lempel-Ziv compression based on 'compress'.

The non-LZW format is generated by code based on **djpeg** by the Independent Jpeg Group.

LICENSE

If you use **ppmtogif** without the **-nolzw** option, you are using a patent on the LZW compression method which is owned by Unisys. The patent has expired (in 2003 in the US and in 2004 elsewhere), so it doesn't matter. While the patent was in force, most people who used **ppmtogif** and similar programs did so without a license from Unisys to do so. Unisys typically asked $5000 for a license for trivial use of the patent. Unisys never enforced the patent against trivial users.

Rumor has it that IBM also owns or owned a patent covering **ppmtogif**.

A replacement for the GIF format that never required any patents to use is the PNG format.

Table Of Contents

NAME

ppmtoicr - convert a PPM image into NCSA ICR format

SYNOPSIS

ppmtoicr

[**-windowname** *name*]

[**-expand** *expand*]

[**-display** *display*]

[**-rle**]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoicr reads a PPM file as input. Produces an NCSA Telnet Interactive Color Raster graphic file as output.

If *ppmfile* is not supplied, **ppmtoicr** reads from Standard Input.

Interactive Color Raster (ICR) is a protocol for displaying raster graphics on workstation screens. The protocol is implemented in NCSA Telnet for the Macintosh version 2.3. The ICR protocol shares characteristics of the Tektronix graphics terminal emulation protocol. For example, escape sequences are used to control the display.

ppmtoicr will output the appropriate sequences to create a window of the dimensions of the input pixmap, create a colormap of up to 256 colors on the display, then load the picture data into the window.

Note that there is no icrtoppm tool - this transformation is one way.

OPTIONS

-windowname *name*

Output will be displayed in *name* (Default is to use *ppmfile* or 'untitled' if the input is from Standard Input).

-expand *expand*

Output will be expanded on display by factor *expand* (For example, a value of 2 will cause four pixels to be displayed for every input pixel.)

-display *display*

Output will be displayed on screen numbered *display*

-rle Use run-length encoded format for display. (This will nearly always result in a quicker display, but may skew the colormap).

EXAMPLES

To display a PPM file named **ppmfile** using the protocol:

```
ppmtoicr ppmfile
```

This will create a window named *ppmfile* on the display with the correct dimensions for *ppmfile*, create and download a colormap of up to 256 colors, and download the picture into the window. You may achieve the same effect with the following sequence:

```
ppmtoicr ppmfile > filename
cat filename
```

To display a GIF file using the protocol in a window titled after the input file, zoom the displayed image by a factor of 2, and run-length encode the data:

```
giftopnm giffile | ppmtoicr -w giffile -r -e 2
```

LIMITATIONS

The protocol uses frequent fflush() calls to speed up display. If you save the output to a file for later display via **cat**, **ppmtoicr** will draw much more slowly. In either case, increasing the blocksize limit on the display will speed up transmission substantially.

SEE ALSO

ppm(1)

NCSA Telnet for the Macintosh, University of Illinois at Urbana-Champaign (1989)

AUTHOR

Table Of Contents

NAME

ppmtoilbm - convert a PPM image into an ILBM file

SYNOPSIS

ppmtoilbm

[**-maxplanes**|**-mp** *N*]

[**-fixplanes**|**-fp** *N*]

[**-ham6**|**-ham8**]

[**-dcbits**|**-dcplanes**rgb**]**

[**-normal**|**-hamif**|**-hamforce**|**-24if**|**-24force**| **-dcif**|**-dcforce**|**-cmaponly**]

[**-ecs**|**-aga**]

[**-compress**|**-nocompress**]

[**-cmethod** *type*]

[**-map** *ppmfile*]

[**-savemem**]

[**ppmfile**]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoilbm reads a PPM image as input. Produces an ILBM file as output. **ppmtoilbm** understands the following ILBM types:

- Normal ILBMs with 1-16 planes
- Amiga HAM with 3-16 planes
- 24 bit
- Color map (BMHD + CMAP chunk only, nPlanes = 0)
- Unofficial direct color. 1-16 planes for each color component.

Chunks written: BMHD, CMAP, CAMG (only for HAM), BODY (not for colormap files) unofficial DCOL chunk for direct color ILBM.

OPTIONS

Options marked with (*) can be prefixed with a ’no’, e.g. ’-nohamif’. All options can be abbreviated to their shortest unique prefix.

-maxplanes | **-mp** *n*
(default 5, minimum 1, maximum 16) Maximum planes to write in a normal ILBM. If the pixmap does not fit into <n> planes, ppmtoilbm writes a HAM file (if -hamif is used), a 24bit file (if -24if is used) or a direct color file (if -dcif is used) or aborts with an error.

-fixplanes | **-fp** *b*
(min 1, max 16) If a normal ILBM is written, it will have exactly <n> planes.

-hambits | **-hamplanes** *n*
(default 6, min 3, max 16) Select number of planes for HAM picture. The current Amiga hardware supports 6 and 8 planes, so for now you should only use this values.

-normal
Turns off -hamif/-24if/-dcif, -hamforce/-24force/-dcforce and -cmaponly. Also sets compression type to byterun1.

This is the default.

-hamif (*)

-24if (*)

-dcif (*)
Write a HAM/24bit/direct color file if the image does not fit into <maxplanes> planes.

-hamforce (*)

-24force (*)

-dcforce (*)
Write a HAM/24bit/direct color file.

-dcbits | **-dcplanes** *r g b*
(default 5, min 1, max 16). Select number of bits for red, green & blue in a direct color ILBM.

-ecs Shortcut for: -hamplanes 6 -maxplanes 5

This is the default.

-aga Shortcut for: **-hamplanes 8 -maxplanes 8**

-ham6 Shortcut for: **-hamplanes 6 -hamforce**

-ham8 Shortcut for: **-hamplanes 8 -hamforce**

-compress (*)
This is the default. Compress the BODY chunk. The default compression method is byterun1. Compression requires building the ILBM image in memory; turning compression off allows stream-writing of the image, but the resulting file will usually be 30% to 50% larger. Another alternative is the -savemem option, this will keep memory requirements for compression at a minimum, but is very slow.

-cmethod none|byterun1
This option does the same thing as **-compress**.

-map *ppmfile*
Write a normal ILBM using the colors in <ppmfile> as the colormap. The colormap file also determines the number of planes, a **-maxplanes** or **-fixplanes** option is ignored.

-cmaponly
Write a colormap file: only BMHD and CMAP chunks, no BODY chunk, nPlanes = 0.

-savemem
See the **-compress** option.

LIMITATIONS

HAM pictures will always get a grayscale colormap; a real color selection algorithm might give better results. On the other hand, this allows row-by-row operation on HAM images, and all HAM images of the same depth (no. of planes) share a common colormap, which is useful for building HAM animations.

REFERENCES

Amiga ROM Kernel Reference Manual - Devices (3rd Ed.) Addison Wesley, ISBN 0-201-56775-X

SEE ALSO

ppm(1), **ilbmtoppm**(1)

AUTHORS

Modified October 1993 by Ingo Wilken (*Ingo.Wilken@informatik.uni-oldenburg.de*)

NAME

ppmtojpeg - replaced by pnmtojpeg

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtojpeg was replaced in Netpbm 9.19 (September 2001) by **pnmtojpeg**(1).

pnmtojpeg is backward compatible with **ppmtojpeg** except that with PGM or PBM input, it generates JPEG output in the special grayscale format.

Table Of Contents

NAME

ppmtoleaf - convert PPM image to Interleaf image format

SYNOPSIS

ppmtoleaf

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoleaf reads an Interleaf image file as input and generates a PPM image as output.

Interleaf is a now-defunct (actually purchased ca. 2000 by BroadVision) technical publishing software company.

SEE ALSO

ppm(1), **pnmquant**(1)

AUTHOR

Table Of Contents

NAME

ppmtolj - convert a PPM image to an HP LaserJet PCL 5 Color file

SYNOPSIS

ppmtolj

[**-gamma** *val*]

[**-resolution 75**|**100**|**150**|**300**|**600**]

[**-delta**]

[**-float**]

[**-noreset**]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtolj reads a PPM image as input and converts it into a color file suitable to be printed by an HP color PCL 5 printer.

OPTIONS

-delta Apply delta row compression to reduce the size of the pcl file.

-gamma *int*
Gamma correct the image using the integet parameter as a gamma (default 0).

-float Suppress positioning information. The default is to write the sequence ESC&l0E to the output.

-noreset
Don’t write the reset sequence to the begining and end of the output.

-resolution
Set the required output resolution 75|100|150|300|600

SEE ALSO

HP PCL 5 & Color Reference Guide, **pnmtopclxl.html**(1), **pbmtolj.html**(1), **ppmtopj**(1), **thinkjettopbm**(1), **ppm.html**(1)

AUTHOR

NAME

ppmtomap − see http://netpbm.sourceforge.net/doc//ppmtomap.html

DESCRIPTION

ppmtomap is part of the Netpbm package. Netpbm documentation is kept in HTML format.

Please refer to <http://netpbm.sourceforge.net/doc//ppmtomap.html>.

If that doesn't work, also try <http://netpbm.sourceforge.net> and emailing Bryan Henderson, bryanh@giraffe-data.com.

Note that making the documentation available this way was a choice of the person who installed Netpbm on this system. It is also possible to install Netpbm such that you would simply see the documentation instead of the message you are reading now.

Table Of Contents

NAME

ppmtomitsu - convert a PPM image to a Mitsubishi S340-10 file

SYNOPSIS

ppmtomitsu

[-sharpness *val*]

[**-enlarge** *val*]

[**-media** *string*]

[**-copy** *val*]

[**-dpi300**]

[**-tiny**]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtomitsu reads a PPM image as input and converts it into a format suitable to be printed by a Mitsubishi S340-10 printer, or any other Mitsubishi color sublimation printer.

The Mitsubishi S340-10 Color Sublimation printer supports 24bit color. Images of the available sizes take so long to transfer that there is a fast method, employing a lookuptable, that ppmtomitsu will use if there is a maximum of 256 colors in the pixmap. ppmtomitsu will try to position your image to the center of the paper, and will rotate your image for you if xsize is larger than ysize. If your image is larger than the media allows, ppmtomitsu will quit with an error message. (We decided that the media were too expensive to have careless users produce misprints.) Once data transmission has started, the job can't be stopped in a sane way without resetting the printer. The printer understands putting together images in the printers memory; ppmtomitsu doesn't utilize this as pnmcat etc provide the same functionality and let you view the result on-screen, too. The S340-10 is the lowest common denominator printer; for higher resolution printers there's the dpi300 option. The other printers also support higher values for enlarge eg, but I don't think that's essential enough to warrant a change in the program.

OPTIONS

-sharpness *1-4*

'sharpness' designation. Default is to use the current sharpness.

-enlarge *1-3*

Enlarge by a factor; Default is 1 (no enlarge)

-media {**A**|**A4**|**AS**|**A4S**}

Designate the media you're using. Default is 1184 x 1350, which will fit on any media. A is 1216 x 1350, A4 is 1184 x 1452, AS is 1216 x 1650 and A4S is 1184 x 1754. A warning: If you specify a different media than the printer currently has, the printer will wait until you put in the correct media or switch it off.

-copy *1-9*

The number of copies to produce. Default is 1.

-dpi300
Double the number of allowed pixels for a S3600-30 Printer in S340-10 compatibility mode. (The S3600-30 has 300 dpi).

-tiny Memory-safing, but always slow. The printer will get the data line-by-line in 24bit. It's probably a good idea to use this if your machine starts paging a lot without this option.

REFERENCES

Mitsubishi Sublimation Full Color Printer S340-10 Specifications of Parallel Interface LSP-F0232F

SEE ALSO

pnmquant(1), **pamscale**(1), **ppm**(1)

AUTHOR

 (*spz@specklec.mpifr-bonn.mpg.de*)

Table Of Contents

NAME

ppmtompeg - encode an MPEG-1 bitstream

SYNOPSIS

ppmtompeg [*options*] *parameter-file*

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtompeg produces an MPEG-1 video stream. param_file is a parameter file which includes a list of input files and other parameters. The file is described in detail below.

To understand this program, you need to understand something about the complex MPEG-1 format. One source of information about this standard format is **Introduction** to MPEG (1).

OPTIONS

The **-gop**, **-combine_gops**, **-frames**, and **-combine_frames** options are all exclusive.

-stat stat_file

This option causes **ppmtompeg** to append the statistics that it write to Standard Output to the file *stat_file* as well. The statistics use the following abbreviations: bits per block (bpb), bits per frame (bpf), seconds per frame (spf), and bits per second (bps).

These statistics include how many I, P, and B frames there were, and information about compression and quality.

-quiet *num_seconds*

causes **ppmtompeg** not to report remaining time more often than every *num_seconds* seconds (unless the time estimate rises, which will happen near the beginning of the run). A negative value tells **ppmtompeg** not to report at all. 0 is the default (reports once after each frame). Note that the time remaining is an estimate and does not take into account time to read in frames.

-realquiet

causes **ppmtompeg** to run silently, with the only screen output being errors. Particularly useful when reading input from stdin.

-no_frame_summary

This option prevents **ppmtompeg** from printing a summary line for each frame

-float_dct

forces **ppmtompeg** to use a more accurate, yet more computationally expensive version of the DCT.

-gop *gop_num*

causes **ppmtompeg** to encode only the numbered GOP (first GOP is 0). The parameter file is the same as for normal usage. The output file will be the normal output file with the suffix **.gop.***gop_num*. **ppmtompeg** does not output any sequence information.

-combine_gops

causes **ppmtompeg** simply to combine some GOP files into a single MPEG output stream. **ppmtompeg** inserts a sequence header and trailer. In this case, the parameter file needs only to contain the SIZE value, an output file, and perhaps a list of input GOP files (see below).

If you don't supply a list of input GOP files is used, then **ppmtompeg** assumes you're using the same parameter file you used when you created the input (with the **-gop** option) and calculates the corresponding gop filenames itself. If this is not the case, you can specify input GOP files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use GOP_INPUT_DIR, GOP_INPUT, and GOP_END_INPUT. If no input GOP files are specified, then the default is to use the output file name with suffix **.gop.***gop_num*, with *gop_num* starting from 0, as the input files.

Thus, unless you're mixing and matching GOP files from different sources, you can simply use the same parameter file for creating the GOP files (**-gop**) and for later turning them into an MPEG stream (**-combine_gops**).

-frames *first_frame last_frame*

This option causes **ppmtompeg** to encode only the frames numbered *first_frame* to *last_frame*, inclusive. The parameter file is the same as for normal usage. The output will be placed in separate files, one per frame, with the file names being the normal output file name with the suffix **.frame.***frame_num*. No GOP header information is output. (Thus, the parameter file need not include the GOP_SIZE value)

Use **ppmtompeg -combine_frames** to combine these frames later into an MPEG stream.

-combine_frames

This option causes **ppmtompeg** simply to combine some individual MPEG frames (such as you might have created with an earlier run of **ppmtompeg -frames**) into a single MPEG stream. Sequence and GOP headers are inserted appropriately. In this case, the parameter file needs to contain only the SIZE value, the GOP_SIZE value, an output file, and perhaps a list of frame files (see below).

The parameter file may specify input frame files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use FRAME_INPUT_DIR, FRAME_INPUT, and FRAME_END_INPUT. If no input frame files are specified, then the default is to use the output file name with suffix **.frame.***frame_num*, with *frame_num* starting from 0, as the input files.

-nice This option causes **ppmtompeg** to run any remote processes "nicely," i.e. at low priority. (This is relevant only if you are running **ppmtompeg** in parallel mode. Otherwise, there are no remote processes). See 'man nice.'

-max_machines *num_machines*

This option causes **ppmtompeg** to use no more than *num_machines* machines as slaves for use in parallel encoding.

-snr This option causes **ppmtompeg** to include the signal-to-noise ratio in the reported statistics. Prints SNR (Y U V) and peak SNR (Y U V) for each frame. In summary, prints averages of luminance only (Y). SNR is defined as 10*log(variance of original/variance of error). Peak SNR is defined as 20*log(255/RMSE). Note that **ppmtompeg** runs a little slower when you use this option.

-mse This option causes **ppmtompeg** to report the mean squared error per block. It also automatically reports the quality of the images, so there is no need to specify **-snr** then.

-bit_rate_info *rate_file*
This option makes **ppmtompeg** write bit rate information into the file *rate_file*. Bit rate information is bits per frame, and also bits per I-frame-to-I-frame.

-mv_histogram
This option causes **ppmtompeg** to print a histogram of the motion vectors as part of statistics. There are three histograms -- one for P frame, one for forward B frame, and one for backward B frame motion vectors.

The output is in the form of a matrix, each entry corresponding to one motion vector in the search window. The center of the matrix represents (0,0) motion vectors.

-debug_sockets
This option causes **ppmtompeg** to print to Standard Output messages that narrate the communication between the machines when you run **ppmtompeg** in parallel mode .

-debug_machines
This option causes **ppmtompeg** to print to Standard Output messages that narrate the progress of the conversion on the various machines when you run **ppmtompeg** in parallel mode .

PARAMETER FILE

The parameter file **must** contain the following lines (except when using the **-combine_gops** or **-combine_frames** options):

PATTERN *pattern*
This statement specifies the pattern (sequence) of I frames, P frames, and B frames. *pattern* is just a sequence of the letters I, P, and B with nothing between. Example:

PATTERN IBBPBBPBBPBBPBB

See I Frames, P Frames, B Frames .

OUTPUT *output file*
This names the file where the output MPEG stream goes.

INPUT_DIR *directory*
This statement tells where the input images (frames) come from. If each frame is in a separate file, *directory* is the directory where they all are. You may use **.** to refer to the current directory. A null *directory* refers to the root directory of the system file tree.

To have **ppmtompeg** read all the frames serially from Standard Input, specify
INPUT_DIR stdin

INPUT
This line must be followed by a list of the input files (in display order) and then the line **END_INPUT**.

There are three types of lines between INPUT and END_INPUT. First, a line may simply be

the name of an input file. Second, the line may be of the form *single_star_expr* **[***x-y***]**. *single_star_expr* can have a single ***** in it. It is replaced by all the numbers between x and y inclusive. So, for example, the line **tennis*.ppm [12-15]** refers to the files tennis12.ppm, tennis13.ppm, tennis14.ppm, tennis15.ppm.

Uniform zero-padding occurs, as well. For example, the line **football.*.ppm [001-130]** refers to the files football.001.ppm, football.002.ppm, ..., football.009.ppm, football.010.ppm, ..., football.130.ppm.

The third type of line is: *single_star_expr* **[***x-y+s***]**, where the line is treated exactly as above, except that we skip by *s*. Thus, the line **football.*.ppm [001-130+4]** refers to the files football.001.ppm, football.005.ppm, football.009.ppm, football.013.ppm, etc.

Furthermore, a line may specify a shell command to execute to generate lines to be interpreted as described above, as if those lines were in the parameter file instead. Use back ticks, like in the Bourne Shell, like this:

‘cat myfilelist‘

If input is from Standard Input (per the **INPUT_DIR** statement), **ppmtompeg** ignores the **INPUT**/**END_INPUT** block, but it still must be present.

BASE_FILE_FORMAT {**PPM** | **PNM** | **YUV** |
JPEG | **JMOVIE**} **ppmtompeg** must convert all input files to one of the following formats as a first step of processing: PNM, YUV, JPEG(v4), or JMOVIE. (The conversion may be trivial if your input files are already in one of these formats). This line specifies which of the four formats. PPM is actually a subset of PNM. The separate specification is allowed for backward compatibility. Use PNM instead of PPM in new applications.

INPUT_CONVERT *conversion_command*
You must specify how to convert a file to the base file format. If no conversion is necessary, then you would just say:

INPUT_CONVERT *

Otherwise, *conversion_command* is a shell command that causes an image in the format your specified with **BASE_FILE_FORMAT** to be written to Standard Output. **ppmtompeg** executes the command once for each line between **INPUT** and **END_INPUT** (which is normally, but not necessarily, a file name). In the conversion command, **ppmtompeg** replaces each ’*’ with the contents of that line.

If you had a bunch of gif files, you might say:
INPUT_CONVERT giftopnm *

If you have a bunch of separate a.Y, a.U, and a.V files (where
the U and V have already been subsampled), then you might say:

INPUT_CONVERT cat *.Y *.U *.V

Input conversion is not allowed with input from stdin, so use

INPUT_CONVERT *

as described above.

SIZE *width***x***height*

width and *height* are the width and height of each frame in pixels.

When **ppmtompeg** can get this information from the input image files, it ignores the **SIZE** parameter and you may omit it.

When the image files are in YUV format, the files don't contain dimension information, so **SIZE** is required.

When **ppmtompeg** is running in parallel mode, not all of the processes in the network have access to the image files, so **SIZE** is required and must give the same dimensions as the input image files.

YUV_SIZE *width***x***height*
This is an obsolete synonym of **SIZE**.

YUV_FORMAT {**ABEKAS** | **PHILLIPS** | **UCB** |
EYUV | *pattern*} This is meaningful only when **BASE_FILE_FORMAT** specifies YUV format, and then it is required. It specifies the sub-format of the YUV class.

GOP_SIZE *n*
n is the number of frames in a Group of Pictures. Except that because a GOP must start with an I frame, **ppmtompeg** makes a GOP as much longer than *n* as it has to to make the next GOP start with an I frame.

Normally, it makes sense to make your GOP size a multiple of your pattern length (the latter is determined by the PATTERN parameter file statement).

See Group Of Pictures .

SLICES_PER_FRAME *n*
n is roughly the number of slices per frame. Note, at least one MPEG player may complain if slices do not start at the left side of an image. To ensure this does not happen, make sure the number of rows is divisible by SLICES_PER_FRAME.

PIXEL {**FULL** | **HALF**}
use half-pixel motion vectors, or just full-pixel ones It is usually important that you use half-pixel motion vectors, because it results in both better quality and better compression.

RANGE *n*
Use a search range of *n* pixels in each of the four directions from a subject pixel. (So the search window is a square *n**2 pixels on a side).

PSEARCH_ALG {**EXHAUSTIVE** | **TWOLEVEL** |
SUBSAMPLE | **LOGARITHMIC**} This statement tells **ppmtompeg** what kind of search technique (algorithm) to use for P frames. You select the desired combination of speed and compression. **EXHAUSTIVE** gives the best compression, but **LOGARITHMIC** is the fastest. **TWOLEVEL** is an exhaustive full-pixel search, followed by a local half- pixel search around the best full-pixel vector (the PIXEL option is ignored for this search technique).

BSEARCH_ALG {**SIMPLE** | **CROSS2** | **EXHAUSTIVE**}

This statement tells **ppmtompeg** what kind of search technique (algorithm) to use for B frames. **SIMPLE** means find best forward and backward vectors, then interpolate. **CROSS2** means find those two vectors, then see what backward vector best matches the best forward vector, and vice versa. **EXHAUSTIVE** does an n-squared search and is *extremely* slow in relation to the others (**CROSS2** is about half as fast as **SIMPLE**).

IQSCALE *n*

Use *n* as the qscale for I frames.
See Qscale .

PQSCALE *n*

Use *n* as the qscale for P frames.
See Qscale .

BQSCALE *n*

Use *n* as the qscale for B frames.
See Qscale .

REFERENCE_FRAME {**ORIGINAL** | **DECODED**}

This statement determines whether **ppmtompeg** uses the original images or the decoded images when computing motion vectors. Using decoded images is more accurate and should increase the playback quality of the output, but it makes the encoding take longer and seems to give worse compression. It also causes some complications with parallel encoding. (see the section on parallel encoding). One thing you can do as a trade-off is select **ORIGINAL** here, and lower the qscale (see **QSCALE** if the quality is not good enough.

Original or Decoded? (Normalized)

Reference	Compression	Speed	Quality I	Quality P	Quality B
Decoded	1000	1000	1000	969	919
Original	885	1373	1000	912	884

The following lines are optional:

FORCE_ENCODE_LAST_FRAME

This statement is obsolete. It does nothing.

Before Netpbm 10.26 (January 2005), **ppmtompeg** would drop trailing B frames from your movie, since a movie can't end with a B frame. (See I Frames, P Frames, B Frames . You would have to specify **FORCE_ENCODE_LAST_FRAME** to stop that from happening and get the same function that **ppmtompeg** has today.

NIQTABLE

This statement specifies a custom non-intra quantization table. If you don't specify this statement, **ppmtompeg** uses a default non-intra quantization table.

The 8 lines immediately following **NIQTABLE** specify the quantization table. Each line defines a table row and consists of 8 integers, whitespace-delimited, which define the table columns.

IQTABLE
This is analogous to NIQTABLE, but for the intra quantization table.

ASPECT_RATIO *ratio*
This statement specifies the aspect ratio for **ppmtompeg** to specify in the MPEG output. I'm not sure what this is used for.

ratio must be 1.0, 0.6735, 0.7031, 0.7615, 0.8055, 0.8437, 0.8935, 0.9157, 0.9815, 1.0255, 1.0695, 1.0950, 1.1575, or 1.2015.

FRAME_RATE *rate*
This specifies the frame rate for **ppmtompeg** to specify in the MPEG output. Some players use this value to determine the playback rate.

rate must be 23.976, 24, 25, 29.97, 30, 50, 59.94, or 60.

BIT_RATE *rate*
This specifies the bit rate for Constant Bit Rate (CBR) encoding.

rate must be an integer.

BUFFER_SIZE *size*
This specifies the value **ppmtompeg** is to specify in the MPEG output for the Video Buffering Verifier (VBV) buffer size needed to decode the sequence.

A Video Verifying Buffer is a buffer in which a decoder keeps the decoded bits in order to match the uneven speed of the decoding with the required constant playback speed.

As **ppmtompeg** encodes the image, it simulates the decoding process in terms of how many bits would be in the VBV as each frame gets decoded, assuming a VBV of the size you indicate.

If you specify the **WARN_VBV_UNDERFLOW** statement, **ppmtompeg** issues a warning each time the simulation underflows the buffer, which suggests that an underflow would occur on playback, which suggests the buffer is too small.

If you specify the **WARN_VBV_OVERFLOW** statement, **ppmtompeg** issues a warning each time the simulation overflows the buffer, which suggests that an overflow would occur on playback, which suggests the buffer is too small.

WARN_VBV_UNDERFLOW

WARN_VBV_OVERFLOW
See **BUFFER_SIZE**.

These options were new in Netpbm 10.26 (January 2005). Before that, **ppmtompeg** issued the warnings always.

The following statements apply only to parallel operation:

PARALLEL

This statement, paired with **END PARALLEL**, is what causes **ppmtompeg** to operate in parallel mode. See Parallel Operation .

END PARALLEL

This goes with **PARALLEL**.

PARALLEL_TEST_FRAMES *n*

The master starts off by measuring each slave's speed. It does this by giving each slave *n* frames to encode and noting how long the slave takes to finish. These are not just test frames, though -- they're real frames and the results become part of the output. **ppmtompeg** is old and measures time in undivided seconds, so to get useful timings, specify enough frames that it will take at least 5 seconds to process them. The default is 10.

If you specify **FORCE_I_ALIGN**, **ppmtompeg** will increase the test frames value enough to maintain the alignment.

If there aren't enough frames for every slave to have the indicated number of test frames, **ppmtompeg** will give some slaves fewer.

