NAME

VGLBitmapAllocateBits, **VGLBitmapCopy**, **VGLBitmapCreate**, **VGLBitmapDestroy**, **VGLBitmapPutChar**, **VGLBitmapString**, **VGLBlankDisplay**, **VGLBox**, **VGLCheckSwitch**, **VGLClear**, **VGLEllipse**, **VGLEnd**, **VGLFilledBox**, **VGLFilledEllipse**, **VGLGetXY**, **VGLInit**, **VGLLine**, **VGLKeyboardInit**, **VGLKeyboardEnd**, **VGLKeyboardGetCh**, **VGLMouseInit**, **VGLMouseMode**, **VGLMouseSetImage**, **VGLMouseSetStdImage**, **VGLMouseStatus**, **VGLPanScreen**, **VGLSetBorder**, **VGLSetPalette**, **VGLSetPaletteIndex**, **VGLSetVScreenSize**, **VGLSetXY**, **VGLTextSetFontFile** - Video Graphics Library functions

LIBRARY

Video Graphics Library (libvgl, -lvgl)

SYNOPSIS

#include <sys/fbio.h>
#include <sys/consio.h>
#include <sys/kbio.h>
#include <vgl.h>

int
VGLInit(*int mode*);

void
VGLEnd(*void*);

void
VGLCheckSwitch(*void*);

int
VGLTextSetFontFile(*char *filename*);

int
VGLKeyboardInit(*int code*);

void
VGLKeyboardEnd(*void*);

int
VGLKeyboardGetCh(*void*);

int

VGLMouseInit(*int mode*);

void
VGLMouseMode(*int mode*);

int
VGLMouseStatus(*int *x, int *y, char *buttons*);

void
VGLMouseSetImage(*VGLBitmap *AndMask, VGLBitmap *OrMask*);

void
VGLMouseSetStdImage(*void*);

u_long
VGLGetXY(*VGLBitmap *object, int x, int y*);

void
VGLSetXY(*VGLBitmap *object, int x, int y, u_long color*);

void
VGLLine(*VGLBitmap *object, int x1, int y1, int x2, int y2, u_long color*);

void
VGLBox(*VGLBitmap *object, int x1, int y1, int x2, int y2, u_long color*);

void
VGLFilledBox(*VGLBitmap *object, int x1, int y1, int x2, int y2, u_long color*);

void
VGLEllipse(*VGLBitmap *object, int xc, int yc, int a, int b, u_long color*);

void
VGLFilledEllipse(*VGLBitmap *object, int xc, int yc, int a, int b, u_long color*);

*VGLBitmap **
VGLBitmapCreate(*int type, int xsize, int ysize, byte *bits*);

void
VGLBitmapDestroy(*VGLBitmap *object*);

int
VGLBitmapAllocateBits(*VGLBitmap *object*);

int
VGLBitmapCopy(*VGLBitmap *src, int srcx, int srcy, VGLBitmap *dst, int dstx, int dsty, int width, int hight*);

void
VGLBitmapPutChar(*VGLBitmap *Object, int x, int y, byte ch, u_long fgcol, u_long bgcol, int fill, int dir*);

void
VGLBitmapString(*VGLBitmap *Object, int x, int y, char *str, u_long fgcol, u_long bgcol, int fill, int dir*);

void
VGLClear(*VGLBitmap *object, u_long color*);

void
VGLSetPalette(*byte *red, byte *green, byte *blue*);

void
VGLSetPaletteIndex(*byte color, byte red, byte green, byte blue*);

void
VGLSetBorder(*byte color*);

int
VGLSetVScreenSize(*VGLBitmap *object, int vxsize, int vysize*);

int
VGLPanScreen(*VGLBitmap *object, int x, int y*);

void
VGLBlankDisplay(*int blank*);

DESCRIPTION

Libvgl is a library that enables the programmer access to the graphics modes supported by the console driver (syscons). The library takes care of programming the actual video hardware, and provides a number of simple functions to do various graphic operations. There is also support for a mouse via the

standard mouse system in FreeBSD, see mouse(4), including the ability to transparently have a mouse pointer superimposed on the graphic image currently being worked on. The library takes care of screen switching by storing the current image in memory before switching to another virtual console, and restoring when the user switches back. This allows several graphic applications at once, but on different virtual consoles.

Below is a short description of the various functions:

VGLInit() initialize the library and set up the graphic mode *mode*.

VGLEnd() terminate graphic mode, and restore the screenmode that was active before **VGLInit**() was called.

VGLCheckSwitch() if the program goes into longer periods of processing without doing any graphics output, calling this function occasionally will allow the system to switch screens.

VGLTextSetFontFile() instruct the char/string functions to use the font in file *filename* instead of the builtin font.

VGLKeyboardInit() set up the keyboard in the "raw" I/O mode and specify the key code to be used. *code* must be VGL_XLATEKEYS, VGL_CODEKEYS, or VGL_RAWKEYS. When VGL_XLATEKEYS is specified, the keyboard translates the raw keyboard scan code into a character code. If VGL_RAWKEYS is used, the raw keyboard scan code is read as is. VGL_CODEKEYS is the intermediate key code; each key is assigned a unique code whereas more than one raw scan code may be generated when a key is pressed.

