

SciTech MGL

Professional Graphics Library

Reference Guide

Version 4.0

SciTech Software, Inc.
505 Wall Street
Chico, CA 95928

Main: (530) 894-8400
FAX: (530) 894-9069
www.scitechsoft.com

Information in the document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of SciTech Software, Inc.

© 1996-8 SciTech Software, Inc. All rights reserved.

SciTech Software, Inc.
505 Wall Street
Chico, CA 95928 USA
(530) 894-8400
(530) 894-9069 FAX

SciTech Display Doctor, SciTech MGL, SciTech SuperVGA Kit, UniVBE, UVBELib, and WinDirect are trademarks of SciTech Software, Inc.

OpenGL is a registered trademark of Silicon Graphics, Inc.

All other marks are trademarks or registered trademarks of their respective companies.

Contents

MGL Library Overview	1
Environment detection and initialization	2
Event Handling.....	2
Mouse Handling.....	3
Device context information and manipulation	3
Color and palette manipulation	4
Device clearing.....	5
Game Framework	5
Sprite Manager.....	6
OpenGL Rendering	6
Viewport and clip rectangle manipulation.....	7
Pixel plotting	7
Scanline color scanning	8
High performance drawing interface functions	8
Line drawing and clipping.....	8
Polyline, marker and pixel drawing.....	9
Polygon drawing	9
Rectangle drawing.....	9
Ellipse drawing.....	10
Text attribute manipulation	10
Text drawing	11
BitBlt support	11
Linear OffscreenDC BitBlt support.....	11
Monochrome bitmap manipulation.....	12
Double buffering support.....	12
Hardware scrolling or panning.....	12
Region management	13
Region algebra	13
RGB to 8 bit halftone dithering routines	14
Resource loading/unloading.....	14
Rectangle and Point manipulation	14
Random number generation routines	15
Fixed point utility routines.....	15
Memory allocation and clearing.....	15
Platform Specific: MGLDOS	16
Platform Specific: MGLWIN.....	16
MGL Library Reference	19
<i>Reference Entry Template.....</i>	<i>19</i>

<i>CPU_getProcessorSpeed</i>	20
<i>CPU_getProcessorType</i>	21
<i>CPU_haveMMX</i>	22
<i>EVT_asciiCode</i>	23
<i>EVT_flush</i>	24
<i>EVT_getNext</i>	25
<i>EVT_halt</i>	27
<i>EVT_isKeyDown</i>	28
<i>EVT_peekNext</i>	29
<i>EVT_post</i>	30
<i>EVT_repeatCount</i>	31
<i>EVT_scanCode</i>	32
<i>EVT_setTimerTick</i>	33
<i>GM_chooseMode</i>	34
<i>GM_cleanup</i>	35
<i>GM_exit</i>	36
<i>GM_findMode</i>	37
<i>GM_getDoDraw</i>	38
<i>GM_getExitMainLoop</i>	39
<i>GM_getHaveWin31</i>	40
<i>GM_getHaveWin95</i>	41
<i>GM_getHaveWinNT</i>	42
<i>GM_init</i>	43
<i>GM_initPath</i>	45
<i>GM_initSysPalNoStatic</i>	46
<i>GM_initWindowPos</i>	47
<i>GM_mainLoop</i>	48
<i>GM_processEvents</i>	50
<i>GM_processEventsWin</i>	51
<i>GM_realizePalette</i>	52
<i>GM_registerEventProc</i>	53
<i>GM_registerMainWindow</i>	54
<i>GM_setAppActivate</i>	55
<i>GM_setDrawFunc</i>	56
<i>GM_setDriverOptions</i>	57
<i>GM_setEventFunc</i>	59
<i>GM_setExitFunc</i>	60
<i>GM_setGameLogicFunc</i>	61
<i>GM_setKeyDownFunc</i>	62
<i>GM_setKeyRepeatFunc</i>	63
<i>GM_setKeyUpFunc</i>	64
<i>GM_setMode</i>	65
<i>GM_setModeFilterFunc</i>	68
<i>GM_setModeSwitchFunc</i>	69
<i>GM_setMouseDownFunc</i>	71
<i>GM_setMouseMoveFunc</i>	72
<i>GM_setMouseUpFunc</i>	73
<i>GM_setPalette</i>	74

<i>GM_setPreModeSwitchFunc</i>	75
<i>GM_setSuspendAppCallback</i>	76
<i>GM_startOpenGL</i>	78
<i>GM_swapBuffers</i>	79
<i>GM_swapDirtyBuffers</i>	80
<i>LZTimerCount</i>	81
<i>LZTimerLap</i>	82
<i>LZTimerOff</i>	83
<i>LZTimerOn</i>	84
<i>MGL_FixDiv</i>	85
<i>MGL_FixMul</i>	86
<i>MGL_FixMulDiv</i>	88
<i>MGL_activatePalette</i>	89
<i>MGL_appActivate</i>	90
<i>MGL_availableBitmap</i>	91
<i>MGL_availableCursor</i>	92
<i>MGL_availableFont</i>	93
<i>MGL_availableIcon</i>	94
<i>MGL_availableJPEG</i>	95
<i>MGL_availableMemory</i>	96
<i>MGL_availableModes</i>	97
<i>MGL_availablePCX</i>	98
<i>MGL_availablePages</i>	99
<i>MGL_backfacing</i>	100
<i>MGL_beep</i>	101
<i>MGL_beginDirectAccess</i>	102
<i>MGL_beginPixel</i>	103
<i>MGL_bitBlt</i>	104
<i>MGL_bitBltCoord</i>	105
<i>MGL_bitBltLin</i>	108
<i>MGL_bitBltLinCoord</i>	109
<i>MGL_buildMonoMask</i>	111
<i>MGL_calloc</i>	113
<i>MGL_changeDisplayMode</i>	114
<i>MGL_charWidth</i>	116
<i>MGL_checkIdentityPalette</i>	117
<i>MGL_clearDevice</i>	118
<i>MGL_clearRegion</i>	119
<i>MGL_clearViewport</i>	120
<i>MGL_clipLineFX</i>	121
<i>MGL_computePixelAddr</i>	122
<i>MGL_copyRegion</i>	123
<i>MGL_createCustomDC</i>	124
<i>MGL_createDisplayDC</i>	125
<i>MGL_createLinearOffscreenDC</i>	128
<i>MGL_createMemoryDC</i>	129
<i>MGL_createOffscreenDC</i>	131
<i>MGL_createScrollingDC</i>	133

<i>MGL_createStereoDisplayDC</i>	135
<i>MGL_createWindowedDC</i>	138
<i>MGL_defRect</i>	140
<i>MGL_defRectPt</i>	141
<i>MGL_defaultAttributes</i>	142
<i>MGL_defaultColor</i>	143
<i>MGL_delay</i>	144
<i>MGL_destroyDC</i>	145
<i>MGL_detectGraph</i>	146
<i>MGL_diffRegion</i>	150
<i>MGL_diffRegionRect</i>	151
<i>MGL_disjointRect</i>	152
<i>MGL_divotSize</i>	153
<i>MGL_divotSizeCoord</i>	154
<i>MGL_doubleBuffer</i>	155
<i>MGL_drawBorder</i>	156
<i>MGL_drawBorderCoord</i>	157
<i>MGL_drawGlyph</i>	158
<i>MGL_drawHDivider</i>	159
<i>MGL_drawRegion</i>	160
<i>MGL_drawStr</i>	161
<i>MGL_drawStrXY</i>	162
<i>MGL_drawVDivider</i>	163
<i>MGL_driverName</i>	164
<i>MGL_ellipse</i>	165
<i>MGL_ellipseArc</i>	166
<i>MGL_ellipseArcCoord</i>	167
<i>MGL_ellipseArcEngine</i>	169
<i>MGL_ellipseCoord</i>	170
<i>MGL_ellipseEngine</i>	171
<i>MGL_emptyRect</i>	173
<i>MGL_emptyRegion</i>	174
<i>MGL_endDirectAccess</i>	175
<i>MGL_endPixel</i>	176
<i>MGL_equalPoint</i>	177
<i>MGL_equalRect</i>	178
<i>MGL_equalRegion</i>	179
<i>MGL_errorMsg</i>	180
<i>MGL_exit</i>	181
<i>MGL_fadePalette</i>	182
<i>MGL_fatalError</i>	184
<i>MGL_fclose</i>	185
<i>MGL_fillEllipse</i>	186
<i>MGL_fillEllipseArc</i>	187
<i>MGL_fillEllipseArcCoord</i>	188
<i>MGL_fillEllipseCoord</i>	189
<i>MGL_fillPolygon</i>	190
<i>MGL_fillPolygonCnvx</i>	191

<i>MGL_fillPolygonCnvxFX</i>	192
<i>MGL_fillPolygonFX</i>	193
<i>MGL_fillRect</i>	195
<i>MGL_fillRectCoord</i>	196
<i>MGL_fillRectPt</i>	197
<i>MGL_fopen</i>	198
<i>MGL_fread</i>	199
<i>MGL_free</i>	200
<i>MGL_freeRegion</i>	201
<i>MGL_fseek</i>	202
<i>MGL_ftell</i>	203
<i>MGL_fwrite</i>	204
<i>MGL_getActivePage</i>	205
<i>MGL_getArcCoords</i>	206
<i>MGL_getAspectRatio</i>	207
<i>MGL_getAttributes</i>	208
<i>MGL_getBackColor</i>	209
<i>MGL_getBitmapFromDC</i>	210
<i>MGL_getBitmapSize</i>	211
<i>MGL_getBitmapSizeExt</i>	212
<i>MGL_getBitsPerPixel</i>	213
<i>MGL_getBorderColors</i>	214
<i>MGL_getCP</i>	215
<i>MGL_getCharMetrics</i>	216
<i>MGL_getClipMode</i>	217
<i>MGL_getClipModeDC</i>	218
<i>MGL_getClipRect</i>	219
<i>MGL_getClipRectDC</i>	220
<i>MGL_getColor</i>	221
<i>MGL_getColorMapMode</i>	222
<i>MGL_getDefaultPalette</i>	223
<i>MGL_getDirectDrawActiveSurface</i>	224
<i>MGL_getDirectDrawObject</i>	225
<i>MGL_getDirectDrawOffscreenSurface</i>	226
<i>MGL_getDirectDrawPalette</i>	227
<i>MGL_getDirectDrawPrimarySurface</i>	228
<i>MGL_getDisplayStart</i>	229
<i>MGL_getDivot</i>	230
<i>MGL_getDivotCoord</i>	231
<i>MGL_getDriver</i>	232
<i>MGL_getFont</i>	233
<i>MGL_getFontMetrics</i>	234
<i>MGL_getFullScreenWindow</i>	235
<i>MGL_getGlyphHeight</i>	236
<i>MGL_getGlyphWidth</i>	237
<i>MGL_getHalfTonePalette</i>	238
<i>MGL_getHardwareFlags</i>	239
<i>MGL_getJPEGSize</i>	240

<i>MGL_getJPEGSizeExt</i>	241
<i>MGL_getLineStipple</i>	242
<i>MGL_getLineStippleCount</i>	243
<i>MGL_getLineStyle</i>	244
<i>MGL_getMarkerColor</i>	245
<i>MGL_getMarkerSize</i>	246
<i>MGL_getMarkerStyle</i>	247
<i>MGL_getMode</i>	248
<i>MGL_getPCXSize</i>	249
<i>MGL_getPCXSizeExt</i>	250
<i>MGL_getPalette</i>	251
<i>MGL_getPaletteEntry</i>	252
<i>MGL_getPaletteSize</i>	253
<i>MGL_getPaletteSnowLevel</i>	254
<i>MGL_getPenBitmapPattern</i>	255
<i>MGL_getPenPixmapPattern</i>	256
<i>MGL_getPenSize</i>	257
<i>MGL_getPenStyle</i>	258
<i>MGL_getPixel</i>	259
<i>MGL_getPixelCoord</i>	260
<i>MGL_getPixelCoordFast</i>	261
<i>MGL_getPixelFast</i>	262
<i>MGL_getPixelFormat</i>	263
<i>MGL_getPolygonType</i>	264
<i>MGL_getSpaceExtra</i>	265
<i>MGL_getTextDirection</i>	266
<i>MGL_getTextJustify</i>	267
<i>MGL_getTextSettings</i>	268
<i>MGL_getTextSize</i>	269
<i>MGL_getTickResolution</i>	270
<i>MGL_getTicks</i>	271
<i>MGL_getViewport</i>	272
<i>MGL_getViewportDC</i>	273
<i>MGL_getViewportOrg</i>	274
<i>MGL_getViewportOrgDC</i>	275
<i>MGL_getVisualPage</i>	276
<i>MGL_getWinDC</i>	277
<i>MGL_getWriteMode</i>	278
<i>MGL_getX</i>	279
<i>MGL_getY</i>	280
<i>MGL_glChooseVisual</i>	281
<i>MGL_glCreateContext</i>	282
<i>MGL_glDeleteContext</i>	284
<i>MGL_glEnumerateDrivers</i>	285
<i>MGL_glGetProcAddress</i>	286
<i>MGL_glGetVisual</i>	288
<i>MGL_glHaveHWOpenGL</i>	289
<i>MGL_glMakeCurrent</i>	290

<i>MGL_glRealizePalette</i>	291
<i>MGL_glResizeBuffers</i>	292
<i>MGL_glSetDriver</i>	293
<i>MGL_glSetOpenGLType</i>	294
<i>MGL_glSetPalette</i>	295
<i>MGL_glSetVisual</i>	296
<i>MGL_glSwapBuffers</i>	298
<i>MGL_globalToLocal</i>	299
<i>MGL_globalToLocalDC</i>	300
<i>MGL_halfTonePixel</i>	301
<i>MGL_halfTonePixel555</i>	302
<i>MGL_halfTonePixel565</i>	303
<i>MGL_haveWidePalette</i>	304
<i>MGL_init</i>	305
<i>MGL_initWindowed</i>	307
<i>MGL_insetRect</i>	309
<i>MGL_isCurrentDC</i>	310
<i>MGL_isDisplayDC</i>	311
<i>MGL_isMemoryDC</i>	312
<i>MGL_isSimpleRegion</i>	313
<i>MGL_isWindowedDC</i>	314
<i>MGL_leftTop</i>	315
<i>MGL_line</i>	316
<i>MGL_lineCoord</i>	317
<i>MGL_lineCoordFX</i>	318
<i>MGL_lineEngine</i>	319
<i>MGL_lineFX</i>	320
<i>MGL_lineRel</i>	321
<i>MGL_lineRelCoord</i>	322
<i>MGL_lineTo</i>	323
<i>MGL_lineToCoord</i>	324
<i>MGL_loadBitmap</i>	325
<i>MGL_loadBitmapExt</i>	327
<i>MGL_loadBitmapIntoDC</i>	328
<i>MGL_loadBitmapIntoDCExt</i>	330
<i>MGL_loadCursor</i>	331
<i>MGL_loadCursorExt</i>	332
<i>MGL_loadFont</i>	333
<i>MGL_loadFontExt</i>	334
<i>MGL_loadIcon</i>	335
<i>MGL_loadIconExt</i>	337
<i>MGL_loadJPEG</i>	338
<i>MGL_loadJPEGExt</i>	340
<i>MGL_loadJPEGIntoDC</i>	341
<i>MGL_loadJPEGIntoDCExt</i>	343
<i>MGL_loadPCX</i>	344
<i>MGL_loadPCXExt</i>	346
<i>MGL_loadPCXIntoDC</i>	347

<i>MGL_loadPCXIntoDCExt</i>	349
<i>MGL_localToGlobal</i>	350
<i>MGL_localToGlobalDC</i>	351
<i>MGL_makeCurrentDC</i>	352
<i>MGL_makeSubDC</i>	353
<i>MGL_malloc</i>	354
<i>MGL_mapToPalette</i>	355
<i>MGL_marker</i>	356
<i>MGL_maxCharWidth</i>	357
<i>MGL_maxColor</i>	358
<i>MGL_maxPage</i>	359
<i>MGL_maxx</i>	360
<i>MGL_maxxDC</i>	361
<i>MGL_maxy</i>	362
<i>MGL_maxyDC</i>	363
<i>MGL_memcpy</i>	364
<i>MGL_memcpyVIRTDEST</i>	365
<i>MGL_memcpyVIRTSRC</i>	366
<i>MGL_memset</i>	367
<i>MGL_memsetl</i>	368
<i>MGL_memsetw</i>	369
<i>MGL_mirrorGlyph</i>	370
<i>MGL_modeDriverName</i>	371
<i>MGL_modeFlags</i>	372
<i>MGL_modeName</i>	373
<i>MGL_modeResolution</i>	374
<i>MGL_moveRel</i>	375
<i>MGL_moveRelCoord</i>	376
<i>MGL_moveTo</i>	377
<i>MGL_moveToCoord</i>	378
<i>MGL_newRegion</i>	379
<i>MGL_offsetRect</i>	380
<i>MGL_offsetRegion</i>	381
<i>MGL_optimizeRegion</i>	382
<i>MGL_packColor</i>	383
<i>MGL_packColorFast</i>	384
<i>MGL_packColorRGB</i>	385
<i>MGL_packColorRGBFast</i>	386
<i>MGL_packColorRGBFast2</i>	387
<i>MGL_pixel</i>	388
<i>MGL_pixelCoord</i>	389
<i>MGL_pixelCoordFast</i>	390
<i>MGL_pixelFast</i>	391
<i>MGL_polyLine</i>	392
<i>MGL_polyMarker</i>	393
<i>MGL_polyPoint</i>	394
<i>MGL_ptInRect</i>	395
<i>MGL_ptInRectCoord</i>	396

<i>MGL_ptInRegion</i>	397
<i>MGL_ptInRegionCoord</i>	398
<i>MGL_putBitmap</i>	399
<i>MGL_putBitmapMask</i>	400
<i>MGL_putBitmapSection</i>	401
<i>MGL_putBitmapTransparent</i>	402
<i>MGL_putBitmapTransparentSection</i>	404
<i>MGL_putDivot</i>	406
<i>MGL_putIcon</i>	407
<i>MGL_putMonoImage</i>	408
<i>MGL_random</i>	409
<i>MGL_randoml</i>	410
<i>MGL_realColor</i>	411
<i>MGL_realizePalette</i>	412
<i>MGL_rect</i>	413
<i>MGL_rectCoord</i>	414
<i>MGL_rectPt</i>	415
<i>MGL_registerAllDispDrivers</i>	416
<i>MGL_registerAllDispDriversExt</i>	417
<i>MGL_registerAllMemDrivers</i>	419
<i>MGL_registerAllOpenGLDrivers</i>	420
<i>MGL_registerDriver</i>	421
<i>MGL_registerEventProc</i>	424
<i>MGL_registerFullScreenWindow</i>	425
<i>MGL_resizeWinDC</i>	426
<i>MGL_restoreAttributes</i>	427
<i>MGL_result</i>	428
<i>MGL_resume</i>	429
<i>MGL_rgbBlue</i>	430
<i>MGL_rgbColor</i>	431
<i>MGL_rgbGreen</i>	432
<i>MGL_rgbRed</i>	433
<i>MGL_rgnEllipse</i>	434
<i>MGL_rgnEllipseArc</i>	435
<i>MGL_rgnGetArcCoords</i>	436
<i>MGL_rgnLine</i>	437
<i>MGL_rgnLineCoord</i>	438
<i>MGL_rgnLineCoordFX</i>	439
<i>MGL_rgnLineFX</i>	440
<i>MGL_rgnSolidEllipse</i>	441
<i>MGL_rgnSolidEllipseArc</i>	442
<i>MGL_rgnSolidRect</i>	443
<i>MGL_rgnSolidRectCoord</i>	444
<i>MGL_rgnSolidRectPt</i>	445
<i>MGL_rightBottom</i>	446
<i>MGL_rotateGlyph</i>	447
<i>MGL_rotatePalette</i>	448
<i>MGL_saveBitmapFromDC</i>	449

<i>MGL_saveJPEGFromDC</i>	450
<i>MGL_savePCXFromDC</i>	452
<i>MGL_scanLeftForColor</i>	454
<i>MGL_scanLeftWhileColor</i>	455
<i>MGL_scanLine</i>	456
<i>MGL_scanRightForColor</i>	457
<i>MGL_scanRightWhileColor</i>	458
<i>MGL_sectRect</i>	459
<i>MGL_sectRectCoord</i>	460
<i>MGL_sectRectFast</i>	461
<i>MGL_sectRectFastCoord</i>	462
<i>MGL_sectRegion</i>	463
<i>MGL_sectRegionRect</i>	464
<i>MGL_setACCELDriver</i>	465
<i>MGL setActivePage</i>	466
<i>MGL_setAppInstance</i>	467
<i>MGL_setAspectRatio</i>	468
<i>MGL_setBackColor</i>	469
<i>MGL_setBlueCodeIndex</i>	470
<i>MGL_setBorderColors</i>	471
<i>MGL_setBufSize</i>	472
<i>MGL_setClipMode</i>	473
<i>MGL_setClipModeDC</i>	474
<i>MGL_setClipRect</i>	475
<i>MGL_setClipRectDC</i>	476
<i>MGL setColor</i>	477
<i>MGL setColorCI</i>	478
<i>MGL setColorMapMode</i>	479
<i>MGL setColorRGB</i>	480
<i>MGL setDefaultPalette</i>	481
<i>MGL setDisplayStart</i>	482
<i>MGL setFileIO</i>	484
<i>MGL setLineStipple</i>	485
<i>MGL setLineStippleCount</i>	486
<i>MGL setLineStyle</i>	487
<i>MGL setMainWindow</i>	489
<i>MGL setMarkerColor</i>	490
<i>MGL setMarkerSize</i>	491
<i>MGL setMarkerStyle</i>	492
<i>MGL setOpenGLFuncs</i>	493
<i>MGL setPalette</i>	494
<i>MGL setPaletteEntry</i>	496
<i>MGL setPaletteSnowLevel</i>	497
<i>MGL setPenBitmapPattern</i>	498
<i>MGL setPenPixmapPattern</i>	499
<i>MGL setPenSize</i>	500
<i>MGL setPenStyle</i>	501
<i>MGL setPolygonType</i>	502

<i>MGL_setRelViewport</i>	503
<i>MGL_setRelViewportDC</i>	504
<i>MGL_setResult</i>	505
<i>MGL_setSpaceExtra</i>	506
<i>MGL_setStereoSyncType</i>	507
<i>MGL_setSuspendAppCallback</i>	508
<i>MGL_setTextDirection</i>	510
<i>MGL_setTextJustify</i>	511
<i>MGL_setTextSettings</i>	512
<i>MGL_setTextSize</i>	513
<i>MGL_setUserEventFilter</i>	514
<i>MGL_setViewport</i>	516
<i>MGL_setViewportDC</i>	517
<i>MGL_setViewportOrg</i>	518
<i>MGL_setViewportOrgDC</i>	519
<i>MGL_setVisualPage</i>	520
<i>MGL_setWinDC</i>	522
<i>MGL_setWriteMode</i>	523
<i>MGL_singleBuffer</i>	524
<i>MGL_sizeX</i>	525
<i>MGL_sizeY</i>	526
<i>MGL_srand</i>	527
<i>MGL_startStereo</i>	528
<i>MGL_stopStereo</i>	529
<i>MGL_stretchBLT</i>	530
<i>MGL_stretchBitmap</i>	531
<i>MGL_stretchBitmapSection</i>	532
<i>MGL_stretchBltCoord</i>	533
<i>MGL_surfaceAccessType</i>	535
<i>MGL_suspend</i>	536
<i>MGL_swapBuffers</i>	537
<i>MGL_textBounds</i>	538
<i>MGL_textHeight</i>	539
<i>MGL_textWidth</i>	540
<i>MGL_transBlt</i>	541
<i>MGL_transBltCoord</i>	542
<i>MGL_transBltLin</i>	544
<i>MGL_transBltLinCoord</i>	545
<i>MGL_traverseRegion</i>	547
<i>MGL_underScoreLocation</i>	548
<i>MGL_unionRect</i>	549
<i>MGL_unionRectCoord</i>	550
<i>MGL_unionRegion</i>	551
<i>MGL_unionRegionOfs</i>	552
<i>MGL_unionRegionRect</i>	553
<i>MGL_unloadBitmap</i>	554
<i>MGL_unloadCursor</i>	555
<i>MGL_unloadFont</i>	556

<i>MGL_unloadIcon</i>	557
<i>MGL_unpackColor</i>	558
<i>MGL_unpackColorFast</i>	559
<i>MGL_unpackColorRGB</i>	560
<i>MGL_unpackColorRGBFast</i>	561
<i>MGL_unregisterAllDrivers</i>	562
<i>MGL_useEvents</i>	563
<i>MGL_useFont</i>	564
<i>MGL_useLocalMalloc</i>	565
<i>MGL_vSync</i>	566
<i>MGL_vecFontEngine</i>	567
<i>MGL_zbufferAccessType</i>	569
<i>MS_available</i>	570
<i>MS_drawCursor</i>	571
<i>MS_getPos</i>	572
<i>MS_hide</i>	573
<i>MS_moveTo</i>	574
<i>MS_obscure</i>	575
<i>MS_setCursor</i>	576
<i>MS_setCursorColor</i>	577
<i>MS_show</i>	578
<i>SPR_draw</i>	579
<i>SPR_mgrAddOpaqueBitmap</i>	580
<i>SPR_mgrAddTransparentBitmap</i>	581
<i>SPR_mgrEmpty</i>	583
<i>SPR_mgrExit</i>	584
<i>SPR_mgrInit</i>	585
<i>SPR_mgrOffscreenCacheFull</i>	587
<i>SPR_mgrReloadHW</i>	588
<i>SPR_mgrUsingOffscreenDC</i>	589
<i>SVGA_setBank</i>	590
<i>SVGA_setBankC</i>	591
<i>ULZElapsedTime</i>	592
<i>ULZReadTime</i>	593
<i>ULZTimerCount</i>	594
<i>ULZTimerLap</i>	595
<i>ULZTimerOff</i>	596
<i>ULZTimerOn</i>	597
<i>ULZTimerResolution</i>	598
<i>ZTimerInit</i>	599
Data Structure Reference	600
<i>CPU_largeInteger</i>	600
<i>CPU_processorType</i>	601
<i>GMDC</i>	602
<i>GM_driverOptions</i>	604
<i>GM_keyCodes</i>	605
<i>GM_modeFlagsType</i>	606
<i>GM_modeInfo</i>	607

<i>GM_stretchType</i>	608
<i>MGLDC</i>	609
<i>MGLVisual</i>	612
<i>MGL_COLORS</i>	613
<i>MGL_WIN_COLORS</i>	614
<i>MGL_bdrStyleType</i>	615
<i>MGL_clipType</i>	616
<i>MGL_colorModes</i>	617
<i>MGL_driverType</i>	618
<i>MGL_errorType</i>	620
<i>MGL_eventMaskType</i>	622
<i>MGL_eventModMaskType</i>	623
<i>MGL_eventMsgMaskType</i>	624
<i>MGL_eventType</i>	625
<i>MGL_fontType</i>	626
<i>MGL_glContextFlagsType</i>	627
<i>MGL_glOpenGLType</i>	629
<i>MGL_hardwareFlagsType</i>	630
<i>MGL_lineStyleType</i>	632
<i>MGL_markerStyleType</i>	633
<i>MGL_modeFlagsType</i>	634
<i>MGL_modeType</i>	637
<i>MGL_palRotateType</i>	640
<i>MGL_penStyleType</i>	641
<i>MGL_polygonType</i>	642
<i>MGL_refreshRateType</i>	643
<i>MGL_stereoBufType</i>	644
<i>MGL_stereoSyncType</i>	645
<i>MGL_surfaceAccessFlagsType</i>	646
<i>MGL_suspendAppCodesType</i>	647
<i>MGL_suspendAppFlagsType</i>	649
<i>MGL_textDirType</i>	650
<i>MGL_textJustType</i>	651
<i>MGL_waitVRTFlagType</i>	652
<i>MGL_writeModeType</i>	653
<i>arc_coords_t</i>	654
<i>attributes_t</i>	655
<i>bitmap_t</i>	657
<i>color_t</i>	659
<i>cursor_t</i>	660
<i>event_t</i>	661
<i>fileio_t</i>	663
<i>fix32_t</i>	664
<i>font_t</i>	665
<i>fxpoint_t</i>	667
<i>gmode_t</i>	668
<i>icon_t</i>	670
<i>metrics_t</i>	671

<i>palette_t</i>	673
<i>pattern_t</i>	674
<i>pixel_format_t</i>	675
<i>pixmap_t</i>	678
<i>point_t</i>	679
<i>rect_t</i>	680
<i>region_t</i>	681
<i>text_settings_t</i>	682
Index	683

MGL Library Overview

This document contains the reference manual for the SciTech MGL Graphics Library and associated supplemental libraries. Please consult the [MGL Getting Started and Programmer's Guide](#) for more information on programming with MGL.

All functions in MGL work with an MGL device context (MGLDC), which can either be a specific device context passed to the function, or the currently active device context. All routines that take a specific MGLDC pointer, don't work with the values in the currently active device context, which means that these routines can be used to update a different device context than the currently active one. However if the device context passed is actually the currently active context, all the changes made to the specific device context will also be made to the currently active context as well.

Most of the routines in MGL are bound to the currently active device context, which can be changed with the `MGL_makeCurrentDC` function. When a context is made the currently active context, the values in the device context are *cached* in a special global data area for speed. Hence you should *not* change the values in a device context directly if that context is currently active, but should use the proper MGL functions to do so. When the active device context is changed to a new device context, the values in the currently cached device context are then flushed back to the original device context.

Because there is a lot of copying going on when changing active device context, this is something that you would not normally do between rasterizing individual primitives. Normally you will have one main device context for all your rasterizing output (either a system buffer or a display device context) that you will be rasterizing into, and this context will generally not change. MGL will however not perform any copying if you attempt to set the active device context to the same device context.

You can also set the active device context to nothing, by passing a NULL to `MGL_makeCurrentDC`. If you explicitly destroy a device context, you must ensure that that device context is *not* the currently active device context!

Environment detection and initialization

The following functions are used to detect the installed hardware, initialize MGL and obtain information about a particular device. Most of the functions listed below will be used by the application to tailor it's use of MGL functions depending on the underlying system configuration, and to provide system information to the end user.

MGL_availableMemory	MGL_availableModes
MGL_availablePages	MGL_changeDisplayMode
MGL_detectGraph	MGL_driverName
MGL_errorMsg	MGL_exit
MGL_fatalError	MGL_getDriver
MGL_getHardwareFlags	MGL_getMode
MGL_init	MGL_isDisplayDC
MGL_isMemoryDC	MGL_isWindowedDC
MGL_modeDriverName	MGL_modeName
MGL_modeResolution	MGL_registerAllDispDrivers
MGL_registerAllMemDrivers	MGL_registerDriver
MGL_result	MGL_setBufSize
MGL_setFileIO	MGL_setResult
MGL_surfaceAccessType	MGL_unregisterAllDrivers
MGL_zbufferAccessType	

Event Handling

The following 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 application to easily combine the keyboard and mouse into a single system for interacting with the user.

EVT_asciiCode	EVT_flush
EVT_getNext	EVT_halt
EVT_peekNext	EVT_post
EVT_repeatCount	EVT_scanCode
EVT_setTimerTick	

Mouse Handling

The following functions allow the application to fully control the mouse cursor, such as showing and hiding the cursor and changing it's shape and color.

MS_available	MS_drawCursor
MS_getPos	MS_hide
MS_moveTo	MS_obscure
MS_setCursor	MS_setCursorColor
MS_show	

Device context information and manipulation

The following functions provide the API routines that are used to change the attributes of the currently active device context, such as the current foreground and background colors, and the current pen pattern. These routines also provide information about device contexts, such as the resolution, pixel depth, aspect ratio etc.

Note that some of these functions are bound to a specific device context, and if this device context is not the currently active context, the active context will not be affected.

Note that all MGL color values are passed to MGL as either a color index or a packed RGB value, packed for the correct format expected by the device context. Rather than setting the color value directory, you can use the utility functions MGL_setColorCI and MGL_setColorRGB to set the color given a color index or an RGB color value. When running in RGB modes, the MGL_setColorCI will convert the color index to the proper MGL color value using the devices color lookup table. When running in color index modes, the MGL_setColorRGB will set the color to the color index with the color palette entry that is the closest to the passed in RGB color value.

MGL_defaultAttributes	MGL_defaultColor
MGL_getAspectRatio	MGL_getAttributes
MGL_getBackColor	MGL_getBitsPerPixel
MGL_getColor	MGL_getColorMapMode
MGL_getLineStipple	MGL_getLineStippleCount
MGL_getLineStyle	MGL_getMarkerColor
MGL_getMarkerSize	MGL_getMarkerStyle

MGL_getPenBitmapPattern	MGL_getPenPixmapPattern
MGL_getPenSize	MGL_getPenStyle
MGL_getPixelFormat	MGL_getPolygonType
MGL_getWriteMode	MGL_isCurrentDC
MGL_makeCurrentDC	MGL_maxColor
MGL_maxPage	MGL_restoreAttributes
MGL_setAspectRatio	MGL_setBackColor
MGL_setColor	MGL_setColorCI
MGL_setColorMapMode	MGL_setColorRGB
MGL_setLineStipple	MGL_setLineStippleCount
MGL_setLineStyle	MGL_setMarkerColor
MGL_setMarkerSize	MGL_setMarkerStyle
MGL_setPenBitmapPattern	MGL_setPenPixmapPattern
MGL_setPenSize	MGL_setPenStyle
MGL_setPolygonType	MGL_setWriteMode
MGL_sizeX	MGL_sizeY

Color and palette manipulation

The following functions provide full control over colors in MGL. MGL can be running in either color index modes (4 and 8 bits per pixel) and RGB modes (15 bits and above TrueColor modes, and also RGB 8 bit dithered modes). In color index modes, the final color of a pixel is determined by the color lookup table or palette associated with the device context. MGL provides functions for setting and retrieving color values in the color lookup tables for a device context, and for realizing the color palette on hardware devices.

Note that *all* device contexts have a color lookup table, even memory device contexts and RGB display device contexts. MGL will convert color values in 4 and 8 bit memory devices on the fly when BitBlt'ing to any other display device context with 8 bits or more of pixel depth (finding the closest color if an exact match is not found, and using the color lookup table for RGB devices). In order for maximum BitBlt performance, you must ensure that the color table in the memory device is identical to the color table in the display device, in which case no bitmap translation will be performed. If you know that all your bitmaps have identity palettes, you can use the function MGL_checkIdentityPalette to turn on and off this checking code to obtain higher performance. Note however that this does *not* turn off identity palette checking when BitBlt'ing to a windowed device context under

Windows, as this cannot be turned off.

Also provided are functions and macros to pack RGB color values into proper MGL color values to be passed the to MGL. All MGL color values are passed to MGL as either a color index or a packed RGB value, packed for the correct format expected by the device context. The MGL_pack* family of functions is used to pack color values from 8 bit RGB tuples into the proper MGL packed RGB color values.