PARALLEL_TIME_CHUNKS *t*

When you specify this statement, the master attempts to feed work to the slaves in chunks that take *t* seconds to process. It uses the speed measurement it made when it started up (see PARALLEL_TEST_FRAMES) to decide how many frames to put in the chunk. This statement obviously doesn't affect the first batch of work sent to each slave, which is the one used to measure the slave's speed.

Smaller values of *t* increase communication, but improve load balancing. The default is 30 seconds.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

PARALLEL_CHUNK_TAPER

When you specify this statement, the master distributes work like with PARALLEL_TIME_CHUNKS, except that the master chooses the number of seconds for the chunks. It starts with a large number and, as it gets closer to finishing the job, reduces it. That way, it reduces scheduling overhead when precise scheduling isn't helpful, but still prevents a slave from finishing early after all the work has already been handed out to the other slaves, and then sitting idle while there's still work to do.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

PARALLEL_PERFECT

If this statement is present, **ppmtompeg** schedules on the assumption that each machine is about the same speed. The master will simply divide up the frames evenly between the slaves -- each slave gets the same number of frames. If some slaves are faster than others, they will finish first and remain idle while the slower slaves continue.

This has the advantage of minimal scheduling overhead. Where slaves have different speeds, though, it makes inefficient use of the fast ones. Where slaves are the same speed, it also has the disadvantage that they all finish at the same time and feed their output to the single Combine Server in a burst, which makes less efficient use of the Combine Server and thus can increase the total elapsed time.

You may specify only one of PARALLEL_TIME_CHUNKS, PARALLEL_CHUNK_TAPER, and PARALLEL_PERFECT. PARALLEL_CHUNK_TAPER is usually best.

RSH *remote_shell_command*

ppmtompeg executes the shell command *remote_shell_command* to start a process on another machine. The default command is **rsh**, and whatever command you specify must have compatible semantics. **ssh** is usually compatible. The command **ppmtompeg** uses is one like this: **ssh remote.host.com -l username shellcommand**.

Be sure to set up **.rhosts** files or SSH key authorizations where needed. Otherwise, you'll have to type in passwords.

On some HP machines, **rsh** is the restricted shell, and you want to specify **remsh**.

FORCE_I_ALIGN

This statement forces each slave to encode a chunk of frames which is a multiple of the pattern length (see **PATTERN**). Since the first frame in any pattern is an I frame, this forces each chunk encoded by a slave to begin with an I frame.

This document used to say there was an argument to **FORCE_I_ALIGN** which was the number of frames **ppmtompeg** would use (and was required to be a multiple of the pattern length). But **ppmtompeg** has apparently always ignored that argument, and it does now.

KEEP_TEMP_FILES

This statement causes **ppmtompeg** not to delete the temporary files it uses to transmit encoded frames to the combine server. This means you will be left with a file for each frame, the same as you would get with the **-frames** option.

This is mostly useful for debugging.

This works only if you're using a shared filesystem to communicate between the servers.

This option was new in Netpbm 10.26 (January 2005).

Parameter File Notes

If you use the **-combine_gops** option, then you need to specify only the SIZE and OUTPUT values in the parameter file. In addition, the parameter file may specify input GOP files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use GOP_INPUT_DIR, GOP_INPUT, and GOP_END_INPUT. If you specify no input GOP files, then **ppmtompeg** uses by default the output file name with suffix **.gop.***gop_num*, with *gop_num* starting from 0, as the input files.

If you use the **-combine_frames** option, then you need to specify only the SIZE, GOP_SIZE, and OUTPUT values in the parameter file. In addition, the parameter file may specify input frame files in the same manner as normal input files -- except instead of using INPUT_DIR, INPUT, and END_INPUT, use FRAME_INPUT_DIR, FRAME_INPUT, and FRAME_END_INPUT. If no input frame files are specified, then the default is to use the output file name with suffix **.frame.***frame_num*, with *frame_num* starting from 0, as the input files.

Any number of spaces and tabs may come between each option and value. Lines beginning with **#** are ignored. Any other lines are ignored except for those between INPUT and END_INPUT. This allows you to use the same parameter file for normal usage and for **-combine_gops** and **-combine_frames**.

The file format is case-sensitive so all keywords should be in upper case.

The statements may appear in any order, except that the order within a block statement (such as INPUT

... END INPUT) is significant.

ppmtompeg is prepared to handle up to 16 B frames between reference frames when encoding with input from stdin. (To build a modified **ppmtompeg** with a higher limit, change the constant B_FRAME_RUN in frame.c and recompile).

GENERAL USAGE INFORMATION

Qscale

The quantization scale values (qscale) give a trade-off between quality and compression. Using different Qscale values has very little effect on speed. The qscale values can be set separately for I, P, and B frames.

You select the qscale values with the **IQSCALE**, **PQSCALE**, and **BSCALE** parameter file statements.

A qscale value is an integer from 1 to 31. Larger numbers give better compression, but worse quality. In the following, the quality numbers are peak signal-to-noise ratio, defined as: **signal-to-noise formula** where MSE is the mean squared error.

Flower garden tests:

Qscale vs Quality

Qscale	I Frames	P Frames	B Frames
1	43.2	46.3	46.5
6	32.6	34.6	34.3
11	28.6	29.5	30.0
16	26.3	26.8	28.6
21	24.7	25.0	27.9
26	23.5	23.9	27.5
31	22.6	23.0	27.3

Qscale vs Compression

Qscale	I Frames	P Frames	B Frames
1	2	2	2
6	7	10	15
11	11	18	43
16	15	29	97
21	19	41	173
26	24	56	256
31	28	73	330

Search Techniques

There are several different motion vector search techniques available. There are different techniques available for P frame search and B frame search. Using different search techniques present little difference in quality, but a large difference in compression and speed.

There are 4 types of P frame search: Exhaustive, TwoLevel, SubSample, and Logarithmic.

There are 3 types of B frame search: Exhaustive, Cross2, and Simple.

The recommended search techniques are TwoLevel and Logarithmic for P frame search, and Cross2 and Simple for B frame search. Here are some numbers comparing the different search methods:

P frame Motion Vector Search (Normalized)

Technique	Compression [1]	Speed [2]	Quality [3]
Exhaustive	1000	1000	1000
SubSample	1008	2456	1000

TwoLevel	1009	3237	1000
Logarithmic	1085	8229	998

B frame Motion Vector Search (Normalized)

Technique	Compression [1]	Speed [2]	Quality [3]
Exhaustive	1000	1000	1000
Cross2	975	1000	996
Simple	938	1765	991

[1] Smaller numbers are better compression.

[2] Larger numbers mean faster execution.

[3] Larger numbers mean better quality.

For some reason, Simple seems to give better compression, but it depends on the image sequence.

Select the search techniques with the **PSEARCH_ALG** and **BSEARCH_ALG** parameter file statements.

Group Of Pictures (GOP)

A Group of Pictures (GOP) is a roughly independently decodable sequence of frames. An MPEG video stream is made of one or more GOP's. You may specify how many frames should be in each GOP with the **GOP_SIZE** parameter file statement. A GOP always starts with an I frame.

Instead of encoding an entire sequence, you can encode a single GOP. To do this, use the **-gop** command option. You can later join the resulting GOP files at any time by running **ppmtompeg** with the **-combine_gops** command option.

Slices

A slice is an independently decodable unit in a frame. It can be as small as one macroblock, or it can be as big as the entire frame. Barring transmission error, adding slices does not change quality or speed; the only effect is slightly worse compression. More slices are used for noisy transmission so that errors are more recoverable. Since usually errors are not such a problem, we usually just use one slice per frame.

Control the slice size with the **SLICES_PER_FRAME** parameter file statement.

Some MPEG playback systems require that each slice consist of whole rows of macroblocks. If you are encoding for this kind of player, if the height of the image is H pixels, then you should set the SLICES_PER_FRAME to some number which divides H/16. For example, if the image is 240 pixels (15 macroblocks) high, then you should use only 15, 5, 3, or 1 slices per frame.

Note: these MPEG playback systems are really wrong, since the MPEG standard says this doesn't have to be so.

Search Window

The search window is the window in which **ppmtompeg** searches for motion vectors. The window is a square. You can specify the size of the square, and whether to allow half-pixel motion vectors or not, with the **RANGE** and **PIXEL** parameter file statements.

I Frames, P Frames, B Frames

In MPEG-1, a movie is represented as a sequence of MPEG frames, each of which is an I Frame, a P Frame, or a B Frame. Each represents an actual frame of the movie (don't get confused by the dual use

of the word "frame." A movie frame is a graphical image. An MPEG frame is a set of data that describes a movie frame).

An I frame ("intra" frame) describes a movie frame in isolation -- without respect to any other frame in the movie. A P frame ("predictive" frame) describes a movie frame by describing how it differs from the movie frame described by the latest preceding I or P frame. A B frame ("bidirectional" frame) describes a movie frame by describing how it differs from the the movie frames described by the nearest I or P frame before *and* after it.

Note that the first frame of a movie must be described by an I frame (because there is no previous movie frame) and the last movie frame must be described by an I or P frame (because there is no subsequent movie frame).

Beyond that, you can choose which frames are represented by which types. You specify a pattern, such as IBPBP and **ppmtompeg** simply repeats it over and over throughout the movie. The pattern affects speed, quality, and stream size. Here is a chart which shows some of the trade-offs:

Comparison of I/P/B Frames (Normalized)

Frame Type	Size	Speed	Quality
I frames	1000	1000	1000
P frames	409	609	969
B frames	72	260	919

(this is with constant qscale)

A standard sequence is IBBPBBPBBPBBPBB.

Select the sequence with the **PATTERN** parameter file statement.

Since the last MPEG frame cannot be a B frame (see above), if the pattern you specify indicates B frames for the last movie frames, **ppmtompeg** makes it an I frame instead.

Before Netpbm 10.26 (January 2005), **ppmtompeg** instead drops the trailing B frames by default, and you need the **FORCE_ENCODE_LAST_FRAME** parameter file statement to make it do this.

The MPEG frames don't appear in the MPEG-1 stream in the same order that the corresponding movie frames appear in the movie -- the B frames come after the I and P frames on which they are based. For example, if the movie is 4 frames that you will represent with the pattern IBBP, the MPEG-1 stream will start with an I frame describing movie frame 0. The next frame in the MPEG-1 stream is a P frame describing movie frame 3. The last two frames in the MPEG-1 stream are B frames describing movie frames 1 and 2, respectively.

Specifying Input and Output Files

Specify the input frame images with the **INPUT_DIR**, **INPUT**, **END_INPUT**, **BASE_FILE_FORMAT**, **SIZE**, **YUV_FORMAT** and **INPUT_CONVERT** parameter file statements.

Specify the output file with the **OUTPUT** parameter file statement.

Statistics

ppmtompeg can generate a variety of statistics about the encoding. See the **-stat**, **-snr**, **-mv_histogram**, **-quiet**, **-no_frame_summary**, and **-bit_rate_info** options.

PARALLEL OPERATION

You can run **ppmtompeg** on multiple machines at once, encoding the same MPEG stream. When you do, the machines are used as shown in the following diagram. We call this 'parallel mode.'

ppmtompeg-par.gif

To do parallel processing, put the statement

PARALLEL

in the parameter file, followed by a listing of the machines, one machine per line, then

END_PARALLEL

Each of the machine lines must be in one of two forms. If the machine has filesystem access to the input files, then the line is:

machine user executable

The executable is normally **ppmtompeg** (you may need to give the complete path if you've built for different architectures). If the machine does not have filesystem access to the input files, the line is:

REMOTE *machine user executable parameter file*

The **-max_machines** command option limits the number of machines **ppmtompeg** will use. If you specify more machines in the parameter file than **-max_machines** allows, **ppmtompeg** uses only the machines listed first. This is handy if you want to experiment with different amounts of parallelism.

In general, you should use full path file names when describing executables and parameter files. This *includes* the parameter file argument on the original invocation of **ppmtompeg**.

All file names must be the same on all systems (so if e.g. you're using an NFS filesystem, you must make sure it is mounted at the same mountpoint on all systems).

Because not all of the processes involved in parallel operation have easy access to the input files, you must specify the **SIZE** parameter file statement when you do parallel operation.

The machine on which you originally invoke **ppmtompeg** is the master machine. It hosts a 'combine server,', a 'decode server,' and a number of 'i/o servers,' all as separate processes. The other machines in the network (listed in the parameter file) are slave machines. Each hosts a single process that continuously requests work from the master and does it. The slave process does the computation to encode MPEG frames. It processes frames in batches identified by the master.

The master uses a remote shell command to start a process on a slave machine. By default, it uses an **rsh** shell command to do this. But use the **RSH** parameter file statement to control this. The shell command the master executes remotely is **ppmtompeg**, but with options to indicate that it is to perform slave functions.

The various machines talk to each other over TCP connections. Each machine finds and binds to a free TCP port number and tells its partners the port number. These port numbers are at least 2048.

Use the PARALLEL_TEST_FRAMES, PARALLEL_TIME_CHUNKS, and PARALLEL_PERFECT parameter file statements to control the way the master divides up work among the slaves.

Use the **-nice** command option to cause all slave processes to run "nicely," i.e. as low priority processes. That way, this substantial and long-running CPU load will have minimal impact on other, possibly interactive, users of the systems.

SPEED

Here is a look at **ppmtompeg** speed, in single-node (not parallel) operation:

Compression Speed

Machine Type	Macroblocks per second[1]
HP 9000/755	280
DEC 3000/400	247
HP 9000/750	191
Sparc 10	104
DEC 5000	68

[1] A macroblock is a 16x16 pixel square

The measurements in the table are with inputs and outputs via a conventional locally attached filesystem. If you are using a network filesystem over a single 10 MB/s Ethernet, that constrains your speed more than your CPU speed. In that case, don't expect to get better than 4 or 5 frames per second no matter how fast your CPUs are.

Network speed is even more of a bottleneck when the slaves do not have filesystem access to the input files -- i.e. you declare them REMOTE.

Where I/O is the bottleneck, size of the input frames can make a big difference. So YUV input is better than PPM, and JPEG is better than both.

When you're first trying to get parallel mode working, be sure to use the **-debug_machines** option so you can see what's going on. Also, **-debug_sockets** can help you diagnose communication problems.

AUTHORS

- Kevin Gong - University of California, Berkeley, *keving@cs.berkeley.edu*
- Ketan Patel - University of California, Berkeley, *kpatel@cs.berkeley.edu*
- Dan Wallach - University of California, Berkeley, *dwallach@cs.berkeley.edu*
- Darryl Brown - University of California, Berkeley, *darryl@cs.berkeley.edu*
- Eugene Hung - University of California, Berkeley, *eyhung@cs.berkeley.edu*
- Steve Smoot - University of California, Berkeley, *smoot@cs.berkeley.edu*

Table Of Contents

NAME

ppmtoneo - convert a PPM into an Atari Neochrome .neo file

SYNOPSIS

ppmtoneo

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoneo reads a PPM image as input and produces an Atari Neochrome .neo file as output.

SEE ALSO

neotoppm(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmtopcx - convert a PPM image to a PCX file

SYNOPSIS

ppmtopcx

[**-24bit**]

[**-8bit**]

[**-packed**]

[**-stdpalette**]

[**-palette=***palettefile*]

[**-planes=***planes*]

[**-xpos=***cols*]

[**-ypos=***rows*]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopcx reads a PPM image as input and produces a PCX file as output. The type of the PCX file depends on the number of colors in the pixmap:

16 colors or fewer:
1 bit/pixel, 1-4 planes.

more than 16 colors, but no more than 256:
8 bits/pixel, 1 plane, colormap at the end of the file.

More than 256 colors:
24bit truecolor file (8 bits/pixel, 3 planes).

You can override some of that and explicitly choose the format with the options below.

OPTIONS

-24bit Produce a 24bit truecolor PCX file, even if the image has 256 colors or fewer.

-8bit Produce an 8bit (256 colors) PCX file, even if the image has 16 colors or fewer.

This option was added in Netpbm 10.18 (August 2003).

-packed

Use 'packed pixel' format for files with 16 colors or fewer: 1, 2, or 4 bits/pixel, 1 plane.

-stdpalette

Instead of computing a palette from the colors in the image, use a standard, built-in 16 color palette. If the image contains a color that is not in the standard palette, **ppmtopcx** fails.

The standard palette is not only a set of colors, but a specific mapping of palette indexes to colors. E.g. red is 4.

You can use **pnmremap** with a suitable PPM image of the standard palette to adapt your image to use exactly those colors in the palette so that **ppmtopcx -stdpalette** will work on it.

The file **pcxstd.ppm**, part of Netpbm, contains the standard palette.

Although the PCX header tells exactly what palette is used in the file, some older PCX interpreters do not use that information. They instead assume the standard palette. If you don't use the **-stdpalette** option, **ppmtopcx**, **ppmtopcx** may create an image that uses a different palette (a rearrangement of the same colors) and then one of these older interpreters would interpret the colors in the image wrong.

You cannot specify this option along with **-palette**.

This option was new in Netpbm 10.22 (April 2004).

-palette=*palettefile*

Instead of computing the palette from the colors in the image, use the palette from the file *palettefile*. If the palette contains a color that is not in that palette, **ppmtopcx** fails.

The palette file must be a PPM image that contains one pixel for each color in the palette. It doesn't matter what the aspect ratio of the palette image is. The order of the colors in the PCX palette is the order of the pixels in the PPM image in standard western reading order (left to right, top to bottom). If there is a duplicate color in the palette, **ppmtopcx** chooses between them arbitrarily in building the PCX raster.

You would need this only if you have a PCX reader that can't read the palette that is in the PCX file and instead assumes some particular palette. See also the **-stdpalette** option.

If your input image might contain colors other than those in your palette, you can convert the input image to one that contains only those colors in your palette with **pnmremap**.

You cannot specify this along with **-stdpalette**.

This option was new in Netpbhm 10.25 (October 2004).

-planes=*planes*

Generate a PCX file with *planes* planes, even though the number of colors in the image could be represented in fewer. This makes the file larger, but some PCX interpreters are capable of processing only certain numbers of planes.

This is meaningful only when **ppmtopcx** generates an image in the 16 color palette format without packed pixels. Consequently, you cannot specify this option together with **-24bit** or **-8bit** or **-packed**.

The valid values for *planes* are 1, 2, 3, and 4. By default, **ppmtopcx** chooses the smallest number of planes that can represent the colors in the image. E.g. if there are 5 colors, **ppmtopcx** chooses 3 planes.

This option was new in Netpbm 10.21 (March 2004).

-xpos=*cols*

-ypos=*rows*

These options set the position of the image in some field (e.g. on a screen) in columns to the right of the left edge and rows below the top edge. The PCX format contains image position information. Don't confuse this with the position of an area of interest within the image. For example, using **pnmpad** to add a 10 pixel left border to an image and then converting that image to PCX with xpos = 0 is not the same as converting the original image to PCX and setting xpos = 10.

The values may be from -32767 to 32768.

The default for each is zero.

SEE ALSO

pcxtoppm(1), **ppm**(1)

AUTHORS

Based on previous work by Michael Davidson.

Table Of Contents

NAME

ppmtopgm - convert a PPM image to a PGM image

SYNOPSIS

ppmtopgm

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopgm reads a PPM as input and produces a PGM as output. The output is a 'black and white' rendering of the original image, as in a black and white photograph. The quantization formula **ppmtopgm** uses is g = .299 r + .587 g + .114 b.

Note that although there is a **pgmtoppm** program, it is not necessary for simple conversions from pgm to ppm , because any ppm program can read pgm (and pbm) files automatically. **pgmtoppm** is for colorizing a pgm file. Also, see **ppmtorgb3** for a different way of converting color to gray. And **ppmdist** generates a grayscale image from a color image, but in a way that makes it easy to differentiate the original colors, not necessarily a way that looks like a black and white photograph.

QUOTE

Cold-hearted orb that rules the night
Removes the colors from our sight
Red is gray, and yellow white
But we decide which is right
And which is a quantization error.

SEE ALSO

pgmtoppm(1), **ppmtorgb3**(1), **rgb3toppm**(1), **ppmdist**(1), **ppm**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

ppmtopi1 - convert a PPM image into an Atari Degas .pi1 file

SYNOPSIS

ppmtopi1

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopi1 reads a PPM image as input and produces an Atari Degas .pi1 file as output.

SEE ALSO

pi1toppm(1), **ppm**(1), **pbmtopi3**(1), **pi3topbm**(1)

AUTHOR

Table Of Contents

NAME

ppmtopict - convert a PPM image to a Macintosh PICT file

SYNOPSIS

ppmtopict

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopict reads a PPM image as input and produces a Macintosh PICT file as output.

The generated file is only the data fork of a picture. You will need a program such as *mcvert* to generate a Macbinary or a BinHex file that contains the necessary information to identify the file as a PICT file to MacOS.

Even though PICT supports 2 and 4 bits per pixel, **ppmtopict** always generates an 8 bits per pixel file.

LIMITATIONS

The picture size field is correct only if the output is to a file since writing into this field requires seeking backwards on a file. However the PICT documentation seems to suggest that this field is not critical anyway since it is only the lower 16 bits of the picture size.

SEE ALSO

picttoppm(1), **ppm**(1), **mcvert**

AUTHOR

Table Of Contents

NAME

ppmtopj - convert a PPM image to an HP PaintJet file

SYNOPSIS

ppmtopj

[**-gamma** *val*]

[**-xpos** *val*]

[**-ypos** *val*]

[**-back** {**dark**|**lite**}]

[**-rle**]

[**-center**]

[**-render** { **none** | **snap** | **bw** | **dither** | **diffuse** | **monodither** | **monodiffuse** | **clusterdither** | **monoclusterdither** }]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopj reads a PPM image as input and converts it into a format suitable to be printed by an HP PaintJet printer.

For best results, the input file should be in 8-color RGB form; i.e. it should have only the 8 binary combinations of full-on and full-off primaries. You could convert your input to this format like this:

```
pamseq 3 1 testimg.ppm >8color.pam
pnmremap -map 8color.pam testimg.pam | ppmtopj
```

Or you could use use

```
ppmdither -red 2 -green 2 -blue 2
```

OPTIONS

-rle Run length encode the image. (This can result in larger images)

-back Enhance the foreground by indicating if the background is light or dark compated to the foreground.

-render *alg*

Use an internal rendering algorithm (default dither).

-gamma *int*

Gamma correct the image using the integer *int* as a gamma (default 0).

-center Center the image to an 8.5 by 11 page

-xpos *pos*
Move by *pos* pixels in the x direction.

-ypos *pos*
Move by *pos* pixels in the y direction.

SEE ALSO

HP PaintJet XL Color Graphics Printer User's Guide, **pnmtopclxl.html**(1), **pjtoppm.html**(1), **pnmdepth**(1), **pnmremap**(1), **pamseq**(1), **ppmdither**(1), **pbmtolj.html**(1), **ppmtolj**(1), **thinkjet-topbm**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

ppmtopjxl - convert a PPM image to an HP PaintJet XL PCL file

SYNOPSIS

ppmtopjxl [**-nopack**] [**-gamma** *n*] [**-presentation**] [**-dark**] [**-diffuse**] [**-cluster**] [**-dither**] [**-xshift** *s*] [**-yshift** *s*] [**-xshift** *s*] [**-yshift** *s*] [{**-xsize**|**-width**|**-xscale**} *s*] [{**-ysize**|**-height**|**-yscale**} *s*] [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopjxl reads a PPM image as input and produces a PCL file suitable for printing on an HP PaintJet XL printer as output.

The generated file is not suitable for printing on a normal PrintJet printer. The **-nopack** option generates a file which does not use the normal TIFF 4.0 compression method. This file might be printable on a normal PaintJet printer (not an XL).

The **-gamma** option sets the gamma correction for the image. The useful range for the PaintJet XL is approximately 0.6 to 1.5.

You an alter the rendering algorithm used for images with the **-dither, -cluster,** and **-diffuse** options. These options select ordered dithering, clustered ordered dithering, or error diffusion respectively. You can use the **-dark** option to enhance images with a dark background when they are reduced in size. The **-presentation** option turns on presentation mode, in which two passes are made over the paper to increase ink density. You should use this only for images where quality is critical.

You can resize the image by setting the **-xsize** and **-ysize** options. The parameter to either of these options is interpreted as the number of dots to set the width or height to, but you may append an optional dimension of '**pt**' (points), '**dp**' (decipoints), '**in**' (inches), or '**cm**' (centimetres). If you specify only one dimension, **ppmtopjxl** will scale the other one appropriately.

The options **-width** and **-height** are synonyms of **-xsize** and **-ysize.**

You can alternatively use the **-xscale** and **-yscale** options to scale the image by a simple factor.

You can shift the image on the page with the **-xshift** and **-yshift** options. These move the image the specified dimensions right and down.

SEE ALSO

ppm(1)

AUTHOR

Angus Duggan

Table Of Contents

NAME

ppmtoppm - copy PPM image

SYNOPSIS

ppmtoppm

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoppm simply copies a PPM image from Standard Input to Standard Output. This may seem an unnecessary duplication of **cat**, but remember that a PPM program can read a PBM or PGM image as if it were PPM. So **ppmtoppm** can read either a PBM, PGM, or PPM image and produce a PPM image as output.

Even that is of limited usefulness because of the fact that almost any program to which you would feed the resulting PPM image could also just take the original image directly. However, sometimes you really need a true PPM image.

When you know you have a PGM image and want a PPM image, **pgmtoppm** is a more general way to do that conversion. When you know you have a PBM image, use that and **pbmtopgm**.

SEE ALSO

pgmtopgm(1), **pnmtopnm**(1), **pamtopnm**(1), **pgmtoppm**(1), **pbmtopgm**(1), **ppm**(1), **ppm**(1), **ppm**(1),

HISTORY

This program was added to Netpbm in Release 10.9 (September 2002).

Table Of Contents

NAME

ppmtopuzz - convert a PPM image to an X11 ’puzzle’ file

SYNOPSIS

ppmtopuzz

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtopuzz reads a PPM image as input and produces an X11 ’puzzle’ file as output. A ’puzzle’ file is for use with the **puzzle** program included with the X11 distribution. **puzzle**’s **-picture** flag lets you specify an image file.

SEE ALSO

ppm(1), **puzzle**

AUTHOR

Table Of Contents

NAME

ppmtorgb3 - separate a PPM image into three PGMs

SYNOPSIS

ppmtorgb3

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtorgb3 reads a PPM image as input and writes three PGM images as output, one each for red, green, and blue.

ppmtorgb3 constructs the output filenames by taking the input filename, stripping off any extension, and appending **.red**, **.grn**, **.blu**. For example, separating **lenna.ppm** would result in **lenna.red**, **lenna.grn**, and **lenna.blu**. If the input comes from stdin, the names are **noname.red**, **noname.grn**, and **noname.blu**.

SEE ALSO

rgb3toppm(1), **pamchannel**(1), **ppmtopgm**(1), **pgmtoppm**(1), **ppm**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

ppmtosixel - convert a PPM image to DEC sixel format

SYNOPSIS

ppmtosixel

[**-raw**]

[**-margin**]

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtosixel reads a PPM image as input and produces sixel commands (SIX) as output. The output is formatted for color printing, e.g. for a DEC LJ250 color inkjet printer.

If RGB values from the PPM file do not have maxval=100, **ppmtosixel** rescales them to maxval 100. A printer control header and a color assignment table begin the SIX file. Image data is in a compressed format by default. A printer control footer ends the image file.

