

SDL_bgi 2.4.4 Quick Reference

Guido Gonzato, PhD

May 4, 2021



Contents

1	Introduction to SDL_bgi	3	<code>getarccoords()</code>	12
1.1	Constants	4	<code>getaspectratio()</code>	12
1.2	Environment Variables	9	<code>getbkcolor()</code>	12
1.3	Colours	9	<code>getcolor()</code>	12
1.4	Fonts	9	<code>getdefaultpalette()</code>	12
1.5	Note for SDL2 Programmers	9	<code>getdrivername()</code>	12
			<code>getfillpattern()</code>	12
			<code>getfillsettings()</code>	12
			<code>getgraphmode()</code>	12
2	Standard BGI Graphics Functions	10	<code>getimage()</code>	13
	<code>arc()</code>	10	<code>getlinesettings()</code>	13
	<code>bar()</code>	10	<code>getmaxcolor()</code>	13
	<code>bar3d()</code>	10	<code>getmaxmode()</code>	13
	<code>circle()</code>	10	<code>getmaxx()</code>	13
	<code>cleardevice()</code>	10	<code>getmaxy()</code>	13
	<code>clearviewport()</code>	11	<code>getmodename()</code>	13
	<code>closegraph()</code>	11	<code>getmoderange()</code>	13
	<code>detectgraph()</code>	11	<code>getpalette()</code>	13
	<code>drawpoly()</code>	11	<code>getpalettesize()</code>	13
	<code>ellipse()</code>	11	<code>getpixel()</code>	14
	<code>fillellipse()</code>	11	<code>gettextsettings()</code>	14
	<code>fillpoly()</code>	11	<code>getviewsettings()</code>	14
	<code>floodfill()</code>	11	<code>getvisualpage()</code>	14
	<code>getactivepage()</code>	11		

getx()	14	setwritemode()	19
gety()	14	textheight()	19
graphdefaults()	14	textwidth()	19
grapherrormsg()	14		
graphresult()	14	3 Non-Graphics Functions and	
imagesize()	15	Macros	19
initgraph()	15	getch()	19
installuserdriver()	15	kbhit()	20
installuserfont()	15	random()	20
line()	15		
linereel()	16	4 SDL_bgi Additions	20
lineto()	16	ALPHA_VALUE()	20
moverel()	16	BLUE_VALUE()	20
moveto()	16	COLOR()	20
outtext()	16	COLOR32()	20
outtextxy()	16	colorRGB()	20
pieslice()	16	GREEN_VALUE()	21
putimage()	16	IS_BGI_COLOR()	21
putpixel()	16	IS_RGB_COLOR()	21
rectangle()	16	RED_VALUE()	21
registerbgidriver()	17	_putpixel()	21
registerbgifont()	17	closewindow()	21
restorecrtmode()	17	event()	21
sector()	17	eventtype()	21
setactivepage()	17	getbuffer()	22
setallpalette()	17	getcurrentwindow()	22
setaspectratio()	17	getevent()	22
setbkcolor()	17	getleftclick()	22
setcolor()	17	getlinebuffer()	22
setfillpattern()	17	getmaxheight()	22
setfillstyle()	18	getmaxwidth()	22
setgraphbufsize()	18	getmiddleclick()	22
setgraphmode()	18	getmouseclick()	22
setlinestyle()	18	getrightclick()	23
setpalette()	18	getscreensize()	23
settextjustify()	18	initpalette()	23
settextstyle()	18	initwindow()	23
setusercharsize()	19	ismouseclick()	23
setviewport()	19	mouseclick()	23
setvisualpage()	19	mousex()	23
		mousey()	23

<code>putbuffer()</code>	23	<code>setblendmode()</code>	25
<code>putlinebuffer()</code>	24	<code>setcurrentwindow()</code>	25
<code>readimagefile()</code>	24	<code>setrgbcolor()</code>	25
<code>refresh()</code>	24	<code>setrgbpalette()</code>	25
<code>resetwinoptions()</code>	24	<code>setwinoptions()</code>	25
<code>resizepalette()</code>	24	<code>setwintitle()</code>	25
<code>sdlbgiauto()</code>	24	<code>showerrorbox()</code>	26
<code>sdlbgifast()</code>	24	<code>showinfobox()</code>	26
<code>sdlbgislow()</code>	24	<code>swapbuffers()</code>	26
<code>setalpha()</code>	25	<code>writeimagefile()</code>	26
<code>setbkrgbcolor()</code>	25	<code>xkbhit()</code>	26

