MegaGraph Graphics Library
Library Reference Manual

Version 2.0
April 1995

CONTENTS

Introduction 1

Chapter 1 - Library Overview 2

Event Functions 2
Mouse Functions 2
Graphics Functions 3
Platform Specific Functions 7

Chapter 2 - Alphabetical Function Reference 9

Reference Entry Template 9

Chapter 3 - C++ Encapsulation 108

Chapter 4 - Data Structures 109

Chapter 5 - File Formats 110

INTRODUCTION

The MegaGraph Graphics Library (MGL) is a full featured graphics library for displaying high performance graphics on personal computers. It provides fast, low level 2D drawing primitives that can be used to implement video games, user interface software, and even real-time 3D animation. It will work in any video mode that has 16 or more colors, including HiColor and TrueColor.

The MGL is aimed at the experienced programmer, so no tutorial documentation or information on generally using libraries to write and successfully compile programs is given. For more information you will need to consult the documentation that came with your compiler.

This document does not provide any information on graphics programming concepts, or general programming information about the MGL. Consult the MGL Programmers Guide for this information.

Following is a summary of the chapters in this reference:

Chapter 1 - Library Overview discusses the various components of MGL and lists all of the functions of MGL by category.

Chapter 2 - Alphabetical Function Reference is a complete description of each function in MGL.

Chapter 3 - C++ Encapsulation describes the C++ encapsulation of MGL.

Chapter 4 - Data Structures is a reference of all of the various data structures used by MGL functions.

Chapter 5 - File Formats describes the file formats used by MGL.

CHAPTER

1

Event Functions

These functions provide a set of low level routines to maintain a queue of keyboard and mouse events. Naturally the events are not restricted to just keyboard and mouse events, but can be any type of user defined event. This allows the program to easily combine the keyboard and mouse into a single system for interacting with the user.

EVT_flush
EVT_getNext
EVT_halt
EVT_peekNext
EVT_post
EVT_setTimerTick

Mouse Functions

These functions provide a high level language interface to the mouse dirver, allowing the user program to fully control the mouse and to install a custom mouse event handler. Full control of the mouse in both text and graphics modes is provided.

MS_available
MS_hide
MS_moveTo
MS_obscure
MS_setCursor
MS_setCursorColor
MS_show

Graphics Functions

These functions provide fast low-level 2D drawing as well as high level window management. They are broken up into general sub-categories.

Environment detection and initialization

MGL_availableModes
MGL_availablePages
MGL_defaultAttributes
MGL_detectGraph
MGL_driverName
MGL_errorMsg
MGL_exit
MGL_fatalError
MGL_getDriver
MGL_getMode
MGL_init
MGL_isDisplayDC
MGL_isMemoryDC
MGL_isWindowedDC
MGL_modeDriverName
MGL_modeName
MGL_registerAllDispDrivers
MGL_registerAllMemDrivers
MGL_registerDriver
MGL_result
MGL_setBufSize
MGL_setResult
MGL_surfaceAccessType
MGL_zbufferAccessType

Color and palette manipulation

MGL_getDefaultPalette
MGL_getPalette
MGL_getPaletteEntry
MGL_getPaletteSize
MGL_realColor
MGL_realizePalette
MGL_setDefaultPalette
MGL_setPalette
MGL_setPaletteEntry

Generic device context information and manipulation

MGL_getBitsPerPixel
MGL_getColorMapMode
MGL_getPixelFormat
MGL_maxColor
MGL_maxPage
MGL_setColorMapMode
MGL_sizex
MGL_sizey

Double buffering support

MGL_doubleBuffer
MGL_getActivePage
MGL_getVisualPage
MGL_setActivePage
MGL_setVisualPage
MGL_singleBuffer
MGL_swapBuffers
MGL_vSync

Zbuffering support

MGL_zBegin
MGL_zShareZBuffer

Routines to change the active global device context

MGL_isCurrentDC
MGL_makeCurrentDC

Current device context information and manipulation

MGL_defaultColor
MGL_getAspectRatio
MGL_getAttributes
MGL_getBackColor
MGL_getBorderColors
MGL_getColor
MGL_getMarkerColor
MGL_getMarkerSize
MGL_getMarkerStyle
MGL_getPenBitmapPattern
MGL_getPenPixmapPattern
MGL_getPenSize
MGL_getPenStyle
MGL_getPolygonType
MGL_getWriteMode
MGL_restoreAttributes
MGL_setAspectRatio
MGL_setBorderColors
MGL_setMarkerColor
MGL_setMarkerSize
MGL_setMarkerStyle
MGL_setPenBitmapPattern
MGL_setPenPixmapPattern
MGL_setPenSize
MGL_setPenStyle
MGL_setPolygonType
MGL_setWriteMode
MGL_unpackColor
MGL_unpackColorRGB

Device clearing

MGL_clearViewport

Viewport and clip rectangle/region manipulation

MGL_getClipMode
MGL_getClipRect
MGL_getViewport
MGL_globalToLocal
MGL_localToGlobal
MGL_maxx
MGL_maxy
MGL_setClipMode
MGL_setClipRect
MGL_setRelViewport
MGL_setViewport

Pixel plotting

MGL_getPixel
MGL_getPixelCoord
MGL_getPixelCoordFast
MGL_getPixelFast

Line drawing and clipping

MGL_clipLineFX
MGL_getCP
MGL_getX
MGL_getY
MGL_line
MGL_lineCoord
MGL_lineCoordFX
MGL_lineCoordFast
MGL_lineFX
MGL_lineEngine
MGL_lineRel
MGL_lineRelCoord
MGL_lineTo
MGL_lineToCoord
MGL_moveRel
MGL_moveRelCoord

MGL_lineCoordFastFX
MGL_lineFast
MGL_lineFastFX
MGL_moveTo
MGL_moveToCoord

Polygon drawing

MGL_fillPolygon
MGL_fillPolygonFX
MGL_fillPolygonFast
MGL_fillPolygonFastFX

3D rasterization routines

MGL_cLineCoordFast
MGL_cLineFast
MGL_czLineCoordFast
MGL_czLineFast
MGL_rgbLineCoordFast
MGL_rgbLineFast
MGL_rgbzLineCoordFast
MGL_rgbzLineFast
MGL_zClearCoord
MGL_zCoord
MGL_zLineCoordFast
MGL_zLineFast

HiColor and TrueColor shade table

MGL_setShadeTable

Polyline drawing

MGL_marker
MGL_polyLine
MGL_polyMarker
MGL_polyPoint

Rectangle drawing

MGL_fillRect
MGL_fillRectCoord
MGL_rect
MGL_rectCoord

Psuedo 3D border drawing

MGL_drawBorder
MGL_drawBorderCoord
MGL_drawHDivider
MGL_drawVDivider

Ellipse drawing

MGL_ellipse
MGL_ellipseArc
MGL_ellipseArcCoord
MGL_ellipseArcEngine
MGL_ellipseCoord
MGL_ellipseEngine
MGL_fillEllipse
MGL_fillEllipseArc
MGL_fillEllipseArcCoord
MGL_fillEllipseCoord

Text attribute manipulation

MGL_charWidth
MGL_getCharMetrics
MGL_getFontMetrics
MGL_getSpaceExtra
MGL_getTextDirection
MGL_getTextJustify
MGL_getTextSettings
MGL_getTextSize
MGL_maxCharWidth
MGL_setSpaceExtra
MGL_setTextDirection
MGL_setTextJustify
MGL_setTextSettings
MGL_setTextSize
MGL_textHeight
MGL_textWidth
MGL_underScoreLocation

Text drawing

MGL_drawStr
MGL_drawStrXY
MGL_getFont
MGL_useFont
MGL_vecFontEngine

BitBlt support

MGL_bitBlt
MGL_bitBltCoord
MGL_getDivot
MGL_getDivotCoord
MGL_putBitmap
MGL_putIcon
MGL_putMonoImage
MGL_stretchBlt
MGL_stretchBltCoord

Monochrome bitmap manipulation

MGL_drawGlyph
MGL_getGlyphHeight
MGL_getGlyphWidth
MGL_mirrorGlyph
MGL_rotateGlyph

Region management

MGL_clearRegion
MGL_copyRegion
MGL_drawRegion
MGL_freeRegion
MGL_newRegion

Region generation primitives

MGL_rgnEllipse
MGL_rgnEllipseArc
MGL_rgnGetArc
MGL_rgnGetArcCoords
MGL_rgnLineCoordFX
MGL_rgnPolygon
MGL_rgnPolygonFast
MGL_rgnSolidEllipse

MGL_rgnLine
MGL_rgnLineCoord
MGL_rgnSolidEllipseArc
MGL_rgnSolidRectCoord

Region alegbra

MGL_diffRegion
MGL_diffRegionRect
MGL_emptyRegion
MGL_equalRegion
MGL_offsetRegion
MGL_sectRegion
MGL_sectRegionRect
MGL_unionRegion
MGL_unionRegionOfs
MGL_unionRegionRect
MGL_ptInRegionCoord

Region traversal

MGL_traverseRegion

RGB to 8 bit halftone dithering routines

MGL_getHalfTonePalette

Resource loading/unloading

MGL_availableBitmap
MGL_availableCursor
MGL_availableFont
MGL_availableIcon
MGL_loadBitmap
MGL_loadBitmapIntoDC
MGL_loadCursor
MGL_loadFont
MGL_loadIcon
MGL_saveBitmapFromDC
MGL_unloadBitmap
MGL_unloadCursor
MGL_unloadFont
MGL_unloadIcon

Rectangle and Point manipulation

MGL_defRect
MGL_defRectPt
MGL_sectRect
MGL_unionRect

Platform Specific Functions

MGL currently supports both Microsoft Windows and DOS as target run-time environments. These functions provide platform specific implementations.

MGLDOS

MGL_beep
MGL_createDisplayDC
MGL_getTickResolution
MGL_getTicks

MGL_createMemoryDC
MGL_delay
MGL_destroyDC
MGL_getPaletteSnowLevel
MGL_resume
MGL_setPaletteSnowLevel
MGL_suspend

MGLWIN

MGL_initWindowed
MGL_createDisplayDCp
MGL_createWindowedDC
MGL_createMemoryDC
MGL_destroyDC
MGL_resizeWinDC
MGL_setWinDC
MGL_getTicks
MGL_getTickResolution
MGL_delay
MGL_beep
MGL_setPaletteSnowLevel
MGL_getPaletteSnowLevel

CHAPTER
2

Reference Entry Template

Function Summary of what the function does.

Syntax <type> function(<type> parameter[,...])

Prototype in header.h

This lists the header file(s) containing the prototype for the function. The prototype of a function may be contained in more than one header file, in which case all the files would be listed, so use whichever one is more appropriate.

Parameters Briefly describes each of the function parameters.

Remarks This section describes what the function does, the parameteres it take and any details you might need to know in order to get full use out of the function.

Return value This section describes the value returned by the function (if any).

See also This section gives a list of other related functions in the library that may be of interest.

EVT_flush

Function Flushes all events of a specified type from the event queue.

Syntax void EVT_flush(int mask);

Prototype in event.h

Remarks Flushes (removes) all pending events of the type specified in the 'mask' parameter from the event queue. You may combine the masks for different event types with a simple logical OR.

Return value None.

See also EVT_halt, EVT_delay

EVT_getNext

Function Retrieves the next pending event from the event queue.

Syntax bool EVT_getNext(event *evt,int mask);

Prototype in event.h

Remarks Retrieves the next pending event defined specified by the 'mask' parameter from the event queue. You may combine the masks for different event types with a simple logical OR. The event queue is adjusted to reflect the new state after the event has been removed.

The event is returned in the structure pointed to by the 'evt' parameter, which is defined in the "event.h" header file as:

```
typedef struct {
  uint      what;
  long      when;
  int       where_x;
  int       where_y;
  long      message;
  int       modifiers;
  } event;
```

The what field defines the type of event that occurred, and will contain one of the following codes defined in "event.h":

```
#define NULLEVT     0x0000
#define KEYDOWN     0x0001
#define KEYREPEAT   0x0002
#define KEYUP       0x0004
#define MOUSEDOWN   0x0008
#define MOUSEUP     0x0010
#define MOUSEMOVE   0x0020
#define TIMERTICK   0x0040
#define USEREVT     0x0080
```

All application specific event should begin with the USEREVT code and build from there. Since the event code is stored in a 16 bit integer, there is a maximum of 16 different event codes that can be distinguished. You can store extra information about the event in the 'message' field to distinguish between events of the same class (for instance the button used in a MOUSEDOWN event).

The when field contains the time in ticks since midnight that the event occurred. This field can be useful to determine whether two mouse clicks were performed close enough together to be recognised as a 'double click'.

The where_x and where_y fields contain the location of the mouse cursor at the time of the specified mouse event. These fields are not used for keyboard events, and may contain event specific information for user defined events if required.

The message field is a 32 bit long integer that contains event specific information. This field is used to store the ASCII code and keyboard scan code of keyboard events (see EVT_asciiCode and EVT_scanCode to extract this information from events), and button press information for mouse events. It may contain any event specific information for user defined events. The bit masks for the MOUSEDOWN and MOUSEUP event message fields is defined in "event.h" as:

```
#define LEFTBMASK   0x0001
#define RIGHTBMASK  0x0004
#define BOTHBMASK   0x0005
#define MIDDLEMASK  0x0010
#define LEFTMBMASK  0x0011
#define RIGHTMBMASK 0x0014
#define ALLBMASK    0x0015
```

The modifiers field contains information about the state of the keyboard shift modifiers (ctrl, alt and shift) and the state of the mouse buttons for mouse events. The modifier masks are defined in "event.h" as:

```
#define LEFTBUT     0x0001
#define RIGHTBUT    0x0002
#define MIDDLEBUT   0x0004
#define RIGHTSHIFT  0x0008
#define LEFTSHIFT   0x0010
#define CTRLSTATE   0x0020
#define ALTSTATE    0x0040
#define LEFTCTRL    0x0080
#define LEFTALT     0x0100
#define SHIFTKEY    (LEFTSHIFT | RIGHTSHIFT)
```

If an event of the specified type was not in the event queue, the what field of the event will be set to NULLEVT, and EVT_getNext will return false.

KEYUP event's only report the raw keyboatd scancode for the key that was pressed, and does not do any conversion of this scancode to standard ASCII character codes. Note also that the scan codes returned by the BIOS are slightly different to the raw scancodes, since the BIOS modifies the scancodes for some keys when a keyboard modifier is held down (ie: Shift-F1 etc).

The TIMERTICK event is used to report that a specified time interval has elapsed since the last TIMERTICK event occurred. See EVT_setTimerTick() for information on how to set this up.

Return value True if an event was pending, false if not.

See also EVT_peekNext, EVT_post, EVT_setTimerTick

EVT_halt

Function Halts program until a specified event occurs.

Syntax void EVT_halt(event *evt,int mask);

Prototype in event.h

Remarks Halts program execution until an event of the type specified by 'mask' has occurred. You may combine the masks for different event types with a simple logical OR.

The event is returned in the structure pointed to by 'evt'. The event that was caught is removed from the event queue and returned. Refer to EVT_getNext for a description of the event structure.

Note that EVT_halt will return immediately if there is already a pending event of the specified type waiting in the event queue. You can call EVT_flush to flush all pending events of the specified type to unconditionally halt program execution.

Return value None.

See also EVT_delay, EVT_flush

EVT_peekNext

Function Peeks at the next pending event in the event queue.

Syntax bool EVT_peekNext(event *evt,int mask);

Prototype in event.h

Remarks Peeks at the next pending event defined specified by the 'mask' parameter in the event queue. In contrast to EVT_getNext, the event is not removed from the event queue. You may combine the masks for different event types with a simple logical OR.

Refer to EVT_getNext for a description of the event structure.

Return value True if an event is pending, false if not.

See also EVT_getNext, EVT_post

EVT_post

Function Posts a user defined event to the event queue

Syntax bool EVT_post(int what,long message,ms_status *stat,int modifiers);

Prototype in event.h

Remarks This routine is normally used to post user events to the event queue, but may also be used to post both mouse and keyboard events to the event queue.

The what field contains the type of event to post, while the message and modifiers fields contain the information to store in the associated parts of the event structure. The stat field is used to pass mouse status information, such as the button state and location of the mouse cursor (refer to the mouse interface documentation for information on the ms_status structure) to be stored in the posted event.

If you pass a NULL for the stat field, the event posting routine will obtain the current mouse status information from the mouse module for you.

If you pass a -1 as the value for the modifiers, the modifier field will be set to the current state of the keyboard modifiers along with the current state of the mouse buttons (this is taken from the status info passed or the info obtain from the mouse module).

Return value True if event was posted, false if event queue is full.

See also EVT_getNext, EVT_peekNext

EVT_setTimerTick

Function Set the interval between TIMERTICK events

Syntax int EVT_setTimerTick(int ticks);

Prototype in event.h

Remarks This routine sets the number of ticks between each posting of the TIMERTICK event to the event queue. The TIMERTICK event is off by default. You can turn off the posting of TIMERTICK events by setting the tick interval to 0.

One tick is approximately equal to 1/18.2 of a second. To work out a precise interval given in seconds, use the following expression:

```
ticks = secs * (1193180.0 / 65536.0);
```

Return value Old value of timer tick interval

MGL_availableBitmap

Function Verifies the existance of a bitmap.

Syntax
```
bool MGL_availableBitmap(const char *bitmapName);
```

Prototype in mgraph.h

Parameters bitmapName - Name of bitmap file to check for

Remarks Attempt to locate the bitmap file specified by 'bitmapName', and verify that it is available for use.

Return value True if the bitmap file exists, false if not.

See also MGL_loadBitmap

MGL_availableCursor

Function Verifies the existence of a cursor.

Syntax
```
bool MGL_availableCursor(const char *cursorName);
```

Prototype in mgraph.h

Parameters cursorName - Name of cursor file to check for

Remarks Attempts to locate the cursor file specified by 'cursorName' and verify that it is available for use.

Return value True if the cursor file exists, false if not.

See also `MGL_loadCursor`

MGL_availableFont

Function Determines if a specific font file is available for use.

Syntax bool MGL_availableFont(const char *fontname);

Prototype in mgraph.h

Parameters fontname - Relative filename of the required font file

Remarks MGL_availableFont() is used to determine if a particular font file exists and is available for use. It will check first in the directory where the MGL driver files were located, and then in the current directory. If the font file is found it will return true, otherwise it will return false.