OPTIONS

-raw If you specify this, each pixel will be explicitly described in the image file. If **-raw** is not specified, output will default to compressed format in which identical adjacent pixels are replaced by 'repeat pixel' commands. A raw file is often an order of magnitude larger than a compressed file and prints much slower.

-margin

If you don't specify **-margin**, the image will start at the left margin (of the window, paper, or whatever). If you *do* specify **-margin**, a 1.5 inch left margin will offset the image.

PRINTING

Generally, sixel files must reach the printer unfiltered. Use the lpr -x option or **cat filename > /dev/tty0?**.

LIMITATIONS

Upon rescaling, truncation of the least significant bits of RGB values may result in poor color conversion. If the original PPM maxval was greater than 100, rescaling also reduces the image depth. While the actual RGB values from the ppm file are more or less retained, the color palette of the LJ250 may not match the colors on your screen. This seems to be a printer limitation.

SEE ALSO

ppm(1)

AUTHOR

Table Of Contents

NAME

ppmtoterm - convert a PPM image to a ANSI ISO 6429 ascii image

SYNOPSIS

ppmtoterm

[*ppmfile*]

All options can be abbreviated to their shortest unique prefix. You may use two hyphens instead of one. You may separate an option name and its value with white space instead of an equals sign.

DESCRIPTION

This program is part of **Netpbm**(1).

This program tries to produce an accurate representation of a PPM image on an terminal that implements the ANSI ISO 6429 standard. It aproximates colors, finding the minimum cartesian distance between the input RGB vectors and the ones in the generated palette. As the available color palette is somewhat restricted, you get the best results when the colors in the original image are few and the RGB intensities are close to zero, half of maximum, and maximum.

You can usually get good results with cartoons or images with plain colors (no gradients). With photos, results can vary, but are usually not very accurate.

The output image has one line for each row and one character for each column of the input image. E.g. an 80 pixel by 25 pixel PPM image would fill up an 80x25 terminal screen. Use **pamscale** or **pamcut** to make your image fit properly on your screen.

The image starts at the current cursor position on the terminal screen. Each successive row starts at Column 0 on the screen. If you want to shift the image up or down, for example to center it, use **pnmpad** on the input.

This program was born with the objective of displaying nice color images on the linux console, e.g. a proper logo at Linux boot.

pbmto4425 does a similar thing for black and white images, using line drawing characters, on some terminals.

OPTIONS

None.

SEE ALSO

pamscale(1), **pamcut**(1), **pbmtoascii**(1), **pbmto4425**(1), **ppm**(1)

AUTHOR

HISTORY

This program was new in Netpbm 10.9 (August 2002).

NAME

ppmtotga - replaced by pamtotga

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtotga was replaced in Netpbm 10.6 (July 2002) by **pamtotga**(1).

pamtotga is backward compatible with **ppmtotga**, but also recognizes PAM input, including that with an alpha channel.

NAME

ppmtouil - replaced by pamtouil

DESCRIPTION

This program is part of **Netpbm**(1). In May 2002, **ppmtouil** was extended and renamed to **pamtouil**(1).

Table Of Contents

NAME

ppmtowinicon - convert PPM image into a Windows .ico file

SYNOPSIS

ppmtowinicon

[**-andpgms**]

[**-output=***output.ico*]

[*ppmfile* [*andfile*] ...]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtowinicon reads one or more PPM images as input and produces a Microsoft Windows .ico file as output.

A Windows icon contains 1 or more images, at different resolutions and color depths. When Windows wants to display the icon, it searches through the images to find the one the best matches the number of colors and resolution of the display.

Microsoft recommends including at least the following formats in each icon.

- 16 x 16 - 4 bpp
- 32 x 32 - 4 bpp
- 48 x 48 - 8 bpp

If you don't specify any input files, input is from Standard Input.

Output is to Standard Output unless you specify **-output**.

Transparency

If you specify the **-andmask** option, you get (partly) transparent icons. In that case, your arguments are pairs of file names, with the first file name being that of the image and the second file name being that of a standard Netpbm PGM transparency mask (see the **pgm**format**specification**(1)).

In a .ico file, there is no such thing as partial transparency (translucency). Where the PGM mask says completely opaque, the icon will be opaque. Everywhere else, the icon will be transparent. Note that as with any Netpbm program, you can use a PBM image for the transparency mask and **ppmtowinicon** will treat it like a PGM.

The and mask is like an alpha mask, except for what it signifies in the "not opaque" areas. In the usual case, the foreground image is black in those areas, and in that case the areas are fully transparent -- the background shows through the icon. But in general, a not opaque pixel signifies that the background and foreground should be merged as follows: The intensities of the color components in the forge-ground and background are represented as binary numbers, then corresponding bits of the background and foreground intensities are exlusive-or'ed together. So there is a sort of reverse video effect.

If you don't want this special effect and instead want straightforward transparency, use the **-truetrans-parent** option. This causes **ppmtowinicon** to make the base image black everywhere your trans-parency mask says transparent, regardless of what color you input image is at that location.

If you don't specify **-andmask**, **ppmtowinicon** puts all-opaque and masks into the .ico file.

OPTIONS

-andpgms

Include transparency information in the icons. See the transparency section .

-output=*output.ico*

Name of output file. By default, **ppmtowinicon** writes the icon to Standard Output.

-truetransparent

Make transparency in the icon normal instead of the special reverse video effect. See the transparency section .

SEE ALSO

winicontoppm(1), **ppm**(1) **pgm**(1)

AUTHOR

Table Of Contents

NAME

ppmtoxpm - convert a PPM image to an X11 pixmap

SYNOPSIS

ppmtoxpm [**-name=***xpmname*] [**-hexonly**] [**-rgb=***rgb-textfile*] [**-alphamask=***pgmfile*] [*ppmfile*]

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoxpm reads a PPM image as input and produces X11 pixmap (version 3) as output. This format can be loaded by the XPM library.

In the XPM output, colors may be identified by name, such as "Red", or in hexadecimal, for example "#FF0000". In the hexadecimal format, there may be from 1 through 4 hexadecimal digits per RGB component.

By default, **ppmtoxpbm** tries to find a name for each color in the image in the system color dictionary , and if it finds one, uses it. If it doesn't it uses hexadecimal. You can force **ppmtoxpbm** to use hexadecimal only with the **-hexonly** option. You can specify a different color dictionary with the **-rgb** option.

When **ppmtoxpm** uses the hexadecimal format for identifying a color, it uses the one that uses the least number of hexadecimal digits that it takes to represent the maxval of the input PPM. E.g. if the maxval of the input PPM is 100, **ppmtoxpm** uses 2 digits per component, as in "#FF0000".

Some programs do not properly handle one-digit-per-component hexadecimal color specifiers. They see the wrong colors. To produce an XPM that such a program can handle, make sure the maxval of the input PPM is greater than 15, such as by running it through **pnmdepth 255**.

Color Code Lengths - Image Size

In the XPM format, there is a palette ('color map') that assigns each color in the image to a unique sequence of printable characters called a color code, and a raster that identifies the color of each pixel of the image with one of those color codes. The length of the color code affects the size of the image stream.

All color codes in an image are the same length, and **ppmtoxpm** tries to make it as short as possible. That length is, of course, determined by the number of colors in the image. **ppmtoxpm** counts the colors in the image, excluding those that will be transparent in the output due to your alpha mask, and chooses a color code length accordingly. There are 92 printable characters that can be used in a color code. Therefore, if you have 92 or fewer colors, your color codes will be one character. If you have more than 92 but not more than 92 * 92, your color codes will be two characters. And so on.

There's one exception to the above: If you specify an alpha mask (the **-alpha** option, one unique color code represents 'transparent.' This is true even if the alpha mask doesn't actually produce any transparent pixels. So subtract one from the number of possible colors if you use **-alpha**.

OPTIONS

-name=*xpmname*

This option specifies the prefix string which is specified in the resulting XPM output. If you don't use the **-name** otpion, **ppmtoxpm** defaults to the filename (without extension) of the *ppmfile* parameter. If you do not specify **-name** or *ppmfile* (i.e. your input is from Standard Input), the prefix string defaults to the string **noname**.

-hexonly

This option says never to put color names in the XPM file, but rather to identify names by hexadecimal strings that explicitly identify RGB component intensities. This means the reader of the file need not have access to a suitable color dictionary to interpret it.

This option was introduced in Netpbm 10.15 (April 2003). Before that, it was the default, overridden by specifying **-rgb**.

-rgb=*rgb-textfile*

This option names the file in which the color dictionary you want to use resides. By default, **ppmtoxpm** uses the system color dictionary .

This option in meaningless when you specify **-hexonly**.

Before Netpbm 10.15 (April 2003), **ppmtoxpm** did not default to the system color dictionary. If you didn't specify **-rgb**, **ppmtoxpbm** would use only hexadecimal color specifiers.

-alphamask=*pgmfile*

This option names a PGM file to use as an alpha (transparency) mask. The file must contain an image the same dimensions as the input image. **ppmtoxpm** marks as transparent any pixel whose position in the alpha mask image is at most half white.

If you don't specify **-alphamask**, **ppmtoxpm** makes all pixels in the output opaque.

ppmcolormask is one way to generate an alpha mask file. You might also generate it by extracting transparency information from an XPM file with the **-alphaout** option to **xpmtoppm**.

There are similar options on other Netpbm converters that convert from formats that include transparency information too.

SEE ALSO

ppmcolormask(1), **xpmtoppm**(1), **pnmdepth**(1), **ppm**(1) XPM Manual by Arnaud Le Hors *lehors@mirsa.inria.fr*

AUTHOR

Upgraded to XPM2 by Paul Breslaw, Mecasoft SA, Zurich, Switzerland (*paul@mecazh.uu.ch*), November 8, 1990.

Upgraded to XPM version 3 by Arnaud Le Hors(*lehors@mirsa.inria.fr*), April 9, 1991.

Table Of Contents

NAME

ppmtoyuv - convert a PPM image to an Abekas YUV file

SYNOPSIS

ppmtoyuv [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoyuvReads a PPM image as input and produces an Abekas YUV file as output.

The output file contains a raster of four byte YUV codes, each uniquely associated with two side-by-side pixels in the image. The raster contains rows in order from top to bottom, and within each row columns from left to right. So the output file size in bytes is twice the number of pixels in the image.

Each YUV code is associated with two pixels from the input image that we will call the left pixel and the right pixel. The 2nd byte of the code is the Y value of the left pixel. The 4th byte of the code is the Y value of the right pixel. The 1st byte of the code is an average of the U value of the pixel *to the left of the left pixel*, the left pixel, and the right pixel. The 3rd byte of the code is analogous for V values. These averages are weighted arithmetic means where the left pixel is weighted double what the other two pixels are weighted.

SEE ALSO

yuvtoppm(1), **ppmtoeyuv**(1), **ppmtoyuvsplit**(1), **ppm**(1)

AUTHOR

Marc Boucher *marc@PostImage.COM*, based on Example Conversion Program, A60/A64 Digital Video Interface Manual, page 69.

Table Of Contents

NAME

ppmtoyuvsplit - convert a PPM image to 3 subsampled raw YUV files

SYNOPSIS

ppmtoyuvsplit *basename* [*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtoyuvsplit reads a PPM image as input. Produces 3 raw files *basename*.Y, *basename*.U and *basename.V* as output.

The output files are the subsampled raw YUV representation of the input PPM image, as required by the Stanford MPEG codec. The Y output file contains a byte for each pixel in the image, with the rows going from top to bottom and the columns within each row going left to right. The U and V output files are arranged similarly, except that each byte represents a square of 4 pixels of the image. The value is the arithmetic mean of the value for each of those 4 pixels. Hence, the Y file is 4 times the size of the U file or V file.

The YUV values are scaled according to CCIR.601, as assumed by MPEG.

SEE ALSO

yuvsplittoppm(1), **ppmtoyuv**(1), **ppmtoeyuv**(1), **ppmtompeg**(1), **ppm**(1)

AUTHOR

Based on ppmtoyuv.c

Table Of Contents

NAME

ppmtv - make a PPM image look like taken from an American TV

SYNOPSIS

ppmtv *dimfactor*

[*ppmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ppmtv reads a PPM image as input and dims every other row of image data down by the specified dim factor. This factor may be in the range of 0.0 (the alternate lines are totally black) to 1.0 (original image).

This creates an effect similar to what I've once seen in the video clip 'You could be mine' by Guns'n'Roses. In the scene I'm talking about you can see John Connor on his motorbike, looking up from the water trench (?) he's standing in. While the camera pulls back, the image gets 'normal' by brightening up the alternate rows of it. I thought this would be an interesting effect to try in MPEG. I did not yet check this out, however. Try for yourself.

SEE ALSO

ppm(1), **ppmdim**(1)

AUTHOR

Table Of Contents

NAME

ppmwheel - make a PPM image of a color wheel

SYNOPSIS

ppmwheel *diameter*

DESCRIPTION

This program is part of **Netpbm**(1).

ppmwheel produces a PPM image of a color wheel of the specified diameter inside a white square just large enough to hold it.

The color wheel is based on the HSV color model. Hues are distributed angularly around the circle and the values are distributed radially and the saturation is zero everywhere. The values are zero at the center, increasing linearly to maximum at the edge. The maximum value corresponds to the maxval of the PPM image.

Hence, the image contains all of the secondary colors based on the red, green, and blue primary colors. A secondary color is one that is composed of light of at most two of the three primary colors.

OPTIONS

None.

SEE ALSO

ppmcie(1), **ppmrainbow**(1), **ppm**(1)

HISTORY

ppmwheel was added to Netpbm in Release 10.14 (March 2003).

AUTHOR

Table Of Contents

NAME

psidtopgm - convert PostScript 'image' data to a PGM image

SYNOPSIS

psidtopgm

width height bits/sample

[*imagedata*]

DESCRIPTION

This program is part of **Netpbm**(1).

psidtopgm reads the 'image' data from a PostScript file as input and produces a PGM image as output.

This program is obsoleted by **pstopnm**.

What follows was written before **pstopnm** existed.

This is a very simple and limited program, and is here only because so many people have asked for it. To use it you have to *manually* extract the readhexstring data portion from your PostScript file, and then give the width, height, and bits/sample on the command line. Before you attempt this, you should *at least* read the description of the 'image' operator in the PostScript Language Reference Manual.

It would probably not be too hard to write a script that uses this filter to read a specific variety of PostScript image, but the variation is too great to make a general-purpose reader. Unless, of course, you want to write a full-fledged PostScript interpreter...

SEE ALSO

pnmtops(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

pstopnm - convert a PostScript file to a PNM image

SYNOPSIS

pstopnm

[**-stdout**]

[**-forceplain**]

[**-help**]

[**-dpi**=*dpi*]

[**-xsize**=*pixels*] [**-ysize**=*pixels*]

[**-xborder**=*frac*] [**-yborder**=*frac*]

[**-landscape**]

[**-portrait**]

[**-nocrop**]

[**-pbm**

|**-pgm**

|**-ppm**]

[**-llx**=*s*] [**-lly**=*s*] [**-urx**=*s*] [**-ury**=*s*]

[**-verbose**]

[**-xmax**=*pixels*] [**-ymax**=*pixels*]

psfile[**.ps**]

OPTION USAGE

Minimum unique abbreviation of option is acceptable. You may use double hyphens instead of single hyphen to denote options. You may use white space in place of the equals sign to separate an option name from its value.

DESCRIPTION

This program is part of **Netpbm**(1).

pstopnm reads a PostScript file as input and produces PBM, PGM, or PPM images as output. This program simply uses GhostScript to render a PostScript file with its PNM device drivers. If you don't have GhostScript installed or the version you have installed was not built with the relevant PNM device drivers, **pstopnm** will fail. You can see if you have the proper environment by issuing the command `gs --help` . If it responds and lists under 'Available Devices' **pbm**, **pbmraw**, **pgm**, **pgmraw**, **pnm**, **pnmraw**, **ppm**, or **ppmraw**, you're in business.

pstopnm uses the value of the **GHOSTSCRIPT** environment variable as the file name for the Ghostscript program. If **GHOSTSCRIPT** is not set, **pstopnm** searches your **PATH** for a regular file

named **gs**. If it doesn't find one, it assumes Ghostscript is in the file **/usr/bin/gs**.

pstopnm does not use the Netpbm libraries to generate the output files, so may not be entirely consistent with most Netpbm programs.

psfile[**.ps**] is the name of the input file. **pstopnm** will add the **ps** to the end of the name you specify if no file exists by the exact name you specify, but one with added does. Use **-** to indicate Standard Input.

If you use the **-stdout** option, **pstopnm** outputs images of all the pages as a multi-image file to Standard Output. Otherwise, **pstopnm** creates one file for each page in the Postscript document. The files are named as follows: If the input file is named **psfile.ps**, the name of the files will be **psfile001.ppm**, **psfile002.ppm**, etc. The filetype suffix is **.ppm**, **.pgm**, or **.pbm**, depending on which kind of output you choose with your invocation options. If the input file name does not end in **.ps**, the whole file name is used in the output file name. For example, if the input file is named **psfile.old**, the output file name is **psfile.old001.ppm**, etc.

Note that the output file selection is inconsistent with most Netpbm programs, because it does not default to Standard Output. This is for historical reasons, based on the fact that the Netpbm formats did not always provide for a sequence of images in a single file.

Each output image contains a rectangular area of the page to which it pertains. See the Dimensions section for details on what part of the input image goes into the output image and how big it is in the output and what borders and margins are in the output image.

It has been reported that on some Postscript Version 1 input, Ghostscript, and therefore **pstopnm**, produces no output. To solve this problem, you can convert the file to Postscript Version 3 with the program **ps2ps**. It is reported that the program **pstops** does not work.

Dimensions

This section describes what part of the input image gets used in the output and the dimensions of the output, including borders and background.

Note that an output image is associated with a single input page.

pstopnm starts by taking a rectangular area from the input page. That is called the subject image.

pstopnm may add borders to the subject image to form what is called the bordered subject image.

pstopnm places the bordered subject image in the center of the output image and clips the edges as necessary to fit the computed output image size.

The location of the subject image in the Postscript input page is defined by four numbers, the lower left corner and the upper right corner x and y coordinates. These coordinates are usually specified by the BoundingBox DSC statement (a Postscript comment) in the PostScript file, but they can be overridden by the user by specifying one or more of the following options: **-llx**, **-lly**, **-urx**, and **-ury**.

The presence and thickness of a border to be added to the subject image to form the bordered subject image is controlled by the options **-xborder** and **-yborder**. If **pstopnm** does not find a BoundingBox statement in the input, and you don't specify image area coordinates on the command line, **pstopnm** uses default values. If your input is from Standard Input, **pstopnm** does not use the BoundingBox values (due to the technical difficulty of extracting that information and still feeding the file to Ghostscript), so you either have to specify the image area coordinates or take the default.

The output image size is a confusing thing. In a Postscript file, things have spatial dimensions. For example, a particular line may be 3 centimeters long. A Postscript printer is supposed to print the line 3 centimeters long, using however many pixels that takes, without regard to how big the sheet of paper on which it is printing is. In a PNM image, by contrast, there is no spatial dimension; there are only pixels. You might have a line that is 100 pixels long, but the PNM image says nothing about how long that line should be on a printed page.

pstopnm fills the role of a Postscript printer. The PNM image is a virtual printed page. **pstopnm** must determine how many pixels it will use in the output image to represent an inch of input image, which is the "output device resolution." Think of it as the number of dots per inch the virtual printer prints on the virtual page.

The simplest thing is for you to tell **pstopnm** exactly what output device resolution to use, using the **-dpi** option. If you say for example **-dpi=300** and the bordered subject image is 2 inches by 2 inches,

the PNM output will be 600 pixels by 600 pixels.

Or you can set the output image dimensions with **-xsize** and **-ysize**. For example, if you say **-xsize=1000 -ysize=1000** and the bordered subject image is 2 inches by 2 inches, the output image is 1000 by 1000 pixels, with each pixel representing 1/500 inch of input image.

If you specify one of **-xsize** and **-ysize** and not the other, **pstopnm** defaults the other such that the output image has the same aspect ratio as the bordered subject image.

If you specify neither the output size nor the output device resolution, **pstopnm** does some weird computation which exists mainly for historical reasons:

If you specify **-nocrop**, **pstopnm** uses the values of **-xmax** and **-ymax** for the output image dimensions. These default to 612 and 792 pixels, respectively.

The final case, the default, is where you don't specify any size or resolution options of **-nocrop**. This is the most complicated case. In this case, **pstopnm** first chooses an output device resolution that would generate the number of pixels indicated by **-xmax** and **-ymax** from the bordered subject image. Then, based on that resolution, it chooses an output image size that is just large enough to accomodate the subject image (no borders). Remember (above) that **pstopnm** trims the edges of the bordered subject image to fit the computed output size.

OPTIONS

-forceplain

forces the output file to be in plain (text) format. Otherwise, it is in raw (binary) format. See **pbm**(1), etc.

-llx=*bx* selects *bx* as the lower left corner x coordinate (in inches) on the Postscript input page of the subject image. See the Dimensions section .

-lly=*by* selects *by* as the lower left corner y coordinate (in inches) on the Postscript input page of the subject image. See the Dimensions section .

-landscape

renders the image in landscape orientation.

-portrait

renders the image in portrait orientation.

-nocrop

This option causes **pstopnm** to make the output image exactly the dimensions of the bordered subject image. By default, **pstopnm** makes the output image the dimensions specified by **-xmax** and **-ymax**. See the Dimensions section .

-pbm

-pgm

-ppm selects the format of the output file. By default, all files are rendered as portable pixmaps (ppm format).

-stdout causes output to go to Standard Output instead of to regular files, one per page (see description of output files above). Use **pnmsplit** to extract individual pages from Standard Output.

-urx=*tx*

selects *tx* as the upper right corner x coordinate (in inches) on the Postscript input page of the subject image. See the Dimensions section .

-ury=*ty*

selects *ty* as the upper right corner y coordinate (in inches) on the Postscript input page of the subject image. See the Dimensions section .

-verbose

prints processing information to stdout.

-xborder=*frac*

specifies that the left and right borders added to the subject image are to be *frac* times the subject image width. The default value is 0.1. See the Dimensions section .

-xmax=*xmax*

specifies that the output image is to be *xmax* pixels wide. The default is 612. See the Dimensions section .

-xsize=*xsize*

specifies that the output image is to be *xsize* pixels wide. See the Dimensions section .

-yborder=*frac*

specifies that the top and bottom borders added to the subject image are to be *frac* times the subject image height. The default value is 0.1. See the Dimensions section .

-ymax=*ymax*

specifies that the output image is to be *ymax* pixels high. The default is 792. See the Dimensions section .

-ysize=*ysize*

specifies that the output image is to be *ymax* pixels high. See the Dimensions section .

-dpi=*dpi*

specifies the output device resolution, in dots per inch, of the Postscript printer that **pstopnm** simulates. This is the number of PNM pixels **pstopnm** generates for each inch of image. See the Dimensions section .

This option was new in Netpbm 10.21 (March 2004).

LIMITATIONS

The program will produce incorrect results with PostScript files that initialize the current transformation matrix. In these cases, page translation and rotation will not have any effect. To render these files, probably the best bet is to use the following options:

```
pstopnm -xborder 0 -yborder 0 -portrait -nocrop file.ps
```

Additional options may be needed if the document is supposed to be rendered on a medium different from letter-size paper.

SEE ALSO

gs, **pnmtops**(1), **psidtopgm**(1), **pbmtolps**(1), **pbmtoepsi**(1), **pnmsplit**(1), **pstofits**

COPYRIGHT

AUTHOR

Alberto Accomazzi, WIPL, Center for Astrophysics.

Table Of Contents

NAME

qrttoppm - convert output from the QRT ray tracer to a PPM image

SYNOPSIS

qrttoppm

[*qrtfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

qrttoppm reads a QRT file as input and and produces a PPM image as output.

SEE ALSO

ppm(1)

AUTHOR

Table Of Contents

NAME

rasttopnm - convert a Sun rasterfile to a PNM image

SYNOPSIS

rasttopnm

[*rastfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

rasttopnm reads a Sun rasterfile as input and produces a PNM image as output. The type of the output file depends on the input file - if it's black and white, **rasttopnm** writes a PBM image. If it's grayscale, **rasttopnm** writes a PGM. If it's color, **rasttopnm** writes a PPM. The program tells you which type it is generating.

SEE ALSO

pnmtorast(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

rawtopgm - convert raw grayscale bytes to a PGM image

SYNOPSIS

rawtopgm

[**-bpp** [**1**|**2**]]

[**-littleendian**]

[**-maxval** *N*]

[**-headerskip** *N*]

[**-rowskip** *N*]

[**-tb**|**-topbottom**]

[*width height*]

[*imagefile*]

DESCRIPTION

This program is part of **Netpbm**(1).

rawtopgm reads raw grayscale values as input and produces a PGM image as output. The input file is just a sequence of pure binary numbers, either one or two bytes each, either bigendian or littleendian, representing gray values. They may be arranged either top to bottom, left to right or bottom to top, left to right. There may be arbitrary header information at the start of the file (to which **rawtopgm** pays no attention at all other than the header's size).

Arguments to **rawtopgm** tell how to interpret the pixels (a function that is served by a header in a regular graphics format).

The *width* and *height* parameters tell the dimensions of the image. If you omit these parameters, **rawtopgm** assumes it is a quadratic image and bases the dimensions on the size of the input stream. If this size is not a perfect square, **rawtopgm** fails.

When you don't specify *width* and *height*, **rawtopgm** reads the entire input stream into storage at once, which may take a lot of storage. Otherwise, **rawtopgm** ordinarily stores only one row at a time.

If you don't specify *imagefile*, or specify **-**, the input is from Standard Input.

The PGM output is to Standard Output.

OPTIONS

-maxval *N*

N is the maxval for the gray values in the input, and is also the maxval of the PGM output image. The default is the maximum value that can be represented in the number of bytes used for each sample (i.e. 255 or 65535).

-bpp [**1**|**2**]

tells the number of bytes that represent each sample in the input. If the value is **2**, The most significant byte is first in the stream.

The default is 1 byte per sample.

-littleendian

says that the bytes of each input sample are ordered with the least significant byte first. Without this option, **rawtopgm** assumes MSB first. This obviously has no effect when there is only one byte per sample.

-headerskip *N*

rawtopgm skips over *N* bytes at the beginning of the stream and reads the image immediately after. The default is 0.

This is useful when the input is actually some graphics format that has a descriptive header followed by an ordinary raster, and you don't have a program that understands the header or you want to ignore the header.

-rowskip *N*

If there is padding at the ends of the rows, you can skip it with this option. Note that rowskip need not be an integer. Amazingly, I once had an image with 0.376 bytes of padding per row. This turned out to be due to a file-transfer problem, but I was still able to read the image.

Skipping a fractional byte per row means skipping one byte per multiple rows.

-bt -bottomfirst

By default, **rawtopgm** assumes the pixels in the input go top to bottom, left to right. If you specify **-bt** or **-bottomfirst**, **rawtopgm** assumes the pixels go bottom to top, left to right. The Molecular Dynamics and Leica confocal format, for example, use the latter arrangement.

If you don't specify **-bt** when you should or vice versa, the resulting image is upside down, which you can correct with **pamflip**.