MGL_checkIdentityPalette	MGL_fadePalette
MGL_getDefaultPalette	MGL_getPalette
MGL_getPaletteEntry	MGL_getPaletteSize
MGL_mapToPalette	MGL_packColor
MGL_packColorFast	MGL_packColorRGB
MGL_packColorRGBFast	MGL_packColorRGBFast2
MGL_realColor	MGL_realizePalette
MGL_rgbBlue	MGL_rgbColor
MGL_rgbGreen	MGL_rgbRed
MGL_rotatePalette	MGL_setDefaultPalette
MGL_setPalette	MGL_setPaletteEntry
MGL_unpackColor	MGL_unpackColorFast
MGL_unpackColorRGB	MGL_unpackColorRGBFast

Device clearing

The following functions provide for clearing the entire device or just the current viewport to the background color.

MGL_clearDevice	MGL_clearViewport
-----------------	-------------------

Game Framework

The Game Framework abstracts most of the code overhead required to write full-screen and windowed applications. Using the Game Framework frees you from the tedium of writing windows classes and allows you to concentrate on what's really important, your game. The Game Framework library consists of the following functions.

GM_chooseMode	GM_setDrawFunc
GM_cleanup	GM_setDriverOptions

GM_exit	GM_setEventFunc
GM_findMode	GM_setExitFunc
GM_getDoDraw	GM_setGameLogicFunc
GM_getExitMainLoop	GM_setKeyDownFunc
GM_getHaveWin31	GM_setKeyRepeatFunc
GM_getHaveWin95	GM_setKeyUpFunc
GM_getHaveWinNT	GM_setMode
GM_init	GM_setModeFilterFunc
GM_initPath	GM_setModeSwitchFunc
GM_initSysPalNoStatic	GM_setMouseDownFunc
GM_initWindowPos	GM_setMouseMoveFunc
GM_mainLoop	GM_setMouseUpFunc
GM_processEvents	GM_setPalette
GM_processEventsWin	GM_setPreModeSwitchFunc
GM_realizePalette	GM_setSuspendAppCallback
GM_registerEventProc	GM_startOpenGL
GM_registerMainWindow	GM_swapBuffers
GM_setAppActivate	

Sprite Manager

The Sprite Manager provides a seamless interface for using offscreen video memory to store sprites and other bitmaps for extremely fast sprite animation. The Sprite Manager can virtualize framebuffers, and provides for support for rectangular and linear framebuffering. The Sprite Manager library is comprised of these functions:

SPR_draw	SPR_mgrExit
SPR_mgrAddOpaqueBitmap	SPR_mgrInit
SPR_mgrAddOpaqueBitmap	SPR_mgrOffscreenCacheFull
SPR_mgrAddTransparentBitmap	SPR_mgrReloadHW
SPR_mgrEmpty	SPR_mgrUsingOffscreenDC

OpenGL Rendering

The SciTech MGL includes complete hardware and software support for OpenGL rendering in full-screen and windowed graphics modes. OpenGL support is encapsulated in the following functions:

MGL_glChooseVisual	MGL_glRealizePalette
--------------------	----------------------

MGL_glCreateContext	MGL_glResizeBuffers
MGL_glDeleteContext	MGL_glSetDriver
MGL_glEnumerateDrivers	MGL_glSetOpenGLType
MGL_glGetProcAddress	MGL_glSetPalette
MGL_glGetVisual	MGL_glSetVisual
MGL_glHaveHWOOpenGL	MGL_glSwapBuffers
MGL_glMakeCurrent	

Viewport and clip rectangle manipulation

The following functions provide the ability to change the current viewport that is used to display all subsequent drawing operations, and to change the current clipping rectangle within the current viewport. The clip rectangle is always set in local viewport coordinates, and can never be larger than the current viewport.

Also provided are functions for converting coordinates between global screen coordinates and local viewport coordinates, and for obtaining the dimensions of the current viewport.

MGL_getClipMode	MGL_getClipModeDC
MGL_getClipRect	MGL_getClipRectDC
MGL_getViewport	MGL_getViewportDC
MGL_getViewportOrg	MGL_getViewportOrgDC
MGL_globalToLocal	MGL_globalToLocalDC
MGL_localToGlobal	MGL_localToGlobalDC
MGL_maxx	MGL_maxxDC
MGL_maxy	MGL_maxyDC
MGL_setClipMode	MGL_setClipModeDC
MGL_setClipRect	MGL_setClipRectDC
MGL_setRelViewport	MGL_setRelViewportDC
MGL_setViewport	MGL_setViewportDC
MGL_setViewportOrg	MGL_setViewportOrgDC

Pixel plotting

The following functions provide support for displaying single pixels, and for retrieving the color of single pixels for a device context. The *Fast* style functions require that you call the MGL_beginPixel function to set the hardware for drawing multiple pixels as fast as possible.

MGL_beginPixel	MGL_endPixel
MGL_getPixel	MGL_getPixelCoord
MGL_getPixelCoordFast	MGL_getPixelFast
MGL_pixel	MGL_pixelCoord
MGL_pixelCoordFast	MGL_pixelFast

Scanline color scanning

The following functions provide support for building high performance floodfill type operations on device contexts. They provide the ability to scan through a device context searching for specific color values or color changes on a scanline by scanline basis, and are implemented as fast as possible for each display device context.

MGL_scanLeftForColor	MGL_scanLeftWhileColor
MGL_scanRightForColor	MGL_scanRightWhileColor

High performance drawing interface functions

The following functions put the hardware into special high speed drawing states for drawing various types of primitives. All of the MGL *Fast* style functions require that one of the functions be called before the fast drawing code is called, and eliminate time consuming hardware setup code in the low level rasterizing code for the device context. Generally you should try to draw as many primitives of the same type between calls to the begin and end functions.

MGL_beginDirectAccess	MGL_endDirectAccess
-----------------------	---------------------

Line drawing and clipping

The following functions provide support for drawing lines, and clipping lines to a fixed point clipping rectangle. MGL provides support for rasterizing lines with integer endpoints or with fixed point endpoints for maximum precision. MGL will actually rasterize all lines with 16.16 precision internally, allowing for the line endpoints to lie on non-integer pixel boundaries, so the integer line drawing routines simply end up calling the fixed point routines, so you should always try to use the fixed point routines where possible for maximum performance.

MGL_clipLineFX	MGL_getCP
----------------	-----------

MGL_getX	MGL_getY
MGL_line	MGL_lineCoord
MGL_lineCoordFX	MGL_lineEngine
MGL_lineFX	MGL_lineRel
MGL_lineRelCoord	MGL_lineTo
MGL_lineToCoord	MGL_moveRelCoord
MGL_moveRel	MGL_moveTo
MGL_moveToCoord	MGL_scanLine

Polyline, marker and pixel drawing

The following functions provide support for drawing sets of integer coordinate lines, markers and pixels. Markers in MGL are small circles, rectangles or crosses used to mark elements in 2D charts.

MGL_marker	MGL_polyLine
MGL_polyMarker	MGL_polyPoint

Polygon drawing

The following polygon routines come in two flavors. One takes vertex coordinates in integer format (16 or 32 bit integers depending on memory model) and the other takes vertex coordinates in 16.16 fixed point format. The fixed point format version is always the fastest, and in fact the integer versions simply convert all the vertices to fixed points and call the fixed point routine. The integer versions are mainly still in the API for compatibility with the MGL 1.x releases. If you are writing new code, or porting old code, you should always used the fixed point polygon rasterizing routines.

MGL_fillPolygon	MGL_fillPolygonCnvx
MGL_fillPolygonCnvxFX	MGL_fillPolygonFX

Rectangle drawing

The following functions provide support for drawing outlined rectangles and solid rectangles filled with the current pen pattern.

MGL_fillRect	MGL_fillRectCoord
--------------	-------------------

MGL_fillRectPt
MGL_rectCoord

MGL_rect
MGL_rectPt

Ellipse drawing

The following functions provide support for drawing ellipses and elliptical arcs, either as outlines or solid filled with the current pen pattern. Note that MGL provides the ability to draw ellipses given a center coordinate and integer radii, or by providing a bounding rectangle for the ellipse. The bounding rectangle versions allow you to draw any sized ellipse, some of which will have the center coordinate located on a non-integer half-pixel boundary.

MGL_ellipse
MGL_ellipseArcCoord
MGL_ellipseCoord
MGL_fillEllipse
MGL_fillEllipseArcCoord
MGL_getArcCoords

MGL_ellipseArc
MGL_ellipseArcEngine
MGL_ellipseEngine
MGL_fillEllipseArc
MGL_fillEllipseCoord

Text attribute manipulation

The following functions provide the ability to control text rasterizing attributes such as the horizontal and vertical justification, text direction and text size (for vector fonts only). Also provided are functions for measuring the size of text in the current font for appropriate character placement computations.

MGL_charWidth
MGL_getFontMetrics
MGL_getTextDirection
MGL_getTextSettings
MGL_maxCharWidth
MGL_setTextDirection
MGL_setTextSettings
MGL_textBounds
MGL_textWidth

MGL_getCharMetrics
MGL_getSpaceExtra
MGL_getTextJustify
MGL_getTextSize
MGL_setSpaceExtra
MGL_setTextJustify
MGL_setTextSize
MGL_textHeight
MGL_underScoreLocation

Text drawing

The following functions provide the ability to draw text in the current text attributes, such as the justification and direction.

The `MGL_vecFontEngine` function allows you to rasterize vector fonts by calling your own drawing routines, which can be used to do things like pass the vertices for vector fonts through 2D and 3D transformations to draw 2D and 3D transformed text.

<code>MGL_drawStr</code>	<code>MGL_drawStrXY</code>
<code>MGL_getFont</code>	<code>MGL_useFont</code>
<code>MGL_vecFontEngine</code>	

BitBlt support

The following functions provide support for copying blocks of image data between device contexts, or between areas of the same device context. The data can be copied using a simple solid copy, with arbitrary stretching and shrinking of the data to a different-sized destination rectangle and with source transparency for drawing transparent sprites for animation.

SciTech MGL also fully supports rasterizing of monochrome bitmaps, which can be used for fast masking operations and for rasterizing custom monochrome text.

<code>MGL_bitBlt</code>	<code>MGL_bitBltCoord</code>
<code>MGL_divotSize</code>	<code>MGL_divotSizeCoord</code>
<code>MGL_getDivot</code>	<code>MGL_getDivotCoord</code>
<code>MGL_putBitmap</code>	<code>MGL_putBitmapMask</code>
<code>MGL_putBitmapSection</code>	<code>MGL_putBitmapTransparent</code>
<code>MGL_putBitmapTransparentSection</code>	<code>MGL_putDivot</code>
<code>MGL_putIcon</code>	<code>MGL_putMonoImage</code>
<code>MGL_stretchBitmap</code>	<code>MGL_stretchBitmapSection</code>
<code>MGL_stretchBLT</code>	<code>MGL_stretchBltCoord</code>
<code>MGL_transBlt</code>	<code>MGL_transBltCoord</code>

Linear OffscreenDC BitBlt support

The following functions are specialized functions for performing hardware accelerated BitBlt operations between linear offscreen memory DC's and the display DC. Many hardware devices do not support linear offscreen

memory, but can only support the normal rectangular offscreen display device contexts.

See Platform Specific: MGLWIN for more information on linear offscreen devices.

MGL_bitBltLin
MGL_transBltLin

MGL_bitBltLinCoord
MGL_transBltLinCoord

Monochrome bitmap manipulation

The following functions provide support for drawing monochrome glyphs (used to implement custom text rasterizing routines) and performing rotations and other operations on monochrome bitmaps.

MGL_buildMonoMask
MGL_getGlyphHeight
MGL_mirrorGlyph

MGL_drawGlyph
MGL_getGlyphWidth
MGL_rotateGlyph

Double buffering support

The following functions provide support for hardware double buffering on display devices that have two display buffers. Hardware double buffering is used to achieve flicker free animation, but drawing to a hidden portion of display memory and then instantly updating the screen by changing the visible display page to the previously hidden display page.

MGL_doubleBuffer
MGL_getVisualPage
MGL_setVisualPage
MGL_swapBuffers

MGL_getActivePage
MGL_setActivePage
MGL_singleBuffer

Hardware scrolling or panning

The following functions provide the capability to scroll around within a hardware scrolling or panning device context. For combining hardware scrolling and double buffering, you can use MGL_setDisplayStart and

MGL_setVisualPage together.

MGL_getDisplayStart

MGL_setDisplayStart

Region management

The following functions provide support for managing complex regions, and generating new complex region primitives. Complex regions are used to represent 2D arbitrarily complex regions as unions of smaller rectangles, and can represent shapes with complex outlines, holes in the middle and even totally disjoint areas. These routines allow you to create, copy, free and draw such regions, as well as generate regions with specific shapes that can be combined with other regions to produce more complex shapes.

You can also traverse a complex region, which allows you to call a particular function for every rectangle in the union of rectangles that make up the complex region. This allows you to use complex regions to maintain visible update areas and complex clipping regions for building high performance GUI user interface libraries running on top of MGL.

MGL_clearRegion

MGL_drawRegion

MGL_newRegion

MGL_rgnEllipse

MGL_rgnGetArcCoords

MGL_rgnLineCoord

MGL_rgnLineFX

MGL_rgnSolidEllipseArc

MGL_rgnSolidRectCoord

MGL_traverseRegion

MGL_copyRegion

MGL_freeRegion

MGL_optimizeRegion

MGL_rgnEllipseArc

MGL_rgnLine

MGL_rgnLineCoordFX

MGL_rgnSolidEllipse

MGL_rgnSolidRect

MGL_rgnSolidRectPt

Region algebra

The following functions provide support for performing Boolean arithmetic operations on complex regions to produce new complex regions. These Boolean arithmetic routines form the heart of the MGL complex region support, and allow you to construct arbitrarily complex shapes from the simple region primitives that MGL can pre-generate for you.

MGL_diffRegion

MGL_diffRegionRect

MGL_emptyRegion	MGL_equalRegion
MGL_isSimpleRegion	MGL_offsetRegion
MGL_ptInRegionCoord	MGL_sectRegion
MGL_sectRegionRect	MGL_unionRegion
MGL_unionRegionOfs	MGL_unionRegionRect

RGB to 8 bit halftone dithering routines

The following functions allow you to obtain a copy of an MGL halftone palette, which is used for RGB to 8 bit real time dithering, and for performing the dither operation on a single pixel so you can implement your own real-time dithering routines.

MGL_getHalfTonePalette	MGL_halfTonePixel
------------------------	-------------------

Resource loading/unloading

The following functions provide support for loading and unloading MGL resources, such as bitmaps, icons, cursors and fonts. MGL can load standard Windows 3.1 bitmaps, icons, cursor and font files, so you can use your favorite Windows resource editors to build these for your application programs.

MGL_availableBitmap	MGL_availableCursor
MGL_availableFont	MGL_availableIcon
MGL_availablePCX	MGL_getBitmapSize
MGL_getBitmapFromDC	MGL_getPCXSize
MGL_loadBitmap	MGL_loadPCX
MGL_loadBitmapIntoDC	MGL_loadCursor
MGL_loadFont	MGL_loadIcon
MGL_loadPCXIntoDC	MGL_saveBitmapFromDC
MGL_savePCXFromDC	MGL_unloadBitmap
MGL_unloadCursor	MGL_unloadFont
MGL_unloadIcon	

Rectangle and Point manipulation

The following functions provide support for performing Boolean operations on simple rectangles. Note that some operations are not exact, and sometimes the result cannot be properly represented as a single rectangle. If you need to obtain the proper result, you will need to use the complex

region routines.

MGL_defRect	MGL_defRectPt
MGL_disjointRect	MGL_emptyRect
MGL_equalPoint	MGL_equalRect
MGL_insetRect	MGL_leftTop
MGL_offsetRect	MGL_ptInRect
MGL_ptInRectCoord	MGL_ptInRegion
MGL_insetRect	MGL_rightBottom
MGL_sectRectCoord	MGL_unionRect
MGL_unionRectCoord	

Random number generation routines

The following functions provide support for generating *fast* random numbers with either 16 or 32 bits of range.

MGL_random	MGL_randoml
MGL_srand	

Fixed point utility routines

The following functions provide support for performing simple fixed point arithmetic. MGL passes most coordinates to drawing routines in 16.16 fixed point format, and these routines provide support for fast multiplication and division operations with these numbers. If you need more functionality for fixed point arithmetic, contact SciTech Software for more information about our full featured fixed point library.

MGL_backfacing	MGL_FixDiv
MGL_FixMul	MGL_FixMulDiv

Memory allocation and clearing

The following routines provide support for allocating, freeing and clearing memory blocks that are larger than 64Kb in size, and work properly even in 16 bit real mode code.

MGL_calloc	MGL_free
MGL_malloc	MGL_memcpy

MGL_memcpyVIRTDST
MGL_memset
MGL_memsetw

MGL_memcpyVIRTSRC
MGL_memsetl
MGL_useLocalMalloc

Platform Specific: MGLDOS

The following functions are specific to the MSDOS version of MGL. Most of the functions found in the MGLDOS specific bindings are also present in the other platform specific bindings, so you can write common code that will compile for any of the supported platforms.

Note that MGL provides support for two different types of offscreen device contexts for handling devices with hardware accelerated BitBlt functions. The normal offscreen device context interprets the offscreen display memory as a rectangular region that can be addressed in a fashion similar to the normal display device. Sprites and bitmaps stored in the offscreen memory must be addressed using the bounding rectangles for each bitmap, requiring the offscreen memory to be divided up into multiple rectangles. However the linear offscreen device context interprets the offscreen display memory as a linear block of memory, and sprites and other bitmaps can be cached one after the other in the offscreen device, and are referenced according to their offset in the offscreen display device, allowing for much more efficient storage of sprites and bitmaps in the offscreen buffer.

Some functions, like the MGL_suspend and MGL_resume functions are specific to MSDOS to handle issues like being able to shell out to DOS.

MGL_beep
MGL_createLinearOffscreenDC
MGL_createOffscreenDC
MGL_delay
MGL_getPaletteSnowLevel
MGL_getTicks
MGL_setPaletteSnowLevel
MGL_useEvents

MGL_createDisplayDC
MGL_createMemoryDC
MGL_createScrollingDC
MGL_destroyDC
MGL_getTickResolution
MGL_resume
MGL_suspend

Platform Specific: MGLWIN

The following functions are specific to the Windows version of MGL. Most of the functions found in the MGLWIN specific bindings are also present in

the other platform specific bindings, so you can write common code that will compile for any of the supported platforms.

MGL can run in the Windows environment as either a windowed application (a normal Windows application using MGL for drawing to offscreen DIB's) or as a fullscreen application (using our WinDirect technology). The API for fullscreen applications is almost identical to that for all other fullscreen environments. However when running in a windowed environment, all event handling should be done using the normal Windows methods, and MGL should be used simply for drawing data to an offscreen DIB.

Note also that the Windowed environment is currently the only supported environment if you plan to target Windows NT. To run only on Windows NT, you must initialize MGL with the MGL_initWindowed function rather than the normal MGL_init function. The MGL_init function will attempt to load the fullscreen DLL libraries, which will not load under Windows NT. Also if you plan to target a windowed only environment for Windows 95, then by using MGL_initWindowed you won't have to ship the WinDirect fullscreen DLL's with your application.

Note that MGL provides support for two different types of offscreen device contexts for handling devices with hardware accelerated BitBlt functions. The normal offscreen device context interprets the offscreen display memory as a rectangular region that can be addressed in a fashion similar to the normal display device. Sprites and bitmaps stored in the offscreen memory must be addressed using the bounding rectangles for each bitmap, requiring the offscreen memory to be divided up into multiple rectangles. However the linear offscreen device context interprets the offscreen display memory as a linear block of memory, and sprites and other bitmaps can be cached one after the other in the offscreen device, and are referenced according to their offset in the offscreen display device, allowing for much more efficient storage of sprites and bitmaps in the offscreen buffer.

MGL_activatePalette	MGL_beep
MGL_createDisplayDC	MGL_createLinearOffscreenDC
MGL_createMemoryDC	MGL_createOffscreenDC
MGL_createScrollingDC	MGL_createWindowedDC
MGL_delay	MGL_destroyDC
MGL_getFullScreenWindow	MGL_getPaletteSnowLevel
MGL_getTickResolution	MGL_getTicks
MGL_initWindowed	MGL_registerEventProc

MGL_setAppInstance
MGL_setPaletteSnowLevel
MGL_setWinDC

MGL_setMainWindow
MGL_setSuspendAppCallback

MGL Library Reference

Reference Entry Template

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.

Return value

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

Description

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

See Also

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

CPU_getProcessorSpeed

Returns the speed of the processor in Mhz.

Declaration

```
ulong CPU_getProcessorSpeed(void)
```

Prototype In

ztimer.h

Return Value

Processor speed in Mhz.

Description

This function returns the speed of the CPU in Mhz. Note that if the speed cannot be determined, this function will return 0.

See Also

CPU_getProcessorType, *CPU_haveMMX*

CPU_getProcessorType

Returns the type of processor in the system.

Declaration

```
uint CPU_getProcessorType(void)
```

Prototype In

ztimer.h

Return Value

Numerical identifier for the installed processor

Description

Returns the type of processor in the system. Note that if the CPU is an unknown Pentium family processor that we don't have an enumeration for, the return value will be greater than or equal to the value of CPU_UnkPentium (depending on the value returned by the CPUID instruction).

See Also

CPU_getProcessorSpeed, *CPU_haveMMX*

CPU_haveMMX

Returns true if the processor supports Intel MMX extensions.

Declaration

```
bool CPU_haveMMX(void)
```

Prototype In

ztimer.h

Return Value

True if MMX is available, false if not.

Description

This function determines if the processor supports the Intel MMX extended instruction set. If the processor is not an Intel or Intel clone CPU, this function will always return false.

See Also

CPU_getProcessorType, *CPU_getProcessorSpeed*

EVT_asciiCode

Macro to extract the ASCII code from a message.

Declaration

```
uchar EVT_asciiCode(  
    ulong message)
```

Prototype In

mgraph.h

Parameters

message Message to extract ASCII code from

Return Value

ASCII code extracted from the message.

Description

Macro to extract the ASCII code from the message field of the *event_t* structure. You pass the message field to the macro as the parameter and the ASCII code is the result, for example:

```
event_t myEvent;  
uchar   code;  
code = MGL_asciiCode(myEvent.message);
```

See Also

EVT_scanCode, *EVT_repeatCount*

EVT_flush

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

Declaration

```
void MGLAPI EVT_flush(  
    uint mask)
```

Prototype In

mgraph.h

Parameters

mask

Mask specifying the types of events that should be removed

Description

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

See Also

EVT_getNext, *EVT_halt*, *EVT_peekNext*

EVT_getNext

Retrieves the next pending event from the event queue.

Declaration

```
bool MGLAPI EVT_getNext(  
    event_t *evt,  
    uint mask)
```

Prototype In

mgraph.h

Parameters

evt
mask

Pointer to structure to return the event info in
Mask specifying the types of events that should be
removed

Return Value

True if an event was pending, false if not.

Description

Retrieves the next pending event from the event queue, and stores it in a *event_t* structure. The mask parameter is used to specify the type of events to be removed, and can be any logical combination of any of the flags defined by the *MGL_eventType* enumeration.

The what field of the event contains the event code of the event that was extracted. All application specific events should begin with the EVT_USEREVT code and build from there. Since the event code is stored in an integer, there is a maximum of 16 different event codes that can be distinguished (32 for the 32 bit version). 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 EVT_MOUSEDOWN event).

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 the return value will return false.

The `EVT_TIMERTICK` event is used to report that a specified time interval has elapsed since the last `EVT_TIMERTICK` event occurred. See *`EVT_setTimerTick()`* for information on how to enable timer tick events and to set the timer interval.

See Also

`EVT_flush`, `EVT_halt`, `EVT_peekNext`, `EVT_setTimerTick`

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

Declaration

```
void MGLAPI EVT_halt(  
    event_t *evt,  
    uint mask)
```

Prototype In

mgraph.h

Parameters

<i>evt</i>	Pointer to
<i>mask</i>	Mask specifying the types of events that should be removed

Description

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

See Also

EVT_getNext, EVT_halt, EVT_peekNext

EVT_isKeyDown

Determines if a specified key is currently down.

Declaration

```
bool MGLAPI EVT_isKeyDown(  
    uchar scanCode)
```

Prototype In

mgraph.h

Parameters

scanCode Scan code to test

Return Value

True if the specified key is currently held down.

Description

This function determines if a specified key is currently down at the time that the call is made. You simply need to pass in the scan code of the key that you wish to test, and SciTech MGL will tell you if it is currently down or not. SciTech MGL does this by keeping track of the up and down state of all the keys.

EVT_peekNext

Peeks at the next pending event in the event queue.

Declaration

```
bool MGLAPI EVT_peekNext(  
    event_t *evt,  
    uint mask)
```

Prototype In

mgraph.h

Parameters

evt
mask

Pointer to structure to return the event info in
Mask specifying the types of events that should be
removed

Return Value

True if an event is pending, false if not.

Description

Peeks at the next pending event of the specified type in the event queue. The mask parameter is used to specify the type of events to be peeked at, and can be any logical combination of any of the flags defined by the *MGL_eventType* enumeration.

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.

See Also

EVT_flush, *EVT_getNext*, *EVT_halt*

EVT_post

Posts a user defined event to the event queue

Declaration

```
bool MGLAPI EVT_post(  
    ulong which,  
    uint what,  
    ulong message,  
    ulong modifiers)
```

Prototype In

mgraph.h

Parameters

<i>what</i>	Type code for message to post
<i>message</i>	Event specific message to post
<i>modifiers</i>	Event specific modifier flags to post

Return Value

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

Description

This routine is used to post user defined events to the event queue.

See Also

EVT_flush, EVT_getNext, EVT_peekNext, EVT_halt

EVT_repeatCount

Macro to extract the repeat count from a message.

Declaration

```
short EVT_repeatCount(  
    ulong message)
```

Prototype In

mgraph.h

Parameters

message Message to extract repeat count from

Return Value

Repeat count extracted from the message.

Description

Macro to extract the repeat count from the message field of the event structure. The repeat count is the number of times that the key repeated before there was another keyboard event to be place in the queue, and allows the event handling code to avoid keyboard buffer overflow conditions when a single key is held down by the user. If you are processing a key repeat code, you will probably want to check this field to see how many key repeats you should process for this message.

See Also

EVT_asciiCode, EVT_repeatCount

EVT_scanCode

Macro to extract the keyboard scan code from a message.

Declaration

```
uchar EVT_scanCode(  
    ulong message)
```

Prototype In

mgraph.h

Parameters

message Message to extract scan code from

Return Value

Keyboard scan code extracted from the message.

Description

Macro to extract the keyboard scan code from the message field of the event structure. You pass the message field to the macro as the parameter and the scan code is the result, for example:

```
event_t myEvent;  
uchar   code;  
code = MGL_scanCode(myEvent.message);
```

See Also

EVT_asciiCode, *EVT_repeatCount*

EVT_setTimerTick

Set the interval between EVT_TIMERTICK events.

Declaration

```
int MGLAPI EVT_setTimerTick(  
    int ticks)
```

Prototype In

mgraph.h

Parameters

ticks New value for timer tick interval (in milliseconds)

Return Value

Old value of timer tick interval

Description

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

Under Windows, one tick is approximately equal to 1/1000 of a second (millisecond).

Under MSDOS, 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);
```

GM_chooseMode

Display a dialog box to choose a fullscreen graphics mode

Declaration

```
bool MGLAPI GM_chooseMode(  
    GM_modeInfo *mode,  
    bool *startWindowed)
```

Prototype In

gm\gm.h

Parameters

<i>mode</i>	Place to return the selected mode information
<i>startWindowed</i>	True if use wishes to start windowed, false if not

Return Value

True if a mode was chosen, false on error or if the user clicked cancel.

Description

This function will bring up a dialog box allowing the user to interactively choose the graphics mode to be used for fullscreen modes, as well as allowing then to change the OpenGL implementation, WinDirect and DirectDraw support and also force the game to start in a window or fullscreen. This is mostly a convenience function which is great for debugging, testing and demonstration purposes. Note that if you do call this function, you must add the resources for the dialog box used to your application, which are located in the SCITECH\INCLUDE\GM\GMDLG.RC resource file.

GM_cleanup

Cleans up the Game Framework and restores original mode

Declaration

```
void MGLAPI GM_cleanup(void)
```

Prototype In

gm\gm.h

Description

This function calls the users registered exit function, exits SciTech MGL (which puts the system back into text mode for DOS or GDI mode for Windows) and then does some final Game Framework cleanup. Normally you wont call this function unless you have replaced the *GM_mainLoop* function with your own custom version.

See Also

GM_mainLoop, *GM_processEvents*, *GM_processEventsWin*

GM_exit

Tells the Game Framework to exit the main loop

Declaration

```
void MGLAPI GM_exit(void)
```

Prototype In

gm\gm.h

Description

This function simple lets the Game Framework know that the user wants to exit and to clean up and exit from the main loop. After a call to this function, the Game Framework will return from *GM_mainLoop* when it begins to process the next frame.

See Also

GM_mainLoop

GM_findMode

Finds an available mode that has the desired resolution and color depth.

Declaration

```
bool MGLAPI GM_findMode(  
    GM_modeInfo *mode,  
    int xRes,  
    int yRes,  
    int bits)
```

Prototype In

gm\gm.h

Parameters

<i>mode</i>	Place to store the returned graphics mode information
<i>xRes</i>	X resolution of the mode to find
<i>yRes</i>	Y resolution of the mode to find
<i>bits</i>	Color depth of the mode to find (-1 for don't care)

Return Value

True if a valid mode was found, false if not found.

Description

This function searches the list of available graphics modes for one that matches the desired resolution and color depth. This is most useful for finding a good default graphics mode to start your game in if the user has not selected a default mode yet. Note that this function searches for the mode from the top of the list backwards, so that we find the highest performance 320x200 and 320x240 modes (i.e.: the Linear Framebuffer modes rather than the VGA ModeX or Standard VGA modes).

GM_getDoDraw

Returns true if drawing is allowed

Declaration

```
bool MGLAPI GM_getDoDraw(void)
```

Prototype In

gm\gm.h

Return Value

True if drawing is allowed

Description

This function returns the value of the global variable GM_doDraw, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getExitMainLoop

Returns true if the main loop should exit

Declaration

```
bool MGLAPI GM_getExitMainLoop(void)
```

Prototype In

gm\gm.h

Return Value

True if the main loop should exit

Description

This function returns the value of the global variable GM_exitMainLoop, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getHaveWin31

Returns true if we are running on Windows 3.1

Declaration

```
bool MGLAPI GM_getHaveWin31(void)
```

Prototype In

gm\gm.h

Return Value

True when running on Windows 3.1

Description

This function returns the value of the global variable GM_haveWin31, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getHaveWin95

Returns true if we are running on Windows 95

Declaration

```
bool MGLAPI GM_getHaveWin95(void)
```

Prototype In

gm\gm.h

Return Value

True when running on Windows 95

Description

This function returns the value of the global variable GM_haveWin95, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getHaveWinNT

Returns true if we are running on Windows NT

Declaration

```
bool MGLAPI GM_getHaveWinNT(void)
```

Prototype In

gm\gm.h

Return Value

True when running on Windows NT

Description

This function returns the value of the global variable GM_haveWinNT, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

Initializes the Game Framework

Declaration

```
GMDC * MGLAPI GM_init(  
    const char *windowTitle)
```

Prototype In

gm\gm.h

Parameters

windowTitle Title for window in windowed modes and on task bar

Return Value

Pointer to the game framework context object

Description

This function initializes the Game Framework and must be called before you attempt to set a graphics mode. Once this function has been called, the Game Framework will have enumerated all the available graphics modes and stored this information into the `modeList` field of the `GMDC` structure returned from this function. It is then up to the application to find a suitable mode and initialized it with a call to `GM_setMode`.

Before you can do anything useable with the Game Framework, after you have called the `GM_init` function, you must then register a number of function callbacks with the Game Framework that it will call to implement the 'body' of the game (similar to C++ virtual functions, but in C). Two of the most important are `GM_setDrawFunc` and `GM_setGameLogicFunc`. If you want to respond to keyboard commands you will probably also want to call `GM_setKeyDownFunc` as well.

Note: *The Game Framework is responsible for creating the main window used by the game. Hence the value you pass in for `windowTitle` will be the main title for your games window in windows modes, as well as the title that the user will see when your game is minimised to the task bar in Windows 95 and Windows NT 4.0.*

Note: *The Game Framework only creates and maintains a single window for the life of the game, and on switches between windowed and fullscreen modes will automatically change the attributes of the main window for the appropriate mode. This way your game only needs to register a single main*

initialization time, and avoids the problems of re-starting DirectSound during mode switches (and hence you sound can continue to play as you switch on the fly between resolutions and fullscreen and windowed modes).

See Also

*GM_setDriverOptions, GM_setMode, GM_setDrawFunc,
GM_setGameLogicFunc*

GM_initPath

Disables support for static system palette colors

Declaration

```
void MGLAPI GM_initPath(  
    const char *MGLPath)
```

Prototype In

gm\gm.h

Parameters

MGLPath path to MGL resources

Description

This function tells the Game Framework where to find the MGL resources such as bitmaps, fonts etc. By default SciTech MGL always looks in the current directory, however you can use this to point to a directory that contains the resources such as on a CD-ROM drive. Functions such as *MGL_loadBitmap* and *MGL_loadFont* will use this path to locate the files.

See Also

GM_init

GM_initSysPalNoStatic

Disables support for static system palette colors

Declaration

```
void MGLAPI GM_initSysPalNoStatic(  
    bool flag)
```

Prototype In

gm\gm.h

Parameters

flag True to disable static system colors

Description

This function is used to inform the Game Framework whether you wish to bypass the static system palette in your game and use all available colors (except 0 and 255 which are always black and white respectively). By default the static colors are left alone and SciTech MGL will modify the hardware palette to use the proper static colors (by default this value is set to false and the static system colors cannot be changed).

See Also

GM_init

GM_initWindowPos

Sets the default window position for windowed modes

Declaration

```
void MGLAPI GM_initWindowPos(  
    int x,  
    int y)
```

Prototype In

gm\gm.h

Parameters

x	X coordinate for top left corner of window
y	Y coordinate for top left corner of window

Description

This function tells the Game Framework where you want the top left corner of the window to be positioned for windowed modes. By default the Game Framework will center the window on the screen the first time the window is created, and you can use this function to override the default behavior.

See Also

GM_init

GM_mainLoop

Runs the main loop for the Game Framework

Declaration

```
void MGLAPI GM_mainLoop(void)
```

Prototype In

gm\gm.h

Description

This function is the main event loop for the Game Framework, and controls execution of the game until the game exists. This function is responsible for farming out events to the event handling callbacks registered by the game along with calling the game's gameLogic and draw callbacks. Note that if we are suspended on the task bar we continue to process events and call the gameLogic function (so networking can continue) however we skip the call the to drawing function to avoid attempting to write to video memory we no longer own.

The main loop is a simple loop constructed of the following steps (note that MyGameLogic and MyDrawFrame are assumed to be your game logic and draw functions that you would normally register with the Game Framework before calling the *GM_mainLoop* function):

