libXbgi Quick Reference

Guido Gonzato, PhD

August 26, 2014

1 Introduction

libXbgi is an X11-Xlib implementation of the old Borland Graphics Interface (BGI), which was part of Turbo/Borland C compilers for DOS. libXbgi is part of ptoc[1], written by Dr. Konstantin Knizhnik.

libXbgi is one of the easiest ways to do graphics programming in C under X11, on Linux and other Unix-like systems. This is a minimal program that opens a graphics window and draws 1000 random lines:

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

int main (void)
{
  int i, gd = DETECT, gm;
  initgraph (&gd, &gm, "");
  setbkcolor (BLACK);
  cleardevice ();
  outtextxy (0, 0, "Drawing 1000 lines...");
  for (i = 0; i < 1000; i++) {
    setcolor (1 + random (15));
    line ( random(getmaxx()), random(getmaxy()),
    random(getmaxx()), random(getmaxy()) );
  }
  getch ();
  closegraph ();
  return 0;
}
```

The program includes the header file `graphics.h`, which contains all necessary definitions. The call to **initgraph()** opens a graphics window; from now on, graphics functions may be called. **closegraph()** closes the window.

Within the graphics window, pixel coordinates range from (0, 0) (upper left corner) to (**getmaxx()**, **getmaxy()**) (lower right corner.)

Some graphic functions set the coordinates of the last drawing position, defined as CP (Current Position). At any given moment, a foreground color, background color, line style, line thickness, fill pattern, and fill color, are defined. A viewport (subwindow) may be defined. All of these parameters can be changed using appropriate functions.

[1] http://garret.ru/lang.html

2 Constants

Many constants are defined in `graphics.h`. The most important are the following:

```
/* COLOR DEFINITIONS */

#define BLACK             0
#define BLUE              1
#define GREEN             2
#define CYAN              3
#define RED               4
#define MAGENTA           5
#define BROWN             6
#define LIGHTGRAY         7
#define DARKGRAY          8
#define LIGHTBLUE         9
#define LIGHTGREEN        10
#define LIGHTCYAN         11
#define LIGHTRED          12
#define LIGHTMAGENTA      13
#define YELLOW            14
#define WHITE             15

#define MAXCOLORS         15
#define MAXRGBCOLORS      4096

/* FILL STYLES */

#define EMPTY_FILL        0
#define SOLID_FILL        1
#define LINE_FILL         2
#define LTSLASH_FILL      3
#define SLASH_FILL        4
#define BKSLASH_FILL      5
#define LTBKSLASH_FILL    6
#define HATCH_FILL        7
#define XHATCH_FILL       8
#define INTERLEAVE_FILL   9
#define WIDE_DOT_FILL     10
#define CLOSE_DOT_FILL    11
#define USER_FILL         12

/* LINE STYLES */

#define SOLID_LINE        0
#define DOTTED_LINE       1
#define CENTER_LINE       2
#define DASHED_LINE       3
#define USERBIT_LINE      4

/* LINE THICKNESS */

#define NORM_WIDTH        1
#define THICK_WIDTH       3

/* DOTTED LINE STYLES */
```

```
#define DOTTEDLINE_LENGTH  2
#define CENTRELINE_LENGTH  4
#define DASHEDLINE_LENGTH  2

/* FONTS */

#define DEFAULT_FONT       0
#define TRIPLEX_FONT       1
#define SMALL_FONT         2
#define SANSSERIF_FONT     3
#define GOTHIC_FONT        4

/* TEXT DIRECTION */

#define HORIZ_DIR          0
#define VERT_DIR           1

/* TEXT ALIGNMENT */

#define LEFT_TEXT          0
#define CENTER_TEXT        1
#define RIGHT_TEXT         2
#define BOTTOM_TEXT        0
#define TOP_TEXT           2

/* DRAWING MODE */

#define COPY_PUT           0
#define XOR_PUT            1
#define OR_PUT             2
#define AND_PUT            3
#define NOT_PUT            4

/* MOUSE EVENTS */

#define WM_LBUTTONDOWN  1       // left button
#define WM_MBUTTONDOWN  2       // middle button
#define WM_RBUTTONDOWN  3       // right button
#define WM_WHEELUP      4       // wheel up
#define WM_WHEELDOWN    5       // wheel down
```

3 Structures

Some of the BGI functions use the following structs:

```
struct arccoordstype {
        int x;
        int y;
        int xstart;
        int ystart;
        int xend;
        int yend;
};
```