VGLKeyboardEnd() when you have finished using the keyboard, call this function.

VGLKeyboardGetCh() read one byte from the keyboard. As the keyboard I/O is in the "raw" input mode, the function will not block even if there is no input data, and returns 0.

VGLMouseInit() initialize the mouse. The optional on-screen mouse pointer is shown if the argument is VGL_MOUSESHOW.

VGLMouseMode() either shows the mouse pointer if the argument is VGL_MOUSESHOW, or hides the mouse pointer if the argument is VGL_MOUSEHIDE.

VGLMouseStatus() returns the current mouse pointer coordinates and button state in *x*, *y*, buttons. The return value reflects if the mouse pointer is currently shown on screen or not.

VGLMouseSetImage() with this function it is possible to change the image of the mouse pointer on screen.

VGLMouseSetStdImage() this function restores the mouse pointer to the standard arrow.

VGLGetXY() retrieves the color of the pixel located at *x, y,* coordinates of the *object* argument, and returns it as a u_long value.

VGLSetXY() sets the color of the pixel located at *x, y,* coordinates of the *object* argument to *color* u_long value.

VGLLine() draw a line from *x1, y1* to *x2, y2* in color *color*.

VGLBox() draw a box with upper left hand corner at *x1, y1* and lower right hand corner at *x2, y2* in color *color*.

VGLFilledBox() draw a filled (solid) box with upper left hand corner at *x1, y1* and lower right hand corner at *x2, y2* in color *color*.

VGLEllipse() draw an ellipse centered at *xc, yc* make it *a* pixels wide, and *b* pixels high in color *color*.

VGLFilledEllipse() draw a filled (solid) ellipse centered at *xc, yc* make it *a* pixels wide, and *b* pixels high in color *color*.

VGLBitmapCreate() create a bitmap object and initialize it with the specified values and bit data. *type* must be MEMBUF for the in-memory bitmap. *bits* may be NULL so that bitmap data may be associated later.

There also is a macro, **VGLBITMAP_INITIALIZER**(*type*, *xsize*, *ysize*, *bits*) to initialize a statically declared bitmap object.

VGLBitmapDestroy() free the bitmap data and the bitmap object.

VGLBitmapAllocateBits() allocate a bit data buffer for the specified object.

VGLBitmapCopy() copy a rectangle of pixels from bitmap *src* upper left hand corner at *srcx, srcy* to bitmap *dst* at *dstx, dsty* of the size *width, height*.

VGLBitmapPutChar() write the character *ch* at position *x, y* in foreground color *fgcol*. If *fill* is != 0, use the color *bgcol* as background otherwise the background is transparent. The character is drawn in the

direction specified by the argument *dir*.

VGLBitmapString() write the string *str* at position *x, y* in foreground color *fgcol*. If *fill* is != 0, use the color *bgcol* as background otherwise the background is transparent. The string is drawn in the direction specified by the argument *dir*.

VGLClear() clears the entire bitmap to color *color*.

VGLSetPalette() this function sets the palette used, the arguments *red, green, blue* should point to byte arrays of 256 positions each.

VGLSetPaletteIndex() set the palette index *color* to the specified RGB value.

VGLSetBorder() set the border color to color *color*.

VGLSetVScreenSize() change the virtual screen size of the display. Note that this function must be called when our vty is in the foreground. And *object* must be *VGLDisplay*. Passing an in-memory bitmap to this function results in error.

The desired virtual screen width may not be achievable because of the video card hardware. In such case the video driver (and underlying video BIOS) may choose the next largest values. Always examine *object->VXsize* and *VYsize* after calling this function, in order to see how the virtual screen is actually set up.

In order to set up the largest possible virtual screen, you may call this function with arbitrary large values.

```
VGLSetVScreenSize(10000, 10000);
```

VGLPanScreen() change the origin of the displayed screen in the virtual screen. Note that this function must be called when our vty is in the foreground. *object* must be *VGLDisplay*. Passing an in-memory bitmap to this function results in error.

VGLBlankDisplay() blank the display if the argument *blank* != 0. This can be done to shut off the screen during display updates that the user should first see when it is done.

Program termination and signal processing

It is important to call **VGLEnd**() before terminating the program. Care must be taken if you install signal handlers and try to call **VGLEnd**() and exit(3) to end the program. If a signal is caught while the program is inside **libvgl** functions, **VGLEnd**() may not be able to properly restore the graphics hardware.

The recommended way to handle signals and program termination is to have a flag to indicate signal's delivery. Your signal handlers set this flag but do not terminate the program immediately. The main part of the program checks the flag to see if it is supposed to terminate, and calls **VGLEnd**() and exit(3) if the flag is set.

Note that **VGLInit**() installs its internal signal handlers for SIGINT, SIGTERM, SIGSEGV, and SIGBUS, and terminates the program at appropriate time, after one of these signals is caught. If you want to have your own signal handlers for these signals, install handlers *after* **VGLInit**().

SIGUSR1 and SIGUSR2 are internally used by **libvgl** to control screen switching and the mouse pointer, and are not available to **libvgl** client programs.

HISTORY

The **vgl** library appeared in FreeBSD 3.0.

AUTHORS

Søren Schmidt <*sos@FreeBSD.org*>