This option causes **rawtopgm** to read the entire input stream into storage at once, which may take a lot of storage. Ordinarly, **rawtopgm** stores only one row at a time.

For backwards compatibility, **rawtopgm** also accepts **-tb**
and **-topbottom** to mean exactly the same thing. The reasons these are named backwards is that the original author thought of it as specifying that the wrong results of assuming the data is top to bottom should be corrected by flipping the result top for bottom. Today, we think of it as simply specifying the format of the input data so that there are no wrong results.

SEE ALSO

pgm(1), **rawtoppm**(1), **pamflip**(1)

AUTHORS

 Modified June 1993 by Oliver Trepte, *oliver@fysik4.kth.se*

Table Of Contents

NAME

rawtoppm - convert a stream of raw RGB bytes to a PPM image

SYNOPSIS

rawtoppm

[**-headerskip** *N*]

[**-rowskip** *N*]

[**-rgb**|**-rbg**|**-grb** |**-gbr**|**-brg**|**-bgr**]

[**-interpixel**|**-interrow**]

width height

[*imagedata*]

DESCRIPTION

This program is part of **Netpbm**(1).

rawtoppm reads raw RGB bytes as input and produces a PPM image as output. The input file is just RGB bytes. You have to specify the width and height on the command line, since the program obviously can't get them from the file. **rawtoppm** assumes the maxval of the input samples is 255, and makes the maxval of the output PPM 255.

rawtoppm assumes the pixels come top first in the input stream. If they are actually bottom first, the resulting PPM is upside down, so run it through **pamflip -tb**.

OPTIONS

-headerskip

Skip over this many bytes at the beginning of the input stream. Use this option when the input has some kind of header followed by a raster suitable for **rawtoppm**.

-rowskip

Skip this many bytes at the end of each row of the raster. (Some input streams have padding at the end of rows).

-rgb -rbg -grb -gbr -brg -bgr

This option specifies the order of the color components for each pixel. The default is **-rgb**.

-interpixel -interrow

These options specify how the colors are interleaved. The default is **-interpixel**, meaning interleaved by pixel. A byte of red, a byte of green, and a byte of blue, or whatever color order you specified. **-interrow** means interleaved by row - a row of red, a row of green, a row of blue, assuming standard rgb color order. An **-interplane** option - all the red pixels, then all the green, then all the blue - would be an obvious extension, but is not implemented. You could get the same effect by splitting the file into three parts (perhaps using **dd**), turning each part into a PGM file with **rawtopgm**, and then combining them with **rgb3toppm**.

SEE ALSO

ppm(1), **rawtopgm**(1), **rgb3toppm**(1), **pamflip**(1)

AUTHOR

Table Of Contents

NAME

rgb3toppm - combine three PGM images (R, G, B) into one PPM image

SYNOPSIS

rgb3toppm

redpgmfile greenpgmfile bluepgmfile

DESCRIPTION

This program is part of **Netpbm**(1).

rgb3toppm reads three PGM images as input, taking each to represent one component (red, green, and blue) of a color image. It produces a PPM image as output.

SEE ALSO

ppmtorgb3(1), **pamstack**(1), **pgmtoppm**(1), **ppmtopgm**(1), **ppm**(1), **pgm**(1)

AUTHOR

Table Of Contents

NAME

rletopnm - convert a Utah Raster Tools RLE image file to a PNM image file.

SYNOPSIS

rletopnm

[**--alphaout=**{*alpha-filename*,**-**}] [**--headerdump**|**-h**]

[**--verbose**|**-v**]

[**--plain**|**-p**]

[*rlefile*|**-**]

All options may be abbreviated to their minimum unique abbreviation and options and arguments may be in any order.

DESCRIPTION

This program is part of **Netpbm**(1).

rletopnm converts Utah Raster Toolkit RLE image files to PNM image files. **rletopnm** handles 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). **rletopnm** generates a PPM file for all these cases except for the Grayscale file, for which **rletopnm** generates a PGM file.

rlefile is the RLE input file. If it is absent or **-**, the input comes from Standard Input.

OPTIONS

--alphaout=*alpha-filename*

rletopnm creates a PGM (portable graymap) file containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **--alphaout**, **rletopnm** does not generate an alpha file, and if the input image has an alpha channel, **rletopnm** simply discards it.

If you specify **-** as the filename, **rletopnm** writes the alpha output to Standard Output and discards the image.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

--verbose

This option causes **rletopnm** to operate in verbose mode. It prints messages about what it's doing, including the contents of the RLE image header, to Standard Error.

--headerdump

This option causes **rletopnm** to operate in header dump mode. It prints the contents of the RLE image header to Standard Error, but does not produce any other output.

--plain This option causes the PNM output file to be in the 'plain' (text) format, instead of the default 'raw' (binary) format. See **ppm**(1)**and pgm**(1)**for** details on the difference.

EXAMPLES

rletopnm --verbose lenna.rle >lenna.ppm

While running in verbose mode, convert lenna.rle to PPM format and store the resulting image as lenna.ppm.

rletopnm --headerdump file.rle

Dump the header information of the RLE file called file.rle.

rletopnm --alphaout=dartalpha.pgm dart.rle >dart.ppm

Convert RLE file dart.rle to PPM format as dart.ppm. Store the alpha channel of dart.rle as dartalpha.pgm (if dart.rle doesn't have an alpha channel, store a fully transparent alpha mask as dartalpha.pgm).

SEE ALSO

pnmtorle(1), **pnmconvol**(1), **pnm**(1), **ppm**(1), **pgm**(1),

AUTHOR

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

Modifications by Eric Haines to support raw and plain formats.

Modifications by Bryan Henderson to create alpha files and use mnemonic options.

Table Of Contents

NAME

sbigtopgm - convert an SBIG CCDOPS file to PGM

SYNOPSIS

sbigtopgm

[*sbigfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

sbigtopgm reads an an image file in the native format used by the Santa Barbara Instrument Group (SBIG) astronomical CCD cameras, and produces a PGM image as output. Additional information on SBIG cameras and documentation of the file format is available at the Web site, http://www.sbig.com/ .

SEE ALSO

pgm(1)

AUTHOR

John Walker (**http://www.fourmilab.ch/**), January 1998. This program is in the public domain.

Table Of Contents

NAME

sgitopnm - convert a SGI image file to PNM

SYNOPSIS

sgitopnm

[**-verbose**]

[**-channel** *c*]

[*SGIfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

sgitopnm reads an SGI image file as input and produces a PGM image for a 2-dimensional (1 channel) input file, and a PPM image for a 3-dimensional (3 or more channels) input file.

Alternatively, the program produces a PGM image of any one of the channels in the input file.

OPTIONS

-verbose

Give some information about the SGI image file.

-channel *c*

Extract channel *c* of the image as a PGM image. Without this option, **sgitopnm** extracts the first 3 channels as a PPM image or, if the input has only 1 channel, extracts that as a PGM image, and if the input has 2 channels, fails.

REFERENCES

SGI Image File Format documentation (draft v0.95) by Paul Haeberli (*paul@sgi.com*). Available via ftp at sgi.com:graphics/SGIIMAGESPEC.

SEE ALSO

pnm(1), **pnmtosgi**(1)

AUTHOR

Table Of Contents

NAME

sirtopnm - convert a Solitaire file to PNM

SYNOPSIS

sirtopnm

[*sirfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

sirtopnm reads a Solitaire Image Recorder file as input and produces a PNM image as output. The type of the output file depends on the input file - if it's an MGI TYPE 17 file, **sirtopnm** generates a PGM file. If it's an MGI TYPE 11 file, **sirtopnm** generates PPM. The program tells you which type it is writing.

SEE ALSO

pnmtosir(1), **pnm**(1)

AUTHOR

Table Of Contents

NAME

sldtoppm - convert an AutoCAD slide file to a PPM image

SYNOPSIS

sldtoppm [**-adjust**] [**-dir**] [{**-height**|**-ysize**} *s*] [**-info**] [{**-lib**|**-Lib**} *name*] [**-scale** *s*] [**-verbose**] [{**-width**|**-xsize**} *s*] [*slidefile*]

All options can be abbreviated to their shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

sldtoppm reads an AutoCAD® slide file and outputs a PPM image. If you don't specify *slidefile*, **sldtoppm** reads input from Standard Input. **sldtoppm** uses the **ppmdraw** library to convert the vector and polygon information in the slide file to a raster; see the file **ppmdraw.h** for details on this package.

OPTIONS

-adjust If the display on which the slide file was created had non-square pixels, when you process the slide with **sldtoppm** and don't specify **-adjust**, **sldtoppm** issues the following warning;

Warning - pixels on source screen were non-square.

Specifying **-adjust** will correct the image width to compensate. Specifying the **-adjust** option causes **sldtoppm** to scale the width of the image so that pixels in the resulting portable pixmap are square (and hence circles appear as true circles, not ellipses). The scaling is performed in the vector domain, before scan converting the objects. The results are, therefore, superior in appearance to what you'd obtain were you to perform the equivalent scaling with **pamscale** after the bitmap had been created.

-dir The input is assumed to be an AutoCAD slide library file. A directory listing each slide in the library is printed on standard error.

-height *size*
Scales the image in the vector domain so it is *size* pixels in height. If you don't specify **-width** or **-xsize**, **sldtoppm** adjusts the width to preserve the pixel aspect ratio.

-info Dump the slide file header on standard error, displaying the original screen size and aspect ratio among other information.

-lib *name*
Extracts the slide with the given *name* from the slide library given as input. **sldtoppm** converts the specified *name* to upper case.

-Lib *name*
Extracts the slide with the given *name* from the slide library given as input. **sldtoppm** uses *name* in the case specified; it does not convert it to upper case.

-scale *s* Scales the image by factor *s*, which may be any floating point value greater than zero. **sldtoppm** does the scaling after aspect ratio adjustment, if any. Since it does the scaling in the vector domain, before rasterisation, the results look much better than running the output of **sldtoppm** through **pamscale**.

-verbose

Dumps the slide file header and lists every vector and polygon to Standard Error.

-width *size*

Scales the image in the vector domain so it is *size* pixels wide. If you don't specify **-height** or **-ysize**, **sldtoppm** adjusts the height to preserve the pixel aspect ratio.

-xsize *size*

Scales the image in the vector domain so it is *size* pixels wide. If you don't specify **-height** or **-ysize**, **sldtoppm** adjusts the height to preserve the pixel aspect ratio.

-ysize *size*

Scales the image in the vector domain so it is *size* pixels in height. If you don't specify **-width** or **-xsize**, **sldtoppm** adjusts the width to preserve the pixel aspect ratio.

LIMITATIONS

sldtoppm can convert only Level 2 slides. Level 1 format has been obsolete since the advent of AutoCAD Release 9 in 1987, and was not portable across machine architectures.

Slide library items with names containing 8 bit (such as ISO) or 16 bit (Kanji, for example) characters may not be found when chosen with the **-lib** option unless **sldtoppm** was built with character set conversion functions appropriate to the locale. You can always retrieve slides from libraries regardless of the character set by using the **-Lib** option and specifying the precise name of library member. Use the **-dir** option to list the slides in a library if you're unsure of the exact name.

SEE ALSO

AutoCAD Reference Manual: *Slide File Format*, **pamscale**(1), **ppm**(1)

AUTHOR

John Walker
Autodesk SA
Avenue des Champs-Montants 14b
CH-2074 MARIN
Suisse/Schweiz/Svizzera/Svizra/Switzerland
Usenet:*kelvin@Autodesk.com*
Fax:038/33 88 15
Voice:038/33 76 33

Table Of Contents

NAME

spctoppm - convert an Atari compressed Spectrum file to a PPM

SYNOPSIS

spctoppm

[*spcfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

spctoppm reads an Atari compressed Spectrum file as input and produces a PPM image as output.

SEE ALSO

sputoppm(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

spottopgm - convert SPOT satellite images to a PGM image

SYNTAX

spottopgm [**-1**|**-2**|**-3**] [*Firstcol Firstline Lastcol Lastline*] *inputfile*

OPTIONS

-1

-2

-3 Extract the given color from the SPOT image. The colors are infrared, visible light, and ultraviolet, although I don't know which corresponds to which number. If the image is in color, **spottopgm** announces this on Standard Error. The default color is 1.

PARAMETERS

Firstcol Firstline Lastcol Lastline

Extract the specified rectangle from the SPOT image. Most SPOT images are 3000 lines long and 3000 or more columns wide. Unfortunately, the SPOT format only gives the width and not the length. The width is printed on standard error. The default rectangle is the width of the input image by 3000 lines.

DESCRIPTION

This program is part of **Netpbm**(1).

spottopgm converts the named **inputfile** to PGM format, defaulting to the first color and the whole SPOT image unless you specify otherwise with the options.

LIMITATIONS

spottopgm doesn't determine the length of the input file; this would involve two passes over the input file. It defaults to 3000 lines instead.

spottopgm could extract a three-color image (as a PPM), but I didn't feel like making the program more complicated than it is now. Besides, there is no one-to-one correspondence between red, green, blue and infrared, visible and ultraviolet.

I've had only a limited number of SPOT images to play with, and therefore wouldn't guarantee that this will work on any other images.

AUTHOR

Warren Toomey *wkt@csadfa.cs.adfa.oz.au*

SEE ALSO

pgm(1)

Table Of Contents

NAME

sputoppm - convert an Atari uncompressed Spectrum file to a PPM image

SYNOPSIS

sputoppm

[*spufile*]

DESCRIPTION

This program is part of **Netpbm**(1).

sputoppm reads an Atari uncompressed Spectrum file as input and produces a PPM image as output.

SEE ALSO

spctoppm(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

tgatoppm - convert TrueVision Targa file to a PPM image

SYNOPSIS

tgatoppm

[**--alphaout=**{*alpha-filename*,**-**}]

[**--headerdump**]

tga-filename

All options can be abbreviated to their shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

tgatoppm reads a TrueVision Targa file as input and produces a PPM image as output.

OPTIONS

--alphaout=*alpha-filename*

tgatoppm creates a PGM image containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **--alphaout**, **tgatoppm** does not generate an alpha file, and if the input image has an alpha channel, **tgatoppm** simply discards it.

If you specify **-** as the filename, **tgatoppm** writes the alpha output to Standard Output and discards the image.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

--headerdump

Causes **tgatoppm** to dump information from the TGA header to Standard Error.

SEE ALSO

ppmtotga(1), **pamcomp**(1), **ppm**(1)

AUTHOR

Partially based on tga2rast, version 1.0, by Ian J. MacPhedran.

Table Of Contents

NAME

thinkjettopbm - convert HP ThinkJet printer commands file to PBM

SYNOPSIS

thinkjettopbm

[**-d**]

[*thinkjet_file*]

DESCRIPTION

This program is part of **Netpbm**(1).

thinkjettopbm reads HP ThinkJet printer commands from the standard input, or *thinkjet_file* if specified, and writes a PBM image to Standard Output.

thinkjettopbm silently ignores text and non-graphics command sequences.

The **-d** option turns on debugging messages which are written to the standard error stream.

LIMITATIONS

The program handles only a small subset of ThinkJet command sequences, but enough to convert screen images from older HP test equipment.

SEE ALSO

pnmtopclx.html(1), **pbmtolj.html**(1), **ppmtopj**(1), **ppmtopj**(1), **thinkjettopbm**(1), **pbm**(1), **pjtoppm**(1)

AUTHOR

Table Of Contents

NAME

tifftopnm - convert a TIFF file into a PNM image

SYNOPSIS

tifftopnm

[**-alphaout=**{*alpha-filename*,**-**}] [**-headerdump**] [**-respectfillorder**] [**-byrow**] [*tiff-filename*]

You may abbreviate any option to its shortest unique prefix. You may use two hyphens instead of one in options. You may separate an option and its value either by an equals sign or white space.

DESCRIPTION

This program is part of **Netpbm**(1).

tifftopnm reads a TIFF file as input and produces a PNM image as output. The type of the output file depends on the input file - if it's black & white, generates a PBM image; if it's grayscale, generates a PGM image; otherwise, a PPM image. The program tells you which type it is writing.

If the TIFF file contains multiple images (multiple 'directories,' **tifftopnm** generates a multi-image PNM output stream. Before Netpbm 10.27 (March 2005), however, it would just ignore all but the first input image.

This program cannot read every possible TIFF file -- there are myriad variations of the TIFF format. However, it does understand monochrome and gray scale, RGB, RGBA (red/green/blue with alpha channel), CMYK (Cyan-Magenta-Yellow-Black ink color separation), and color palette TIFF files. An RGB file can have either single plane (interleaved) color or multiple plane format. The program reads 1-8 and 16 bit-per-sample input, the latter in either bigendian or littlendian encoding. Tiff directory information may also be either bigendian or littendian.

There are many TIFF formats that **tifftopnm** can read only if the image is small enough to fit in memory. **tifftopnm** uses the TIFF library's TIFFRGBAImageGet() function to process the TIFF image if it can get enough memory for TIFFRGBAImageGet() to store the whole image in memory at once (that's what TIFFRGBAImageGet() does). If not, **tifftopnm** uses a more primitive row-by-row conversion strategy using the raw data returned by TIFFReadScanLine() and native intelligence. That native intelligence does not know as many formats as TIFFRGBAImageGet() does. And certain compressed formats simply cannot be read with TIFFReadScanLine().

Before Netpbm 10.11 (October 2002), **tifftopnm** never used TIFFRGBAImageGet(), so it could not interpret many of the formats it can interpret today.

There is no fundamental reason that this program could not read other kinds of TIFF files even when they don't fit in memory all at once. The existing limitations are mainly because no one has asked for more.

The PNM output has the same maxval as the Tiff input, except that if the Tiff input is colormapped (which implies a maxval of 65535) the PNM output has a maxval of 255. Though this may result in lost information, such input images hardly ever actually have more color resolution than a maxval of 255 provides and people often cannot deal with PNM files that have maxval > 255. By contrast, a non-colormapped Tiff image that doesn't need a maxval > 255 doesn't *have* a maxval > 255, so when **tifftopnm** sees a non-colormapped maxval > 255, it takes it seriously and produces a matching output maxval.

Another exception is where the TIFF maxval is greater than 65535, which is the maximum allowed by the Netpbm formats. In that case, **tifftopnm** uses a maxval of 65535, and you lose some information in the conversion.

The *tiff-filename* argument names the regular file that contains the Tiff image. If you specify '-' or don't specify this argument, **tfftopnm** uses Standard Input. In either case, the file must be seekable. That means no pipe, but any regular file is fine.

OPTIONS

-alphaout=*alpha-filename*

tifftopnm creates a PGM file containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **-alphaout**,

tifftopnm does not generate an alpha file, and if the input image has an alpha channel, **tifftopnm** simply discards it.

If you specify **-** as the filename, **tifftopnm** writes the alpha output to Standard Output and discards the image.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

-respectfillorder

By default, **tifftopnm** ignores the 'fillorder' tag in the TIFF input, which means it may incorrectly interpret the image. To make it follow the spec, use this option. For a lengthy but engaging discussion of why **tifftopnm** works this way and how to use the **-respectfillorder** option, see the note on fillorder below.

-byrow This option can make **tifftopnm** run faster.

tifftopnm has two different ways to do the conversion from Tiff to PNM, using two different facilities of the TIFF library:

Whole Image

Decode the entire image into memory at once, using TIFFRGBAImageGet(), then convert to PNM and output row by row.

Row By Row

Read, convert, and output one row at a time using TIFFReadScanline().

Whole Image is preferable because the Tiff library does more of the work, which means it understands more of the Tiff format possibilities now and in the future. Also, some compressed TIFF formats don't allow you to extract an individual row.

Row By Row uses far less memory, which means with large images, it can run in environments where Whole Image cannot and may also run faster. And because Netpbm code does more of the work, it's possible that it can be more flexible or at least give better diagnostic information if there's something wrong with the TIFF.

In Netpbm, we stress function over performance, so by default we try Whole Image first, and if we can't get enough memory for the decoded image or TIFFRGBAImageGet() fails, we fall back to Row By Row. But if you specify the **-byrow** option, **tifftopnm** will not attempt Whole Image. If Row By Row does not work, it simply fails.

See Color Separation (CMYK) TIFFs for a description of one way Row By Row makes a significant difference in your results.

Whole Image costs you precision when your TIFF image uses more than 8 bits per sample. TIFFRGBAImageGet() converts the samples to 8 bits. **tifftopnm** then scales them back to maxval 65535, but the lower 8 bits of information is gone.

Before Netpbm 10.11 (October 2002), **tifftopnm** always did Row By Row. Netpbm 10.12 always tried Whole Image first. **-byrow** came in with Netpbm 10.13 (January 2003).

-headerdump

Dump TIFF file information to stderr. This information may be useful in debugging TIFF file conversion problems.

NOTES

Fillorder

There is a piece of information in the header of a TIFF image called 'fillorder.' The TIFF specification quite clearly states that this value tells the order in which bits are arranged in a byte in the description of the image's pixels. There are two options, assuming that the image has a format where more than one pixel can be represented by a single byte: 1) the byte is filled from most significant bit to least significant bit going left to right in the image; and 2) the opposite.

However, there is confusion in the world as to the meaning of fillorder. Evidence shows that some people believe it has to do with byte order when a single value is represented by two bytes.

These people cause TIFF images to be created that, while they use a MSB-to-LSB fillorder, have a fillorder tag that says they used LSB-to-MSB. A program that properly interprets a TIFF image will not end up with the image that the author intended in this case.

For a long time, **tifftopnm**

did not understand fillorder itself and assumed the fillorder was MSB-to-LSB regardless of the fillorder tag in the TIFF header. And as far as I know, there is no legitimate reason to use a fillorder other than MSB-to-LSB. So users of **tifftopnm**

were happily using those TIFF images that had incorrect fillorder tags.

So that those users can continue to be happy, **tifftopnm**

today continues to ignore the fillorder tag unless you tell it not to. (It does, however, warn you when the fillorder tag does not say MSB-to-LSB that the tag is being ignored).

If for some reason you have a TIFF image that actually has LSB-to-MSB fillorder, and its fillorder tag correctly indicates that, you must use the **-respectfillorder**

option on **tifftopnm**

to get proper results.

Examples of incorrect TIFF images are at ftp://weather.noaa.gov. They are apparently created by a program called **faxtotiff**.

This note was written on January 1, 2002.

Color Separation (CMYK) TIFFs

Some TIFF images contain color information in CMYK form, whereas PNM images use RGB. There are various formulas for converting between these two forms, and **tifftopnm** can use either of two.

The TIFF library (Version 3.5.4 from libtiff.org) uses Y=(1-K)*(1-B) (similar for R and G) in its TIFFRGBAImageGet() service. When **tifftopnm** works in Whole Image mode, it uses that service, so that's the conversion you get.

But when **tifftopnm** runs in Row By Row mode, it does not use TIFFRGBAImageGet(), and you get what appears to be more useful: Y=1-(B+K). This is the inverse of what **pnmtotiffcmyk** does.

See the **-byrow** option for more information on Whole Image versus Row By Row mode.

Before Netpbm 10.21 (March 2004), **tifftopnm** used the Y=(1-K)*(1-B) formula always.

SEE ALSO

pnmtotiff(1), **pnmtotiffcmyk**(1), **pamcomp**(1), **pnm**(1)

AUTHOR

Derived by Jef Poskanzer from tif2ras.c, which is Copyright (c) 1990 by Sun Microsystems, Inc. Author: Patrick J. Naughton (*naughton@wind.sun.com*).

NAME

vidtopppm - archived but not working

DESCRIPTION

This program is part of **Netpbm**(1).

vidtoppm is supposed to convert Parallax XVideo JPEG to sequence of PPM files. It's source code is part of the Netpbm package for archival purposes, but it does not presently build because it requires header files that aren't in the package. If someone comes up with a use for **vidtoppm**, we might be able to make it work.

vidtoppm was in the Berkeley MPEG encoder package from which Netpbm's **ppmtompeg** was derived.

Table Of Contents

NAME

wbmptopbm - convert a wireless bitmap (wbmp) file to a PBM

SYNOPSIS

wbmptopbm

[*wbmpfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

wbmptopbm reads a wbmp file as input and produces a PBM image as output.

LIMITATIONS

wbmptopbm recognizes only WBMP type 0. This is the only type specified in the WAP 1.1 specifications.

SEE ALSO

pbm(1), **pbmtowbmp**(1),

Wireless Application Environment Specification.

AUTHOR

Table Of Contents

NAME

winicontoppm - convert a Windows .ico image into 1 or more PPM images

SYNOPSIS

winicontoppm [**-writeands**] [**-allicons**|**-bestqual**] [**-multippm**] [**-verbose**] [*iconfile*] [*ppmdestfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

winicontoppm reads a Microsoft Windows .ico file and converts it to one or more PPMs.

A Windows icon contains one or more images, at different resolutions and color depths. Each image has an 'and' mask, which contains transparancy data.

By default, the output goes to Standard Output. If you specify *ppmdestfile*, output goes into one or more files named as follows. If it's just one file (i.e. you specify the **-multippm** option or don't specify **-allicons**), the file name is *ppmdestfile***.ppm**. If it's multiple files, their file names are *ppmdestfile***_1.ppm**, *ppmdestfile***_2.ppm**, etc.

When you specify the **-writeands** option, the file names above are modified to include the string **xor** as in *ppmdestfile***_xor.ppm** or *ppmdestfile***_xor_1.ppm**.

winicontoppm can convert .ico images with 1, 4, 8, 24, or 32 bits per pixel. Before Netpbm 10.15 (April 2003), it could not handle 24 and 32.

OPTIONS

-writeands

For each icon written, also write the 'and' (transparancy) mask as a separate PBM file. It's name is of the form *ppmdestfile***_and.pbm** or *ppmdestfile***_and_1.pbm**.

-allicons

Extract all images from the .ico file.

-bestqual

Extract only the best quality (largest, then highest bpp) image from the .ico file.

-multippm

Write all PPMs to a single file.

SEE ALSO

ppmtowinicon(1), **bmptopnm**(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

xbmtopbm - convert an X11 or X10 bitmap to a PBM image

SYNOPSIS

xbmtopbm

[*bitmapfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

xbmtopbm reads an X11 or X10 bitmap as input and produces a PBM image as output.

SEE ALSO

pbmtoxbm(1), **pbmtox10bm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

ximtoppm - convert an Xim file to a PPM image

SYNOPSIS

ximtoppm

[**--alphaout=**{*alpha-filename*,**-**}] [*ximfile*]

You can abbreviate any option to its shortest unique prefix.

DESCRIPTION

This program is part of **Netpbm**(1).

ximptoppm reads an Xim file as input and produces a PPM image as output. The Xim toolkit is included in the contrib tree of the X.V11R4 release.

OPTIONS

--alphaout=*alpha-filename*

ximtoppm creates a PGM file containing the alpha channel values in the input image. If the input image doesn't contain an alpha channel, the *alpha-filename* file contains all zero (transparent) alpha values. If you don't specify **--alphaout**, **ximtoppm** does not generate an alpha file, and if the input image has an alpha channel, **ximtoppm** simply discards it.

If you specify - as the filename, **ximtoppm** writes the alpha output to Standard Output and discards the image.

