Manual de Referencia Rápida de PNGwriter

Versión 0.4.4 (7/ VIII / 2004)

http://pngwriter.sourceforge.net/

Introducción

Este es el Manual de Referencia Rápida de PNGwriter. *Es un resumen de las funciones de* PNGwriter. *Para una descripción más detallada, ver el archivo header* `pngwriter.h`

Notas Generales

Es importante recordar que todas las funciones que aceptan un argumento del tipo "`const char *`*" tambien aceptaran "*`char *`*". Esto es así para poder tener un nombre de archivo cambiante (para crear varias imágenes PNG en serie con nombres distintos, por ejemplo), y para permitir el uso de objetos de tipo string, los que pueden ser fácilmente convertidos en* `const char *` *(si* `elString` *es un objeto de tipo string, entonces puede ser usado como un* `const char *` *llamando a* `elString.c_str()`*).*

Tambien es importante recordar que cuando una función tiene un coeficiente de color como argumento, ese argumento puede ser un `int` *de* `0` *a* `65535` *o un* `double` *de* `0.0` *a* `1.0`. *Debes asegurarte que estás llamando a la función adecuada. Recuerda que 1 es un int, mientras que 1.0 e sun double, y esto determinará la versión de la función que usarás. No cometas el error de llamar por ejemplo a* `plot(x, y, 0.0, 0.0, 65535)`, *porque no hay un plot(int, int, double, double, int). Además, nota que* `plot()` *y* `read()` *(y las funciones que las usan internamente) están protegidas contra el uso de argumentos negativos o fuera de rango.* `read()` *devolverá* `0` *cuando se llame fuera del rango de la imagen. Esto incluso puede ser útil como 'zero-padding' si es que lo necesitaras.*

Compilación

Una compilación típica se verá así:

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

Si no compilaste PNGwriter *con soporte de FreeType, entonces quita los flags que tengan que ver con FreeType y agrega* `-DNO_FREETYPE`.

Constructor

El constructor requiere el ancho y la altura de la imagen, el color de fondo para la imagen, y el nombre del archivo (un puntero o simplemente "`miarchivo.png`*"). El constructor por defecto crea una instancia de* PNGwriter *de* `250x250` *pixeles, fondo blanco y nombre "out.png". **Sugerencia**: el nombre del archivo puede ser especificado tan fácilmente como escribir* `mypng(300, 300, 0.0, "myfile.png");`. ***Sugerencia**: Si vas a crear una instancia de* PNGwriter *para abrir con ella una imagen PNG ya existente, entonces la altura y el ancho pueden ser* `1` *pixel, y el tamaño quedará determinado automáticamente luego de abrir el PNG con readfromfile().*

```
pngwriter();
```

```
pngwriter(const pngwriter &rhs);
pngwriter(    int ancho, int altura, int backgroundcolour, char * filename);
pngwriter(    int ancho, int altura, double backgroundcolour, char *
              filename);
pngwriter(    int ancho, int altura, int backgroundcolour, const char *
              filename);
pngwriter(    int ancho, int altura, double backgroundcolour,
              const char * filename);
```

Operador de Asignación

PNGwriter *sobrecarga el operador de asignación =.*

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

Plotear

Los pixeles se enumeran partiendo desde `(1, 1)` *y van hasta (*`ancho`, `altura`*). Al igual como sucede con muchas funciones en* PNGwriter*, esta función ha sido sobrecargada para aceptar argumentos tipo int o double para los coeficientes de color. Si son de tipo* `int`*, van desde* `0` *a* `65535`*. Si son de tipo* `double`*, van desde* `0.0` *hasta* `1.0`*.* ***Sugerencia****: Para plotear usando rojo, escribe* `plot( x, y, 0.0, 0.0, 1.0).` *Para hacer rosado, agrega una constante a los tres coeficientes:* `plot( x, y, 0.4, 0.4, 1.0).` ***Sugerencia****: Si no obtienes nada en tu imagen PNG, acérdate de llamar a la función close() de la instancia antes que tu programa termine, y que la posición x e y de los pixeles que ploteas esté dentro del rango de tu imagen. Si no lo están, entonces* PNGwriter *no se quejará-- es responsabilidad tuya asegurar que esto no suceda.* ***Sugerencia****: Si tratas de plotear con un coeficiente de color fuera de rango, se asumirá un coeficiente de color máximo o mínimo, de acuerdo al coeficiente usado. Por ejemplo, el intentar plotear* `plot(x, y, 1.0, -0.2, 3.7)` *lo hará usando* `0.0` *para el coeficiente verde y* `1.0` *para el coeficiente rojo.*

```
void plot(int x, int y, int rojo, int verde, int azul);
void plot(int x, int y, double rojo, double verde, double azul);
```