```
GM_exitMainLoop = false;
while (!GM_exitMainLoop) {
    GM_processEvents();
    MyGameLogic();
    if (GM_doDraw)
        MyDrawFrame();
}
GM_cleanup();
```

If you wish to replace the main loop with your own, you can take the existing main loop code and replace it with your own variations. The *GM_processEvents* functions processes events via the MGL event handling functions and dispatches them to the registered event callbacks. If you wish you can call *GM_processEventsWin* instead, which will simply flush the windows message queue and you will be expected to handle all keyboard and mouse events with the window procedure registered with *GM_registerEventProc*. Hence an alternate main loop with all event handling

done in regular window procedure would be coded as follows:

```
GM_exitMainLoop = false;
GM_registerEventProc(MyWindowProc);
while (!GM_exitMainLoop) {
    GM_processEventsWin();
    MyGameLogic();
    if (GM_doDraw)
        MyDrawFrame();
}
GM_cleanup();
```

Note: *If you do replace the main loop with your own, make absolutely sure that you don't call your draw function if the global variable `GM_doDraw` is set to false, otherwise your game could lock the system when the user Alt-Tabs away and back to the desktop. Also make sure that you call `GM_cleanup` on the way out.*

Note: *To exit the main loop, call the `GM_exit` function which lets the main loop know that it should exit on the next iteration.*

Note: *You must call `GM_processEvents` or `GM_processEventsWin` in your main loop to ensure that the Game Framework has a chance to process some internal functions every loop.*

See Also

`GM_init`, `GM_setDrawFunc`, `GM_setGameLogicFunc`, `GM_exit`, `GM_processEvents`, `GM_processEventsWin`, `GM_cleanup`

GM_processEvents

Processes all events for the current iteration of the main loop

Declaration

```
void MGLAPI GM_processEvents(void)
```

Prototype In

gm\gm.h

Description

This function is the event processing handler for the main event loop for the Game Framework. This function is responsible for farming out events to the event handling callbacks registered by the game.

Note: *If you wish to process messages in your game using a regular Windows window procedure, replace the GM_mainLoop function with your own and call GM_processEventsWin instead.*

See Also

GM_mainLoop, GM_processEventsWin, GM_cleanup

GM_processEventsWin

Processes all Windows events for the current iteration of the main loop

Declaration

```
void MGLAPI GM_processEventsWin(void)
```

Prototype In

gm\gm.h

Description

This function is the message processing handler for the main event loop for the Game Framework. This function basically processes all windows messages and passes them to the window procedure for handling. Note that this function does not use SciTech MGL's event handling routines, and is specific to Windows. Essentially this function implements the following:

```
while ( PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) ) {  
    TranslateMessage( &msg );  
    DispatchMessage( &msg );  
}
```

See Also

GM_mainLoop, *GM_processEvents*, *GM_cleanup*

GM_realizePalette

Programs the hardware color palette

Declaration

```
void MGLAPI GM_realizePalette(  
    int numColors,  
    int startIndex,  
    int waitVRT)
```

Prototype In

gm\gm.h

Parameters

<i>numColors</i>	Number of colors to set
<i>startIndex</i>	Starting index in device context palette
<i>waitVRT</i>	True to wait for vertical retrace

Description

This function programs the hardware color palette from the current display device context. You should first call *GM_setPalette* to set the color palette entries to the values that you require before calling this function.

Note: *If we have a memory back buffer we also realize the palette values for this to ensure we have an identity palette for blit operations for maximum speed.*

See Also

GM_setPalette

GM_registerEventProc

Registers a user event procedure with the Game Framework

Declaration

```
void MGLAPI GM_registerEventProc(  
    MGL_WNDPROC winproc)
```

Prototype In

gm\gm.h

Parameters

winproc

User window procedure to register

Description

This function registers a user window procedure with the Game Framework. The primary purpose of this function is to allow your game to process regular windows messages (such as CD-Audio notification messages and other messages not automatically handled by the Game Framework) in your game. To use this function, simply write a regular window procedure as you would normally for a real window, but instead of registering the window procedure with in the RegisterClass function (the Game Framework calls RegisterClass for you during initialization) call this function to register your window procedure with the Game Framework.

Note: *This function is only available in the Windows version of the Game Framework.*

GM_registerMainWindow

Registers a user main window with the Game Framework

Declaration

```
void MGLAPI GM_registerMainWindow(MGL_HWND  
    hwndMain)
```

Prototype In

gm\gm.h

Parameters

hwndMain

Handle to the main window for the application

Description

This function registers a user main window with the Game Framework. The primary purpose of this function is to allow your main game code to do the creation of the main window that is used by the Game Framework, instead of letting the Game Framework libraries do it for you. This is mostly to support integrating the Game Framework code with existing game code that already does window creation and message handling.

Note that you should only call this function *after* you have called *GM_init* to initialize the Game Framework.

Note: *If you use this function to register a main window, do not use the GM_registerEventProc function to register your window procedure with the Game Framework!!*

Note: *This function is only available in the Windows version of the Game Framework.*

GM_setAppActivate

Sets the application activate callback function

Declaration

```
void MGLAPI GM_setAppActivate(  
    GM_activateFunc func)  
typedef void (*GM_activateFunc)(  
    bool active)
```

Prototype In

gm\gm.h

Parameters

func Application activate callback function to register

Description

This function sets the application activate callback function for your Game Framework game. This function is called in windowed modes whenever the activation status of your game changes, which can occur if the window is minimised to the task bar or if the user switches away to another application using *Alt-Tab*. Your callback is passed a flag that indicates whether your game is now currently active or not, and should be used to enable and disable support for things such as CD-Audio when your application loses activation (or the current focus).

Note: *The Game Framework contains built in support for enabling and disabling the static system palette colors when running in windowed modes, and will automatically switch back to static system color mode when your window loses the activation focus. Hence you should not attempt to change this in your application activate callback (if you want to be able to use the static system colors in a window, call GM_initSysPalNoStatic(true) before you initialize the Game Framework.*

Note: *This function is only called in windowed modes, and the equivalent function for fullscreen modes is set with the GM_setSuspendAppCallback function.*

See Also

GM_init, GM_setSuspendAppCallback, GM_initSysPalNoStatic

GM_setDrawFunc

Sets the draw callback function

Declaration

```
void MGLAPI GM_setDrawFunc(  
    GM_drawFunc func)  
typedef void (*GM_drawFunc)(void)
```

Prototype In

gm\gm.h

Parameters

func Draw callback function to register

Description

This function sets the draw callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) once per frame to draw the current frame in the game. Note that the Game Framework expects this function both draw the current frame and swap the buffers to make them visible using *GM_swapBuffers* or *GM_swapDirtyBuffers*.

Note: *In order to be able to continue running your games main logic loops while the user has switch away (i.e.: Alt-Tab) from your game in Windows, while the application is minimised we continue to process messages and call the registered game logic callback, however the draw callback will not be called until the application is restored to fullscreen mode. Hence your draw callback should not contain any game logic functionality, but only contain code to draw the current frame in the game.*

See Also

GM_init, *GM_swapBuffers*, *GM_swapDirtyBuffers*

GM_setDriverOptions

Sets driver registration options for the Game Framework

Declaration

```
void MGLAPI GM_setDriverOptions(  
    GM_driverOptions *opt)
```

Prototype In

gm\gm.h

Parameters

opt Parameter block containing driver registration options

Description

This function tells the Game Framework which driver technologies you want to support in your game. By default all of them are enabled, and you can use this function to disable certain driver technologies at runtime for compatibility in the field. You may call this function as many times as you wish to change the driver options on the fly, and if the values change the Game Framework will re-enumerate the list of available graphics modes for you.

Note that the *GM_driverOptions* structure also contains the *modeFlags* field which represents the color depths that you will be supporting in your application, so that the Game Framework will only enumerate modes that your game can support. For instance if you only support 8bpp modes, then pass a value of *GM_MODE_8BPP*. If you support 8bpp and 15/16bpp then pass in a value of *GM_MODE_8BPP | GM_MODE_16BPP*. Note also that you can change the supported mode flags at any time, which is useful if your software renderer only supports 8bpp modes, while in 3D hardware accelerated modes you want to support all available color depths.

Note: *The Game Framework enumerates both 15bpp (5:5:5) and 16bpp (5:6:5) modes when you pass in a value of GM_MODE_16BPP, since both of these modes will likely be available on end user systems. Also note that it is equally likely that in some cases only one of these formats and not the other is supported on the end user system, so your code will have to be able to support both formats for compatibility.*

Note: *If your game does not require hardware OpenGL support (i.e.: you are not using OpenGL in your game), then you should set the useOpenGLHW flag to false to make sure that the OpenGL drivers are not registered with the MGL Library Reference (which would require the OpenGL runtime DLL's to be*

installed on your end users system).

Note: *We recommend that you provide support for disabling both DirectDraw and WinDirect modes via a command line switch in your game, in case either of these two technologies have problems on your customer machines. Please see the Game Framework sample code for ideas on how to do this.*

See Also

GM_init

GM_setEventFunc

Sets the event callback function

Declaration

```
void MGLAPI GM_setEventFunc(  
    GM_eventFunc func)  
typedef void (*GM_eventFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Event callback function to register

Description

This function sets the general event callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all non key and non mouse related events in the order that they are entered by the user (i.e.: timer or user events).

GM_setExitFunc

Sets the exit callback function

Declaration

```
void MGLAPI GM_setExitFunc(  
    GM_exitFunc func)  
typedef void (*GM_exitFunc)(void)
```

Prototype In

gm\gm.h

Parameters

func Exit callback function to register

Description

This function sets the exit callback function for your Game Framework game and is called by the Game Framework main loop just before calling *MGL_exit* to shut down SciTech MGL and return to windowed mode before returning to your code. If you have any code that must be called before exiting from fullscreen mode, you should register it with this function.

Note: *This function will always be called after the mode switch has occurred and the system is in the new graphics mode.*

See Also

GM_init, *GM_mainLoop*

GM_setGameLogicFunc

Sets the game logic callback function

Declaration

```
void MGLAPI GM_setGameLogicFunc(  
    GM_gameFunc func)  
typedef void (*GM_gameFunc)(void)
```

Prototype In

gm\gm.h

Parameters

func Game logic callback function to register

Description

This function sets the game logic callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) once per frame to update the game logic for the next frame after drawing the current frame.

Note: *In order to be able to continue running your games main logic loops while the user has switch away (i.e.: Alt-Tab) from your game in Windows, while the application is minimised we continue to process messages and call the registered game logic callback, however the draw callback will not be called until the application is restored to fullscreen mode. Hence your game logic callback should not contain any code that performs drawing to the screen, as all that code should be located in your draw callback function.*

See Also

GM_init, *GM_setDrawFunc*

GM_setKeyDownFunc

Sets the key down callback function

Declaration

```
void MGLAPI GM_setKeyDownFunc(  
    GM_keyDownFunc func)  
typedef void (*GM_keyDownFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Key down callback function to register

Description

This function sets the key down callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key down events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events, or one that handles both key down and key repeat events and register the same function with the Game Framework for both event types.

Note: *The key down callback will not be called for key repeat events (i.e.: the user holds a key down). If you wish to capture key repeat events, use GM_setKeyRepeatFunc.*

See Also

GM_init, GM_setKeyRepeatFunc, GM_setKeyUpFunc

GM_setKeyRepeatFunc

Sets the key repeat callback function

Declaration

```
void MGLAPI GM_setKeyRepeatFunc(  
    GM_keyRepeatFunc func)  
typedef void (*GM_keyRepeatFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Key repeat callback function to register

Description

This function sets the key repeat callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key repeat events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events, or one that handles both key down and key repeat events and register the same function with the Game Framework for both event types.

See Also

GM_setKeyDownFunc, *GM_setKeyUpFunc*

GM_setKeyUpFunc

Sets the key up callback function

Declaration

```
void MGLAPI GM_setKeyUpFunc(  
    GM_keyUpFunc func)  
typedef void (*GM_keyUpFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Key up callback function to register

Description

This function sets the key up callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key up events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events if you wish.

See Also

GM_setKeyDownFunc, *GM_setKeyRepeatFunc*

GM_setMode

Switches to a fullscreen or windowed graphics mode

Declaration

```
bool MGLAPI GM_setMode(  
    GM_modeInfo *info,  
    bool windowed,  
    int pages,  
    bool forceSysMem)
```

Prototype In

gm\gm.h

Parameters

<i>info</i>	Structure describing the mode to initialize
<i>windowed</i>	True if the mode should be in a window
<i>pages</i>	Number of pages for hardware buffering
<i>forceSysMem</i>	Flag to force a system memory buffer for all drawing

Return Value

True on success, false on error

Description

This function sets a graphics mode for the game given the passed in mode information. If the windowed parameter is set to true, a windowed mode will be set otherwise a fullscreen graphics mode will be set. You must pass in one of the modes listed in the *GMDC* modeList returned from *GM_init* to this function. When the Game Framework switches to the windowed equivalent to a fullscreen graphics mode, we search for the closest 1:1 aspect ratio window size for the mode that corresponds to the fullscreen mode chosen. Hence for a 320x200 fullscreen mode we choose a 320x240 window size, for 320x400 and 320x480 we choose a 320x240 window size and for 640x350 and 640x400 we choose a 640x480 window size. This way we won't end up with a window on the desktop that is not a 1:1 aspect ratio and desktop modes are generally 1:1 aspect ratio.

The number of pages passed in is used to allocate the specified number of hardware video memory pages for multi-buffering and may be any value. The Game Framework will however lower this value to the maximum that the hardware supports, and if only 1 hardware page is available a system memory back buffer will be created automatically. Hence if you want to

always try and use triple buffering if available, pass a value of 3 when you call this function.

Note also that the Game Framework automatically handles switching between windowed and fullscreen modes on the fly. By default the Game Framework contains code to provide two methods of switching to fullscreen modes when running in windowed modes:

- When the user hits the *Alt-Enter* key combination
- When the user clicks the *Maximise* button on the games title bar

Likewise when the game is running in a fullscreen mode and the user hits the *Alt-Enter* key, the video mode will automatically be switched to windowed mode. Unless you have registered a mode switch callback with *GM_setModeSwitchFunc*, on the fly switching between fullscreen and windowed modes is disabled.

Note: *You may call this function while already in a windowed or fullscreen graphics mode, which will cause the current graphics mode to be changed on the fly.*

Note: *If you wish to force the Game Framework to always create a system memory buffer for rendering, regardless of how many display pages are available, set the *forceSysMem* field to true when calling this function. Note also that if you have requested multiple buffers, all those buffers will still be used so you will see no tearing, however all rendering will be performed to a system buffer which will be copied to the screen when you swap the buffers with *GM_swapBuffers*.*

Drawing directly to a hardware linear framebuffer is usually extremely fast and will provide the most efficient method of rendering. However if your rendering requires reads from the framebuffer (for effects such as blending), this will be very slow over the PCI bus and your code will be faster if you force a system member back buffer.

Note: *If the user dynamically switches the resolution or color depth of the windows desktop while your application is running, if you have registered a mode switch callback with the Game Framework (via *GM_setModeSwitchFunc*) then the Game Framework will switch to a new windowed mode to cause the memory back buffers to be re-allocated for the most optimal format for the Windows display mode.*

See Also

GM_init, *GM_setModeSwitchFunc*

GM_setModeFilterFunc

Sets the custom mode filter for mode enumeration

Declaration

```
void MGLAPI GM_setModeFilterFunc(  
    GM_modeFilterFunc filter)  
typedef bool (*GM_modeFilterFunc)(  
    int xRes,  
    int yRes,  
    int bits,  
    ulong flags)
```

Prototype In

gm\gm.h

Parameters

filter New mode filter to set

Description

This function allows you to register a mode filter callback with the Game Framework, which will be called during mode enumeration and will allow you to apply your own custom filtering code to the list of available video modes. Hence you can use this function to filter out all non 1:1 aspect ratio modes for instance.

See Also

GM_init

GM_setModeSwitchFunc

Sets the mode switch callback function

Declaration

```
void MGLAPI GM_setModeSwitchFunc(  
    GM_modeSwitchFunc func)  
typedef void (*GM_modeSwitchFunc)(  
    GM_modeInfo *mode,  
    bool windowed)
```

Prototype In

gm\gm.h

Parameters

func Mode switch callback function to register

Description

This function sets the mode switch callback function for your Game Framework game and is called by the Game Framework when automatically switching on the fly between windowed and fullscreen modes. By default this handler is set to NULL, and unless you call this function support for switching on the fly between windowed and fullscreen modes is disabled. By default the Game Framework contains code to provide two methods of switching to fullscreen modes when running in windowed modes:

- When the user hits the *Alt-Enter* key combination
- When the user clicks the *Maximise* button on the games title bar

Likewise when the game is running in a fullscreen mode and the user hits the *Alt-Enter* key, the video mode will automatically be switched to windowed mode. In order to support auto-switching between fullscreen and windowed modes, all the MGL device contexts will be destroyed and re-created during the switch, so you will have to include other code to re-initialize SciTech MGL to the state that the game is currently in (i.e.: setting the color palette, etc.) in your mode switch callback. You will also need to code your game in such as way that it can handle dynamic resolution changes on the fly.

See Also

GM_init, *GM_setMode*

GM_setMouseDownFunc

Sets the mouse down callback function

Declaration

```
void MGLAPI GM_setMouseDownFunc(  
    GM_mouseDownFunc func)  
typedef void (*GM_mouseDownFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Mouse down callback function to register

Description

This function sets the mouse down callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse down events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all mouse events, or one that handles both mouse down and mouse up events and register the same function with the Game Framework for both event types.

See Also

GM_init, *GM_setMouseUpFunc*, *GM_setMouseMoveFunc*

GM_setMouseMoveFunc

Sets the mouse move callback function

Declaration

```
void MGLAPI GM_setMouseMoveFunc(  
    GM_mouseMoveFunc func)  
typedef void (*GM_mouseMoveFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Mouse move callback function to register

Description

This function sets the mouse move callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse move events in the order that they are entered by the user.

See Also

GM_init, *GM_setMouseDownFunc*, *GM_setMouseUpFunc*

GM_setMouseUpFunc

Sets the mouse up callback function

Declaration

```
void MGLAPI GM_setMouseUpFunc(  
    GM_mouseUpFunc func)  
typedef void (*GM_mouseUpFunc)(  
    event_t *evt)
```

Prototype In

gm\gm.h

Parameters

func Mouse up callback function to register

Description

This function sets the mouse up callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse up events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all mouse events, or one that handles both mouse down and mouse up events and register the same function with the Game Framework for both event types.

See Also

GM_init, *GM_setMouseDownFunc*, *GM_setMouseMoveFunc*

GM_setPalette

Sets the color palette entries

Declaration

```
void MGLAPI GM_setPalette(  
    palette_t *pal,  
    int numColors,  
    int startIndex)
```

Prototype In

gm\gm.h

Parameters

<i>pal</i>	Array of palette values to set
<i>numColors</i>	Number of colors to set
<i>startIndex</i>	Starting index in device context palette

Description

This function sets the palette values for the currently activate Game Framework device context to the values passed in pal. Note that this function does not program the hardware palette, but simply updates the internal palette values for the MGL device context. Once you have set the values in the palette, you should then call *GM_realizePalette* to program the hardware palette entries from the values currently stored in the MGL display device context.

Note: *If we have a memory back buffer we also set the palette values for this to ensure we have an identity palette for blit operations for maximum speed.*

See Also

GM_realizePalette

GM_setPreModeSwitchFunc

Sets the pre-mode switch callback function

Declaration

```
void MGLAPI GM_setPreModeSwitchFunc(  
    GM_preModeSwitchFunc func)  
typedef bool (*GM_preModeSwitchFunc)(  
    GM_modeInfo *mode,  
    bool windowed)
```

Prototype In

gm\gm.h

Parameters

func Pre-mode switch callback function to register

Description

This function sets the pre-mode switch callback function for your Game Framework game and is called by the Game Framework when automatically switching on the fly between windowed and fullscreen modes. If you have registered a mode switch function with *GM_setModeSwitchFunc*, your pre-mode switch function will be called before the existing mode is destroyed, giving your code a change to destroy any internal data structures that might need to be cleaned up before the mode is destroyed and the new one created.

See Also

GM_init, *GM_setMode*, *GM_setModeSwitchFunc*

GM_setSuspendAppCallback

Sets the suspend app callback function

Declaration

```
void MGLAPI GM_setSuspendAppCallback(  
    MGL_setSuspendAppCallback saveState)
```

Prototype In

gm\gm.h

Parameters

func

Suspend app callback function to register

Description

This function sets the suspend app callback function for your Game Framework game and is called by the Game Framework whenever the user switches away from your game (suspends it, such as with Alt-Tab). The Game Framework registers a default suspend application callback with SciTech MGL to do most of the handling for you. Note however that by default the Game Framework suspend app callback passes a return value of MGL_SUSPEND_APP back to MGL which will suspend execution of your game until it has been restored. The Game Framework has code to automatically ensure that the draw callback is not called when the game is minimised and gets re-enabled when the game is restored again if you return a value of MGL_NO_SUSPEND_APP from your registered callback (this way you can continue to run networking code in the background to keep other network players running if the server is temporarily minimised).

You should register your own version of this function to handle extra things during suspend and restores such as suspending CD-Audio sound playback or other stuff not automatically handled by the underlying multi-media libraries. A typical suspend application callback might be coded as follows:

```

int ASMAPI SuspendAppProc(MGLDC *dc,int flags)
{
    if (flags == MGL_DEACTIVATE) {
        // Disable CD-Audio
        // Do other disabling stuff
        return MGL_NO_SUSPEND_APP;
    }
    else if (flags == MGL_REACTIVATE) {
        // Re-enable CD-Audio
        // Do other re-enabling stuff
        return MGL_NO_SUSPEND_APP;
    }
}

```

See Also

GM_init, *GM_setSuspendAppProc*

GM_startOpenGL

Starts OpenGL graphics for the game

Declaration

```
bool MGLAPI GM_startOpenGL(  
    MGL_glContextFlagsType flags)
```

Prototype In

gm\gm.h

Parameters

flags

OpenGL Rendering Context flags

Return Value

True on success, false on error

Description

This functions enables support for the OpenGL 3D API for the Game Framework, and after this call you must do all rendering via calls to the OpenGL API. The flags parameter (of type *MGL_glContextFlagsType*) is used to specify the type of OpenGL rendering context that you want, such as if you want RGB or color index mode, single or double buffering, an alpha buffer, an accumulation buffer, a depth buffer (z-buffer) and a stencil buffer.

If you pass in a value of *MGL_GL_VISUAL* for the flags parameter, SciTech MGL will use the OpenGL visual that was set by a previous call to *MGL_glSetVisual*. Hence if you require more control over the type of OpenGL rendering context that is created, you can call *MGL_glChooseVisual* and *MGL_glSetVisual* before calling this function. Note that you should **not** call *MGL_glCreateContext* when using the Game Framework, but call this function instead.

Note: *After this function has been called, the current rendering context will have been made the current OpenGL rendering context with a call to *MGL_glMakeCurrent*, so you can simply start issuing OpenGL rendering commands to start drawing after calling this function.*

See Also

GM_setMode, MGL_glChooseVisual, MGL_glSetVisual

GM_swapBuffers

Swaps the display buffers for the game

Declaration

```
void MGLAPI GM_swapBuffers(  
    MGL_waitVRTFlagType waitVRT)
```

Prototype In

gm\gm.h

Parameters

waitVRT Wait for vertical retrace flag

Description

Swaps the display buffers for the Game Framework game. If there are multiple hardware display pages enabled for the game, the waitVRT flag (of *MGL_waitVRTFlagType*) is used to determine if SciTech MGL should wait for the vertical retrace before swapping display pages or not.

If you started the Game Framework and requested a system memory back buffer, enabled stretching or there was only one hardware display page available, the Game Framework will blit the entire system memory back buffer to the display device context and then perform the hardware page flip.

Note: *If you are intentionally using a system memory back buffer, you may want to maintain a set of dirty rectangles for your display pages and call GM_swapDirtyBuffers to swap only the dirty portions of the frames to speed things up. This is most useful for games that don't update large portions of the screen very frequently.*

See Also

GM_swapDirtyBuffers

GM_swapDirtyBuffers

Swaps the display buffers for the game with dirty rectangles

Declaration

```
void MGLAPI GM_swapDirtyBuffers(  
    region_t *dirty,  
    MGL_waitVRTFlagType waitVRT)
```

Prototype In

gm\gm.h

Parameters

<i>dirty</i>	Region of dirty rectangles to blit
<i>waitVRT</i>	Wait for vertical retrace flag

Description

Swaps the display buffers for the Game Framework game by blitting the list of dirty rectangles to the display. The list of dirty rectangles is passed in as an MGL region, which you can construct using the MGL region manipulation functions.

If there are multiple hardware display pages enabled for the game, the waitVRT flag (of *MGL_waitVRTFlagType*) is used to determine if SciTech MGL should wait for the vertical retrace before swapping display pages or not.

Note: *You should make sure you first call MGL_optimiseRegion before you call this function to minimise the number of rectangles in the dirty rectangle list. If you don't do this, the result will be the same but it may take longer to perform the blitting.*

Note: *If you did not specifically request a system memory back buffer, this function will behave identically to GM_swapBuffers and no blitting will occur.*

See Also

GM_swapBuffers

LZTimerCount

Returns the current count for the Long Period Zen Timer.

Declaration

```
ulong LZTimerCount(void)
```

Prototype In

ztimer.h

Return Value

Count that has elapsed in microseconds.

Description

Returns the current count that has elapsed between calls to *LZTimerOn* and *LZTimerOff* in microseconds.

See Also

LZTimerOn, *LZTimerOff*, *LZTimerLap*

LZTimerLap

Returns the current count for the Long Period Zen Timer and keeps it running.

Declaration

```
ulong LZTimerLap(void)
```

Prototype In

ztimer.h

Return Value

Count that has elapsed in microseconds.

Description

Returns the current count that has elapsed since the last call to *LZTimerOn* in microseconds. The time continues to run after this function is called so you can call this function repeatedly.

See Also

LZTimerOn, *LZTimerOff*, *LZTimerCount*

LZTimerOff

Stops the Long Period Zen Timer counting.

Declaration

```
void LZTimerOff(void)
```

Prototype In

ztimer.h

Description

Stops the Long Period Zen Timer counting and latches the count. Once you have stopped the timer you can read the count with *LZTimerCount*. If you need highly accurate timing, you should use the on and off functions rather than the lap function since the lap function does not subtract the overhead of the function calls from the timed count.

See Also

LZTimerOn, *LZTimerLap*, *LZTimerCount*

LZTimerOn

Starts the Long Period Zen Timer counting.

Declaration

```
void LZTimerOn(void)
```

Prototype In

ztimer.h

Description

Starts the Long Period Zen Timer counting. Once you have started the timer, you can stop it with *LZTimerOff* or you can latch the current count with *LZTimerLap*.

The Long Period Zen Timer uses a number of different high precision timing mechanisms to obtain microsecond accurate timings results whenever possible. The following different techniques are used depending on the operating system, runtime environment and CPU on the target machine. If the target system has a Pentium CPU installed which supports the Read Time Stamp Counter instruction (RDTSC), the Zen Timer library will use this to obtain the maximum timing precision available.

Under 32-bit Windows, if the Pentium RDTSC instruction is not available, we first try to use the Win32 QueryPerformanceCounter API, and if that is not available we fall back on the timeGetTime API which is always supported.

Under 32-bit DOS, if the Pentium RDTSC instruction is not available, we then do all timing using the old style 8253 timer chip. The 8253 timer routines provide highly accurate timings results in pure DOS mode, however in a DOS box under Windows or other Operating Systems the virtualization of the timer can produce inaccurate results.

Note: *Because the Long Period Zen Timer stores the results in a 32-bit unsigned integer, you can only time periods of up to 2^{32} microseconds, or about 1hr 20mins. For timing longer periods use the Ultra Long Period Zen Timer.*

See Also

LZTimerOff, LZTimerLap, LZTimerCount

Divides a fixed point number by another.

Declaration

```
fix32_t WINAPI MGL_FixDiv(  
    fix32_t f,  
    fix32_t g)
```

Prototype In

mgraph.h

Parameters

<i>a</i>	Fixed point number to divide
<i>b</i>	Fixed point number to divide by

Return Value

Result of the division.

Description

Divides a fixed point number by another fixed point number. The idea is relatively simple; We want to set up a 64 bit dividend to be divided by our 32 bit divisor, which will give us a new 32 bit result.

See Also

MGL_FixMul, *MGL_FixMulDiv*

Multiplies two fixed point number in 16.16 format

Declaration

```
fix32_t ASMAPI MGL_FixMul(  
    fix32_t f,  
    fix32_t g)
```

Prototype In

mgraph.h

Parameters

<i>a</i>	First number to multiply
<i>b</i>	Second number to multiply

Return Value

Result of the multiplication.

Description

Multiplies two fixed point number in 16.16 format together and returns the result. We cannot simply multiply the two 32 bit numbers together since we need to shift the 64 bit result right 16 bits, but the result of a FXFixed multiply is only ever 32 bits! Thus we must resort to computing it from first principles (this is slow and should ideally be re-coded in assembler for the target machine).

We can visualise the fixed point number as having two parts, a whole part and a fractional part:

$$\text{FXFixed} = (\text{whole} + \text{frac} * 2^{-16})$$

Thus if we multiply two of these numbers together we get a 64 bit result:

$$\begin{aligned} & (f_whole + f_frac * 2^{-16}) * (g_whole + g_frac * 2^{-16}) \\ &= \\ & (f_whole * g_whole) + \\ & (f_whole * g_frac) * 2^{-16} + \\ & (g_whole * f_frac) * 2^{-16} + \\ & (f_frac * g_frac) * 2^{-32} \end{aligned}$$

To convert this back to a 64 bit fixed point number to 32 bit format we simply shift it right by 16 bits (we can round it by adding 2^{-17} before doing this shift). The formula with the shift integrated is what is used below.

Natrually you can alleviate most of this if the target machine can perform a native 32 by 32 bit multiplication (since it will produce a 64 bit result).

See Also

MGL_FixDiv, MGL_FixMulDiv

Multiplies a fixed point number by another and divides by a third number.

Declaration

```
fix32_t ASMAPI MGL_FixMulDiv(  
    fix32_t a,  
    fix32_t b,  
    fix32_t c)
```

Prototype In

mgraph.h

Parameters

<i>a</i>	First number to multiply
<i>b</i>	Second number to multiply
<i>c</i>	Third number to divide by

Return Value

Results of the multiplication and division.

Description

This function multiplies a 16.16 fixed point number by another producing a 32.32 intermediate result. This 32.32 result is then divided by another 16.16 number to produce a 16.16 result. Because this routine maintains maximum precision during the multiplication stage, you can multiply numbers that would normally overflow the standard *MGL_FixMul* function.

See Also

MGL_FixMul, *MGL_FixDiv*

MGL_activatePalette

Activates the Windows palette for a windowed DC.

Declaration

```
bool MGLAPI MGL_activatePalette(  
    MGLDC *dc,  
    bool unrealize)
```

Prototype In

mglwin.h

Parameters

<i>dc</i>	Device context (must be a Windowed DC)
<i>unrealize</i>	True if the palette should be unrealized first before realizing it.

Return Value

True if the palette has changed and the window will be repainted, or false if the palette did not change.

Description

Activates the Windows palette for a windowed device context for an MGL windowed application running under Windows. This function is usually in response to the WM_PALETTECHANGED and WM_QUERYNEWPALETTE messages as follows:

If you are chaining between SYSPAL_STATIC and SYSPAL_NOSTATIC modes, you should always pass true for the unrealize flag to ensure that the palette is correctly updated.

See Also

MGL_setWinDC

MGL_appActivate

Function to call when your app becomes active.

Declaration

```
void MGLAPI MGL_appActivate(  
    MGLDC *winDC,  
    bool active)
```

Prototype In

mglwin.h

Parameters

winDC
active

Currently active windowed DC
True if app is active, false if not

Description

Tracks whether a game application is currently active for palette management. Forces a repaint if the system palette usage is set to no-static.

MGL_availableBitmap

Determines if the specified bitmap file is available for use.

Declaration

```
bool MGLAPI MGL_availableBitmap(  
    const char *bitmapName)
```

Prototype In

mgraph.h

Parameters

bitmapName Name of bitmap file to check for

Return Value

True if the bitmap file exists, false if not.

Description

Attempt to locate the specified bitmap file, and verify that it is available for use. See *MGL_loadBitmap* for more information on the algorithm that MGL uses when searching for bitmap files on disk.

See Also

MGL_loadBitmap

MGL_availableCursor

Determines if the specified cursor file is available for use.

Declaration

```
bool MGLAPI MGL_availableCursor(  
    const char *cursorName)
```

Prototype In

mgraph.h

Parameters

cursorName Name of cursor file to check for

Return Value

True if the cursor file exists, false if not.

Description

Attempt to locate the specified mouse cursor, and verify that it is available for use. See *MGL_loadCursor* for more information on the algorithm that MGL uses when searching for mouse cursor files on disk.

See Also

MGL_loadCursor

MGL_availableFont

Determines if a specific font file is available for use.

Declaration

```
bool MGLAPI MGL_availableFont(  
    const char *fontname)
```

Prototype In

mgraph.h

Parameters

fontname Relative filename of the required font file

Return Value

True if font file is available, false if not.

Description

Attempt to locate the specified font file, and verify that it is available for use. See *MGL_loadFont* for more information on the algorithm that MGL uses when searching for font files on disk.

See Also

MGL_loadFont

MGL_availableIcon

Determines if the specified icon file is available for use.

Declaration

```
bool MGLAPI MGL_availableIcon(  
    const char *iconName)
```

Prototype In

mgraph.h

Parameters

iconName Name of icon file to check for

Return Value

True if the icon file exists, False if not.

Description

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

See Also

MGL_loadIcon

MGL_availableJPEG

Determines if the specified JPEG bitmap file is available for use.

Declaration

```
bool MGLAPI MGL_availableJPEG(  
    const char *JPEGName)
```

Prototype In

mgraph.h

Parameters

JPEGName Name of JPEG bitmap file to check for

Return Value

True if the JPEG bitmap file exists, false if not.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

Description

Attempt to locate the specified JPEGformat bitmap file and verify that it is available for use. See MGL_loadJPEG for more information on the algorithm that uses MGL uses when searching for the JPEG files on disk.

See Also

MGL_loadJPEG

MGL_availableMemory

Determines how much memory is available.

Declaration

```
void MGL_availableMemory(  
    ulong *physical,  
    ulong *total)
```

Prototype In

mgraph.h

Parameters

physical
total

Place to return the amount of physical memory available
Place to return the total memory available (including
virtual memory)

Description

Returns a measurement of how much physical system memory and total system memory (including virtual memory) is available to the application.

Note: *This measurement is only valid at program startup before you have made any calls to the standard library malloc/free functions or the MGL_malloc/free functions. It is useful to determine if there is enough memory installed on the machine when the program starts. If you need to keep track of how much memory is available to your application, you will either have to use operating system specific services or keep track of all calls to malloc/free after you program has started.*

Note of course that in a multitasking operating system like Windows or OS/2, the amount of available memory may suddenly change if another application allocates memory while yours is still running.

See Also

MGL_malloc, MGL_fopen

MGL_availableModes

Returns a list of all available graphics modes.

Declaration

```
uchar * MGLAPI MGL_availableModes(void)
```

Prototype In

mgraph.h

Return Value

Pointer to list of available graphics modes

Description

Returns a list of all the currently available graphics modes. You may call this routine before *MGL_init* is called to determine what graphics modes are available before actually initializing a particular graphics mode. You must ensure however that the *MGL_detectGraph* routine is called before you call this routine.

The list of available graphics modes is returned as a table of 8-bit integer values. The table is terminated with a -1(0xFF). MGL supports a number of different video mode resolutions, ranging from 320x200 up to 1600x1200 with color ranges from 16 colors up to 16.7 million colors. See the *MGL_modeType* type for a list of all the valid MGL graphics modes.

See Also

MGL_detectGraph, *MGL_init*, *MGL_availablePages*

MGL_availablePCX

Determines if the specified PCX bitmap file is available for use.

Declaration

```
bool MGLAPI MGL_availablePCX(  
    const char *PCXName)
```

Prototype In

mgraph.h

Parameters

PCXName Name of PCX bitmap file to check for

Return Value

True if the PCX bitmap file exists, false if not.

Description

Attempt to locate the specified PCX format bitmap file and verify that it is available for use. See MGL_loadPCX for more information on the algorithm that uses MGL uses when searching for the PCX files on disk.

See Also

MGL_loadPCX

MGL_availablePages

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

Declaration

```
int MGLAPI MGL_availablePages(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode MGL mode number to query

Return Value

Number of available video pages for mode, -1 for invalid mode number.