```
struct date {
        int da_year;
        int da_day;
        int da_mon;
};

struct fillsettingstype {
        int pattern;
        int color;
};

struct linesettingstype {
        int linestyle;
        unsigned int upattern;
        int thickness;
};

struct palettetype {
        unsigned char size;
        signed char colors[MAXCOLORS + 1];
};

struct textsettingstype {
        int font;
        int direction;
        int charsize;
        int horiz;
        int vert;
};

struct viewporttype {
        int left;
        int top;
        int right;
        int bottom;
        int clip;
};

struct bgi_info {
        int colour_index;
        char *colour_name;
        unsigned long pixel_value;
};

struct rgb_colour {
        int colour_index;
        unsigned long pixel_value;
};
```

4 Standard Graphics Functions

The following are standard BGI functions, as implemented for example in Turbo C. They are all prototyped in `graphics.h`.

Unless otherwise specified, graphics routines draw shapes using the current drawing color,

i.e. as specified by **setcolor()**.

`void` **arc** (`int` *x*, `int` *y*, `int` *stangle*, `int` *endangle*, `int` *radius*);

Draws a circular arc centered at (*x*, *y*), with a radius given by *radius*, traveling from *stangle* to *endangle*. The angle for **arc()** is reckoned counterclockwise, with 0 degrees at 3 o' clock, 90 degrees at 12 o' clock, etc.

Note: The *linestyle* parameter does not affect arcs, circles, ellipses, or pieslices. Only the *thickness* parameter is used.

`void` **bar** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*);

Draws a filled-in rectangle (bar), filled using the current fill pattern and fillcolor. The bar is not outlined; to draw an outlined two-dimensional bar, use **bar3d()** with *depth* equal to 0.

`void` **bar3d** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*, `int` *depth*, `int` *topflag*);

Draws a three-dimensional, filled-in rectangle (bar), filled using the current fill pattern and fillcolor. The three-dimensional outline of the bar is drawn in the current line style and color. The bar's depth, in pixels, is given by *depth*. If *topflag* is nonzero, a top is put on.

`void` **circle** (`int` *x*, `int` *y*, `int` *radius*);

Draws a circle of the given *radius* at (*x*, *y*).

Note: The *linestyle* parameter does not affect arcs, circles, ellipses, or pieslices. Only the *thickness* parameter is used.

`void` **cleardevice** (`void`);

Clears the graphics screen, filling it with the current background color. The CP is moved to (0, 0).

`void` **clearviewport** (`void`);

Clears the viewport, filling it with the current background color. The drawing CP is moved to (0, 0), relative to the viewport.

`void` **closegraph** (`void`);

Closes the graphics system.

`void` **detectgraph** (`int` **graphdriver*, `int` **graphmode*);

Detects the graphics driver and graphics mode to use by checking the hardware. In libXbgi, the defaults are VGA and VGAHI (for compatibility with old programs).

`void` **drawpoly** (`int` *numpoints*, `int` **polypoints*);

Draws a polygon of *numpoints* vertices. *polypoints* is a pointer to a sequence of (2 * *numpoints*) integers; each pair gives the x and y coordinate of each vertex.

`void` **ellipse** (`int` x, `int` y, `int` *stangle*, `int` *endangle*, `int` *xradius*, `int` *yradius*);

Draws an ellipse centered at (x, y), with axes given by *xradius* and *yradius*, traveling from *stangle* to *endangle*.

`void` **fillellipse** (`int` x, `int` y, `int` *xradius*, `int` *yradius*);

Draws an ellipse centered at (x, y), with axes given by *xradius* and *yradius*, and fills it using the current fill color and fill pattern.

`void` **fillpoly** (`int` *numpoints*, `int` **polypoints*);

Draws a polygon of *numpoints* vertices and fills it using the current fill color and fill pattern.

`void` **floodfill** (`int` x, `int` y, `int` *border*);

Fills an enclosed area, containing the x and y points and bounded by the *border* color. The area is filled using the current fill color.

Note: **floodfill** does not use the current fill pattern.

`void` **getarccoords** (`struct arccoordstype` **arccoords*);

Gets the coordinates of the last call to **arc()**, filling the *arccoords* structure.