Plotear HSV

Igual que el anterior, pero con esta función el color de un pixel en la posición `(x, y)` *puede ser cambiado en el espacio de colores de Hue, Saturation, Value. Los coeficientes de color van desde* `0` *a* `65535` *si son de tipo* `int`*, o desde* `0.0` *hasta* `1.0` *si son de tipo* `double`*.*

```
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

Con esta función averiguamos el color del pixel `(x, y).` *Si "color" es* `1`*, devolverá el coeficiente de color rojo, si es* `2`*, el verde, si es* `3`*, el azul. El valor devuelto será de tipo* `int` *y estará entre* `0` *y* `65535`*.*

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

Read, Promedio

Lo mismo que el anterior, pero el promedio de los tres coeficientes de color es devuelto.

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

dRead
Igual que Read, pero el valor devuelto será de tipo `double` *y estará entre* `0.0` *y* `1.0`.
```
double dread(int x, int y, int color);
```

dRead, Promedio
Lo mismo que el anterior, pero el promedio de los tres coeficientes de color es devuelto.
```
double dread(int x, int y);
```

Read HSV
Con esta función averiguamos el color del pixel `(x, y)`*, pero en el espacio de colores de HSV. Si "color" es* `1`*, devolverá el coeficiente de color Hue, si es* `2`*, el Saturation, si es* `3`*, el Value. El valor devuelto será de tipo* `int` *y estará entre* `0` *y* `65535`.
```
int readHSV(int x, int y, int color);
```

dRead HSV
Igual que el anterior, pero el valor devuelto será de tipo `double` *y estará entre* `0.0` *y* `1.0`.
```
double dreadHSV(int x, int y, int color);
```

Borrar todo
Toda la imagen se pinta de negro.
```
void clear(void);
```

Cerrar
Cerrar la instancia de la clase, y escribir la imagen al disco.
```
void close(void);
```

Cambiar el nombre
Para cambiar el nombre del archivo una vez que la instancia de PNGwriter *ha sido creada. Si el argumento es un* `long unsigned int`*, por ejemplo* `77`*, el nombre del archivo será* `000000077.png`.
```
void pngwriter_rename(char * nuevonombre);
void pngwriter_rename(const char * nuevonombre);
void pngwriter_rename(long unsigned int index);
```

Figuras
Disponibles en versión `int` *y* `double`
```
void line(    int xdesde, int ydesde,
              int xhasta, int yhasta,
              int rojo, int verde,int azul);
void line(    int xdesde, int ydesde,
              int xhasta, int yhasta,
              double rojo, double verde,double azul);
void square(  int xdesde, int ydesde,
              int xhasta, int yhasta,
```