Description

Returns the number of pages of physical video memory available for a specific MGL graphics mode. You may call this routine before the *MGL_init* routine is used to initialize a particular graphics modes. You must ensure however that the *MGL_detectGraph* routine is called before you call this routine. Thus you can filter out support for modes that do not have the required number of hardware video pages that your application requires.

See Also

MGL_detectGraph, *MGL_init*, *MGL_availableModes*

MGL_backfacing

Determines if a polygon is backfacing.

Declaration

```
int ASMAPI MGL_backfacing(  
    fix32_t dx1,  
    fix32_t dy1,  
    fix32_t dx2,  
    fix32_t dy2)
```

Prototype In

mgraph.h

Parameters

<i>dx1</i>	change in x along first edge
<i>dy1</i>	change in y along first edge
<i>dx2</i>	change in x along second edge
<i>dy2</i>	change in y along second edge

Return Value

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

Description

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.

Note that this function is written to correctly calculate the results for screen space coordinates, which can cause overflow with a normal 16.16 fixed point multiply if this is calculated directly using calls to *MGL_FixMul*.

MGL_beep

Generate a beep on the PC speaker.

Declaration

```
void MGLAPI MGL_beep(  
    int freq,  
    int msec)
```

Prototype In

mgldos.h, mglwin.h

Parameters

<i>freq</i>	frequency of the beep
<i>milliseconds</i>	length of beep in milliseconds

Description

Beeps the PC speaker at the specified frequency for the specified length of time.

MGL_beginDirectAccess

Enables direct framebuffer access.

Declaration

```
void ASMAPI MGL_beginDirectAccess(void)
```

Prototype In

mgraph.h

Description

Enables direct framebuffer access so that you can directly rasterize to the linear framebuffer memory using your own custom routines. Note that calling this function is absolutely necessary when using hardware acceleration, as this function correctly arbitrates between the hardware accelerator graphics engine and your direct framebuffer rasterizing code.

See Also

SV_endDirectAccess

MGL_beginPixel

Setup for high speed pixel drawing.

Declaration

```
void WINAPI MGL_beginPixel(void)
```

Prototype In

mgraph.h

Description

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* and *MGL_getPixel* 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 need to plot more than a single pixel at a time.

See Also

MGL_endPixel, *MGL_pixel*, *MGL_getPixel*.

Blts a block of image data from one device context into another.

Declaration

```
void MGL_bitBlt(  
    MGLDC *dst,  
    MGLDC *src,  
    rect_t r,  
    int dstLeft,  
    int dstTop,  
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>r</i>	Rectangle defining are to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_bitBltCoord*, however it takes entire rectangles as parameters instead of coordinates.

See Also

MGL_bitBltCoord, *MGL_stretchBlt*, *MGL_transBlt*

Blts a block of image data from one device context into another.

Declaration

```
void MGLAPI 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

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of image to Blt from
<i>top</i>	Top coordinate of image to Blt from
<i>right</i>	Right coordinate of image to Blt from
<i>bottom</i>	Bottom coordinate of image to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from one device context to another. The source and destination rectangles may overlap even if the source and destination device contexts are the same, and MGL will correctly handle the overlapping regions. This routine has been highly optimized for absolute maximum performance, so it will provide the fastest method of copying bitmap data between device contexts. To obtain absolute maximum performance, you should align the source and destination bitmaps on DWORD boundaries (4 pixels for 8 bit, 2 pixels for 15/16 bit) and the low level device driver code will special case this for maximum performance.

This function will correctly handle Blts across device contexts with differing pixel depths, and will perform the necessary pixel format translation to convert from the source device to the destination device. Note that although the code to implement this is highly optimized, this can be a time consuming operation so you should attempt to pre-convert all bitmaps to the current display device pixel format for maximum performance if using this routine for sprite animation.

MGL does however have special case code to specifically handle translation of 24 bit RGB format bitmaps (the standard RGB DIB format used by Video for Windows) to all 8 bit and above pixel formats. When converting from 24 bit to 8 bit, MGL will dither bitmaps in real time from 24 bit to the 8 bit halftone palette. This provides a solid foundation to build real time 24 bit motion video playback in all supported video modes in MGL.

Note that when *MGL_bitBlt* is called for 4 and 8 bit source bitmaps MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for Blting bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device, or you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

This routine can also be used to perform hardware accelerated Blts between offscreen memory devices and the display device when running in fullscreen modes, providing the hardware accelerator (if present) can support this operation.

The write mode operation specifies how the source image data should be combined with the destination image data. Write modes supported by the SciTech MGL are enumerated in *MGL_writeModeType*.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note that the MGL/Lite libraries support the *MGL_bitBlt* functions, but only from system memory device contexts to display memory device contexts, the write mode operation is ignored for all calls and no palette translation or color conversion is supported.

The destination rectangle is clipped according to the current clipping rectangles for the destination device context.

See Also

MGL_bitBlt, *MGL_stretchBlt*, *MGL_transBlt*

MGL_bitBltLin

Copies a block of image data from a linear offscreen device context to a display device.

Declaration

```
void MGL_bitBltLin(  
    MGLDC *dst,  
    MGLDC *src,  
    ulong srcOfs,  
    rect_t dstRect,  
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	destination device context (must be a display device)
<i>src</i>	source device context (must be a linear offscreen device)
<i>dstRect</i>	rectangle defining source image
<i>op</i>	write mode to use during Blt

Description

This function is the same as *MGL_bitBltLin*, however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_bitBltLinCoord, *MGL_transBltLin*

MGL_bitBltLinCoord

Copies a block of image data from a linear offscreen device context to a display device.

Declaration

```
void MGLAPI MGL_bitBltLinCoord(  
    MGLDC *dst,  
    MGLDC *src,  
    ulong srcOfs,  
    int dstLeft,  
    int dstTop,  
    int dstRight,  
    int dstBottom,  
    int op)
```

Prototype In
mgraph.h

Parameters

<i>dst</i>	destination device context (must be a display device)
<i>src</i>	source device context (must be a linear offscreen device)
<i>srcOfs</i>	Starting offset of bitmap in the source device surface
<i>dstLeft</i>	left coordinate of destination image
<i>dstTop</i>	top coordinate of destination image
<i>dstRight</i>	right coordinate of destination image
<i>dstBottom</i>	bottom coordinate of destination image
<i>op</i>	write mode to use during Blt

Description

This function is similar to *MGL_bitBlt* except that the source device context must be for a linear offscreen device and the destination device context must be for a display device. The difference is that this function copies a bitmap as a linear chunk of memory from the offscreen memory to the display memory using the hardware accelerator. Hence you can store bitmaps in the offscreen memory device contiguously, which provides for much more efficient utilization of the hardware video memory for storing sprites and bitmaps.

Note however that not all hardware accelerators can support linear offscreen memory, in which case you would not be able to create a linear

offscreen device context.

The write mode operation specifies how the source image data should be combined with the destination image data. Write modes supported by the SciTech MGL are enumerated in *MGL_writeModeType*.

The destination rectangle is clipped according to the current clipping rectangles for the destination device context.

See Also

MGL_bitBltLin, *MGL_transBltLin*

MGL_buildMonoMask

Create a monochrome bitmap mask given a transparent color.

Declaration

```
bitmap_t * MGLAPI MGL_buildMonoMask(  
    bitmap_t *bitmap,  
    color_t transparent)
```

Prototype In

mgraph.h

Parameters

<i>bitmap</i>	bitmap to build monochrome mask from
<i>transparent</i>	transparent color to mask out

Return Value

Pointer to allocated bitmap mask, or NULL on error.

Description

Attempts to build a monochrome bitmap mask that can be used to rasterize transparent bitmaps. This is useful for devices that have hardware BitBlt capabilities but lack hardware transparent BitBlt support. Everywhere the transparent color is found, the mask will be 0, and everywhere else it will be 1. Once you have created a monochrome bitmap mask, you can rasterize a transparent bitmap by replacing all the transparent pixels in the original bitmap with 0's, and then doing the following:

1. Set the foreground color to black and drawing the bitmap mask with *MGL_putBitmapMask*. Everywhere that the bitmap has data a black pixel will be used to 'punch' a hole in the display.
2. Use the *MGL_bitBlt* function to draw the original bitmap on the display using the *MGL_OR_MODE* write mode. Everywhere that a pixel is transparent in the original source bitmap, the destination will remain the same. Everywhere that a pixel is non-transparent the pixel on the destination will be replaced with the pixel in the source bitmap.

Note that only 8+ bits per pixel bitmaps are supported, and this function will fail on lower color depths.

See Also

MGL_transBltLin, MGL_putBitmapMask, MGL_bitBlt

MGL_calloc

Allocate and clear a large memory block.

Declaration

```
void * MGLAPI MGL_calloc(  
    long s,  
    long n)
```

Prototype In

mgraph.h

Parameters

s	size of unit in bytes
n	number of contiguous s-byte units to allocate

Return Value

Pointer to allocated memory if successful, NULL if out of memory.

Description

Allocates a block of memory of length ($s * n$), and clears the allocated area with zeros (0). Note that unlike the standard C calloc function, this routine will properly handle allocations of blocks of memory larger than 64Kb in 16 bit real mode environments.

If you have changed the memory allocation routines with the *MGL_useLocalMalloc* function, then calls to this function will actually make calls to the local memory allocation routines that you have registered.

See Also

MGL_malloc, *MGL_free*, *MGL_memset*, *MGL_useLocalMalloc*

MGL_changeDisplayMode

Changes the current fullscreen mode or switches to windowed mode.

Declaration

```
bool MGLAPI MGL_changeDisplayMode(  
    int mode)
```

Prototype In

mgldos.h, mglwin.h

Parameters

mode New display mode to use

Return Value

True if the mode is available, false if mode is invalid.

Description

This function changes the current fullscreen display mode used by MGL, or informs MGL that you are about to switch to windowed mode (for Windows versions). The application should destroy all display and offscreen device contexts currently allocated before calling this function, and then re-create all the required device contexts for the new mode after calling this function. A typical code sequence to change display modes would be as follows:

```
MGLDC *dc;  
... init MGL and create DC as per normal ...  
MGL_destroyDC(dc);  
MGL_changeDisplayMode(grSVGA_640x480x256);  
dc = MGL_createDisplayDC();  
... mode has been changed to the new mode ...
```

Note that if there are any active display device contexts and offscreen device contexts when this function is called, they will be destroyed by this call and the system will be reset back to text mode. However none of the device contexts will be re-created and it is up to the application to recreate all necessary device contexts.

If you are using this function to change display modes on the fly in MGL and you wish to allow the user to switch to a windowed mode under Windows, you must call this function with the `grWINDOWED` parameter before you create your windowed window, or call *MGL_exit* after finishing

in fullscreen modes. For example the following code might be used to switch to a windowed mode.

```
// Destroy the existing fullscreen mode and DC's
MGL_destroyDC (mgldc);
MGL_destroyDC (memdc);
mgldc = memdc = NULL;

// Signal to MGL that we are going windowed
MGL_changeDisplayMode(grWINDOWED);

// Create the windowed window
window = CreateWindow(...);
ShowWindow(window, SW_SHOWDEFAULT);
```

See Also

MGL_init, *MGL_createDisplayDC*

MGL_charWidth

Returns the width of a character in pixels.

Declaration

```
int MGLAPI MGL_charWidth(  
    char ch)
```

Prototype In

mgraph.h

Parameters

ch Character to measure

Return Value

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

Description

Return the width of the specified character, given the currently active font and attribute settings.

See Also

MGL_textWidth, MGL_textHeight, MGL_useFont

MGL_checkIdentityPalette

Turns on or off identity palette checking.

Declaration

```
bool MGLAPI MGL_checkIdentityPalette(  
    bool enable)
```

Prototype In

mgraph.h

Parameters

enable

True to enable identity palette checking, false to disable

Return Value

Old value of the identity palette check flag.

Description

Turns on or off the checking of identity palette mappings for MGL. This is a global flag, and by default, identity palette checking is turned on. When *MGL_bitBlt* is called for 4 and 8 bit source bitmaps MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. You can use this function to tell MGL that all bitmaps will have identity palettes and that you will handle all pixel translations yourself, which allows the BitBlt code to run with higher performance.

See Also

MGL_bitBlt

MGL_clearDevice

Clears the currently active display page.

Declaration

```
void WINAPI MGL_clearDevice(void)
```

Prototype In

mgraph.h

Description

This function 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_clearRegion

Clears the specified region to an empty region.

Declaration

```
void MGLAPI MGL_clearRegion(  
    region_t *r)
```

Prototype In

mgraph.h

Parameters

r region to be cleared

Description

This function clears the specified region to an empty region, freeing up all the memory used to store the region data.

See Also

MGL_newRegion, *MGL_copyRegion*, *MGL_freeRegion*

MGL_clearViewport

Clears the currently active viewport.

Declaration

```
void MGLAPI MGL_clearViewport(void)
```

Prototype In

mgraph.h

Description

This function will clear the currently active display page viewport to 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 with an arbitrary pattern instead, as this function always clears the viewport to the solid background color.

See Also

MGL_clearDevice, *MGL_fillRect*

Clips a line to a specified fixed point clipping rectangle.

Declaration

```
bool MGLAPI MGL_clipLineFX(  
    fix32_t *fx1,  
    fix32_t *fy1,  
    fix32_t *fx2,  
    fix32_t *fy2,  
    fix32_t left,  
    fix32_t top,  
    fix32_t right,  
    fix32_t bottom)
```

Prototype In

mgraph.h

Parameters

<i>fx1</i>	Pointer to x coordinate of first endpoint to clip
<i>fy1</i>	Pointer to y coordinate of first endpoint to clip
<i>fx2</i>	Pointer to x coordinate of second endpoint to clip
<i>fy2</i>	Pointer to y coordinate of second endpoint to clip
<i>left</i>	Left coordinate of clip rectangle
<i>top</i>	Top coordinate of clip rectangle
<i>right</i>	Right coordinate of clip rectangle
<i>bottom</i>	Bottom coordinate of clip rectangle

Return Value

True if clipped line is inside the clip rectangle, false if completely outside.

Description

This function will clip the line defined by the endpoints (x1,y1) and (x2,y2) to the specified clipping rectangle. If the line is accepted (i.e. it intersects with the clipping rectangle) then the routine will return true and the line endpoints will be updated to reflect the clipped line's new endpoints. If the line is completely outside of the clipping rectangle, the routine will return false.

Note that all coordinates are passed in 16.16 fixed point format, as is the clipping rectangle, to provide for maximum precision in the clipping

calculations.

MGL_computePixelAddr

Computes the address of a pixel in the device context surface.

Declaration

```
void * MGLAPI MGL_computePixelAddr(  
    MGLDC *dc,  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

dc	Device context to compute the pixel address for
x	X coordinate of pixel to address
y	Y coordinate of pixel to address

Return Value

Pointer to the start of the pixel in the surface of the device context.

Description

This function computes the address of a pixel in the surface of a specific device context. This function is most useful for developing custom rendering routines that draw directly to the surface of a device context, and will compute the address of the pixel correctly regardless of the color depth of the device context. Essentially this function computes the following:

```
addr = dc->surface + (y * bytesPerLine) + (x *  
bytesPerPixel)
```

If you are going to be doing a lot of address calculations, it will be faster to optimise your code to do the calculations directly in place (such as with a macro) and specifically to eliminate the last multiply if you know in advance what color depth you are working with (i.e.: change it to be x , $x*2$, $x*3$ or $x*4$ depending on the color depth).

Note: *You cannot use this function to address the device context surface if the device surface access type returned by `MGL_getSurfaceAccessType` is set to `MGL_NO_ACCESS`.*

MGL_copyRegion

Create a copy of the specified region.

Declaration

```
region_t * MGLAPI MGL_copyRegion(  
    const region_t *r)
```

Prototype In

mgraph.h

Parameters

s Pointer to source region

Return Value

Pointer to the copied region, or NULL if out of memory.

Description

Copies the definition for an entire region and returns a pointer to the newly created region. The space for the copied region is allocated from the region memory pool, which MGL uses to maintain a local memory allocation scheme for regions to increase performance.

If there is not enough memory to copy the region, this routine will return NULL.

See Also

MGL_newRegion, *MGL_freeRegion*, *MGL_clearRegion*

Creates a new custom memory device context.

Declaration

```
MGLDC * MGLAPI MGL_createCustomDC(  
    int xSize,  
    int ySize,  
    int bitsPerPixel,  
    pixel_format_t *pf,  
    int bytesPerLine,  
    void *surface,  
    MGL_HBITMAP hbm)
```

Prototype In mgraph.h

Parameters

<i>xSize</i>	X resolution for the memory context
<i>ySize</i>	Y resolution for the memory context
<i>bitsPerPixel</i>	Pixel depth for the memory context
<i>pf</i>	Pixel format for memory context
<i>bytesPerLine</i>	Buffer pitch for memory context
<i>surface</i>	Pointer to surface memory for context
<i>hbm</i>	Handle to HBITMAP for DIB section

Return Value

Pointer to the allocated memory device context.

Description

This function is useful for creating a display context in memory provided by an application, allowing the SciTech MGL to render to memory it does not own. (e.g. custom hardware with LFB's).

For Windows if the hbm parameter is not NULL, it is assumed that the original memory was created by a call to CreateDIBSection and the hbm parameter is a handle to the bitmap object for the DIB section. This parameter can be used to blit the memory DC image to a windowed DC using the standard Windows GDI blit functions. The hbm parameter is not necessary for fullscreen modes.

MGL_createDisplayDC

Create a new display device context.

Declaration

```
MGLDC * MGLAPI MGL_createDisplayDC(  
    int numBuffers)
```

Prototype In

mgraph.h

Parameters

numBuffers

Number of buffers to allocate for double/multi-buffering.

Return Value

Pointer to the newly created display device context, NULL on failure

Description

Creates a new display device context for drawing information directly to the hardware display device in fullscreen graphics modes. If there are no currently active display device contexts, MGL will start the graphics mode specified in the *MGL_init* call or the previous call to *MGL_changeDisplayMode* (which is used to override the setting from *MGL_init*), and will initialize the specific device driver. If however a display device context already exists, we simply create a new device context to interface to the currently active display driver, so that you can create two independent device contexts that address the same display device (with totally independent attributes).

Note that when running under Windows, as soon as you have created the first display device context, the system will have been switched into the fullscreen graphics mode, and the normal Windows GDI functions will not be available for drawing to the screen (under Windows95 you can use the *MGL_getWinDC* function to get a Windows compatible HDC for drawing to the MGL device context however). When you wish to return to GDI mode, a call to *MGL_exit* will destroy all allocated display device contexts and return the system to GDI mode. Note however that SciTech MGL *late binds* the return to GDI mode if you simply destroy all active display device contexts. Hence SciTech MGL will remain in fullscreen mode after destroying the last active display device context in order to provide a smooth transition between fullscreen graphics modes. Hence if you wish to switch from

640x480x256 to 800x600x256, SciTech MGL will remain in fullscreen mode after destroying the 640x480x256 device context and will simply switch directly to the 800x600x256 graphics mode when you create the new display device context for this mode (after calling *MGL_changeDisplayMode* to make 800x600x256 the new active display mode). This way switches between fullscreen modes are clean and do not result in an intermediate switch back to GDI mode.

However if you wish to return to GDI mode, such as when switching from fullscreen mode to windowed mode, you have to call *MGL_changeDisplayMode*(grWINDOWED) to inform SciTech MGL that you wish to switch to windowed mode and SciTech MGL will force the switch back to GDI mode at this time.

If you intend to use double or multi-buffered graphics using the display device, you should set the numBuffers to the number of buffers that you require, so that the device will be properly configured for multi-buffered operation. Note that if you request more buffers than is currently available, this function will fail. Hence you should first call *MGL_availablePages* to determine how many buffers can be used in the desired graphics mode.

Once the display device context has been allocated, if the surface pointer of the *MGLDC* structure is not NULL, you can directly access the surface of the device context as a flat linear framebuffer (only available in 32 bit versions) or as a banked framebuffer. However for some display devices this surface is actually a virtual linear framebuffer, and you must ensure that if you rasterize directly to the surface that you only access it with BYTE, WORD or DWORD aligned accesses. If you access it with a non-aligned location that spans a SuperVGA bank boundary, you will hang the system by causing an infinite page fault loop. You can check the surface access flags with the *MGL_surfaceAccessType* function to determine if the surface is a banked, virtualized or hardware linear framebuffer. If the *MGL_surfaceAccessType* function return *MGL_NOACCESS* the framebuffer will be banked and if you wish to rasterize directly to it you will need to use the *SVGA_setBank* functions to change banks. In banked modes the surface pointer points to the start of the banked framebuffer window (i.e.: 0xA0000).

If the surface is not a hardware linear framebuffer, you may wish to do all your rasterizing to a memory device context (always accessible as a linear buffer) and then use *MGL_bitBlt* get it to the screen, letting MGL take care of the virtualization issues or bank switching issues. In most cases we have tested, this usually leads to higher performance than rasterizing directly to a

banked or virtual linear framebuffer directly. If the surface is a real hardware linear framebuffer you should try to always rasterize directly to the surface, as this usually provides the maximum possible performance for MGL and your own custom rasterizing code.

The dimensions and pixel format of the device context surface are stored in the *gmode_t* field of the *MGLDC* structure, and you should use these values to write your own direct rasterizing code.

Note that all device contexts have an associated color palette, even RGB device contexts. In RGB modes the color palette is used for converting color index pixel values to RGB values during BitBlt operations and with the *MGL_realColor* and *MGL_setColorCI* function.

Note: *Due to the dynamic nature of the speed of both graphics memory and system memory, and due to the fact that each tends to increase in speed at different rates we have found that on some systems rendering directly to the hardware linear framebuffer provides a larger performance improvement. However with new Pentium II systems coming out with faster memory subsystems, this may change and it may be faster to draw to system memory and then MGL_bitBlt it to the screen. But then again with the onset of AGP for graphic adapters, this may change again and make it much faster to render directly to video memory. Hence we always recommend that you provide the option to do both in your product, and if necessary profile each method on the end users system and select the one that provides the highest performance (don't ever assume you know which is going to be the fastest!).*

See Also

MGL_init, *MGL_destroyDC*, *MGL_createOffscreenDC*, *MGL_createMemoryDC*, *MGL_createWindowedDC*, *MGL_createStereoDisplayDC*, *MGL_changeDisplayMode*, *MGL_modeFlags*

MGL_createLinearOffscreenDC

Creates a new linear offscreen display device context.

Declaration

```
MGLDC * MGLAPI MGL_createLinearOffscreenDC(void)
```

Prototype In

mgraph.h

Return Value

Pointer to the newly created linear offscreen device context, NULL if not valid.

Description

Creates a new linear offscreen display device context for storing bitmaps linearly in offscreen video memory when running in hardware accelerated graphics modes. You must already have created a valid display device context before this function is called, otherwise this function will return NULL. If the display device does not support hardware accelerated linear offscreen display memory (many do not), this function will also return NULL. See the *MGL_result* error code for more information on why the function failed.

A linear offscreen DC is almost identical to a normal offscreen device context, and you can rasterize directly to it as a rectangular region. However it is primarily intended for storing bitmaps and sprites in offscreen memory linearly (i.e. one after each other in memory, not as rectangular bitmaps) for better memory efficiency on devices that can support this. All devices that support a linear offscreen device context will also support a normal offscreen device context.

See Also

MGL_init, *MGL_destroyDC*, *MGL_createDisplayDC*, *MGL_createOffscreenDC*, *MGLDC*

MGL_createMemoryDC

Create a new memory device context.

Declaration

```
MGLDC * MGLAPI MGL_createMemoryDC(  
    int xSize,  
    int ySize,  
    int bitsPerPixel,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>xSize</i>	x resolution for the memory context
<i>ySize</i>	y resolution for the memory context
<i>bitsPerPixel</i>	Pixel depth for the memory context
<i>pf</i>	Pixel format for memory context (NULL for 8 bits and below)

Return Value

Pointer to the allocated memory device context, NULL if not enough memory.

Description

Creates a new memory device context, allocating the necessary memory resources to hold the surface of the memory device given the specified resolution and pixel depth. The surface of a memory device context is always allocated using the appropriate operating system specific functions, and you can always directly access the surface of the device context via the surface pointer of the *MGLDC* structure. If you do directly access the surface, the dimensions and pixel format of the device context surface are stored in the *gmode_t* field of the *MGLDC* structure, and you should use these values to write your own direct rasterizing code.

For memory device contexts with pixel depths greater than 8 bits per pixel, you must also pass a valid *pixel_format_t* structure which defines the pixel format to be used for the device context. If you wish to create a memory device context for your main rasterizing context which you then wish to Blt to the screen, you must ensure that you use the same pixel format for the memory device as the display device for the current graphics mode,

otherwise the pixel formats will be translated on the fly by the *MGL_bitBlt* function resulting in very low performance. You can use the *MGL_getPixelFormat* function to obtain the pixel format information for the display device context you are using.

Note that before you can call this routine, you must ensure that you have registered the appropriate packed pixel device drivers that you are interested in using in your code with the *MGL_registerDriver* function. The process of registering the device drivers ensure that the code will be linked in when you link your application (by default the code will not be linked, to save space in the resulting executable). You can simply call *MGL_RegisterAllDispDrivers* to register packed pixel drivers for all possible pixel depths, however this will end up linking in a lot of code that may not be necessary for your application.

Note that all device contexts have an associated color palette, even RGB device contexts. In RGB modes the color palette is used for converting color index pixel values to RGB values during BitBlt operations and with the *MGL_realColor* and *MGL_setColorCI* function.

See Also

MGL_createDisplayDC, *MGL_destroyDC*, *MGL_createWindowedDC*, *MGLDC*

MGL_createOffscreenDC

Creates a new offscreen display device context.

Declaration

```
MGLDC * MGLAPI MGL_createOffscreenDC(void)
```

Prototype In

mgraph.h

Return Value

Pointer to the newly created offscreen device context, NULL if not valid.

Description

Creates a new offscreen display device context for rasterizing to offscreen video memory when running in hardware accelerated video modes. You must already have created a valid display device context before this function is called, otherwise this function will return NULL. Also if the display device does not support hardware accelerated offscreen display memory, this function will also return NULL indicating that the device context could not be created. See the *MGL_result* error code for more information on why the function failed.

Once the display device context has been allocated, if the surface pointer of the *MGLDC* structure is not NULL, you can directly access the surface of the device context as a flat linear framebuffer (only available in 32 bit versions) or as a banked framebuffer. However for some display devices this surface is actually a virtual linear framebuffer, and you must ensure that if you rasterize directly to the surface that you only access it with BYTE, WORD or DWORD aligned accesses. If you access it with a non-aligned location that spans a SuperVGA bank boundary, you will hang the system by causing an infinite page fault loop. You can check the surface access flags with the *MGL_surfaceAccessType* function to determine if the surface is a banked, virtualized or hardware linear framebuffer. If the *MGL_surfaceAccessType* function return *MGL_NOACCESS* the framebuffer will be banked and if you wish to rasterize directly to it you will need to use the *SVGA_setBank* functions to change banks. In banked modes the surface pointer points to the start of the banked framebuffer window (i.e.: 0xA0000).

The dimensions and pixel format of the device context surface are stored in the *gmode_t* field of the *MGLDC* structure, and you should use these values to write your own direct rasterizing code. Note also that for offscreen device contexts, you can also rasterize directly to any part for the offscreen

buffering using MGL rasterizing routines, which will use the hardware accelerator where possible.

See Also

MGL_init, *MGL_destroyDC*, *MGL_createDisplayDC*,
MGL_createLinearOffscreenDC, *MGLDC*

MGL_createScrollingDC

Create a new hardware scrolling display device context.

Declaration

```
MGLDC * MGLAPI MGL_createScrollingDC(  
    int virtualX,  
    int virtualY,  
    int numBuffers)
```

Prototype In

mgraph.h

Parameters

<i>virtualX</i>	Virtual width of desired mode
<i>virtualY</i>	Virtual height of desired mode
<i>numBuffers</i>	Number of buffers for multibuffering

Return Value

Pointer to the newly created hardware scrolling display device context, or NULL if not enough memory.

Description

Creates a new display device context for drawing information directly to the hardware display device in fullscreen graphics modes. Essentially this function is identical to *MGL_createDisplayDC*, however hardware scrolling (or panning) is supported. Some hardware devices may not support hardware scrolling, in which case this function will fail and return a NULL. In these cases you should provide an alternative method of scrolling the display, such as drawing to a memory device context and copying the appropriate portion of the image to the display.

If there are no currently active display device contexts, MGL will start the video mode specified in the *MGL_init* call or the previous call to *MGL_changeDisplayMode* (which is used to override the setting from *MGL_init*), and will initialize the specific device driver. If however a display device context already exists, we simply create a new device context to talk to the currently active display driver, so that you can actually create two independent device contexts that address the same display device (with totally independent attributes).

Once you have created a hardware scrolling device context, the display starting coordinate will be set to (0,0) within the virtual image. To hardware pan around within the virtual image, you can use the *MGL_setDisplayStart* function to change the display starting x and y coordinates.

See Also

MGL_createMemoryDC, *MGL_createDisplayDC*, *MGL_destroyDC*,
MGL_setWinDC, *MGL_activatePalette*, *MGL_initWindowed*,
MGL_changeDisplayMode

MGL_createStereoDisplayDC

Create a new display device context for stereo LC shutter glasses

Declaration

```
MGLDC * MGLAPI MGL_createStereoDisplayDC(  
    int numBuffers,  
    int refreshRate)
```

Prototype In

mgraph.h

Parameters

numBuffers
refreshRate

Number of buffers to allocate for double/multi-buffering.
Requested refresh rate for the graphics mode

Return Value

Pointer to the newly created display device context, NULL on failure.

Description

Creates a new display device context for drawing information directly to the hardware display device in fullscreen graphics modes. Essentially this function is identical to *MGL_createDisplayDC*, however support for LC shutter glasses is provided and SciTech MGL will take care of automatically flipping between the left and right images to create the stereo display. In some cases we may not be able to initialise support for LC shutter glasses, and in this case this function will return NULL.

If there are no currently active display device contexts, MGL will start the video mode specified in the *MGL_init* call or the previous call to *MGL_changeDisplayMode* (which is used to override the setting from *MGL_init*), and will initialize the specific device driver. If however a display device context already exists, we simply create a new device context to talk to the currently active display driver, so that you can actually create two independent device contexts that address the same display device (with totally independent attributes).

When running in stereo mode, SciTech MGL actually allocates twice the number of buffers that you request for drawing images, since we need one buffer for the left eye image and another buffer for the right eye image (i.e.: if you request two stereo buffers for double buffering, SciTech MGL will actually allocate room for four). The reason for this is that when displaying

one of the stereo buffers, SciTech MGL will automatically *swap* between the left and right eye images at every vertical retrace. It also sends a signal to the LC shutter glasses to tell them to block out the image for the eye that should not be seeing the image on the screen (i.e.: when the left image is being displayed, the shutter over the right eye will be blacked out). Hence by drawing images with slightly different viewing parameters (i.e.: as viewed from the left or right eye when doing 3D rendering), the user sees a single image with complete with visual depth cues!

When running in stereo mode, you have to tell SciTech MGL which buffer you want to draw to when drawing the left or right eye images. Just like you normally do in double and multi-buffering, you use the *MGL_setActivePage* function to tell SciTech MGL the active display page you wish to draw to. However in stereo modes you must also pass in the *MGL_LEFT_BUFFER* or *MGL_RIGHT_BUFFER* values to tell SciTech MGL which eye you are drawing for. For instance to draw to stereo page 1, left eye you would use *MGL_setActivePage(1 | MGL_LEFT_BUFFER)*, and for the right eye you would use *MGL_setActivePage(1 | MGL_RIGHT_BUFFER)*.

Note: *In OpenGL rendering modes, changing the draw buffer is done with the OpenGL *glDrawBuffer(GL_BACK_LEFT)* and *glDrawBuffer(GL_BACK_RIGHT)* functions instead of using *MGL_setActivePage*.*

One of the biggest drawbacks to viewing stereo images using LC shutter glasses is that the refresh rate viewed in each eye is exactly half that of the refresh rate of the display mode. Hence if running in a display mode with a 60Hz refresh rate, the user will experience an overall refresh rate of 30Hz per eye! As you can image this can be extremely tiresome for extended viewing periods, so to get around this SciTech MGL allows you to pass in a value to request a higher refresh rate for the mode. Ideally you want to try and use a refresh rate that is twice the desired refresh rate per eye, such as 120Hz for viewing images at 60Hz, however you *must* allow the user to override or suggest a desired refresh rate as many older monitors may not be capable of displaying an image at a high refresh rate like 120Hz.

The refresh rate value that you pass in is a *suggested* value in that SciTech MGL will attempt to set the refresh rate to this value, however if the hardware does not support that refresh rate the next lowest available refresh rate will be used instead. In some situations where no refresh rate control is available, the value will be ignored and the adapter default refresh rate will be used. If you don't care about the refresh rate and want to use the adapter

default setting, pass in a value of `MGL_DEFAULT_REFRESH`.

Note: *In the USA and Canada, the mains frequency runs at 60Hz, and all fluorescent lights will be illuminating your room at frequency of 60Hz. If you use a refresh rate that is not a multiple of the mains frequency and you are viewing the image in a room with fluorescent lights, you may experience severe beating at a frequency that is the difference between the monitor refresh rate and the fluorescent light frequency (i.e.: at 100Hz you will experience a 20Hz annoying beat frequency). In order to get around this problem, always try to use a frequency that is double the mains frequency such as 120Hz to avoid the beating, or have the user turn off their fluorescent lights!*

When you create a stereo display device context, SciTech MGL does not automatically start stereo page flipping, and you must start this with a call to `MGL_startStereo`. You can also turn stereo mode on or off at any time (i.e.: you can turn it off when you go to your menu system) using the `MGL_stopStereo` and `MGL_startStereo` functions. Note that when stereo mode is disabled, SciTech MGL always displays from the left eye buffer.

When viewing an image with LC shutter glasses, SciTech MGL also needs to know how to communicate with the LC shutter glasses and let them know when the left and right eye images are being displayed on the screen. This communication mechanism is called the *stereo sync*, and SciTech MGL supports a number of different types of stereo sync mechanisms (blue codes, serial port, parallel port and hardware stereo sync). You can use the `MGL_setStereoSyncType` function to change the stereo sync method used to communicate with the LC shutter glasses.

See Also

`MGL_createDisplayDC`, `MGL_destroyDC`, `MGL_changeDisplayMode`,
`MGL_startStereo`, `MGL_stopStereo`, `MGL_setStereoSyncType`,
`MGL_setBlueCodeIndex`

MGL_createWindowedDC

Create a new windowed device context.

Declaration

```
MGLDC * MGLAPI MGL_createWindowedDC(  
    MGL_HWND hwnd)
```

Prototype In

mglwin.h

Parameters

hwnd

Window handle with which to associate new device context

Return Value

Pointer to the allocated windowed device context, or NULL if not enough memory.

Description

Creates a new windowed device context for drawing information into a window on the Windows desktop. When you create a Windowed device context, you associate it with a standard Windows HWND for the window that you wish MGL to display its output on. Windowed device contexts are special device contexts in that you cannot directly access the surface for the device, nor can you actually use the MGL rasterizing functions to draw on the device surface. The only rasterizing functions supported are the *MGL_bitBlt* and *MGL_stretchBlt* for Blting data from memory device contexts to the window on the desktop.

However in order to change the color palette values for the data copied to the window, you must use the MGL palette functions on the windowed display device context. Note that MGL automatically takes care of creating a proper Windows identity palette for the windowed device context, so as long as you program the same palette values for the windowed device and the memory device you should get the maximum performance Blting speed.

MGL automatically uses the highest performance method for implementing the BitBlt operations under Windows, and requires the WinG library to be installed if the target platform is Windows 3.1 or Windows NT 3.1. When using WinG, the only pixel depth supported for Windowed device contexts

is 8 bits per pixel, and the only pixel format valid for BitBlt operations is 8 bit memory device contexts. You can still create memory device contexts with higher pixel formats, but you will need to Blt the data to a real 8 bit DC before you can display it in the window.

If the target platform is Windows 95 or Windows NT 3.5 or later, MGL will use CreateDIBSection for maximum performance, and can create and Blt memory device contexts of any pixel depth to the display device context. For maximum performance you should create your memory device contexts with the same pixel format used by the windowed display device context.

Note that if you wish to only use windowed output and you intend to target Windows NT for your application, you must use the *MGL_initWindowed* function to initialize MGL. This will not attempt to load the WinDirect full screen support DLL's, which are not compatible with Windows NT.

See Also

MGL_createMemoryDC, *MGL_createDisplayDC*, *MGL_destroyDC*,
MGL_setWinDC, *MGL_activatePalette*, *MGL_initWindowed*

Create a new rectangle.

Declaration

```
rect_t MGLAPI MGL_defRect(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>Left</i>	Left coordinate of rectangle
<i>Top</i>	Top coordinate of rectangle
<i>Right</i>	Right coordinate of rectangle
<i>bottom</i>	Bottom coordinate of rectangle

Return Value

Newly created rectangle returned by value.