`void` **getaspectratio** (`int` **xasp*, `int` **yasp*);

Retrieves the current graphics mode's aspect ratio. In libXbgi, *xasp* and *yasp* are both 10000 (i.e. the pixels are square).

`int` **getbkcolor** (`void`);

Returns the current background color.

`int` **getcolor** (`void`);

Returns the current drawing (foreground) color.

`struct palettetype` ***getdefaultpalette** (`void`);

Returns the palette definition structure.

`char *`**getdrivername** (`void`);

Returns a pointer to a string containing the name of the current graphics driver.

`void` **getfillpattern** (`char *` *pattern*);

Copies the user-defined fill pattern, as set by setfillpattern, into the 8-byte area pointed to by pattern.

`void` **getfillsettings** (`struct fillsettingstype` **fillinfo*);

Fills the `fillsettingstype` structure pointed to by *fillinfo* with information about the current fill pattern and fill color.

`int` **getgraphmode** (`void`);

Returns the current graphics mode.

`void` **getimage** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*, `void` **bitmap*);

Copies a bit image of the specified region into the memory pointed by *bitmap*.

`void` **getlinesettings** (`struct linesettingstype` **lineinfo*);

Fills the `linesettingstype` structure pointed by *lineinfo* with information about the current line style, pattern, and thickness.

`int` **getmaxcolor** (`void`);

Returns the maximum color value available. Using the standard BGI palette, it's `MAXCOLORS`; using the RGB palette, it's `MAXRGBCOLORS`.

`int` **getmaxmode** (`void`);

Returns the maximum mode number for the current driver.

`int` **getmaxx** (`void`);

Returns the maximum x screen coordinate.

`int` **getmaxy** (`void`);

Returns the maximum y screen coordinate.

`char* getmodename (int mode_number);`

Returns a pointer to a string containing the name of the specified graphics mode.

`void getmoderange (int graphdriver, int *lomode, int *himode);`

Gets the range of valid graphics modes. The *graphdriver* parameter is ignored.

`void getpalette (struct palettetype *palette);`

Fills the `palettetype` structure pointed by *palette* with information about the current palette's size and colors.

`int getpalettesize (void);`

Returns the size of the palette (`MAXCOLORS` + 1 or `MAXRGBCOLORS` + 1).

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

Returns the color of the pixel located at (x, y).

`void gettextsettings (struct textsettingstype *texttypeinfo);`

Fills the `textsettingstype` structure pointed to by *texttypeinfo* with information about the current text font, direction, size, and justification.

`void getviewsettings (struct viewporttype *viewport);`

Fills the `viewporttype` structure pointed to by *viewport* with information about the current viewport.

`int getx (void);`

Returns the current viewport's x coordinate.

`int gety (void);`

Returns the current viewport's y coordinate.

`void graphdefaults (void);`

Resets all graphics settings to their defaults: sets the viewport to the entire screen, moves the CP to (0, 0), sets the default palette colors, the default drawing and background color, the default fill style and pattern, the default text font and justification.

`char*` **grapherrormsg** (`int` *errorcode*);

Returns a pointer to the error message string associated with *errorcode*, returned by **graphresult()**.

`int` **graphresult** (`void`);

Returns the error code for the last unsuccessful graphics operation and resets the error level to grOk.

`unsigned` **imagesize** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*);

Returns the size in bytes of the memory area required to store a bit image.

`void` **initgraph** (`int` **graphdriver*, `int` **graphmode*, `char` **pathtodriver*);

Initializes the graphics system. In libXbgi, use X11 as *graphdriver*, then choose a suitable graphics mode (listed in `graphics.h`) as *graphmode*. The *pathtodriver* argument is ignored.

`int` **installuserdriver** (`char` **name*, `int` (**detect*)(`void`));

Unimplemented; not used by libXbgi.

`int` **installuserfont** (`char` **name*);

Unimplemented; not used by libXbgi.

`void` **line** (`int` *x1*, `int` *y1*, `int` *x2*, `int` *y2*);

Draws a line between two specified points; the CP is not updated.

`void` **linerel** (`int` *dx*, `int` *dy*);

