NAME

zimg − render 2d data of arbitrary format

SYNOPSIS

zimg [*options*] [*file ...*]

DESCRIPTION

The **zimg** program generates png images from arbitrary formatted 2-D ascii or binary data. Optionally, jpeg, ppm or pgm images can be generated instead of png.

INPUT FILES

If no input files are given, **zimg** reads from stdin.

If the input file ends in **.gz**, **.bz2**, **.z** or **.Z** and your operating system supports the popen() call, the file is filtered through gunzip, bunzip2 or uncompress, respectively.

If an input file is not accessible, the availability of the file with the suffixes **.gz**, **.bz2**, **.z** or **.Z** appended is checked (**multiview**).

If no command line options are specified, **zimg** assumes plain ascii input, where the dimensions of the image are taken from the first two convertable integers and comment lines are marked by a hash '#' mark at the very first column of each line. If ascii data is read the first lines are parsed for zimg **modelines** until the first data point is read. See below in the section **MODELINES**.

Currently the 'ESRF data format' (edf and ehf) is the only binary input format which is detected automatically by **zimg** (if this option was not disabled at compile time). In the case of ehf, the data is read from a different file as specified by the tag *EDF_BinaryFileName*.

If the size was not specified and no binary option was selected, the first bytes of the source file are examined to check if the source is already a png file. This works only on systems where there's no difference between reading ascii and binary data (UNIX). If the source file is a png, all options except the colormap, labelling and output options are ignored. This is mainly useful for labelling a png file later (or a png file that was not created by **zimg**). It can also be used to just convert png to jpeg. Or you can specify a different colormap, to change the colormap entries of the source png. Note that this check for source png's is not available if **zimg** was linked with an older libgd version. If the source is a png, the **--dump-colormap** switch dumps the colormap entries of the png file instead of the zimg colors.

OPTIONS

In almost all cases you'll need to specify at least a few command line options to tell **zimg** something about the input format and specify the output style.

Note, that the order of command line options is important: in the case of mutually exclusive options the last one wins; in the case of a switch which turns an option on, a second occurence of this switch turns the option off again. Example: -x -x does nothing. This is useful for overwriting options which were set in the resource files ~/.zimgrc and .zimgrc (see section FILES). If an options overwrites another option (e.g. in '--red --blue', the second switch would overwrite the first), this is silently ignored.

Some switches can be defined more than once and are *position dependent*. These switches include

--differentiate

--curvature
--smooth
--logarithmic
--fabs
--absolute
--relative

This means that in
zimg --relative=10-60 --differentiate
the relative scaling applies to the raw data, but in
zimg --differentiate --relative=10-60
the relative scaling applies to the differentiated data.

Spaces separating the single-letter options from their parameters are optional.

The availability of some options are compile-time dependent. Use the **--help** option to get a list of valid options for your binary version of **zimg**.

-r, --size=*width*[,*height*]
specifies the *width* and *height* of the 2-D data. If *height* is not given it is set equal to *width*. The separator 'x' (instead of ',') can be used also for backwards compatibility.

-M, --matrix
Input is assumed to be in ascii matrix format. The ascii input file should have the data in rows and columns. The number of rows is the image dimension in vertical direction. The number of columns is the image in horizontal direction. The dimension of the data can be overwritten by the --*size* switch.

-p, --pattern=*pattern*
Mark comment blocks. This option is used when reading unformatted ascii data. Each line will be truncated at *pattern*. This options might be used repeatedly to specify more than one comment pattern. This option was not really tested and might be still buggy. For the future it is planned to include also regular expressions.

-n, --column=*number*
read column *number* (ascii input). Multiple columns can be selected by using this option repeatedly. Currently this is limited to the 32 first columns. If *number* is omitted, the data of all columns will be used.

--skip[=*bytes/lines*]
skip this amount of *lines* when reading ascii data. Skip *bytes*, if reading binary data. Note that the unit is *bytes*, irrespective of the input type (short ..). If the optional size argument is missing, the option is reset to its default. This default behavior is as follows: for binary data, all available data is read from the input stream and the header size (which will be skipped) is then calculated as the difference between the size of input data and the size of the image as specified by the -r switch. Or in short: the last <width>x<height> items of the input stream will be taken. If you don't like this feature you can disable it explicitly by using the switch '--skip=0'. For ascii data no skipping will be done by default.