```
              int rojo, int verde,int azul);
void square(  int xdesde, int ydesde,
              int xhasta, int yhasta,
              double rojo, double verde,double azul);
void filledsquare( int xdesde, int ydesde,
                   int xhasta, int yhasta,
                   int rojo, int verde,int azul);
void filledsquare( int xdesde, int ydesde,
                   int xhasta, int yhasta,
                   double rojo, double verde,double azul);
void circle(  int xcentro, int ycentro, int radio,
              int rojo, int verde, int azul);
void circle(  int xcentro, int ycentro, int radio,
              double rojo, double verde, double azul);
void filledcircle( int xcentro, int ycentro, int radio,
                   int rojo, int verde, int azul);
void filledcircle( int xcentro, int ycentro, int radio,
                   double rojo, double verde, double azul);
```

Curva Bezier

(llamado así en honor del frances Pierre Bézier de Regie Renault) Una colección de fórmulas para describir líneas y superficies curvas, usados por primera vez en 1972 para modelar las lineas curvas de automóviles. (de The Free On-line Dictionary of Computing)
Ver http://www.moshplant.com/direct-or/bezier/ *para una de las muchas descripciones disponibles acerca de las curvas de bezier. Hay cuatro puntos usados para definir la curva: los dos extremos se llaman los puntos de anclaje, mientras que los otros puntos, que describen la curvatura, se llaman puntos de control. Moviendo los puntos de control podemos modificar la forma de la curva.*

```
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);
```

Leer Desde Archivo

Abre el archivo PNG ya existente y lo incorpora a su almacenamiento.

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

Averiguar Altura

Cuando abres un PNG ya existente, puedes averiguar su altura con esta función.

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

Averiguar Ancho
Cuando abres un PNG ya existente, puedes averiguar su ancho con esta función.
```
int getwidth(void);
```

Escribir PNG
Escribe el PNG al disco. Después de esto, todavía puedes seguir alterando la instancia de PNGwriter.
```
void write_png(void);
```

Elegir Nivel de Compresión
Elegir el nivel de compresion que será usada para la imagen. `-1` *selecciona la compresión por defecto,* `0` *es ninguna,* `9` *es la mejor compresión.*
```
void setcompressionlevel(int nivel);
```

Averiguar Bit Depth
Cuando abres un PNG ya existente, puedes averiguar su profundidad de bits con esta función.
```
int getbitdepth(void);
```

Averiguar Colour Type
Cuando abres un PNG ya existente, puedes averiguar su color type con esta función.
```
int getcolortype(void);
```

Cambiar Gamma
Cambiar el gamma de la imagen (filegamma). El valor prefijado de 0.5 debería estar bien.
```
void setgamma(double gamma);
```

Averiguar Gamma
Averigua el coeficiente de gamma de la imagen. Esto es experimental.
```
double getgamma(void);
```

Cambiar Texto
Cambia el texto contenido en el header del archivo PNG. Si esta función no se llama, se usara el texto por defecto.
```
void settext( char * title, char * author,
              char * description, char * software);
void settext( const char * title, const char * author,
              const char * description, const char * software);
```

Versión
Da la versión de PNGwriter.
```
double version(void);
```

Plotear Texto
Usa la librería FreeType2 para plotear texto en la imagen. `face_path` *es el path donde se*

encuentra un archivo de font TrueType (`.ttf`*). (FreeType2 también puede usar otros tipos).* `fontsize` *especifica el tamaño aproximado del font dibujado, en pixeles.* `x_start` *e* `y_start` *especifican la ubicación de la esquina inferior izquierda del string de texto.* `angle` *es el ángulo que el texto formará con la horizontal, en radiantes.* `text` *es el texto por dibujar. Las coordenadas de color pueden ser doubles de* `0.0` *a* `1.0` *o* `ints` *de* `0` *a* `65535`*. **Sugerencia**:* PNGwriter *instala algunos fonts en* `/usr/local/share/pngwriter/fonts` *para que puedas comenzar. Otra sugerencia: recuerda de agregar* `-DNO_FREETYPE` *a tus flags de compilación si* PNGwriter *fue compilado sin soporte de FreeType.*

```
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);
```

Plotear Texto UTF-8

Igual que el anterior, pero el texto que se ploteará está codificado en UTF-8. Por qué querrías algo así? Para poder plotear caracteres con acentos, y todos los caracteres disponibles en un font TrueType, como por ejemplo para plotear texto en Japonés, Chino y otros idiomas no restringidos al espacio ASCII estándar de 128 caracteres. ***Sugerencia**: La manera más rápida de convertir un string a UTF-8 es escribirlo en un editor de texto adecuado y luego guardarlo como un archivo con codificación UTF-8, el cual posteriormente podrá ser leído en modo binario por tu programa.*

