SDL_bgi 2.0.2 Quick Reference

Guido Gonzato, PhD

September 28, 2015

1 Introduction

SDL_bgi is an SDL2-based implementation of Borland Graphics Interface (BGI), a graphics library that was part of Turbo/Borland C compilers for DOS. It represented the *de facto* standard for computer graphics in the late eighties–early nineties, especially in education.

SDL_bgi is one of the easiest ways to do graphics programming in C. It is much simpler (but obviously, less powerful) than SDL, OpenGL and the like. Teachers may find SDL_bgi a useful tool for introductory computer graphics courses.

For example, this is a minimal program that opens a window and draws 1000 random lines:

```
#include <SDL2/SDL_bgi.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 `SDL_bgi.h`, which contains all necessary definitions. The call to **initgraph()** opens a window; from now on, graphics functions may be called. **closegraph()** closes the window.

Within the window, pixel coordinates range from (0, 0), the upper left corner, to (**getmaxx()**, **getmaxy()**), the 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, background and fill colour, line style, line thickness, and fill pattern, are defined. A viewport (subwindow) may also be defined, with or without clipping. All of these parameters can be changed using appropriate functions.

2 Constants

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

```
// everything gets drawn here

extern SDL_Window   *bgi_window;
extern SDL_Renderer *bgi_renderer;
extern SDL_Texture  *bgi_texture;

// available visual pages

#define VPAGES 2

// BGI fonts

#define DEFAULT_FONT   0 // 8x8

#define HORIZ_DIR      0
#define VERT_DIR       1

#define USER_CHAR_SIZE 0

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

// BGI colours

#define MAXCOLORS      15
#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

// line style, thickness, and drawing mode

#define NORM_WIDTH      1
#define THICK_WIDTH     3

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

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

// fill styles

#define EMPTY_FILL      0
#define SOLID_FILL      1

// mouse buttons

#define WM_LBUTTONDOWN  SDL_BUTTON_LEFT
#define WM_MBUTTONDOWN  SDL_BUTTON_MIDDLE
#define WM_RBUTTONDOWN  SDL_BUTTON_RIGHT
#define WM_WHEELUP      4 // SDL_MOUSEWHEEL no longer exists
#define WM_WHEELDOWN    5 // ditto
#define WM_MOUSEMOVE    SDL_MOUSEMOTION

#define PALETTE_SIZE    4096
#define KEY_HOME        SDLK_HOME
#define KEY_LEFT        SDLK_LEFT
#define KEY_UP          SDLK_UP
#define KEY_RIGHT       SDLK_RIGHT
#define KEY_DOWN        SDLK_DOWN
#define KEY_PGUP        SDLK_PAGEUP
#define KEY_PGDN        SDLK_PAGEDOWN
#define KEY_END         SDLK_END
#define KEY_INSERT      SDLK_INSERT
#define KEY_DELETE      SDLK_DELETE
#define KEY_F1          SDLK_F1
#define KEY_F2          SDLK_F2
#define KEY_F3          SDLK_F3
#define KEY_F4          SDLK_F4
#define KEY_F5          SDLK_F5
#define KEY_F6          SDLK_F6
#define KEY_F7          SDLK_F7
#define KEY_F8          SDLK_F8
#define KEY_F9          SDLK_F9
#define KEY_F10         SDLK_F10
```