Actually, an Xim image can contain an arbitrary fourth channel -- it need not be an Alpha channel. **ximtoppm** extracts any fourth channel it finds as described above; it doesn't matter if it is an alpha channel or not.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

SEE ALSO

pamcomp(1), **ppm**(1)

AUTHOR

Table Of Contents

NAME

xpmtoppm - convert an X11 pixmap to a PPM image

SYNOPSIS

xpmtoppm

[**--alphaout=**{*alpha-filename*,**-**}] [**-verbose**]

[*xpmfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

xpbtoppm reads an X11 pixmap (XPM version 1 or 3) as input and produces a PPM image as output.

OPTIONS

--alphaout=*alpha-filename*

xpmtoppm creates a PBM file containing the transparency mask for the image. If the input image doesn't contain transparency information, the *alpha-filename* file contains all white (opaque) alpha values. If you don't specify **--alphaout**, **xpmtoppm** does not generate an alpha file, and if the input image has transparency information, **xpmtoppm** simply discards it.

If you specify **-** as the filename, **xpmtoppm** writes the alpha output to Standard Output and discards the image.

See **pamcomp**(1)**for**one**way**to**use** the alpha output file.

--verbose

xpmtoppm prints information about its processing on Standard Error.

LIMITATIONS

The recogized XPM version 3 features are limited. Comments can only be single lines and there must be for every pixel a default colorname for a color type visual.

SEE ALSO

ppmtoxpm(1), **pamcomp**(1), **ppm**(1)

AUTHOR

Upgraded to work with XPM version 3 by Arnaud Le Hors <*lehors@mirsa.inria.fr*>, Tue Apr 9 1991.

Table Of Contents

NAME

xvminitoppm - convert a XV ’thumbnail’ picture to PPM

SYNOPSIS

xvminitoppm

[*xvminipic*]

DESCRIPTION

This program is part of **Netpbm**(1).

xvminittoppm reads a XV ’thumbnail’ picture (a miniature picture generated by the ’VisualSchnauzer’ browser) as input and produces PPM image as output.

SEE ALSO

ppm(1), **xv** manual

AUTHOR

Table Of Contents

NAME

xwdtopnm - convert an X11 or X10 window dump file to a PNM image

SYNOPSIS

xwdtopnm [**-verbose**] [**-headerdump**] [*xwdfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

xwdtopnm reads an X11 or X10 window dump file as input and produces a PNM image as output. The type of the output image depends on the input file - if it's black and white, the output is PBM. If it's grayscale, the output is PGM. Otherwise, it's PPM. The program tells you which type it is writing.

Using this program, you can convert anything you can display on an X workstation's screen into a PNM image. Just display whatever you're interested in, run the **xwd** program to capture the contents of the window, run it through **xwdtopnm**, and then use **pamcut** to select the part you want.

Note that a pseudocolor XWD image (typically what you get when you make a dump of a pseudocolor X window) has maxval 65535, which means the PNM file that **xwdtopnm** generates has maxval 65535. Many older image processing programs (that aren't part of the Netpbm package and don't use the Netpbm programming library) don't know how to handle a PNM image with maxval greater than 255 (because there are two bytes instead of one for each sample in the image). So you may want to run the output of **xwdtopnm** through **pnmdepth** before feeding it to one of these old programs.

OPTIONS

-verbose

This option causes **xwdtopnm** to display handy information about the input image and the conversion process

-headerdump

This option causes **xwdtopnm** to display the contents of the X11 header. It has no effect when the input is X10. This option was new in Netpbm 10.26 (December 2004).

SEE ALSO

pnmtoxwd(1), **pnm**(1), **xwd** man page

AUTHOR

Table Of Contents

NAME

ybmtopbm - convert a Bennet Yee ’face’ file to PBM

SYNOPSIS

ybmtopbm

[*facefile*]

DESCRIPTION

This program is part of **Netpbm**(1).

ymtopbm reads a file acceptable to the **face** and **xbm** programs by Bennet Yee (*bsy+@cs.cmu.edu*). and writes a PBM image as output.

SEE ALSO

pbmtoybm(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

yuvsplittoppm - convert separate Y, U, and V files into a PPM image

SYNOPSIS

yuvsplittoppm

basename width height [**-ccir601**]

DESCRIPTION

This program is part of **Netpbm**(1).

yuvsplittoppm reads three files, containing the YUV components, as input. These files are *basename*.Y, *basename*.U, and *basename*.V. Produces a portable pixmap on stdout.

Since the YUV files are raw files, the dimensions *width* and *height* must be specified on the command line.

OPTIONS

-ccir601

Assumes that the YUV triplets are scaled into the smaller range of the CCIR 601 (MPEG) standard. Else, the JFIF (JPEG) standard is assumed.

SEE ALSO

ppmtoyuvsplit(1), **yuvtoppm**(1), **ppm**(1)

AUTHOR

Marcel Wijkstra <*wijkstra@fwi.uva.nl*>, based on **ppmtoyuvsplit**.

Table Of Contents

NAME

yuvtoppm - convert Abekas YUV bytes to PPM

SYNOPSIS

yuvtoppm

width height [*imagedata*]

DESCRIPTION

This program is part of **Netpbm**(1).

yuvtoppm reads raw Abekas YUV bytes as input and produces a PPM image as output. The input file is just YUV bytes. You have to specify the width and height on the command line, since the program obviously can't get them from the file. **yuvotppm** assumes the maxval of the input is 255.

SEE ALSO

ppmtoyuv(1), **ppm**(1)

AUTHOR

Marc Boucher <*marc@PostImage.COM*>, based on Example Conversion Program, A60/A64 Digital Video Interface Manual, page 69.

Table Of Contents

NAME

zeisstopnm - convert a Zeiss confocal file to PNM

SYNOPSIS

zeisstopnm

[**-pgm** | **-ppm**]

[*zeissfile*]

DESCRIPTION

This program is part of **Netpbm**(1).

zeisstopnm reads a Zeiss confocal file as input and produces a PNM image as output.

By default, the exact type of the output depends on the input file: If it's grayscale a PGM image; otherwise a PPM. The program tells you which type it is writing. You can override the default with the **-pgm** and **-ppm** options.

OPTIONS

-pgm Force the output to be in PGM format.

-ppm Force the output to be in PPM format.

SEE ALSO

pnm(1)

AUTHOR

NAME

libnetpbm − general introduction to the netpbm library

DESCRIPTION

libnetpbm is a C programming library for reading, writing, and manipulating Netpbm images. It also contains a few general graphics manipulation tools, but it is not intended to be a graphics tools library. For graphics tools, Netpbm expects you to run the Netpbm programs. From a C program, the **libnetpbm** function **pm_system()** makes this easy. However, since it creates a process and execs a program, this may be too heavyweight for some applications.

To use **libnetpbm** services in your C program, #include the **pam.h** interface header file. For historical reasons, you can also get by in some cases with **pbm.h**, **pgm.h**, **ppm.h**, or **pnm.h**, but there's really no point to that anymore.

The **libnetpbm** functions are divided into these categories:

- PBM functions. These have names that start with **pbm** and work only on PBM images.
- PGM functions. These have names that start with **pgm** and work only on PGM images.
- PPM functions. These have names that start with **ppm** and work only on PPM images.
- PNM functions. These have names that start with **pnm** and work on PBM, PGM, and PPM images.
- PAM functions. These also have names that start with **pnm** and work on all the Netpbm image types.
- PM functions. These are utility functions that aren't specific to any particular image format.

For new programming, you rarely need to concern yourself with the PBM, PGM, PPM, and PNM functions, because the newer PAM functions do the same thing and are easier to use. For certain processing of bi-level images, the PBM functions are significantly more efficient, though.

libnetpbm has a backward compatibility feature that means a function designed to read one format can read some others too, converting on the fly. In particular, a function that reads a PGM image will also read a PBM image, but converts it as it reads it so that for programming purposes, it is a PGM image. Similarly, a function that reads PPM can read PBM and PGM as well. And a function that reads PBM, PGM, or PPM can read a PAM that has an equivalent tuple type.

For each of the five classes of **libnetpbm** image processing functions, **libnetpbm** has in in-memory representation for a pixel, a row, and a whole image. Do not confuse this format with the actual image format, as you would see in a file. The **libnetpbm** in-memory format is designed to make programming very easy. It is sometimes extremely inefficient, even more than the actual image format. For example, a pixel that a PPM image represents with 3 bytes, **libnetpbm**'s PAM functions represent with 16 bytes. A pixel in a PBM image is represented by a single bit, but the PNM functions represent that pixel in memory with 96 bits.

See **Libnetpbm**User's**Manual**(1)**for**the basics on using **libnetpbm** in a program.

You can look up the reference information for a particular function in **The**libnetpbm**Directory**(1).

Before Netpbm release 10 (June 2002), this library was split into four: libpbm, libpgm, libppm, and libpnm. That's largely the reason for the multiple sets of functions and scattered documentation.

Table Of Contents

NAME

libnetpbm_image − overview of netpbm image-processing functions

DESCRIPTION

This reference manual covers functions in the **libnetpbm** library for processing images, using the Netpbm image formats and the **libnetpbm** in-memory image formats.

For historical reasons as well as to avoid clutter, it does not cover the largely obsolete PBM, PGM, PPM, and PNM classes of **libnetpbm** functions. For those, see **PBM** Function Manual (1), **PGM**Function**Manual**(1), **PPM**Function**Manual**(1),**and PNM**Function**Manual**(1).**Note**that**you**do*not* need those functions to process PBM, PGM, PPM, and PNM images. The functions in this manual are sufficient for that.

For introductory and general information using **libnetpbm**, see **Libnetpbm**User's**Guide**(1).

libnetpbm also contains functions that are not specifically oriented toward processing image data. Read about those in the **Libnetpbm**Utility**Manual**(1).

To use these services, #include **pam.h**.

Types

Here are some important types that you use with **libnetpbm**:

sample A sample of a Netpbm image. See the format specifications -- as an example, the red intensity of a particular pixel of a PPM image is a sample. This is an integer type.

tuple A tuple from a PAM image or the PAM equivalent of a PNM image. See the PAM format specification -- as an example, a pixel of a PPM image would be a tuple. A tuple is an array of samples.

samplen
Same as **sample**, except in normalized form. This is a floating point type with a value in the range 0..1. 0 corresponds to a PAM/PNM sample value of 0. 1 corresponds to a PAM/PNM sample value equal to the image's maxval.

tuplen The same as **tuple**, except composed of normalized samples (**samplen**) intead of regular samples (**sample**).

The main argument to most of the PAM functions is the address of a **pam** structure, which is defined as follows:

struct pam { int *size* **int** *len* **FILE ***file* **int** *format* **int** *plainformat* **int** *height* **int** *width* **int** *depth* **sample** *maxval* **int** *bytes_per_sample* **char** *tuple_type***[256]; }**

See The Libnetbm User's Guide for information on the **pam** structure.

Macros

PNM_MAXMAXVAL is the maximum maxval that Netpbm images could historically have: 255. Many programs aren't capable of handling Netpbm images with a maxval larger than this. It's named this way for backward compatibility -- it had this name back when it was *the* maximum maxval.

PNM_OVERALLMAXVAL is the maximum maxval that Netpbm images can have today (65535).

PBM_FORMAT, **RPBM_FORMAT**, **PGM_FORMAT**, **RPGM_FORMAT**, **PPM_FORMAT**, **RPPM_FORMAT**, and **PAM_FORMAT** are the format codes of the various Netpbm formats. **RPBM_FORMAT** is the raw PBM format and **PBM_FORMAT** is the plain PBM format, and so on. See the *format* member of the **pam** structure .

PAM_FORMAT_TYPE(*format***)** gives the type of a format, given the format code. The types of formats are PBM, PGM, PPM, and PAM and macros for the type codes are, respectively, PBM_TYPE, PGM_TYPE, PPM_TYPE, and PAM_TYPE. Note that there are more format codes then there are format types because there are different format codes for the plain and raw subformats of each format.

Functions

These interfaces are declared in **pam.h**.

Memory Management

Synopsis

tuple ** pnm_allocpamarray(struct pam *_pamP_);

tuple * pnm_allocpamrow(struct pam *_pamP_);

void pnm_freepamarray(tuple **_tuplearray_, struct pam *_pamP_);

void pnm_freepamrow(tuple *_tuplerow_);

tuple * allocpamtuple(struct pam *_pamP_);

void pnm_freepamtuple(tuple *tuple* **);**

tuplen * pnm_allocpamrown(struct pam *_pamP_);

void pnm_freepamrown(tuple *_tuplenrow_);

Description

pnm_allocpamarray() allocates space for an array of tuples. **pnm_freepamarray()** frees an array space allocated by **pnm_allocpamarray()** or **pnm_readpam()**.

pnm_allocpamrow() allocates space for a row of a PAM image, in basic form. **pnm_freepamrow()** frees it.

pnm_allocpamrown() is the same as **pnm_allocpamrow()** except that it allocates space for a PAM row in the normalized form. **pnm_freepamrown()** is similarly like **pnm_freepamrow**.

Reading Netpbm Files

Synopsis

void pnm_readpaminit(FILE *_file_, struct pam *_pamP_, int *size***);**

void pnm_readpamrow(struct pam *_pamP_, tuple *_tuplerow_);

tuple ** pnm_readpam(FILE *_file_, struct pam *_pamP_, int *size***);**

void pnm_readpamrown(struct pam *_pamP_, tuplen *_tuplenrow_);

Description

pnm_readpaminit() reads the header of a Netpbm image.

See above for a general description of the *pamP* argument.

pnm_readpaminit() returns the information from the header in the ***pamP* structure. It does not require any members of ***pamP* through **tuple_type** to be set at invocation, and sets all of those members. It expects all members after **tuple_type** to be meaningful.

size is the size of the ***pamP* structure as understood by the program processing the image. **pnm_readpaminit()** does not attempt to use or set any members of the structure beyond that. The point of this argument is that the definition of the structure may change over time, with additional fields being added to the end. This argument allows **pnm_readpaminit** to distinguish between a new program that wants to exploit the additional features and an old program that cannot (or a new program that just doesn't want to deal with the added complexity). At a minimum, this size must contain the members up through **tuple_type**. You should use the **PAM_STRUCT_SIZE** macro to compute this argument. E.g. **PAM_STRUCT_SIZE(tuple_type)**.

The function expects to find the image file positioned to the start of the header and leaves it positioned to the start of the raster.

pnm_readpamrow() reads a row of the raster from a Netpbm image file. It expects all of the members of the ***pamP** structure to be set upon invocation and does not modify any of them. It expects to find the file positioned to the start of the row in question in the raster and leaves it positioned just after it. It returns the row as the array of tuples *tuplerow*, which must already have its column pointers set up so that it forms a C 2-dimensional array. The leftmost tuple is Element 0 of this array.

pnm_readpam() reads an entire image from a PAM or PNM image file and allocates the space in which to return the raster. It expects to find the file positioned to the first byte of the image and leaves it positioned just after the image.

The function does not require ***pamP* to have any of its members set and sets them all. *size* is the storage size in bytes of the ***pamP* structure, normally **sizeof(struct pam)**.

The return value is a newly allocated array of the rows of the image, with the top row being Element 0 of the array. Each row is represented as **pnm_readpamrow()** would return.

The return value is also effectively a 3-dimensional C array of samples, with the dimensions corresponding to the height, width, and depth of the image, in that order.

pnm_readpam() combines the functions of **pnm_allocpamarray()**, **pnm_readpaminit()**, and iterations of **pnm_readpamrow()**. It may require more dynamic storage than you can afford.

pnm_readpamrown() is like **pnm_readpamrow()** except that it returns the row contents in normalized form (composed of normalized tuples (**tuplen**) instead of basic form (**tuple**).

pnm_readpaminit() and **pnm_readpam** abort the program with a message to Standard Error if the PAM or PNM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

Writing Netpbm Files

Synopsis

void pnm_writepaminit(struct pam *_pamP_);

void pnm_writepamrow(struct pam *_pamP_, const tuple *_tuplerow_);

void pnm_writepam(struct pam *_pamP_, const tuple * const *_tuplearray_);

void pnm_writepamrown(struct pam *_pamP_, const tuplen *_tuplerown_);

Description

pnm_writepaminit() writes the header of a PAM or PNM image and computes some of the fields of the pam structure.

See above for a description of the *pamP* argument.

The following members of the ***pamP* structure must be set upon invocation to tell the function how and what to write. **size**, **len**, **file**, **format**, **height**, **width**, **depth**, **maxval**, **tuple_type**.

pnm_writepaminit() sets the **plainformat** and **bytes_per_sample** members based on the information supplied.

pnm_writepamrow() writes a row of the raster into a PAM or PNM image file. It expects to find the file positioned where the row should start and leaves it positioned just after the row. The function requires all the elements of ***pamP* to be set upon invocation and doesn't modify them.

tuplerow is an array of tuples representing the row. The leftmost tuple is Element 0 of this array.

pnm_writepam() writes an entire PAM or PNM image to a PAM or PNM image file. It expects to find the file positioned to where the image should start and leaves it positioned just after the image.

The following members of the ***pamP* structure must be set upon invocation to tell the function how and what to write: **size**, **len**, **file**, **format**, **height**, **width**, **depth**, **maxval**, **tuple_type**.

pnm_writepam() sets the **plainformat** and **bytes_per_sample** members based on the information supplied.

tuplearray is an array of rows such that you would pass to **pnm_writepamrow()**, with the top row being Element 0 of the array.

pnm_writepam() combines the functions of **pnm_writepaminit()**, and iterations of **pnm_writepamrow()**. Its raster input may be more storage than you can afford.

pnm_writepamrown() is like **pnm_writepamrow()** except that it takes the row contents in normalized form (composed of normalized tuples (**tuplen**) instead of basic form (**tuple**).

Transforming Pixels

Synopsis

```
void pnm_YCbCrtuple( tuple tuple, double *YP, double *CrP, double *CbP);

void pnm_YCbCr_to_rgbtuple( const struct pam * const pamP,
  tuple          const tuple,
  double         const Y,
  double         const Cb,
  double         const Cr,
  int *          const overflowP );

extern double pnm_lumin_factor[3];

void pnm_normalizetuple(
 struct pam * const pamP,
 tuple      const tuple,
 tuplen     const tuplen);

void pnm_unnormalizetuple(
 struct pam * const pamP,
 tuplen     const tuplen,
 tuple      const tuple);

void pnm_normalizeRow(
 struct pam *       const pamP,
 const tuple *      const tuplerow,
 pnm_transformMap * const transform,
 tuplen *           const tuplenrow);

void pnm_unnormalizeRow(
 struct pam *       const pamP,
 const tuplen *     const tuplenrow,
 pnm_transformMap * const transform,
 tuple *            const tuplerow);

void pnm_gammarown(
 struct pam * const pamP,
 tuplen *     const row );

void pnm_ungammarown(
 struct pam * const pamP,
 tuplen *     const row );

void pnm_applyopacityrown(
 struct pam * const pamP,
 tuplen *     const tuplenrow );
```

```
void pnm_unapplyopacityrown(
 struct pam * const pamP,
 tuplen *     const tuplenrow );


pnm_transformMap * pnm_creategammatransform(
 const struct pam * const pamP );

void pnm_freegammatransform(
 const pnm_transformMap * const transform,
 const struct pam *       const pamP );

pnm_transformMap * pnm_createungammatransform(
 const struct pam * const pamP );

void pnm_freeungammatransform(
 const pnm_transformMap * const transform,
 const struct pam *       const pamP );
```

Description

pnm_YCbCrtuple() returns the Y/Cb/Cr luminance/chrominance representation of the color represented by the input tuple, assuming that the tuple is an RGB color representation (which is the case if it was read from a PPM image). The output components are based on the same scale (maxval) as the input tuple, but are floating point nonetheless to avoid losing information due to rounding. Divide them by the maxval to get normalized [0..1] values.

pnm_YCbCr_to_rgbtuple() does the reverse. *pamP* indicates the maxval for the returned *tuple*, and the *Y*, *Cb*, and *Cr* arguments are of the same scale.

It is possible for *Y*, *Cb*, and *Cr* to describe a color that cannot be represented in RGB form. In that case, **pnm_YCbCr_to_rgbtuple()** chooses a color as close as possible (by clipping each component to 0 and the maxval) and sets *overflowP true. It otherwise sets *overflowP false.

pnm_lumin_factor[] is the factors (weights) one uses to compute the intensity of a color (according to some standard -- I don't know which). pnm_lumin_factor[0] is for the red component, [1] is for the green, and [2] is for the blue. They add up to 1.

pnm_gammarown() and **pnm_ungammarown()** apply and unapply gamma correction to a row of an image using the same transformation as **pm_gamma()** and **pm_ungamma()** . Note that these operate on a row of normalized tuples (**tuplen**, not **tuple**).

pnm_applyopacity() reduces the intensity of samples in accordance with the opacity plane of an image. The opacity plane, if it exists, tells how much of the light from that pixel should show when the image is composed with another image. You use **pnm_applyopacity()** in preparation for doing such a composition. For example, if the opacity plane says that the top half of the image is 50% opaque and the bottom half 100% opaque, **pnm_applyopacity()** will reduce the intensity of each sample of each tuple (pixel) in the upper half of the image by 50%, and leave the rest alone.

If the image does not have an opacity plane (i.e. its tuple type is not one that **libnetpbm** recognizes as having an opacity plane), **pnm_applyopacity()** does nothing (which is the same as assuming opacity 100%). The tuple types that **libnetpbm** recognizes as having opacity are **RGB_ALPHA** and **GRAYSCALE_ALPHA**.

pnm_unapplyopacity() does the reverse. It assumes the intensities are already reduced according to the opacity plane, and raises back to normal.

pnm_applyopacity() works on (takes as input and produces as output) *normalized, intensity-proportional* tuples. That means you will typically read the row from the image file with **pnm_readpamrown()** and then gamma-correct it with **pnm_ungammarown()**, and then do **pnm_applyopacity()**. You then manipulate the row further (perhaps add it with other rows you've processed similarly), then do **pnm_unapplyopacity()**, then **pnm_gammarown()**, then **pnm_writegammarown()**.

pnm_normalizeTuple() and **pnm_unnormalizeTuple()** convert between a **tuple** data type and a **tuplen** data type. The former represents a sample value using the same unsigned integer that is in the PAM image, while the latter represents a sample value as a number scaled by the maxval to the range 0..1. I.e. **pnm_normalizeTuple()** divides every sample value by the maxval and **pnm_unnormalizeTuple()** multiples every sample by the maxval.

pnm_normalizeRow() and **pnm_unnormalizeRow()** do the same thing on an entire tuple row, but also have an extra feature: You can specify a transform function to be applied in addition. Typically, this is a gamma transform function. You can of course more easily apply your transform function separately from normalizing, but doing it all at once is usually way faster. Why? Because you can use a lookup table that is indexed by an integer on one side and produces a floating point number on the other. To do it separately, you'd either have to do floating point arithmetic on the normalized value or do the transform on the integer values and lose a lot of precision.

If you don't have any transformation to apply, just specify **NULL** for the *transform* argument and the function will just normalize (i.e. divide or multiply by the maxval).

Here's an example of doing a transformation. The example composes two images together, something that has to be done with intensity-linear sample values.

```
pnm_transformMap * const transform1 = pnm_createungammatransform(&inpam1);
pnm_transformMap * const transform2 = pnm_createungammatransform(&inpam2);
pnm_transformMap * const transformOut = pnm_creategammatransform(&outpam);

pnm_readpamrow(&inpam1, inrow1);
pnm_readpamrow(&inpam2, inrow2);

pnm_normalizeRow(&inpam1, inrow1, transform1, normInrow1);
pnm_normalizeRow(&inpam2, inrow2, transform2, normInrow2);

for (col = 0; col < outpam.width; ++col)
   normOutrow[col] = (normInrow1[col] + normInrow2[col])/2;

pnm_unnormalizeRow(&outpam, normOutrow, transformOut, outrow);

pnm_writepamrow(&outpam, outrow);
```

To specify a transform, you must create a special **pnm_transformMap** object and pass it as the *transform* argument. Typically, your transform is a gamma transformation because you want to work in intensity-proportional sample values and the PAM image format uses gamma-adjusted ones. In that case, just use **pnm_creategammtransform()** and **pnm_createungammatransform()** to create this object and don't worry about what's inside it.

pnm_creategammatransform() and **pnm_createungammatransform()** create objects that you use with **pnm_normalizeRow()** and **pnm_unnormalizeRow()** as described above. The created object describes a transform that applies or reverses the ITU Rec 709 gamma adjustment that is used in PAM visual images and normalizes or unnormalizes the sample values.

pnm_freegammatransform() and **pnm_freeungammatransform()** destroy the objects.

Miscellaneous

Synopsis

void pnm_checkpam(struct pam **pamP*, const enum pm_check_type *check_type*, enum pm_check_code **retvalP*);

void pnm_nextimage(FILE **file*, int * const *eofP*);

Description

pnm_checkpam() checks for the common file integrity error where the file is the wrong size to contain the raster, according to the information in the header.

pnm_nextimage()positions a Netpbm image input file to the next image in it (so that a subsequent **pnm_readpaminit()** reads its header).

NAME

libnetpbm_ug − netpbm sample code

The Libnetpbm programming library is part of **Netpbm**(1).

Example

Here is an example of a C program that uses **libnetpbm** to read a Netpbm image input and produce a Netpbm image output.

```
/* Example program fragment to read a PAM or PNM image
   from stdin, add up the values of every sample in it
   (I don't know why), and write the image unchanged to
   stdout. */

#include <pam.h>

struct pam inpam, outpam;
unsigned int row;

pnm_init(&argc, argv);

pnm_readpaminit(stdin, &inpam, PAM_STRUCT_SIZE(tuple_type));

outpam = inpam; outpam.file = stdout;

pnm_writepaminit(&outpam);

tuplerow = pnm_allocpamrow(&inpam);

for (row = 0; row < inpam.height; row++) {
   unsigned int column;
   pnm_readpamrow(&inpam, tuplerow);
   for (column = 0; column < inpam.width; ++column) {
      unsigned int plane;
      for (plane = 0; plane < inpam.depth; ++plane) {
         grand_total += tuplerow[column][plane];
      }
   }
   pnm_writepamrow(&outpam, tuplerow); }

pnm_freepamrow(tuplerow);
```

Guide To Using Libnetpbm

libnetpbm classes

In this section, we cover only the PAM functions in **libnetpbm**. As described in **the** introduction to **libnetpbm** (1), there are four other classes of image processing functions (PBM, PGM, PPM, PNM). They are less important, since you can do everything more easily with the PAM functions, but if you're working on old programs or need the extra efficiency those older functions can sometimes provide, you can find them documented as here: **PBM**Function**Manual**(1), **PGM**Function**Manual**(1), **PPM**Function**Manual**(1),**and PNM**Function**Manual**(1).

In case you're wondering, what makes the PAM functions easier to use is:

- Each function handles all the formats. It does so without converting to a common format, so your program can treat the different formats differently if it wants. However, the interface

makes it easy for your program to ignore the differences between the formats if that's what you want.

- The PAM function parameter lists convey most information about the image with which you're working with a single **pam** structure, which you can build once and use over and over, whereas the older functions require you to pass up to 5 pieces of image information (height, width, etc.) as separate arguments to every function.

THE pam STRUCTURE

The PAM functions take most of their arguments in the form of a single **pam** structure. This is not an opaque object, but just a convenient way to organize the information upon which most the functions depend. So you are free to access or set the elements of the structure however you want. But you will find in most cases it is most convenient to call **pnm_readpaminit()** or **pnm_writepaminit()** to set the fields in the **pam** structure before calling any other pam functions, and then just to pass the structure unchanged in all future calls to pam functions.