Draws a line from the CP to a point that is (*dx*,*dy*) pixels from the CP. The CP is then advanced by (*dx*, *dy*).

`void` **lineto** (`int` *x*, `int` *y*);

Draws a line from the CP to (*x*, *y*), then moves the CP to (*dx*, *dy*).

`void` **moverel** (`int` *dx*, `int` *dy*);

Moves the CP by (*dx*, *dy*) pixels.

`void` **moveto** (`int` *x*, `int` *y*);

Moves the CP to the position (*x*, *y*), relative to the viewport.

`void` **outtext** (`char *`*textstring*);

Outputs *textstring* at the CP.

`void` **outtextxy** (`int` *x*, `int` *y*, `char *`*textstring*);

Outputs *textstring* at (*x*, *y*).

`void` **pieslice** (`int` *x*, `int` *y*, `int` *stangle*, `int` *endangle*, `int` *radius*);

Draws and fills a pie slice centered at (*x*, *y*), with a radius given by *radius*, traveling from *stangle* to *endangle*.

`void` **putimage** (`int` *left*, `int` *top*, `void` **bitmap*, `int` *op*);

Puts the bit image pointed to by *bitmap* onto the screen, with the upper left corner of the image placed at (*left*, *top*). *op* specifies the drawing mode (COPY_PUT, etc).

`void` **putpixel** (`int` *x*, `int` *y*, `int` *color*);

Plots a point at (*x*,*y*) in the color defined by *color*.

`void` **rectangle** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*);

Draws a rectangle delimited by (*left*,*top*) and (*right*,*bottom*).

`int` **registerbgidriver** (`void` (**driver*)(`void`));

Unimplemented; not used by libXbgi.

`int` **registerbgifont** (`void` (**font*)(`void`));

Unimplemented; not used by libXbgi.

`void` **restorecrtmode** (`void`);

In libXbgi, this function just clears the graphics window.

`void` **sector** (`int` *x*, `int` *y*, `int` *stangle*, `int` *endangle*, `int` *xradius*, `int` *yradius*);

Draws and fills an elliptical pie slice centered at (*x*, *y*), horizontal and vertical radii given by *xradius* and *yradius*, traveling from *stangle* to *endangle*.

`void` **setactivepage** (`int` *page*);

Makes *page* the active page for all subsequent graphics output.

`void` **setallpalette** (`struct palettetype *`*palette*);

Sets the current palette to the values given in *palette*.

`void` **setaspectratio** (`int` *xasp*, `int` *yasp*);

Changes the default aspect ratio of the graphics. In libXbgi, this function is not necessary as the pixels are square.

`void` **setbkcolor** (`int` *color*);

Sets the current background color using the default palette.

`void` **setcolor** (`int` *color*);

Sets the current drawing color using the default palette.

`void` **setfillpattern** (`char *`*upattern*, `int` *color*);

Sets a user-defined fill pattern. *upattern* is a pointer to a sequence of 8 bytes; each byte corresponds to 8 pixels in the pattern; each bit set to 1 is plotted as a pixel.

`void` **setfillstyle** (`int` *pattern*, `int` *color*);

Sets the fill pattern and fill color. *pattern* can be `EMPTY_FILL`, `SOLID_FILL`, `LINE_FILL`, `LTSLASH_FILL`, `SLASH_FILL`, `BKSLASH_FILL`, `LTBKSLASH_FILL`, `HATCH_FILL`, `XHATCH_FILL`, `INTERLEAVE_FILL`, `WIDE_DOT_FILL`, `CLOSE_DOT_FILL`, or `USER_FILL`.

To provide a user-defined fill pattern, please use **setfillpattern()**.

unsigned **setgraphbufsize** (`unsigned` *bufsize*);

Unimplemented; not used by libXbgi.

`void` **setgraphmode** (`int` *mode*);

Sets the graphics to *mode* and clears the window.

`void` **setlinestyle** (`int` *linestyle*, `unsigned` *upattern*, `int` *thickness*);

Sets the line width and style for all lines drawn by **line()**, **lineto()**, **rectangle()**, **drawpoly()**, etc. The line style can be `SOLID_LINE`, `DOTTED_LINE`, `CENTER_LINE`, `DASHED_LINE`, and `USERBIT_LINE`; the latter is user defined. The line thickness in pixels can be `NORM_WIDTH`, `THICK_WIDTH`, or a value expressed in pixels.

Arcs, circles, ellipses, and pieslices are not affected by *linestyle*, but are affected by *thickness*.

upattern is a pointer to a sequence of 2 bytes; each byte corresponds to 8 pixels in the pattern; each bit set to 1 is plotted as a pixel.

`void` **setpalette** (`int` *colornum*, `int` *color*);

Changes the standard palette *colornum* to *color*.

`void` **settextjustify** (`int` *horiz*, `int` *vert*);

Sets text justification. Text output will be justified around the CP horizontally and vertically; settings are `LEFT_TEXT`, `CENTER_TEXT`, `RIGHT_TEXT`, `BOTTOM_TEXT`, and `TOP_TEXT`.