Return value True if font file is available, false if not.

See also MGL_loadFont()

MGL_availableIcon

Function Determines if the specified icon file is available for use.

Syntax bool MGL_availableIcon(const char *iconName);

Prototype in mgraph.h

Parameters iconName - Name of icon file to check for.

Remarks Attempt to locate the icon file specified by 'iconName', and verify that it is available for use.

Return value True if the icon file exists, False if not.

See also MGL_loadIcon

MGL_availableModes

Function Returns a list of all available video modes.

Syntax uchar *MGL_availableModes(void);

Prototype in mgraph.h

Parameters None

Remarks MGL_availableModes() will return a list of all the currently available video modes. You may call this routine before MGL_init() is called to determine what video modes are available before actually initialising a particular video mode. You must ensure however that the MGL_detectGraph() routine is called before you call the MGL_availableModes() routine.

The list of available video modes is returned as a table of 8-bit integer values. The table is terminate with a -1(0xFF).

Return value Pointer to list of available video modes

See also MGL_detectGraph(), MGL_init(), MGL_availablePages()

MGL_availablePages

Function Determine the number of available video pages for a specific video mode.

Syntax int MGL_availablePages(int mode);

Prototype in mgraph.h

Parameters mode - MGL mode number to query.

Remarks MGL_availablePages() will return the number of pages of pysical video memory available for a specific MGL video mode. You must ensure that you have called MGL_detectGraph() before you call this function.

You may call this routine before the MGL_init routine is used to intialise a particular video modes. Thus you can filter out support for modes that do not have the require number of hardware video pages available for use.

Return value Number of available video pages for mode, -1 on error.

See also MGL_detectGraph(), MGL_init(), MGL_availableMode()

MGL_backfacing

Function Determines if a polygon is backfacing

Syntax int MGL_backfacing(fix32_t dx1,fix32_t dy1,fix32_t dx2,fix32_t dy2);

Prototype in mgraph.h

Parameters dx1 - change in x along first edge
dy1 - change n y along first edge
dx2 - change in x along second edge
dy2 - chang in y along second edge

Remarks Determine whether a polygon is backfacing given two fixed point vectors. The vectors need to be derived from two consecutive counterclockwise edges of the polygon in order for this function to return accurate results.

Return value 1 if the polygon if backfacing, 0 if it is frontfacing

See also

MGL_beginDrawing

Function Setup for high performance drawing operation.

Syntax void MGL_beginDrawing(void);

Prototype in mgraph.h

Remarks MGL_beginDrawing() must be called before calling any of the special high performance drawing routines, such as the MGL_lineFast() and associated routines. MGL_beginDrawing will put the video hardware into a state that can be used to render the primitives much faster than otherwise possible.

All high performance drawing primitives end the word 'Fast', and must be bracketed between calls to MGL_beginDrawing() and MGL_endDrawing(). You may mix and match any of the fast drawing routines and normal library functions within an MGL_beginDrawing() and MG_endDrawing() block. You can also nest calls to MGL_beginDrawing() and MGL_endDrawing() (hence you can write your own functions that use the fast rendering routines but do not require the user to call MGL_beginDrawing() before calling your routine).

You *must* ensure that you call the MGL_endDrawing() routine after drawing a set of high performance primitives.

See also MGL_endDrawing(), MGL_*fast()

MGL_beginPixel

Function Setup for high speed pixel drawing.

Syntax void MGL_beginPixel(void);

Prototype in mgraph.h

Remarks MGL_beginPixel() sets up the video hardware for plotting single pixels as fast a possible. You *must* call this routine before calling any of the MGL_*Pixel() routines to ensure correct operation, and you *must* call the MGL_endPixel() routine after you have finished.

This routine is intended primarily to ensure fast operation if you intend to plot more than a single pixel at a time, so it will be done as fast as possible.

See also MGL_endPixel(), MGL_*Pixel

MGL_beginShadedDrawing

Function Setup for smooth shaded drawing operation.

Syntax void MGL_beginShadedDrawing(void);

Prototype in mgraph.h

Remarks MGL_beginShadedDrawing() must be called before any of the smooth shading routines are called, such as the MGL_rgbzTriFast() routine. This will put the video hardware into a special mode for doing high peformance smooth shaded rendering.

You should *not* call any other routines except the smooth shading routines between calls to MGL_beginShadedDrawing() and MGL_endShadedDrawing(), and you should not nest calls to this routine.

See also MGL_endShadedDrawing, MGL_beginDrawing

MGL_BeginZDrawing

Function Setup for z-buffering drawing operation.

Syntax void MGL_beginZDrawing(void);

Prototype in mgraph.h

Parameters None

Remarks MGL_beginZDrawing() must be called before any of the z-buffering routines are called, such as the MGL_zTriFast() routine. This will put the hardware into a special mode for doing high peformance z-buffered rendering.

Return value None.

See also MGL_endZDrawing()

MGL_beginZShadedDrawing

Function Setup for smooth shaded, z-buffered drawing.

Syntax void MGL_beginZShadedDrawing(void)

Prototype in mgraph.h

Parameters None

Remarks MGL_beginZShadedDrawing() must be called before any of the shaded,z-buffering routines are called, such as the MGL_czTriFast() routine. This will put the hardware into a special mode for doing high peformance z-buffered, smooth shaded rendering.

Return value None

See also MGL_endZShadedDrawing()

MGL_bitBlt

Function Blt a block of image data from on device context to another.

Syntax void MGL_bitBlt(MGLDC *dst,MGLDC *src,rect_t r,,int dstLeft, int dstTop, int top);

Prototype in mgraph.h

Parameters dst - destination device context
src - Source device context
r - Source rectangle
dstLeft - left destination coordinate
dstTop - top destination coordinate
op - Write mode to use during Blt

Remarks Blts a block of image data form one device context into another. This function will correctly handle Blt's across device contexts of differing pixel depth.

The destination image will be clipped to the current clipping rectangle.

Return value None

See also MGL_bitBltCoord, MGL_stretchBlt, stretchBltCoord

MGL_bitBltCoord

Function Blt a block of image data from on device context to another.

Syntax void PUBAPI MGL_bitBltCoord(MGLDC *dst,MGLDC *src,int left,int top,int right,int bottom,int dstLeft,int dstTop,int op);

Prototype in mgraph.h

Parameters dst - destination device context
src - source device context
left - left coordinate of source image
top - top coordinate of source image
right - right coordinate of source image
bottom - bottom coordinate of source image
dstLeft - left coordinate of destination
dstTop - right coordinate of destination
op - write mode to use during Blt

Remarks Blts a block of image data form one device context into another. This function will correctly handle Blt's across device contexts of differing pixel depth.

The destination image will be clipped to the current clipping rectangle.

Return value None

See also MGL_bitBlt, MGL_stretchBltCoord, MGL_stretchBlt

MGL_charWidth

Function Returns the width of a character in pixels.

Syntax int MGL_charWidth(char ch);

Prototype in mgraph.h

Parameters ch - Character to measure.

Remarks MGL_charWidth() will return the width of the specified character, given the currently active font and attribute settings.

Return value Width of the character in pixels (will depend on currently active font)

See also MGL_textWidth(), MGL_useFont()

MGL_clearDevice

Function Clears the currently active display page on the active device..

Syntax void MGL_clearDevice(void);

Prototype in mgraph.h

Remarks MGL_clearDevice() will clear the entire currently active display page in the current background color. This is the fastest way to clear an entire display page, but if you wish to only clear a portion of the page, use the MGL_clearViewport() routine instead.

See also MGL_clearViewport()

MGL_clearViewport

Function Clears the currently active viewport.

Syntax void MGL_clearViewport(void);

Prototype in mgraph.h

Remarks MGL_clearViewport() will clear the currently active display page viewport in the current background color. This is the fastest way to clear a rectangular viewport, but you may also wish to use the MGL_fillRect() routine to fill in an arbitrary pattern instead, as the MGL_clearViewport() always clears the viewport in a solid color.

See also MGL_clearDevice()

MGL_clipLine

Function Clips a line to a specified clipping rectangle.

Syntax bool MGL_clipLine(point *p1,point *p2,rect r);

Prototype in mgraph.h

Parameters p1,p2 - Endpoints of line to clip.
r - Rectangle to clip the line to

Remarks MGL_clipLine() will clip the line from point p1(x,y) to point p2(x.y) to the specified clipping rectangle. If the line is accepted (ie: it intersects with the clipping rectangle) then the routine will return true and the line enpoints will be updated to reflect the clipped lines new endpoints. If the line is completely outside of the clipping rectangle, the routine will return false.

Return value True if line is visible, false if completely outside rectangle.

MGL_copyImage

Function Copies of block of video memory.

Syntax void MGL_copyImage(rect r,int dl,int dt,int sp,int dp)

Prototype in mgraph.h

Parameters r - Rectangle specifying the region to copy
dl - Left coordinate of destination location
dt - Top coordinate of destination location
sp - Source image page
dp - Destination image page

Remarks MGL_copyImage() copies the data of a rectangular region of the specified display page to another location, either on the same page or on a different page. If the source and destination regions overlap, the image will be correctly copied.

NOTE: No clipping is done in this routine, so you can copy data outside of the current viewport.

See also MGL_copyImageCoord()

MGL_copyImageCoord

Function Copies of block of video memory.

Syntax void MGL_copyImageCoord(int left,int top,int right,int bottom,int dl,int dt,int sp,int dp)

Prototype in mgraph.h

Parameters left - Left coordinate of source region
top - Top coordinate of source region
right - Right coordinate of source region
bottom - Bottom coordinate of source region
dl - Left coordinate of destination location
dt - Top coordinate of destination location
sp - Source image page
dp - Destination image page

Remarks MGL_copyImageCoord() copies the data of a rectangular region of the specified display page to another location, either on the same page or on a different page. If the source and destination regions overlap, the image will be correctly copied.

NOTE: No clipping is done in this routine, so you can copy data outside of the current viewport.

See also MGL_copyImage()

MGL_defaultAttributes

Function Reset global attributes to their default values.

Syntax void MGL_defaultAttributes(void);

Prototype in mgraph.h

Remarks MGL_defaultAttributes() resets all of the global attributes to their default values. MGL_defaultAttributes() does not reset the current palette.

See also MGL_getAttributes(), MGL_setAttributes(), MGL_getDefaultPalette()

MGL_defaultColor

Function Returns the default color value for the video mode.

Syntax color_t MGL_defaultColor(void);

Prototype in mgraph.h

Remarks MGL_defaultColor() returns a default color values for the current video mode. This color value is white if the palette has not been changed, and will always be white in direct color modes.

Return value Default color value for current video mode.

See also MGL_setColor(), MGL_getColor()

MGL_detectGraph

Function Detects the currently installed video hardware.

Syntax void MGL_detectGraph(int far *graphdriver,int far *graphmode);

Prototype in mgraph.h

Parameters graphdriver - Pointer to graphics device driver id
graphmode - Pointer to default video mode for device

Remarks MGL_detectGraph() autodetects the presence of all the standard graphics adapters supported by the MGL. If no suitable hardware is detected, it returns grNONE in the graphdriver parameter. If suitable hardware is detected, it will return an appropriate device driver id number in the graph driver parameter and a suggested default video mode in the graphmode parameter.

The video device drivers currently supported by the MGL are:

grDETECT - Auto detect the graphics subsystem
grNONE - No graphics hardware detected
grEGA - Standard EGA with 256k RAM
grVGA - Standard VGA
grSVGA - VESA VBE compliant Super VGA
grSVGA_S3 - S3 accelerated SuperVGA

The MGL supports a number of different video mode resolutions, ranging from 320x200 up to 1280x1024 with color ranges from 16 colors up to 16.7 million colors. Thus the video mode id for 320x200 16 color EGA graphics mode is grEGA_320x200x16, and the 1280x1024 16.7 million color mode is grSVGA_1280x1024x16m. Please consult the MGRAPH.H header file for a full list of defined video modes (to large to list here).

Once you have called MGL_detectGraph(), you can then call the MGL_availableModes and MGL_availablePages to determine all the available video modes for the currently installed driver and the number of available video pages for each video mode.

See also MGL_init()

MGL_disjointRect

Function Determines if two rectangles are disjoint.

Syntax bool MGL_disjointRect(recy r1,rect r2)

Prototype in mgraph.h

Parameters r1 - First rectangle to test
r2 - Second rectangle to test

Remarks MGL_disjointRect() will determine if two rectangles are non-overlapping (ie: the intersection between the rectangles is zero). If they are disjoint, it will return true, otherwise it will return false.

Return value True if the rectangles are disjoint, false if not.

See also MGL_emptyRect(), MGL_equalRect(), MGL_offsetRect(), MGL_insetRect(), MGL_ptInRect().

MGL_divotSize

Function Number of bytes required to store the divot.

Syntax long MGL_divotSize(rect r)

Prototype in mgraph.h

Parameters r - Rectangle defining the divot area.

Remarks MGL_divotSize() determines the number of bytes required to store a particular video memory divot.

Return value Size of the specified divot in bytes.

See also MGL_getDivot(), MGL_getDivotCoord(), MGL_putDivot()

MGL_divotSizeCoord

Function Number of bytes required to store the divot.

Syntax long MGL_divotSizeCoord(int left,int top,int right,int bottom)

Prototype in mgraph.h

Parameters left - Left coordinate of divot area
top - Top coordinate of divot area
right - Right coordinate of divot area
bottom - Bottom coordinate of divot area

Remarks MGL_divotSizeCoord() determines the number of bytes required to store a particular video memory divot.

Return value Size of the specified divot in bytes.

See also MGL_getDivot(), MGL_getDivotCoord(), MGL_putDivot()

MGL_doubleBuffer

Function Sets up for double buffering if possible.

Syntax bool MGL_doubleBuffer(void);

Prototype in mgraph.h

Remarks MGL_doubleBuffer() will attempt to enable double buffering for the current video mode. If double buffering is enabled, then the routine will return true, otherwise it will return false and single buffer mode will still be enabled.

MGL_doubleBuffer() will attempt to perform double buffering using an offscreen memory buffer if only one video memory page is supported by the hardware, so it is a transparent method of enabling simple double buffered animation.

NOTE: The current version of the MGL does not yet support off screen memory buffers (mostly because of memory requirements in real mode - the protected mode version of the library will, and limited support will be provided in real mode for the next version).

Return value True if double buffering is now enabled.

See also MGL_singleBuffer(), MGL_swapBuffers()

MGL_drawBorder

Function Draws a 3D border around the specified rectangle.

Syntax void MGL_drawBorder(rect r,int style,int thickness)

Prototype in mgraph.h

Parameters r - Rectangle defining the region to draw the border around
style - Type of border to draw
thickness - Thickness of the border in pixels

Remarks MGL_drawBorder() draws a 3D style border around the specified rectangle in the specified border style and thickness. The different border styles are defined in the MGRAPH.H header file as:

BDR_INSET - Interior is inset into display
BDR_OUTSET - Interior is outset from display
BDR_OUTLINE - Border is a 3D outline

Thus if you wish to create the effect of a 3D style push button, you would use the BDR_OUTSET style to make it appear to be protruding from the display screen. To make it look like it is depressed, use the BDR_INSET style to make it appear to be inset into the display. The BDR_OUTLINE style is used to create a 3D looking rectangle just inside the rectangle boundaries for grouping things together in a user interface dialog box.

The border is completely contained within the specified bounding rectangle.

The color of the border's highlights are determined by the currently active border color values.

See also MGL_drawBorderCoord(), MGL_drawHDivider(), MGL_drawVDivider(), MGL_setBorderColors()

MGL_drawBorderCoord

Function Draws a 3D border around the specified rectangle.

Syntax MGL_drawBorderCoord(int left,int top,int right,int bottom,int style,int thickness);

Prototype in mgraph.h

Parameters r - Rectangle defining the region to draw the border around
style - Type of border to draw
thickness - Thickness of the border in pixels

Remarks MGL_drawBorderCoord() draws a 3D style border around the specified rectangle in the specified border style and thickness. The different border styles are defined in the MGRAPH.H header file as:

BDR_INSET - Interior is inset into display
BDR_OUTSET - Interior is outset from display
BDR_OUTLINE - Border is a 3D outline

Thus if you wish to create the effect of a 3D style push button, you would use the BDR_OUTSET style to make it appear to be protruding from the display screen. To make it look like it is depressed, use the BDR_INSET style to make it appear to be inset into the display. The BDR_OUTLINE style is used to create a 3D looking rectangle just inside the rectangle boundaries for grouping things together in a user interface dialog box.

The border is completely contained within the specified bounding rectangle.

The color of the border's highlights are determined by the currently active border color values.

See also MGL_drawBorder(), MGL_drawHDivider(), MGL_drawVDivider(), MGL_setBorderColors()

MGL_drawHDivider

Function Draws a 3D looking horizontal dividing line

Syntax void MGL_drawHDivider(int y,int x1,int x2);

Prototype in mgraph.h

Parameters y - Y coordinate to draw the line at
x1 - Starting X coordinate
x2 - Ending X coordinate

Remarks MGL_drawHDivider() draws a 3D horizontal dividing line beginning at the position (x1,y) and ending at the position (x2,y).

The color of the dividing line is determine by the currently active border colors values.

See also MGL_drawVDivider(), MGL_drawBorder(), MGL_setBorderColors()

MGL_drawStr

Function Draws a text string on the display.

Syntax void MGL_drawStr(const char *str);

Prototype in mgraph.h

Parameters str - String to display

Remarks MGL_drawStr() draws the specified string at the current position (CP) in the current drawing color, write mode, font, text direction and justification. The CP is moved so that drawing will begin directly after the end of the string, only if the horizontal justification is set to LEFT_TEXT, otherwise the CP is not moved.

See also MGL_drawStrXY(), MGL_textHeight(), MGL_textWidth(), MGL_useFont()

MGL_drawStrXY

Function Draws a text string on the display.

Syntax void MGL_drawStrXY(int x,int y,const char *str);

Prototype in mgraph.h

Parameters x - X coordinate to begin rendering the string at
y - Y coordinate to begin rendering the string at
str - String to display

Remarks MGL_drawStrXY() draws the specified string at the specified (x,y) position in the current drawing color, write mode, font, text direction and justification.