1 Introduction to SDL_bgi

SDL_bgi is a multiplatform, fast, SDL2-based implementation of BGI: the Borland Graphics Interface also known as GRAPHICS.H. BGI was a graphics library provided by Borland Turbo C / Borland C++ compilers for DOS, and it was very popular in the late eighties–early nineties. It became the *de facto* standard for computer graphics, especially in education. For more information, please see https://en.wikipedia.org/wiki/Borland_Graphics_Interface.

However, SDL_bgi is not meant to be just a replica of BGI; it aims to be a functionally equivalent superset. It provides many extensions for ARGB colours, multiple windows, bitmap and vector fonts, and mouse support. SDL_bgi implements nearly all extensions provided by another popular BGI implementation, WinBGIm, and adds several more. Please see <http://www.codecutter.net/tools/winbgim> for more information on WinBGIm.

SDL_bgi is one of the easiest libraries to do graphics programming in C. It is much simpler to use than SDL2, OpenGL and the like; obviously, it's less complete. Teachers may find SDL_bgi a useful tool for introductory computer graphics courses.

This is a minimal program that opens a window and draws 1000 random lines:

```
#include <graphics.h>

int main (int argc, char *argv[])
{
    int i, gd = DETECT, gm;
    initgraph (&gd, &gm, ""); // default: 800 x 600
    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 in turn includes `SDL_bgi.h` that 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.

1.1 Constants

`SDL_bgi.h` contains many definitions the programmer should be aware of. The most important are the following:

```

#ifndef _SDL_BGI_H
#define _SDL_BGI_H

#ifndef __GRAPHICS_H
#define __GRAPHICS_H

// SDL2 stuff
#include <SDL2/SDL.h>
#include <SDL2/SDL_keycode.h>
#include <SDL2/SDL_mouse.h>

#include <stdio.h> // for fprintf()
#include <stdlib.h> // for exit(), calloc()
#include <math.h> // for sin(), cos()
#include <string.h> // for strlen(), memcpy()

#define SDL_BGI_VERSION 2.4.4

enum { NOPE, YEAH } ;
#define BGI_WINTITLE_LEN 512 // more than enough

// number of concurrent windows that can be created

#define NUM_BGI_WIN 16

// everything gets drawn here

extern SDL_Window *bgi_window;
extern SDL_Renderer *bgi_renderer;
extern SDL_Texture *bgi_texture;
extern Uint32 PALETTE_SIZE;

// available visual pages

```

```

#define VPAGES 4

// BGI fonts

enum {
    DEFAULT_FONT,      // 8x8 bitmap
    TRIPLEX_FONT,     // trip.h
    SMALL_FONT,       // litt.h
    SANS_SERIF_FONT,  // sans.h
    GOTHIC_FONT,      // goth.h
    SCRIPT_FONT,      // scri.h
    SIMPLEX_FONT,     // simp.h
    TRIPLEX_SCR_FONT, // tscr.h
    COMPLEX_FONT,     // lcom.h
    EUROPEAN_FONT,    // euro.h
    BOLD_FONT,        // bold.h
    LAST_SPEC_FONT
};

enum { HORIZ_DIR, VERT_DIR };

#define USER_CHAR_SIZE 0

enum {
    LEFT_TEXT, CENTER_TEXT, RIGHT_TEXT,
    BOTTOM_TEXT = 0, TOP_TEXT = 2
};