The fields are:

size The storage size in bytes of this entire structure.

len The length, in bytes, of the information in this structure. The information starts in the first byte and is contiguous. This cannot be greater than **size**. **size** and **len** can be used to make programs compatible with newer and older versions of the Netpbm libraries.

file The file.

format The format code of the raw image. This is **PAM_FORMAT** unless the PAM image is really a view of a PBM, PGM, or PPM image. Then it's **PBM_FORMAT**, **RPBM_FORMAT**, etc.

plainformat
This is a boolean value and means: The format above is a plain (text) format as opposed to a raw (binary) format. This is entirely redundant with the **format** member and exists as a separate member only for computational speed.

height The height of the image in rows.

width The width of the image in number of columns (tuples per row).

depth The depth of the image (degree of or number of samples in each tuple).

maxval
The maxval of the image. See definitions in **pam**(1).

bytes_per_sample
The number of bytes used to represent each sample in the image file. See the format definition in **pam**(1).**This** is entirely redundant with **maxval**. It exists as a separate member for computational speed.

tuple_type
The tuple type of the image. See definitions in **pam**(1).**Netpbm**does**not**define**any**values**for**this except the following, which are used for a PAM image which is really a view of a PBM, PGM, or PPM image: **PAM_PBM_TUPLETYPE**, **PAM_PGM_TUPLETYPE**,

PAM_PPM_TUPLETYPE.

allocation_depth

The number of samples for which memory is allocated for any tuple associated with this PAM structure. This must be at least as great as 'depth'. Only the first 'depth' of the samples of a tuple are meaningful.

The purpose of this is to make it possible for a program to change the type of a tuple to one with more or fewer planes.

0 means the allocation depth is the same as the image depth.

PLAIN VERSUS RAW FORMAT

The PNM formats each come in two varieties: the older plain (text) format and the newer raw (binary) format. There are different format codes for the plain and raw formats, but which of the two formats the pnm and pam functions write is independent of the format code you pass to them.

The pam functions always write raw formats. If you specify the format code for a plain format, a pam function assumes instead the raw version of that format.

The pnm functions choose between plain and raw based on the *forceplain* parameter that every write-type pnm function has. If this boolean value is true, the function writes the plain version of the format specified by the format code. If it is false, the function writes the raw version of the format specified by the format code.

We are trying to stamp out the older plain formats, so it would be a wise choice not to write a program that sets *forceplain* true under any circumstance. A user who needs a plain format can use the **pnmtoplainpnm** program to convert the output of your program to plain format.

Reference

The **Libnetpbm**Netpbm**Image** Processing Manual (1) describes the the **libnetpbm** functions for processing image data.

The **Libnetpbm**Utility**Manual**(1) describes the functions that are not specifically related to the Netpbm image formats.

Table Of Contents

NAME

libpbm - libnetpbm functions to read and write PBM image files

SYNOPSIS

#include pbm.h

bit **pbm_allocarray(int *cols*, **int** *rows*);

bit *pbm_allocrow(int *cols*);

pbm_freearray(bit **bits*, **int *rows*);

**pbm_freerow(bit *bitrow*);

void pbm_readpbminit(FILE * *fp*, **int ***colsP*, **int ***rowsP*, **int ***formatP*);

void pbm_readpbmrow(FILE * *fp*, **bit ***bitrow*, **int** *cols*, **int** *format*);

void pbm_readpbmrow_packed(FILE * *fp*, **unsigned char * const** *packed_bits*, **const int** *cols*, **const int** *format*);

void bit pbm_readpbm(FILE *** *fp*, **int ***colsP*, **int ***rowsP*);

void pbm_writepbminit(FILE * *fp*, **int** *cols*, **int** *rows*, **int** *forceplain*);

void pbm_writepbmrow(FILE * *fp*, **bit ***bitrow*, **int** *cols*, **int** *forceplain*);

void pbm_writepbmrow_packed(FILE * *fp*, **unsigned char * const** *packed_bits*, **const int** *cols*, **const int** *forceplain*);

void pbm_writepbm(FILE * *fp*, **bit **bits**, **int** *cols*, **int** *rows*, **int** *forceplain*);

#define pbm_packed_bytes(*cols***) ...**

void pbm_nextimage(FILE ***file*, **int * const *eofP*);

void pbm_check(FILE * *file*, **const enum pm_check_type** *check_type*, **const int** *format*, **const int** *cols*, **const int** *rows*, **enum pm_check_code * const** *retval*);

DESCRIPTION - PBM-SPECIFIC ROUTINES

These library functions are part of **Netpbm**(1).

TYPES AND CONSTANTS

typedef ... bit;

#define PBM_WHITE ...

#define PBM_BLACK ...

Each **bit** should contain only the values of **PBM_WHITE** or **PBM_BLACK**.

#define PBM_FORMAT ...

#define RPBM_FORMAT ...

#define PBM_TYPE PBM_FORMAT

#define PBM_FORMAT_TYPE(*f***) ...**

These are for distinguishing different file formats and types.

INITIALIZATION

pbm_init() is identical to **pm_init()**.

MEMORY MANAGEMENT

pbm_allocarray() allocates an array of bits. **pbm_allocrow()** allocates a row of the given number of bits. **pbm_freearray()** frees the array allocated with **pbm_allocarray()** containing the given number of rows. **pbm_freerow()** frees a row of bits.

READING PBM IMAGE FILES

pbm_readpbminit() reads the header from a PBM image in a PBM file, filling in the rows, cols and format variables. **pbm_readpbmrow()** reads a row of bits into the *bitrow* array. Format and cols were filled in by **pbm_readpbminit()**.

pbm_readpbmrow_packed() is like **pbm_readrow()** except instead of returning a **bits** array, it returns an array *packed_bits* of bytes with the pixels of the image row packed into them. The pixels are in order from left to right across the row and from the beginning of the array to the end. Within a byte, the bits are in order from the most significant bit to the least significant bit. If the number of pixels in the row is not a multiple of 8, the last byte returned is padded on the least signficant bit side with undefined bits. White is represented by a **PBM_WHITE** bit; black by **PBM_BLACK**.

pbm_readpbm() reads an entire bitmap file into memory, returning the allocated array and filling in the rows and cols variables. This function combines **pbm_readpbminit()**, **pbm_allocarray()** and **pbm_readpbmrow()**.

pbm_readpbminit() and **pbm_readpbm** abort the program with a message to Standard Error if the PBM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

ppm_readppminit() and **ppm_readppm** abort the program with a message to Standard Error if the PPM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

WRITING PBM IMAGE FILES

pbm_writepbminit() writes the header for a PBM image in a PBM file. *forceplain* is a boolean value specifying that a plain format (text) file to be written, as opposed to a raw format (binary) one. **pbm_writepbmrow()** writes a row to a PBM file. **pbm_writepbmrow_packed()** is the same as **pbm_writepbmrow()** except that you supply the row to write as an array of bytes packed with bits instead of as a **bits** array. The format of *packed_bits* is the same as that returned by **pbm_readpbmrow()**.

pbm_writepbm() writes the header and all data for a PBM image to a PBM file. This function combines **pbm_writepbminit()** and **pbm_writepbmrow()**.

MISCELLANEOUS

pbm_nextimage() positions a PBM input file to the next image in it (so that a subsequent **pbm_readpbminit()** reads its header).

Immediately before a call to **pbm_nextimage()**, the file must be positioned either at its beginning (i.e. nothing has been read from the file yet) or just after an image (i.e. as left by a **pbm_readpbmrow()** of the last row in the image).

In effect, then, all **pbm_nextimage()** does is test whether there is a next image or the file is positioned at end-of-file.

If **pbm_nextimage()** successfully positions to the next image, it returns ***eofP** false (0). If there is no next image in the file, it returns ***eofP** true . If it can't position or determine the file status due to a file error, it issues an error message and exits the program with an error exit code.

pbm_check() checks for the common file integrity error where the file is the wrong size to contain all the image data. **pbm_check()** assumes the file is positioned after an image header (as if **pbm_readpbminit()** was the last operation on the file). It checks the file size to see if the number of bytes left in the file are the number required to contain the image raster. If the file is too short, **pbm_check()** causes the program to exit with an error message and error completion code. Otherwise, it returns one of the following values (enumerations of the **enum pm_check_code** type) as ***retval**:

PM_CHECK_OK

The file's size is exactly what is required to hold the image raster.

PM_CHECK_UNKNOWN_TYPE
format is not a format whose size **pbm_check()** can anticipate. The only format with which **pbm_check()** can deal is raw PBM format.

PM_CHECK_TOO_LONG
The file is longer than it needs to be to contain the image raster. The extra data might be another image.

PM_CHECK_UNCHECKABLE
The file is not a kind that has a predictable size, so there is no simple way for **pbm_check()** to know if it is the right size. Only a regular file has predictable size. A pipe is a common example of a file that does not.

check_type must have the value **PM_CHECK_BASIC** (an enumerated value of the **pm_check_type** enumerated type). Otherwise, the effect of **pbm_check()** is unpredictable. This argument exists for future backward compatible expansion of the function of **pbm_check()**.

SEE ALSO

libpgm(1), **libppm**(1), **libpnm**(1), **pbm**(1)

AUTHOR

Table Of Contents

NAME

libpgm - libnetpbm functions to read and write PGM image files

SYNOPSIS

#include pgm.h

void pgm_init(int **argcP*, char **argv*[]);

gray ** pgm_allocarray(int *cols*, **int** *rows* **);**

gray * pgm_allocrow(int*cols* **);**

void pgm_freearray(gray ****grays*, **int*rows* **);**

void pgm_freerow(gray **grayrow*);

void pgm_readpgminit(FILE **fp*, int **colsP*, int **rowsP*, gray **maxvalP*, int **formatP*);

void pgm_readpgmrow(FILE **fp*, gray **grayrow*, int *cols*, **gray** *maxval*, **int** *format* **);**

gray ** pgm_readpgm(FILE **fp*, int **colsP*, int **rowsP*, gray **maxvalP*);

void pgm_writepgminit(FILE * fp , int *cols*, **int** *rows*, **gray** *maxval*, **int** *forceplain* **);**

void pgm_writepgmrow(FILE **fp*, gray **grayrow*, int cols, *gray* *maxval*, **int** *forceplain* **);**

void pgm_writepgm(FILE **fp*, gray ** *grays*, **int *cols*, **int** *rows*, **gray** *maxval*, **int** *forceplain* **);**

void pgm_writepgm(FILE **fp*, gray ****grays*, **int *cols*, **int** *rows*, **gray** *maxval*, **int** *forceplain* **);**

void pgm_nextimage(FILE **file*, int * const *eofP*);

void pgm_check(FILE * *file*, const enum pm_check_type *check_type*, **const int** *format*, **const int** *cols*, **const int** *rows*, **const int** *maxval*, **enum pm_check_code * const** *retval*);

typedef ... gray;

#define PGM_MAXMAXVAL ...

#define PGM_OVERALLMAXVAL ...

extern gray pgm_pbmmaxval;

#define PGM_FORMAT ...

#define RPGM_FORMAT ...

#define PGM_TYPE PGM_FORMAT

#define

PGM_FORMAT_TYPE(*format*) ...

DESCRIPTION

These library functions are part of **Netpbm**(1).

TYPES AND CONSTANTS

Each **gray** should contain only the values between **0** and **PGM_OVERALLMAXVAL**. **pgm_pbmmaxval** is the maxval used when a PGM program reads a PBM file. Normally it is 1; however, for some programs, a larger value gives better results.

PGM_OVERALLMAXVAL is the maximum value of a maxval in a PGM file. **PGM_MAXMAXVAL** is the maximum value of a maxval in a PGM file that is compatible with the PGM format as it existed before April 2000. It is also the maximum value of a maxval that results in the minimum possible raster size for a particular image. I.e an image with a maxval higher than **PGM_MAXMAXVAL** cannot be read or generated by old PGM processing programs and requires more file space.

PGM_FORMAT is the format code for a Plain PGM format image file. **RPGM_FORMAT** is the format code for a Raw PGM format image file. **PGM_TYPE** is the format type code for the PGM formats. **PGM_FORMAT_TYPE** is a macro that generates code to compute the format type code of a

PBM or PGM format from the format code which is its argument.

INITIALIZATION

pgm_init is identical to **pm_init()**.

MEMORY MANAGEMENT

pgm_allocarray() allocates an array of grays.

pgm_allocrow() allocates a row of the given number of grays.

pgm_freearray() frees the array allocated with **pgm_allocarray()** containing the given number of rows.

pgm_freerow() frees a row of grays allocated with **pgm_allocrow()**.

READING FILES

If a function in this section is called on a PBM format file, it translates the PBM file into a PGM file on the fly and functions as if it were called on the equivalent PGM file. The *format* value returned by **pgm_readpgminit()** is, however, not translated. It represents the actual format of the PBM file.

pgm_readpgminit() reads the header of a PGM file, returning all the information from the header and leaving the file positioned just after the header.

pgm_readpgmrow() reads a row of grays into the *grayrow* array. *format*, *cols*, and *maxval* are the values returned by **pgm_readpgminit()**.

pgm_readpgm() reads an entire PGM image into memory, returning the allocated array as its return value and returning the information from the header as *rows*, *cols*, and *maxval*. This function combines **pgm_readpgminit()**, **pgm_allocarray()**, and **pgm_readpgmrow()**.

pgm_readpgminit() and **pgm_readpgm** abort the program with a message to Standard Error if the PGM image header is not syntactically valid, including if it contains a number too large to be processed using the system's normal data structures (to wit, a number that won't fit in a C 'int').

WRITING FILES

pgm_writepgminit() writes the header for a PGM file and leaves it positioned just after the header.

forceplain is a logical value that tells **pgm_writepgminit()** to write a header for a plain PGM format file, as opposed to a raw PGM format file.

pgm_writepgmrow() writes the row *grayrow* to a PGM file. For meaningful results, *cols*, *maxval*, and *forceplain* must be the same as was used with **pgm_writepgminit()**.

pgm_writepgm() write the header and all data for a PGM image. This function combines **pgm_writepgminit()** and **pgm_writepgmrow()**.

MISCELLANEOUS

pgm_nextimage() positions a PGM input file to the next image in it (so that a subsequent **pgm_readpgminit()** reads its header).

pgm_nextimage() is analogous to **pbm_nextimage()**, but works on PGM and PBM files.

pgm_check() checks for the common file integrity error where the file is the wrong size to contain all the image data.

pgm_check() is analogous to **pbm_check()**, but works on PGM and PBM files.

SEE ALSO

libpbm(1), **libppm**(1), **libpnm**(1)

Table Of Contents

NAME

libpm - netpbm utility functions

DESCRIPTION

These library functions are part of **Netpbm**(1).

This page documents functions in the Netpbm subroutine library that are not directly related to image data.

For introductory and general information using **libnetpbm**, see **Libnetpbm**User's**Guide**(1).

The most commonly used **libnetpbm** functions are those that read and write and process Netpbm images. Those are documented in **Libnetpbm**Netpbm**Image**Processing Manual (1)

To use these services, #include **pam.h**.

Functions

Initialization

Entry Points void pm_init(int *_argcP_, char *_argv_[]);

Description

All Netpbm programs must call **pm_init()** just after startup, before they process their arguments. **pm_init()**, among other things, processes Netpbm universal arguments and removes them from the argument list.

A program that isn't a Netpbm program, but just uses **libnetpbm** services, need not invoke **pm_init**.

File Or Image Stream Access

Entry Points

FILE *pm_openr(char * *name***)**

FILE *pm_openw(char * *name* **);**

FILE *pm_openr_seekable(const char * *name* **);**

FILE *pm_close(FILE * *fp* **);**

void pm_tell2(FILE * *fileP*, **pm_filepos *** *fileposP*, **unsigned int** *fileposSize* **);**

unsigned int pm_tell(FILE * *fileP* **);**

void pm_seek2(FILE * *fileP*, **const pm_filepos *** *fileposP*, **unsigned int** *fileposSize* **);**

void pm_seek(FILE * *fileP*, **unsigned long** *filepos* **);**

char *pm_read_unknown_size(FILE * *fp*, **long *** *nread* **);**

Description

An image stream is just a file stream (represented in the standard C library as type **FILE ***).

These routines work on files > 2 GiB if the underlying system does, using the standard large file interface. Before Netpbm 10.15 (April 2003), though, they would fail to open any file that large or process any offset in a file that could not be represented in 32 bits.

pm_openr() opens the given file for reading, with appropriate error checking. A filename of - is taken to mean Standard Input. **pm_openw()** opens the given file for writing, with appropriate error checking. **pm_close()** closes the file descriptor, with appropriate error checking.

pm_openr_seekable() appears to open the file just like **pm_openr()**, but the file thus opened is guaranteed to be seekable (you can use ftell() and fseek() on it). **pm_openr_seekable()** pulls this off by copying the entire file to a temporary file and giving you the handle of the temporary file, if it has to. If the file you name is a regular file, it's already seekable so **pm_openr_seekable()** just does the same thing

as **pm_openr()**.

But if it is, say, a pipe, it isn't seekable. So **pm_openr_seekable()** reads the pipe until EOF into a temporary file, then opens that temporary file and returns the handle of the temporary file. The temporary file is seekable.

The file **pm_openr_seekable()** creates is one that the operating system recognizes as temporary, so when you close the file, by any means, it gets deleted.

You need a seekable file if you intend to make multiple passes through the file. The only alternative is to read the entire image into memory and work from that copy. That may use too much memory. Note that the image takes less space in the file cache than in a buffer in memory. As much as 96 times less space! Each sample is an integer in the buffer, which is usually 96 bits. In the file, a sample may be as small as 1 bit and rarely more than 8 bits.

pm_tell2() returns a handle for the current position of the image stream (file), whether it be the header or a row of the raster. Use the handle as an argument to **pm_seek2()** to reposition the file there later. The file must be seekable (which you can ensure by opening it with **pm_openr_seekable()**) or this may fail.

The file position handle is of type **pm_filepos**, which is intended to be opaque, i.e. used only with these two functions. In practice, it is a file offset and is 32 bits or 64 bits depending upon the capability of the underlying system. For maximum backward and forward compatibility, the functions that take or return a **pm_filepos** have a *fileposSize* argument for the size of the data structure. In C, simply code **sizeof(pm_filepos)** for that.

pm_seek() and **pm_tell** are for backward compatibility only. Do not use them in new code. These functions are not capable of handle positions in files whose byte offset cannot be represented in 32 bits.

pm_tell2() and **pm_seek2()** replaced **pm_tell()** and **pm_seek()** in Netpbm 10.15 (April 2003).

pm_read_unknown_size() reads an entire file or input stream of unknown size to a buffer. It allocates more memory as needed. The calling routine has to free the allocated buffer with **free()**.

pm_read_unknown_size() returns a pointer to the allocated buffer. The **nread** argument returns the number of bytes read.

Endian I/O

Entry Points

void pm_readchar(FILE * *in*, **char *** *sP* **);**

void pm_writechar(FILE * *out*, **char** *s* **);**

int pm_readbigshort(FILE * *in*, **short *** *sP* **);**

int pm_writebigshort(FILE * *out*, **short** *s* **);**

int pm_readbiglong(FILE * *in*, **long *** *lP* **);**

int pm_writebiglong(FILE * *out*, **long** *l* **);**

int pm_readlittleshort(FILE * *in*, **short *** *sP* **);**

int pm_writelittleshort(FILE * *out*, **short** *s* **);**

int pm_readlittlelong(FILE * *in*, **long *** *lP* **);**

int pm_writelittlelong(FILE * *out*, **long** *l* **);**

void pm_readcharu(FILE * *in*, **char *** *sP* **);**

void pm_writecharu(FILE * *out*, **char** *s* **);**

int pm_readbigshortu(FILE * *in*, **short *** *sP* **);**

int pm_writebigshortu(FILE * *out*, **short** *s* **);**

int pm_readbiglongu(FILE * *in*, **long *** *lP* **);**

int pm_writebiglongu(FILE * *out*, **long** *l* **);**

int pm_readlittleshortu(FILE * *in***, short *** *sP* **);**

int pm_writelittleshortu(FILE * *out***, short** *s* **);**

int pm_readlittlelongu(FILE * *in***, long *** *lP* **);**

int pm_writelittlelongu(FILE * *out***, long** *l* **);**

Description

pm_readchar(), **pm_writechar()**, **pm_readbigshort()**, **pm_writebigshort()**, **pm_readbiglong()**, **pm_writebiglong()**, **pm_readlittleshort()**, **pm_writelittleshort()**, **pm_readlittlelong()**, and **pm_writelittlelong()** are routines to read and write 1-byte, 2-byte, and 4-byte pure binary integers in either big- or little-endian byte order. Note that a 'long int' C type might be wider than 4 bytes, but the 'long' routines still read and write 4 bytes.

pm_readbiglongu(), etc. (names ending in **u**) are the same except they work on unsigned versions of the type.

The routines with declared return values always return 0. Before Netpbm 10.27 (March 2005), they returned -1 on failure, including EOF. Now, they issue an error message to Standard Error and abort the program if the I/O fails or encounters EOF.

The 1-byte routines were new in Netpbm 10.27 (March 2005). The unsigned versions were new somewhere around Netpbm 10.21 (2004).

Maxval Arithmetic

Entry Points

int pm_maxvaltobits(int *maxval* **);**

int pm_bitstomaxval(int *bits* **);**

unsigned int pm_lcm(unsigned int *x***, unsigned int** *y***, unsigned int** *z***, unsigned int** *limit* **);**

Description

pm_maxvaltobits() and **pm_bitstomaxval()** convert between a maxval and the minimum number of bits required to hold it.

pm_lcm() computes the least common multiple of 3 integers. You also specify a limit and if the LCM would be higher than that limit, **pm_lcm()** just returns that limit.

Gamma Arithmetic

Entry Points

float pm_gamma(float *intensity* **);**

float pm_ungamma(float *brightness* **);**

Description

In graphics processing, there are two common ways of representing numerically the intensity of a pixel, or a component of a pixel.

The obvious way is with a number that is directly proportional to the light intensity (e.g. 10 means twice as many milliwatts per square centimeter as 5). There are two problems with this:

- To the human eye, a 1 milliwatt per square centimeter difference in a bright image is much less apparent than a 1 milliwatt per square centimeter difference in a dark image. So if you have a fixed number of bits in which to store the intensity value, you're wasting resolution at the bright end skimping on it at the dark end.

- Monitor inputs and camera outputs aren't directly proportional to the light intensity they project or detect.

For these reasons, light intensities are often represented in graphics processing by an exponential scale. The transfer function is called a gamma function and the resulting numbers are called gamma-corrected or gamma-adjusted. There are various gamma functions. The Netpbm formats specify that intensities are represented by gamma-adjusted numbers of a particular gamma transfer function.

These functions let you convert back and forth between these two scales, using the same gamma transfer function that is specified in the Netpbm format specifications.

pm_gamma709 converts from an intensity-proportional intensity value to a gamma-adjusted intensity value (roughly proportional to brightness, which is the human subjective perception of intensity), using the CIE Rec. 709 gamma transfer function.

pm_ungamma709 is the inverse of **pm_gamma709**.

Messages

Entry Points

void pm_message(char * *fmt*, ...);

Description

pm_message() is a **printf()** style routine to write an informational message to the Standard Error file stream. **pm_message()** suppresses the message, however, if the user specified the **-quiet** option on the command line. See the initialization functions, e.g. **pnm_init()**, for information on the **-quiet** option. Note that Netpbm programs are often used interactively, but also often used by programs. In the interactive case, it is nice to issue messages about what the program is doing, but in the program case, such messages are usually undesirable. By using **pm_message()** for all your messages, you make your program usable in both cases. Without any effort on your part, program users of your program can avoid the messages by specifying the **-quiet** option.

If you're just issuing an error message and plan to abort the program, **pm_error()** may be more convenient for you.

System Utilities

- **pm_system**(1)
- **pm_tmpfile**(1)

Keyword Matching

Entry Points

Description

This subroutine is obsolete. It used to be used for command line option processing. Today, you can do better option processing more easily with the shhopt facility. See any recent program in the Netpbm package for an example.

pm_keymatch() does a case-insensitive match of **str** against **keyword**. **str** can be a leading substring of **keyword**, but at least **minchars** must be present.

Table Of Contents

NAME

libpnm - libnetpbm functions to read and write PNM image files

SYNOPSIS

#include <pnm.h>

void pnm_init(int ****argcP*, char ****argv***[]);**

xel ** pnm_allocarray(int *cols***, int** *rows***);**

xel * pnm_allocrow(int *cols***);**

void pnm_freearray(xel *****xels*, int** *rows***);**

void pnm_freerow(xel ****xelrow*);**

void pnm_readpnminit(FILE ****fp*, int ****colsP***, int ****rowsP***, xelval ****maxvalP***, int ****formatP** **);**

void pnm_readpnmrow(FILE ****fp*, xel ****xelrow***, int** *cols***, xelval** *maxval***, int** *format* **);**

xel ** pnm_readpnm(FILE ****fp*, int ****colsP***, int ****rowsP***, xelval ****maxvalP***, int* formatP** *);*

void pnm_writepnminit(FILE * fp , int *cols***, int** *rows***, xelval** *maxval***, int** *format***, int** *forceplain***);**

void pnm_writepnmrow(FILE ****fp*, xel ****xelrow***, int cols**, **xelval** *maxval***, int** *format***, int** *forceplain* **);**

void pnm_writepnm(FILE ****fp*, xel **** *xels***, int** *cols***, int** *rows***, xelval** *maxval***, int** *format***, int** *forceplain* **);**

void pnm_check(FILE * *file***, const enum pm_check_type** *check_type***, const int** *format***, const int** *cols***, const int** *rows***, const xelval** *maxval***, enum pm_check_code ****retvalP***);**

void pnm_promoteformatrow(xel ****xelrow*, int** *cols***, xelval** *maxval***, int** *format***, xelval** *newmaxval***, int** *newformat***);**

void pnm_promoteformat(xel *****xels*, int** *cols***, xelval** *maxval***, int** *format***, xelval** *newmaxval***, int** *newformat***);**

xel pnm_whitexel(xelval *maxval***, int** *format***);**

xel pnm_blackxel(xelval *maxval***, int** *format***);**

void pnm_invertxel(xel ****x*, xelval** *maxval***, int** *format***);**

xel pnm_backgroundxelrow(xel ****xelrow*, int** *cols***, xelval** *maxval***, int** *format***);**

xel pnm_backgroundxel(xel *****xels*, int** *cols***, int** *rows***, xelval** *maxval***, int** *format***);**

typedef ... xelval;

typedef ... xel;

extern xelval pnm_pbmmaxval;

#define PNM_ASSIGN1(*x***,***v***) ...**

#define PNM_GET1(*x***) ...**

#define PNM_EQUAL(*x***,***y***) ...**

#define PNM_FORMAT_TYPE(*format***) ...**

DESCRIPTION

These library functions are part of **Netpbm**(1).

PNM TYPES AND CONSTANTS

Each **xel** contains three **xelval**s, each of which should contain only the values between **0** and **PNM_MAXMAXVAL**, inclusive. **pnm_pbmmaxval** is the maxval used when a PNM program reads a PBM file. Normally it is 1; however, for some programs, a larger value gives better results.