--options=/*regexp*/*switches*
apply *switches*, if the input filename matches *regexp*. Especially useful if used in a **.zimgrc** file. The separator character can be any character which is not present in regexp, for example **--options**=#/*usr/local#switches*. See also the **--input-filter** switch. **--options** is only available if your system provides the POSIX regular expression functions regcomp / regexec (check **zimg -V** for availability).

--input-filter=*filter*
filter input files thru *filter*. If *filter* contains a %s, it is substituted with the file name, else the file-name is appended to the *filter*. This switch is especially useful if used in a **.zimgrc** file. Example: --options='/.int$/--input-filter=int < %s'. See also the **--options** switch. **--input-filter** is only available if your system supports the popen() call (check **zimg -V** for availability).

-o, --output=*path*
normally **zimg** writes to stdout. You don't ever need this option, if you redirect stdout to you destination file or pipe. If more than one input file is given, *path* should be an existing directory where the output files should be stored. In this case the output names are constructed from *path*, the input file name and a suffix ".gif", ".png", ".jpg", "ppm" or "pgm" depending on the gd version which was used at compile time and eventually the **-j**, **-P**, **--ppm** or **--pgm** switch. Using - for *path* will force stdout also for multiple input files; however, this makes sense only with *ppm* and *pgm* output image formats because png and jpeg formats do not allow multiple images in one file (only the first one would be read).

--big-endian
binary input data is big-endian. This option overwrites **--swap** and can be overwritten by **--swap** and **--little-endian**. This switch does nothing if the machine where **zimg** runs is also big-endian, otherwise the words get swapped.

--little-endian
binary input data is little-endian. This option is the opposite of big-endian, overwrites **--swap** and can be overwritten by **--swap** and **--big-endian**. This switch does nothing if the machine where **zimg** runs is also little-endian, otherwise the words get swapped.

--swap
swap bytes when reading binary input. This might be necessary when using binary files from different platforms. If **zimg** runs on a little-endian machine (e.g. Intel) **--swap** assumes the binary input data created by a big-endian machine and vice versa. All three switches **--swap**, **--big-endian** and **--little-endian** overwrite each other and therefore it is recommended to use only one of these switches.

-f, --float
read binary float (normally 4 bytes) data.

-d, --double
read binary double (normally 8 bytes) data.

--char
read signed char data.

--short
read signed short data.

--int read signed int data.

--long-int
read signed long-int data.

-c, --unsigned-char
read unsigned char data.

-s, --unsigned-short
read unsigned short data.

-i, --unsigned-int
read unsigned int data.

--unsigned-long-int
read unsigned long int data.

--complex-float[*=(abs/length|phase|real|imaginary)*]
input data is *binary* complex float. (ascii parsing of complex numbers is not implemented yet. The value of the switch determines how to display the complex numbers. The default is *abs* or the alias *length* (the length of the complex number).

--complex-double*=(abs/length|phase|real|imaginary)*
Like *--complex-float*, but for complex double *binary* input data.

--red
use a red scale color map.

--blue
use a blue scale color map.

--grey, --gray
use a grey scale color map.

-m, --colormap[*=path*]
use custom colormap from the file *path*. The file must hold r g b triplets with values ranging from 0 to 0xff (255). There must be exactly one triplet per line, where empty lines and lines beginning with a hash '#' mark are skipped. The maximum number of colors is defined in zimg.h and it is currently 240 (16 colors are reserved for "LINE COLORS"). The colormap file is searched in the current directory, then in ~/.zimg/cmap, then in /usr/local/share/zimg/cmap. If the optional argument *number* is omitted, the colormap is reset to the default.

-m, --colormap[*=red[,green[,blue]]*]
create a colormap using predefined colormap formulae. red, green and blue must be integers between -36 and 36. If *blue* and/or *green* are missing, they're set to green or red respectively (so a gray colormap value can be created by specifying red only). The numbers select one of the predefined formulae which are used to map the z value to a color intensity. Negative numbers invert the color intensity. Currently the following formulae are defined where x ranges from [0, 1]: If the optional argument *number* is omitted, the colormap is reset to the default. Note: these formulae are the same as in **gnuplot** (version 3.8 and later), where you can try and test them using commands *[set|show|test] palette*.

