PNGwriter Quick Reference Manual

Version 0.4.4 (7/ VIII / 2004)

http://pngwriter.sourceforge.net/

Introduction

This is the PNGwriter *Quick Reference Manual. It is a summary of the functions that* PNGwriter *provides. For a more detailed description, see the* `pngwriter.h` *header file.*

General Notes

It is important to remember that all functions that accept an argument of type "`const char *`*" will also accept "*`char *`*". This is done so you can have a changing filename (to make many PNG images in series with a different name, for example), and to allow you to use* `string` *type objects which can be easily turned into* `const char *` *(if* `theString` *is an object of type* `string`*, then it can be used as a* `const char *` *by calling* `theString.c_str()`*).*

It is also important to remember that whenever a function has a colour coeffiecient as its argument, that argument can be either an `int` *from* `0` *to* `65535` *or a* `double` *from* `0.0` *to* `1.0`*. You must make sure that you are calling the function with the type that you want. Remember that* `1` *is an* `int`*, while* `1.0` *is a* `double`*, and will thus determine what version of the function will be used. Do not make the mistake of calling for example* `plot(x, y, 0.0, 0.0, 65535)`*, because there is no* `plot(int, int, double, double, int)`*. Also, please note that* `plot()` *and* `read()` *(and the functions that use them internally) are protected against entering, for example, a colour coefficient that is over* `65535` *or over* `1.0`*. Similarly, they are protected against negative coefficients.* `read()` *will return* `0` *when called outside the image range. This is actually useful as zero-padding should you need it.*

Compilation

A typical compilation would look like this:

```
g++ my_program.cc -o my_program `freetype-config --cflags`
     -I/usr/local/include  -L/usr/local/lib -lpng -lpngwriter -lz -lfreetype
```

If you did not compile PNGwriter *with FreeType support, then remove the FreeType-related flags and add* `-DNO_FREETYPE` *above.*

Constructor

The constructor requires the width and the height of the image, the background colour for the image and the filename of the file (a pointer or simply "`myfile.png`*"). The default constructor creates a* PNGwriter *instance that is* `250x250`*, white background, and filename* `"out.png"`*.* ***Tip****: The filename can be given as easily as: pngwriter* `mypng(300, 300, 0.0, "myfile.png");` ***Tip****: If you are going to create a* PNGwriter *instance for reading in a file that already exists, then width and height can be 1 pixel, and the size will be automatically adjusted once you use readfromfile().*

```
pngwriter();
```

```
pngwriter(const pngwriter &rhs);
pngwriter(int width, int height, int backgroundcolour, char * filename);
pngwriter(    int width, int height, double backgroundcolour,
              char * filename);
pngwriter(    int width, int height, int backgroundcolour,
              const char * filename);
pngwriter(    int width, int height, double backgroundcolour,
              const char * filename);
```

Assignment Operator

PNGwriter *overloads the assignment operator =.*

```
pngwriter & operator = (const pngwriter & rhs);
```

Plot

The pixels are numbered starting from (1, 1) and go to (width, height). If the colour coefficients are of type `int`, *they go from* `0` *to* `65535`. *If they are of type* `double`, *they go from* `0.0` *to* `1.0`. ***Tip****: To plot using red, then specify* `plot(x, y, 0.0, 0.0, 1.0)`. *To make pink, just add a constant value to all three coefficients, like this:* `plot(x, y, 0.4, 0.4, 1.0)`. ***Tip****: If nothing is being plotted to your PNG file, make sure that you remember to close() the instance before your program is finished, and that the x and y position is actually within the bounds of your image. If either is not, then* PNGwriter *will not complain-- it is up to you to check for this!* ***Tip****: If you try to plot with a colour coefficient out of range, a maximum or minimum coefficient will be assumed, according to the given coefficient. For example, attempting to plot* `plot(x, y, 1.0,-0.2,3.7)` *will set the green coefficient to* `0.0` *and the red coefficient to* `1.0`.

```
void plot(int x, int y, int red, int green, int blue);
void plot(int x, int y, double red, double green, double blue);
```