See also MGL_drawStrXY(), MGL_textHeight(), MGL_textWidth(), MGL_useFont()

MGL_drawVDivider

Function Draws a 3D looking vertical dividing line

Syntax void MGL_drawVDivider(int x,int y1,int y2);

Prototype in mgraph.h

Parameters x - X coordinate to begin drawing divider
y1 - Starting Y coordinate
y2 - Ending Y coordinate

Remarks MGL_drawVDivider() draws a 3D vertical dividing line beginning at the position (x,y1) and ending at the position (x,y2).

The color of the dividing line is determine by the currently active border colors values.

See also MGL_drawHDivider(), MGL_drawBorder(), MGL_setBorderColors()

MGL_driverName

Function Returns a string describing the name of the device driver.

Syntax char *MGL_driverName(int driver);

Prototype in mgraph.h

Parameters driver - Device driver id number.

Remarks MGL_driverName() returns a string describing the a device driver given a device driver number.

Return value Pointer to device driver name string.

See also MGL_modeName()

MGL_ellipse

Function Draws an ellipse outline defined by a bounding rectangle.

Syntax bool MGL_ellipse(rect extentRect);

Prototype in mgraph.h

Parameters extentRect - Bounding rectangle defining the ellipse.

Remarks MGL_ellipse() draws the outline of an ellipse just inside the mathematical boundary of *extentRect*. The ellipse outline is drawn in the current pen color, style and size.

Note: This routine is more versatile than the equivalent MGL_ellipseCoord() routine, as you can have an ellipse with odd diameters values, which you cannot get with the MGL_ellipseCoord() routine.

Return value True if ellipse was scan converted, false if an error occured (MGL_buf too small)..

See also MGL_ellipseCoord(), MGL_fillEllipse(), MGL_fillEllipseCoord()

MGL_ellipseArc

Function Draws an elliptical arc outline defined by a bounding rectangle.

Syntax bool MGL_ellipseArc(rect extentRect,int startAngle,int endAngle);

Prototype in mgraph.h

Parameters extentRect - Bounding rectangle defining the arc.
startAngle - Starting angle for arc (in degrees)
endAngle - Ending angle for arc (in degrees).

Remarks MGL_ellipseArc() draw the outline of an elliptical arc just inside the mathematical boundary of *extentRect*. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is treated MOD 360 also. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the centre of the rectangle through it's top right corner, even if the rectangle isn't square.

The ellipse outline if drawn in the current pen color, style and size (note that the current version of the MGL only works with a 1x1 pen size with a solid pattern for elliptical arcs).

Note: This routine is more versatile than the equivalent MGL_ellipseArcCoord() routine, as you can have an elliptical arc with odd diameters values, which you cannot get with the MGL_ellipseArcCoord() routine.

Return value True if the ellipse arc was scan converted, false if an error occurred (MGL_buf too small).

See also MGL_ellipseArcCoord(), MGL_fillEllipseArc(), MGL_fillEllipseArcCoord()

MGL_ellipseArcCoord

Function Draws an elliptical arc outline defined by a centre and radii.

Syntax bool MGL_ellipseArcCoord(int x,int y,int xradius,int yradius,int startAngle,int endAngle);

Prototype in mgraph.h

Parameters x,y - Centre of the elliptical arc.
xradius - X radius for the elliptical arc
yradius - Y radius for the elliptical arc
startAngle - Starting angle for arc (in degrees)
endAngle - Ending angle for arc (in degrees).

Remarks MGL_ellipseArcCoord() draws the outline of an elliptical arc given the centre, radii and starting and endling angles. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is treated MOD 360 also. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the centre of the rectangle through it's top right corner, even if the rectangle isn't square.

The ellipse outline if drawn in the current pen color, style and size (note that the current version of the MGL only works with a 1x1 pen size with a solid pattern).

Note: This routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and it provided for compatabilty with other graphics packages).

Return value True if the ellipse arc was scan converted, false if an error occurred (MGL_buf too small).

See also MGL_ellipseArc(), MGL_fillEllipseArc(), MGL_fillEllipseArcCoord()

MGL_ellipseCoord

Function Draws an ellipse outline defined by a bounding rectangle.

Syntax bool MGL_ellipseCoord(int x,int y,int xradius,int yradius);

Prototype in mgraph.h

Parameters x,y - Centre of the ellipse
xradius - X radius for the ellipse
yradius - Y radius for the ellipse

Remarks MGL_ellipse() draws the outline of an ellipse given the centre and radii for the ellipse. The ellipse outline if drawn in the current pen color, style and size.

Note: This routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and it provided for compatability with other graphics packages).

Return value True if ellipse was scan converted, false if an error occured (MGL_buf too small)..

See also MGL_ellipse(), MGL_fillEllipse(), MGL_fillEllipseCoord()

MGL_ellipseEngine

Function Generates the set of points on an ellipse.

Syntax void MGL_ellipseEngine(rect extentRect,void (*setup)(int topY,int botY,int left,int right),void (*set4pixels)(bool inc_x,bool inc_y,bool region1),void (*finished)(void));

Prototype in mgraph.h

Parameters extentRect - Bounding rectangle defining the ellipse.
setup - Routine called to intialise pixel plotting routines.
set4pixels - Routine called repeatedly for each set of 4 pixels.
finished - Routine called to complete plotting pixels.

Remarks MGL_ellipseEngine() is the same engine used to generate all ellipse's and elliptical arc in the MGL. You can call it to generate the set of points on an ellipse, calling your own user defined plotting routines.

The *setup* routine is called before any pixel are plotted with the coordinates of the 4 seed points in the four ellipse quadrants.

The *set4pixels* routine is called repeatedly for each set of 4 pixels to be plotted, and specified whether the coordinates in the x and y directions should be incremented or remain the same. This state of the 4 pixel coordinates will need to be maintained by the user supplied routines.

The *finished* routine is called to clean up after generating all the points on the ellipse, such as releasing memory and rendering the ellipse if the rendering was deferred.

MGL_emptyRect

Function Determines if a rectangle is empty.

Syntax bool MGL_emptyRect(rect r)

Prototype in mgraph.h

Parameters r - rectangle to test

Remarks MGL_emptyRect() determine if a rectangle is empty. A rectangle is defined as being empty if the right coordinate is less than or equal to the left coordinate, or if the bottom coordinate is less than or equal to the top coordinate.

Return value True if rectangle is empty, false if not.

See also MGL_disjointRect(), MGL_equalRect(), MGL_offsetRect(), MGL_insetRect(), MGL_ptInRect().

MGL_endDrawing

Function Finish high performance drawing operation.

Syntax void MGL_endDrawing(void);

Prototype in mgraph.h

Remarks MGL_endDrawing() is called to end a set of high performance drawing operations that we started with the MGL_beginDrawing() operation.

All high performance drawing primitives end the word 'Fast', and must be bracketed between calls to MGL_beginDrawing() and MGL_endDrawing(). You may mix and match any of the fast drawing routines and normal library functions within an MGL_beginDrawing() and MG_endDrawing() block. You can also nest calls to MGL_beginDrawing() and MGL_endDrawing() (hence you can write your own functions that use the fast rendering routines but do not require the user to call MGL_beginDrawing() before calling your routine).

You *must* ensure that you call the MGL_endDrawing routine after drawing a set of high performance primitives.

See also MGL_beginDrawing(), MGL_*fast()

MGL_endGouraud

Function Complete Gouraud Shaded drawing operation.

Syntax void MGL_endGouraud(void);

Prototype in mgraph.h

Remarks MGL_endGouraud() is called to end the rendering of gouraud shaded primitives, such as the MGL_fillGouraudPolygon() routine.

You should *not* call any other routines except the gouraud shading routines between calls to MGL_beginGouraud and MGL_endGouraud, and you should not nest calls to this routine.

See also MGL_beginGouraud()

MGL_endPixel

Function End high speed pixel drawing operation.

Syntax void MGL_endPixel(void);

Prototype in mgraph.h

Remarks MGL_endPixel() ends a set of high speed pixel drawing operations.

This routine is intended primarily to ensure fast operation if you intend to plot more than a single pixel at a time, so it will be done as fast as possible.

See also MGL_beginPixel()

MGL_equalPoint

Function Determines if two points are equal.

Syntax bool MGL_equalPoint(point p1,point p2);

Prototype in mgraph.h

Parameters p1 - First point to compare
p2 - Second point to compare

Remarks MGL_equalPoint() determines if the value of two coordinates are equal, returning true if they are.

Return value True if the points are equal, false if not.

MGL_equalRect

Function Determines if two rectangles are equal.

Syntax bool MGL_equalRect(rect r1,rect r2);

Prototype in mgraph.h

Parameters r1 - First rectangle to compare
r2 - Second rectangle to compare

Remarks MGL_equalRect() compares the values of two rectangles and determines if they are equal, returning true if they are.

Return value True if the rectangles are equal.

See also MGL_disjointRect(), MGL_emptyRect(), MGL_offsetRect(), MGL_insetRect(), MGL_ptInRect().

MGL_errorMsg

Function Returns a string describing an error condition.

Syntax char *MGL_errorMsg(int err);

Prototype in mgraph.h

Parameters err - Error code to obtain string for.

Remarks MGL_errorMsg() returns a pointer to a string describing a specified error condition. You can use this to convert the error codes from a numerical id return by MGL_result() to a string which you can display for the users of your programs.

Return value Pointer to string describing the error condition.

See also MGL_result().

MGL_exit

Function Closes down the graphics subsystem.

Syntax void MGL_exit(void);

Prototype in mgraph.h

Remarks MGL_exit() closes down the graphics subsystem, deallocating any memory allocated for use by the MGL, and restoring the system back into the original text mode active before the MGL was started.

See also MGL_init()

MGL_fadePalette

Function Fades the values for programming the palette.

Syntax bool MGL_fadePalette(palette *dest,palette *src,int numColors,uchar intensity);

Prototype in mgraph.h

Parameters dest - Pointer to destination palette structure.
src - Pointer to source palette structure.
numColors - Number of colors in palette to fade
intensity - Intensity value for the output palette values.

Remarks MGL_fadePalette() will take the values from one palette structure, fade the values and store them into a second palette structure. The actual hardware palette will not be programmed, so you will then need to make a call to MGL_setPalette() to make the changes visiable.

The *intensity* value is a number between 0 and 255 that defines the intensity of the output values. An intensity of 255 will produce the same output values as the input values. An intensity of 128 will product values in the output palette that are half the intensity of the input palette.

The palette values are manipulated as arrays of the *palette* structure, defined as:

```
typedef struct {
  uchar  red;
  uchar  green;
  uchar  blue;
  } palette;
```

where red, green and blue define the color components for that entry. Each color component is defined as an 8 bit value with a range of 0-255.

If the entire output palette is zero, then the routine will return true, otherwise it will return true.

Return value True if the entire output palette is zero, false if not.

See also MGL_setPalette(), MGL_getPalette(), MGL_rotatePalette()

MGL_fillEllipse

Function Fills an ellipse outline defined by a bounding rectangle.

Syntax bool MGL_fillEllipse(rect extentRect);

Prototype in mgraph.h

Parameters extentRect - Bounding rectangle defining the ellipse.

Remarks MGL_fillEllipse() fills an ellipse just inside the mathematical boundary of *extentRect*. The ellipse is filled in the current pen color and style.

Note: This routine is more versatile than the equivalent MGL_fillEllipseCoord() routine, as you can have an ellipse with odd diameters values, which you cannot get with the MGL_fillEllipseCoord() routine.

Return value True if ellipse was scan converted, false if an error occured (MGL_buf too small)..

See also MGL_ellipse(), MGL_ellipseCoord(), MGL_fillEllipseCoord()

MGL_fillEllipseArc

Function Fills an elliptical arc defined by a bounding rectangle.

Syntax bool MGL_fillEllipseArc(rect extentRect,int startAngle,int endAngle);

Prototype in mgraph.h

Parameters extentRect - Bounding rectangle defining the arc.
startAngle - Starting angle for arc (in degrees)
endAngle - Ending angle for arc (in degrees).

Remarks MGL_fillEllipseArc() fills an elliptical arc forming a wedge, just inside the mathematical boundary of *extentRect*. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is treated MOD 360 also. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the centre of the rectangle through it's top right corner, even if the rectangle isn't square.

The elliptical arc is filled in the current pen color and style.

Note: This routine is more versatile than the equivalent MGL_fillEllipseArcCoord() routine, as you can have an elliptical arc with odd diameter values, which you cannot get with the MGL_fillEllipseArcCoord() routine.

Return value True if the ellipse arc was scan converted, false if an error occurred (MGL_buf too small).

See also MGL_fillEllipseArcCoord(), MGL_ellipseArc(), MGL_ellipseArcCoord()

MGL_fillEllipseArcCoord

Function Fills an elliptical arc defined by a centre and radii.

Syntax bool MGL_fillEllipseArcCoord(int x,int y,int xradius,int yradius,int startAngle,int endAngle);

Prototype in mgraph.h

Parameters x,y - Centre of the elliptical arc.
xradius - X radius for the elliptical arc
yradius - Y radius for the elliptical arc
startAngle - Starting angle for arc (in degrees)
endAngle - Ending angle for arc (in degrees).

Remarks MGL_fillEllipseArcCoord() fills an elliptical arc forming a wedge, given the centre, radii and starting and endling angles. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is treated MOD 360 also. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the centre of the rectangle through it's top right corner, even if the rectangle isn't square.

The elliptical arc is filled in the current pen color and style.

Note: This routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and it provided for compatabilty with other graphics packages).

Return value True if the ellipse arc was scan converted, false if an error occurred (MGL_buf too small).

See also MGL_ellipseArc(), MGL_ellipseArcCoord(), MGL_fillEllipseArc().

MGL_fillEllipseCoord

Function Fills an ellipse defined by a bounding rectangle.

Syntax bool MGL_fillEllipseCoord(int x,int y,int xradius,int yradius);

Prototype in mgraph.h

Parameters x,y - Centre of the ellipse
xradius - X radius for the ellipse
yradius - Y radius for the ellipse

Remarks MGL_fillEllipseCoord() fills an ellipse given the centre and radii for the ellipse. The ellipse is filled in the current pen color and style.

Note: This routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and it provided for compatability with other graphics packages).

Return value True if ellipse was scan converted, false if an error occured (MGL_buf too small)..

See also MGL_ellipse(), MGL_ellipseCoord(), MGL_fillEllipse()

MGL_fillGouraudPolygon

Function Fills a convex polygon with smooth shading.

Syntax bool MGL_fillGouraudPolygon(int count,point *vArray,color_t *cArray,int xOffset,int yOffset);

Prototype in mgraph.h

Parameters count - Number of vertices in polygon
vArray - Array of vertices in polygon
cArray - Array of vertex colors
xOffset - X coordinate offset value
yOffset - Y coordinate offset value

Remarks MGL_fillGouraudPolygon() scan converts a filled *convex* polygon, while smoothly interpolating the color values between the vertices in the polygon. A *convex* polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right & left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

In order to ensure correct scan conversion of polygons with shared vertices, the right and bottom edges of the polygons are not scan converted. All vertices are offset by (xOffset,yOffset).

Note: You *must* call MGL_beginGouraud() before calling this routine, and you *must* call MGL_endGouraud() after drawing a set of one or more shaded polygons. You cannot draw anything else at the same time.

Return value True if the polygon was scan converted, false if an error occurred.

See also MGL_fillPolygon(), MGL_fillPolygonFast(), MGL_beginGouraud(), MGL_endGouraud().

MGL_fillPolygon

Function Fills an arbitrary polygon.

Syntax bool MGL_fillPolygon(int count,point *vArray,int xOffset,int yOffset);

Prototype in mgraph.h

Parameters
count - Number of vertices in polygon
vArray - Array of vertices in polygon
xOffset - X coordinate offset value
yOffset - Y coordinate offset value

Remarks MGL_fillPolygon() scan converts a filled arbitrary polygon in the current color and style. By default the routine will determine the type of the polygon being scan converted, and will scan convert *convex* polygons using a faster scan conversion routine, otherwise a general polygon scan conversion routine will be used. Thus you can scan convert any type of polygon that you desire.

A *convex* polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right & left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Non simple or self intersecting polygons will be scan converted using the standard in/out rule, where points are defined as being inside after crossing the first edge in the polygon, and then alternate between defined as inside then outside after crossing each active edge in the polygon.

In order to ensure correct scan conversion of polygons with shared vertices, the right and bottom edges of the polygons are not scan converted. All vertices are offset by (xOffset,yOffset).

You may also use the MGL_setPolygonType() routine to specify the type of polygons being scan converted. This may be AUTO_POLYGON, CONVEX_POLYGON or COMPLEX_POLYGON. Explicitly setting the polygon type will speed the drawing process.

Return value True if the polygon was scan converted, false if an error occurred.

See also MGL_fillPolygonFast(), MGL_fillGouraudPolygon, MGL_setPolygonType()

MGL_fillPolygonFast

Function Fills a convex polygon quickly.

Syntax bool MGL_fillPolygonFast(int count,point *vArray,int xOffset,int yOffset);

Prototype in mgraph.h

Parameters count - Number of vertices in polygon
vArray - Array of vertices in polygon
xOffset - X coordinate offset value
yOffset - Y coordinate offset value

Remarks MGL_fillPolygonFast() scan converts a filled *convex* polygon in the current color and style. A *convex* polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right & left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

This is the fastest method to render polygonal shapes such as triangles and quadrilaterals.

In order to ensure correct scan conversion of polygons with shared vertices, the right and bottom edges of the polygons are not scan converted. All vertices are offset by (xOffset,yOffset).

Note: You *must* call MGL_beginDrawing() before calling this routine, and you *must* call MGL_endDrawing() after scan converting one or more convex polygons. You may call other MGL routines at the same time however.

Return value True if the polygon was scan converted, false if an error occurred.

See also MGL_fillPolygon(), MGL_fillGouraudPolygon(), MGL_beginDrawing(), MGL_endDrawing().

MGL_fillRect

Function Draws a filled rectangle.

Syntax void MGL_fillRect(rect r);

Prototype in mgraph.h

Parameters r - Rectangle to fill

Remarks MGL_fillRect() fills a rectangle in the current current drawing attributes. The mathematical definition of a rectangle does not include the right and bottom endpoints, so effectively the right and bottom endpoints are not scan converted (solving problems with shared vertices).