0 x = 0
1 x = 0.5
2 x = 1
3 x = x (identity)
4 x = x * x
5 x = x * x * x
6 x = x * x * x * x
7 x = sqrt(x)
8 x = sqrt(sqrt(x))
9 x = sin(90 * x)
10 x = cos(90 * x)

```
11  x = fabs(x - 0.5);
12  x = (2 * x - 1) * (2 * x - 1);
13  x = sin(180 * x);
14  x = fabs(cos(180 * x));
15  x = sin(360 * x);
16  x = cos(360 * x);
17  x = fabs(sin(360 * x));
18  x = fabs(cos(360 * x));
19  x = fabs(sin(720 * x));
20  x = fabs(cos(720 * x));
21  x = 3 * x;
22  x = 3 * x - 1;
23  x = 3 * x - 2;
24  x = fabs(3 * x - 1);
25  x = fabs(3 * x - 2);
26  x = (1.5 * x - 0.5);
27  x = (1.5 * x - 1.0);
28  x = fabs(1.5 * x - 0.5);
29  x = fabs(1.5 * x - 1.0);
30
      if (x <= 0.25)
        return 0;
      if (x >= 0.57)
        return 1;
      x = x / 0.32 - 0.78125;
31
      if (x <= 0.42)
        return 0;
      if (x >= 0.92)
        return 1;
      x = 2 * x - 0.84;
32
      if (x <= 0.42)
        x *= 4;
      else
        x = (x <= 0.92) ? -2 * x + 1.84 : x / 0.08 - 11.5;
33
      x = fabs(2 * x - 0.5);
34
      x = 2 * x;
35
      x = 2 * x - 0.5;
36
      x = 2 * x - 1;
```

-b, --cbox, --colorbox[=*n*]
draw a labelled colorbox right to the image. If the number of labels *n* isn't given, it is calculated automatically according to the image and font heights. Labelling can be turned off by specifying --colorbox=0.

--cbox-fmt, --cbox-format=*format*
format is a c sprintf format string for floats, e.g. %3.6g (see the sprintf(3) manual), which is used for formatting the colorbox legend. Turns on **--cbox** implicitely.

--cbox-label=*string*
Print *string* next to the colorbox (e.g. a unit for the colorbox numbers). Turns on **--cbox** implicitely. *string* can be a multiline string, see for example the **--label** switch.

--dump-colormap
dump a colormap to stdout as it can be read back with the *-m* switch. This can be useful for manually editing and reading back the colormap. The **--dump-colormap** switch disables most of the other switches -- no processing of data files is done. The only exception is if the source files are png files: in this case not the (specified) colormap of zimg, but the color entries of the source png are dumped.

-I, --invert
invert the selected color map.

-x, --xor[=*color*]
do an exclusive or with the specified *color* (defaults to white). For the gray scale color map this equivalent to the **--invert** switch.

--differentiate
apply a discrete differentiation (1'st derivative) to the data. This is a *position dependent* switch

-u, --curvature
display the curvature (2'nd derivative) of the data. This is a *position dependent* switch

--smooth[=*threshold*]
wipe out hot spots. The average and sigma of the nearest neighbors of each pixel are calculated. If the pixel's value is greater than ('threshold' * sigma + average), it will be set to the average of the neighbors. This is a *position dependent* switch

-z, --crange=*min,max*
sets data range for the color mappings to *min* to *min*. Default is autoscaled color range according to the image data values. The partial notations **--crange**=*min,* and **--crange**=*,max* can be used, denoting the missing limiting value to be autoscaled.