Description

Creates a new rectangle given a set of coordinates.

See Also

MGL_defRectPt

MGL_defRectPt

Create a new rectangle from two points

Declaration

```
rect_t MGLAPI MGL_defRectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Parameters

<i>leftTop</i>	Upper left coordinate of rectangle
<i>rightBottom</i>	Lower right coordinate of rectangle

Return Value

Newly created rectangle returned by value.

Description

Creates a new rectangle given a set of points.

See Also

MGL_defRect

MGL_defaultAttributes

Declaration

```
void MGLAPI MGL_defaultAttributes(  
    MGLDC *dc)
```

Prototype In
mgraph.h

Parameters

dc device context to be reset

Description

This function resets all of the device context attributes to their default values.

See Also

MGL_getAttributes, MGL_restoreAttributes, MGL_getDefaultPalette

MGL_defaultColor

Returns the value for current default color (always white but value may vary).

Declaration

```
color_t MGLAPI MGL_defaultColor(void)
```

Prototype In

mgraph.h

Return Value

Default color value for current video mode (always white).

Description

Returns the default color value 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. However, the numerical value for white will vary depending on the color depth.

See Also

MGL_setColor, *MGL_getColor*

MGL_delay

Do nothing for a specified amount of time.

Declaration

```
void MGLAPI MGL_delay(  
    int msec)  
void MGL_delay(int msec)
```

Prototype In

mgldos.h, mglwin.h

Parameters

milliseconds Number of milliseconds to delay

Description

Delay processing for the specified number of milliseconds.

MGL_destroyDC

Destroy a given device context.

Declaration

```
bool MGLAPI MGL_destroyDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to be destroyed

Return Value

True if context was destroyed, false on error.

Description

Destroys a specified device context, freeing up all resources allocated by the device context. This can fail for a number of reasons, so check the *MGL_result* code to determine the cause of the failure.

If the device context that was destroyed was the last active display device context, the video mode is reset back to the original video mode (or back to the normal GDI desktop for Windows). Note that calling *MGL_exit* automatically destroys all currently allocated device contexts.

See Also

MGL_createDisplayDC, *MGL_createOffscreenDC*,
MGL_createLinearOffscreenDC, *MGL_createMemoryDC*,
MGL_createWindowedDC

MGL_detectGraph

Detects the currently installed video hardware.

Declaration

```
void MGLAPI MGL_detectGraph(  
    int *driver,  
    int *mode)
```

Prototype In

mgraph.h

Parameters

<i>driver</i>	Pointer to graphics device driver id
<i>mode</i>	Pointer to default graphics mode for device

Description

This function autodetects the presence of all the standard graphics adapters supported by 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 driver parameter and a suggested default graphics mode in the mode parameter.

The fullscreen device drivers currently supported by MGL are enumerated in *MGL_driverType*.

If you pass a value of grDETECT to the this routine, it will attempt to find the highest performance driver that has currently been registered by the *MGL_registerDriver* function. However you can also pass explicit values in the driver parameter, and this will automatically set the desired maximum performance driver. For instance you may pass the value of grVGA for the driver id, in which case the detection code will ignore all the VESA and SuperVGA device drivers, so the only device driver that will be used will be the standard VGA ones. This can be used to allow the user to disable certain drivers (like accelerated drivers) to get around problems in the field relating to incompatible video hardware.

If you have registered all available drivers, the priority ordering of which will be used depends on the capabilities of the underlying display hardware, and the drivers will be chosen in the following order (first one in table is selected in preference to ones below it):

<i>Driver</i>	Highest performance driver selected
<i>VBE/AF</i>	ACCEL8/16/24/32
<i>VBE 2.0</i>	LINEAR8/16/24/32
<i>DirectDraw</i>	DDRAW8/16/24/32
<i>VBE 1.2</i>	SVGA4/8/16/24/32
<i>Standard VGA</i>	VGA4, VGAX, VGA8

For instance if you had DirectDraw installed on your system and MGL found either a VBE 2.0 or VBE/AF driver, those modes supported by the VBE drivers will be used in preference to the DirectDraw modes. If however the DirectDraw driver has additional modes not supported by the VBE 2.0 or VBE/AF (for instance 320x240x256), the DirectDraw drivers would be used for those modes. If however you had DirectDraw installed and only had a VBE 1.2 driver available, the DirectDraw drivers will be used for all modes whenever possible.

To change this default behavior you can selectively register only those drivers you wish to use before calling this function or *MGL_init*. A typical sequence of code to register drivers and allows the program to force WinDirect or DirectDraw using the settings of useDirectDraw or useWinDirect might be as follows:

```

MGL_unregisterAllDrivers();
if (useWinDirect) {
    MGL_registerDriver(MGL_VGA4NAME, VGA4_driver);
    MGL_registerDriver(MGL_VGAXNAME, VGAX_driver);
    MGL_registerDriver(MGL_SVGA4NAME, SVGA4_driver);
    MGL_registerDriver(MGL_SVGA8NAME, SVGA8_driver);
    MGL_registerDriver(MGL_SVGA16NAME, SVGA16_driver);
    MGL_registerDriver(MGL_SVGA24NAME, SVGA24_driver);
    MGL_registerDriver(MGL_SVGA32NAME, SVGA32_driver);
    MGL_registerDriver(MGL_VGA8NAME, VGA8_driver);
    MGL_registerDriver(MGL_LINEAR8NAME, LINEAR8_driver);

MGL_registerDriver(MGL_LINEAR16NAME, LINEAR16_driver);

MGL_registerDriver(MGL_LINEAR24NAME, LINEAR24_driver);

MGL_registerDriver(MGL_LINEAR32NAME, LINEAR32_driver);
    MGL_registerDriver(MGL_ACCEL8NAME, ACCEL8_driver);
    MGL_registerDriver(MGL_ACCEL16NAME, ACCEL16_driver);
    MGL_registerDriver(MGL_ACCEL24NAME, ACCEL24_driver);
    MGL_registerDriver(MGL_ACCEL32NAME, ACCEL32_driver);
}
if (useDirectDraw) {
    MGL_registerDriver(MGL_DDRAW8NAME, DDRAW8_driver);
    MGL_registerDriver(MGL_DDRAW16NAME, DDRAW16_driver);
    MGL_registerDriver(MGL_DDRAW24NAME, DDRAW24_driver);
    MGL_registerDriver(MGL_DDRAW32NAME, DDRAW32_driver);
}
driver = grDETECT;
mode = grDETECT;
MGL_detectGraph(&driver, &mode);
...

```

MGL supports a number of different video mode resolutions, ranging from 320x200 up to 1600x1200 with color ranges from 16 colors up to 16.7 million colors. You can construct the mode identifiers easily for every desired resolution, given the name of the mode and the resolution. For instance the 320x200x256 mode is named `grVGA_320x200x256` while the 1280x1024x256 mode is named `grSVGA_1280x1024x256`. Please consult the MGL header files for a complete list of all the available video modes.

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

MGL_modeResolution function to get numeric resolution information about all the available graphics modes, so you can search for specific graphics modes according to the desired resolution and/or color depth.

See Also

MGL_init, *MGL_availableModes*, *MGL_availablePages*, *MGL_modeResolution*

MGL_diffRegion

Compute the Boolean difference of two regions.

Declaration

```
bool MGLAPI MGL_diffRegion(  
    region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.h

Parameters

- | | |
|------------------|---|
| <i>r1</i> | Region from which r2 is subtracted, which also becomes the result region. |
| <i>r2</i> | Region to be subtracted from r1 |

Return Value

True if the difference is valid, false if an empty region was created.

Description

Computes the Boolean difference of two regions by subtracting the area covered by region r2 from region r1, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

See Also

MGL_diffRegionRect, *MGL_unionRegion*, *MGL_sectRegion*

MGL_diffRegionRect

Compute the Boolean difference of a region and a rectangle.

Declaration

```
bool MGLAPI MGL_diffRegionRect(  
    region_t *r1,  
    const rect_t *r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	Region from which r2 is subtracted, which also becomes the result region.
<i>r2</i>	Rectangle to be subtracted from r1

Return Value

True if the difference is valid, false if an empty region was created.

Description

Computes the Boolean difference of a region and a simple rectangle by subtracting the area covered by rectangle r2 from region r1, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

This routine will produce a simple region with only a single bounding rectangle if the original region was also a simple rectangle and the resulting region is also a single rectangle, which makes it more efficient if the region to be subtracted is a rectangle.

See Also

MGL_diffRegion, *MGL_unionRegion*, *MGL_sectRegion*

MGL_disjointRect

Determines if two rectangles are disjoint.

Declaration

```
bool MGL_disjointRect(  
    rect_t r1,  
    rect_t r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	First rectangle to test
<i>r2</i>	Second rectangle to test

Return Value

True if the rectangles are disjoint, false if they overlap.

Description

This function determines whether two rectangles are disjoint, which is true if the rectangles do not overlap at any coordinates.

See Also

MGL_emptyRect, *MGL_equalRect*, *MGL_unionRect*, *MGL_sectRect*

MGL_divotSize

Number of bytes required to store a divot of specified size.

Declaration

```
long MGL_divotSize(  
    MGLDC *dc,  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to measure divot size from
<i>r</i>	Bounding rectangle of the divot area

Return Value

Size of the specified divot in bytes.

Description

This function is the same as *MGL_divotSizeCoord* however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_divotSizeCoord, *MGL_getDivot*, *MGL_putDivot*

MGL_divotSizeCoord

Number of bytes required to store a divot of specified size.

Declaration

```
long MGLAPI MGL_divotSizeCoord(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to measure divot size from
<i>left</i>	Left coordinate of divot area
<i>top</i>	Top coordinate of divot area
<i>right</i>	Right coordinate of divot area
<i>bottom</i>	Bottom coordinate of divot area

Return Value

Size of the specified divot in bytes.

Description

Determines the number of bytes required to store a divot of the specified size taken from the current device context. A divot is a portion of video memory that needs to be temporarily saved and restored, such as implementing pull down menus and pop up dialog boxes. A divot must always be saved and restored to the same area, and will extend the dimensions of the area covered to obtain the maximum possible performance for saving and restoring the memory.

See Also

MGL_divotSize, *MGL_getDivot*, *MGL_putDivot*

MGL_doubleBuffer

Enables double buffering for the specified display device context.

Declaration

```
bool MGLAPI MGL_doubleBuffer(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Return Value

True if double buffering is now enabled, false if not.

Description

Enables double buffered graphics mode for the specified device context if possible. When the device context is in double buffered mode, all active output is sent to the hidden backbuffer, while the current front buffer is being displayed. You then make calls to *MGL_swapBuffers* to swap the front and back buffers so that the previously hidden backbuffer is instantly displayed.

If you intend to start double buffered graphics, you should make sure you call the *MGL_createDisplayDC* function with the double buffer flag set to true, so that some of offscreen video memory will be allocated for the backbuffer. If the device context only has one video page available, double buffering cannot be started and this function will fail.

See Also

MGL_singleBuffer, *MGL_swapBuffers*

MGL_drawBorder

Draws a 3d border around a specified rectangle.

Declaration

```
void MGLAPI MGL_drawBorder(  
    rect_t r,  
    int style,  
    int thickness)
```

Prototype In

mgraph.h

Parameters

<i>r</i>	Rectangle to draw border around (Syntax 2)
<i>style</i>	Style of border to draw (INSET, OUTSET etc)
<i>thickness</i>	Thickness of the border in pixels

Description

This function is the same as *MGL_drawBorderCoord*, however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_drawBorderCoord

MGL_drawBorderCoord

Draws a 3d border around a specified rectangle.

Declaration

```
void MGLAPI MGL_drawBorderCoord(  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int style,  
    int thickness)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left edge of rectangle
<i>right</i>	Right edge of rectangle
<i>bottom</i>	Bottom edge of rectangle
<i>style</i>	Style of border to draw (INSET, OUTSET etc)
<i>thickness</i>	Thickness of the border in pixels

Description

If the style is set to BDR_INSET, the border is colored so that it look like the interior of the rectangle is inset into the display. If style is set to BDR_OUTSET, the border is colored so that the interior looks like it is raised. If style is set to OUTLINE, the border is drawn as a 3d 2 pixels thick outline.

See Also

MGL_drawBorder

MGL_drawGlyph

Draws a monochrome glyph.

Declaration

```
void MGLAPI MGL_drawGlyph(  
    font_t *font,  
    int x,  
    int y,  
    uchar glyph)
```

Prototype In

mgraph.h

Parameters

<i>font</i>	Font containing the glyphs
<i>x</i>	x coordinate to draw glyph at
<i>y</i>	y coordinate to draw glyph at
<i>glyph</i>	Index of glyph to draw

Description

Rasterizes the specified monochrome glyph from the font file in the current color at the specified location. This is effectively the same as drawing a monochrome bitmap, but by storing all your monochrome bitmaps in a font file, the glyphs will be stored as efficiently as possible.

See Also

MGL_rotateGlyph, *MGL_mirrorGlyph*

MGL_drawHDivider

Draws a 3d horizontal dividing line.

Declaration

```
void MGLAPI MGL_drawHDivider(  
    int y,  
    int x1,  
    int x2)
```

Prototype In

mgraph.h

Parameters

y	Y coordinate to draw line at
x1	Starting x coordinate
x2	Ending x coordinate

Description

The line is drawn is two pixels thick, ending at y+1.

MGL_drawRegion

Draw a solid complex region.

Declaration

```
void MGLAPI MGL_drawRegion(  
    int x,  
    int y,  
    const region_t *rgn)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to draw region at
<i>y</i>	y coordinate to draw region at
<i>rgn</i>	region to draw

Description

Draws the complex region at the specified location in the current pattern and write mode.

See Also

MGL_newRegion, *MGL_copyRegion*, *MGL_freeRegion*, *MGL_diffRegion*,
MGL_unionRegion, *MGL_sectRegion*

MGL_drawStr

Draws a text string at the current position.

Declaration

```
void MGLAPI MGL_drawStr(  
    const char *str)
```

Prototype In

mgraph.h

Parameters

str String to display

Description

Draws a 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 MGL_LEFT_TEXT, otherwise the CP is not moved.

See Also

MGL_drawStrXY, MGL_textHeight, MGL_textWidth, MGL_useFont

MGL_drawStrXY

Draws a text string at the specified position.

Declaration

```
void MGLAPI MGL_drawStrXY(  
    int x,  
    int y,  
    const char *str)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to begin rasterizing the string at
<i>y</i>	y coordinate to begin rasterizing the string at
<i>str</i>	String to display

Description

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

See Also

MGL_drawStr, MGL_textHeight, MGL_textWidth, MGL_useFont

MGL_drawVDivider

Draws a 3d vertical dividing line.

Declaration

```
void MGLAPI MGL_drawVDivider(  
    int x,  
    int y1,  
    int y2)
```

Prototype In

mgraph.h

Parameters

x	X coordinate to draw line at
y1	Starting y coordinate
y2	Ending y coordinate

Return Value

void

Description

The line is drawn is two pixels wide, ending at x+1.

MGL_driverName

Returns a string describing the name of the device driver.

Declaration

```
const char * MGLAPI MGL_driverName(  
    int driver)
```

Prototype In

mgraph.h

Parameters

Driver Device driver id number

Return Value

Pointer to device driver name string.

See Also

MGL_modeName, *MGL_modeDriverName*

MGL_ellipse

Draws an ellipse outline.

Declaration

```
void MGLAPI MGL_ellipse(rect_t extentRect)
```

Prototype In

mgraph.h

Parameters

extentRect Bounding rectangle for the ellipse

Description

Draws the outline of an ellipse given the bounding rectangle for the ellipse. The ellipse outline is drawn in the current pen color, style and size just inside the mathematical boundary of the bounding rectangle for the ellipse.

See Also

MGL_ellipseCoord, *MGL_fillEllipse*, *MGL_ellipseArc*, *MGL_fillEllipseArc*

MGL_ellipseArc

Draws an elliptical arc outline.

Declaration

```
void MGLAPI MGL_ellipseArc(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the arc (syntax 2)
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Draws 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 also treated MOD 360. 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 center of the rectangle through it's top right corner, even if the rectangle isn't square. The ellipse outline is drawn in the current pen color, style and size.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*, *MGL_ellipse*, *MGL_fclose*, *MGL_fillEllipse*

MGL_ellipseArcCoord

Draws an elliptical arc outline.

Declaration

```
void MGLAPI MGL_ellipseArcCoord(  
    int x,  
    int y,  
    int xradius,  
    int yradius,  
    int startAngle,  
    int endAngle)
```

Prototype In
mgraph.h

Parameters

<i>x</i>	<i>x</i> coordinate for the center of elliptical arc (syntax 1)
<i>xradius</i>	<i>x</i> radius for the elliptical arc (syntax 1)
<i>yradius</i>	<i>y</i> radius for the elliptical arc (syntax 1)
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Draws the outline of an elliptical given the center and radii for the ellipse.

StartAngle specifies where the arc begins and is treated MOD 360.

EndAngle specifies where the arc ends and is also treated MOD 360. 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 center of the rectangle through it's top right corner, even if the rectangle isn't square. The ellipse outline is drawn in the current pen color, style and size.

Note that this routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics packages). The *MGL_ellipseArc* routine is more versatile than this, as you can have an ellipse with odd diameter values, which you cannot get with the this routine.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*, *MGL_ellipse*, *MGL_fclose*, *MGL_fillEllipse*

MGL_ellipseArcEngine

Generates the set of points on an elliptical arc.

Declaration

```
void MGLAPI MGL_ellipseArcEngine(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle,  
    arc_coords_t *ac,  
    void (ASMAPI *plotPoint)(int x,int y))
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the arc
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)
<i>ac</i>	Place to store computed arc coordinates
<i>plotPoint</i>	Function to call for every point on the elliptical arc

Description

This routine generates the set of points on a elliptical arc, and is the same code used to generate elliptical arcs internally in MGL. You can call it to generate the set of points on an elliptical arc, calling your own plotPoint routine for every point on the arc. The points on the arc are rasterized in order from the starting angle to the ending angle. After the arc has been drawn, the arc coordinates are returned, which contains the actual center, starting and ending points for the arc.

See Also

MGL_ellipseEngine, MGL_lineEngine

Draws an ellipse outline.

Declaration

```
void MGLAPI MGL_ellipseCoord(  
    int x,  
    int y,  
    int xradius,  
    int yradius)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	<i>y</i> coordinate for the center of ellipse (syntax 1)
<i>xradius</i>	x radius for the ellipse (syntax 1)
<i>yradius</i>	y radius for the ellipse (syntax 1)

Description

Draws the outline of an ellipse given the center and radii for the ellipse. The ellipse outline is drawn in the current pen color, style and size just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that this routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics libraries). The *MGL_ellipse* routine is more versatile than this, as you can have an ellipse with odd diameter values, which you cannot get with this routine.

See Also

MGL_ellipse, *MGL_fillEllipse*, *MGL_ellipseArc*, *MGL_fillEllipseArc*

Declaration

```
void MGLAPI MGL_ellipseEngine(  
    rect_t extentRect,  
    void (ASMAPI *setup)(  
        int topY,  
        int botY,  
        int left,  
        int right),  
    void (ASMAPI *set4pixels)(  
        bool inc_x,  
        bool inc_y,  
        bool region1),  
    void (ASMAPI *finished)(void))
```

Prototype In mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the ellipse
<i>setup</i>	Routine called to initialize pixel plotting routines
<i>set4pixels</i>	Routine called repeatedly for each set of 4 pixels
<i>finished</i>	Routine called to complete plotting pixels

Description

This routine generates the set of points on a ellipse, and is the same code used to generate ellipses internally in 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 pixels 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 rasterizing the ellipse if the rasterizing was deferred.

See Also

MGL_ellipseArcEngine, MGL_lineEngine

MGL_emptyRect

Determines if a rectangle is empty.

Declaration

```
bool MGL_emptyRect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r The rectangle to test

Return Value

True if the rectangle is empty, otherwise false.

Description

Determines if a rectangle is empty or not. 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.

MGL_emptyRegion

Determines if a region is empty.

Declaration

```
bool MGLAPI MGL_emptyRegion(  
    const region_t *r)
```

Prototype In

mgraph.h

Parameters

r region to test

Return Value

True if region is empty, false if not.

Description

Determines if a region is empty or not. A region is defined as being empty if the bounding rectangle's 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.

See Also

MGL_equalRegion, MGL_unionRegion, MGL_diffRegion, MGL_sectRegion, MGL_offsetRegion, MGL_ptInRegion, MGL_ptInRegionCoord

MGL_endDirectAccess

Disables direct framebuffer access.

Declaration

```
void WINAPI MGL_endDirectAccess(void)
```

Prototype In

mgraph.h

Description

Disables direct framebuffer access so that you can use the accelerator functions to draw to the framebuffer memory. Note that calling this function is absolutely necessary when using hardware acceleration, as this function and the corresponding *MGL_beginDirectAccess* correctly arbitrate between the hardware accelerator graphics engine and your direct framebuffer writes.

See Also

MGL_beginDirectAccess

MGL_endPixel

Ends high speed pixel drawing operation.

Declaration

```
void WINAPI MGL_endPixel(void)
```

Prototype In

mgraph.h

Description

This function ends a set of high speed pixel drawing operations, started with a call to *MGL_beginPixel*. This routine is intended primarily to ensure fast operation if you intend to plot more than a single pixel at a time.

See Also

MGL_beginPixel

MGL_equalPoint

Compares two points to determine if they are equal.

Declaration

```
bool MGL_equalPoint(  
    point_t p1,  
    point_t p2)
```

Prototype In

mgraph.h

Parameters

p1

The first point to compare.

p2

The second point to compare.

Return Value

True if the points are equal, false if they are not.

MGL_equalRect

Compares two rectangles for equality.

Declaration

```
bool MGL_equalRect(  
    rect_t r1,  
    rect_t r2)
```

Prototype In

mgraph.h

Parameters

r1
r2

First rectangle to compare
Second rectangle to compare

Return Value

True if the rectangles are equal, false if not.

See Also

MGL_equalPoint, *MGL_equalRegion*

MGL_equalRegion

Determines if two regions are equal.

Declaration

```
bool MGLAPI MGL_equalRegion(  
    const region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	First region to compare
<i>r2</i>	Second region to compare

Return Value

True if the regions are equal, false if not.

Description

Determines if two regions are equal, by comparing the bounding rectangles and the definitions for both of the regions.

MGL_errorMsg

Returns a string describing an error condition code.

Declaration

```
const char * MGLAPI MGL_errorMsg(  
    int err)
```

Prototype In

mgraph.h

Parameters

err Error code to obtain string for

Return Value

Pointer to string describing the error condition.

Description

Returns a pointer to a string describing a specified error code. 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.

See Also

MGL_result.

MGL_exit

Closes down the graphics subsystem.

Declaration

```
void MGLAPI MGL_exit(void)
```

Prototype In

mgraph.h

Description

This function closes down the graphics subsystem, deallocating any memory allocated for use by MGL, and restoring the system back into the original text mode that was active before MGL was started. This routine also properly removes all interrupt handlers and other system services that MGL hooked when it was initialized.

You must call this routine before you exit your application, to ensure that the system is properly terminated.

See Also

MGL_init, *MGL_initWindowed*

MGL_fadePalette

Fades the values for a color palette.

Declaration

```
bool ASMAPI MGL_fadePalette(  
    MGLDC *dc,  
    palette_t *fullIntensity,  
    int numColors,  
    int startIndex,  
    uchar intensity)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context with palette to fade
<i>fullIntensity</i>	Pointer to full intensity palette to fade from
<i>numColors</i>	Number of colors in palette to fade
<i>startIndex</i>	Starting index of first color to fade
<i>intensity</i>	Intensity for the final output palette (0 - 255)

Return Value

True if the entire output palette is black, false if not.

Description

This routine will take the values from a full intensity *palette_t* structure, fade the values and store them into a device context palette. The actual hardware palette will not be programmed at this stage, so you will then need to make a call to *MGL_realizePalette* to make the changes visible.

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 and an intensity of 0 produces an all black palette.

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

See Also

MGL_setPalette, MGL_getPalette, MGL_rotatePalette, MGL_realizePalette

MGL_fatalError

Declare a fatal error and exit gracefully.

Declaration

```
void MGLAPI MGL_fatalError(  
    const char *msg)  
void MGL_fatalError(const char *msg)
```

Prototype In

mgraph.h

Parameters

<i>msg</i>	Message to display
-------------------	--------------------

Description

A fatal internal error has occurred, so we shutdown the graphics systems, display the error message and quit. You should call this function to display your own internal fatal errors.

MGL_fclose

Closes an open disk file.

Declaration

```
int MGLAPI MGL_fclose(  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

f Pointer to file to close

Return Value

0 on success, EOF on an error.

Description

This function is identical to the C library `fclose` function, but goes via MGL's internal file handling function pointers, which by default simply point to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with `MGL_setFileIO`, for custom I/O handling.

MGL_fillEllipse

Fills an ellipse.

Declaration

```
void MGLAPI MGL_fillEllipse(  
    rect_t extentRect)
```

Prototype In

mgraph.h

Parameters

ExtentRect

Bounding rectangle defining the ellipse

Description

Fills an ellipse given either the center and radii for the ellipse, or the bounding rectangle for the ellipse. The ellipse is filled in the current pen color and style just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that while this routine can only work with integer semi-major and semi-minor axes, it can sometimes be easier to work with (and is provided for compatibility with other graphics packages). *MGL_fillEllipseCoord* is a more versatile routine, as it allows ellipses with odd diameter values, which you cannot get with *MGL_fillEllipse*.

See Also

MGL_ellipse, *MGL_ellipseArc*, *MGL_fillEllipseArc*, *MGL_fillEllipseCoord*

MGL_fillEllipseArc

Fills an elliptical arc.

Declaration

```
void MGLAPI MGL_fillEllipseArc(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

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 also treated MOD 360. 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 center of the rectangle through it's top right corner, even if the rectangle isn't square.

This routine is more versatile than *MGL_fillEllipseArcCoord*, as it allows an ellipse with odd diameter values, which you cannot get with the *MGL_fillEllipseArcCoord*.

See Also

MGL_fillEllipseArc, *MGL_ellipseArc*, *MGL_ellipse*, *MGL_fclose*, *MGL_fillEllipse*

MGL_fillEllipseArcCoord

Fills an elliptical arc.

Declaration

```
void MGLAPI MGL_fillEllipseArcCoord(  
    int x,  
    int y,  
    int xradius,  
    int yradius,  
    int startAngle,  
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate for center of arc
<i>xradius</i>	x radius for the arc
<i>yradius</i>	y radius for the arc
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Fills an elliptical arc given the center and radii for the ellipse. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is also treated MOD 360. 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 center of the rectangle through it's top right corner, even if the rectangle isn't square.

See Also

MGL_fillEllipseArc, *MGL_ellipseArc*, *MGL_ellipse*, *MGL_fclose*, *MGL_fillEllipse*

MGL_fillEllipseCoord

Fills an ellipse.

Declaration

```
void MGLAPI MGL_fillEllipseCoord(  
    int x,  
    int y,  
    int xradius,  
    int yradius)
```

Prototype In

mgraph.h

Parameters

x	x coordinate for center of ellipse (syntax 1)
y	y coordinate for center of ellipse (syntax 1)
xradius	x radius for the ellipse (syntax 1)
yradius	y radius for the ellipse (syntax 1)
extentRect	Bounding rectangle defining the ellipse (syntax 2)

Description

Fills an ellipse given either the center and radii for the ellipse, or the bounding rectangle for the ellipse. The ellipse is filled in the current pen color and style just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that the first routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics packages). The second routine is more versatile than the first, as you can have an ellipse with odd diameter values, which you cannot get with the first routine.

See Also

MGL_ellipse, *MGL_ellipseArc*, *MGL_fillEllipseArc*

MGL_fillPolygon

Fills an arbitrary polygon.

Declaration

```
void MGLAPI MGL_fillPolygon(  
    int count,  
    point_t *vArray,  
    int xOffset,  
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polygon
<i>vArray</i>	Array of vertices in polygon
<i>xOffset</i>	x coordinate offset value
<i>yOffset</i>	y coordinate offset value

Description

This function is provided for backwards compatibility, and expects the array of points to be passed in as integers. Internally SciTech MGL works in fixed point, so this function simply converts the coordinates to fixed point and passes them to the *MGL_fillPolygonFX* function. Hence *MGL_fillPolygonFX* is more efficient, so you should use that version instead.

See Also

MGL_setPolygonType

MGL_fillPolygonCnvx

Scan converts a filled convex polygon.

Declaration

```
void MGLAPI MGL_fillPolygonCnvx(  
    int count,  
    point_t *vArray,  
    int xOffset,  
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function is provided for backwards compatibility, and expects the array of points to be passed in as integers. Internally SciTech MGL works in fixed point, so this function simply converts the coordinates to fixed point and passes them to the *MGL_fillPolygonCnvxFX* function. Hence *MGL_fillPolygonCnvxFX* is more efficient, so you should use that version instead.

Note: All vertices are offset by (xOffset,yOffset).

See Also

MGL_fillPolygonCnvxFX

MGL_fillPolygonCnvxFX

Scan converts a filled convex polygon.

Declaration

```
void ASMAPI MGL_fillPolygonCnvxFX(  
    int count,  
    fxpoint_t *vArray,  
    int vinc,  
    fix32_t xOffset,  
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

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 and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: *All vertices are offset by (xOffset,yOffset).*

See Also

MGL_fillPolygonFX

MGL_fillPolygonFX

Fills an arbitrary polygon.

Declaration

```
void ASMAPI MGL_fillPolygonFX(  
    int count,  
    fxpoint_t *vArray,  
    int vinc,  
    fix32_t xOffset,  
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polygon
<i>vArray</i>	Array of vertices in polygon
<i>vinc</i>	Increment to get to next vertex in bytes
<i>xOffset</i>	x coordinate offset value
<i>yOffset</i>	y coordinate offset value

Description

These routines rasterize a filled arbitrary polygon in the current color and style. By default the routine will determine the type of the polygon being rasterized, and will rasterize convex polygons using a faster scan conversion routine, otherwise a general polygon scan conversion routine will be used. Thus you can rasterize 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 non-simple).

Non-simple or self intersecting polygons will be rasterized 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 subsequent active edges in the polygon.

You may also use the *MGL_setPolygonType* routine to specify the type of polygons being rasterized. This may be MGL_AUTO_POLYGON,

MGL_CONVEX_POLYGON or MGL_COMPLEX_POLYGON. Explicitly setting the polygon type will speed the drawing process.

As with all MGL polygon rasterizing routines, this routine does not rasterize the pixels down the right hand side or the bottom edges of the polygon. This ensures that pixels along shared edges of polygons are not rasterized twice, which can cause annoying pixel flashes in animation code. Note also that the edges in the polygon will always be rasterized from top to bottom, to ensure that all shared edges will actually generate the same set of vertices, eliminating the possibility of pixel dropouts between shared edges in polygons.

See Also

MGL_setPolygonType

MGL_fillRect

Draws a filled rectangle.

Declaration

```
void MGL_fillRect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to be filled (syntax 2)

Description

This function is the same as *MGL_fillRectCoord*, however it takes an entire rectangle as the parameter instead of coordinates.

See Also

MGL_fillRectCoord, *MGL_fillRectPt*, *MGL_rect*

MGL_fillRectCoord

Draws a filled rectangle.

Declaration

```
void MGLAPI MGL_fillRectCoord(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left coordinate of rectangle (syntax 1)
<i>right</i>	Right coordinate of rectangle (syntax 1)
<i>bottom</i>	Bottom coordinate of rectangle (syntax 1)

Description

Fills a rectangle in the current drawing attributes. The mathematical definition of a rectangle does not include the right and bottom edges, so effectively the right and bottom edges are not rasterized (solving problems with shared edges).

See Also

MGL_fillRect, *MGL_fillRectPt*, *MGL_rect*

MGL_fillRectPt

Draws a filled rectangle.