PNM XEL MANIPULATIONS

The **PNM_GET1** macro extracts a single value from an xel, when you know it's from a PBM or PGM file. When it's from a PPM file, use **PPM_GETR()**, **PPM_GETG()**, and **PPM_GETB()**.

The **PNM_ASSIGN1** macro assigns a single value to an xel, when you know it's from a PBM or PGM file. When it's from a PPM file, use **PPM_ASSIGN**.

The **PNM_EQUAL** macro checks two xels for equality. The **PNM_FORMAT_TYPE** macro computes a format type code from a format code. The format types are PBM, PGM, PPM, and PAM. But note that PBM, PGM, and PPM each are two different formats: a plain one and a raw one. So there are four format types, but seven formats. **PNM_FORMAT_TYPE** does not work on the PAM format code.

INITIALIZATION

pnm_init is identical to **pm_init()**.

MEMORY MANAGEMENT

pnm_allocarray() allocates space for an array of xels. **pnm_freearray()** frees an array space allocated by **pnm_allocarray()** or **pnm_readpnm()**.

pnm_allocrow() allocates space for a row of a PNM image. **pnm_freerow()** frees it.

READING PNM FILES

pnm_readpnminit() is similar to **pnm_readpaminit()**, but reads only PNM images and has a different parameter list.

pnm_readpnmrow() is similar to **pnm_readpamrow()** but only works on PNM images and has a different parameter list and returns the row as an array of xels instead of tuples.

pnm_readpnm() is similar to **pnm_readpam()** except that it reads only PNM images and uses a different parameter list and returns an array of rows such that **pnm_readpnmrow()** would return rather than such that **pnm_readpamrow()** would return.

WRITING FILES

pnm_writepnminit() is similar to **pnm_writepaminit()** except that it can write only a PNM header and has a different parameter list.

forceplain is a binary value. True (nonzero) means to write the image in the plain (ASCII) version of the selected format. False (zero) means to write it in the raw (binary) version of the selected format. See **PNM**format**specification**(1).

pnm_writepnmrow() is similar to **pnm_writepamrow()** except that it works only on PNM images and has a different parameter list and takes an array of xels instead of an array of tuples. See the description of *forceplain* above.

pnm_writepnm() is similar to **pnm_writepam()** except that it works only on PNM image, has a different parameter list, and takes an array of rows of xels instead of an array of rows of tuples. See the description of *forceplain* above.

MISCELLANEOUS

pnm_check() is similar to **pnm_checkpam()** except it works only on PNM images.

pnm_check() is identical to **ppm_check()**.

PNM FORMAT PROMOTION

pnm_promoteformatrow() promotes a row of xels from one maxval and format to a new set. Use this when you are combining multiple anymaps of different types - just take the maximum of the maxvals and the maximum of the formats, and promote them all to that.

pnm_promoteformat() promotes an entire anymap.

PNM XEL MANIPULATION

pnm_whitexel() and **pnm_blackxel()** return a white or black xel, respectively, for the given *maxval* and *format*.

pnm_invertxel() inverts an xel.

pnm_backgroundxelrow() figures out an appropriate background xel based on the row of xels *xelrow*, which is *cols* xels wide, has maxval *maxval*, and represents an image with format *format*.

This estimate works best when the row is the top or bottom row of the image.

pnm_backgroundxel() does the same thing as **pnm_backgroundxelrow()**, except based on an entire image instead of just one row. This tends to do a slightly better job than **pnmbackgroundxelrow()**.

SEE ALSO

Libnetpbm(1), **Libnetpbm**User's**Guide**(1), **Libnetpbm**Directory**(1)**, **pbm**(1), **pgm**(1), **ppm**(1), **pam**(1), **libpbm**(1), **libpgm**(1), **libppm**(1)

AUTHOR

Table Of Contents

NAME

libppm - functions for PPM programs

SYNOPSIS

#include ppm.h

void ppm_init(

int ****argcP*,**

char ****argv*[]**

);

pixel ** ppm_allocarray(

int *cols***,**

int *rows* **);**

pixel * ppm_allocrow(

int *cols* **);**

void ppm_freearray(

pixel *****pixels*,**

int *rows* **);**

void ppm_freerow(

pixel ****pixelrow*);**

void ppm_readppminit(

FILE ****fp*,**

int ****colsP*,**

int ****rowsP*,**

pixval ****maxvalP*,**

int ****formatP* **);

void ppm_readppmrow(

FILE ****fp*,**

pixel ****pixelrow*,**

int *cols***,**

pixval *maxval***,**

int *format* **);**

pixel ** ppm_readppm(

FILE ****fp*,**

int ****colsP*,**

int ****rowsP*,**

pixvalP ****maxvalP* **);
void ppm_writeppminit(

FILE * fp ,

int *cols***,**

int *rows***,**

pixval *maxval***,**

int *forceplain* **);**
void ppm_writeppmrow(

FILE ****fp*,**

pixel ****pixelrow*,**

int cols,

pixval *maxval***,**

int *forceplain* **);**
void ppm_writeppm(

FILE ****fp*,**

pixel ** *pixels***,**

int *cols***,**

int *rows***,**

pixval *maxval***,**

int *forceplain* **);**
void ppm_writeppm(

FILE ****fp*,**

pixel *****pixels*,**

int *cols***,**

int *rows***,**

pixval *maxval***,**

int *forceplain* **);**
void ppm_nextimage(

FILE ****file*,**

int * const *eofP***);**

void ppm_check(

FILE * *file***,**

const enum pm_check_type *check_type***,**

const int *format***,**

const int *cols***,**

const int *rows***,**

const int *maxval***, enum pm_check_code * const** *retval***);**

typedef ... pixel;

typedef ... pixval;

#define PPM_MAXMAXVAL ...

#define PPM_OVERALLMAXVAL ...

#define PPM_FORMAT ...

#define RPPM_FORMAT ...

#define PPM_TYPE PPM_FORMAT

#define

PPM_FORMAT_TYPE(*format***)**

...

extern pixval ppm_pbmmaxval;

pixval PPM_GETR(pixel *p***)**

pixval PPM_GETG(pixel *p***)**

pixval PPM_GETB(pixel *p***)**

void PPM_ASSIGN(pixel *p***,**

pixval *red***, pixval** *grn***, pixval** *blu***)**

int PPM_EQUAL(pixel *p***, pixel** *q***)**

int PPM_ISGRAY(pixel *p***)**

void PPM_DEPTH(pixel *newp***, pixel** *p***,**

pixval *oldmaxval***, pixval** *newmaxval***)**

float PPM_LUMIN(pixel *p***)**

float PPM_CHROM_R(pixel *p***)**

float PPM_CHROM_B(pixel *p***)**

pixel ppm_parsecolor(char * *colorname***, pixval** *maxval***)**

char * ppm_colorname(pixel * *colorP***, pixval** *maxval***, int** *hexok***)**

void ppm_readcolornamefile(const char * *fileName*, **int** *mustOpen*, **colorhash_table *** *chtP*, **const char ***** *colornamesP* **)**

DESCRIPTION

These library functions are part of **Netpbm**(1).

TYPES AND CONSTANTS

Each **pixel** contains three **pixval**s, each of which should contain only the values between **0** and **PPM_MAXMAXVAL**. **ppm_pbmmaxval** is the maxval used when a PPM program reads a PBM file. Normally it is 1; however, for some programs, a larger value gives better results.

MANIPULATING PIXELS

The macros **PPM_GETR**, **PPM_GETG**, and **PPM_GETB** retrieve the red, green, or blue sample, respectively, from the given pixel.

The **PPM_ASSIGN** macro assigns the given values to the red, green, and blue samples of the given pixel.

The **PPM_EQUAL** macro tests two pixels for equality.

The **PPM_ISGRAY** macro tests a pixel for being gray. It returns true if and only if the color of pixel *p* is black, white, or gray.

The **PPM_DEPTH** macro scales the colors of pixel *p* according the old and new maxvals and assigns the new values to *newp*. It is intended to make writing ppmtowhatever easier.

The **PPM_LUMIN**, **PPM_CHROM_R**, and **PPM_CHROM_B** macros determine the luminance, red chrominance, and blue chrominance, respectively, of the pixel *p*. The scale of all these values is the same as the scale of the input samples (i.e. 0 to maxval for luminance, -maxval/2 to maxval/2 for chrominance).

Note that the macros do it by floating point multiplication. If you are computing these values over an entire image, it may be significantly faster to do it with multiplication tables instead. Compute all the possible products once up front, then for each pixel, just look up the products in the tables.

INITIALIZATION

ppm_init() is identical to **pm_init**.

MEMORY MANAGEMENT

ppm_allocarray() allocates an array of pixels.

ppm_allocrow() allocates a row of the given number of pixels.

ppm_freearray() frees the array allocated with **ppm_allocarray()** containing the given number of rows.

ppm_freerow() frees a row of pixelss allocated with **ppm_allocrow()**.

READING FILES

If a function in this section is called on a PBM or PGM format file, it translates the PBM or PGM file into a PPM file on the fly and functions as if it were called on the equivalent PPM file. The *format* value returned by **ppm_readppminit()** is, however, not translated. It represents the actual format of the PBM or PGM file.

ppm_readppminit() reads the header of a PPM file, returning all the information from the header and leaving the file positioned just after the header.

ppm_readppmrow() reads a row of pixels into the *pixelrow* array. *format*, *cols*, and *maxval* are the values returned by **ppm_readppminit()**.

ppm_readppm() reads an entire PPM image into memory, returning the allocated array as its return value and returning the information from the header as *rows*, *cols*, and *maxval*. This function combines **ppm_readppminit()**, **ppm_allocarray()**, and **ppm_readppmrow()**.

WRITING FILES

ppm_writeppminit() writes the header for a PPM file and leaves it positioned just after the header.

forceplain is a logical value that tells **ppm_writeppminit()** to write a header for a plain PPM format file, as opposed to a raw PPM format file.

ppm_writeppmrow() writes the row *pixelrow* to a PPM file. For meaningful results, *cols*, *maxval*, and *forceplain* must be the same as was used with **ppm_writeppminit()**.

ppm_writeppm() write the header and all data for a PPM image. This function combines **ppm_writeppminit()** and **ppm_writeppmrow()**.

MISCELLANEOUS

ppm_nextimage() positions a PPM input file to the next image in it (so that a subsequent **ppm_readppminit()** reads its header).

ppm_nextimage() is analogous to **pbm_nextimage()**, but works on PPM, PGM, and PBM files.

ppm_check() checks for the common file integrity error where the file is the wrong size to contain all the image data.

ppm_check() is analogous to **pbm_check()**, but works on PPM, PGM, and PBM files.

COLOR NAMES

System Color Dictionary

Netpbm uses the system's X11 color dictionary (usually in **/usr/lib/X11/rgb.txt**). This is the same file the X Window System typically uses to associate colors with their names.

The color dictionary that Netpbm uses is in the file whose name is the value of the **RGBDEF** environment variable. If **RGBDEF** is not set, Netpbm defaults to the first existing file from this list:

- **/usr/lib/X11/rgb.txt**
- **/usr/openwinlib/rgb.txt**
- **/usr/X11R6/lib/X11/rgb.txt**

You can see the color names from a typical X11 color dictionary, which is probably very close to what is on your system, along with the colors, here . **This** website (1) shows a bunch of other versions you could use.

Netpbm is packaged with a color dictionary. A standard Netpbm installation installs this file as "misc/rgb.txt" in the Netpbm directory. This color dictionary has colors from everywhere the Netpbm maintainer could find them, and is a superset of XFree 86's color dictionary.

ppm_parsecolor

ppm_parsecolor() interprets a color specification and returns a pixel of the color that it indicates. The color specification is ASCII text, in one of these formats:

- a name, as defined in the system color dictionary .
- An X11-style hexadecimal specifier: `rgb:`*r*`/`*g*`/`*b*, *where r, g, and b are each 1- to 4-digit hexadecimal numbers. For each, the maxval is the maximum number that can be represented in the number of hexadecimal digits given. Example:* `rgb:01/ff/8000` *specifies 1/255 red intensity, maximum green intensity, and about half blue intensity.*
- An X11-style decimal specifier: `rgbi:`*r*`/`*g*`/`*b*, *where r, g, and b are floating point numbers from 0 to 1.*

- an old-X11-style hexadecimal triple: `#rgb`, `#rrggbb`, `#rrrgggbbb`, or `#rrrrgggg-bbbb`.

- A triplet of decimal floating point numbers from 0.0 to 1.0, representing red, green, and blue intensities respectively, separated by commas. E.g. `1.0,0.5,.25`. This is for backwards compatibility; it was in use before MIT came up with the similar and preferred rgbi style).

If the color specification does not conform to any of these formats, including the case that it is a name, but is not in the system color dictionary, **ppm_parsecolor() throws**an**error**(1).

ppm_colorname

ppm_colorname() returns a string that describes the color of the given pixel. If a system color dictionary is available and the color appears in it, **ppm_colorname()** returns the name of the color from the file. If the color does not appear in a system color dictionary and *hexok* is true, **ppm_colorname()** returns a hexadecimal color specification triple (#rrggbb). If a system color dictionary is available but the color does not appear in it and *hexok* is false, **ppm_colorname()** returns the name of the closest matching color in the color file. Finally, if there is no system color dictionary available and *hexok* is false, **ppm_colorname()** fails and exits the program with an error message.

The string returned is in static libppm library storage which is overwritten by every call to **ppm_colorname()**.

ppm_readcolornamefile

ppm_readcolornamefile() reads the entire contents of the color dictionary in the file named *fileName* into data structures you can use to access it easily.

The function returns all the color names as an array of null-terminated strings. It mallocs the space for this array and returns its address at *colornamesP*. **(*colornamesP)[***i***]** is the address of the first character in the null-terminated string that is the name of the *i*th color in the dictionary.

The function also returns a **colorhash_table** (see COLOR INDEXING) that matches all these color names up to the colors they represent. It mallocs the space for the **colorhash_table** and returns its address at *chtP*. The number that the **colorhash_table** associates with each color is the index into the color name array described above of the name of that color.

You may specify a null pointer for *fileName* to indicate the default color dictionary.

mustOpen is a boolean. If it is nonzero, the function fails and aborts the program if it is unable to open the specified color dictionary file. If it is zero, though, it simply treats an unopenable color dictionary as an empty one. The colorhash and color name array it returns contain no colors or names.

ppm_readcolornamefile() was new in Netpbm 10.15 (April 2003).

COLOR INDEXING

Sometimes in processing images, you want to associate a value with a particular color. Most often, that's because you're generating a color mapped graphics format. In a color mapped graphics format, the raster contains small numbers, and the file contains a color map that tells what color each of those small numbers refers to. If your image has only 256 colors, but each color takes 24 bits to describe, this can make your output file much smaller than a straightforward RGB raster would.

So, continuing the above example, say you have a **pixel** value for chartreuse and in your output file and you are going to represent chartreuse by the number 12. You need a data structure that allows your program quickly to find out that the number for a chartreuse **pixel** is 12. Netpbm's color indexing data types and functions give you that.

colorhash_table is a C data type that associates an integer with each of an arbitrary number of colors. It is a hash table, so it uses far less space than an array indexed by the color's RGB values would.

The problem with a **colorhash_table** is that you can only look things up in it. You can't find out what

colors are in it. So Netpbm has another data type for representing the same information, the poorly but historically named **colorhist_vector**. A **colorhist_vector** is just an array. Each entry represents a color and contains the color's value (as a **pixel**) and the integer value associated with it. The entries are filled in starting with subscript 0 and going consecutively up for the number of colors in the histogram.

(The reason the name is poor is because a color histogram is only one of many things that could be represented by it).

colorhash_table ppm_alloccolorhash()

This creates a **colorhash_table** using dynamically allocated storage. There are no colors in it. If there is not enough storage, it exits the program with an error message.

void ppm_freecolorhash()

This destroys a **ppm_freecolorhash** and frees all the storage associated with it.

int ppm_addtocolorhash(colorhash_table cht, const pixel * const colorP, const int value)

This adds the specified color to the specified **colorhash_table**
and associates the specified value with it.

You must ensure that the color you are adding isn't already present in the **colorhash_table**.

There is no way to update an entry or delete an entry from a **colorhash_table**.

int ppm_lookupcolor(const colorhash_table cht, const pixel * const colorP)

This looks up the specified color in the specified **colorhash_table**. It returns the integer value associated with that color.

If the specified color is not in the hash table, the function returns -1. (So if you assign the value -1 to a color, the return value is ambiguous).

colorhist_vector ppm_colorhashtocolorhist(const colorhash_table cht,

const int ncolors)

This converts a **colorhash_table** to a **colorhist_vector**. The return value is a new **colorhist_vector** which you must eventually free with **ppm_freecolorhist()**.

ncolors is the number of colors in **cht**. If it has more colors than that, **ppm_colorhashtocolorhist** does not create a **colorhist_vector** and returns NULL.

colorhash_table ppm_colorhisttocolorhash(const colorhist_vector chv, const int ncolors)

This poorly named function does *not* convert from a **colorhist_vector** to a **colorhash_table**.

It does create a **colorhash_table** based on a **colorhist_vector** input, but the integer value for a given color in the output is not the same as the integer value for that same color in the input. **ppm_colorhisttocolorhash()** ignores the integer values in the input. In the output, the integer value for a color is the index in the input **colorhist_vector** for that color.

You can easily create a color map for an image by running **ppm_computecolorhist()** over the image, then **ppm_colorhisttocolorhash()** over the result. Now you can use **ppm_lookupcolor()** to find a unique color index for any pixel in the input.

If the same color appears twice in the input, **ppm_colorhisttocolorhash()** exit the program with an error message.

ncolors is the number of colors in **chv**.

The return value is a new **colorhash_table** which you must eventually free with **ppm_freecolorhash()**.

COLOR HISTOGRAMS

The Netpbm libraries give you functions to examine a Netpbm image and determine what colors are in it and how many pixels of each color are in it. This information is known as a color histogram. Netpbm uses its **colorhash_table** data type to represent a color histogram.

colorhash_table ppm_computecolorhash(pixel ** const pixels, const int cols, const int rows, const int maxcolors, int* const colorsP)

This poorly but historically named function generates a **colorhash_table** whose value for each color is

the number of pixels in a specified image that have that color. (I.e. a color histogram). As a bonus, it returns the number of colors in the image.

(It's poorly named because not all **colorhash_table**s are color histograms, but that's all it generates).

pixels, **cols**, and **rows** describe the input image.

maxcolors is the maximum number of colors you want processed. If there are more colors that that in the input image, **ppm_computecolorhash()** returns NULL as its return value and stops processing as soon as it discovers this. This makes it run faster and use less memory. One use for **maxcolors** is when you just want to find out whether or not the image has more than N colors and don't want to wait to generate a huge color table if so. If you don't want any limit on the number of colors, specify **maxcolors=0**.

ppm_computecolorhash() returns the actual number of colors in the image as ***colorsP**, but only if it is less than or equal to **maxcolors**.

colorhash_table ppm_computecolorhash2(FILE * const ifp, const int cols, const int rows, const pixval maxval, const int format,

const int maxcolors, int* const colorsP)

This is the same as **ppm_computecolorhash()** except that instead of feeding it an array of pixels in storage, you give it an open file stream and it reads the image from the file. The file must be positioned after the header, at the raster. Upon return, the file is still open, but its position is undefined.

maxval and **format** are the values for the image (i.e. information from the file's header).

colorhist_vector ppm_computecolorhist(pixel ** pixels, int cols, int rows, int maxcolors, int * colorsP)

This is like **ppm_computecolorhash()** except that it creates a **colorhist_vector** instead of a **colorhash_table**.

If you supply a nonzero **maxcolors** argument, that is the maximum number of colors you expect to find in the input image. If there are more colors than you say in the image, **ppm_computecolorhist()** returns a null pointer as its return value and nothing meaningful as ***colorsP**.

If not, the function returns the new **colorhist_vector** as its return value and the actual number of colors in the image as ***colorsP**. The returned array has space allocated for the specified number of colors regardless of how many actually exist. The extra space is at the high end of the array and is available for your use in expanding the **colorhist_vector**.

If you specify **maxcolors=0**, there is no limit on the number of colors returned and the return array has space for 5 extra colors at the high end for your use in expanding the **colorhist_vector**.

colorhist_vector ppm_computecolorhist2(FILE * ifp, int cols, int rows, int maxcolors, pixval maxval, int format, int * colorsP)

This is the same as **ppm_computecolorhist()** except that instead of feeding it an array of pixels in storage, you give it an open file stream and it reads the image from the file. The file must be positioned after the header, at the raster. Upon return, the file is still open, but its position is undefined.

SEE ALSO

pbm(1), **pgm**(1), **libpbm**(1)

AUTHOR

Name

pm_system - run a Netpbm program with program input and output

Synopsis

```
#include <netpbm/pm_system.h>

pm_system(void                stdinFeeder(int, void *),
          void *        const feederParm,
          void                stdoutAccepter(int, void *),
          void *        const accepterParm,
          const char *  const shellCommand);
```

Example

This simple example converts a PNM image on Standard Input to a JFIF (JPEG) image on Standard Output. In this case, **pm_system()** is doing no more than **system()** would do.

```
pm_system(NULL, NULL, NULL, NULL, "pnmtojpeg");
```

This example does the same thing, but moves the data through memory buffers to illustrate use with memory buffers, and we throw in a stage to shrink the image too:

```
#include <netpbm/pm_system.h>

char            pnmData[100*1024];   /* Input file better be < 100K */
char            jfifData[100*1024];
struct bufferDesc pnmBuffer;
struct bufferDesc jfifBuffer;
unsigned int      jfifSize;

pnmBuffer.size = fread(pnmData, 1, sizeof(pnmData), stdin);
pnmBuffer.buffer = pnmData;
pnmBuffer.bytesTransferredP = NULL;

jfifBuffer.size = sizeof(jfifData);
jfifBuffer.buffer = jfifData;
jfifBuffer.bytesTransferredP = &jfifSize;

pm_system(&pm_feed_from_memory, &pnmBuffer,
          &pm_accept_to_memory, &jfifBuffer,
          "pamscale .5 | pnmtojpeg");

fwrite(jfifData, 1, jfifSize, stdout);
```

This example reads an image into libnetpbm PAM structures, then brightens it, then writes it out, to illustrate use of **pm_system** with PAM structures.

```
#include <netpbm/pam.h>
#include <netpbm/pm_system.h>

struct pam       inpam;
struct pam       outpam;
tuples **        inTuples;
tuples **        outTuples;
struct pamtuples inPamtuples;
struct pamtuples outPamtuples;

inTuples = pnm_readpam(stdin, &inpam, sizeof(inpam));

outpam = inpam;
```

```
inPamtuples.pamP = &inpam;
inPamtuples.tuplesP = &inTuples;
outPamtuples.pamP = &outpam;
outPamtuples.tuplesP = &outTuples;

pm_system(&pm_feed_from_pamtuples, &inPamtuples,
        &pm_accept_to_pamtuples, &outPamtuples,
        "ppmbrighten -v 100");

outpam.file = stdout;
pnm_writepam(&outpam, outTuples);
```

DESCRIPTION

This library function is part of **Netpbm**(1).

pm_system() is a lot like the standard C library **system()** subroutine. It runs a shell and has that shell execute a shell command that you specify. But **pm_system()** gives you more control over the Standard Input and Standard Output of that shell command than **system()**. **system()** passes to the shell command as Standard Input and Output whatever is the Standard Input and Output of the process that calls **system()**. But with **pm_system()**, you specify as arguments subroutines to execute to generate the shell command's Standard Input stream and to process the shell command's Standard Output stream.

Your Standard Input feeder subroutine can generate the stream in limitless ways. **pm_system()** gives it a file descriptor of a pipe to which to write the stream it generates. **pm_system()** hooks up the other end of that pipe to the shell command's Standard Input.

Likewise, your Standard Output accepter subroutine can do anything it wants with the stream it gets. **pm_system()** gives it a file descriptor of a pipe from which to read the stream. **pm_system()** hooks up the other end of that pipe to the shell command's Standard Output.

The argument *stdinFeeder* is a function pointer that identifies your Standard Input feeder subroutine. **pm_system()** runs it in a child process and waits for that process to terminate (and accepts its completion status) before returning. *feederParm* is the argument that **pm_system()** passes to the subroutine; it is opaque to **pm_system()**.

If you pass *stdinFeeder* = NULL, **pm_system()** simply passes your current Standard Input stream to the shell command (as **system()** would do), and does not create a child process.

The argument *stdoutAccepter* is a function pointer that identifies your Standard Output accepter subroutine. **pm_system()** calls it in the current process. *accepterParm* is an argument analogous to *feederParm*.

If you pass *stdoutAccepter* = NULL, **pm_system()** simply passes your current Standard Output stream to the shell command (as **system()** would do.

The argument *shellCommand* is a null-terminated string containing the shell command that the shell is to execute. It can be any command that means something to the shell and can take a pipe for Standard Input and Output. Example:

```
ppmbrighten -v 100 | pnmdepth 255 | pamscale .5
```

pm_system() creates a child process to run the shell and waits for that process to terminate (and accepts its completion status) before returning.

Applications

The point of **pm_system()** is to allow you write a C program that uses other programs internally, as a shell script would. This is particularly desirable with Netpbm, because Netpbm consists of a lot of programs that perform basic graphic manipulations and you'd like to be able to build a program that does a more sophisticated graphic manipulation by calling the more basic Netpbm programs. These building

block programs typically take input from Standard Input and write output to Standard Output.

The obvious alternative is to use a higher level language -- Bourne Shell or Perl, for example. But often you want your program to do manipulations of your graphical data that are easier and more efficient in C. Or you want to use the Netpbm subroutine library in your program. The Netpbm subroutine library is a C-linkage library; the subroutines in it are not usable from a Bourne Shell or Perl program.

A typical use of **pm_system()** is to place the contents of some graphical image file in memory, run a Netpbm program against it, and have what would ordinarily go into an output file in memory too, for further processing. To do that, you can use the memory buffer Standard Input feeder and Standard Output accepter described below.

If your program uses the Netpbm subroutine library to read, write, and manipulate images, you may have an image in an array of PAM tuples. If you want to manipulate that image with a Netpbm program (perhaps remap the colors using **pnmremap**), you can use the pamtuple Standard Input feeder and Standard Output acceptor described below.

Broken Pipe Behavior

When you set up a shell command to take input from a pipe, as you do with **pm_system()**, you need to understand how pipes work with respect to the programs at either end of the pipe agreeing to how much data is to be transferred. Here are some notes on that.

It is normal to read a pipe before the process on the other end has written the data you hope to read, and it is normal to write to a pipe before the process on the other end has tried to read your data. Writes to a pipe can be buffered until the reading end requests the data. A process reading or writing a pipe can block until the other end is ready. Or a read or write can complete with an indication that the other end is not ready at the moment and therefore no data, or less data than was requested, was transferred.

The pipe is normally controlled by the writing end. When you read from a pipe, you keep reading until the program on the other end of the pipe closes it, and then you get an end-of-file indication. You then normally close the reading end of the pipe, since it is no longer useful.