See also MGL_fillRectCoord(), MGL_fillRectPt().

MGL_fillRectCoord

Function Draws a filled rectangle.

Syntax void MGL_fillRectCoord(int left,int top,int right,int bottom);

Prototype in mgraph.h

Parameters left - Left edge of rectangle
top - Top egde of rectangle
right - Right edge of rectangle
bottom - Bottom edge of rectangle

Remarks MGL_fillRectCoord() fills a rectangle defined by the coordinates of its edges in the current current drawing attributes. The mathematical definition of a rectangle does not include the right and bottom endpoints, so effectively the right and bottom endpoints are not scan converted (solving problems with shared vertices).

See also MGL_fillRect(), MGL_fillRectPt().

MGL_fillRectPt

Function Draws a filled rectangle defined by two points.

Syntax void MGL_fillRectPt(point lt,point rb);

Prototype in mgraph.h

Parameters lt - Point defining the left, top corner of the rectangle
rb - Point defining the right, bottom corner of rectangle.

Remarks MGL_fillRectPt() fills a rectangle defined by two points in the current current drawing attributes. The mathematical definition of a rectangle does not include the right and bottom endpoints, so effectively the right and bottom endpoints are not scan converted (solving problems with shared vertices).

See also MGL_fillRect(), MGL_fillRectCoord().

MGL_getActivePage

Function Returns the currently active hardware page.

Syntax void MGL_getActivePage(void);

Prototype in mgraph.h

Remarks MGL_getActivePage() returns the currently active hardware video page number. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages,

Return value Currently active hardware page.

See also MGL_setActivePage(), MGL_getVisualPage(), MGL_setVisualPage().

MGL_getArcCoords

Function Returns the starting and ending arc coordinates.

Syntax void MGL_getArcCoords(arc_coords *coords);

Prototype in mgraph.h

Parameters coords - Pointer to structure to store coordinates

Remarks MGL_getArcCoords() returns the centre coordinate, and starting and ending points on the ellipse that define the last elliptical arc to be rendered. You can then use these coordinate values to draw a line from the centre of the ellipse to the starting and ending points to complete the outline of the elliptical wedge.

The coordinates are returned in the arc_coords structure which is defined as:

```
typedef struct {
   int   x,y;
   int   startX,startY;
   int   endX,endY;
   } arc_coords;
```

where (x,y) defines the centre of the ellipse, and (startX,startY) defines the starting point on the ellipse and (endX,endY) defines the ending point on the ellipse.

Note: This routine must be called *immediately* after calling the MGL_ellipseArc() routines.

MGL_getAspectRatio

Function Returns the current video mode aspect ratio.

Syntax int MGL_getAspectRatio(void);

Prototype in mgraph.h

Remarks MGL_getAspectRatio() returns the aspect ratio of the currently active output device's physical pixels. This ratio is equal to:

$$\frac{\text{pixel x size}}{\text{pixel y size}} \times 1000$$

The device aspect ratio can be used to display circles and squares on the display device by approximating them with ellipses and rectangles of the appropriate dimensions. Thus in order to determine the number of pixels in the y direction for a square with 100 pixels in the x direction, we can simply use the formula:

y_pixels = ((long)x_pixels * 1000) / aspectratio

Note the cast to a long to avoid arithmetic overflow, as the apsect ratio is returned as an integer value with 1000 being a 1:1 aspect ratio.

Return value Current video mode aspect ratio * 1000.

See also MGL_setAspectRatio()

MGL_getAttributes

Function Returns a copy of the current attribute list

Syntax void MGL_getAttributes(attributes *attr);

Prototype in mgraph.h

Parameters attr - Pointer to structure to store attribute values in

Remarks MGL_getAttributes() returns a copy of the currently active attribute lists. You can use this routine to save the state of the MGL and later restore this state with the MGL_restoreAttributes() routine. The attribute values are returned in a structure define as:

```
typedef struct {
  color_t       color;
  color_t       backColor;
  int           markerSize;
  marker_style  markerStyle;
  color_t       markerColor;
  color_t       bdr_bright;
  color_t       bdr_dark;
  point         CP;
  write_mode    writeMode;
  fill_style    penStyle;
  int           penHeight;
  int           penWidth;
  pattern       penPat;
  pixpattern    penPixPat;
  rect          viewport;
  rect          clipRect;
  bool          clip;
  int           poly_type;
  text_settings tsettings;
  } attributes;
```

Note: This routine does not return any values to do with the current viewport stack. This will need to be maintained separately.

See also MGL_restoreAttributes()

MGL_getBackColor

Function Returns the current background color value.

Syntax color_t MGL_getBackColor(void);

Prototype in mgraph.h

Remarks MGL_getBackColor() returns the current background color value. The background color value is used to clear the display and viewport with the MGL_clearDevice() and MGL_clearViewport() routines, and is also used for filling solid primitives in the BITMAP_PATTERN_OPAQUE fill mode.

Return value Current background color value.

See also MGL_setBackColor(), MGL_getColor(), MGL_setColor()

MGL_getBorderColors

Function Returns the current border color values.

Syntax void MGL_getBorderColors(color_t *bright,color_t *dark);

Prototype in mgraph.h

Parameters bright - Place to store bright color value
dark - Place to store dark color value

Remarks MGL_getBorderColors() returns the currently active border colors values. There are two border color defined by the MGL, the *bright* border color value and the *dark* border color value. These values are used by the MGL_drawBorder(), MGL_drawHDivider() and MGL_drawVDivider() routines to determine the colors to draw the psuedo 3D borders in. These values will be set automatically for you by default when the MGL is initialised, but you will need to change these values if you modify the palette.

See also MGL_setBorderColors(), MGL_drawBorder(), MGL_drawHDivider(), MGL_drawVDivider()

MGL_getCharMetrics

Function Computes the metrics for a specific character.

Syntax void MGL_getCharMetrics(char ch,metrics *metrics);

Prototype in mgraph.h

Parameters metrics - Place to store the resulting metrics.

Remarks MGL_getCharMetrics() computes the character metrics for a specific character. The character metrics define specific characters width, height, asecent, descent and other values. These values can then be used to correctly position the character with pixel precise positioning. The metrics are computed and returned in the metrics structure defined as:

```
typedef struct {
   int      width;
   int      fontWidth;
   int      fontHeight;
   int      ascent;
   int      descent;
   int      leading;
   int      kern;
   } metrics;
```

where width defines the actual width in pixels for the character, fontWidth defines the width of the character including any extra padding between it and the next character and kern defines the actual kern value for the specific character. All the other values will be the same as those for the entire font file.

All values are defined in pixels and will be as accurate as possible given the current fonts scaling factor (currently only vector fonts can be scaled).

See also MGL_getFontMetrics()

MGL_getClipMode

Function Returns the current clipping mode.

Syntax bool MGL_getClipMode(void);

Prototype in mgraph.h

Remarks MGL_getClipMode() returns the current clipping mode. You can selectively turn clipping on and off for the MGL, in order to speed up some operations. Clipping is turned on by default, and generally you will want to leave clipping enabled.

Return value True if clipping is on, false if not.

See also MGL_setClipMode(), MGL_getClipRect(), MGL_setClipRect()

MGL_getClipRect

Function Returns the current clipping rectangle.

Syntax void MGL_getClipRect(rect *clip);

Prototype in mgraph.h

Parameters clip - Place to store the current clipping rectangle

Remarks MGL_getClipRect() returns the current clipping rectangle coordinates. The current clipping rectangle is used to clip all output, and is always defined as being relative to the currently active viewport The clipping rectangle can be no larger than the currently active viewport.

See also MGL_setClipRect, MGL_getClipMode(), MGL_getClipMode().

MGL_getColor

Function Returns the current foreground color.

Syntax color_t MGL_getColor(void);

Prototype in mgraph.h

Remarks MGL_getColor() returns the current foreground color values. The foreground color value is used to draw all primitives.

Return value Current foreground color.

See also MGL_setColor(), MGL_getBackColor(), MGL_setBackColor()

MGL_getCP

Function Returns the current position value.

Syntax void MGL_getCP(point* CP);

Prototype in mgraph.h

Parameters CP - place to store the current position

Remarks MGL_getCP() returns the current position (CP). The CP is the current graphics cursor position, and is used by a number of routines to determine where to being drawing output. You can use the MGL_moveTo() routine to directly move the CP to a new position.

See also MGL_moveTo(), MGL_moveRel(), MGL_lineTo(), MGL_lineRel(), MGL_drawStr().

MGL_getCursorColor

Function Returns the current mouse cursor color.

Syntax color_t MGL_getCursorColor(void);

Prototype in mgraph.h

Remarks MGL_getCursorColor() returns the currently active mouse cursor color. The mouse cursor color is used to determine what foreground color is used to draw the mouse cursor in.

Return value Current mouse cursor color value.

See also MGL_setCursorColor()

MGL_getDefaultPalette

Function Returns the default palette for video mode.

Syntax void MGL_getDefaultPalette(palette *pal);

Prototype in mgraph.h

Parameters pal - Place to store the default palette values.

Remarks MGL_getDefaultPalette() returns a copy of the default palette for the current video mode. This can be used to reset the palette to the original default values that the palette is programmed with when the MGL is initialised.

The palette values are manipulated as arrays of the *palette* structure, defined as:

```
typedef struct {
  uchar  red;
  uchar  green;
  uchar  blue;
  } palette;
```

where red, green and blue define the color components for that entry. Each color component is defined as an 8 bit value with a range of 0-255.

See also MGL_setPalette(), MGL_getPalette()

MGL_getDivot

Function Saves a divot of video memory in system RAM

Syntax void MGL_getDivot(rect r,void *divot)

Prototype in mgraph.h

Parameters r - Rectangle defining the divot to save
divot - Pointer to area to store video memory in

Remarks MGL_getDivot() copies a block of video memory from the current video page into system RAM. A *divot* is defined as being a rectangular area of video memory that you wish to save. However when the divot is saved, it may also save other data slightly outside of the specified area in order to increase the speed at which the divot can be saved and restored. Thus the divot routines save and restore the video memory in the fastest possible manner. This can be used to store the video memory behind pull down menus etc.

You must pre-allocate enough space to hold the entire divot in system RAM. Use the MGL_divotSize() routine to determine the size of the memory block required to store the divot.

See also MGL_putDivot(), MGL_divotSize()

MGL_getDivotCoord

Function Saves a divot of video memory into system RAM

Syntax void MGL_getDivotCoord(int left,int top,int right,int bottom,void *divot);

Prototype in mgraph.h

Parameters left - Left edge of area to save
top - Top egde of area to save
right - Right edge of area to save
bottom - Bottom edge of area to save
divot - Pointer to area to store the video memory in

Remarks MGL_getDivotCoord() copies a block of video memory from the current video page into system RAM. A *divot* is defined as being a rectangular area of video memory that you wish to save. However when the divot is saved, it may also save other data slightly outside of the specified area in order to increase the speed at which the divot can be saved and restored. Thus the divot routines save and restore the video memory in the fastest possible manner. This can be used to store the video memory behind pull down menus etc.

You must pre-allocate enough space to hold the entire divot in system RAM. Use the MGL_divotSize() routine to determine the size of the memory block required to store the divot.

See also MGL_getDivot(), MGL_putDivot(), MGL_divotSize()

MGL_getDriver

Function Returns the current video device driver id

Syntax int MGL_getDriver(void);

Prototype in mgraph.h

Remarks MGL_getDriver() returns the numerical id of the currently active video device driver. This value can be converted in a printable representation with the MGL_driverName() routine.

Return value Current video device driver id.

See also MGL_driverName(), MGL_getMode()

MGL_getFont

Function Returns the currently active font.

Syntax font *MGL_getFont(void);

Prototype in mgraph.h

Remarks MGL_getFont() returns a pointer to the currently active font. The currently active font is used to perform all text output by the MGL.

Return value Pointer to currently active font.

See also MGL_useFont(), MGL_loadFont(), MGL_unloadFont()

MGL_getFontMetrics

Function Returns the currently active font metrics.

Syntax void MGL_getFontMetrics(metrics *metrics);

Prototype in mgraph.h

Parameters metrics - Place to store the font metrics.

Remarks MGL_getFontMetrics() computes the font metrics for the current font. The metrics are computed and returned in the metrics structure defined as:

```
typedef struct {
  int       width;
  int       fontWidth;
  int       fontHeight;
  int       ascent;
  int       descent;
  int       leading;
  int       kern;
  } metrics;
```

where width defines the maximum physical width of all characters in the font, fontWidth defines the maximum width of all the characters in the font including any extra padding between each character. The fontWidth value of each character is used to advance the CP to the start of the next character, and can be larger than the actual character width (in order to put space between the characters). The ascent and descent values define the values from the baseline of the font to the highest and lowest points of all characters. The leading value defines the number of pixels between each line of text, and the kern value defines the maximum kerning value for the font (how far certain characters can extend behind the character start position).

All values are defined in pixels and will be as accurate as possible given the current fonts scaling factor (currently only vector fonts can be scaled).

See also MGL_getCharMetrics().

MGL_getImage

Function Copies of block of video memory to system RAM.

Syntax void MGL_getImage(rect r,void *image)

Prototype in mgraph.h

Parameters r - Rectangle defining the image to save
image - Place to store the image data.

Remarks MGL_getImage() copies of block of video memory into a system RAM buffer. The image can then be modified in system RAM, and restored to video RAM in another location. MGL_getImage() is pixel exact so it only copies the pixels that you specify, so it may be slower than the MGL_getDivot() routine.

You must pre-allocate enough space to hold the entire image in system RAM. Use the MGL_imageSize() routine to determine the size of the memory block required to store the image.

See also MGL_getImageCoord(), MGL_putImage(), MGL_putImageCoord(), MGL_imageSize().

MGL_getImageCoord

Function Copies a block of video memory to system RAM.

Syntax void MGL_getImageCoord(int left,int top,int right,int bottom,void *image);

Prototype in mgraph.h

Parameters left - Left edge of area to save
top - Top egde of area to save
right - Right edge of area to save
bottom - Bottom edge of area to save
image - Pointer to area to store the video memory in

Remarks MGL_getImageCoord copies of block of video memory into a system RAM buffer. The image can then be modified in system RAM, and restored to system RAM in another location. MGL_getImageCoord is pixel exact so it only copies the pixels that you specify, so it may be slower than the MGL_getDivot() routine.

You must pre-allocate enough space to hold the entire image in system RAM. Use the MGL_imageSize() routine to determine the size of the memory block required to store the image.

See also MGL_getImage(), MGL_putImage(), MGL_putImageCoord(), MGL_imageSize()

MGL_getMarkerColor

Function Returns the current marker color value.

Syntax color_t MGL_getMarkerColor(void);

Prototype in mgraph.h

Remarks MGL_getMarkerColor returns the current marker color value. The marker color is used when drawing markers with the MGL_marker() routine.

Return value Current marker color value.

See also MGL_setMarkerColor(), MGL_marker(), MGL_polyMarker()

MGL_getMarkerSize

Function Returns the current marker size value

Syntax int MGL_getMarkerSize(void);

Prototype in mgraph.h

Remarks MGL_getMarkerSize() returns the current marker size. The marker size is used to determine how big to draw the markers that are drawn with the MGL_marker() routine. The size is defined as the dimension from the middle of the marker to the edges, so the actual dimensions of the marker will be twice the maker size plus 1. If marker size of 1 will define a marker that is contained within a rectangle 3 pixels wide.

Return value Current marker size

See also MGL_setMarkerSize(), MGL_setMarkerStyle(), MGL_getMarkerStyle(), MGL_marker(), MGL_polyMarker()

MGL_getMarkerStyle

Function Returns the current marker style.

Syntax int MGL_getMarkerStyle(void);

Prototype in mgraph.h

Remarks MGL_getMarkerStyle() returns the current marker style value. The marker style defines the type of marker to be rendered. Currently the MGL defines the following markers:

MARKER_SQUARE - A solid square
MARKER_CIRCLE - A solid circle
MARKER_X - A cross made of two lines

Return value Current marker style value.

See also MGL_setMarkerSize(), MGL_getMarkerSize(), MGL_setMarkerStyle(), MGL_marker(), MGL_polyMarker()

MGL_getMode

Function Returns the current video mode number.

Syntax int MGL_getMode(void);

Prototype in mgraph.h

Remarks MGL_getMode() returns the currently active video mode number. This number can be converted to a printable form using the MGL_modeName() routine.

Return value Current video mode number.

See also MGL_init(), MGL_modeName()

MGL_getPalette

Function Obtains the currently active palette values.

Syntax void MGL_getPalette(palette *pal,int numColors,int startIndex);

Prototype in mgraph.h

Parameters pal - Place to store the retrieved values
numColors - Number of color values to retrieve
startIndex - Starting palette index value to retrieve

Remarks MGL_getPalette() obtains part or all of the currently active palette values and stores it in the array *pal*. You can specify only a subset of the palette values to be obtained with the startIndex and numColors arguments.

Thus to save the entire palette in a 256 color video mode, you would use (assuming enough space for the palette has been allocated):

```
MGL_getPalette(pal,255,0);
```

or to get the top half of the palette you would use:

```
MGL_getPalette(pal,128,128);
```

You should ensure that you have allocated enough memory to hold all of the palette values that you wish to read. You can use MGL_getPaletteSize() to determine the size required to save the entire palette.

See also MGL_getPaletteEntry(), MGL_setPalette(), MGL_getDefaultPalette()

MGL_getPaletteEntry

Function Returns the value of a single palette entry

Syntax void MGL_getPaletteEntry(int entry,uchar *red,uchar *green,uchar *blue);

Prototype in mgraph.h

Parameters entry - Palette index to read
red - Place to store the red component
green - Place to store the green component
blue - Place to store the green component

Remarks MGL_getPaletteEntry() obtains the color values of a single palette entry. If you wish to obtain more than a single palette index you should use the MGL_getPalette() routine which is faster for multiple entries.

See also MGL_setPaletteEntry(), MGL_getPalette(), MGL_setPalette()

MGL_getPaletteSize

Function Returns the number of entries in the entire palette

Syntax int MGL_getPaletteSize(void);

Prototype in mgraph.h

Remarks MGL_getPaletteSize() returns the number of entries in the entire palette. You should use this routine to determine the size of the entire palette, as the palette is still active even in HiColor and TrueColor video modes (it is implemented in software rather than hardware in these modes).