Declaration

```
void MGL_fillRectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Parameters

leftTop

Top left coordinate of rectangle (syntax 3)

rightBottom

Bottom right coordinate of rectangle (syntax 3)

Description

This function is the same as *MGL_fillRectCoord*, however it takes the top left and bottom right coordinates of the rectangle as two points instead of four coordinates.

See Also

MGL_fillRect, *MGL_fillRectPt*, *MGL_rect*

MGL_fopen

Opens a stream.

Declaration

```
FILE * MGLAPI MGL_fopen(  
    const char *filename,  
    const char *mode)
```

Prototype In

mgraph.h

Parameters

fmode

Filename

Mode to open file in. Refer to your compiler documentation for available modes and values.

Return Value

Pointer to newly opened stream, or NULL in the event of an error.

Description

Attempts to open the specified MGL file in binary mode. This routine will use the standard MGL directory searching algorithm to find the specified file. First an attempt is made to locate the file relative to the `_MGL_path` variable (initialized by the application program via the `MGL_init` call). If this fails, an attempt is made to search for the file relative to the `MGL_ROOT` environment variable if this is present. Lastly MGL searches the current directory for the file (without adding the dir extension). Otherwise it returns NULL.

Reads data from a stream.

Declaration

```
size_t MGLAPI MGL_fread(  
    void *ptr,  
    size_t size,  
    size_t n,  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

<i>ptr</i>	Pointer to block in stream at which to begin read
<i>size</i>	Size of items to be read from stream
<i>n</i>	Number of items to be read from stream
<i>f</i>	Stream to be read

Return Value

Number of items read in, or a short count (possibly 0).

Description

This function is identical to the C library fread function, but goes via MGL's internal file handling function pointers, which by default simply point to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with MGL_setFileIO, for custom I/O handling.

MGL_free

Frees a block of memory.

Declaration

```
void MGLAPI MGL_free(  
    void _HUGE *p)
```

Prototype In

mgraph.h

Parameters

p Pointer to memory block to free

Description

Frees a block of memory previously allocated with either *MGL_calloc* or *MGL_malloc* which are designed to work with memory blocks larger than 64Kb in size even for 16 bit real mode code.

See Also

MGL_calloc, *MGL_malloc*, *MGL_memset*

MGL_freeRegion

Frees all the memory allocated by the complex region.

Declaration

```
void MGLAPI MGL_freeRegion(  
    region_t *r)
```

Prototype In

mgraph.h

Parameters

r Pointer to the region to free

Description

Frees all the memory allocated by the complex region. When you are finished with a complex region you must free it to free up the memory used to represent the union of rectangles.

See Also

MGL_newRegion, *MGL_copyRegion*

Repositions the file pointer on a stream.

Declaration

```
int MGLAPI MGL_fseek(  
    FILE *f,  
    long offset,  
    int whence)
```

Prototype In

mgraph.h

Parameters

<i>Offset</i>	Stream of interest
<i>whence</i>	Offset of location from whence
	New location of file pointer

Return Value

0 if move was successful, otherwise non-zero.

Description

This function is identical to the C library fseek function, but goes via MGL's internal file handling function pointers, which by default simply point to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with MGL_setFileIO, for custom I/O handling.

MGL_ftell

Returns the current file pointer.

Declaration

```
long MGLAPI MGL_ftell(  
    FILE *f )
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Pointer to file of interest
-----------------	-----------------------------

Return Value

Current file pointer on success, -1L on error.

Description

The value returned by *MGL_ftell* can be used in a subsequent call to *MGL_fseek*.

MGL_fwrite

Writes to a stream.

Declaration

```
size_t MGLAPI MGL_fwrite(  
    const void *ptr,  
    size_t size,  
    size_t n,  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

<i>ptr</i>	Pointer to the starting location of data to be written
<i>n</i>	Number of items to be written to file
<i>f</i>	Pointer to the file stream to write the data to

Return Value

The number of items written.

Description

Appends *n* items of data each of length *size* bytes to the given output file. The data written begins at *ptr*. The total number of bytes written is (*n* x *size*).

MGL_getActivePage

Returns the currently active hardware display page.

Declaration

```
int MGLAPI MGL_getActivePage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Currently active hardware display page.

Description

This function returns the currently active hardware display page number. The first hardware display page is 0, the second is 1 and so on. The number of available hardware 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 display pages.

All MGL output is always sent to the currently active hardware display page, and changing the active and visual display pages is used to implement double buffering.

See Also

MGL_setActivePage, *MGL_getVisualPage*, *MGL_setVisualPage*.

MGL_getArcCoords

Returns the starting and ending arc coordinates.

Declaration

```
void WINAPI MGL_getArcCoords(  
    arc_coords_t *coords)
```

Prototype In

mgraph.h

Parameters

coords Pointer to structure to store coordinates

Description

This function returns the center coordinate, and starting and ending points on the ellipse that define the last elliptical arc that was rasterized. You can then use these coordinates to draw a line from the center of the ellipse to the starting and ending points to complete the outline of an elliptical wedge.

Note that you must call this routine immediately after calling the *MGL_ellipseArc* family of routines.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*

MGL_getAspectRatio

Returns the current device context aspect ratio.

Declaration

```
int MGLAPI MGL_getAspectRatio(void)
```

Prototype In

mgraph.h

Return Value

Current video mode aspect ratio * 1000.

Description

This function 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 context aspect ratio can be used to display circles and squares on the 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 code:

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

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

See Also

MGL_setAspectRatio

MGL_getAttributes

Returns a copy of the current rasterizing attributes.

Declaration

```
void MGLAPI MGL_getAttributes(  
    attributes_t *attr)
```

Prototype In

mgraph.h

Parameters

attr Pointer to structure to store attribute values in

Description

This function returns a copy of the currently active attributes. You can use this routine to save the state of MGL and later restore this state with the *MGL_restoreAttributes* routine.

See Also

MGL_restoreAttributes

MGL_getBackColor

Returns the current background color value.

Declaration

```
color_t MGLAPI MGL_getBackColor(void)
```

Prototype In

mgraph.h

Return Value

Current background color value.

Description

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 MGL_BITMAP_OPAQUE fill mode.

See Also

MGL_setBackColor, *MGL_getColor*, *MGL_setColor*

MGL_getBitmapFromDC

Copy a portion of a device context as a lightweight memory bitmap.

Declaration

```
bitmap_t * MGLAPI MGL_getBitmapFromDC(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    bool savePalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save from
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save
<i>savePalette</i>	Save palette with bitmap.

Return Value

Pointer to allocated bitmap, NULL on error.

Description

This function copies a portion of a device context as a lightweight memory bitmap. If this function fails (for instance if out of memory), it will return NULL and you can get the error code from the *MGL_result* function. Otherwise this function will return a pointer to a new lightweight bitmap containing the bitmap data.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_loadBitmap, *MGL_saveBitmapFromDC*

MGL_getBitmapSize

Obtain the dimensions of a bitmap file from disk.

Declaration

```
bool MGLAPI MGL_getBitmapSize(  
    const char *bitmapName,  
    int *width,  
    int *height,  
    int *bitsPerPixel,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>bitmapName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if the bitmap was found, false if not.

Description

This functions loads all the header information for a bitmap file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadBitmapIntoDC* function.

See Also

MGL_loadBitmap, *MGL_loadBitmapExt*

MGL_getBitmapSizeExt

Obtain the dimensions of a bitmap from an open file.

Declaration

```
bool MGLAPI MGL_getBitmapSizeExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int *width,  
    int *height,  
    int *bitsPerPixel,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Pointer to opened bitmap file
<i>dwSize</i>	Size of the bitmap file
<i>dwOffset</i>	Offset into the file
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format

Return Value

True if the bitmap was found, otherwise false.

Description

This function is the same as *MGL_getBitmapSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_getBitmapSize

MGL_getBitsPerPixel

Returns the pixel depth for the device context.

Declaration

```
int MGLAPI MGL_getBitsPerPixel(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Pointer to device context of interest

Return Value

Pixel depth for the device context.

See Also

MGL_getPixelFormat

MGL_getBorderColors

Returns the border colors of the current device context.

Declaration

```
void MGLAPI MGL_getBorderColors(  
    color_t *bright,  
    color_t *dark)
```

Prototype In

mgraph.h

Parameters

bright

Place to store the value for the bright component of the border color

dark

Place to store the value for the dark component of the border color

Description

The border colors are the two colors used to draw horizontal, vertical and rectangular borders with the *MGL_drawBorder* function.

MGL_getCP

Returns the current position value.

Declaration

```
void MGLAPI MGL_getCP(  
    point_t *CP)
```

Prototype In

mgraph.h

Parameters

CP

Place to store the current position

Description

Returns the current position (CP). The CP is the current logical 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_getCharMetrics

Computes the character metrics for a specific character.

Declaration

```
void MGLAPI MGL_getCharMetrics(  
    char ch,  
    metrics_t *m)
```

Prototype In

mgraph.h

Parameters

<i>ch</i>	Character to measure
<i>metrics</i>	Place to store the resulting metrics

Description

This function computes the character metrics for a specific character. The character metrics define specific characters width, height, ascent, descent and other values. These values can then be used to correctly position the character with pixel precise positioning.

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

See Also

MGL_getFontMetrics

MGL_getClipMode

Returns the current clipping mode.

Declaration

```
bool MGLAPI MGL_getClipMode(void)
```

Prototype In

mgraph.h

Return Value

True if clipping is on, false if not.

Description

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

See Also

MGL_getClipModeDC, MGL_setClipMode, MGL_getClipRect, MGL_setClipRect

MGL_getClipModeDC

Returns the current clipping mode for a specific DC.

Declaration

```
bool MGLAPI MGL_getClipModeDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of the target clipping mode

Return Value

True if clipping is on, false if not.

Description

This function is the same as *MGL_getClipMode*, however the device context does not have to be the current device context.

See Also

MGL_getClipMode, *MGL_setClipMode*, *MGL_getClipRect*, *MGL_setClipRect*

MGL_getClipRect

Returns the current clipping rectangle.

Declaration

```
void MGLAPI MGL_getClipRect(  
    rect_t *clip)
```

Prototype In

mgraph.h

Parameters

clip Place to store the current clipping rectangle

Description

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_getClipRectDC, *MGL_setClipRect*, *MGL_getClipMode*,
MGL_getClipMode.

MGL_getClipRectDC

Returns the current clipping rectangle for a specific DC.

Declaration

```
void MGLAPI MGL_getClipRectDC(  
    MGLDC *dc,  
    rect_t *clip)
```

Prototype In

mgraph.h

Parameters

dc
clip

Display device context in which the rectangle is defined
Place to store the current clipping rectangle

Description

This function is the same as *MGL_getClipRect*, however the device context does not have to be the current device context.

See Also

MGL_getClipRect, *MGL_setClipRect*, *MGL_getClipMode*, *MGL_getClipMode*.

MGL_getColor

Returns the current foreground color.

Declaration

```
color_t MGLAPI MGL_getColor(void)
```

Prototype In

mgraph.h

Return Value

Current foreground color.

See Also

MGL_setColor, *MGL_getBackColor*, *MGL_setBackColor*

MGL_getColorMapMode

Returns the current color map mode.

Declaration

```
int MGLAPI MGL_getColorMapMode(void)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current color map mode.

Description

This function returns the current color map mode for the device context. Defined modes are enumerated in *MGL_colorModes*.

The current color map mode only affects 8 bit display modes.

See Also

MGL_setColorMapMode

MGL_getDefaultPalette

Returns the default palette for the device.

Declaration

```
void MGLAPI MGL_getDefaultPalette(  
    MGLDC *dc,  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>pal</i>	Place to store the default palette values

Description

Copies the default palette for the specified device context into the passed palette structure.

Note: *The size of the default palette can be found with a call to MGL_getPaletteSize().*

See Also

MGL_setPalette, MGL_getPalette

MGL_getDirectDrawActiveSurface

Returns the DirectDraw active surface object for a fullscreen video mode

Declaration

```
MGL_LPDDSURF MGLAPI MGL_getDirectDrawActiveSurface(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc Device context to get DirectDraw active surface object for

Return Value

Pointer to the LPDIRECTDRAWSURFACE object.

Description

This function returns a pointer to the LPDIRECTDRAWSURFACE object for the active surface used in MGL fullscreen graphics modes. The active surface is the currently active surface that all drawing is done to, which may not be the same as the primary surface created by SciTech MGL.

If the fullscreen graphics mode is not a DirectDraw mode, this function returns NULL.

See Also

MGL_getDirectDrawPrimarySurface, *MGL_getDirectDrawOffscreenSurface*

MGL_getDirectDrawObject

Returns the DirectDraw object for a fullscreen video mode

Declaration

```
MGL_LPDD MGLAPI MGL_getDirectDrawObject(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc Device context to get DirectDraw object for

Return Value

Pointer to the LPDIRECTDRAW object.

Description

This function returns a pointer to the LPDIRECTDRAW object (the main DirectDraw object used by SciTech MGL) for fullscreen graphics modes. This is intended to allow application programmers to do custom operations with DirectDraw when SciTech MGL is running in DirectDraw modes (such as creating a Direct3D object and using Direct3D with SciTech MGL).

If the fullscreen graphics mode is not a DirectDraw mode, this function returns NULL.

Note: *If you obtain a copy of the DirectDraw object, don't create your own surfaces, but use the surfaces already created by SciTech MGL (you can attach your own z-buffers and texture memory surfaces to these if you wish to use Direct3D directly).*

See Also

MGL_getDirectDrawPrimarySurface

MGL_getDirectDrawOffscreenSurface

Returns the DirectDraw offscreen surface object for a fullscreen video mode

Declaration

```
MGL_LPDDSURF MGLAPI MGL_getDirectDrawOffscreenSurface(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc Device context to get DirectDraw offscreen surface object for

Return Value

Pointer to the LPDIRECTDRAWSURFACE object.

Description

This function returns a pointer to the LPDIRECTDRAWSURFACE object for the offscreen surface used in MGL fullscreen graphics modes. The offscreen surface is the offscreen video memory surface that SciTech MGL creates when you make a call to *MGL_createOffscreenDC* and the installed DirectDraw drivers support hardware BitBlt acceleration. If the device context passed in is not an offscreen display device context, this function returns NULL.

If the fullscreen graphics mode is not a DirectDraw mode, this function returns NULL.

Note: *SciTech MGL offscreen DirectDraw surface will never be made visible on the screen, and is only used for caching bitmaps in offscreen video memory to be moved around using hardware acceleration.*

See Also

MGL_getDirectDrawPrimarySurface

MGL_getDirectDrawPalette

Returns the DirectDraw palette object for a fullscreen video mode

Declaration

```
MGL_LPDDPAL MGLAPI MGL_getDirectDrawPalette(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc

Device context to get DirectDraw palette object for

Return Value

Pointer to the LPDIRECTDRAWPALETTE object.

Description

This function returns a pointer to the LPDIRECTDRAWPALETTE object used in MGL fullscreen graphics modes. The DirectDraw palette object is the palette that is created by SciTech MGL for use in 8 bits per pixel modes, and will be NULL if the video mode is greater than 8 bits per pixel.

If the fullscreen graphics mode is not a DirectDraw mode, this function returns NULL.

See Also

MGL_getDirectDrawPrimarySurface

MGL_getDirectDrawPrimarySurface

Returns the DirectDraw primary surface object for a fullscreen video mode

Declaration

```
MGL_LPDDSURF MGLAPI MGL_getDirectDrawPrimarySurface(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc

Device context to get DirectDraw primary surface object for

Return Value

Pointer to the LPDIRECTDRAWSURFACE object.

Description

This function returns a pointer to the LPDIRECTDRAWSURFACE object for the primary surface used in MGL fullscreen graphics modes. The primary surface is the main surface that all the back buffer surfaces are attached to (the number of attached buffers will depend on the number you requested when you called *MGL_createDisplayDC*). Note that the primary surface may not be the currently active surface for drawing; to get this surface call *MGL_getDirectDrawActiveSurface* instead.

If the fullscreen graphics mode is not a DirectDraw mode, this function returns NULL.

See Also

MGL_getDirectDrawActiveSurface, *MGL_getDirectDrawOffscreenSurface*

MGL_getDisplayStart

Returns the current hardware display starting coordinates.

Declaration

```
void MGLAPI MGL_getDisplayStart(  
    MGLDC *dc,  
    int *x,  
    int *y)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Hardware scrolling device context
<i>x</i>	Place to store the display start x coordinate
<i>y</i>	Place to store the display start y coordinate

Description

This function returns the current hardware display start coordinates for the hardware scrolling display device context. You can change the display start address with the *MGL_setDisplayStart* function.

See Also

MGL_setDisplayStart

MGL_getDivot

Saves a divot of video memory into system RAM.

Declaration

```
void MGL_getDivot(  
    MGLDC dc,  
    rect_t,  
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save divot from
<i>r</i>	Rectangle containing coordinates of divot to save
<i>divot</i>	Pointer to area to store the video memory in

Description

This function is the same as *MGL_getDivotCoord* however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_putDivot, *MGL_divotSize*, *MGL_malloc*

MGL_getDivotCoord

Saves a divot of video memory into system RAM.

Declaration

```
void MGLAPI MGL_getDivotCoord(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save divot from
<i>left</i>	Left coordinate of area to save
<i>top</i>	Top coordinate of area to save
<i>right</i>	Right coordinate of area to save
<i>bottom</i>	Bottom coordinate of area to save
<i>divot</i>	Pointer to area to store the video memory in

Description

This function copies a block of video memory from the active page of the current device context into a system RAM buffer. A divot is defined as being a rectangular area of video memory that you wish to save, however the bounding rectangle for the divot is expanded slightly to properly aligned boundaries for the absolute maximum performance with the current device context. This function is generally used to store the video memory behind pull down menus and pop up dialog boxes, and the memory can only be restored to exactly the same position that it was saved from.

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. Note also that the memory block for the divot may be larger than 64Kb in size, so you should use the *MGL_malloc* family of functions to allocate the memory block.

See Also

MGL_getDivot, MGL_putDivot, MGL_divotSize, MGL_malloc

MGL_getDriver

Returns the current device context driver ID.

Declaration

```
int MGLAPI MGL_getDriver(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current device context driver id.

Description

This function returns the numerical id of the currently active display device driver for the device context. This value is only meaningful for display device drivers where the actual hardware device driver can be different. This value can be converted to a printable representation with the *MGL_driverName* routine.

See Also

MGL_driverName, *MGL_getMode*

MGL_getFont

Declaration

```
font_t * MGLAPI MGL_getFont(void)
```

Prototype In

mgraph.h

Return Value

Pointer to currently active font.

Description

Returns a pointer to the currently active font. The currently active font is used to perform all text output by MGL.

See Also

MGL_useFont, *MGL_loadFont*, *MGL_unloadFont*

MGL_getFontMetrics

Returns the currently active font metrics.

Declaration

```
void MGLAPI MGL_getFontMetrics(  
    metrics_t *m)
```

Prototype In

mgraph.h

Parameters

metrics Place to store the font metrics

Description

This function computes the font metrics for the current font. The metrics are computed in pixels and will be as accurate as possible given the current font's scaling factor (only vector fonts can be scaled however).

See Also

MGL_getCharMetrics

MGL_getFullScreenWindow

Returns the current fullscreen window handle.

Declaration

```
MGL_HWND MGLAPI MGL_getFullScreenWindow(void)
```

Prototype In

mglwin.h

Return Value

Current fullscreen window handle.

Description

This function returns the handle to the current fullscreen window. When you are running in fullscreen modes under Windows, MGL always maintains a fullscreen, topmost window that is used for event handling.

If you are using the DirectSound libraries for sound output, you will need to inform DirectSound what your fullscreen window is so that it can correctly mute the sound for your application when the focus is lost to another application. If you do not do this, when you switch to fullscreen modes all sound output via DirectSound will be muted (assuming you are requesting exclusive mode).

See Also

MGL_registerEventProc

MGL_getGlyphHeight

Returns the height of the glyphs in the specified font.

Declaration

```
int MGLAPI MGL_getGlyphHeight(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font Font of interest

Return Value

Height of all the glyphs in the specified font.

See Also

MGL_getGlyphWidth

MGL_getGlyphWidth

Returns the width of a specified glyph in the specified font.

Declaration

```
int MGLAPI MGL_getGlyphWidth(  
    font_t *font,  
    uchar glyph)
```

Prototype In

mgraph.h

Parameters

<i>font</i>	Font of interest
<i>glyph</i>	Index of glyph in font file to measure

Return Value

Returns the width of the specified glyph.

See Also

MGL_getGlyphHeight

MGL_getHalfTonePalette

Returns a copy of the MGL halftone palette.

Declaration

```
void MGLAPI MGL_getHalfTonePalette(  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

pal Place to store the halftone palette values

Description

This function copies the MGL halftone palette into the specified palette structure. The halftone palette always contains 256 colors, and hence the palette array must contain 256 palette entries. This palette is a special palette used by MGL when running in RGB dithered rasterizing mode for 8 bit video modes. If you intend to enable RGB dithering with the *MGL_setColorMapMode* function, you must get a copy of the halftone palette and program the hardware palette for your display device or windowed device to be the same as this halftone palette.

Note that the MGL halftone palette is compatible with the standard Windows halftone palette, so you can perform 8 bit dithering operations in a window without needing to go into SYSPAL_STATIC mode.

See Also

MGL_setPalette, *MGL_realizePalette*, *MGL_setColorMapMode*

MGL_getHardwareFlags

Returns the current hardware acceleration flags for the display device context.

Declaration

```
long MGLAPI MGL_getHardwareFlags(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current hardware acceleration flags for the device context.

Description

This function returns the current hardware acceleration flags for the display device context (this function returns 0 for non-display device contexts). The set of hardware acceleration flags inform the application of what specific hardware acceleration features the underlying video hardware has, so that the application can tailor its use of specific MGL functions. For instance an application may check if the hardware has transparent BitBlt capabilities for sprite animation, and if not will use its own application specific set of routines that rasterize directly to the display surface rather than using the MGL specific functions.

The set of hardware acceleration flags that are returned will be a logical combination of one or more of the values enumerated in *MGL_hardwareFlagsType*.

MGL_getJPEGSize

Obtain the dimensions of a JPEG file from disk.

Declaration

```
bool MGLAPI MGL_getJPEGSize(  
    const char *JPEGName,  
    int *width,  
    int *height,  
    int *bitsPerPixel,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>JPEGName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if the JPEG file was found, false if not.

Description

This functions loads all the header information for a JPEG file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadJPEGIntoDC* function.

Note that JPEG files are inherently 24-bit, so when you call this function it will always return information for a 24-bit RGB pixel format image.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_loadJPEG, *MGL_loadJPEGIntoDC*

MGL_getJPEGSizeExt

Obtain the dimensions of a JPEG file from an opened file.

Declaration

```
bool MGLAPI MGL_getJPEGSizeExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int *width,  
    int *height,  
    int *bitsPerPixel,  
    pixel_format_t *pf)
```

Prototype In
mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file
<i>dwSize</i>	Size of JPEG file
<i>width</i>	Width of bitmap
<i>height</i>	Height of bitmap
<i>bitsPerPixel</i>	Pixel depth of bitmap
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if JPEG was found, false if not.

Description

This function is the same as *MGL_getJPEGSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

Note that JPEG files are inherently 24-bit, so when you call this function it will always return information for a 24-bit RGB pixel format image.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_getJPEGSize

MGL_getLineStipple

Returns the current line stipple pattern.

Declaration

```
ushort MGLAPI MGL_getLineStipple(void)
```

Prototype In

mgraph.h

Description

Return the current line stipple pattern. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getLineStippleCount

Returns the current line stipple counter.

Declaration

```
uint MGLAPI MGL_getLineStippleCount(void)
```

Prototype In

mgraph.h

Description

Return the current line stipple counter. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getLineStyle

Returns the current line style.

Declaration

```
int MGLAPI MGL_getLineStyle(void)
```

Prototype In

mgraph.h

Description

Returns the current line style. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getMarkerColor

Returns the current marker color value.

Declaration

```
color_t MGLAPI MGL_getMarkerColor(void)
```

Prototype In

mgraph.h

Return Value

Current marker color value.

Description

Returns the current marker color value. The marker color is used when drawing markers with the *MGL_marker* routine.

See Also

MGL_setMarkerColor, *MGL_marker*, *MGL_polyMarker*

MGL_getMarkerSize

Returns the current marker size value.

Declaration

```
int MGLAPI MGL_getMarkerSize(void)
```

Prototype In

mgraph.h

Return Value

Current marker size

Description

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. A marker size of 1 will define a marker that is contained within a rectangle 3 pixels wide.

See Also

MGL_setMarkerSize, *MGL_setMarkerStyle*, *MGL_getMarkerStyle*, *MGL_marker*, *MGL_polyMarker*

MGL_getMarkerStyle

Returns the current marker style.

Declaration

```
int MGLAPI MGL_getMarkerStyle(void)
```

Prototype In

mgraph.h

Return Value

Current marker style value.

Description

Returns the current marker style value. The marker style defines the type of marker to be rasterized. Marker styles defined in the SciTech MGL are enumerated in *MGL_markerStyleType*.

See Also

MGL_setMarkerSize, *MGL_getMarkerSize*, *MGL_setMarkerStyle*, *MGL_marker*, *MGL_polyMarker*

MGL_getMode

Returns the current video mode number for the display device context.

Declaration

```
int MGLAPI MGL_getMode(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current video mode number.

Description

Returns the currently active video mode number. This number can be converted to a printable form using the *MGL_modeName* routine.

See Also

MGL_init, *MGL_modeName*

MGL_getPCXSize

Obtain the dimensions of a PCX file from disk.

Declaration

```
bool MGLAPI MGL_getPCXSize(  
    const char *PCXName,  
    int *width,  
    int *height,  
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>PCXName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth

Return Value

True if the PCX file was found, false if not.

Description

This functions loads all the header information for a PCX file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadPCXIntoDC* function.

See Also

MGL_loadPCX, *MGL_loadPCXIntoDC*

MGL_getPCXSizeExt

Obtain the dimensions of a PCX file from an opened file.

Declaration

```
bool MGLAPI MGL_getPCXSizeExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int *width,  
    int *height,  
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file
<i>dwSize</i>	Size of PCX file
<i>width</i>	Width of bitmap
<i>height</i>	Height of bitmap
<i>bitsPerPixel</i>	Pixel depth of bitmap

Return Value

True if PCX was found, false if not.

Description

This function is the same as *MGL_getPCXSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_getPCXSize

MGL_getPalette

Returns the currently active palette values.

Declaration

```
void MGLAPI MGL_getPalette(  
    MGLDC *dc,  
    palette_t *pal,  
    int numColors,  
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>pal</i>	Place to store the retrieved values
<i>numColors</i>	Number of color values to retrieve
<i>startIndex</i>	Starting palette index value to retrieve

Description

This function copies 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

Returns the value of a single palette entry.

Declaration

```
void MGLAPI MGL_getPaletteEntry(  
    MGLDC *dc,  
    int entry,  
    uchar *red,  
    uchar *green,  
    uchar *blue)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>entry</i>	Palette index to read
<i>red</i>	Place to store the red component
<i>green</i>	Place to store the green component
<i>blue</i>	Place to store the blue component

Description

This function returns the value of a single color palette entry. If you wish to obtain more than a single palette entry you should use the *MGL_getPalette* routine which is faster for multiple entries.

See Also

MGL_setPaletteEntry, *MGL_getPalette*, *MGL_setPalette*

MGL_getPaletteSize

Returns the number of entries in the entire palette.

Declaration

```
int MGLAPI MGL_getPaletteSize(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Number of entries in entire palette.

Description

This function returns the number of entries in the entire palette. You should use this function to determine the size of the entire palette, since the palette is still available in HiColor and TrueColor video modes. For RGB modes the palette is implemented in software rather than hardware, and is used for translating color index values to RGB color values, such as when displaying color index bitmaps in RGB modes.

See Also

MGL_getPalette

MGL_getPaletteSnowLevel

Returns the current palette snow level.

Declaration

```
int MGLAPI MGL_getPaletteSnowLevel(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current palette snow level.

Description

This function returns the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. By default MGL programs use all 256 entries per retrace, but you may need to slow this down on systems with slower hardware that causes snow during multiple palette realization commands.

See Also

MGL_setPaletteSnowLevel

MGL_getPenBitmapPattern

Returns the currently active bitmap pattern.

Declaration

```
void MGLAPI MGL_getPenBitmapPattern(  
    pattern_t *pat)
```

Prototype In

mgraph.h

Parameters

pat Place to store the bitmap pattern

Description

This function copies the currently active bitmap pattern used when rasterizing patterned primitive in the MGL_BITMAP_ TRANSPARENT and MGL_BITMAP_ OPQAUE pen styles. A bitmap pattern is defined as an 8 x 8 pixel monochrome pattern stored as an array of 8 bytes.

When filling in the MGL_BITMAP_ 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 MGL_BITMAP_ 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_setPenPixmapPattern, MGL_setPenStyle, MGL_getPenStyle

MGL_getPenPixmapPattern

Returns the currently active pixmap pattern.

Declaration

```
void MGLAPI MGL_getPenPixmapPattern(  
    pixpattern_t *pat)
```

Prototype In

mgraph.h

Parameters

pat Place to store the pixmap pattern

Description

This function copies the currently active pixmap pattern used when rasterizing patterned primitive in the MGL_PIXMAP pen style. A pixmap pattern is defined as an 8 x 8 pixel color pattern stored as an 8 x 8 array of MGL color values. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_setPenPixmapPattern, *MGL_setPenBitmapPattern*, *MGL_setPenStyle*, *MGL_getPenStyle*

MGL_getPenSize

Returns the current pen size.

Declaration

```
void MGLAPI MGL_getPenSize(  
    int *height,  
    int *width)
```

Prototype In

mgraph.h

Parameters

~~height~~
~~width~~

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

Description

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 primitives are rasterized 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

Returns the current pen style.

Declaration

```
int MGLAPI MGL_getPenStyle(void)
```

Prototype In

mgraph.h

Return Value

Current pen style.

Description

This function returns the currently active pen style. Pen styles supported by the SciTech MGL are enumerated in *MGL_penStyleType*.

When filling in the MGL_BITMAP_ 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 MGL_BITMAP_ OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_setPenStyle, *MGL_setPenBitmapPattern*, *MGL_setPenPixmapPattern*

MGL_getPixel

Returns the color of a specified pixel.

Declaration

```
color_t MGL_getPixel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function is the same as *MGL_getPixelCoord*, however it takes the coordinate of the pixel to read as a point.

See Also

MGL_getPixelCoord, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelCoord

Returns the color of a specified pixel.

Declaration

```
color_t MGLAPI MGL_getPixelCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate of the pixel to read
<i>y</i>	y coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function returns the color of the pixel at the specified coordinate.

See Also

MGL_getPixel, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelCoordFast

Returns the color of a specified pixel.

Declaration

```
color_t MGLAPI MGL_getPixelCoordFast(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

x	x coordinate of the pixel to read
y	y coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function returns the color of the pixel at the specified coordinate. Note that 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 with these fast pixel routines.

See Also

MGL_getPixelFast, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelFast

Returns the color of a specified pixel.

Declaration

```
color_t MGL_getPixelFast(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function is the same as *MGL_getPixelCoordFast*, however it takes the coordinate of the pixel to read as a point.

See Also

MGL_getPixelCoordFast, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelFormat

Returns the current packed pixel format information.

Declaration

```
void MGLAPI MGL_getPixelFormat(  
    MGLDC *dc,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>format</i>	Place to store the pixel format information

Description

This function returns the current pixel format information for the specified device context. This information is used by MGL to encode the packed pixel information, and can be used by your application to work out how to pack values correctly for the RGB device contexts.

See Also

MGL_packColor, *MGL_unpackColor*

MGL_getPolygonType

Returns the current polygon type.

Declaration

```
int MGLAPI MGL_getPolygonType(void)
```

Prototype In

mgraph.h

Return Value

Current polygon type code.

Description

Returns the current polygon type. You can change this value with the *MGL_setPolygonType* to force MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). Polygon types supported by the SciTech MGL are enumerated in *MGL_polygonType*.

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rasterizing. Note that this setting does not affect the specialized triangle and quadrilateral rasterizing routines.

See Also

MGL_setPolygonType, *MGL_fillPolygon*

MGL_getSpaceExtra

Returns the current space extra value.

Declaration

```
int MGLAPI MGL_getSpaceExtra(void)
```

Prototype In

mgraph.h

Return Value

Current space extra value.

Description

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).

See Also

MGL_setSpaceExtra, *MGL_drawStr*

MGL_getTextDirection

Returns the current text direction.

Declaration

```
int MGLAPI MGL_getTextDirection(void)
```

Prototype In

mgraph.h

Return Value

Current text direction.

Description

Returns the current text direction. Directions supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_setTextDirection, *MGL_drawStr*

MGL_getTextJustify

Returns the current text justification.

Declaration

```
void MGLAPI MGL_getTextJustify(  
    int *horiz,  
    int *vert)
```

Prototype In

mgraph.h

Parameters

horiz
vert

Place to store horizontal justification
Place to store vertical justification

Description

Returns the current text justification values. Justification types supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_setTextJustify

MGL_getTextSettings

Declaration

```
void MGLAPI MGL_getTextSettings(  
    text_settings_t *settings)
```

Prototype In

mgraph.h

Parameters

settings Place to store the current text settings

Description

Returns a copy of the currently active text settings. This routine provides a way to save and restore all the values relating to the rasterizing of text in MGL with a single function call.

See Also

MGL_setTextSettings

MGL_getTextSize

Returns the current text scaling factors.

Declaration

```
void MGLAPI MGL_getTextSize(  
    int *numerx,  
    int *denomx,  
    int *numery,  
    int *denomy)
```

Prototype In

mgraph.h

Parameters

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

Description

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

Note: *MGL can only scale vector fonts. Bitmap fonts cannot be scaled.*

See Also

MGL_setTextSize

MGL_getTickResolution

Get duration of a timer tick.

Declaration

```
ulong MGLAPI MGL_getTickResolution(void)
```

Prototype In

mgldos.h, mglwin.h

Return Value

Number of seconds in a timer tick * 1,000,000.

Description

This function returns an unsigned long value indicating the duration of a timer tick in seconds multiplied by one million. The duration of a timer tick changes depending on the target environment, so you should use this function to convert the value to a standard representation.

See Also

MGL_getTicks

MGL_getTicks

Get the current timer tick count.

Declaration

```
ulong MGLAPI MGL_getTicks(void)
```

Prototype In

mgldos.h, mglwin.h

Return Value

Current timer tick count as a 32 bit integer.

Description

This function returns the current timer tick as a 32-bit integer value. The number of ticks in a single second can be determined with the MGL_getTickResolution function.

See Also

MGL_getTickResolution

MGL_getViewport

Returns the currently active viewport.

Declaration

```
void MGLAPI MGL_getViewport(  
    rect_t *view)
```

Prototype In

mgraph.h

Parameters

view Place to store the current viewport

Description

This function returns a copy of the currently active viewport. These dimensions are global to the entire device context surface. When the viewport is changed with this function, the viewport origin is reset to (0,0).

All output in 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 device surface.

See Also

*MGL_getViewportDC, MGL_setViewport, MGL_setRelViewport,
MGL_setViewportOrg, MGL_clearViewport, MGL_setClipRect*

MGL_getViewportDC

Returns the currently active viewport for a specific DC.

Declaration

```
void MGLAPI MGL_getViewportDC(  
    MGLDC *dc,  
    rect_t *view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	Place to store the current viewport

Description

This function is the same as *MGL_getViewport*, however the device context does not have to be the current device context.

See Also

MGL_getViewport, *MGL_setViewport*, *MGL_setRelViewport*,
MGL_setViewportOrg, *MGL_clearViewport*, *MGL_setClipRect*

MGL_getViewportOrg

Returns the current viewport origin.