// BGI colours, including CGA and EGA palettes

enum {
    BLACK      = 0,          EGA_BLACK      = 0,
    BLUE       = 1,          EGA_BLUE       = 1,
    GREEN      = 2, CGA_GREEN = 2, EGA_GREEN      = 2,
    CYAN       = 3, CGA_CYAN  = 3, EGA_CYAN       = 3,
    RED        = 4, CGA_RED    = 4, EGA_RED         = 4,
    MAGENTA    = 5, CGA_MAGENTA = 5, EGA_MAGENTA     = 5,
    BROWN      = 6, CGA_BROWN  = 6, EGA_BROWN        = 6,
    LIGHTGRAY  = 7, CGA_LIGHTGRAY = 7, EGA_LIGHTGRAY   = 7,
    DARKGRAY   = 8,          EGA_DARKGRAY   = 8,
    LIGHTBLUE  = 9,          EGA_LIGHTBLUE  = 9,
    LIGHTGREEN = 10, CGA_LIGHTGREEN = 10, EGA_LIGHTGREEN = 10,
    LIGHTCYAN  = 11, CGA_LIGHTCYAN = 11, EGA_LIGHTCYAN  = 11,
    LIGHTRED   = 12, CGA_LIGHTRED  = 12, EGA_LIGHTRED   = 12,
    LIGHTMAGENTA = 13, CGA_LIGHTMAGENTA = 13, EGA_LIGHTMAGENTA = 13,
    YELLOW     = 14, CGA_YELLOW  = 14, EGA_YELLOW     = 14,
    WHITE      = 15, CGA_WHITE   = 15, EGA_WHITE      = 15,
    MAXCOLORS  = 15
};

// ARGB colours, set by COLOR ()

enum {
    ARGB_FG_COL = 16,
    ARGB_BG_COL = 17,
    ARGB_FILL_COL = 18,
    ARGB_TMP_COL = 19,

```

```

    TMP_COLORS      = 4
};

// line style, thickness, and drawing mode

enum { NORM_WIDTH = 1, THICK_WIDTH = 3 };

enum { SOLID_LINE, DOTTED_LINE, CENTER_LINE, DASHED_LINE, USERBIT_LINE };

enum { COPY_PUT, XOR_PUT, OR_PUT, AND_PUT, NOT_PUT };

// fill styles

enum {
    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, USER_FILL
};

// mouse buttons

#define WM_LBUTTONDOWN  SDL_BUTTON_LEFT
#define WM_MBUTTONDOWN  SDL_BUTTON_MIDDLE
#define WM_RBUTTONDOWN  SDL_BUTTON_RIGHT
#define WM_WHEEL        SDL_MOUSEWHEEL
#define WM_WHEELUP      SDL_USEREVENT
#define WM_WHEELDOWN    SDL_USEREVENT + 1
#define WM_MOUSEMOVE    SDL_MOUSEMOTION

// keys
#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_CAPSLOCK    SDLK_CAPSLOCK
#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_RIGHT_ALT    SDLK_RALT
#define KEY_ALT_GR       SDLK_MODE
#define KEY_LGUI         SDLK_LGUI
#define KEY_RGUI         SDLK_RGUI
#define KEY_MENU         SDLK_MENU
#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_SCROLLLOCK
#define KEY_ESC          SDLK_ESCAPE

#define QUIT             SDL_QUIT

// graphics modes. Expanded from the original GRAPHICS.H

enum {
    DETECT = -1,
    SDL = 0,
    // all modes @ 320x200
    SDL_320x200 = 1, SDL_CGALO = 1, CGA = 1, CGACO = 1, CGAC1 = 1,
    CGAC2 = 1, CGAC3 = 1, MCGACO = 1, MCGAC1 = 1, MCGAC2 = 1,
    MCGAC3 = 1, ATT400C0 = 1, ATT400C1 = 1, ATT400C2 = 1, ATT400C3 = 1,
    // all modes @ 640x200
    SDL_640x200 = 2, SDL_CGAHI = 2, CGAHI = 2, MCGAMED = 2,
    EGALO = 2, EGA64LO = 2,
    // all modes @ 640x350
    SDL_640x350 = 3, SDL_EGA = 3, EGA = 3, EGAHI = 3,
    EGA64HI = 3, EGAMONOHI = 3,
    // all modes @ 640x480
    SDL_640x480 = 4, SDL_VGA = 4, VGA = 4, MCGAHI = 4, VGAHI = 4,
    IBM8514LO = 4,
    // all modes @ 720x348
    SDL_720x348 = 5, SDL_HERC = 5,
    // all modes @ 720x350
    SDL_720x350 = 6, SDL_PC3270 = 6, HERCMONOHI = 6,
    // all modes @ 800x600
    SDL_800x600 = 7, SDL_SVGALO = 7, SVGA = 7,
    // all modes @ 1024x768
    SDL_1024x768 = 8, SDL_SVGAMED1 = 8,
    // all modes @ 1152x900
    SDL_1152x900 = 9, SDL_SVGAMED2 = 9,
    // all modes @ 1280x1024
    SDL_1280x1024 = 10, SDL_SVGAHI = 10,
    // all modes @ 1366x768
    SDL_1366x768 = 11, SDL_WXGA = 11,
    // other
    SDL_USER = 12, SDL_FULLSCREEN = 13
};