-l, --logarithmic[=*scale*]
use a logarithmic color scale where *scale* must be strictly positive. The data is scaled to the range 0 - *scale*, then the log1p () is taken. The default for *scale* is 1. The best way to understand this is to compare the results for different *scale* values. This is a *position dependent* switch

-a, --fabs
take the absolute value of the input data. The long option **--fabs** was named after the c function fabs(), because the **--absolute** switch is used for absolute scaling (see below). This is a *position dependent* switch

--absolute=*min,max*
set everything below *min* to *min* and every thing above *max* to *max*. The partial notations **--abs**=*min,* and **--abs**=*,max* can be used. If both *min* and *max* are given, a - can be used as separator instead of the ,. This is a *position dependent* switch

--relative=*min,max*
same as above, but *min* and *max* are to be given relative (in percent) to the data's *min* and *max*. This is a *position dependent* switch

-N, --no-data, --nda=*[val|@percent[,color]]*
set data points which are equal to *val* to the color *color* (default: black). If *val* isn't specified, or if *@percent* is specified instead of val, the nda value is determined automatically from the border values: The border value which occurs most frequently will be the NDA value. The value *@percent* if given, must be 0 < @percent <= 100. If *@percent* > 1 it is divided by 100, so @75 is

equivalent to @0.75. The *@percent* value gives the fraction of nda border values compared to the number of border pixels which must be reached at least for the automatically determined nda value to be valid. Example: suppose your image is 100 x 200 pixels large, so the number of border pixels is 600 - 2 = 598. If you specify --nda=@.75 and the most frequent border value occurs 350 times, the nda feature won't be applied, since 350 is smaller than 0.75 * 598.

-e, --expr=*string*

filter data through the c-style string expression *string*. This feature is compile-time dependent and only available, if *zimg -V* shows the string *dynaload*. The expression string *string* will be compiled on the fly by "gcc -c -O3 -shared -o". The expression is wrapped in a function which will be called for each data value *z (double)* with the current *x (unsigned int)*, *width (unsigned int)* and *y (unsigned int)*, *height (unsigned int)* values supplied. Example:

```
# zimg -e "cos(z) - (width - x)" ...
```

-R, --expr-source=*file.c*

Use *file.c* as input file name for compiling the expression. If the switch **--expr** is not given, **--expr-source** should point to an existing file which holds the c source for the expression function. If the switch **--expr** is given, it will be wrapped in a c function and stored in the *file.c* given by **--expr-source**. If **--expr-source** is not given, the expression given by **--expr** will be stored in a temporary c-file which will be deleted after the expression evaluation.

The switch **--expr-source** can be used for reusing the expression source:

```
# zimg -e "cos(z) - (width - x)" -R myfunc.c
# zimg -R myfunc.c ... file.dat
```

-O, --expr-object=*file.so*

Use *file.so* as output file name for compiling the expression. If none of the switches **--expr** and **--expr-source** is given, *file.so* should exist and be a valid object file which was probably compiled before by **zimg**. Shared objects are searched in the current directory, then in the directory ~/.zimg/expr, then in /usr/local/lib/zimg/expr and then in the search path of your dynamic loader (refer to the manual pages of dlopen() or shl_load(), depending on the implementation). If **--expr-object** is not given, the expression given by **--expr** or **--expr-source** will be compiled to a temporary shared object file which will be deleted after the expression evaluation.

If at least one of the switches **--expr** or **--expr-source** are given, the expression will be compiled to *file.so*. This can be used for compiling a shared object file for later use:

```
# zimg -e "cos(z) - (width - x)" -O myfunc.so
# zimg -O myfunc.so ... file.dat
```

For complicated expressions it might be useful to create the c-source for the expression with an editor and compile it 'by hand'. Please refer to the manual page of your c-compiler for how to create shared object files (for gcc it is the switch -shared). The shared object must export a function *zimg_expression* with *the zimg_expression_t* as given in zimg.h. Example:

```
#include <zimg.h>

float
zimg_expression(unsigned int x, unsigned int y,
   float z, const zimg_expression_info_t* info)
{
   unsigned int height = info->height;
   return z - (float)x * (float)(height - y);
}
```

-S, --scale=*xy*

-S, --scale=*x,y*

Scaling of the image. If only one number is given, it is used for both directions. Any non-numeric character may be used as separator.

-C, --crop[=*left-right*x*top-bottom*]

crop the raw data to the specified size. Note that the numbers given apply to the raw data, not to the eventually enlarged or binned image. If the optional argument is omitted, the data is cropped automatically: all data with the same value as the border is cropped.