Return value Number of entries in entire palette.

See also MGL_getPalette()

MGL_getPaletteSnowLevel

Function Returns the current palette snow level

Syntax int MGL_getPaletteSnowLevel(void);

Prototype in mgraph.h

Remarks MGL_getPaletteSnowLevel() returns the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. The MGL uses a reasonable default of 100 entries per retrace, but you may want to modify this on faster or slower machines (this should be a user option).

Return value Current palette snow level.

See also MGL_setPaletteSnowLevel()

MGL_getPenBitmapPattern

Function Returns the currently active bitmap pattern

Syntax void MGL_getPenBitmapPattern(pattern *pat);

Prototype in mgraph.h

Parameters pat - Place to store the bitmap pattern

Remarks MGL_getPenBitmapPattern() returns a copy of the currently active bitmap pattern used when rendering patterned primitive in the BITMAP_PATTERN_TRANSPARENT and BITMAP_PATTERN_OPQAUE pen styles. A bitmap pattern is defined as an 8 x 8 pixel bitmap pattern stored as an array of 8 bytes.

When filling in the BITMAP_PATTERN_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the BITMAP_PATTERN_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0.

See also MGL_setPenBitmapPattern(), MGL_setPenStyle(), MGL_getPenStyle()

MGL_getPenSize

Function Returns the current pen size.

Syntax void MGL_getPenSize(int *height,int *width);

Prototype in mgraph.h

Parameters height - Place to store the current pen height
width - Place to store the current pen width

Remarks MGL_getPenSize() return the size of the current pen in pixels. The default pen is 1 pixel by 1 pixel in dimensions, however you can change this to whatever value you like. When primitive are rendered with a pen other than the default, the pixels in the pen always lie to the right and below the current pen position.

See also MGL_setPenSize()

MGL_getPenStyle

Function Returns the current pen style.

Syntax int MGL_getPenStyle(void);

Prototype in mgraph.h

Remarks MGL_getPenStyle() returns the currently active pen style. The MGL supports the following pen styles:

SOLID_PATTERN - Fill with solid color
BITMAP_PATTERN_OPAQUE - Pattern fill
BITMAP_PATTERN_TRANSPARENT - Transparent pattern fill

When filling in the BITMAP_PATTERN_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the BITMAP_PATTERN_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0.

Return value Current pen style.

See also MGL_setPenStyle(), MGL_setPenBitmapPattern()

MGL_getPixel

Function Returns the color of a specified pixel

Syntax color_t MGL_getPixel(point p)

Prototype in mgraph.h

Parameters p - Point defining the pixel to read

Remarks MGL_getPixel() returns the color of the pixel at the coordinates defined by the point *p*.

Note: You *must* ensure that you call the routine MGL_beginPixel() before reading any pixel values and the routine MGL_endPixel() after reading a bunch of pixels.

Return value Color of the specified pixel.

See also MGL_pixel(), MGL_getPixelCoord(), MGL_beginPixel(), MGL_endPixel()

MGL_getPixelCoord

Function Returns the color of a specified pixel

Syntax color_t MGL_getPixelCoord(int x,int y);

Prototype in mgraph.h

Parameters x,y - Coordinates of the pixel to read.

Remarks MGL_getPixelCoord() returns the color of the pixel at the coordinates defined by the point *(x,y)*.

Note: You *must* ensure that you call the routine MGL_beginPixel() before reading any pixel values and the routine MGL_endPixel() after reading a bunch of pixels.

Return value Color of the specified pixel.

See also MGL_pixel(), MGL_getPixel(), MGL_beginPixel(), MGL_endPixel()

MGL_getPixelFormat

Function Returns the current packed pixel format information.

Syntax void MGL_getPixelFormat(pixel_format *format);

Prototype in mgraph.h

Parameters format - Place to store the pixel format information.

Remarks MGL_getPixelFormat() returns the current pixel format information for the currently active video mode. This information is used by the MGL to encode the packed pixel information, and can be used by your application to work out how to pack values correctly for the direct color video modes. This routine should also be used to correctly interpret the format of the pixel information returned by the MGL_getImage() routine. The pixel format structure is defined as follows:

```
typedef struct {
  char   redMask,blueMask;
  char   greenMask,rsvdMask;
  int    redPos,redAdjust;
  int    greenPos,greenAdjust;
  int    bluePos,blueAdjust;
  int    rsvdPos,rsvdAdjust;
  } pixel_format;
```

The Mask values are logical AND masks that can be used to represent the correct number of usable bits in each color value. The Pos values describe where in the packed pixel format each value resides, and the Adjust values descibe the bit shift required to adjust to 8 bit component values. The following code illustrates how these values can be used to unpack values to and from 8 bit RGB tuples:

```
R = (((color) >> redPos) & redMask) << redAdjust;
G = (((color) >> greenPos) & greenMask) << greenAdjust;
B = (((color) >> bluePos) & blueMask) << blueAdjust;
```

and to pack values:

```
color = ((color_t)((R >> redAdjust) & redMask) << redPos)
 | ((color_t)((G >> greenAdjust) & greenMask) << greenPos)
 | ((color_t)((B >> blueAdjust) & blueMask) << bluePos);
```

See also MGL_packColor(), MGL_unpackColor()

MGL_getPolygonType

Function Returns the current polygon type.

Syntax int MGL_getPolygonType(void);

Prototype in mgraph.h

Remarks MGL_getPolygonType() returns the current polygon type. You can change this value with the MGL_setPolygonType() to force the MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). The MGL supports the following polygon types:

AUTO_POLYGON - MGL automatically determines type
CONVEX_POLYGON - All polygons rendered as convex
COMPLEX_POLYGON - All polygons rendered as complex

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rendering.

Return value Current polygon type code.

See also MGL_setPolygonType(), MGL_fillPolygon()

MGL_getSpaceExtra

Function Returns the current space extra value.

Syntax int MGL_getSpaceExtra(void);

Prototype in mgraph.h

Remarks MGL_getSpaceExtra() returns the current space extra value used when drawing text in the current font. The space extra value is normally zero, but can be a positive or negative value. This value can be used to insert extra space between the characters in a font (making this value a large negative value will make the characters run on top of each other).

Return value Current space extra value.

See also MGL_setSpaceExtra(), MGL_drawStr()

MGL_getTextDirection

Function Returns the current text direction

Syntax int MGL_getTextDirection(void);

Prototype in mgraph.h

Remarks MGL_getTextDirection() returns the current text direction. The MGL supports the following text directions:

LEFT_DIR - Text runs to the left (right to left)
UP_DIR - Text runs in the up direction
RIGHT_DIR - Text runs to the right (left to right)
DOWN_DIR - Text runs in the down direction

Currently the MGL only supports directional text with the default 8x8 bitmap font and vector fonts. Bitmap fonts can only be drawn in the RIGHT_DIR direction.

Return value Current text direction.

See also MGL_setTextDirection(), MGL_drawStr()

MGL_getTextJustify

Function Returns the current text justification.

Syntax void MGL_getTextJustify(int *horiz,int *vert);

Prototype in mgraph.h

Parameters horiz - Place to store horizontal justification
vert - Place to store vertical justification

Remarks MGL_getTextJustify() returns the current text justification values. The MGL support the following horizontal justification types:

LEFT_TEXT - Text is left justified
CENTER_TEXT - Text is centered left to right
RIGHT_TEXT - Text is right justified

and the following vertical justification types:

TOP_TEXT - Text is top justified
CENTER_TEXT - Text is centered top to bottom
BASELINE_TEXT - Text is justified to the baseline
BOTTOM_TEXT - Text is bottom justified

See also MGL_setTextJustify()

MGL_getTextSettings

Function Returns the current text settings.

Syntax void MGL_getTextSettings(text_settings *settings);

Prototype in mgraph.h

Parameters settings - Place to store the current text settings

Remarks MGL_getTextSettings() returns the currently active text settings. This routine provides a way to save and restore all the values relating to the rendering of text in the MGL with a single function call. The text settings values are stored in the following structure:

```
typedef struct {
   int      horiz_just;
   int      vert_just;
   int      dir;
   int      sz_numerx;
   int      sz_numery;
   int      sz_denomx;
   int      sz_denomy;
   int      space_extra;
   font     *fnt;
```

where *horiz_just* and *vert_just* define the horizontal and vertical justification values, *dir* defines the current text direction, *space_extra* defines the current space extra value and *fnt* define the currently active font (stored in system memory). The *sz_numerx, sz_numery, sz_demonx* and *sz_denomy* specify the current text scaling factors.

See also MGL_setTextSettings

MGL_getTextSize

Function Returns the current text scaling factors

Syntax void MGL_getTextSize(int *numerx,int *denomx,int *numery,int *denomy);

Prototype in mgraph.h

Parameters numerx - Place to store the x numerator value
denomx - Place to store the x denominator value
numery - Place to store the y numerator value
denomy - Place to store the y denominator value

Remarks MGL_getTextSize() returns the current text scaling factors used by the MGL. The text size values define an integer scaling factor to be used, where the actual values will be computed using the following formula:

$$\text{scaled} = \frac{\text{unscaled} \times \textit{numer}}{\textit{denom}}$$

Note: Currently the MGL can only scale vectors fonts.

See also MGL_setTextSize()

MGL_getViewport

Function Returns the currently active viewport

Syntax void MGL_getViewport(rect *view);

Prototype in mgraph.h

Parameters view - Place to store the current viewport

Remarks MGL_getViewport() returns the dimensions of the currently active viewport. These dimensions are global to the entire display area used by the currently active video device driver.

All output in the MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the video display.

See also MGL_setViewport(), MGL_setRelViewport(), MGL_clearViewport(), MGL_setClipRect()

MGL_getVisualPage

Function Returns the currently visible hardware video page.

Syntax int MGL_getVisualPage(void);

Prototype in mgraph.h

Remarks MGL_getVisualPage() returns the currently visible hardware video page number. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages.

Return value Currently visible hardware video page.

See also MGL_setVisualPage(), MGL_getActivePage(), MGL_setActivePage().

MGL_getWriteMode

Function Returns the current write mode operation

Syntax int MGL_getWriteMode(void);

Prototype in mgraph.h

Remarks MGL_getWriteMode() returns the currently active write mode. The MGL supports the following write mode operations for all output primitives:

REPLACE_MODE - Replace the original pixels
AND_MODE - Logical AND with original pixels
OR_MODE - Logical OR with original pixels
XOR_MODE - Logical XOR with original pixels

Return value Current write mode operation.

See also MGL_setWriteMode()

MGL_getX

Function Returns the X coordinate of the current position.

Syntax int MGL_getX(void);

Prototype in mgraph.h

Remarks MGL_getX() returns the X coordinate of the current position (CP) value. The CP is the current graphics cursor position, and is used by a number of output routines to determine where to being drawing.

Return value X coordinate of current position

See also MGL_getY(), MGL_getCP(), MGL_moveTo()

MGL_getY

Function Returns the Y coordinate of the current position.

Syntax int MGL_getY(void);

Prototype in mgraph.h

Remarks MGL_getY() returns the Y coordinate of the current position (CP) value. The CP is the current graphics cursor position, and is used by a number of output routines to determine where to being drawing.

Return value Y coordinate of current position

See also MGL_getX(), MGL_getCP(), MGL_moveTo()

MGL_globalToLocal

Function Converts a point from global coordinates to local coordinates

Syntax void MGL_globalToLocal(point *p);

Prototype in mgraph.h

Parameters p - Pointer to point to be converted

Remarks MGL_globalToLocal() converts a coordinate from global coordinates to local coordinates. Global coordinates are defined relative to the entire output devices display, while local coordinates are relative to the currently active viewport.

This routine is usually used to convert mouse coordinate values from global screen coordinates to the local coordinate system of the currently active viewport.

See also MGL_localToGlobal()

MGL_gouraudScanLine

Function Draws a smooth shaded scanline.

Syntax void MGL_gouraudScanLine(int y,int x1,int x2,color_t c1,color_t c2);

Prototype in mgraph.h

Parameters y - Y coordinate of scanline
x1 - Starting X coordinate of scanline
x2 - Ending X coordinate of scanline
c1 - Starting color value
c2 - Ending color value

Remarks MGL_gouraudScanLine() draws a smooth shaded scanline between the point *(x1,y)* and the point *(x2,y)*. The color values are smoothly interpolated across the scanline starting with *c1* and ending with *c2*.

Note: You *must* call MGL_beginGouraud() before calling this routine to ensure that the hardware is in the correct state to draws shaded scanlines. You *must* also ensure that you call MGL_endGouraud() after rendering a set of shaded scanlines. You may not call any other MGL routines while you are rendering shaded scanlines.

See also MGL_fillGouraudPolygon(). MGL_beginGouraud(), MGL_endGouraud()

MGL_imageSize

Function Returns the number of bytes required to store an image.

Syntax long MGL_imageSize(rect r);

Prototype in mgraph.h

Parameters r - Rectangle defining the size of the image

Remarks MGL_imageSize() returns the number of bytes required to store an offscreen video memory image. You must use this routine to allocate enough memory to store the image *before* you call the MGL_getImage() routine to read an image from display memory.

Return value Number of bytes required to store the image.

See also MGL_imageSizeCoord(), MGL_getImage(), MGL_getImageCoord(), MGL_putImage(), MGL_putImageCoord().

See also MGL_fillGouraudPolygon(). MGL_beginGouraud(), MGL_endGouraud()

MGL_imageSizeCoord

Function Returns the number of bytes required to store an image.

Syntax long MGL_imageSizeCoord(int left,int top,int right,int bottom);

Prototype in mgraph.h

Parameters left - Left edge of area to save
top - Top egde of area to save
right - Right edge of area to save
bottom - Bottom edge of area to save

Remarks MGL_imageSize() returns the number of bytes required to store an offscreen video memory image. You must use this routine to allocate enough memory to store the image *before* you call the MGL_getImage() routine to read an image from display memory.

Return value Number of bytes required to store the image.

See also MGL_imageSize(), MGL_getImage(), MGL_getImageCoord(), MGL_putImage(), MGL_putImageCoord().

MGL_init

Function Initialises the MGL.

Syntax void MGL_init(int *graphdriver,int *graphmode,const char *pathtodriver);

Prototype in mgraph.h

Parameters graphdriver - Place to store detected graphics device driver id
graphmode - Place to store suggested graphics mode id
pathtodriver - Path to MGL driver files

Remarks

MGL_init() initialises the MGL, sets the specified video mode and clears the display device ready to accept output. If you pass the value grDETECT in the *graphdriver* parameter, MGL_init() will automatically call the MGL_detectGraph() routine to detect the installed video device driver, and initialise itself accordingly. The video mode used in this case will be the one suggested by the installed video device driver, and will be returned in the *graphmode* parameter. If a suitable device is not found for use by the MGL, it will return an error via the MGL_result() routine and pass back the value grNONE in the *graphdriver* parameter.

If you wish to start a video mode other than the default one suggested by the video device driver, you should call the MGL_detectGraph() routine yourself, and pass the value returned for *graphdriver* and your selected video mode in *graphmode* to initialise the mode.

The video device drivers currently supported by the MGL are (this is subject to change - please consult the MGRAPH.H header file for the latest information):

grDETECT - Auto detect the graphics subsystem
grNONE - No graphics hardware detected
grEGA - Standard EGA with 256k RAM
grVGA - Standard VGA
grSVGA - VESA VBE compliant Super VGA
grSVGA_S3 - S3 accelerated SuperVGA

The MGL supports a number of different video mode resolutions, ranging from 320x200 up to 1280x1024 with color ranges from 16 colors up to 16.7 million colors. The name for each video mode can be constructed from the pixel and color resolutions, so the video mode id for 320x200 16 color EGA graphics mode is grEGA_320x200x16, and the 1280x1024 16.7 million color mode is grSVGA_1280x1024x16m. Please consult the MGRAPH.H header file for a full list of defined video modes (subject to change over time).

The *pathtodriver* variable is used by the MGL to locate the video device driver files, and as a starting location when looking for the MGL font files. The MGL will first look in the directory specified by *pathtodriver* (it may be a relative pathname or an absolute pathname) for the files. If it cannot find the device driver files in this directory, it will then look for them in the current directory for the files.

You may also link the video device driver files directly with your program, but you must first register the linked in driver files with the MGL_registerDriver() routine before calling MGL_init().

If anything went wrong duing the initialisation process, the MGL will return a result code via the MGL_result() routine. You can then use this result code to determine the cause of the problem, and use the MGL_errorMsg() routine to display an appropriate error message for the user.

See also

MGL_detectGraph(), MGL_setGraphMode(), MGL_restoreCRTMode(), MGL_result()

MGL_insetRect

Function Inset a rectangle by specified values.

Syntax void MGL_insetRect(rect r,int dx,int dy)

Prototype in mgraph.h

Parameters r - Rectangle to inset
dx - Value to inset the X coordinates by
dy - Value to inset the Y coordinates by

Remarks MGL_insetRect() insets the coordinates of a rectangle by the specified amount in each coordinate direction. The value of *dx* is added to the left coordinate and subtracted from the right coordinate, while the value of *dy* is added to the top coordinate and subtracted from the bottom coordinate. You may use negative values for *dx* and *dy* in order to increase the size of a rectangle rather than descrease it's size.

See also MGL_offsetRect(), MGL_disjoingRect(), MGL_ptInRect()

MGL_line

Function Draws a line.

Syntax void MGL_line(point p1,point p2);

Prototype in mgraph.h

Parameters p1 - Starting point for the line
p2 - Ending point for the line

Remarks MGL_line() draws a line starting at the point *p1* and ending at the point *p2* in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping in on.

See also MGL_lineCoord(), MGL_lineFast(), MGL_lineCoordFast(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_lineCoord

Function Draws a line.

Syntax void MGL_lineCoord(int x1,int y1,int x2,int y2);

Prototype in mgraph.h

Parameters x1,y1 - Starting point for the line
x2,y2 - Ending point for the line

Remarks MGL_lineCoord() draws a line starting at the point *(x1,y1)* and ending at the point *(x2,y2)* in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping in on.

See also MGL_lineCoord(), MGL_lineFast(), MGL_lineCoordFast(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_lineCoordFast

Function Draws a line as fast as possible.