Declaration

```
void MGLAPI MGL_getViewportOrg(  
    point_t *org)
```

Prototype In

mgraph.h

Parameters

org Place to store the viewport origin

Description

This function returns a copy of the currently active viewport origin. When a new viewport is set with the *MGL_setViewport* function, the viewport origin is reset to (0,0), which means that any primitives drawn at pixel location (0,0) will appear at the top left hand corner of the viewport.

You can change the logical coordinate of the viewport origin to any value you please, which will effectively offset all drawing within the currently active viewport. Hence if you set the viewport origin to (10,10), drawing a pixel at (10,10) would make it appear at the top left hand corner of the viewport.

See Also

MGL_getViewportOrgDC, *MGL_setViewport*, *MGL_setViewportOrg*

MGL_getViewportOrgDC

Returns the current viewport origin for a specific DC.

Declaration

```
void MGLAPI MGL_getViewportOrgDC(  
    MGLDC *dc,  
    point_t *org)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>org</i>	Place to store the viewport origin

Description

This function is the same as *MGL_getViewportOrg*, however the device context does not have to be the current device context.

See Also

MGL_getViewportOrg, *MGL_setViewport*, *MGL_setViewportOrg*

MGL_getVisualPage

Returns the currently visible hardware video page.

Declaration

```
int MGLAPI MGL_getVisualPage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Currently visible hardware video page.

Description

This function returns the currently visible hardware video page number for the given device context. 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_setVisualPage, *MGL_getActivePage*, *MGL_setActivePage*

Returns a Windows HDC for drawing on a device context using GDI

Declaration

```
HDC MGLAPI MGL_getWinDC(  
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc

Device context to obtain the Windows HDC for

Return Value

Windows compatible device context handle (HDC) for the device context.

Description

This function allows you to obtain a handle to a Windows compatible device context (HDC) for any MGL device context. Using the returned HDC, you can then call regular GDI drawing functions to draw on the device context, instead of using the MGL drawing functions. This is most useful for drawing objects that are not supported by SciTech MGL, such as Bezier curves and using TrueType fonts.

Note: *It is possible that in some instances (mostly under Windows 3.1) that a valid HDC cannot be created, and this function will return NULL.*

Note: *In order for this function to work for fullscreen device contexts under Windows 95, the MGLDIB.DRV driver must be installed into the WINDOWS\SYSTEM directory on the target machine.*

MGL_getWriteMode

Returns the current write mode operation.

Declaration

```
int MGLAPI MGL_getWriteMode(void)
```

Prototype In

mgraph.h

Return Value

Current write mode operation.

Description

Returns the currently active write mode. Write modes supported by the SciTech MGL for all output primitives are enumerated in *MGL_writeModeType*.

See Also

MGL_setWriteMode

MGL_getX

Returns the x coordinate of the current position.

Declaration

```
int MGLAPI MGL_getX(void)
```

Prototype In

mgraph.h

Return Value

x coordinate of current position

Description

Returns the x coordinate of the current position (CP). The CP is the current graphics cursor position, and is used by a number of output routines to determine where to begin drawing.

See Also

MGL_getY, *MGL_getCP*,

MGL_getY

Returns the y coordinate of the current position.

Declaration

```
int MGLAPI MGL_getY(void)
```

Prototype In

mgraph.h

Return Value

y coordinate of current position

Description

Returns the y coordinate of the current position (CP). The CP is the current graphics cursor position, and is used by a number of output routines to determine where to begin drawing.

See Also

MGL_getX, *MGL_getCP*

MGL_glChooseVisual

Choose an OpenGL visual to best match the passed in visual.

Declaration

```
void MGLAPI MGL_glChooseVisual(  
    MGLDC *dc,  
    MGLVisual *visual)
```

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Structure containing OpenGL visual information

Description

This function examines the visual passed in and modifies the values to best match the capabilities of the underlying OpenGL implementation. If a requested capability is not supported, the structure will be modified for the capabilities that the OpenGL implementation does support (i.e.: lowering the depth buffer size to 16 bits etc).

See Also

MGL_glSetVisual, *MGL_glCreateContext*

MGL_glCreateContext

Creates the OpenGL rendering context for the MGL device context.

Declaration

```
bool MGLAPI MGL_glCreateContext(  
    MGLDC *dc,  
    int flags)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context to enable OpenGL API for
<i>flags</i>	Flags to force type of OpenGL library used

Return Value

True if context was created successfully, false otherwise

Description

This function creates an OpenGL rendering context for the MGL device context, and enables support for OpenGL functions for the MGL device context. To provide a quick and easy way to get SciTech MGL up and running with OpenGL support, you can pass in some simple flags that let SciTech MGL know what OpenGL features you want enabled in the OpenGL visual for the MGL device context, by combining values in the MGL_glContextType enumeration. For instance if you wanted to start OpenGL with support for an RGB, double buffered and z-buffered mode you would pass in:

MGL_GL_RGB | MGL_GL_DOUBLE | MGL_GL_DEPTH

If you wish to have complete control over the OpenGL visual that is used for the MGL device context, you can call *MGL_glChooseVisual* and *MGL_glSetVisual* before calling *MGL_glCreateContext*, and then pass in a value of MGL_GL_VISUAL to tell SciTech MGL to use the visual you have already set for the device context instead of trying to create one from the passed in flags.

Note: *Once you have created an OpenGL rendering context for the MGL device context, you must first call `MGL_glMakeCurrent` to make that specific MGL device context the current OpenGL rendering context before you call any core OpenGL API functions.*

Note: *You must call this function first before you attempt to make any calls to the core OpenGL API functions (such as `glVertex3f` etc), as SciTech MGL will not have initialised its internal dispatch tables until this call is made.*

See Also

`MGL_glChooseVisual`, `MGL_glSetVisual`, `MGL_glMakeCurrent`

MGL_glDeleteContext

Delete the OpenGL rendering context associated with the MGL device context.

Declaration

```
void MGLAPI MGL_glDeleteContext(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc

MGL device context to delete OpenGL rendering context for

Description

This function destroys the OpenGL rendering context for the MGL device context, and calls OpenGL to ensure that no rendering context is still current. You must call this function before you destroy and MGL device context if you have enabled OpenGL via *MGL_glCreateContext*.

Note: *After you have called this function, it is an error to make calls to the OpenGL API as OpenGL will not have a current rendering context active.*

See Also

MGL_glCreateContext, *MGL_glMakeCurrent*

MGL_glEnumerateDrivers

Enumerates the names of all available OpenGL implementations.

Declaration

```
const char ** MGLAPI MGL_glEnumerateDrivers(void)
```

Prototype In

mgraph.h

Return Value

Pointer to a NULL terminated list of available OpenGL driver names.

Description

This function returns a NULL terminated list of OpenGL driver names to the application program, which describes all the available OpenGL drivers on the system. Once you have obtained the driver names, you can change the current OpenGL driver with *MGL_glSetDriver*.

Note that this function will always returns built in names for 'auto', 'microsoft', 'sgi' and 'mesa' and setting a driver to these names corresponds to changing the OpenGL driver type with *MGL_glSetOpenGLType*. However if there are multiple fullscreen hardware OpenGL drivers on the system and registered with SciTech MGL, they will be listed one after the other. For example if you have an ATI 3DRage, a 3Dfx Voodoo and an NEC PowerVR installed in your system and there are fullscreen OpenGL drivers for all these boards registered with SciTech MGL, the names for these drivers would be returned in this list also.

See Also

MGL_glSetDriver, *MGL_glSetOpenGLType*

MGL_glGetProcAddress

Returns the address of an OpenGL extension function.

Declaration

```
void * MGLAPI MGL_glGetProcAddress(  
    const char *procName)
```

Prototype In

mgraph.h

Parameters

dc

MGL device context to get the OpenGL extension function for

Return Value

Address of the specified extension function, NULL if not available.

Description

This function returns the address of an OpenGL extension function if that extension is supported by the OpenGL implementation. Each OpenGL implementation may export a number of OpenGL extension that may not be supported by other OpenGL implementations, and this function is the mechanism you can use to obtain the address of those extension functions.

Note that you should first check to see if an extension is available, but calling the OpenGL function `glGetString(GL_EXTENSIONS)` to get a list of all the available extensions. In order to check for a specific extension by name, you can use the following code:

```
bool checkExtension(const char *name)  
{  
    const char *p = (const char  
)glGetString(GL_EXTENSIONS);  
    while (p = strstr(p, name)) {  
        const char *q = p + strlen(name);  
        if (*q == ' ' || *q == '\\0')  
            return GL_TRUE;  
        p = q;  
    }  
    return GL_FALSE;  
}
```

Note: *It is an error to call this function for an MGL device context that does not have an OpenGL rendering context associated with it via a call to `MGL_glCreateContext`.*

MGL_glGetVisual

Returns the current OpenGL visual for the device context.

Declaration

```
void MGLAPI MGL_glGetVisual(  
    MGLDC *dc,  
    MGLVisual *visual)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Place to store the OpenGL visual information

Description

This function returns the current OpenGL visual that has been set for the MGL device context.

See Also

MGL_glSetVisual, *MGL_glCreateContext*

MGL_glHaveHWOpenGL

Checks for Microsoft OpenGL MCD or ICD hardware acceleration.

Declaration

```
bool MGLAPI MGL_glHaveHWOpenGL(void)
```

Prototype In

mgraph.h

Return Value

True if Microsoft OpenGL is hardware accelerated via an MCD or an ICD.

Description

This function will load the Microsoft OpenGL libraries and determine if there is support for hardware acceleration or not. This function is most useful for determining if OpenGL is hardware accelerated when running in windowed modes.

MGL_glMakeCurrent

Makes a MGL device context the current OpenGL rendering context.

Declaration

```
void MGLAPI MGL_glMakeCurrent(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to make the current OpenGL rendering context.

Description

This function makes the MGL device context the current rendering context for all OpenGL rendering commands. You must first call *MGL_glCreateContext* for the MGL device context to create a valid OpenGL rendering context before you can call this function to make it the rendering current OpenGL rendering context.

SciTech MGL and OpenGL allow you to create multiple rendering context, and to switch between them for output you must use this function to make one of them current at a time. You cannot have more than one OpenGL rendering context current at one time. For instance you may be drawing to a fullscreen OpenGL rendering context, but have an MGL memory device context that you wish to render a small 3D scene into, so you would make the memory device context the current OpenGL rendering context with a call to this function. The any subsequent OpenGL commands will draw to the memory device context instead of the display device context.

See Also

MGL_glCreateContext, *MGL_glDeleteContext*

MGL_glRealizePalette

Realizes the hardware palette for a device context when using OpenGL.

Declaration

```
void MGLAPI MGL_glRealizePalette(  
    MGLDC *dc,  
    int numColors,  
    int startIndex,  
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette for
<i>numColors</i>	Number of colors in the palette
<i>startIndex</i>	Starting index in the palette
<i>waitVRT</i>	True if routine should sync to vertical retrace, false if not.

Description

This function realizes the hardware palette associated with a display device context. Calls to *MGL_glSetPalette* only update the palette values in the color palette for the device context structure, but do not actually program the hardware palette for display device contexts in 4 and 8 bits per pixel modes. In order to program the hardware palette you must call this routine.

This function is identical to the regular *MGL_realizePalette* function, however if you are running OpenGL you must use this function instead.

See Also

MGL_glSetPalette, *MGL_realizePalette*

MGL_glResizeBuffers

Resizes the OpenGL buffers for the windowed device context.

Declaration

```
void MGLAPI MGL_glResizeBuffers(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc

MGL device context to resize the buffers for

Description

This function informs SciTech MGL that the dimensions of a windowed device context have changed, and that the OpenGL rendering buffers need to be re-sized to the new dimensions of the window. It is up to the application programmer to capture the WM_SIZE messages in windowed modes, and call this function when the window size changes to let SciTech MGL correctly update the buffer dimensions.

Note: *This function is not necessary in fullscreen modes.*

MGL_glSetDriver

Sets the currently active OpenGL driver

Declaration

```
bool MGLAPI MGL_glSetDriver(  
    const char *name)
```

Prototype In

mgraph.h

Return Value

True on success, false on failure.

Description

This function changes the currently active OpenGL driver to the name you pass in. Note that the name you pass in must be one of the names returned by the *MGL_glEnumerateDrivers*, or the function will fail.

See Also

MGL_glEnumerateDrivers, *MGL_glSetOpenGLType*

MGL_glSetOpenGLType

Sets the OpenGL implementation type to use.

Declaration

```
void MGLAPI MGL_glSetOpenGLType(  
    int type)
```

Prototype In

mgraph.h

Parameters

type New OpenGL type to set (*MGL_glOpenGLType*)

Description

This function sets the current OpenGL implementation type to use according to the passed in type parameter (*MGL_glOpenGLType* enumeration). In the AUTO mode we automatically determine which version of OpenGL to use depending on the target runtime system. For Win32 unless there is hardware acceleration available we choose Silicon Graphic's OpenGL, but if hardware acceleration is present we use the regular Microsoft OpenGL implementation so we can utilize the hardware. For DOS we currently use the Mesa implementation, but you can also force Mesa to be used for the Windows environment if you wish.

If there is a hardware OpenGL driver registered with SciTech MGL that supports SciTech MGL's fullscreen OpenGL extensions (such as the 3Dfx driver), and the hardware for that driver is present in the system this driver will be used automatically when the OpenGL type is set to MGL_GL_AUTO.

Note: *If you wish to be able to choose between the registered MGL hardware OpenGL drivers within your application, use the MGL_glEnumerateDrivers and MGL_glSetDriver functions instead.*

See Also

MGL_glSetDriver, MGL_glEnumerateDrivers

MGL_glSetPalette

Sets the palette values for a device context when using OpenGL.

Declaration

```
void MGLAPI MGL_glSetPalette(  
    MGLDC *dc,  
    palette_t *pal,  
    int numColors,  
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette for
<i>pal</i>	Palette to set
<i>numColors</i>	Number of colors in the palette
<i>startIndex</i>	Starting index in the palette

Description

This functions sets the color palette for an MGL device context when running in OpenGL rendering modes. This function is identical to the regular *MGL_setPalette* function, however if you are running OpenGL you must use this function instead.

See Also

MGL_glRealizePalette, *MGL_setPalette*

MGL_glSetVisual

Attempts to set the specified OpenGL visual for the MGL device context.

Declaration

```
bool MGLAPI MGL_glSetVisual(  
    MGLDC *dc,  
    MGLVisual *visual)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Structure containing OpenGL visual information

Return Value

True on success, false if visual not supported by OpenGL implementation.

Description

This function sets the passed in OpenGL visual for the MGL device context and makes it the visual that will be used in the call to *MGL_glCreateContext*. Note that this function may fail if the OpenGL visual requested is invalid, and you should call *MGL_glChooseVisual* first to find a visual that best matches the underlying OpenGL implementation. For instance if the OpenGL implementation only supports a 16-bit z-buffer, yet you request a 32-bit z-buffer this function will fail.

The OpenGL visual is used to define the visual capabilities of the OpenGL rendering context that will be created with the *MGL_glCreateContext* function, and includes information such as whether the mode should be an RGB mode or color index mode, whether it should be single buffered or double buffered, whether a depth buffer (zbuffer) should be used and how many bits it should be etc.

Note: *You can only set the visual for a context once, and it is an error to call MGL_glSetVisual more than once for an MGL device context, and you also cannot change a visual once you have set it without first destroying the OpenGL rendering context.*

See Also

MGL_glChooseVisual, MGL_glCreateContext

MGL_glSwapBuffers

Swaps the display buffers for OpenGL rendering.

Declaration

```
void MGLAPI MGL_glSwapBuffers(  
    MGLDC *dc,  
    int waitVRT)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	MGL device context to swap display buffers for
<i>waitVRT</i>	True to wait for vertical retrace

Description

This function swaps the OpenGL rendering buffers, and the current back buffer becomes the front buffer and vice versa. If you are running in a window, the context of the back buffer will be copied to the current window. If you are running in a fullscreen graphics mode with hardware page flipping, this function will do a hardware page flip to swap the buffers.

When the OpenGL buffers are swapped, you should normally allow MGL/OpenGL driver to sync to the vertical retrace to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display. You may however turn off the vertical retrace synching if you are synching up with the retrace period using some other means, or if you are measuring the performance of your application.

MGL_globalToLocal

Converts a point from global to local coordinates.

Declaration

```
void MGLAPI MGL_globalToLocal(  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context in which the point is defined
<i>p</i>	Pointer to point to be converted

Description

This function converts a coordinate from global coordinates to local coordinates. Global coordinates are defined relative to the entire output device context surface, 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_globalToLocalDC, *MGL_localToGlobal*

MGL_globalToLocalDC

Converts a point from global to local coordinates.

Declaration

```
void MGLAPI MGL_globalToLocalDC(  
    MGLDC *dc,  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context in which the point is defined
<i>p</i>	Pointer to point to be converted

Description

This function is the same as *MGL_globalToLocal*, however the device context does not have to be the current device context.

See Also

MGL_globalToLocal, *MGL_localToGlobal*

MGL_halfTonePixel

Compute color for pixel in halftone palette.

Declaration

```
uchar MGLAPI MGL_halfTonePixel(  
    int x,  
    int y,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	<i>x</i> pixel coordinate
<i>R</i>	Red component for pixel color
<i>G</i>	Green component for pixel color
<i>B</i>	Blue component for pixel color

Return Value

Color index for the pixel in MGL halftone palette

Description

This function computes the color index for a specified 24 bit RGB pixel value in the standard MGL halftone palette, and can be used to perform real-time dithering of pixel values from RGB colors to color index colors. This routine uses a fast table lookup and an 8x8 ordered dither to do the conversion.

See Also

MGL_getHalfTonePalette

MGL_halfTonePixel555

Compute color for an 555 format RGB pixel using a halftone dither

Declaration

```
ushort MGLAPI MGL_halfTonePixel555(  
    int x,  
    int y,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In
mgraph.h

Parameters

<i>x</i>	x coordinate of pixel (need only be relative)
<i>y</i>	y coordinate of pixel (need only be relative)
<i>R</i>	R value for pixel (8 bit components)
<i>G</i>	G value for pixel (8 bit components)
<i>B</i>	B value for pixel (8 bit components)

Return Value

Packed RGB 555 format pixel color

Description

This function computes the color of an RGB 555 format pixel using a fast halftone dither algorithm. The resulting color is pre-packed into the correct framebuffer format as part of the dithering process.

See Also

MGL_halfTonePixel565

MGL_halfTonePixel565

Compute color for an 565 format RGB pixel using a halftone dither

Declaration

```
ushort MGLAPI MGL_halfTonePixel565(  
    int x,  
    int y,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In
mgraph.h

Parameters

<i>x</i>	<i>x</i> coordinate of pixel (need only be relative)
<i>R</i>	R value for pixel (8 bit components)
<i>G</i>	G value for pixel (8 bit components)
<i>B</i>	B value for pixel (8 bit components)

Return Value

Packed RGB 565 format pixel color

Description

This function computes the color of an RGB 565 format pixel using a fast halftone dither algorithm. The resulting color is pre-packed into the correct framebuffer format as part of the dithering process.

See Also

MGL_halfTonePixel555

MGL_haveWidePalette

Tests whether the palette for a device context uses an 8-bit wide palette.

Declaration

```
bool MGLAPI MGL_haveWidePalette(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest.

Return Value

True if palette is 8 bits wide, false if 6 bits wide.

Description

This function will return true if the display device context is using an 8-bit per primary hardware palette, or false if it is using a 6-bit per primary hardware palette. The original VGA standard only supports 6-bits per primary for the color palette, giving you the selection of 256 colors out of 256,000 for 8bpp modes. However more modern controllers provide support for 8-bits per primary, giving you a selection of 256 colors out of a total 16.7 million for 8bpp modes.

Initializes MGL for fullscreen operation.

Declaration

```
bool MGLAPI MGL_init(  
    int *pDriver,  
    int *pMode,  
    const char *mglpath)
```

Prototype In

mgldos.h, mglwin.h

Parameters

pDriver

Place to store detected graphics device driver id

mglpath

Path to standard MGL resource files

Return Value

True if successful, false on error.

Description

This function initializes MGL for fullscreen operation, and sets it up to use the specified fullscreen device driver and video mode. If you pass the value *grDETECT* in the driver parameter, this function will automatically call the *MGL_detectGraph* routine to detect the installed video hardware, and initialize itself accordingly (please refer to *MGL_detectGraph* for more information on the detection process). The video mode used in this case will be the one suggested by the installed video device driver, and will be returned in the mode parameter.

Before you can call this routine, you must ensure that you have registered the appropriate device drivers that you are interested in using in your code with the *MGL_registerDriver* function. The process of registering the device drivers ensure that the code will be linked in when you link your application (by default the code will not be linked, to save space in the resulting executable). You can simply call *MGL_registerAllDispDrivers*. 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 first, and pass the value returned for driver and your selected video mode in mode to initialize the specific mode. The fullscreen device drivers currently supported by MGL are enumerated in *MGL_driverType*.

The `mgldpath` variable is used by MGL to locate all MGL resources files in their standard locations. The value passed in here should point to the base directory where all the standard MGL resources are located, which may simply be the current directory (i.e. a value of `“.”`). When MGL is searching for resource files (bitmaps, icons, fonts and cursors) it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap or font file, you should pass the full pathname to the file that you want. If the filename is a simple relative filename (i.e. `“MYFONT.FNT”`), MGL will then search in the standard directories relative to the path specified in `mgldpath`. As a final resort MGL will also look for the files relative to the `MGL_ROOT` environment variable, which can be set to point to a standard location where all MGL files will be stored on the user's machine. The standard locations that MGL will look for the resource files are as follows:

<i>Resource</i>	Base pathname
<i>Bitmaps</i>	<code>mgldpath\BITMAPS</code>
<i>Fonts</i>	<code>mgldpath\FONTS</code>
<i>Icons</i>	<code>mgldpath\ICONS</code>
<i>Cursors</i>	<code>mgldpath\CURSORS</code>

If anything went wrong during the initialization process, 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_initWindowed, MGL_detectGraph, MGL_result

MGL_initWindowed

Initializes MGL for windowed only output.

Declaration

```
bool MGLAPI MGL_initWindowed(  
    const char *mglpath)
```

Prototype In

mglwin.h

Parameters

mglpath Path to standard MGL resource files

Return Value

True if successful, false on error.

Description

This function initializes MGL for windowed only operation. The standard *MGL_init* function sets up MGL for fullscreen output mode, which is not available under Windows NT. Also if you are only interested in using MGL in a real operating system window rather than in fullscreen mode, then using the init function will not load any of the fullscreen graphics drivers and DLL's.

The *mglpath* variable is used by MGL to locate all MGL resources files in their standard locations. The value passed in here should point to the base directory where all the standard MGL resources are located, which may simply be the current directory (i.e. a value of "."). When MGL is searching for resource files (bitmaps, icons, fonts and cursors) it will first attempt to find the files just be using the filename itself. Hence if you wish to look for a specific bitmap or font file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFONT.FNT"), MGL will then search in the standard directories relative to the path specified in *mglpath*. As a final resort MGL will also look for the files relative to the MGL_ROOT environment variable, which can be set to point to a standard location where all MGL files will be stored on the user's machine. The standard locations that MGL will look for the resource files are as follows:

Type	Base pathname
Bitmaps	mglpath\BITMAPS
Fonts	mglpath\FONTS
Icons	mglpath\ICONS
Cursor	mglpath\CURSORS

If anything went wrong during the initialization process, 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_init, *MGL_result*

MGL_insetRect

Insets the coordinates of a rectangle.

Declaration

```
void MGL_insetRect(  
    rect_t r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>r</i>	Rectangle to inset
<i>dx</i>	Amount to inset the x coordinates by
<i>dy</i>	Amount to inset the y coordinates by

Description

This functions insets the rectangle by the specified values. This function effectively performs the following operation on the rectangle:

```
left += dx;  
top += dy;  
right -= dx;  
bottom -= dy;
```

See Also

MGL_offsetRect

MGL_isCurrentDC

Determines if the specified device context is the currently active context.

Declaration

```
bool MGLAPI MGL_isCurrentDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if DC is the currently active context, false otherwise.

Description

This function determines if the passed in device context is the currently active device context. The currently active device context is where all the output from all the MGL rendering functions will be displayed.

See Also

MGL_makeCurrentDC

MGL_isDisplayDC

Determines if the specified device context is a display device context.

Declaration

```
bool MGLAPI MGL_isDisplayDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a display DC, false if not.

Description

This function determines if the passed in device context is a fullscreen display device context, or some other type of device context.

See Also

MGL_isMemoryDC, *MGL_isWindowedDC*

MGL_isMemoryDC

Determines if the specified device context is a memory device context.

Declaration

```
bool MGLAPI MGL_isMemoryDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a memory DC, false if not.

Description

This function determines if the passed in device context is a memory device context, or some other type of device context.

See Also

MGL_isDisplayDC, *MGL_isWindowedDC*

MGL_isSimpleRegion

Returns true if a region is a simple region, otherwise false.

Declaration

```
bool MGL_isSimpleRegion(  
    region_t r)
```

Prototype In

mgraph.h

Parameters

r Region to test.

Description

This function determines if the region is simple or not. A simple region is one that consists of only a single rectangle. This function will not work properly if the region has been through a number of region algebra routines with other non-simple regions, even though the end result may be a single rectangle.

See Also

MGL_unionRegion, *MGL_diffRegion*, *MGL_sectRegion*

MGL_isWindowedDC

Determines if the specified device context is a display device context.

Declaration

```
bool MGLAPI MGL_isWindowedDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a windowed DC, false if not.

Description

This function determines if the passed in device context is a windowed device context, or some other type of device context.

See Also

MGL_isDisplayDC, *MGL_isMemoryDC*

MGL_leftTop

Returns the left top point of a rectangle.

Declaration

```
point_t MGL_leftTop(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to return left top corner coordinate for.

Return Value

Returns the top left coordinate of the rectangle.

See Also

MGL_rightBottom

MGL_line

Draws a line with integer coordinates.

Declaration

```
void MGL_line(  
    point_t p1,  
    point_t p2)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint

Description

This function is the same as *MGL_lineCoord*, however it takes the coordinates of the line as two points.

See Also

MGL_lineCoord

Draws a line with integer coordinates.

Declaration

```
void MGLAPI MGL_lineCoord(  
    int x1,  
    int y1,  
    int x2,  
    int y2)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint

Description

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 is on. Note that this function takes the coordinates of the lines in integer format. MGL draws all lines internally in 16.16 fixed point format, so this routine always converts the coordinates and then calls the *MGL_lineFX* routines. If you need maximum performance, you should call the fixed point line drawing functions instead.

See Also

MGL_line

MGL_lineCoordFX

Draws a line with fixed point coordinates.

Declaration

```
void MGLAPI MGL_lineCoordFX(  
    fix32_t x1,  
    fix32_t y1,  
    fix32_t x2,  
    fix32_t y2)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint

Description

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 is on. Note that the coordinates are passed in 16.16 fixed point format, which provides for maximum precision when the lines are drawn.

See Also

MGL_lineFX

Generates the set of integer points on a line, given fixed point coordinates.

Declaration

```
void MGLAPI MGL_lineEngine(  
    fix32_t x1,  
    fix32_t y1,  
    fix32_t x2,  
    fix32_t y2,  
    void (ASMAPI *plotPoint)(  
        int x,  
        int y))
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint
<i>plotPoint</i>	User supplied pixel plotting routine

Description

This function generates the set of points on a line, and calls a user supplied plotPoint routine for every point generated. The set of points generated will always be in the same order for any two endpoints, no matter which order the endpoints are given.

Note: *Lines must always be passed in with the X1 value less than the X2 value!*

Note: *This routine expects the endpoints to be in 32 bit fixed point 16.16 format, and can scan convert lines with non-integer endpoints.*

See Also

MGL_ellipseEngine, MGL_ellipseArcEngine

MGL_lineFX

Draws a line with fixed point coordinates.

Declaration

```
void MGL_lineFX(  
    fxpoint_t p1,  
    fxpoint_t p2)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint of line
<i>p2</i>	Second endpoint of line

Description

This function is the same as *MGL_lineCoordFX*, however it takes the coordinates of the line as two points.

See Also

MGL_lineCoordFX

MGL_lineRel

Draws a relative line.

Declaration

```
void MGL_lineRel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Amount to offset in (x,y) coordinates

Description

This function is the same as *MGL_lineRelCoord*, however the amount to move the CP by is passed as a point.

See Also

MGL_lineRelCoord, *MGL_moveRelCoord*

MGL_lineRelCoord

Draws a relative line.

Declaration

```
void MGLAPI MGL_lineRelCoord(  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>dx</i>	Amount to offset in x coordinate
<i>dy</i>	Amount to offset in y coordinate

Description

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_lineRel, *MGL_moveRelCoord*

MGL_lineTo

Draws a line from the CP to the specified point.

Declaration

```
void MGL_lineTo(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to draw to

Description

This function is the same as *MGL_lineToCoord*, however the point to draw to is passed as a point.

See Also

MGL_lineTo

MGL_lineToCoord

Draws a line from the CP to the specified point.

Declaration

```
void MGLAPI MGL_lineToCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to draw to
<i>y</i>	y coordinate to draw to
<i>p</i>	Point to draw to

Description

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_lineTo

MGL_loadBitmap

Load a lightweight bitmap file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadBitmap(  
    const char *bitmapName,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>bitmapName</i>	Name of bitmap file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded bitmap file, NULL on error.

Description

Locates the specified bitmap file and loads it into a lightweight bitmap structure. MGL can load any Windows 3.x style bitmap files, including new format bitmap files with colors depths of 15/16 and 32 bits per pixel. Consult the Windows SDK documentation for the format of Windows bitmap files.

If loadPalette is true, the palette values for the bitmap will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the bitmap will be NULL. For small bitmaps you can save space by not loading the palette for the bitmap.

When MGL is searching for bitmap files it will first attempt to find the files just be using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.BMP"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the bitmap, which loads the bitmap with the minimum memory overheads. You can draw the bitmap on any device context surface using the *MGL_putBitmap* function, but you don't have the full flexibility of using a full memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap surface and load the bitmap with the *MGL_loadBitmapIntoDC* function.

See Also

MGL_unloadBitmap, *MGL_availableBitmap*, *MGL_getBitmapSize*,
MGL_loadBitmapIntoDC, *MGL_saveBitmapFromDC*, *MGL_putBitmap*,
MGL_loadBitmapExt

MGL_loadBitmapExt

Locates an open bitmap and loads it into memory from an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadBitmapExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read data from
<i>dwOffset</i>	Offset to start of bitmap in file
<i>dwSize</i>	Size of the file
<i>loadPalette</i>	Should we load the palette values as well?

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadBitmap*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadBitmap, *MGL_loadBitmapIntoDC*

MGL_loadBitmapIntoDC

Loads a bitmap file directly into an existing device context.

Declaration

```
bool MGLAPI MGL_loadBitmapIntoDC(  
    MGLDC *dc,  
    const char *bitmapName,  
    int dstLeft,  
    int dstTop,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>bitmapName</i>	Name of bitmap file to load
<i>dstLeft</i>	Left coordinate to load bitmap at
<i>dstTop</i>	Top coordinate to load bitmap at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

True if the bitmap was loaded, false on error.

Description

Locates the specified bitmap file and loads it into the specified device context at the specified destination coordinates. If the bitmap is of a different pixel depth then the device context that it is being loaded into, the bitmap will be converted as it is loaded to the pixel format of the device context it is being loaded into. MGL can load any Windows 3.x style bitmap files, including new format bitmap files with colors depths of 15/16 and 32 bits per pixel. Consult the Windows SDK documentation for the format of Windows bitmap files.

If loadPalette is true, the palette values for the bitmap will be loaded and stored in the device context's palette. If the device context being loaded into is the currently active display device, the palette will also be realized before the bits in the bitmap are loaded.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific

bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. “MYBMP.BMP”), MGL will then search in the BITMAPS directory relative to the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the *MGL_ROOT* environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_availableBitmap, *MGL_getBitmapSize*, *MGL_loadBitmap*,
MGL_saveBitmapFromDC, *MGL_loadBitmapIntoDCExt*

MGL_loadBitmapIntoDCExt

Locates the specified bitmap file and loads it into a device context.

Declaration

```
bool MGLAPI MGL_loadBitmapIntoDCExt(  
    MGLDC *dc,  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int dstLeft,  
    int dstTop,  
    bool loadPalette)
```

Prototype In mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Pointer to opened bitmap file
<i>dwSize</i>	Size of the bitmap file
<i>dwOffset</i>	Offset into the file
<i>dstLeft</i>	Left coordinate to place bitmap at
<i>dstTop</i>	Top coordinate to place bitmap at
<i>loadPalette</i>	Should we load the palette values as well?

Return Value

True if the bitmap was successfully loaded.

Description

This function is the same as *MGL_loadBitmapIntoDC*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadBitmapIntoDC

MGL_loadCursor

Load a cursor file from disk.

Declaration

```
cursor_t * MGLAPI MGL_loadCursor(  
    const char *cursorName)
```

Prototype In

mgraph.h

Parameters

cursorName Name of cursor file to load

Return Value

Pointer to loaded cursor file, NULL on error.

Description

Locates the specified cursor file and loads it into memory. MGL can load any Windows 3.x style cursor files. Consult the Windows SDK documentation for the format of Windows cursor files.

When MGL is searching for cursor files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific cursor file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYCURS.CUR"), MGL will then search in the CURSORS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the CURSORS directory relative to the MGL_ROOT environment variable.

If the cursor file was not found, or an error occurred while reading the cursor file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadCursor, *MGL_availableCursor*, *MS_setCursor*, *MGL_loadCursorExt*

MGL_loadCursorExt

Load a cursor file from disk from an opened file.

Declaration

```
cursor_t * MGLAPI MGL_loadCursorExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open file to read cursor from (binary mode)
<i>dwOffset</i>	Offset to the start of the cursor file
<i>dwSize</i>	Size of the file

Return Value

Pointer to the loaded cursor file

Description

This function is the same as *MGL_loadCursor*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadCursor

MGL_loadFont

Load a font file for use.

Declaration

```
font_t * MGLAPI MGL_loadFont(  
    const char *fontname)
```

Prototype In

mgraph.h

Parameters

fontname Name of the font file to load

Return Value

Pointer to the loaded font file, NULL on error.

Description

Locates the specified font file and loads it into memory. MGL can load any Windows 2.x style font files (Windows 3.x font files are not supported, but Windows 2.x font files are the standard files even for Windows 3.1. Most resource editors can only output 2.x style font files). Consult the Windows SDK documentation for the format of Windows font files.

When MGL is searching for font files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific font file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFONT.FNT"), MGL will then search in the FONTS directory relative to the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the FONTS directory relative to the MGL_ROOT environment variable.

If the font file was not found, or an error occurred while reading the font file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadFont, *MGL_useFont*, *MGL_availableFont*.

MGL_loadFontExt

Load a font file for use from an opened file.

Declaration

```
font_t * MGLAPI MGL_loadFontExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read font data from
<i>dwOffset</i>	Offset to start of font in file
<i>dwSize</i>	Size of the file in bytes

Return Value

Pointer to the font data, or NULL on error.