-A, --align=*horizontal*[x*vertical*][,*bordercolor*]

align to an integer multiple of the specified pixels. If *vertical* is omitted, it it set to *horizontal*. Example: -A16 will pad the resulting image so that its (both) dimensions are multiples of 16. This is useful, if the images are used to create an mpeg sequence for example (otherwise mpeg_encode will cut the images down so that the dimensions are multiples of 16). The data is centered within the resulting image. The optional argument *bordercolor* must be given as 6-digit hexadecimal number, where the first digits are the red value, the second two digits the green value and the last two digits the blue value (so the color values for each color are between 0 and ff). Note that zimg will try to choose a color out of the existing color map which comes close to what you've specified, but depending on the color map you might not get exactly what you've requested. See also the --*textcolor* switch. If the *bordercolor* is not given, the image border color is chosen to be the most frequent color of the original data's border.

--contours=*levels*[,*log*][,*bg=color*|*fg=color*]

draw contour lines. This option is still experimental. Optional arguments are probably subject to future changes. The optional argument 'log' distributes the contour levels logarithmically over the image. The optional argument *bg=color* forces contour-only drawing i.e. colors the background with the specified color. This color must be given as 3 or 6 digit hex value or by the special keyword 'black'. *fg=color* uses the specified color as contour line color. bg and fg and should not be used together. The contour algorithm is pretty fast and does not spline when enlarging the image.

--interlace

write an interlaced image. **--interlace** is off by default. If you write a jpeg image using the --*jpeg* switch, the **--interlace** switch will be interpreted to write a progressive JPEG.

-g, --gif

write a gif instead of a png image. This switch is only available if libgd supports both png and gif.

-j, --jpeg[=quality]

write a jpeg instead of a png image. *quality* must be an integer number between (inclusive) 0 and 100. If *quality* is omitted, an appropriate default quality is used. Jpeg is only available, if **zimg** was compiled with libgd >= 1.8. If you have set image interlacing using the --*interlace* switch, this switch is interpreted to write a progressive JPEG. Some programs (e.g., Web browsers) can display progressive JPEGs incrementally; this can be useful when browsing over a relatively slow communications link, for example. Progressive JPEGs can also be slightly smaller than sequential (non-progressive) JPEGs.

-P, --ppmorpgm

write a portable pixmap (ppm) or portable graymap (pgm) instead of a png image according to --*gray* palette. These image formats are useful when piping **zimg**'s output directly to other programs (filters). Further, as there can be just a single png or jpeg image in one png or jpeg file, but several ppm or pgm images in one ppm or pgm file, it makes ppm and pgm output very useful for

producing image movies from a series of input data files, see examples below.

--ppm
write a portable pixmap (ppm) instead of a png image.

-P, --pgm
write a portable graymap (pgm) instead of a png image; this should be used only together with --*gray* option, otherwise a function of r,g,b values of a pixel is calculated as its gray value.

-t, --label=*[+-][+-]y,string*
print *string* at the specified position of the FINAL image. 'Final' means the image size with all padding and scaling applied. The string can be a multiline string separated by '\n', e.g. "this\nis\nmultiline\ntext". *x* and *y* can be negative coordinates, in which case they're interpreted as offsets of the right and bottom text bounding box from the right and bottom border of the image respectively. If you specify for example **--label**=*-1-1,string*, the *string* is entirely visible, having the lower right border of its bounding box in the bottom right corner on the image. Long multiline strings should preferably be passed by specifying this option several times (e.g. for each line separately) because of limitations of the option parser. See also STRING ESCAPES.

--vlabel=*[+-][+-]y,string*
same as the **--label** option, but prints the string vertically. See also STRING ESCAPES.

--legend=*string*
print *string* **outside** the image region black on white. The legend is either placed right or botto. of the image depending on the ratio of the resulting total image (smaller ratio wins). The string can be a multiline string separated by '\n', e.g. "this\nis\nmultiline\ntext". The extra space needed for the legend is reserved automatically. Long multiline strings should preferably be passed by specifying this option several times (e.g. for each line separately) because of limitations of the option parser. The *string* can be reset to zero length (e.g. for multifile input together with mode-lines) by using **--legend** w/o string argument. See also STRING ESCAPES.