// error messages
enum graphics_errors {
    grOk = 0,
    grNoInitGraph = -1,
    grNotDetected = -2,
    grFileNotFound = -3,
    grInvalidDriver = -4,
    grNoLoadMem = -5,
};

```

```

    grNoScanMem          = -6,
    grNoFloodMem         = -7,
    grFontNotFound       = -8,
    grNoFontMem          = -9,
    grInvalidMode        = -10,
    grError               = -11,
    grIOerror            = -12,
    grInvalidFont        = -13,
    grInvalidFontNum     = -14,
    grInvalidVersion     = -18
};

// 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 {
    Uint32 size; // unsigned char in Turbo C / Borland C++
    Uint32 colors[MAXCOLORS + 1]; // signed char in Turbo C / Borland C++
};

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

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

```

```
    int clip;
};
```

1.2 Environment Variables

`SDL_BGI_RES`: when set to `VGA`, default resolution will be 640×480 instead of 800×600 . Please see `initgraph()` (page 15) for details.

`SDL_BGI_RATE`: when set to `auto`, automatic screen refresh will be performed. Please see `initgraph()` (page 15) for details.

`SDL_BGI_PALETTE`: when set to `BGI`, the first 16 colours will use the same RGB values as Turbo C 2.01. Please see `initpalette()` (page 23) for details.

1.3 Colours

The default BGI palette includes 16 named colours (`BLACK... WHITE`); standard functions like `setbkcolor()` or `setcolor()` use this palette.

An extended ARGB palette of `PALETTE_SIZE` additional colours can be created and accessed using functions like `setbkrgbcolor()` or `setrbgcolor()` described below. The palette can be resized using `resizepalette()`.

Constants `ARGB_FG_COL`, `ARGB_BG_COL`, `ARGB_FILL_COL`, and `ARGB_TMP_COL` denote the foreground, background, fill, and temporary ARGB colours that can be set with the `COLOR()` function described below.

1.4 Fonts

`SDL_bgi` provides an 8×8 bitmap font and vector fonts decoded from original `.CHR` files; loading `.CHR` fonts from disk is also possible. Please see `settextstyle()` (page 18) for details.

`.CHR` font supported was added by Marco Diego Aurélio Mesquita.

1.5 Note for SDL2 Programmers

The following variables are declared in `SDL_bgi.h`, and are accessible to the programmer:

```
SDL_Window    *bgi_window;
SDL_Renderer  *bgi_renderer;
SDL_Texture   *bgi_texture;
extern Uint32 PALETTE_SIZE;
```

and can be used by native SDL2 functions. That is, you can use BGI and native SDL2 functions together, as in the following code snippet:

```

SDL_Surface *bitmap;
SDL_Texture *texture;
...
bitmap = SDL_LoadBMP ("picture.bmp");
texture = SDL_CreateTextureFromSurface (bgi_renderer, bitmap);
SDL_RenderCopy (bgi_renderer, texture, NULL, NULL);
SDL_RenderPresent (bgi_renderer);
...

```

2 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 colour. 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 colour. The CP is moved to (0, 0).

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

Clears the viewport, filling it with the current background colour. The 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 default graphics mode to use; `SDL` and `SDL_FULLSCREEN`, respectively.

```
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 colour and fill pattern.