Syntax void MGL_lineCoordFast(int x1,int y1,int x2,int y2);

Prototype in mgraph.h

Parameters x1,y1 - Starting point for the line
x2,y2 - Ending point for the line

Remarks MGL_lineCoordFast() draws a line starting at the point *(x1,y1)* and ending at the point *(x2,y2)* in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping in on.

Note: You *must* call the routine MGL_beginDrawing() *before* calling the MGL_lineCoordFast() routine to put the hardware into the correct state fo for high speed line drawing. The routines MGL_line() and MGL_lineCoord() do this once for every line to be drawn, so you can save time by calling MGL_beginDrawing() once before drawing a whole set of lines.

See also MGL_lineCoord(), MGL_lineFast(), MGL_lineCoordFast(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord(), MGL_beginDrawing(), MGL_endDrawing().

MGL_lineEngine

Function Generates the set of points on a line.

Syntax void MGL_lineEngine(int x1,int y1,int x2,int y2,void (*plotPoint)(int x,int y));

Prototype in mgraph.h

Parameters x1,y1 - Starting point for the line
x2,y2 - Ending point for the line
plotPoint - User supplied pixel plotting routine.

Remarks MGL_lineEngine() generates the set of points on a line, and calls a user supplied *plotPoint* routine for every point generated. This is the same routine used for rendering lines internally in the MGL.

The use supplied *plotPoint* routine will be called once for every point on the line. The set of points generated will always be in the same order for any two endpoints, no matter which order the endpoints are given.

See also MGL_ellipseEngine().

MGL_lineFast

Function Draws a line as fast as possible.

Syntax void MGL_lineFast(point p1,point p2);

Prototype in mgraph.h

Parameters p1 - Starting point for the line
p2 - Ending point for the line

Remarks MGL_lineFast() draws a line starting at the point *p1* and ending at the point *p2* in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping in on.

Note: You *must* call the routine MGL_beginDrawing() *before* calling the MGL_lineFast() routine to put the hardware into the correct state fo for high speed line drawing. The routines MGL_line() and MGL_lineCoord() do this once for every line to be drawn, so you can save time by calling MGL_beginDrawing() once before drawing a whole set of lines.

See also MGL_lineCoord(), MGL_lineFast(), MGL_lineCoordFast(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord(), MGL_beginDrawing(), MGL_endDrawing().

MGL_lineRel

Function Draws a relative line from the CP.

Syntax void MGL_lineRel(point p)

Prototype in mgraph.h

Parameters p - Relative point to draw line to.

Remarks MGL_lineRel() draws a line from the current position (CP) to the relative location that is a distance of *p* away from the CP. Thus the location of the next point on the line is *(CP.x + p.x, CP.y + p.y)*. The CP is updated to this value.

See also MGL_moveTo(), MGL_moveRel(), MGL_lineTo(), MGL_lineRel().

MGL_lineRelCoord

Function Draws a relative line.

Syntax void MGL_lineRelCoord(int dx,int dy);

Prototype in mgraph.h

Parameters dx,dy - Relative point to draw line to.

Remarks MGL_lineRelCoord() draws a line from the current position (CP) to the relative location that is a distance of *(dx,dy)* away from the CP. Thus the location of the next point on the line is *(CP.x + dx, CP.y + dy)*. The CP is updated to this value.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_lineTo

Function Draws a line from the CP to the specified point.

Syntax void MGL_lineTo(point p);

Prototype in mgraph.h

Parameters p - Point to draw the line to.

Remarks MGL_lineTo() draws a line from the current position (CP) to the new point*p*. The CP is set to the point *p* on return from this routine.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_lineToCoord

Function Draws a line from the CP to the specified point.

Syntax void MGL_lineToCoord(int x,int y);

Prototype in mgraph.h

Parameters x,y - Point to draw the line to.

Remarks MGL_lineToCoord() draws a line from the current position (CP) to the new point *(x,y)*. The CP is set to the point *(x,y)* on return from this routine.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_loadFont

Function Loads an MGL font file for use.

Syntax font *MGL_loadFont(const char *fontname);

Prototype in mgraph.h

Parameters fontname - Name of the font file to load

Remarks MGL_loadFont() attempts to locate and read in the font specified by *fontname*. The parameter *fontname* is the name of the actual MGL font file on disk, and may include a relative pathname component. MGL_loadFont() first looks for the font file relative to the directory where it found the MGL driver files at initialisation time, and will then look for the font file relative to the current directory if the fontfile was not found.

If the font file was not found, or was not successfully loaded, MGL_loadFont returns NULL, and sets the MGL_result() error code.

Return value Pointer to the loaded font file (NULL if an error occurred).

See also MGL_unloadFont(), MGL_useFont(), MGL_availableFont().

MGL_localToGlobal

Function Converts a point from local coordinates to global coordinates

Syntax void MGL_localToGlobal(point *p);

Prototype in mgraph.h

Parameters p - Pointer to point to be converted

Remarks MGL_localToGlobal() converts a coordinate from local coordinates to global coordinates. Global coordinates are defined relative to the entire output devices display, while local coordinates are relative to the currently active viewport.

See also MGL_globalToLocal()

MGL_marker

Function Draws a marker at the specified coordinate.

Syntax void MGL_marker(point p);

Prototype in mgraph.h

Parameters p - Coordinate to draw the marker at

Remarks MGL_marker() draws a marker in the current marker color, style and size at the specified location. Markers can be used to label the vertices in graphs. Refer to the MGL_setMarkerStyle() routine for a list of the typs of markers supported.

See also MGL_setMarkerSize(), MGL_getMarkerSize(), MGL_setMarkerStyle(), MGL_getMarkerStyle(), MGL_polyMarker()

MGL_maxCharWidth

Function Returns the maximum character width for current font.

Syntax int MGL_maxCharWidth(void);

Prototype in mgraph.h

Remarks MGL_maxCharWidth() returns the maximum character width for the currently active font. You can use this routine to quickly determine if a character will possibly overlap something else on the display screen.

Return value Maximum character width for current font.

See also MGL_getCharMetrics(), MGL_getFontMetrics(), MGL_textHeight(), MGL_textWidth(), MGL_charWidth()

MGL_maxColor

Function Returns the maximum available color value.

Syntax color_t MGL_maxColor(void);

Prototype in mgraph.h

Remarks MGL_maxColor() retutns the values of the largest available color values for the current video modes. This value will always be one less than the number of available colors in that particular video mode.

Return value Maximum color value for current video mode.

MGL_maxPage

Function Returns the maximum available hardware video page.

Syntax int MGL_maxPage(void);

Prototype in mgraph.h

Remarks MGL_maxPage() returns the index of the highest hardware video page that is available. This value will always be one less than the number of hardware video pages available. Some video modes only have one hardware video page available, so this value will be 0.

Return value Index of maximum available hardware video page.

MGL_maxx

Function Returns the current maximum X coordinate

Syntax int MGL_maxx(void);

Prototype in mgraph.h

Remarks MGL_maxx() returns the maximum X coordinate available in the currently active viewport. This value will change if you change the dimensions of the current viewport.

Use the MGL_sizex() routine to determine the dimensions of the physical display area available to the program.

Return value Maximum X coordinate in current viewport.

See also MGL_maxy(), MGL_sizex(), MGL_sizey()

MGL_maxy

Function Returns the current maximum Y coordinate

Syntax int MGL_maxy(void);

Prototype in mgraph.h

Remarks MGL_maxy() returns the maximum Y coordinate available in the currently active viewport. This value will change if you change the dimensions of the current viewport.

Use the MGL_sizey() routine to determine the dimensions of the physical display area available to the program.

Return value Maximum Y coordinate in current viewport.

See also MGL_maxx(), MGL_sizex(), MGL_sizey()

MGL_modeName

Function Returns textual representation of the specified video mode.

Syntax char *MGL_modeName(int mode);

Prototype in mgraph.h

Parameters mode - Video mode number to get name for.

Remarks MGL_modeName() returns a string representing the specified video mode number.

Return value Pointer to a string representing the name of the mode.

See also MGL_driverName().

MGL_moveRel

Function Moves the CP to a new relative location.

Syntax void MGL_moveRel(point p)

Prototype in mgraph.h

Parameters p - Relative point to move the CP to.

Remarks MGL_moveRel() moves the current position (CP) to the relative location that is a distance of *p* away from the CP. Thus the location of the CP moves to is *(CP.x + p.x, CP.y + p.y)*.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_moveRelCoord

Function Moves the CP to a new relative location

Syntax void MGL_moveRelCoord(int dx,int dy);

Prototype in mgraph.h

Parameters dx,dy - Relative point to move the CP to.

Remarks MGL_moveRelCoord() moves the current position (CP) to the relative location that is a distance of *(dx,dy)* away from the CP. Thus the location the CP is moved to is *(CP.x + dx, CP.y + dy)*.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_moveTo

Function Moves the CP to a new location.

Syntax void MGL_moveTo(point p);

Prototype in mgraph.h

Parameters p - Point to move the CP to.

Remarks MGL_moveTo() moves the current position (CP) to the new point*p*.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_moveToCoord

Function Moves the CP to a new location.

Syntax void MGL_moveToCoord(int x,int y);

Prototype in mgraph.h

Parameters x,y - Point to move the CP to.

Remarks MGL_moveToCoord() moves the current position (CP) to the new point *(x,y)*.

See also MGL_moveTo(), MGL_moveToCoord(), MGL_moveRel(), MGL_moveRelCoord(), MGL_lineTo(), MGL_lineToCoord(), MGL_lineRel(), MGL_lineRelCoord().

MGL_offsetRect

Function Offsets a rectangle by a specified amount

Syntax void MGL_offsetRect(rect r,int dx,int dy)

Prototype in mgraph.h

Parameters r - Rectangle to offset
dx - Value to offset the X coordinates by
dy - Value to offset the Y coordinates by

Remarks MGL_offsetRect() moves the start of a rectangle to a new location, by adding the specified offset values to the X and Y coordinate values. The size of the rectangle is not changes, just it's location.

See also MGL_insetRect(), MGL_disjoingRect(), MGL_ptInRect()

MGL_packColor

Function Packs an RGB color value.

Syntax color_t MGL_packColor(uchar R,uchar G,uchar B);

Prototype in mgraph.h

Parameters R - Red color component (0-255)
G - Green color component (0-255)
B - Blue color component (0-255)

Remarks MGL_packColor() takes 8 bit color component values and packs them into a packed color value that is appropriate for the current video mode. This function only works properly in the HiColor and TrueColor modes, as you cannot specifiy RGB values in the color mapped modes.

Return value MGL packed color value appropriate for the current video mode.

See also MGL_unpackColor().

MGL_pixel

Function Draws a pixel at the specified location

Syntax void MGL_pixel(point p)

Prototype in mgraph.h

Parameters p - Point to plot the point at it.

Remarks MGL_pixel() plots a single pixel at the specified location in the current foregrond color.

Note: You *must* call the routine MGL_beginPixel() routine before calling the MGL_pixel(), and you *must* call the MGL_endPixel() routine after you have finished plotting pixels.

See also MGL_pixelCoord(), MGL_beginPixel(), MGL_endPixel().

MGL_pixelCoord

Function Draws a pixel at the specified location

Syntax void MGL_pixelCoord(int x,int y);

Prototype in mgraph.h

Parameters x,y - Point to plot the point at it.

Remarks MGL_pixelCoord() plots a single pixel at the specified location in the current foregrond color.

Note: You *must* call the routine MGL_beginPixel() routine before calling the MGL_pixel(), and you *must* call the MGL_endPixel() routine after you have finished plotting pixels.

See also MGL_pixelCoord(), MGL_beginPixel(), MGL_endPixel().

MGL_polyLine

Function Draws a set of connected lines.

Syntax void MGL_polyLine(int count,point *vArray);

Prototype in mgraph.h

Parameters count - Number of vertices in polyline
vArray - Array of vertices in the polyline

Remarks MGL_polyLine() draws a set of connected line (a polyline). The coordinates of the polyline are specified by *vArray*, and the lines are drawn in the current drawing attributes.

Note: The polyline is not closed by default, so if you wish to draw the outline of a polygon, you will need to add the starting point to the end of the vertex array.

See also MGL_polyMarker(), MGL_polyPoint().

MGL_polyMarker

Function Draws a set of markers.

Syntax void MGL_polyMarker(int count,point *vArray);

Prototype in mgraph.h

Parameters count - Number of markers to draw
vArray - Array of coordinates to draw the markers at

Remarks MGL_polyMarker() draws a set of markers in the current marker color, style and size at the locations passed in the *vArray* parameter.

See also MGL_polyLine(), MGL_polyPoint()

MGL_polyPoint

Function Draws a set of pixels.

Syntax void MGL_polyPoint(int count,point *vArray);

Prototype in mgraph.h

Parameters count - Number of pixels to draw
vArray - Array of coordinates to draw the pixels at

Remarks MGL_polyPoint() draws a set of pixels in the current color at the locations passed in the *vArray* parameter.

See also MGL_polyLine(), MGL_polyPoint()

MGL_popViewport

Function Pops the current viewport off the stack.

Syntax void MGL_popViewport(void);

Prototype in mgraph.h

Remarks MGL_popViewport() pops the viewport off the top of the viewport stack and makes it the new active viewport.

The MGL maintains a viewport stack that you may use to temporarily push viewport settings onto for later retrival.

See also MGL_pushViewport(), MGL_setViewport(), MGL_getViewport()

MGL_ptInRect

Function Determines if a point is in a rectangle.

Syntax bool MGL_ptInRect(point p,rect r);

Prototype in mgraph.h

Parameters p - Point to test
r - Rectangle to test point against

Remarks MGL_ptInRect() determines if a specified point is contained within a particular rectangle.

Return value True if the point is contained in the rectangle, false if not.

See also MGL_ptInRectCoord().

MGL_ptInRectCoord

Function Determines if a point is in a rectangle.

Syntax bool MGL_ptInRectCoord(int x,int y,rect r)

Prototype in mgraph.h

Parameters x,y - Point to test
r - Rectangle to test point against

Remarks MGL_ptInRectCoord() determines if a specified point is contained within a particular rectangle.

Return value True if the point is contained in the rectangle, false if not.

See also MGL_ptInRect()

MGL_pushViewport

Function Pushes the current viewport onto the stack.

Syntax void MGL_pushViewport(void);

Prototype in mgraph.h

Remarks MGL_pushViewport() pushs the current viewport values onto the viewport stack, to be retrieved at a later date.

See also MGL_popViewport(), MGL_setViewport(), MGL_getViewport()

MGL_putDivot

Function Replaces a divot of video memory.

Syntax void MGL_putDivot(void *divot);

Prototype in mgraph.h

Parameters divot - Pointer to the divot to replace

Remarks MGL_putDivot() replaces a region of video memory that was saved previously with an MGL_getDivot() routine. The divot is replaced at the same location that is was taken from on the current display page.

See also MGL_getDivot(), MGL_divotSize(), MGL_getImage(), MGL_putImage()

MGL_putIcon

Function Draws an icon at the specified location.

Syntax void MGL_putIcon(int x,int y,icon_header *icon);

Prototype in mgraph.h

Parameters x,y - Coordinates to draw the icon at
icon - Pointer to the icon data to draw

Remarks MGL_putIcon() draws an icon at the specified location. The icon *must* be in the correct format for the current video mode. The icon is drawn by punching a black hole in the background with the icon mask, and then OR'ing in the the image data for the icon.

MGL_putImage

Function Draws a video memory image on the display

Syntax void MGL_putImage(rect r,void *image,int op)

Prototype in mgraph.h

Parameters r - Rectangle to draw the image at
image - Pointer to the image to draw
op - Write mode operation to use.

Remarks MGL_putImage() blasts the specified video memory image to the display memory at the specified location. If the image is larger than the destination rectangle, it will be clipped to the current destination rectangle extents. You may specify any of the following write mode operations to be used when moving the image to video memory:

REPLACE_MODE - Replace the original pixels
AND_MODE - Logical AND with original pixels
OR_MODE - Logical OR with original pixels
XOR_MODE - Logical XOR with original pixels

See also MGL_putImageCoord(), MGL_getImage(), MGL_getImageCoord(), MGL_imageSize().

MGL_putImageCoord

Function Draws a video memory image on the display

Syntax void MGL_putImageCoord(int left,int top,int right,int bottom,void *image,int op);

Prototype in mgraph.h

Parameters left - Left coordinate to draw image at
top - Top coordinate to draw image at
right - Right coordinate to draw image at
bottom - Bottom coordinate to draw image at
image - Pointer to the image to draw
op - Write mode operation to use.

Remarks MGL_putImageCoord() blasts the specified video memory image to the display memory at the specified location. If the image is larger than the destination rectangle, it will be clipped to the current destination rectangle extents. You may specify any of the following write mode operations to be used when moving the image to video memory:

REPLACE_MODE
AND_MODE
OR_MODE
XOR_MODE

See also MGL_putImage(), MGL_getImage(), MGL_getImageCoord(), MGL_imageSize().

MGL_putMonoImage

Function Plots a monochromatic image.

Syntax void MGL_putMonoImage(int x,int y,int byteWidth,int height,void *image);

Prototype in mgraph.h

Parameters x,y - Coordinate to plot the image at
byteWidth - Width of the image in bytes
height - Height of the image in scanlines
image - Pointer to the buffer holding the image

Remarks MGL_putMonoImage() displays a single color image in the current foreground color. Where a bit is a 1 in the image definition, a pixel is plotted in the foreground color, where a bit is a 0 the original pixels are left alone.

See also MGL_getImage(), MGL_putImage().

MGL_realColor

Function Returns the real color value for a color mapped index.

Syntax color_t MGL_realColor(int color);

Prototype in mgraph.h

Parameters color -

Remarks MGL_realColor() returns a color value appropriate for the current video mode, given a color mapped index. This routine works in all video modes, including the HiColor and TrueColor video modes. In the color mapped modes, the value is simply returned unchanged. However in the HiColor and TrueColor modes, the appropriate color is looked up in the currently active palette. Thus you can still write code for the HiColor and TrueColor modes that works with color mapped indexes (although you cannot do things like hardware palette fades and rotates as the palette is implemented in software).

Return value Real color value for the color mapped index.

See also MGL_setPalette(), MGL_getPalette().

MGL_rect

Function Draws a rectangle.