-F, --font=*integer*
font size. Must be between (inclusive) 1 and 4. If the font size is not specified, it is chosen according to the image dimensions: a larger image gets a larger font size.

-T, --textcolor=*xxxxxx*
The *xxxxxx* must be given as 6-digit or 3-digit hexadecimal number, where the first digit(s) are the red value, the second two digit(s) the green value and the last two digit(s) the blue value (so the color values for each color are between 0 and ff). Note that zimg will try to choose a color out of the existing color map which comes close to what you've specified, but depending on the color map you might not get exactly what you've requested. Alternatively the color can be also specified by one of the predefined color names, see the section "LINE COLORS". If the text color is not given, it will be chosen automatically and should give normally a high contrast.

--line=*[+-]x1[+-]y1[+-]x2[+-]y2*[...]
draw a line or polyline from x1, y1 to x2, y2 (and more vertices) on the FINAL image. Example --line=+10+10+10+20 draws a horizontal line. All coordinates are absolute. Negative coordinates are interpreted from the right and bottom border respectively.

--rline=*[+-]x1[+-]y1[+-]x2[+-]y2*[...]
same as **--line**, but all coordinates except *x1* and *x2* are interpreted relatively to *x1* and *x2*.

-L, --license
Display some license information. **zimg** is published under the terms of a BSD type License.

-V, --version
Prints a version identifier for **zimg** to standard error. This is guaranteed to always contain the string "zimg" and the version number. Additionally the string "png" or "gif" indicates the output format which depends on the gd driver which was compiled in. Compile time options as edf support are appended in brackets.

-v, --verbose
switch on some informational output. Might not be too useful.

--statistics
print histogram like statistics of the **processed** data to stderr.

--help
Prints a help message to stderr and dies.

STRING ESCAPES

The switches **--label**, **--vlabel** and **--legend** accept some string escapes. The replacement of these string escape takes place **after** the data has been read and processed. String escape start with a percent sign '%' followed by a single character which indicates the type of the substitution. The following string escapes are supported:

%c Current time in RFC822-conformant format.

%f The filename of the input file

%m The minimum of the **processed** data.

%M The maximum of the **processed** data.

%i The integral of the **processed** data (valid data points only, see also the --no-data switch).

%{...}
Will be substituted with the standard output of the shell command which is given between the opening **{** and the closing **}**.

LINE COLORS

Some of the switches accept a color specification, e.g. **--no-data**, **--align**, **--contours**, **--textcolor** and **--xor**. These "line colors" are not chosen from the colormap which is used for rendering the data. Line colors can be specified either as 6-digit or 3-digit hex value or as one of the predefined strings "black", "white", "red", "green", "blue", "magenta", "cyan", "yellow". In the case of a 6-digit hex value, the first two digits represent the red value, the second two digits the green value and the last two digits the blue value. In the case of a 3-digit hex value, the first digit represent the red value, the second digit the green value and the last digit the blue value.

MODELINES

If the input data is ascii (or at least the header is ascii), **zimg** looks for **modelines** like some versions of vi or vim do. This is done only until the first data point is read. The format for modelines is as follows:

<space>zimg:<zimg options>:

You can have multiple modelines and even use a modeline to switch to binary input. Example:

```
#!/usr/local/bin/zimg
# zimg: -l10 -C50-550x200-450:
# zimg: -m7,5,15 -t -r547x633 -s --swap:
...
```

Note that in this case the order is important. Everything after the line containing the -s (unsigned short binary) is considered as binary data. So exchanging the two modelines like in the next example would interpret the second modeline as binary data (which is probably not what you want):

```
#!/usr/local/bin/zimg
# zimg: -m7,5,15 -t -r547x633 -s --swap:
# zimg: -l10 -C50-550x200-450:
...
```

EXAMPLES

zimg my.dat > my.png

This one of the most simple examples. The file my.dat is assumed to hold the dimensions of your 2-D data as the first two convertable integers. Everything before the first convertable integer is silently skipped, so it is also valid to have the dimension specifiers after a comment (hash) mark like this:

```
# 128 128
45.12
76.70
...
```

my.dat > my.png

If your operating system supports the specification of an interpreter by preceding it with #! you can make your data file an executable like this (you probably have also to change the permissions of my.dat e.g. ‘chmod +x my.dat’ on UNIX):

```
#!/usr/local/bin/zimg
# 128 128
# zimg:--red:
45.12
76.70
...
```

zimg my.dat.gz | xv -

pipe the output of **zimg** directly to your favorite image viewer (here: **xv**). The file will be filtered by gunzip.