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

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

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

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

```
int getactivepage (void);
```

Returns the active page number.

```
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. pixels are square).

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

Returns the current background colour in the default palette. If the background colour was set by `COLOR()`, `getbkcolor()` returns -1.

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

Returns the current drawing (foreground) colour in the default palette. If the foreground colour was set by `COLOR()`, `getcolor()` returns -1.

```
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 colour.

```
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 colour value available (*MAXCOLORS*) in the default palette. If ARGB colours are being used, it returns *PALETTE_SIZE*.

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

Returns the maximum mode number for the current driver. In *SDL_bgi*, the default is *SDL_FULLSCREEN*.

```
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);
```

Returns 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 colours.

```
int getpalettesize (void);
```

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

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

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

```
void gettextsettings (struct textsettingstype *textypeinfo);
```

Fills the `textsettingstype` structure pointed to by *textypeinfo* 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 getvisualpage (void);
```

Returns the visual page number.

```
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 default values: sets the viewport to the entire screen, moves the CP to $(0, 0)$, sets the default palette colours, the default drawing and background colour, 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. Typically, *graphdriver* is set to `DETECT`, and *graphmode* is not set; these values will set the default resolution (800×600) as `SVGA`. If the environment variable `SDL_BGI_RES` equals `VGA` or `vga`, then `VGA` resolution (640×480) will be forced.

You can also use `NULL` for **graphdriver* and **graphmode* to get the default resolution (800×600), or use `detectgraph()` (see above) to get fullscreen.

Multiple windows can be created, unless a fullscreen window is already present.

Using `initgraph()`, the default 16-colour palette uses the same ARGB values as the original palette in Turbo C. Using `initwindow()`, the default 16-colour palette uses different (possibly, better-looking) ARGB values.

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

Alternatively, automatic screen refresh can be performed according to the value of the `SDL_BGI_RATE` environment variable. If the variable is set to `auto`, screen refresh is automatically performed every *msec* milliseconds; this value is the current screen refresh rate, as given by `SDL_GetDisplayMode()`. If the variable is set to an integer value *msec*, automatic screen refresh will be performed every *msec* milliseconds.

Automatic screen refresh is much faster than the default behaviour; however, it is an experimental feature that may not work on some graphic cards.

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

Unimplemented; not used by `SDL_bgi`.

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

Loads and installs a `.CHR` font from disk. The function returns an integer to be used as first argument in `settextstyle()`.

Note: font metrics of loaded `.CHR` files are not pixel-perfect yet.

```
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.

```
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 pixel at (x, y) in the colour 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 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*.

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

Makes *page* the active page for all subsequent graphics output. In multi-window mode, `setactivepage()` only works for the first window.

```
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 colour in the default palette.

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

Sets the current drawing colour in the default palette. If *color* > MAXCOLORS, then set *color* % MAXCOLORS.

```
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 colour. *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*, which can also be specified using the `COLOR()` function.

```
void setttextjustify (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 setttextstyle (int font, int direction, int charsize);
```

Sets the text font (8×8 bitmap font `DEFAULT_FONT` and vector fonts `TRIPLEX_FONT`, `SMALL_FONT`, `SANS_SERIF_FONT`, `GOTHIC_FONT`, `SCRIPT_FONT`, `SIMPLEX_FONT`, `TRIPLEX_SCR_FONT`), the text direction (`HORIZ_DIR`, `VERT_DIR`), and the size of the characters.

charsize is a scaling factor for the text (max. 10). If *charsize* is 0, the text will either use the default size, or it will be scaled by the values set with `setusercharsize()`.

Experimental feature: if a `.CHR` font is available in the same directory as the running program, it will be loaded and used instead of its internal equivalent.

Note: pixel-perfect font metrics are not supported yet.

```
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. In “fast mode”, the screen is not cleared.

```
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 width in pixels of a string.

3 Non-Graphics Functions and Macros

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

Waits for *millisec* milliseconds. In “slow mode”, a screen refresh is performed. Events that occur during the delay are disregarded.

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

Waits for a key and returns its ASCII or key code (i.e. `KEY_*`). In “slow mode”, a screen refresh is performed. If an `SDL_QUIT` event occurs, `QUIT` is returned.

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

Returns 1 when a key is pressed, excluding special keys (Ctrl, Shift, etc.); in “slow mode”, a screen refresh is performed. If an `SDL_QUIT` event occurs, `QUIT` is returned.

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

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

4 SDL_bgi Additions

The following `SDL_bgi` extensions are mostly compatible with those made available by `WinBGIm`.