Plot HSV

With this function a pixel at coordinates `(x, y)` *can be set to the desired colour, but with the colour coefficients given in the Hue, Saturation, Value colourspace.*

```
void plotHSV(int x, int y, double hue, double saturation, double value);
void plotHSV(int x, int y, int hue, int saturation, int value);
```

Read

With this function we find out what colour the pixel `(x, y)` *is. If "*`colour`*" is* `1`, *it will return the red coefficient, if it is set to* `2`, *the green one, and if it set to* `3`, *the blue colour coefficient will be returned, and this returned value will be of type* `int` *and be between 0 and 65535.*

```
int read(int x, int y, int colour);
```

Read, Average

Same as the above, only that the average of the three colour coefficients is returned.

```
int read(int x, int y);
```

dRead

Same as `read()`*, but the returned value will be of type* `double` *and be between* `0.0` *and* `1.0`.

```
double dread(int x, int y, int colour);
```

dRead, Average

Same as the above, only that the average of the three colour coefficients is returned.

```
double dread(int x, int y);
```

Read HSV

With this function we find out what colour the pixel (x, y) is, but in the Hue, Saturation, Value colourspace. If "colour" is `1`*, it will return the Hue coefficient, if it is set to* `2`*, the Saturation one, and if it set to* `3`*, the Value colour coefficient will be returned, and this returned value will be of type int and be between 0 and 65535.*

```
int readHSV(int x, int y, int colour);
```

dRead HSV

Same as the above, but the returned value will be of type double and be between `0.0` *and* `1.0`.

```
double dreadHSV(int x, int y, int colour);
```

Clear

The whole image is set to black.

```
void clear(void);
```

Close

Close the instance of the class, and write the image to disk.

```
void close(void);
```

Rename

To rename the file once an instance of pngwriter has been created. If the argument is a `long unsigned int`*, for example* `77`*, the filename will be changed to* `0000000077.png`

```
void pngwriter_rename(char * newname);
void pngwriter_rename(const char * newname);
void pngwriter_rename(long unsigned int index);
```

Figures

These functions draw basic shapes. Available in both `int` *and* `double` *versions.*

```
void line(    int xfrom, int yfrom,
              int xto, int yto,
              int red, int green, int blue);
void line(    int xfrom, int yfrom,
              int xto, int yto,
              double red, double green,double blue);
void square(  int xfrom, int yfrom,
              int xto, int yto,
              int red, int green, int blue);
void square(  int xfrom, int yfrom,
```

```
              int xto, int yto,
              double red, double green, double blue);
void filledsquare( int xfrom, int yfrom,
                   int xto, int yto,
                   int red, int green,int blue);
void filledsquare( int xfrom, int yfrom,
                   int xto, int yto,
                   double red, double green,double blue);
void circle(  int xcentre, int ycentre, int radius,
              int red, int green, int blue);
void circle(  int xcentre, int ycentre, int radius,
              double red, double green, double blue);
void filledcircle( int xcentre, int ycentre, int radius,
                   int red, int green, int blue);
void filledcircle( int xcentre, int ycentre, int radius,
                   double red, double green, double blue);
```

Bezier Curve

(After Frenchman Pierre Bézier from Regie Renault) A collection of formulae for describing curved lines and surfaces, first used in 1972 to model automobile surfaces. (from the The Free On-line Dictionary of Computing) See http://www.moshplant.com/direct-or/bezier/ *for one of many available descriptions of bezier curves. There are four points used to define the curve: the two endpoints of the curve are called the anchor points, while the other points, which define the actual curvature, are called handles or control points. Moving the handles lets you modify the shape of the curve.*

```
void bezier(  int startPtX, int startPtY,
              int startControlX, int startControlY,
              int endPtX, int endPtY,
              int endControlX, int endControlY,
              double red, double green, double blue);

void bezier(  int startPtX, int startPtY,
              int startControlX, int startControlY,
              int endPtX, int endPtY,
              int endControlX, int endControlY,
              int red, int green, int blue);
```