zimg --red -r1200x1200 --skip=2400 -S0.5 -s l408_01.image | xv -

read binary data which hold 1200 x 1200 unsigned short data values. Skip the first 2400 bytes. Use the red scale color map. Scale the image down to 600 x 600 pixels and pipe it to xv.

Note that the skip option is not really necessary as long as the data is located at the very end of the input file. In this case the header size would be calculated by **zimg**.

zimg my.png --xlabel=10,10,’love is like oxygen’ --jpeg > my.jpg

read the png as source image, apply a label and write it as jpeg.

zimg -P -o- --crange=0,100 *.edf | animate -delay 50

read all edf files, and animate them with delay of 50 ms.

zimg -P -o- --scale=0.25 --gray *.edf | convert -delay 100 - movie.mng

read all edf files, scale them down and convert them into a gray scale movie (mng is an animated png). Note that not using *--crange* makes color scaling (z-range) individually autoscaled in each

image instead of using the same z-range for all images as in the previous example.

zimg for displaying function expressions w/o input data

First you've to create the c source which holds the function expression, for example sinc.c:

```
#include <zimg.h>
#include <math.h>
float
zimg_expression(unsigned int x, unsigned int y,
    float z, const zimg_expression_info_t* info)
{
  double width = (double)info->width;
  double height = (double)info->height;
  double xd = (double)x - width * 0.5;
  double yd = (double)y - height * 0.5;
  double value;
  xd *= 10 / width;
  yd *= 10 / height;
  value = sqrt(xd*xd+yd*yd);
  return value ? sin(value) / value : 1
}
```

This expression can be displayed w/o any data file by using /dev/zero as input source. As the input data is not used at all, you can use the **-c** switch -- tell **zimg** to treat the input as bytes. Furthermore you've to specify the dimension of the image with the **-r** switch:

```
# zimg -c -r300 --skip=0 -Rsinc.c -Osinc.so < /dev/zero | xv -
```

The object file sinc.so can now be reused in subsequent **zimg** runs, e.g.

```
# zimg -c -r500 --skip=0 -Osinc.so < /dev/zero | xv -
```

Note, that you have to tell **zimg** explicitly not to skip the file header, otherwise it will read from /dev/zero infinitely as it thinks it is the data header.

ENVIRONMENT VARIABLES

HOME

When this variable is found, **zimg** looks there for a file with the name **.zimgrc** which may contain command line options. If expression evaluation support is compiled in, object files are searched in the directory $HOME/.zimg/expr/.

ZIMG_VIEWER

used to view stdout, if zimg detects that stdout goes to a terminal (which is probably not what you want). Defaults to "xv -".

FILES

The following **zimgrc** resource files are read in this order:

```
/usr/local/etc/zimgrc
$HOME/.zimgrc
.zimgrc
zimgrc
```

These files are read before command line parsing is done, so command line
options given on the command line will eventually overwrite previously defined

settings of the resource files.

The **zimgrc** recource files might contain command line options separated by any white spaces including newlines. Blank lines and everything behind a hash '#' mark is considered to be a comment.

BUGS

probably.

SEE ALSO

The official **zimg** web site at:

http://zimg.sourceforge.net/

AUTHOR

Johannes Zellner <johannes@zellner.org>

HISTORY

This version of **zimg** was originally derived from the program z2ppm which was able to write portable pixmap (ppm) files.

CREDITS

Thomas Boutell <boutell@boutell.com>, the author of the **gd** driver.

Petr Mikulik <mikulik@physics.muni.cz> for the color code in getcolor.c, see also http://www.sci.muni.cz/~mikulik/gnuplot.html, the OS/2 port, and options *--crange*, *-P*, *--ppm*, *--pgm*, *-o-*.

Levente Novak <novak@jaguar.dote.hu> for the DOS/DJGPP port.