When you close the reading end of a pipe before getting the end-of-file indication and the writer subsequently tries to write to the pipe, that is an error condition for the writer. In a typical default Unix environment, that error causes the writer to receive a SIGPIP signal and that signal causes the writer process to terminate abnormally. But if, alternatively, the writer has ordered that SIGPIPE be blocked, ignored, or handled, the signal does not cause the death of the writer. Instead, the write operation simply completes with an error indication.

Standard Feeders And Acceptors

You can supply anything you like as a Standard Input feeder or Standard Output acceptor, but the Netpbm subroutine library comes with a few that perform commonly needed functions.

Memory Buffer

These routines are for when you just want to treat an area of memory as a file. If the shell command would ordinarily read a 513 byte regular file from its Standard Input, you want it to take 513 bytes from a certain address in your process' memory. Whatever bytes the shell command wants to write to its output file you want it to store at another address in your process' memory.

The Standard Input feeder for this is called **pm_feed_from_memory**. The Standard Output accepter is **pm_accept_to_memory**.

For both of these, the argument is the address of a **struct bufferDesc**, which is defined as follows:

```
struct bufferDesc {
  unsigned int    size;
  unsigned char * buffer;
  unsigned int *  bytesTransferredP;
};
```

size is the size of the memory buffer and *buffer* is its location in memory (address). The Standard Input feeder will attempt to feed the entire buffer to the shell command's Standard Input; the Standard Output accepter will not accept any more data from the shell command's Standard Output than will fit in the buffer. Both return the actual amount of data read or written, in bytes, at the location identified by *bytesTransferredP*. Unless **bytesTransferredP** is NULL.

Because a process typically terminates abnormally when it is not able to write everything to a pipe that it wanted to, *bytesTransferredP* is not usually useful in the Standard Input feeder case.

Pamtuple

These routines are for when you have images in memory in the data structures used by the PAM family of subroutines in the Netpbm library -- i.e. struct PAM and an array of struct tuple. With these routines, you can run a Netpbm program against such an image just as you would against the same image in a regular file.

The Standard Input feeder for this is called **pm_feed_from_pamtuples**. The Standard Output accepter is **pm_accept_to_pamtuples**.

For both of these, the argument is the address of a **struct pamtuples**, which is defined as follows:

```
struct pamtuples {
   struct pam * pamP;
   tuple ***    tuplesP;
};
```

For the Standard Input feeder, you supply a fully valid struct pam (except it doesn't matter what the *file* field is) and array of tuples.

For the Standard Output Accepter, you supply only space in memory for the struct pam and the address of the tuple array. The routine fills in the struct pam (except leaves the *file* field undefined) and allocates space for the tuple array with malloc(). You are responsible for freeing that memory.

HISTORY

pm_system() was introduced in Netpbm 10.13 (January 2003).

NAME

pm_tmpfile() - tempfile creation routine using TMPFILE

SYNOPSIS

```
#include <netpbm/pm.h>

FILE *
pm_tmpfile(void);
```

EXAMPLE

This simple example creates a temporary file, writes "hello world" to it, then reads back and prints those contents.

```
#include <netpbm/pm.h>

FILE * myfile;

myfile = pm_tmpfile();

fprintf(myfile, "hello world0);

fseek(myfile, 0, SEEK_SET);

fread(buffer, sizeof(buffer), 1, myfile);

fprintf(STDOUT, "temp file contains '%s'0, buffer);

fclose(myfile);
```

DESCRIPTION

This library function is part of **Netpbm**(1).

pm_tmpfile() creates and opens an unnamed temporary file. It is basically the same thing as the standard C library **tmpfile()** function, except that it uses the **TMPFILE** environment variable to decide where to create the temporary file. If **TMPFILE** is not set or is set to something unusable (e.g. too long), **pm_tmpfile()** falls back to the value of the standard C library symbol **P_tmpdir**, just like **tmpfile()**.

Unlike **tmpfile()**, **pm_tmpfile()** never returns NULL. If it fails, it issues a message to Standard Error and aborts the program, like most libnetpbm routines do.

HISTORY

pm_tmpfile() was introduced in Netpbm 10.20 (January 2004).

Created: 17 April 2003

NAME

extendedopacity − theory of netpbm interpolation and extrapolation

DESCRIPTION

This page is a copy of http://www.sgi.com/grafica/interp/ on April 17, 2003, with some slight formatting changes, included in the Netpbm documentation for convenience.

Image Processing By Interpolation and Extrapolation

Paul Haeberli and Douglas Voorhies

Introduction

Interpolation and extrapolation between two images offers a general, unifying approach to many common point and area image processing operations. Brightness, contrast, saturation, tint, and sharpness can all be controlled with one formula, separately or simultaneously. In several cases, there are also performance benefits.

Linear interpolation is often used to blend two images. Blend fractions (alpha) and (1 - alpha) are used in a weighted average of each component of each pixel:

```
out = (1 - alpha)*in0 + alpha*in1
```

Typically alpha is a number in the range 0.0 to 1.0. This is commonly used to linearly interpolate two images. What is less often considered is that alpha may range beyond the interval 0.0 to 1.0. Values above one subtract a portion of in0 while scaling in1. Values below 0.0 have the opposite effect.

Extrapolation is particularly useful if a degenerate version of the image is used as the image to get "away from." Extrapolating away from a black-and-white image increases saturation. Extrapolating away from a blurred image increases sharpness. The interpolation/extrapolation formula offers one-parameter control, making display of a series of images, each differing in brightness, contrast, sharpness, color, or saturation, particularly easy to compute, and inviting hardware acceleration.

In the following examples, a single alpha value is used per image. However other processing is possible, for example where alpha is a function of X and Y, or where a brush footprint controls alpha near the cursor.

Changing Brightness

To control image brightness, we use pure black as the degenerate (zero alpha) image. Interpolation darkens the image, and extrapolation brightens it. In both cases, brighter pixels are affected more.

brightness

Changing Contrast

Contrast can be controlled using a constant gray image with the average image luminance. Interpolation reduces contrast and extrapolation boosts it. Negative alpha generates inverted images with varying contrast. In all cases, the average image luminance is constant.

contrast

If middle gray or the average pixel color is used instead, contrast is again altered, but with middle gray or the average color left unaffected. Shades and colors far away from the chosen value are most affected.

Changing Saturation

To alter saturation, pixel components must move towards or away from the pixel's luminance value. By using a black-and-white image as the degenerate version, saturation can be decreased using interpolation, and increased using extrapolation. This avoids computationally more expensive conversions to and from HSV space. Repeated update in an interactive application is especially fast, since the

luminance of each pixel need not be recomputed. Negative alpha preserves luminance but inverts the hue of the input image.

saturation

Sharpening an Image

Any convolution, such as sharpening or blurring, can be adjusted by this approach. If a blurred image is used as the degenerate image, interpolation attenuates high frequencies to varying degrees, and extrapolation boosts them, sharpening the image by unsharp masking. Varying alpha acts as a kernel scale factor, so a series of convolutions differing only in scale can be done easily, independent of the size of the kernel. Since blurring, unlike sharpening, is often a separable operation, sharpening by extrapolation may be far more efficient for large kernels.

sharpening

Note that global contrast control, local contrast control, and sharpening form a continuum. Global contrast pushes pixel components towards or away from the average image luminance. Local contrast is similar, but uses local area luminance. Unsharp masking is the extreme case, using only the color of nearby pixels.

Combined Processing

An unusual property of this interpolation/extrapolation approach is that all of these image parameters may be altered simultaneously. Here sharpness, tint, and saturation are all altered.

combined

Conclusion

Image applications frequently need to produce multiple degrees of manipulation interactively. Image applications frequently need to interactively manipulate an image by continuously changing a single parameter. The best hardware mechanisms employ a single "inner loop" to achieve a wide variety of effects. Interpolation and extrapolation of images can be a unifying approach, providing a single function that supports many common image processing operations.

Since a degenerate image is sometimes easier to calculate, extrapolation may offer a more efficient method to achieve effects such as sharpening or saturation. Blending is a linear operation, and so it must be performed in linear, not gamma-warped space. Component range must also be monitored, since clamping, especially of the degenerate image, causes inaccuracy.

These image manipulation techniques can be used in paint programs to easily implement brushes that saturate, sharpen, lighten, darken, or modify contrast and color. The only major change needed is to support alpha values outside the range 0.0 to 1.0.

It is surprising and unfortunate how many graphics software packages needlessly limit interpolant values to the range 0.0 to 1.0. Application developers should allow users to extrapolate parameters when practical.

References

For a slightly extended version of this article, see: P. Haeberli and D. Voorhies. *Image Processing by Linear Interpolation and Extrapolation*. IRIS Universe Magazine No. 28, Silicon Graphics, Aug, 1994.

Table Of Contents

NAME

pam - Netpbm common 2-dimensional bitmap format

GENERAL

The PAM image format is a lowest common denominator 2 dimensional map format.

It is designed to be used for any of myriad kinds of graphics, but can theoretically be used for any kind of data that is arranged as a two dimensional rectangular array. Actually, from another perspective it can be seen as a format for data arranged as a three dimensional array.

This format does not define the meaning of the data at any particular point in the array. It could be red, green, and blue light intensities such that the array represents a visual image, or it could be the same red, green, and blue components plus a transparency component, or it could contain annual rainfalls for places on the surface of the Earth. Any process that uses the PAM format must further define the format to specify the meanings of the data.

A PAM image describes a two dimensional grid of tuples. The tuples are arranged in rows and columns. The width of the image is the number of columns. The height of the image is the number of rows. All rows are the same width and all columns are the same height. The tuples may have any degree, but all tuples have the same degree. The degree of the tuples is called the depth of the image. Each member of a tuple is called a sample. A sample is an unsigned integer which represents a locus along a scale which starts at zero and ends at a certain maximum value greater than zero called the maxval. The maxval is the same for every sample in the image. The two dimensional array of all the Nth samples of each tuple is called the Nth plane or Nth channel of the image.

Though the basic format does not assign any meaning to the tuple values, it does include an optional string that describes that meaning. The contents of this string, called the tuple type, are arbitrary from the point of view of the basic PAM format, but users of the format may assign meaning to it by convention so they can identify their particular implementations of the PAM format. Some tuple types are defined as official subformats of PAM. See Defined Tuple Types .

THE LAYOUT

A convenient way to read and write the PAM format accurately is via the **libnetpbm**(1)**C**subroutine**library.**

A PAM file consists of a sequence of one or more PAM images. There are no data, delimiters, or padding before, after, or between images.

Each PAM image consists of a header followed immediately by a raster.

Here is an example header:

P7 WIDTH 227 HEIGHT 149 DEPTH 3 MAXVAL 255 TUPLTYPE RGB ENDHDR

The header begins with the ASCII characters ’P7’ followed by newline. This is the magic number.

Note: **xv** thumbnail images also start with the "P7" magic number. (This and PAM were independent extensions to the Netpbm formats). The rest of the format makes it easy to distinguish PAM from that format, though).

The header continues with an arbitrary number of lines of ASCII text. Each line ends with and is delimited by a newline character.

Each header line consists of zero or more whitespace-delimited tokens or begins with ’#’. If it begins with ’#’ it is a comment and the rest of this specification does not apply to it.

A header line which has zero tokens is valid but has no meaning.

The type of header line is identified by its first token, which is 8 characters or less:

ENDHDR

This is the last line in the header. The header must contain exactly one of these header lines.

HEIGHT
The second token is a decimal number representing the height of the image (number of rows). The header must contain exactly one of these header lines.

WIDTH
The second token is a decimal number representing the width of the image (number of columns). The header must contain exactly one of these header lines.

DEPTH
The second token is a decimal number representing the depth of the image (number of planes or channels). The header must contain exactly one of these header lines.

MAXVAL
The second token is a decimal number representing the maxval of the image. The header must contain exactly one of these header lines.

TUPLTYPE
The header may contain any number of these header lines, including zero. The rest of the line is part of the tuple type. The rest of the line is not tokenized, but the tuple type does not include any white space immediately following **TUPLTYPE** or at the very end of the line. It does not include a newline. If there are multiple **TUPLTYPE** header lines, the tuple type is the concatenation of the values from each of them, separated by a single blank, in the order in which they appear in the header. If there are no **TUPLTYPE** header lines the tuple type is the null string.

The raster consists of each row of the image, in order from top to bottom, consecutive with no delimiter of any kind between, before, or after, rows.

Each row consists of every tuple in the row, in order from left to right, consecutive with no delimiter of any kind between, before, or after, tuples.

Each tuple consists of every sample in the tuple, in order, consecutive with no delimiter of any kind between, before, or after, samples.

Each sample consists of an unsigned integer in pure binary format, with the most significant byte first. The number of bytes is the minimum number of bytes required to represent the maxval of the image.

The Confusing Universe of Netpbm Formats

It is easy to get confused about the relationship between the PAM format and PBM, PGM, PPM, and PNM. Here is a little enlightenment:

"PNM" is not really a format. It is a shorthand for the PBM, PGM, and PPM formats collectively. It is also the name of a group of library functions that can each handle all three of those formats.

'PAM' is in fact a fourth format. But it is so general that you can represent the same information in a PAM image as you can in a PBM, PGM, or PPM image. And in fact a program that is designed to read PBM, PGM, or PPM and does so with a recent version of the Netpbm library, will read an equivalent PAM image just fine and the program will never know the difference.

To confuse things more, there is a collection of library routines called the 'pam' functions that read and write the PAM format, but also read and write the PBM, PGM, and PPM formats. They do this because the latter formats are much older and more popular, so even a new program must work with them. Having the library handle all the formats makes it convenient to write programs that use the newer PAM format as well.

DEFINED TUPLE TYPES

Some tuple types are defined in this specification to specify official subformats of PAM for especially popular applications of the format. Users of the format may also define their own tuple types, and thus their own subformats.

PAM Used For Visual Images

A common use of PAM images is to represent visual images such as are typically represented by images in the older and more concrete PBM, PGM, and PPM formats.

Black And White (PBM)

A black and white image, such as would be represented by a PBM image, has a tuple type of "BLACKANDWHITE". Such a PAM image has a depth of 1 and maxval 1 where the one sample in each tuple is 0 to represent a black pixel and 1 to represent a white one. The height, width, and raster bear the obvious relationship to those of the equivalent PBM image.

Note that in the PBM format, a zero value means white, but in PAM, zero means black.

Grayscale (PGM)

A grayscale image, such as would be represented by a PGM image, has a tuple type of "GRAYSCALE". Such a PAM image has a depth of 1. The maxval, height, width, and raster bear the obvious relationship to those of the equivalent PGM image.

Color (PPM)

A color image, such as would be represented by a PPM image, has a typle type of "RGB". Such a PAM image has a depth of 3. The maxval, height, width, and raster bear the obvious relationship to those of the PPM image. The first plane represents red, the second blue, and the third green.

Transparent

Each of the visual image formats mentioned above has a variation that contains transparency information. In that variation, the tuple type has '_ALPHA' added to it (e.g. 'RGB_ALPHA') and one more plane. The highest numbered plane is the opacity plane (sometimes called an alpha plane or transparency plane).

In this kind of image, the color represented by a pixel is actually a combination of an explcitly specified foreground color and a background color to be identified later.

The planes other than the opacity plane describe the foreground color. A sample in the opacity plane tells how opaque the pixel is, by telling what fraction of the pixel's light comes from the foreground color. The rest of the pixel's light comes from the (unspecified) background color.

For example, in a GRAYSCALE_ALPHA image, assume Plane 0 indicates a gray tone 60% of white and Plane 1 indicates opacity 25%. The foreground color is the 60% gray, and 25% of that contributes to the ultimate color of the pixel. The other 75% comes from some background color. So let's assume further that the background color of the pixel is full white. Then the color of the pixel is 90% of white: 25% of the foreground 60%, plus 75% of the background 100%.

The sample value is the opacity fraction just described, as a fraction of the maxval. Note that it is *not* gamma-adjusted like the foreground color samples.

SEE ALSO

Netpbm(1), **pbm**(1), **pgm**(1), **ppm**(1), **pnm**(1), **libnetpbm**(1)

Table Of Contents

NAME

pbm - Netpbm bi-level image format

DESCRIPTION

This program is part of **Netpbm**(1).

The PBM format is a lowest common denominator monochrome file format. It serves as the common language of a large family of bitmap image conversion filters. Because the format pays no heed to efficiency, it is simple and general enough that one can easily develop programs to convert to and from just about any other graphics format, or to manipulate the image.

The name "PBM" is an acronym derived from "Portable Bit Map."

This is not a format that one would normally use to store a file or to transmit it to someone -- it's too expensive and not expressive enough for that. It's just an intermediary format. In it's purest use, it lives only in a pipe between two other programs.

The format definition is as follows.

A PBM file consists of a sequence of one or more PBM images. There are no data, delimiters, or padding before, after, or between images.

Each PBM image consists of the following:

- A 'magic number' for identifying the file type. A pbm image's magic number is the two characters 'P4'.
- Whitespace (blanks, TABs, CRs, LFs).
- The width in pixels of the image, formatted as ASCII characters in decimal.
- Whitespace.
- The height in pixels of the image, again in ASCII decimal.
- Newline or other single whitespace character.
- A raster of Height rows, in order from top to bottom. Each row is Width bits, packed 8 to a byte, with don't care bits to fill out the last byte in the row. Each bit represents a pixel: 1 is black, 0 is white. The order of the pixels is left to right. The order of their storage within each file byte is most significant bit to least significant bit. The order of the file bytes is from the beginning of the file toward the end of the file.

  A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.
- Characters from a '#' to the next end-of-line, before the width/height line, are comments and are ignored.

There is actually another version of the PBM format, even more more simplistic, more lavishly wasteful of space than PBM, called Plain PBM. Plain PBM actually came first, but even its inventor couldn't stand its recklessly squanderous use of resources after a while and switched to what we now know as the regular PBM format. But Plain PBM is so redundant -- so overstated -- that it's virtually impossible

to break. You can send it through the most liberal mail system (which was the original purpose of the PBM format) and it will arrive still readable. You can flip a dozen random bits and easily piece back together the original image. And we hardly need to define the format here, because you can decode it by inspection.

The difference is:

- There is exactly one image in a file.
- The 'magic number' is 'P1' instead of 'P4'.
- Each pixel in the raster is represented by a byte containing ASCII '1' or '0', representing black and white respectively. There are no fill bits at the end of a row.
- White space in the raster section is ignored.
- You can put any junk you want after the raster, if it starts with a white space character.
- No line should be longer than 70 characters.

Here is an example of a small image in the plain PBM format:

```
P1
# feep.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

You can generate the Plain PBM format from the regular PBM format (first image in the file only) with the **pnmtoplainpnm** program.

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a bitmap.

COMPATIBILITY

Before July 2000, there could be at most one image in a PBM file. As a result, most tools to process PBM files ignore (and don't read) any data after the first image.

SEE ALSO

libnetpbm(1), **pnm**(1), **pgm**(1), **ppm**(1), **pam**(1), **programs**that**process**PBM**(1)**

AUTHOR

Table Of Contents

NAME

pgm - Netpbm grayscale image format

DESCRIPTION

This program is part of **Netpbm**(1).

The PGM format is a lowest common denominator grayscale file format. It is designed to be extremely easy to learn and write programs for. (It's so simple that most people will simply reverse engineer it because it's easier than reading this specification).

A PGM image represents a grayscale graphic image. There are many psueudo-PGM formats in use where everything is as specified herein except for the meaning of individual pixel values. For most purposes, a PGM image can just be thought of an array of arbitrary integers, and all the programs in the world that think they're processing a grayscale image can easily be tricked into processing something else.

The name "PGM" is an acronym derived from "Portable Gray Map."

One official variant of PGM is the transparency mask. A transparency mask in Netpbm is represented by a PGM image, except that in place of pixel intensities, there are opaqueness values. See below.

The format definition is as follows. You can use the **libnetpbm**(1)**C**subroutine**library**to**conveniently** and accurately read and interpret the format.

A PGM file consists of a sequence of one or more PGM images. There are no data, delimiters, or padding before, after, or between images.

Each PGM image consists of the following:

- A 'magic number' for identifying the file type. A pgm image's magic number is the two characters 'P5'.
- Whitespace (blanks, TABs, CRs, LFs).
- A width, formatted as ASCII characters in decimal.
- Whitespace.
- A height, again in ASCII decimal.
- Whitespace.
- The maximum gray value (Maxval), again in ASCII decimal. Must be less than 65536, and more than zero.
- Newline or other single whitespace character.
- A raster of Height rows, in order from top to bottom. Each row consists of Width gray values, in order from left to right. Each gray value is a number from 0 through Maxval, with 0 being black and Maxval being white. Each gray value is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

  A row of an image is horizontal. A column is vertical. The pixels in the image are square and

contiguous.

- Each gray value is a number proportional to the intensity of the pixel, adjusted by the CIE Rec. 709 gamma transfer function. (That transfer function specifies a gamma number of 2.2 and has a linear section for small intensities). A value of zero is therefore black. A value of Maxval represents CIE D65 white and the most intense value in the image and any other image to which the image might be compared.

- Note that a common variation on the PGM format is to have the gray value be 'linear,' i.e. as specified above except without the gamma adjustment. **pnmgamma** takes such a PGM variant as input and produces a true PGM as output.

- In the transparency mask variation on PGM, the value represents opaqueness. It is proportional to the fraction of intensity of a pixel that would show in place of an underlying pixel. So what normally means white represents total opaqueness and what normally means black represents total transparency. In between, you would compute the intensity of a composite pixel of an 'under' and 'over' pixel as under * (1-(alpha/alpha_maxval)) + over * (alpha/alpha_maxval). Note that there is no gamma transfer function in the transparency mask.

- Characters from a '#' to the next end-of-line, before the maxval line, are comments and are ignored.

Note that you can use **pnmdepth** to convert between a the format with 1 byte per gray value and the one with 2 bytes per gray value.

There is actually another version of the PGM format that is fairly rare: 'plain' PGM format. The format above, which generally considered the normal one, is known as the 'raw' PGM format. See **pbm**(1)

for some commentary on how plain and raw formats relate to one another.

The difference in the plain format is:

- There is exactly one image in a file.
- The magic number is P2 instead of P5.
- Each pixel in the raster is represented as an ASCII decimal number (of arbitrary size).
- Each pixel in the raster has white space before and after it. There must be at least one character of white space between any two pixels, but there is no maximum.
- No line should be longer than 70 characters.

Here is an example of a small image in the plain PGM format:

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely

like a PGM.

COMPATIBILITY

Before April 2000, a raw format PGM file could not have a maxval greater than 255. Hence, it could not have more than one byte per sample. Old programs may depend on this.

Before July 2000, there could be at most one image in a PGM file. As a result, most tools to process PGM files ignore (and don't read) any data after the first image.

SEE ALSO

pnm(1), **pbm**(1), **ppm**(1), **pam**(1), **libnetpbm**(1), **programs**that**process**PGM**(1),**

AUTHOR

Table Of Contents

NAME

pnm - Netpbm superformat

DESCRIPTION

This program is part of **Netpbm**(1).

The PNM format is just an abstraction of the PBM, PGM, and PPM formats. I.e. the name 'PNM' refers collectively to PBM, PGM, and PPM.

The more general term 'Netpbm format' refers to the PNM formats plus PAM.

Note that besides being names of formats, PBM, PGM, PPM, and PNM are also classes of programs. A PNM program can take PBM, PGM, or PPM input. That's nothing special -- a PPM program can too. But a PNM program can often produce multiple output formats as well, and a PNM program can see the difference between PBM, PGM, and PPM input and respond to each differently whereas a PPM program sees everything as if it were PPM. This is discussed more in **the** description of the netpbm programs (1).

'pnm' also appears in the names of the most general **Netpbm**library**routines**(1),**some-**of**which**aren't**even** related to the PNM format.

SEE ALSO

ppm(1), **pgm**(1), **pbm**(1), **pbm**(1), **programs**that**process**PNM**(1), libnetpbm**(1)

AUTHOR

Table Of Contents

NAME

PPM - Netpbm color image format

DESCRIPTION

This program is part of **Netpbm**(1).

The PPM format is a lowest common denominator color image file format.

It should be noted that this format is egregiously inefficient. It is highly redundant, while containing a lot of information that the human eye can't even discern. Furthermore, the format allows very little information about the image besides basic color, which means you may have to couple a file in this format with other independent information to get any decent use out of it. However, it is very easy to write and analyze programs to process this format, and that is the point.

It should also be noted that files often conform to this format in every respect except the precise semantics of the sample values. These files are useful because of the way PPM is used as an intermediary format. They are informally called PPM files, but to be absolutely precise, you should indicate the variation from true PPM. For example, 'PPM using the red, green, and blue colors that the scanner in question uses.'

The name "PPM" is an acronym derived from "Portable Pixel Map." Images in this format (or a precursor of it) were once also called "portable pixmaps."

The format definition is as follows. You can use the **libnetpbm**(1)**C**subroutine**library**to**read**and interpret the format conveniently and accurately.

A PPM file consists of a sequence of one or more PPM images. There are no data, delimiters, or padding before, after, or between images.

Each PPM image consists of the following:

- A 'magic number' for identifying the file type. A ppm image's magic number is the two characters 'P6'.
- Whitespace (blanks, TABs, CRs, LFs).
- A width, formatted as ASCII characters in decimal.
- Whitespace.
- A height, again in ASCII decimal.
- Whitespace.
- The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
- Newline or other single whitespace character.

- A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

  A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

- In the raster, the sample values are 'nonlinear.' They are proportional to the intensity of the CIE Rec. 709 red, green, and blue in the pixel, adjusted by the CIE Rec. 709 gamma transfer function. (That transfer function specifies a gamma number of 2.2 and has a linear section for small intensities). A value of Maxval for all three samples represents CIE D65 white and the most intense color in the color universe of which the image is part (the color universe is all the colors in all images to which this image might be compared). CIE Rec. 709 is also known as

ITU-R BT.709.

Note that another popular color space is the newer sRGB. A common variation on PPM is to subsitute this color space for the one specified.

- Note that a common variation on the PPM format is to have the sample values be 'linear,' i.e. as specified above except without the gamma adjustment. **pnmgamma**

  takes such a PPM variant as input and produces a true PPM as output.

- Characters from a '#' to the next end-of-line, before the maxval line, are comments and are ignored.

Note that you can use **pnmdepth**

to convert between a the format with 1 byte per sample and the one with 2 bytes per sample.

There is actually another version of the PPM format that is fairly rare: 'plain' PPM format. The format above, which generally considered the normal one, is known as the 'raw' PPM format. See **pbm**(1)

for some commentary on how plain and raw formats relate to one another.

The difference in the plain format is:

- There is exactly one image in a file.
- The magic number is P3 instead of P6.
- Each sample in the raster is represented as an ASCII decimal number (of arbitrary size).
- Each sample in the raster has white space before and after it. There must be at least one character of white space between any two samples, but there is no maximum. There is no particular separation of one pixel from another -- just the required separation between the blue sample of one pixel from the red sample of the next pixel.
- No line should be longer than 70 characters.

Here is an example of a small pixmap in this format:

```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a pixmap.

COMPATIBILITY

Before April 2000, a raw format PPM file could not have a maxval greater than 255. Hence, it could not have more than one byte per sample. Old programs may depend on this.

Before July 2000, there could be at most one image in a PPM file. As a result, most tools to process PPM files ignore (and don't read) any data after the first image.

SEE ALSO

pnm(1), **pgm**(1), **pbm**(1), **pam**(1), **programs**that**process**PPM**(1)**

AUTHOR