```
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);
```

Interpolación Bilineal de la Imagen

Dado una coordenada de punto flotante (x de `0.0` *a* `width`*, y de* `0.0` *a* `height`*), esta función devolverá la intensidad de color interpolada, determinando el color por colour (donde rojo =* `1`*, verde =* `2` *y azul =* `3`*).* `bilinear_interpolate_read()` *devuelve un* `int` *de* `0` *a* `65535`*, y* `bilinear_interpolate_dread()` *devuelve un* `double` *de* `0.0` *a* `1.0`*.* ***Sugerencia**: Es especialmente útil para agrandar una imagen.*

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

Plotear Combinado

Plotea el color dado por `red`*,* `green`*,* `blue`*, pero combinado con el valor del pixel existente en esa posición.* `opacity` *es un* `double` *de* `0.0` *a* `1.0`*. Por ejemplo,* `0.0` *no cambiará el pixel, y* `1.0` *ploteará el color dado. Cualquier cosa entre estos dos valores resultará en una combinación de los dos colores.*

```
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);
```

Invertir

Invierte la imagen en el espacio de colores RGB.

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

Resize Image

Cambia el tamaño de la instancia de PNGwriter. *Nota: toda la imagen se resetea a negro (esto es un cambio de tamaño, no un re-escalamiento).*

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

Rellenar Con Borde

Todos los pixeles adyacentes al pixel de partida se llenarán con el color de relleno, hasta que se encuentren pixeles del color de borde. Por ejemplo, llamando a `boundary_fill()` *con rojo para el color de borde, en un pixel en alguna parte en el interior de un círculo rojo, llenará el círculo con el color de llenado. Si, por el contrario, el círculo no es del color de borde, el resto de la imagen será pintada. Los coeficientes de color son double desde* `0.0` *a* `1.0` *o int desde* `0` *hasta* `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) ;
```

Rellenar

Todos los pixeles adyacentes al pixel de partida se llenarán con el color de llenado, si son del mismo color que el pixel de partida. Por ejemplo, llamando a `flood_fill()` *alguna parte en el interior de un rectángulo sólido azul lo coloreará entero del color de relleno. Los componentes de color son double desde* `0.0` *hasta* `1.0` *o int desde* `0` *hasta* `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) ;
```

Polígono

Esta función toma un arreglo de valores enteros representando las corrdenadas de los vértices de un polígono. Nota que si quieres un polígono cerrado, debes repetir las coordenadas del primer punto al final del arreglo. También debes especificar cuantos puntos contiene el arreglo. Por ejemplo, si quieres plotear un triángulo, entonces el arreglo contendrá `6` *elementos, y el número de los puntos es* `3`. *Ten mucho cuidado con esto: si especificas el numero equivocado de puntos, tu programa tirará un segfault o ploteará vértices en coordenadas insensatas. Los componentes de color son double desde* `0.0` *hasta* `1.0` *o int desde* `0` *hasta* `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);
```

Plotear CMYK

Plotea un pixel en el espacio de colores Cyan, Magenta, Amarillo, Negro. Nota que este espacio de colores tiene pérdidas, es decir, no puede reproducir todos los colores que puede reproducir RGB. La diferencia, sin embargo, es casi indistinguible. El algoritmo usado es estándar. Los componentes de color son double desde `0.0` *hasta* `1.0` *o int desde* `0` *hasta* `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);
```

Leer CMYK, versión double

Averigua el color de un pixel en el espacio de colores Cyan, Magenta, Amarillo, Negro. Si 'colour' es `1`*, se devolverá el componente Cyan del pixel. Si 'colour' es* `2`*, se devolverá el componente Magenta, y así, hasta* `4`*. El valor devuelto será un double de* `0.0` *hasta* `1.0`.

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

Leer CMYK

Igual que el anterior, pero los componentes que se devuelven son int desde `0` *hasta* `65535`.

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