Read From File

Open the existing PNG image, and copy it into this instance of the class. It is important to mention that not all colour types and bit depths are supported. Try to make sure that your PNG image is of bit depth `8` *or* `16`.

```
void readfromfile(char * name);
void readfromfile(const char * name);
```

Get Height

When you open a PNG with `readfromfile()` *you can find out its height with this function.*

```
int getheight(void);
```

Get Width
When you open a PNG with `readfromfile()` *you can find out its width with this function.*
```
int getwidth(void);
```

Write PNG
Writes the PNG image to disk. You can still change the PNGwriter *instance after this.*
```
void write_png(void);
```

Set Compression Level
Set the compression level that will be used for the image. `-1` *is default,* `0` *is none,* `9` *is best compression.*
```
void setcompressionlevel(int level);
```

Get Bit Depth
When you open a PNG with `readfromfile()` *you can find out its bit depth with this function.*
```
int getbitdepth(void);
```

Get Colour Type
When you open a PNG with `readfromfile()` *you can find out its colour type.*
```
int getcolortype(void);
```

Set Gamma Coeff
Set the image's gamma (file gamma) coefficient. The default value of `0.5` *should be fine.*
```
void setgamma(double gamma);
```

Get Gamma Coeff
Get the image's gamma coefficient. This is experimental.
```
double getgamma(void);
```

Set Text
Sets the text information in the PNG header. If it is not called, the default is used.
```
void settext( char * title, char * author,
              char * description, char * software);
void settext( const char * title, const char * author,
              const char * description, const char * software);
```

Version Number
Returns the PNGwriter *version number.*
```
double version(void);
```

Plot Text
Uses the Freetype2 library to plot text in the image. `face_path` *is the file path to a TrueType font file (*`.ttf`*) (FreeType2 can also handle other types).* `fontsize` *specifices the approximate height of the rendered font in pixels.* `x_start` *and* `y_start` *specify the placement of the lower, left corner of the text string.* `angle` *is the text angle in radians.* `text` *is the text to be*

rendered. The colour coordinates can be `doubles` *from 0.0 to 1.0 or* `ints` *from 0 to 65535.* ***Tip****:* PNGwriter *installs a few fonts in* `/usr/local/share/pngwriter/fonts` *to get you started.* ***Tip****: Remember to add* `-DNO_FREETYPE` *to your compilation flags if* PNGwriter *was compiled without FreeType support.*

```
void plot_text(    char * face_path, int fontsize,
                   int x_start, int y_start,   double angle,
                   char * text,
                   double red, double green, double blue);
void plot_text(    char * face_path, int fontsize,
                   int x_start, int y_start, double angle,
                   char * text,
                   int red, int green, int blue);
```

Plot UTF-8 Text

Same as the above, but the text to be plotted is encoded in UTF-8. Why would you want this? To be able to plot all characters available in a large TrueType font, for example: for rendering Japenese, Chinese and other languages not restricted to the standard 128 character ASCII space. ***Tip****: The quickest way to get a string into UTF-8 is to write it in an adequate text editor, and save it as a file in UTF-8 encoding, which can then be read in in binary mode.*

```
void plot_text_utf8(   char * face_path, int fontsize,
                       int x_start, int y_start, double angle,
                       char * text,
                       double red, double green, double blue);
void plot_text_utf8(   char * face_path, int fontsize,
                       int x_start, int y_start, double angle,
                       char * text,
                       int red, int green, int blue);
```

Bilinear Interpolation of Image

Given a floating point coordinate (`x` *from 0.0 to width,* `y` *from 0.0 to height), this function will return the interpolated colour intensity specified by colour (where red =* `1`*, green =* `2`*, blue =* `3`*).* `bilinear_interpolate_read()` *returns an* `int` *from 0 to 65535, and* `bilinear_interpolate_dread()` *returns a* `double` *from 0.0 to 1.0.* ***Tip****: Especially useful for enlarging an image.*

```
int bilinear_interpolation_read(double x, double y, int colour);
double bilinear_interpolation_dread(double x, double y, int colour);
```