```
int ALPHA_VALUE (int color)
```

Returns the alpha (transparency) component of an ARGB colour in the ARGB palette.

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

Returns the blue component of an ARGB colour in the ARGB palette.

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

Can be used as an argument for `setcolor()`, `setbkcolor()`, `setfillpattern()`, and `setfillstyle()` to set a colour specifying its ARGB components. The colour is stored in `ARGB_TMP_COL`, and it is also copied in one of the following: `ARGB_FG_COL`, `ARGB_BG_COL`, `ARGB_FILL_COL`.

Functions `ALPHA_VALUE()`, `BLUE_VALUE()`, `GREEN_VALUE()`, and `RED_VALUE()` do not work on temporary colours.

```
int COLOR32 (int Uint32color);
```

Can be used as an argument for `setcolor()`, `setbkcolor()`, `setfillpattern()`, and `setfillstyle()` to set a colour as ARGB integer. The colour is stored in `ARGB_TMP_COL`, and it is also copied in one of the following: `ARGB_FG_COL`, `ARGB_BG_COL`, `ARGB_FILL_COL`.

Functions `ALPHA_VALUE()`, `BLUE_VALUE()`, `GREEN_VALUE()`, and `RED_VALUE()` do not work on temporary colours.

```
int colorRGB (int r, int g, int b) (macro)
```

Can be used to compose a 32 bit colour with *r g b* components; the alpha value is set to 0xff. This macro is typically used to set values in memory buffers.

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

Returns the green component of an ARGB colour in the ARGB palette.

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

Returns 1 if the *current* drawing colour is a standard BGI colour (that is, not ARGB). The *color* argument is actually redundant.

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

Returns 1 if the *current* drawing colour is ARGB. The *color* argument is actually redundant.

```
int RED_VALUE (int color)
```

Returns the red component of an ARGB colour in the ARGB palette.

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

Plots a point at (*x, y*) using the current drawing colour. This function may be faster than `putpixel()`.

```
void closewindow (int id);
```

Closes the window identified by *id*.

```
int edelay (int msec);
```

Waits for *msec* milliseconds. In “slow mode”, a screen refresh is performed. If an event occurs during the delay, this function returns 1, otherwise 0. Use `eventtype()` to get the last event.

```
int event (void);
```

Returns 1 if an event (mouse click, key press, or `SDL_QUIT`) has occurred.

```
int eventtype (void);
```

Returns the type of the last event; either `SDL_KEYPRESS` or `SDL_MOUSEBUTTONDOWN`.

```
void getbuffer (Uint32 *buffer);
```

Copies the contents of the active window to *buffer*, which must be a `getmaxy()+1` × `getmaxx()+1` array of *Uint32* in ARGB format.

```
int getcurrentwindow (void);
```

Returns the *id* of the current window.

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

Waits for a keypress, mouse click, or `SDL_QUIT` event, and returns the code of the key, mouse button, or `QUIT`.

```
void getleftclick (void);
```

Waits for the left mouse button to be clicked and released.

```
void getlinebuffer (int y, Uint32 *linebuffer);
```

Copies the *y*-th screen line to *linebuffer*, which must be a `getmaxx()+1` array of *Uint32* in ARGB format.

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

Returns the maximum possible height for a new window (actual screen height in pixels). This function may be called before graphics initialisation.

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

Returns the maximum possible width for a new window (actual screen width in pixels). This function may be called before graphics initialisation.

```
void getmiddleclick (void);
```

Waits for the middle mouse button to be clicked and released.

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

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

```
void getrightclick (void);
```

Waits for the right mouse button to be clicked and released.

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

Reports the screen width and height in *width* and *height*, regardless of current window dimensions. This function may be called before graphics initialisation.

```
void initpalette (void);
```

Initialises the BGI palette to the standard 16 colours. If the environment variable `SDL_BGI_PALETTE` equals `BGI`, the first 16 colours will use the same RGB values as Turbo C 2.01; otherwise, a brighter palette will be used.