```
#define KEY_F11          SDLK_F11
#define KEY_F12          SDLK_F12
#define KEY_LEFT_CTRL    SDLK_LCTRL
#define KEY_RIGHT_CTRL   SDLK_RCTRL
#define KEY_LEFT_SHIFT   SDLK_LSHIFT
#define KEY_RIGHT_SHIFT  SDLK_RSHIFT
#define KEY_LEFT_ALT     SDLK_LALT
#define KEY_LEFT_WIN     SDLK_LSUPER
#define KEY_RIGHT_WIN    SDLK_RSUPER
#define KEY_ALT_GR       SDLK_MODE
#define KEY_TAB          SDLK_TAB
#define KEY_BS           SDLK_BACKSPACE
#define KEY_RET          SDLK_RETURN
#define KEY_PAUSE        SDLK_PAUSE
#define KEY_SCR_LOCK     SDLK_SCROLLOCK
#define KEY_ESC          SDLK_ESCAPE
```

3 Structs

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 BGI Graphics Functions

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

Unless otherwise specified, graphics routines draw shapes using the current drawing colour, 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 measured 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), using the current fill colour and fill pattern. 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), using the current fill colour and fill pattern. 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 SDL_bgi, the default is 800x600 (SVGA).

`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 elliptical arc 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.

Note: as of the current release, only convex polygons are correctly filled, and only using SOLID_FILL.

`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: as of the current release, **floodfill()** only uses SOLID_FILL.

`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 SDL_bgi, *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 (`MAXCOLORS`). If RGB colors are being used, it returns `PALETTE_SIZE`.

`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 SDL_bgi, you can use SDL as *graphdriver*, then choose a suitable graphics mode (listed in `graphics.h`) as *graphmode*. The *pathtodriver* argument is ignored; you can also use NULL for **graphdriver* and **graphmode*.

After **initgraph()**, all graphics commands are immediately displayed, as in the original BGI. This makes drawing very slow; you may want to use **initwindow()** instead.

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

Initializes the graphics system, opening a *width*×*height* window. If either *width* or *height* is 0, then SDL_FULLSCREEN will be used.
The user is in charge of updating the screen using **refresh()**.

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

Unimplemented; not used by SDL_bgi.

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

Unimplemented; not used by SDL_bgi.

`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*. The pie slice is filled using the current fill colour.
Note: as of the current release, the pie slice is filled only using SOLID_FILL.

`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*).

`void` **refresh** (`void`);

Updates the screen.

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

Unimplemented; not used by SDL_bgi.

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

Unimplemented; not used by SDL_bgi.

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

Hides 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*.

Note: as of the current release, the sector is filled only using SOLID_FILL.

`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 SDL_bgi, this function is not necessary since 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` *upattern*, `int` *color*);

Sets the fill pattern and fill color. *upattern* is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern.

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

Unimplemented; not used by SDL_bgi.

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

Shows the window that was hidden by **restorecrtmode()**. The *mode* parameter is ignored

`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`, or `USERBIT_LINE`; in the latter case, the user provides a 16-bit number (*upattern*) whose bits set to 1 will be plotted as pixels.

The line thickness can be set with `NORM_WIDTH` or `THICK_WIDTH`.

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

`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 (only `DEFAULT_FONT` is actually available), 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()**.

`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. In "slow mode", a screen refresh is performed.

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

Waits for a key and returns its ASCII code. In "slow mode", a screen refresh is performed.

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

Returns 1 when a key is pressed, including Ctrl, Shift, Alt, and so on. In "slow mode", a screen refresh is performed.

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

Returns a random number between 0 and *range - 1*.

6 SDL_bgi 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*)

Returns the red component of an RGB color.

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

Returns the green component of an RGB color.

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

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.

`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*) using the current drawing color. This function is faster than **putpixel()**, since it is not immediately displayed.

`void` **readimagefile** (`char *` *filename*, `int` *x1*, `int` *y1*, `int` *x2*, `int` *y2*);

Reads a `.bmp` file and displays it immediately at (*x1*, *y1*). If (*x2*, *y2*) are not 0, the bitmap is stretched to fit the rectangle *x1*,*y1*—*x2*,*y2*; otherwise, the bitmap is clipped as necessary.

`void` **refresh** (`void`);

Updates the screen contents, i.e. displays all graphics.

`void` **sdlfastmode** (`void`);

Triggers "fast mode", i.e. **refresh()** is needed to display graphics.

`void` **sdlslowmode** (`void`);

Triggers "slow mode", i.e. **refresh()** is not needed to display graphics.

`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.

`void` **writeimagefile** (`char *` *filename*, `int` *left*, `int` *top*, `int` *right*, `int` *bottom*);

Writes a `.bmp` file from the screen rectangle defined by *left*,*top*—*right*,*bottom*.