Plot Blend

Plots the colour given by red, green blue, but blended with the existing pixel value at that position. opacity is a `double` *that goes from* `0.0` *to* `1.0`*. For example,* `0.0` *will not change the pixel at all, and* `1.0` *will plot the given colour. Anything in between will be a blend of both pixel levels.*

```
void plot_blend(   int x, int y, double opacity,
                   int red, int green, int blue);
void plot_blend(   int x, int y, double opacity,
                   double red, double green,   double blue);
```

Invert

Inverts the image in RGB colourspace.

```
void invert(void);
```

Resize Image

Resizes the PNGwriter *instance. Note: All image data is set to black (this is a resizing, not a scaling, of the image).*

```
void resize(int width, int height);
```

Boundary Fill

All pixels adjacent to the start pixel will be filled with the fill colour, until the boundary colour is encountered. For example, calling `boundary_fill()` *with the boundary colour set to red, on a pixel somewhere inside a red circle, will fill the entire circle with the desired fill colour. If, on the other hand, the circle is not the boundary colour, the rest of the image will be filled. The colour components are either* `doubles` *from* `0.0` *to* `1.0` *or* `int`*s from* `0` *to* `65535`.

```
void boundary_fill(int xstart, int ystart,
         double boundary_red, double boundary_green, double boundary_blue,
         double fill_red, double fill_green, double fill_blue);
void boundary_fill( int xstart, int ystart,
                    int boundary_red, int boundary_green, int boundary_blue,
                    int fill_red, int fill_green, int fill_blue) ;
```

Flood Fill

All pixels adjacent to the start pixel will be filled with the fill colour, if they are the same colour as the start pixel. For example, calling `flood_fill()` *somewhere in the interior of a solid blue rectangle will colour the entire rectangle the fill colour. The colour components are either doubles from* `0.0` *to* `1.0` *or ints from* `0` *to* `65535`.

```
void flood_fill(   int xstart, int ystart,
                   double fill_red, double fill_green, double fill_blue);
void flood_fill(   int xstart, int ystart,
                   int fill_red, int fill_green, int fill_blue) ;
```

Polygon

This function takes an array of integer values containing the coordinates of the vertexes of a polygon. Note that if you want a closed polygon, you must repeat the first point's coordinates for the last point. It also requires the number of points contained in the array. For example, if you wish to plot a triangle, the array will contain `6` *elements, and the number of points is* `3`. *Be very careful about this; if you specify the wrong number of points, your program will either segfault of produce points at nonsensical coordinates. The colour components are either* `double`*s from* `0.0` *to* `1.0` *or* `int`*s from* `0` *to* `65535`.

```
void polygon( int * points, int number_of_points,
              double red, double green, double blue);
void polygon( int * points, int number_of_points,
              int red, int green, int blue);
```

Plot CMYK

Plot a point in the Cyan, Magenta, Yellow, Black colourspace. Please note that this colourspace

is lossy, i.e. it cannot reproduce all colours on screen that RGB can. The difference, however, is barely noticeable. The algorithm used is a standard one. The colour components are either doubles from `0.0` *to* `1.0` *or ints from* `0` *to* `65535`.

```
void plotCMYK(     int x, int y,
                   double cyan, double magenta, double yellow, double black);
void plotCMYK(     int x, int y,
                   int cyan, int magenta, int yellow, int black);
```

Read CMYK, double version

Get a pixel in the Cyan, Magenta, Yellow, Black colourspace. if 'colour' is `1`*, the Cyan component will be returned as a double from* `0.0` *to* `1.0`*. If 'colour is* `2`*, the Magenta colour component will be returned, and so on, up to* `4`.

```
double dreadCMYK(int x, int y, int colour);
```

Read CMYK

Same as the above, but the colour components returned are an int from `0` *to* `65535`.

```
int readCMYK(int x, int y, int colour);
```