Syntax void MGL_rect(rect r);

Prototype in mgraph.h

Parameters r - Rectangle to draw

Remarks MGL_rect() draws a rectangle in the current drawing attributes at the specified location. The rectangle is drawn within the mathematical boundary of the rectangle, so effectively the right and bottom edges of the rectangle are not drawn (solving problems with shared edges). Degenerate rectangles are not drawn.

See also MGL_rectCoord(), MGL_rectPt().

MGL_rectCoord

Function Draws a rectangle.

Syntax void MGL_rectCoord(int left,int top,int right,int bottom);

Prototype in mgraph.h

Parameters left - Left coordinate of the rectangle
top - Top coordinate of the rectangle
right - Right coordinate of the rectangle
bottom - Bottom coordinate of the rectangle

Remarks MGL_rectCoord() draws a rectangle in the current drawing attributes at the specified location. The rectangle is drawn within the mathematical boundary of the rectangle, so effectively the right and bottom edges of the rectangle are not drawn (solving problems with shared edges). Degenerate rectangles are not drawn.

See also MGL_rect(), MGL_rectPt().

MGL_rectPt

Function Draws a rectangle.

Syntax void MGL_rectPt(point lt,point rb);

Prototype in mgraph.h

Parameters lt - Left top coordinate of the rectangle
rb - Right bottom coordinate of the rectangle

Remarks MGL_rectPt() draws a rectangle in the current drawing attributes at the specified location. The rectangle is drawn within the mathematical boundary of the rectangle, so effectively the right and bottom edges of the rectangle are not drawn (solving problems with shared edges). Degenerate rectangles are not drawn.

See also MGL_rect(), MGL_rectCoord().

MGL_registerDriver

Function Registers a user loaded or linked device driver

Syntax int MGL_registerDriver(const char *name,void *driver);

Prototype in mgraph.h

Parameters name - Name of driver to register
driver - Pointer to the start of the driver in memory

Remarks MGL_registerDriver() registers a linked in or user loaded graphics driver with the graphics system. The driver must be one of the standard graphics device drivers supported by the library. The loaded driver is checked to ensure that it actually is a valid MGL device driver.

This routine is usually used to inform the MGL that the user has linked the device driver code directly with the application (or has loaded it onto the heap) so should not be dynamically loaded from disk at runtime.

Return value grOK on success, grBadDriver if driver passed was invalid

See also MGL_init()

MGL_restoreAttributes

Function Restores a previously saved attribute list

Syntax void MGL_restoreAttributes(attributes *attr);

Prototype in mgraph.h

Parameters attr - Pointer to the attribute list to restore

Remarks MGL_restoreAttributes() restores a set of attributes that were saved with the MGL_getAttributes() routine. The attributes list represents the current state of the MGL. The value of the color palette is not changed by this routine.

See also MGL_getAttributes

MGL_restoreCRTMode

Function Resets the system back into text mode

Syntax void MGL_restoreCRTMode(void);

Prototype in mgraph.h

Remarks MGL_restoreCRTMode() resets the system back into the original text mode that was present before the MGL was started. If the system was in the EGA/VGA 43/50 line modes, this mode will be reset to its original state. The MGL can be re-started with the MGL_setGraphMode() routine.

See also MGL_init(), MGL_setGraphMode()

MGL_result

Function Returns result code of the last graphics operation

Syntax int MGL_result(void);

Prototype in mgraph.h

Remarks MGL_result() returns the result code of the last graphics operation. The internal result code is reset back to grOK on return from this routine, so you should only call the routine once after the graphics operation. The following result codes are returned by MGL_result:

grOK No error
grNoInit Graphics driver has not been installed
grNotDetected Graphics hardware was not detected
grDriverNotFound Graphics driver file was not found
grBadDriver File loaded was not a graphics driver
grLoadMem Not enough memory to load graphics driver
grInvalidMode Invalid graphics mode for selected driver
grInvalidDriver Driver number is invalid
grError General graphics error
grInvalidName Invalid driver name
grNoMem Not enough memory to perform operation
grNoModeSupport Select video mode not supported by hardware
grInvalidFont Invalid font data
grBadFontFile File loaded was not a font file
grFontNotFound Font file was not found
grOldDriver Driver file is an incompatible old version

Return value Result code of the last graphics operation

See also MGL_setResult()

MGL_rotatePalette

Function Rotates the values in a palette structure

Syntax void MGL_rotatePalette(palette *pal,int numColors,int direction);

Prototype in mgraph.h

Parameters pal - Pointer to the palette array to rotate
numColors - Number of colors entries in the palette array
direction - Direction to rotate the palette entries

Remarks MGL_rotatePalette() rotates the values in the passed palette array in the specified direction. The palette may be any arbitrary size, so you can rotate just a subset of the values in the current physical palette. Note that this routine has not effect on the currently active palette. In order to make the new rotated palette active you will need to call the MGL_setPalette() routine.

The direction of rotation should be one of the following:

PAL_ROTATE_UP - Rotate palette entries up in memory
PAL_ROTATE_DOWN - Rotate palette entries down in memory

When the direction specified is PAL_ROTATE_UP, the first entry in the palette is moved to the last position in the palette, and all the remaining entries are move one position down in the array. The the direction specified is PAL_ROTATE_DOWN, the last entry is moved into the first entry of the palette, and the remaining entries are all moved on position up in the array.

See also MGL_setPalette(), MGL_getPalette()

MGL_scanLeftForColor

Function Scans left in video memory for a specified color

Syntax int MGL_scanLeftForColor(int x,int y,color_t color);

Prototype in mgraph.h

Parameters x,y - Location to begin scanline search
color - Color value to search for

Remarks MGL_scanLeftForColor() begins scanning in video memory at the specified location for the specified color. The search begins at the location (x,y) and searches left along the scanline from this point and returns the x coordinate of the pixel if one is found, or -1 if the search went beyond the left edge of the display screen.

No clipping or viewport mapping is performed by this routine, but can be performed after calling this routine. This routine can be used as the basis of a high performance floodfill operation. Have a look in the file FFILL.C in the EXAMPLES directory which uses this routine to implement a fast floodfill operation.

Return value X coordinate of pixel if found, or -1 if search hit left edge of display

See also MGL_scanRightForColor(), MGL_scanLeftWhileColor(), MGL_scanRightWhileColor()

MGL_scanLeftWhileColor

Function Scans left in video memory for any color but the specified color

Syntax int MGL_scanLeftWhileColor(int x,int y,color_t color);

Prototype in mgraph.h

Parameters x,y - Location to begin scanline search
color - Color value to search on

Remarks MGL_scanLeftWhileColor() begins scanning in video memory at the specified location and continues to scan while the pixels are the same as the specified seed color. The search begins at the location (x,y) and searches left along the scanline from this point and returns the x coordinate of the first pixel found that is not of the specified color, or -1 if the search went beyond the left edge of the display screen.

No clipping or viewport mapping is performed by this routine, but can be performed after calling this routine. This routine can be used as the basis of a high performance floodfill operation. Have a look in the file FFILL.C in the EXAMPLES directory which uses this routine to implement a fast floodfill operation.

Return value X coordinate of pixel if found, -1 if search hit left edge of display

See also MGL_scanLeftForColor(), MGL_scanRightForColor(), MGL_scanRightWhileColor()

MGL_scanLine

Function Fills a specified scanline

Syntax void MGL_scanLine(int y,int x1,int x2);

Prototype in mgraph.h

Parameters y - Y coordinate of scanline to fill
x1 - Starting X coordinate of scanline to fill
x2 - Ending X coordinate of scanline to fill

Remarks MGL_scanLine() fills the specified portion of a scanline in the current attributes and fill pattern. This can be used to implement higher level complex fills, such as region fills, floodfills etc.

See also MGL_penStyle(), MGL_setPenBitmapPattern(), MGL_setPenPixmapPattern()

MGL_scanRightForColor

Function Scans right in video memory for a specified color

Syntax int MGL_scanRightForColor(int x,int y,color_t color);

Prototype in mgraph.h

Parameters x,y - Location to begin scanline search
color - Color value to search for

Remarks MGL_scanRightForColor() begins scanning in video memory at the specified location for the specified color. The search begins at the location (x,y) and searches right along the scanline from this point and returns the x coordinate of the pixel if one is found, or one more than the maximum X coordinate if the search went beyond the right edge of the display screen.

No clipping or viewport mapping is performed by this routine, but can be performed after calling this routine. This routine can be used as the basis of a high performance floodfill operation. Have a look in the file FFILL.C in the EXAMPLES directory which uses this routine to implement a fast floodfill operation.

Return value X coordinate of pixel if found, or maxx+1 if search hit right edge of display

See also MGL_scanLeftForColor(), MGL_scanLeftWhileColor(), MGL_scanRightWhileColor()

MGL_scanRightWhileColor

Function Scans right in video memory for any color but the specified color

Syntax int MGL_scanRightWhileColor(int x,int y,color_t color);

Prototype in mgraph.h

Parameters x,y - Location to begin scanline search
color - Color value to search on

Remarks MGL_scanRightWhileColor() begins scanning in video memory at the specified location and continues to scan while the pixels are the same as the specified seed color. The search begins at the location (x,y) and searches right along the scanline from this point and returns the x coordinate of the first pixel found that is not of the specified color, or one more than the maximum X coordinate if the search went beyond the right edge of the display screen.

No clipping or viewport mapping is performed by this routine, but can be performed after calling this routine. This routine can be used as the basis of a high performance floodfill operation. Have a look in the file FFILL.C in the EXAMPLES directory which uses this routine to implement a fast floodfill operation.

Return value X coordinate of pixel if found, maxx+1 if search hit right edge of display

See also MGL_scanLeftForColor(), MGL_scanRightForColor(), MGL_scanLeftWhileColor()

MGL_setActivePage

Function Sets the currently active hardware display page.

Syntax void MGL_setActivePage(int page);

Prototype in mgraph.h

Parameters page - Number of active hardware display page to use

Remarks MGL_setActivePage() sets the currently active hardware video page number to which all output from the MGL is sent to. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages,

See also MGL_getActivePage(), MGL_setVisualPage(), MGL_getVisualPage()

MGL_setAspectRatio

Function Sets the current video modes aspect ratio

Syntax void MGL_setAspectRatio(int aspectRatio);

Prototype in mgraph.h

Parameters aspectRatio - New value for the aspect ratio

Remarks MGL_setAspectRatio() sets the aspect ratio of the currently active output device's physical pixels. This ratio is equal to:

$$\frac{\text{pixel x size}}{\text{pixel y size}} \times 1000$$

The device aspect ratio can be used to display circles and squares on the display device by approximating them with ellipses and rectangles of the appropriate dimensions. Thus in order to determine the number of pixels in the y direction for a square with 100 pixels in the x direction, we can simply use the formula:

y_pixels = ((long)x_pixels * 1000) / aspectratio

Note the cast to a long to avoid arithmetic overflow, as the apsect ratio is returned as an integer value with 1000 being a 1:1 aspect ratio.

Normally you should not need to change the aspect ratio, as the MGL will automatically determine the correct aspect ratio for the current display mode.

See also MGL_getAspectRatio()

MGL_setBackColor

Function Sets the currently active background color

Syntax void MGL_setBackColor(color_t color);

Prototype in mgraph.h

Parameters color - New background color value

Remarks MGL_setBackColor() sets the current background color value. The background color value is used to clear the display and viewport with the MGL_clearDevice() and MGL_clearViewport() routines, and is also used for filling solid primitives in the BITMAP_PATTERN_OPAQUE fill mode.

Note: The value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the MGL_packColor() routine to pack 24 bit RGB values for direct color video modes.

See also MGL_getBackColor(), MGL_setColor(), MGL_getColor(), MGL_packColor()

MGL_setBorderColors

Function Sets the current border color values.

Syntax void MGL_setBorderColors(color_t bright,color_t dark);

Prototype in mgraph.h

Parameters bright - New value for border bright color
dark - New value for border dark color

Remarks MGL_setBorderColors() sets the currently active border colors values. There are two border color defined by the MGL, the *bright* border color value and the *dark* border color value. These values are used by the MGL_drawBorder(), MGL_drawHDivider() and MGL_drawVDivider() routines to determine the colors to draw the psuedo 3D borders in. These values will be set automatically for you by default when the MGL is initisalised, but you will need to change these values if you modify the palette.

Note: The value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the MGL_packColor() routine to pack 24 bit RGB values for direct color video modes.

See also MGL_getBorderColors(), MGL_packColor()

MGL_setBufSize

Function Sets the size of the internal MGL buffer.

Syntax void MGL_setBufSize(unsigned size);

Prototype in mgraph.h

Parameters size - New size of the internal MGL buffer

Remarks MGL_setBufSize() sets the size of the internal MGL buffer. When the MGL is rendering primitives, it needs a buffer of local scratch space that it can uses for temporary results while rendering the primitives.

The default size of this buffer is 4096 bytes and is adequate for most needs. If however you attempt to render some primitives and the MGL runs out of local storage space you would need to increase the size of this internal buffer.

This routine *must* be called *before* the MGL is initialised for the first time.

See also MGL_init()

MGL_setClipMode

Function Sets the clipping mode for the MGL

Syntax void MGL_setClipMode(bool mode);

Prototype in mgraph.h

Parameters mode - True for clipping to be turned on, false for no clipping.

Remarks MGL_setClipMode() sets the current clipping mode. You can selectively turn clipping on and off for the MGL, in order to speed up some operations. Clipping is turned on by default, and generally you will want to leave clipping enabled, however if you are doing your own rendering and perform your own clipping you may want to turn this off for extra performance from the MGL.

See also MGL_getClipMode()

MGL_setClipRect

Function Sets the current clipping rectangle

Syntax void MGL_setClipRect(rect clip);

Prototype in mgraph.h

Parameters clip - New clipping rectangle to be used.

Remarks MGL_setClipRect() sets the current clipping rectangle coordinates. The current clipping rectangle is used to clip all output, and is always defined as being relative to the currently active viewport The clipping rectangle can be no larger than the currently active viewport, and will be truncated if an attempt it made to allow clipping outside of the active viewport.

See also MGL_getClipRect, MGL_setViewport(), MGL_getViewport()

MGL_setColor

Function Sets the current foreground color

Syntax void MGL_setColor(color_t color);

Prototype in mgraph.h

Parameters color - New foreground color value

Remarks MGL_setColor() sets the current foreground color values. The foreground color value is used to draw all primitives.

Note: The value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the MGL_packColor() routine to pack 24 bit RGB values for direct color video modes.

See also MGL_getColor(), MGL_setBackColor(), MGL_getBackColor(), MGL_packColor()

MGL_setCursorColor

Function Sets the current mouse cursor color.

Syntax void MGL_setCursorColor(color_t color);

Prototype in mgraph.h

Parameters color - New mouse cursor color value

Remarks MGL_setCursorColor() sets the currently active mouse cursor color. The mouse cursor color is used to determine what foreground color is used to draw the mouse cursor in.

Note: The value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the MGL_packColor() routine to pack 24 bit RGB values for direct color video modes.

See also MGL_getCursorColor(), MGL_packColor()

MGL_setDefaultPalette

Function Resets the palette to the MGL defaut values.

Syntax void MGL_setDefaultPalette(void);

Prototype in mgraph.h

Remarks MGL_setDefaultPalette() sets the palette to the current MGL default values for the current video mode. This can be used to reset the palette to the original default values that the palette is programmed with when the MGL is initialised.

See also MGL_getDefaultPalette(), MGL_setPalette(), MGL_getPalette()

MGL_setGraphMode

Function Restarts graphics mode operation.

Syntax void MGL_setGraphMode(void);

Prototype in mgraph.h

Remarks MGL_setGraphMode() restarts the graphics mode operation of the MGL after MGL_restoreCRTMode has been called to put the system back into the original text mode. To start the graphics mode initialially, you call the MGL_int() routine.

See also MGL_restoreCRTMode(), MGL_init()

MGL_setMarkerColor

Function Sets the current marker color value.

Syntax void MGL_setMarkerColor(color_t color);

Prototype in mgraph.h

Parameters color - New marker color to set

Remarks MGL_setMarkerColor() sets the current marker color value. The marker color is used when drawing markers with the MGL_marker() routine.

Note: The value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the MGL_packColor() routine to pack 24 bit RGB values for direct color video modes.

See also MGL_getMarkerColor(), MGL_marker(), MGL_polyMarker(), MGL_packColor()

MGL_setMarkerSize

Function Sets the current marker size value.

Syntax void MGL_setMarkerSize(int size);

Prototype in mgraph.h

Parameters size - New marker size

Remarks MGL_setMarkerSize() sets the current marker size. The marker size is used to determine how big to draw the markers that are drawn with the MGL_marker() routine. The size is defined as dimension from the middle of the marker to the edges, so the actual dimensions of the marker will be approximately twice the maker size. If marker size of 1 will define a marker that is contained within a rectangle 3 pixels wide.

See also MGL_getMarkerSize(), MGL_marker(), MGL_polyMarker()

MGL_setMarkerStyle

Function Sets the current marker style.

Syntax void MGL_setMarkerStyle(int style);

Prototype in mgraph.h

Parameters style - New marker style value.

Remarks MGL_setMarkerStyle() sets the current marker style value. The marker style defines the type of marker to be rendered. Currently the MGL defines the following markers:

MARKER_SQUARE - A solid square
MARKER_CIRCLE - A solid circle
MARKER_X - A cross made of two lines

See also MGL_getMarkerStyle(), MGL_marker(), MGL_polyMarker()

MGL_setPalette

Function Sets the currently active palette values.

Syntax void MGL_setPalette(palette *pal,int numColors,int startIndex);

Prototype in mgraph.h

Parameters pal - Pointer to array of palette values to program
numColors - Number of colors to program from the array
startIndex - Starting index to program from in the array

Remarks MGL_setPalette() sets part or all of the currently active palette given the values passed in the array *pal*. You can specify only a subset of the palette values to be modified with the startIndex and numColors arguments.

Thus to set the entire palette in a 256 color video mode, you would use:

```
MGL_setPalette(pal,255,0);
```

or to set the top half of the palette you would use:

```
MGL_setPalette(pal,128,128);
```

Note that the MGL will ensure that the palette is programmed without snow depending on the value of the current snow level defined by calling the MGL_setPaletteSnowLevel() routine, and will wait for the start of a vertical retrace before the first entry is programmed.