```
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. Multiple windows can be created, unless a fullscreen window is already present.

The user must update the screen as needed using `refresh()`, or use `sdlbgiauto()`.

```
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 putbuffer (Uint32 *buffer);
```

Copies *buffer* to the current window. *buffer* must be a `getmaxy()+1 × getmaxx()+1` array of *Uint32* in ARGB format. This function is faster than direct pixel manipulation.

```
void putlinebuffer (int y, Uint32 *buffer);
```

Copies *linebuffer* to the *y* coordinate in the current window. *linebuffer* must be a `getmaxx()+1` array of *Uint32* in ARGB format. This function is faster than direct pixel manipulation.

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

Reads a `.bmp` file and displays it immediately at (x_1, y_1) . If (x_2, y_2) are not 0, the bitmap is stretched to fit the rectangle $x_1, y_1—x_2, y_2$; otherwise, the bitmap is clipped as necessary.

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

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

```
void setwinoptions (int id, char *title, int x, int y);
```

Resets the window title *title* and position to (x, y) of an existing window identified by *id*. *x* and *y* can be set to `SDL_WINDOWPOS_CENTERED` or `SDL_WINDOWPOS_UNDEFINED`. If either *x* or *y* is -1, the position parameters are ignored.

```
int resizepalette (Uint32 newsize);
```

Resizes the ARGB palette to *newsized*; returns 0 if successful, 1 otherwise. The initial size of the ARGB palette is 4096.

```
void sdlbgiauto (void);
```

Triggers “auto mode”, i.e. `refresh()` is performed automatically. Caveat: it may not work on some graphics cards.

```
void sdlbgifast (void);
```

Triggers “fast mode”, i.e. `refresh()` is needed to display graphics.

```
void sdlbgislow (void);
```

Triggers “slow mode”, i.e. `refresh()` is not needed to display graphics.

```
void setalpha (int col, Uint8 alpha);
```

Sets alpha transparency for colour *col* to *alpha* (0–255); 0 means full transparency, 255 full opacity. `setalpha()` works with colours in both palettes.

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

Sets the current background colour using the *n*-th colour entry in the ARGB palette.

```
void setblendmode (int blendmode);
```

Sets the blend mode to be used with screen refresh. *blendmode* can be `SDL_BLENDMODE_NONE` (default in “slow mode”) or `SDL_BLENDMODE_BLEND`. The latter enables alpha blending.

```
void setcurrentwindow (int id);
```

Sets the current active window to *id*.

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

Sets the current drawing colour using the *n*-th colour entry in the ARGB palette.

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

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

```
void setwinoptions (char *title, int x, int y, Uint32 flags);
```

Sets the window title *title*, the initial position to (*x*, *y*), and SDL2 flags OR’ed together. *x* and *y* can be set to `SDL_WINDOWPOS_CENTERED` or `SDL_WINDOWPOS_UNDEFINED`.

If *title* is an empty string, the window title is set to the default value `SDL_bgi`.

If either *x* or *y* is -1, the position parameters are ignored.

If *flags* is -1, the parameter is ignored; otherwise, only the values `SDL_WINDOW_FULLSCREEN`, `SDL_WINDOW_FULLSCREEN_DESKTOP`, `SDL_WINDOW_SHOWN`, `SDL_WINDOW_HIDDEN`, `SDL_WINDOW_BORDERLESS`, and `SDL_WINDOW_MINIMIZED` can be applied.

```
void setwintitle (intid, char *title);
```

Sets the title of the window identified by *id*.

```
void showerrorbox (const char *message);
```

Opens an error message box with the specified message. The message box waits for the user to click on the OK button.

```
void showinfobox (const char *message);
```

Opens an information message box with the specified message. The message box waits for the user to click on the OK button.

```
int swapbuffers (void);
```

Swaps the current active and the current visual graphics pages.

```
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*.

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

Returns 1 when any key is pressed, including special keys (Ctrl, Shift, etc.); in “slow mode”, a screen refresh is performed. If an `SDL_QUIT` event occurs, `QUIT` is returned.

This document is a free manual, released under the GNU Free Documentation License (FDL) v. 1.3 or later.