`void` **settextstyle** (`int` *font*, `int` *direction*, `int` *charsize*);

Sets the text font (`DEFAULT_FONT`, `TRIPLEX_FONT`, `SMALL_FONT`, `SANSSERIF_FONT`, `GOTHIC_FONT`), the direction in which text is displayed (`HORIZ_DIR`, `VERT_DIR`), and the size of the characters. If *charsize* is an integer number, the text will be scaled by that number; if it is 0, the text will be scaled by **setusercharsize()**.

Note: as of version 363, this function has a bug; *charsize* has no effect.

`void` **setusercharsize** (`int` *multx*, `int` *divx*, `int` *multy*, `int` *divy*);

Lets the user change the character width and height. If a previous call to **settextstyle()** set *charsize* to 0, the default width is scaled by *multx*/*divx*, and the default height is scaled by *multy*/*divy*.

`void` **setviewport** (`int` *left*, `int` *top*, `int` *right*, `int` *bottom*, `int` *clip*);

Sets the current viewport for graphics output. If *clip* is nonzero, all drawings will be clipped (truncated) to the current viewport.

`void` **setvisualpage** (`int` *page*);

Sets the visual graphics page number.

`void` **setwritemode** (`int` *mode*);

Sets the writing mode for line drawing. *mode* can be `COPY_PUT`, `XOR_PUT`, `OR_PUT`, `AND_PUT`, and `NOT_PUT`.

`int` **textheight** (`char *`*textstring*);

Returns the height in pixels of a string.

`int` **textwidth** (`char *`*textstring*);

Returns the height in pixels of a string.

5 Non-Graphics Functions and Macros

`void` **delay** (`int` *millisec*);

Waits for *millisec* milliseconds.

`int` **getch** (`void`);

Waits for a key and returns its ASCII code.

`int` **kbhit** (`void`);

Returns 1 when a key is pressed, except Ctrl, Shift, Alt, and so on.

`int` **random** (`int` *range*) (macro)

Returns a random number between 0 and *range*.

6 libXbgi Additions

`int` **COLOR** (`int` *r*, `int` *g*, `int` *b*);

Can be used as an argument for **setcolor** and **setbkcolor** to set an RBG color.

`int` RED_VALUE(`int` *color*) (macro)

Returns the red component of an RGB color.

`int` GREEN_VALUE(`int` *color*) (macro)

Returns the green component of an RGB color.

`int` BLUE_VALUE(`int` *color*) (macro)

Returns the blue component of an RGB color.

`int` IS_BGI_COLOR(`int` *color*);

Returns 1 if the current color is a standard BGI color (not RGB). The argument is actually redundant.

`int` IS_RGB_COLOR(`int` *color*);

Returns 1 if the current color is in RGB mode. The argument is actually redundant.

`unsigned long` **converttorgb** (`int` *color*);

Converts a BGI color to its RGB equivalent.

`int` **getevent** (`void`);

Waits for a keypress or mouse click, and returns the code of the mouse button or key that was pressed.

`int` **getmaxheight** (`void`);

Returns the screen (root window) height.

`int` **getmaxwidth** (`void`);

Returns the screen (root window) width.

`void` **getmouseclick** (`int` *kind*, `int` *x, `int` *y);

Sets the x,y coordinates of the last *kind* button click expected by **ismouseclick**.

`int` **initwindow** (`int` *width*, `int` *height*);

Opens a *width*×*height* graphics window.

`int` **ismouseclick** (`int` *kind*);

Returns 1 if the *kind* mouse button was clicked.

`int` **mouseclick** (`void`);

Returns the code of the mouse button that was clicked, or 0 if none was clicked.

`int` **mousex** (`void`);

Returns the X coordinate of the last mouse click.

`int` **mousey** (`void`);

Returns the Y coordinate of the last mouse click.

`void` **_putpixel** (`int` x, `int` y);

Plots a point at (x,y) in the current drawing color. This function is faster than **putpixel**.

`void` **setbkrgbcolor** (`int` n);

Sets the current background color using using the n-th color index in the RGB palette.

`void` **setrgbcolor** (`int` n);

Sets the current drawing color using the n-th color index in the RGB palette.

`void` **setrgbpalette** (`int` n, `int` r, `int` g, `int` b);

Sets the n-th entry in the RGB palette specifying the r, g, and b components.
Using **setrgbpalette** and **setrgbcolor** is faster than setting colors with **setcolor** with a **COLOR** argument.

`int` **xkbhit** (`void`);

Returns 1 when any key is pressed, including Ctrl, Shift, Alt, and so on.