See also MGL_getPalette(), MGL_setPaletteEntry(), MGL_setPaletteSnowLevel(), MGL_getPaletteSnowLevel()

MGL_setPaletteEntry

Function

Syntax void MGL_setPaletteEntry(int entry,uchar red,uchar green,uchar blue);

Prototype in mgraph.h

Parameters entry - Palette index to program
red - Red component for palette entry
green - Green component for palette entry
blue - Blue component for palette entry

Remarks MGL_setPaletteEntry() sets the color values of a single palette entry. If you wish to set more than a single palette index you should use the MGL_setPalette() routine which is faster for multiple entries. The main reason for this is that this routine will wait for a vertical retrace before the palette entry is programmed to ensure that snow does not occur.

See also MGL_getPaletteEntry(), MGL_setPalette(), MGL_getPalette()

MGL_setPaletteSnowLevel

Function Sets the current palette snow level

Syntax void MGL_setPaletteSnowLevel(int level);

Prototype in mgraph.h

Parameters level - New snow level

Remarks MGL_setPaletteSnowLevel() sets the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. The MGL uses a reasonable default of 100 entries per retrace, but you may want to modify this on faster or slower machines (this should be a user option). In the future the MGL will automatically determine the optimum value for this at initialisation time.

See also MGL_getPaletteSnowLevel(), MGL_setPalette()

MGL_setPenBitmapPattern

Function Sets the currently active bitmap pattern.

Syntax void MGL_setPenBitmapPattern(pattern *pat);

Prototype in mgraph.h

Parameters pat - New bitmap pattern to use.

Remarks MGL_setPenBitmapPattern() sets the currently active bitmap pattern used when rendering patterned primitive in the BITMAP_PATTERN_TRANSPARENT and BITMAP_PATTERN_OPQAUE pen styles. A bitmap pattern is defined as an 8 x 8 pixel bitmap pattern stored as an array of 8 bytes. The pattern value is copied from the passed structure.

When filling in the BITMAP_PATTERN_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the BITMAP_PATTERN_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0.

See also MGL_getPenBitmapPattern(), MGL_setPenStyle(), MGL_getPenStyle()

MGL_setPenSize

Function Sets the current pen size

Syntax void MGL_setPenSize(int height,int width);

Prototype in mgraph.h

Parameters height - Height of the pen in pixels
width - Width of the pen in pixels

Remarks MGL_setPenSize() sets the size of the current pen in pixels. The default pen is 1 pixel by 1 pixel in dimensions, however you can change this to whatever value you like. When primitive are rendered with a pen other than the default, the pixels in the pen always lie to the right and below the current pen position.

See also MGL_getPenSize()

MGL_setPenStyle

Function Sets the current pen style

Syntax void MGL_setPenStyle(int style);

Prototype in mgraph.h

Parameters style - New pen style to use.

Remarks MGL_getPenStyle() returns the currently active pen style. The MGL supports the following pen styles:

SOLID_PATTERN - Fill with solid color
BITMAP_PATTERN_OPAQUE - Pattern fill
BITMAP_PATTERN_TRANSPARENT - Transparent pattern fill

When filling in the BITMAP_PATTERN_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the BITMAP_PATTERN_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0.

See also MGL_getPenStyle(), MGL_setPenBitmapPattern()

MGL_setPolygonType

Function Sets the current polygon type

Syntax void MGL_setPolygonType(int type);

Prototype in mgraph.h

Parameters type - New polygon type

Remarks MGL_setPolygonType() sets the current polygon type. You can change this value to force the MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). The MGL supports the following polygon types:

AUTO_POLYGON - MGL automatically determines type
CONVEX_POLYGON - All polygons rendered as convex
COMPLEX_POLYGON - All polygons rendered as complex

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rendering.

See also MGL_getPolygonType(), MGL_fillPolygon()

MGL_setRelViewport

Function Sets a viewport relative to the current one

Syntax void MGL_setRelViewport(rect view);

Prototype in mgraph.h

Parameters view - Bounding rectangle for the new viewport.

Remarks MGL_setRelViewport() sets the current viewport to the viewport specified by view, where view is relative to the currently active viewport. The new viewport is restricted to fall within the bounds of the currently active viewport.

All output in the MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the video display.

See also MGL_getViewport(), MGL_setViewport(), MGL_clearViewport(), MGL_setClipRect()

MGL_setResult

Function Sets the internal MGL result flag.

Syntax void MGL_setResult(int result)

Prototype in mgraph.h

Parameters result - New internal result flag

Remarks MGL_setResult() sets the internal MGL result flag to the specified value. This routine is primarily for extension libraries, but you can use it to add your own extension functions to the MGL that will return result codes in the same manner as the MGL.

See also MGL_getResult()

MGL_setSpaceExtra

Function Sets the current space extra value.

Syntax void MGL_setSpaceExtra(int extra);

Prototype in mgraph.h

Parameters extra - New space extra value.

Remarks MGL_setSpaceExtra() sets the current space extra value used when drawing text in the current font. The space extra vaue is normally zero, but can be a positive or negative value. This value can be used to insert extra space between the characters in a font (making this value a large negative value will make the characters run on top of each other).

See also MGL_getSpaceExtra(), MGL_drawStr()

MGL_setTextDirection

Function Sets the current text direction

Syntax void MGL_setTextDirection(int direction);

Prototype in mgraph.h

Parameters direction - New text direction value

Remarks MGL_setTextDirection() sets the current text direction. The MGL supports the following text directions:

LEFT_DIR - Text runs to the left (right to left)
UP_DIR - Text runs in the up direction
RIGHT_DIR - Text runs to the right (left to right)
DOWN_DIR - Text runs in the down direction

Currently the MGL only supports directional text with the default 8x8 bitmap font and vector fonts. Bitmap fonts can only be drawn in the RIGHT_DIR direction.

See also MGL_getTextDirection(), MGL_drawStr()

MGL_setTextJustify

Function Sets the current text horizontal and vertical justification

Syntax void MGL_setTextJustify(int horiz,int vert);

Prototype in mgraph.h

Parameters horiz - New horiztonal text justification value
vert - New vertical text justification value

Remarks MGL_setTextJustify() sets the current text justification values. The MGL supports the following horizontal justification types:

LEFT_TEXT - Text is left justified
CENTER_TEXT - Text is centered left to right
RIGHT_TEXT - Text is right justified

and the following vertical justification types:

TOP_TEXT - Text is top justified
CENTER_TEXT - Text is centered top to bottom
BASELINE_TEXT - Text is justified to the baseline
BOTTOM_TEXT - Text is bottom justified

See also MGL_getTextJustify()

MGL_setTextSettings

Function Restores the current text settings.

Syntax void MGL_setTextSettings(text_settings *settings);

Prototype in mgraph.h

Parameters settings - Text settings to restore

Remarks MGL_setTextSettings() restores a set of previously saved text settings. This routine is provides a way to save and restore all the values relating to the rendering of text in the MGL with a single function call. The text settings values are stored in the following structure:

```
typedef struct {
   int      horiz_just;
   int      vert_just;
   int      dir;
   int      sz_numerx;
   int      sz_numery;
   int      sz_denomx;
   int      sz_denomy;
   int      space_extra;
   font     *fnt;
```

where *horiz_just* and *vert_just* define the horizontal and vertical justification values, *dir* defines the current text direction, *space_extra* defines the current space extra value and *fnt* define the currently active font (stored in system memory). The *sz_numerx, sz_numery, sz_demonx* and *sz_denomy* specify the current text scaling factors.

See also MGL_getTextSettings()

MGL_setTextSize

Function Sets the current text scaling factors

Syntax void MGL_setTextSize(int numerx,int denomx,int numery,int denomy);

Prototype in mgraph.h

Parameters numerx - X scaling numerator value
denomx - X scaling denominator value
numery - Y scaling numerator value
denomy - Y scaling denominator value

Remarks MGL_setTextSize() sets the current text scaling factors used by the MGL. The text size values define an integer scaling factor to be used, where the actual values will be computed using the following formula:

$$\text{scaled} = \frac{\text{unscaled} \times \textit{numer}}{\textit{denom}}$$

Note: Currently the MGL can only scale vectors fonts.

See also MGL_getTextSize()

MGL_setViewport

Function Sets the currently active viewport

Syntax void MGL_setViewport(rect view);

Prototype in mgraph.h

Parameters view - New global viewport bounding rectangle

Remarks MGL_setViewport() sets the dimensions of the currently active viewport. These dimensions are global to the entire display area used by the currently active video device driver.

All output in the MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the video display.

See also MGL_getViewport(), MGL_setRelViewport(), MGL_clearViewport(), MGL_setClipRect()

MGL_setViewStackSize

Function Set the size of the viewport stack

Syntax void MGL_setViewStackSize(int size)

Prototype in mgraph.h

Parameters size - Number of viewports on the viewport stack

Remarks MGL_setViewStackSize() sets the size of the viewport stack used by the MGL. Be default the MGL uses a stack of 20 viewports, however if you encounter problems with viewport stack overflow, you can call this routine to resize the viewport stack _before_ calling MGL_init() for the first time.

MGL_setVisualPage

Function Sets the currently visible hardware video page.

Syntax int MGL_setVisualPage(int page);

Prototype in mgraph.h

Parameters page - New visible hardware page number

Remarks MGL_setVisualPage() sets the currently visible hardware video page number. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages.

See also MGL_getVisualPage(), MGL_getActivePage(), MGL_setActivePage().

MGL_setWriteMode

Function Sets the current write mode operation

Syntax int MGL_setWriteMode(int mode);

Prototype in mgraph.h

Parameters mode - New write mode operation to use

Remarks MGL_setWriteMode() sets the currently active write mode. The MGL supports the following write mode operations for all output primitives:

REPLACE_MODE - Replace the original pixels
AND_MODE - Logical AND with original pixels
OR_MODE - Logical OR with original pixels
XOR_MODE - Logical XOR with original pixels

See also MGL_getWriteMode()

MGL_singleBuffer

Function Returns the system back to single buffered mode.

Syntax void MGL_singleBuffer(void);

Prototype in mgraph.h

Remarks MGL_singleBuffer() puts the system back into single buffer mode. The active display page is made to be the same as the current visual display page for hardware double buffering. This may or may not be the first hardware video page.

See also MGL_doubleBuffer(), MGL_swapBuffers()

MGL_sizex

Function Returns the total device x coordinate dimensions

Syntax int MGL_sizex(void);

Prototype in mgraph.h

Remarks MGL_sixex() returns the total number of pixels available along the X coordinate axis for the currently active output device. This is different to the MGL_maxx() routine which returns the dimensions of the currently active viewport.

Return value Number of pixels in X direction for entire device - 1

See also MGL_sizey(), MGL_maxx(), MGL_maxy()

MGL_sizey

Function Returns the total device y coordinate dimensions

Syntax int MGL_sizey(void);

Prototype in mgraph.h

Remarks MGL_sixey() returns the total number of pixels available along the Y coordinate axis for the currently active output device. This is different to the MGL_maxy() routine which returns the dimensions of the currently active viewport.

Return value Number of pixels in Y direction for entire device - 1

See also MGL_sizex(), MGL_maxx(), MGL_maxy()

MGL_swapBuffers

Function Swaps the currently active front and back buffers

Syntax void MGL_swapBuffers(void);

Prototype in mgraph.h

Remarks MGL_swapBuffers() swaps the currently active front and back buffers. This routine should only be called after the MGL_doubleBuffer() routine has been called to initialise the double buffering. Once double buffering has been set up, all output from the MGL will go to the current offscreen buffer, and the output can be made visible with the MGL_swapBuffers() routine. This routine is normally used to achieve smooth animation for complex scenes.

See also MGL_doubleBuffer(), MGL_singleBuffer()

MGL_textHeight

Function Returns the height of the current font in pixels

Syntax int MGL_textHeight(void);

Prototype in mgraph.h

Remarks MGL_textHeight() returns the height of the currently active font in pixels. This includes any scaling transformations that are applied to the font and will be as accurate as possible at the resolution of the display device.

Return value Height of the current font in pixels

See also MGL_textWidth(), MGL_drawStr(), MGL_getCharMetrics(), MGL_getFontMetrics()

MGL_textWidth

Function Returns the width of the charcter string in pixels

Syntax int MGL_textWidth(const char *str);

Prototype in mgraph.h

Parameters str - Character string to measure

Remarks MGL_textWidth() returns the width of the specified character string using the dimensions of the currently active font in pixels. This includes any scaling transformations that are applied to the font and will be as accurate as possible at the resolution of the display device.

Return value Width of the character string in pixels

See also MGL_textHeight(), MGL_drawStr(), MGL_getCharMetrics(), MGL_getFontMetrics()

MGL_underScoreLocation

Function Returns the location to begin drawing an underscore for the font.

Syntax void MGL_underScoreLocation(int *x, int *y, const char *str)

Prototype in mgraph.h

Parameters x - X coordinate to be passed to MGL_drawStrXY()
y - Y coordinate to be passed to MGL_drawStrXY()
str - String to measure

Remarks MGL_underScoreLocation() takes an (x,y) location that would normallybe be used to draw a string with MGL_drawStrXY(), and adjusts the coordinates to begin at the under score location for the current font, in the current drawing attributes. Thus the entire character string can be underlined by drawing a line starting at the computed underscore location and extending for MGL_textWidth() pixels in length.

See also MGL_drawStrXY(), MGL_textWidth()

MGL_unloadFont

Function Unloads the specified font from memory.

Syntax void MGL_unloadFont(font *font);

Prototype in mgraph.h

Parameters font - Pointer to loaded font file to unload.

Remarks MGL_unloadFont() attempts to unload the specified font if the font is a valid font that was loaded with the MGL_loadFont() routine. The memory occupied by the font will be released to the global heap.

Return value True if font was unloaded, false if font was invalid.

See also MGL_loadFont()

MGL_unpackColor

Function Unpacks a packed 24 bit color value into RGB components.

Syntax void MGL_unpackColor(color_t color,uchar *R,uchar *G,uchar *B);

Prototype in mgraph.h

Parameters color - Color value to unpack.
R - Place to store the red component
G - Place to store the green component
B - Place to store the blue component

Remarks MGL_unpackColor() takes a packed color value in the correct format for the current color mode, and extracts the red, green and blue components. Note that the color values may not be the same as when you packed them with MGL_packColor() if the color mode is a 15 or 16 bit mode because of loss of precision. The values are scaled back into the normal 24 bit RGB space.

See also MGL_packColor(), MGL_getPixelFormat()

MGL_packColor

Function Packs an RGB triple into the correct format for current mode.

Syntax color_t MGL_packColor(uchar R, uchar G, uchar B)

Prototype in mgraph.h

Parameters R - Red component of color
G - Green component of color
B - Blue component of color

Remarks MGL_packColor() takes a 24 bit RGB triple and converts it to the correct pack pixel format required by the current video mode. This packed pixel color value can then be passed to routines that require an MGL color_t color value such as MGL_setColor() etc.

Return value MGL pack pixel color value representing the specified color.

See also MGL_unpackColor(), MGL_getPixelFormat(), MGL_setColor()

MGL_useFont

Function Sets the currently active font.

Syntax bool MGL_useFont(font *font)

Prototype in mgraph.h

Parameters font - New font to use.

Remarks MGL_useFont() selects the specified font as the currently active font for use by the MGL. If the font data is invalid, the MGL result flag is set and the routine will return false.

Note: Do not unload a font file if it is currently in use by the MGL!

Return value True if the font was valid and selected, false if not.

See also MGL_drawStr()

MGL_vecFontEngine

Function Generates the commands to draw vector font.

Syntax bool MGL_vecFontEngine(int x, int y, const char *str, void (*move)(int x,int y), void (*draw)(int x,int y));

Prototype in mgraph.h

Parameters x,y - Coordinate to start drawing text at
str - Character string to draw
move - Routine to call to perform a move operation
draw - Routine to call to perform a draw operation

Remarks MGL_vecFontEngine() calls a set of user supplied routines to render the characters in a vector font. This allows the vector fonts to be drawn in 2D or 3D floating point coordinate systems by transforming each of the operations needed to draw each character, or in any coordinate system that the users desires.

The *move* routine is called to move the cursor to a new location, and the *draw* routine is used to perform a draw operation from the current location to the specified location. Each character in the vector font is started with a move operation.

Note that the coordinates passed to the move and draw routines will be offset from the point (x,y), where the point (x,y) is the origin of the first character (ie: it lies on its baseline). Note also that the coordinates will be relative to the origin with the origin at the lower left corner of each character (ie: inverse of normal device coordinate yaxis values).

This routine does not honor the standard scaling factors, but simply draws the characters at character size of (1,1,1,1) (because scaling will be done by the user supplied move and draw routines).

If the passed font is NOT a valid vector font, this routine returns false.

Return value True if string correctly rendered, false if font is not a vector font.

See also MGL_drawStr(), MGL_useFont()

MGL_vSync

Function Waits for a vertical refresh signal.

Syntax void MGL_vSync(void);

Prototype in mgraph.h

Remarks MGL_vSync() waits for a vertical refresh signal from the video display before returning. This can be used to sync your graphics output to the vertical sync pulse to produce flicker free animation or to produce animation that runs at a constant speed.

MS_obscure

Function Hides the mouse cursor from view during graphics output.

Syntax void MS_obscure(void);

Prototype in mgraph.h

Remarks MS_obscure() hides the mouse cursor from view in order to perform graphics output using the MGL. If the graphics device driver supports a hardware cursor, this is handled by the hardware, otherwise it is removed from the display. You should call this routine rather than MS_hide() in order to temporarily hide the cursor during graphics output as the MS_hide() routine can produce incorrect results if call in quick succession for systems that support a hardware mouse cursor.

See also MS_unobscure()

MS_unobscure

Function Restores the mouse cursor to view after graphics output.

Syntax void MS_unobscure(void);

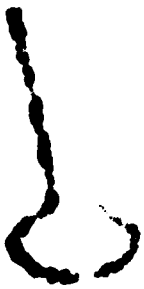
Prototype in mgraph.h

Remarks MS_unobscure() redisplays the cursor again after screen output has finished.

See also MS_obscure()

CHAPTER 3

Chapter 3 - C++ Encapsulation of MGL
- About C++ Implementation
- Sample Class Description
- Alphabetical List of Class Descriptions



CHAPTER

4

Chapter 4 - Data Structures
About MGL Data Structures
Listing of Data Structure Descriptions