Description

This function is the same as *MGL_loadFont*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadFont

Load an icon file from disk.

Declaration

```
icon_t * MGLAPI MGL_loadIcon(  
    const char *iconName,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>iconName</i>	Name of icon file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded icon file, NULL on error.

Description

Locates the specified icon file and loads it into memory. MGL can load any Windows 3.x style icon files of any dimensions. Generally icon files will be either 32x32 or 64x64 in size. Consult the Windows SDK documentation for the format of Windows font files.

If loadPalette is true, the palette values for the icon will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the bitmap will be NULL.

When MGL is searching for icon files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific icon file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYICON.ICO"), MGL will then search in the ICONS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the ICONS directory relative to the MGL_ROOT environment variable.

If the icon file was not found, or an error occurred while reading the icon file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadIcon, MGL_availableIcon, MGL_putIcon, MGL_loadIconExt

MGL_loadIconExt

Load an icon file from disk from an open file.

Declaration

```
icon_t * MGLAPI MGL_loadIconExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Pointer to open icon file
<i>dwOffset</i>	Offset into the icon file
<i>dwSize</i>	Size of the icon file
<i>loadPalette</i>	If true, the palette is loaded as well

Return Value

Pointer to the loaded icon file.

Description

This function is the same as *MGL_loadIcon*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadIcon

MGL_loadJPEG

Load a JPEG bitmap file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadJPEG(  
    const char *JPEGName,  
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>JPEGName</i>	Name of JPEG file to load
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded JPEG file, NULL on error.

Description

Locates the specified JPEG file and loads it into a lightweight bitmap structure. Because JPEG files are inherently 24-bit, when you load a JPEG file with this function it will always be decoded as a 24-bit RGB bitmap file, unless you set num8BitColors parameter. If the num8BitColors parameter is set to a value other than 0, it causes the JPEG decoder to quantize down to an 8 bits per pixel bitmap with an optimized floyd-steinberg dither (better than SciTech MGL's simple halftone dithering, but the bitmap will contain a custom palette). The number of significant colors in the output image will be set to the value you specify, so you can use this to quantise down to a color table smaller than 8-bits per pixel. To decode as a 24-bit image, simply set this field to 0.

Note that if you set num8BitColors to -1, the JPEG decoder will decode the image as a grayscale bitmap which is faster than decoding the full color image (useful for preview operations etc). Note that images that are decoded as grayscale are 8-bits per pixel with a grayscale color map and should display as true grayscale images in all color depths.

If you wish to load the bitmap as a different color depth or pixel format use the *MGL_loadJPEGIntoDC* function. SciTech MGL will however properly

decode grayscale JPEG files, but they will be loaded as a 24-bit bitmap since SciTech MGL does not natively support grayscale bitmaps.

When MGL is searching for JPEG files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific JPEG file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.JPG"), MGL will then search in the BITMAPS directory relative to the path specified in `mgldpath` variable that was passed to `MGL_init`. As a final resort MGL will also look for the files in the BITMAPS directory relative to the `MGL_ROOT` environment variable.

If the JPEG file was not found, or an error occurred while reading the JPEG file, this function will return NULL. You can check the `MGL_result` error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the JPEG file, which loads the bitmap with the minimum memory overheads. You can draw the JPEG file on any device context surface using the `MGL_putBitmap` function, but you don't have the full flexibility of using a memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap data and load the bitmap with the `MGL_loadJPEGIntoDC` function.

Note: *In order to use this function you must link with the separate `JPEG.LIB` library for your compiler.*

See Also

`MGL_unloadBitmap`, `MGL_availableJPEG`, `MGL_getJPEGSize`,
`MGL_loadJPEGIntoDC`, `MGL_saveJPEGFromDC`, `MGL_putBitmap`,
`MGL_loadJPEGExt`

MGL_loadJPEGExt

Load a JPEG bitmap file from disk using an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadJPEGExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file within open file
<i>dwSize</i>	Size of JPEG file in bytes
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadJPEG*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_loadJPEG, *MGL_loadBitmap*

MGL_loadJPEGIntoDC

Loads a JPEG file directly into an existing device context.

Declaration

```
bool MGLAPI MGL_loadJPEGIntoDC(  
    MGLDC *dc,  
    const char *JPEGName,  
    int dstLeft,  
    int dstTop,  
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>JPEGName</i>	Name of JPEG file to load
<i>dstLeft</i>	Left coordinate to load JPEG at
<i>dstTop</i>	Top coordinate to load JPEG at
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

True if the JPEG file was loaded, false on error.

Description

Locates the specified JPEG file and loads it into the specified device context at the specified destination coordinates. If the JPEG is of a different pixel depth than the device context that it is being loaded into, the JPEG will be converted as it is loaded to the pixel format of the device context it is being loaded into.

If the num8BitColors parameter is set to a value other than 0, it causes the JPEG decoder to quantize down to an 8 bits per pixel bitmap with an optimized floyd-steinberg dither (better than SciTech MGL's simple halftone dithering) using the palette of the destination device context as the source for the dither. The number of significant colors used from the destination device contexts palette for the dither is set by the num8BitColors parameter. If however the destination device context is not 8-bits per pixel, the image is simply decoded as a 24-bit bitmap.

Note that if you set num8BitColors to -1, the JPEG decoder will decode the image as a grayscale bitmap which is faster than decoding the full color image (useful for preview operations etc). Note that images that are decoded as grayscale are 8-bits per pixel with a grayscale color map and should display as true grayscale images in all color depths. If the destination device context is 8-bits per pixel, the color palette will be changed to a grayscale color palette.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFILE.JPEG"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_availableJPEG, MGL_getJPEGSize, MGL_loadJPEG, MGL_saveJPEGFromDC

MGL_loadJPEGIntoDCExt

Loads a JPEG file directly into an existing device context using an open file.

Declaration

```
bool MGLAPI MGL_loadJPEGIntoDCExt(  
    MGLDC *dc,  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int dstLeft,  
    int dstTop,  
    int num8BitColors)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file within open file
<i>dwSize</i>	Size of JPEG file in bytes
<i>dstLeft</i>	Left coordinate to load JPEG at
<i>dstTop</i>	Top coordinate to load JPEG at
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded JPEG file, NULL on error.

Description

This function is the same as *MGL_loadJPEGIntoDC*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_loadJPEGIntoDC, *MGL_loadBitmapIntoDC*

Load a lightweight PCX file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadPCX(  
    const char *PCXName,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>PCXName</i>	Name of PCX file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded PCX file, NULL on error.

Description

Locates the specified PCX file and loads it into a lightweight bitmap structure. MGL can load any 1 or 8 bits per pixel PCX file (currently 4 and 24 bits per pixel formats are not supported).

If loadPalette is true, the palette values for the PCX will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the PCX will be NULL. For small PCX files you can save space by not loading the palette for the PCX.

When MGL is searching for PCX files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific PCX file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.PCX"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the PCX file was not found, or an error occurred while reading the PCX file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the PCX file, which loads the bitmap with the minimum memory overheads. You can draw the PCX file on any device context surface using the *MGL_putBitmap* function, but you don't have the full flexibility of using a memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap data and load the bitmap with the *MGL_loadPCXIntoDC* function.

See Also

MGL_unloadBitmap, *MGL_availablePCX*, *MGL_getPCXSize*,
MGL_loadPCXIntoDC, *MGL_savePCXFromDC*, *MGL_putBitmap*,
MGL_loadPCXExt

MGL_loadPCXExt

Load a lightweight PCX file from disk using an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadPCXExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file within open file
<i>dwSize</i>	Size of PCX file in bytes
<i>loadPalette</i>	If true, palette values are loaded as well.

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadPCX*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadPCX, *MGL_loadBitmap*

MGL_loadPCXIntoDC

Loads a PCX file directly into an existing device context.

Declaration

```
bool MGLAPI MGL_loadPCXIntoDC(  
    MGLDC *dc,  
    const char *PCXName,  
    int dstLeft,  
    int dstTop,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>PCXName</i>	Name of PCX file to load
<i>dstLeft</i>	Left coordinate to load PCX at
<i>dstTop</i>	Top coordinate to load PCX at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

True if the PCX file was loaded, false on error.

Description

Locates the specified PCX file and loads it into the specified device context at the specified destination coordinates. If the PCX is of a different pixel depth than the device context that it is being loaded into, the PCX will be converted as it is loaded to the pixel format of the device context it is being loaded into. MGL can load any 1 or 8 bits per pixel PCX file (currently 4 and 24 bits per pixel formats are not supported).

If loadPalette is true, the palette values for the PCX will be loaded and stored in the device context's palette. If the device context being loaded into is the currently active display device, the palette will also be realized before the bits in the bitmap are loaded.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e.

“MYFILE.PCX”), MGL will then search in the BITMAPS directory relative to

the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the *MGL_ROOT* environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_availablePCX, *MGL_getPCXSize*, *MGL_loadPCX*, *MGL_savePCXFromDC*

MGL_loadPCXIntoDCExt

Load a lightweight PCX file from disk using an open file.

Declaration

```
bool MGLAPI MGL_loadPCXIntoDCExt(  
    MGLDC *dc,  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize,  
    int dstLeft,  
    int dstTop,  
    bool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file within open file
<i>dwSize</i>	Size of PCX file in bytes
<i>dstLeft</i>	Left coordinate to load PCX at
<i>dstTop</i>	Top coordinate to load PCX at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded PCX file, NULL on error.

Description

This function is the same as *MGL_loadPCXIntoDC*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadPCXIntoDC, *MGL_loadBitmapIntoDC*

MGL_localToGlobal

Converts a point from local to global coordinates.

Declaration

```
void MGLAPI MGL_localToGlobal(  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

p Pointer to point to be converted

Description

This function converts a coordinate from local coordinates to global coordinates. Global coordinates are defined relative to the entire output device display, while local coordinates are relative to the currently active viewport.

See Also

MGL_localToGlobalDC, MGL_globalToLocal

MGL_localToGlobalDC

Converts a point from local to global coordinates for a specific DC.

Declaration

```
void MGLAPI MGL_localToGlobalDC(  
    MGLDC *dc,  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context in which the point is defined
<i>p</i>	Pointer to point to be converted

Description

This function is the same as *MGL_localToGlobal*, however the device context does not have to be the current device context.

See Also

MGL_localToGlobal, *MGL_globalToLocal*

MGL_makeCurrentDC

Make a device context the current device context.

Declaration

```
MGLDC * MGLAPI MGL_makeCurrentDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc

New device context to make the current context

Return Value

Previous current device context (possibly NULL).

Description

This function makes the specified device context the current device context. The current device context is the one that is used for all the output rasterizing routines, and when a device context is made current, a copy of the device context is cached internally for maximum speed in the rasterizing code. While a device context is current, changes made to the global state of the current device context are changed in the cached copy only and are not updated in the original device context. When the device context is changed however, the values in the cached device context are flushed back to the original device context to keep it up to date. You can flush the current device context explicitly by passing a NULL for the new current device context.

Because of this caching mechanism, changing the current device context is an expensive operation, so you should try to minimize the need to change the current device context. Normally you will maintain a single device context that will be used for all rasterizing operations, and leave this as your current device context. If the device context specified is already the current device context, this function simply does nothing.

See Also

MGL_isCurrentDC

MGL_makeSubDC

Restricts the output from the display device context to a specified output region.

Declaration

```
void MGLAPI MGL_makeSubDC(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to convert
<i>left</i>	Minimum left coordinate for all subsequent output
<i>top</i>	Minimum top coordinate for all subsequent output
<i>right</i>	Maximum right coordinate for all subsequent output
<i>bottom</i>	Maximum bottom coordinate for all subsequent output

Description

This is similar to setting a viewport for the device, however this routine fully restricts all output to this specified rectangle, effectively making a DC that is a subsize of the original DC size. Once this call is made, the size of the zbuffer and system memory buffer need for double buffering with this DC are reduced to the specified size. Normally the dimensions for the entire display device would be used.

Allocate a block of memory.

Declaration

```
void * MGLAPI MGL_malloc(  
    long size)
```

Prototype In

mgraph.h

Parameters

size Size of block to allocate in bytes

Return Value

Pointer to allocated block, or NULL if out of memory.

Description

Allocates a block of memory of length *size*. Note that unlike the standard C malloc function, this routine will properly handle allocations of blocks of memory larger than 64Kb in 16 bit real mode environments.

If you have changed the memory allocation routines with the *MGL_useLocalMalloc* function, then calls to this function will actually make calls to the local memory allocation routines that you have registered.

See Also

MGL_calloc, *MGL_free*, *MGL_memset*, *MGL_useLocalMalloc*

MGL_mapToPalette

Map the colors of a memory device context to match a new palette.

Declaration

```
void MGLAPI MGL_mapToPalette(  
    MGLDC *dc,  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

dc
pal

Memory device context to map (8 bits per pixel only)
New palette to map to

Description

This function maps the pixels of an 8 bits per pixel memory device context to the specified palette, and then sets the palette for the device context to the new palette. This function actually translates every pixel in the device context's surface to the new palette, by looking for the entry in the new palette that is the closest to color of the original pixel in the old palette (the one currently active before this routine was called).

See Also

MGL_setPalette

MGL_marker

Draws a marker at the specified coordinate.

Declaration

```
void MGLAPI MGL_marker(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Coordinate to draw the marker at

Description

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_markerStyleType* type for an enumeration of the types of markers supported.

See Also

MGL_setMarkerSize, *MGL_getMarkerSize*, *MGL_setMarkerStyle*,
MGL_getMarkerStyle, *MGL_polyMarker*

MGL_maxCharWidth

Returns the maximum character width for current font.

Declaration

```
int MGLAPI MGL_maxCharWidth(void)
```

Prototype In

mgraph.h

Return Value

Maximum character width for current font.

Description

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 device surface.

See Also

MGL_getCharMetrics, *MGL_getFontMetrics*, *MGL_textHeight*, *MGL_textWidth*, *MGL_charWidth*

MGL_maxColor

Returns the maximum available color value.

Declaration

```
color_t MGLAPI MGL_maxColor(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

Maximum color value for current video mode.

Description

Returns the value of the largest available color value for the current video mode. This value will always be one less than the number of available colors in that particular video mode.

MGL_maxPage

Returns the maximum available hardware video page index.

Declaration

```
int MGLAPI MGL_maxPage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check.

Return Value

Index of maximum available hardware video page.

Description

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.

MGL_maxx

Returns the current maximum x coordinate.

Declaration

```
int MGLAPI MGL_maxx(void)
```

Prototype In

mgraph.h

Return Value

Maximum x coordinate in current viewport.

Description

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 application.

See Also

MGL_maxxDC, *MGL_maxy*, *MGL_sizex*, *MGL_sizey*

MGL_maxxDC

Returns the current maximum x coordinate for a specific DC.

Declaration

```
int MGLAPI MGL_maxxDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of the target x coordinate

Return Value

Maximum x coordinate in current viewport.

Description

This function is the same as *MGL_maxx*, however the device context does not have to be the current device context.

See Also

MGL_maxx, *MGL_maxy*, *MGL_sizex*, *MGL_sizey*

MGL_maxy

Returns the current maximum y coordinate.

Declaration

```
int MGLAPI MGL_maxy(void)
```

Prototype In

mgraph.h

Return Value

Maximum y coordinate in current viewport.

Description

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.

See Also

MGL_maxyDC, *MGL_maxx*, *MGL_sizex*, *MGL_sizey*

MGL_maxyDC

Returns the current maximum y coordinate for a specific DC.

Declaration

```
int MGLAPI MGL_maxyDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of the target y coordinate

Return Value

Maximum y coordinate in current viewport.

Description

This function is the same as *MGL_maxy*, however the device context does not have to be the current device context.

See Also

MGL_maxy, *MGL_maxx*, *MGL_sizex*, *MGL_sizey*

MGL_memcpy

Copies a block of memory as fast as possible.

Declaration

```
void      ASMAPI MGL_memcpy(void *dst,void *src,int n)
void MGLAPI MGL_memcpy(
    void *dst,
    void *src,
    int n)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library memcpy function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

See Also

MGL_memcpyVIRT SRC, *MGL_memcpyVIRT DST*

MGL_memcpyVIRTDEST

Copies a block of memory as fast as possible.

Declaration

```
void WINAPI MGL_memcpyVIRTDEST(  
    void *dst,  
    void *src,  
    int n)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library memcpy function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

This function is identical to *MGL_memcpy* except that it is virtual linear framebuffer safe, and should be used for copying data where the destination pointer resides in a virtualized linear surface.

See Also

MGL_memcpyVIRTSRC, *MGL_memcpy*

MGL_memcpyVIRTsrc

Copies a block of memory as fast as possible.

Declaration

```
void WINAPI MGL_memcpyVIRTsrc(  
    void *dst,  
    void *src,  
    int n)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library memcpy function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

This function is identical to *MGL_memcpy* except that it is virtual linear framebuffer safe, and should be used for copying data where the source pointer resides in a virtualized linear surface.

See Also

MGL_memcpy, *MGL_memcpyVIRTdst*

MGL_memset

Clears a memory block with 8-bit values.

Declaration

```
void MGLAPI MGL_memset(  
    void _HUGE *p,  
    int c,  
    long n)
```

Prototype In
mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of bytes to clear

Description

This function clears a memory block to the specified 8 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments.

See Also

MGL_memsetw, *MGL_memsetl*

MGL_memsetl

Clears a memory block with 32-bit values.

Declaration

```
void MGLAPI MGL_memsetl(  
    void _HUGE *p,  
    long c,  
    long n)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of dwords to clear

Description

This function clears a memory block to the specified 32 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments, and allows you to specify a specific 32 bit value to be cleared.

See Also

MGL_memset, *MGL_memsetw*

MGL_memsetw

Clears a memory block with 16-bit values.

Declaration

```
void MGLAPI MGL_memsetw(  
    void _HUGE *p,  
    int c,  
    long n)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of words to clear

Description

This function clears a memory block to the specified 16 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments, and allows you to specify a specific 16 bit value to be cleared.

See Also

MGL_memset, *MGL_memsetl*

MGL_mirrorGlyph

Computes mirror image of a glyph (monochrome bitmap).

Declaration

```
void MGLAPI MGL_mirrorGlyph(  
    uchar *dst,  
    uchar *src,  
    int byteWidth,  
    int height)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination glyph buffer
<i>src</i>	Source glyph buffer
<i>byteWidth</i>	Width of the glyph in bytes
<i>height</i>	Height of the glyph in scanlines

Description

This function computes the mirror image glyph of the source glyph, and stores the value in the destination buffer. The source buffer is not modified, and the mirror image glyph will be no larger than the original glyph.

See Also

MGL_rotateGlyph, *MGL_drawGlyph*

MGL_modeDriverName

Get a the name of the device driver used for particular graphics mode.

Declaration

```
const char * MGLAPI MGL_modeDriverName(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode graphics mode number

Return Value

Pointer to the name of device driver serving this mode

Description

This function returns the name of the device driver that is currently being used to support the specified graphics mode. MGL provides a number of device drivers for supporting the fullscreen graphics mode resolutions depending on the capabilities of the underlying hardware. This function allows you to determine which driver is currently being used to support each mode.

See Also

MGL_driverName, *MGL_modeName*

MGL_modeFlags

Returns the mode flags for the specific graphics mode.

Declaration

```
ulong MGLAPI MGL_modeFlags(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode graphics mode number to get flags for

Return Value

Flags for the specific graphics mode, or 0 for invalid mode.

Description

This function returns mode flags for the specified mode. The mode flags are available after calling *MGL_detectGraph*, and provides information about the hardware capabilities of the graphics mode, such as whether it supports 2D or 3D acceleration, video acceleration, refresh rate control and hardware stereo support. You can use these flags to make choices about the graphics mode to use for your application prior to initialising a specific graphics mode.

Specific mode flags are enumerated in *MGL_modeFlagsType*.

See Also

MGL_modeName, *MGL_driverName*, *MGL_modeResolution*

MGL_modeName

Returns the name of the specified graphics mode.

Declaration

```
const char * MGLAPI MGL_modeName(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode graphics mode number to get name for

Return Value

Pointer to a string representing the name of the mode.

See Also

MGL_driverName

MGL_modeResolution

Returns the resolution and pixel depth of a specific graphics mode.

Declaration

```
int MGLAPI MGL_modeResolution(  
    int mode,  
    int *xRes,  
    int *yRes,  
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>mode</i>	graphics mode number to get resolution for
<i>xRes</i>	Place to store the x resolution
<i>yRes</i>	Place to store the y resolution
<i>bitsPerPixel</i>	Place to store the pixel depth

Return Value

True on success, false for an invalid graphics mode number.

Description

This function returns the pixel resolution and depth of the specified MGL mode number, and can be used to display this information to the end user, or to search for specific graphics modes for use by an application program depending on the desired resolution or color depth.

See Also

MGL_modeName, *MGL_driverName*

MGL_moveRel

Moves the CP to a new relative location.

Declaration

```
void MGL_moveRel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p

Use coordinates of this point as offsets

Description

This function is the same as *MGL_moveRelCoord*, however it takes the amount to move as a point.

See Also

MGL_moveRelCoord

MGL_moveRelCoord

Moves the CP to a new relative location.

Declaration

```
void MGLAPI MGL_moveRelCoord(  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>dx</i>	Amount to move x coordinate
<i>dy</i>	Amount to move y coordinate

Description

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_moveRel

MGL_moveTo

Moves the CP to a new location.

Declaration

```
void MGL_moveTo(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p New Point for CP

Description

This function is the same as *MGL_moveToCoord*, however it takes the coordinate to move to as a point.

See Also

MGL_moveToCoord

MGL_moveToCoord

Moves the CP to a new location.

Declaration

```
void MGLAPI MGL_moveToCoord(  
    int x,  
    int y)
```

Prototype In
mgraph.h

Parameters

<i>x</i>	New x coordinate for CP
<i>y</i>	New y coordinate for CP

Description

Moves the current position (CP) to the new point (x,y).

See Also

MGL_moveTo

MGL_newRegion

Allocate a new complex region.

Declaration

```
region_t * MGLAPI MGL_newRegion(void)
```

Prototype In

mgraph.h

Return Value

Pointer to the new region, NULL if out of memory.

Description

Allocates a new complex region. The new region is empty when first created. Note that MGL maintains a local memory pool for all region allocations in order to provide the maximum speed and minimum memory overheads for region allocations.

See Also

MGL_freeRegion, *MGL_unionRegion*, *MGL_diffRegion*, *MGL_sectRegion*

MGL_offsetRect

Offsets a rectangle by the specified amount.

Declaration

```
void MGL_offsetRect(  
    rect_t r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>r</i>	Rectangle to offset
<i>dx</i>	Amount to offset x coordinates by
<i>dy</i>	Amount to offset y coordinates by

Description

This function offsets the specified rectangle by the dx and dy coordinates. This function effectively performs the following operation on the rectangle:

```
left += dx;  
top += dy;  
right += dx;  
bottom += dy;
```

See Also

MGL_insetRect

MGL_offsetRegion

Offsets a region by the specified amount.

Declaration

```
void MGLAPI MGL_offsetRegion(  
    region_t *r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>r</i>	Region to offset
<i>dx</i>	Amount to offset x coordinates by
<i>dy</i>	Amount to offset y coordinates by

Description

This function offsets the specified region by the dx and dy coordinates, by actually modifying all the coordinate locations for every rectangle in the union of rectangles that constitutes the region by the specified coordinates.

See Also

MGL_unionRegion, *MGL_diffRegion*, *MGL_sectRegion*

MGL_optimizeRegion

Optimizes the union of rectangles in the region to the minimum number of rectangles.

Declaration

```
void MGLAPI MGL_optimizeRegion(  
    region_t *r)
```

Prototype In

mgraph.h

Parameters

r Region to optimize

Description

This function optimizes the specified region by traversing the region structure looking for identical spans in the region. The region algebra functions (*MGL_unionRegion*, *MGL_diffRegion*, *MGL_sectRegion* etc.) do not fully optimize the resulting region to save time, so after you have created a complex region you may wish to call this routine to optimize it.

Optimizing the region will find the minimum number of rectangles required to represent that region, and will result in faster drawing and traversing of the resulting region.

See Also

MGL_unionRegion, *MGL_diffRegion*, *MGL_sectRegion*

Pack an RGB tuple into an MGL color.

Declaration

```
color_t ASMAPI MGL_packColor(  
    pixel_format_t *pf,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function takes an RGB tuple of 8 bit color components and packs them into an MGL packed color value. The color components are packed according to the specified pixel format information, which can be obtained directly from the mode information for a bitmap or an MGL device context. When running in dithered 8 bit modes you may wish to use the *MGL_packColorRGB* functions which will be faster.

See Also

MGL_packColorFast, *MGL_packColorRGB*, *MGL_unpackColor*

MGL_packColorFast

Pack an RGB tuple into an MGL color.

Declaration

```
color_t MGL_packColorFast(  
    pixel_format_t *pf,  
    uchar r,  
    uchar g,  
    uchar b)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function is the same as *MGL_packColor*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_packColor, *MGL_packColorRGB*, *MGL_unpackColor*

MGL_packColorRGB

Pack an RGB tuple into a packed 24 bit color.

Declaration

```
color_t ASMAPI MGL_packColorRGB(  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)
<i>c</i>	Color into which to pack values (syntax 3)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function takes an RGB tuple of 8 bit color components and packs them into a 24 bit packed color value. This function is useful for packing 24 bit color values to be passed to MGL when running in dithered 8 bit modes.

See Also

MGL_packColorRGBFast

MGL_packColorRGBFast

Pack an RGB tuple into a packed 24 bit color.

Declaration

```
color_t MGL_packColorRGBFast(  
    uchar r,  
    uchar b,  
    uchar g)
```

Prototype In

mgraph.h

Parameters

<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)
<i>c</i>	Color into which to pack values (syntax 3)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function is the same as *MGL_packColorRGB*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_packColorRGB

MGL_packColorRGBFast2

Pack an RGB tuple into a packed 24 bit color.

Declaration

```
void MGL_packColorRGBFast2(  
    color_t c,  
    uchar r,  
    uchar g,  
    uchar b)
```

Prototype In

mgraph.h

Parameters

<i>c</i>	Color into which to pack values (syntax 3)
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Description

This function is the same as *MGL_packColorRGBFast*, however it is implemented as a macro and hence is more efficient. It also packs the color value directly into the *c* parameter without requiring you to pass a pointer (because it is a macro).

See Also

MGL_packColorRGBFast, *MGL_packColorRGB*

MGL_pixel

Draws a pixel at the specified location.

Declaration

```
void MGL_pixel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to plot pixel at

Description

This function is the same as *MGL_pixelFast*, however it takes the coordinate of the pixel to plot as a point.

See Also

MGL_pixelCoord

MGL_pixelCoord

Draws a pixel at the specified location.

Declaration

```
void MGLAPI MGL_pixelCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to plot pixel at
<i>y</i>	y coordinate to plot pixel at

Description

Plots a single pixel at the specified location in the current foreground color.

See Also

MGL_pixel

MGL_pixelCoordFast

Draws a pixel at the specified location as fast as possible.

Declaration

```
void MGLAPI MGL_pixelCoordFast(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

x	x coordinate to plot pixel at
y	y coordinate to plot pixel at

Description

Plots a single pixel at the specified location in the current foreground color. This routine is designed to allow plotting of multiple pixels as fast as possible.

Note that you must call *MGL_beginPixel* before calling this function, and you must call *MGL_endPixel* after you have finished plotting a number of pixels.

See Also

MGL_pixelFast, *MGL_beginPixel*, *MGL_endPixel*.

MGL_pixelFast

Draws a pixel at the specified location as fast as possible.

Declaration

```
void MGL_pixelFast(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to plot pixel at

Description

This function is the same as *MGL_pixelCoordFast*, however it takes the coordinate of the pixel to plot as a point.

See Also

MGL_pixelCoordFast, *MGL_beginPixel*, *MGL_endPixel*.

Draws a set of connected lines.

Declaration

```
void MGLAPI MGL_polyLine(  
    int count,  
    point_t *vArray)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polyline
<i>vArray</i>	Array of vertices in the polyline

Description

This function 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 that 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

Draws a set of markers.

Declaration

```
void MGLAPI MGL_polyMarker(  
    int count,  
    point_t *vArray)
```

Prototype In

mgraph.h

Parameters

count

Number of vertices in polyline

vArray

Array of coordinates to draw the markers at

Description

This function 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

Draws a set of pixels.

Declaration

```
void MGLAPI MGL_polyPoint(  
    int count,  
    point_t *vArray)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polyline
<i>vArray</i>	Array of coordinates to draw the pixels at

Description

This function draws a set of pixels in the current color at the locations passed in the vArray parameter.

See Also

MGL_polyLine, *MGL_polyMarker*

MGL_ptInRect

Returns true if supplied point is within the definition of a rectangle, otherwise false.

Declaration

```
bool MGL_ptInRect(  
    point_t p,  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Point to test
<i>r</i>	Rectangle to test

Return Value

True if supplied point is within the definition of the specified rectangle.

Description

This function tests whether a point is within the bounds of a rectangle or not. This version takes the coordinates of the rectangle as two points.

See Also

MGL_ptInRect

MGL_ptInRectCoord

Returns true if supplied point is within the definition of a rectangle, otherwise false.

Declaration

```
bool MGL_ptInRectCoord(  
    int x,  
    int y,  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	X coordinate of rectangle to test
<i>y</i>	Y coordinate of rectangle to test
<i>r</i>	Rectangle to test

Return Value

True if supplied point is within the definition of the specified rectangle.

Description

This function tests whether a point is within the bounds of a rectangle or not.

See Also

MGL_ptInRect

MGL_ptInRegion

Determines if a point is contained in a specified region.

Declaration

```
bool MGL_ptInRegion(  
    point_t p,  
    region_t r)
```

Prototype In

mgraph.h

Parameters

<i>p</i> <i>rgn</i>	point structure containing coordinate to test Region to test
--------------------------------------	---

Return Value

True if the point is contained in the region, false if not.

Description

This function is the same as *MGL_ptInRegionCoord*, however it takes the coordinate of the point to test as a point not two coordinates.

See Also

MGL_ptInRegion

MGL_ptInRegionCoord

Determines if a point is contained in a specified region.

Declaration

```
bool MGLAPI MGL_ptInRegionCoord(  
    int x,  
    int y,  
    const region_t *rgn)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to test for inclusion
<i>y</i>	y coordinate to test for inclusion
<i>rgn</i>	Region to test

Return Value

True if the point is contained in the region, false if not.

Description

This function determines if a specified point is contained within a particular region. Note that if the region has a hole it in, and the point lies within the hole, then the point is classified as not being included in the region.

See Also

MGL_ptInRegion

MGL_putBitmap

Draw a lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmap(  
    MGLDC *dc,  
    int x,  
    int y,  
    const bitmap_t *bitmap,  
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a lightweight bitmap at the specified location. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

Note that for maximum performance when displaying 8 bit bitmaps, you should ensure that the color palette for the bitmap is identical to the device context, otherwise the pixels in the bitmap will be translated during the draw operation.

Supported write modes are enumerated in *MGL_writeModeType*.

See Also

MGL_loadBitmap

MGL_putBitmapMask

Draw a lightweight bitmap mask in the specified color.

Declaration

```
void MGLAPI MGL_putBitmapMask(  
    MGLDC *dc,  
    int x,  
    int y,  
    const bitmap_t *mask,  
    color_t color)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>mask</i>	Monochrome bitmap mask to display
<i>color</i>	Color to draw in

Description

Draws a lightweight monochrome bitmap at the specified location. This is just a simply utility function that draws the monochrome bitmap in a specified color in replace mode as fast as possible.

See Also

MGL_loadBitmap

MGL_putBitmapSection

Draw a section of a lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmapSection(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int dstLeft,  
    int dstTop,  
    const bitmap_t *bitmap,  
    int op)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a section of a lightweight bitmap at the specified location. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

Note that for maximum performance when displaying 8 bit bitmaps, you should ensure that the color palette for the bitmap is identical to the device context, otherwise the pixels in the bitmap will be translated during the draw operation.

See Also

MGL_loadBitmap

MGL_putBitmapTransparent

Draw a transparent lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmapTransparent(  
    MGLDC *dc,  
    int x,  
    int y,  
    const bitmap_t *bitmap,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

Draws a transparent lightweight bitmap at the specified location with either source or destination transparency. When transferring the data with source transparency, pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

The pixel depth and pixel format for the source bitmap and the device contexts must be the same or this routine will simply do nothing. This

routine also only works with pixel depths that are at least 8 bits deep.

See Also

MGL_loadBitmap, MGL_putBitmapTransparentSection

MGL_putBitmapTransparentSection

Draw a section of a transparent lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmapTransparentSection(  
    MGLDC *dc,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int dstLeft,  
    int dstTop,  
    const bitmap_t *bitmap,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

Draws a section of a transparent lightweight bitmap at the specified location with either source or destination transparency. When transferring the data with source transparency, pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be

updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

The pixel depth and pixel format for the source bitmap and the device contexts must be the same or this routine will simply do nothing. This routine also only works with pixel depths that are at least 8 bits deep.

See Also

MGL_loadBitmap, MGL_putBitmapTransparent

MGL_putDivot

Replaces a divot of video memory to the location from which it was copied.

Declaration

```
void MGLAPI MGL_putDivot(  
    MGLDC *dc,  
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to restore the divot to
<i>divot</i>	Pointer to the divot to replace

Description

This function replaces a rectangle of video memory that was saved previously with the *MGL_getDivot* function. The divot is replaced at the same location that it was taken from on the current device context.

A divot is defined as being a rectangular area of video memory that you wish to save, however the bounding rectangle for the divot is expanded slightly to properly aligned boundaries for the absolute maximum performance with the current device context. This function is generally used to store the video memory behind pull down menus and pop up dialog boxes, and the memory can only be restored to exactly the same position that it was saved from.

See Also

MGL_divotSize

Draw an icon at the specified location.

Declaration

```
void MGLAPI MGL_putIcon(  
    MGLDC *dc,  
    int x,  
    int y,  
    const icon_t *icon)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to draw icon on
<i>x</i>	x coordinate to draw icon at
<i>y</i>	y coordinate to draw icon at
<i>icon</i>	Icon to display

Description

Draws an icon at the specified location on the current device context. The icon may be in any color format, and will be translated as necessary to the color format of the display device context. The icon is drawn by punching a black hole in the background with the icon mask, and then OR'ing in the image bitmap for the icon.

See Also

MGL_loadIcon

MGL_putMonoImage

Draw a monochrome bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putMonoImage(MGLDC *dc,  
    int x,  
    int y,  
    int byteWidth,  
    int height,  
    void *image)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to draw bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>byteWidth</i>	Width of the bitmap in bytes
<i>height</i>	Height of the bitmap in scanlines
<i>image</i>	Pointer to the buffer holding the bitmap

Description

This function draws a monochrome bitmap in the current foreground color on the current device context. Where a bit is a 1 in the bitmap definition, a pixel is plotted in the foreground color, where a bit is a 0 the original pixels are left alone. This function can be used to implement fast hardware pixel masking for drawing fast transparent bitmaps on devices that do not have a native hardware transparent BitBlt function.

MGL_random

Generate a random 16-bit number between 0 and max.

Declaration

```
ushort ASMAPI MGL_random(  
    ushort max)
```

Prototype In

mgraph.h

Parameters

max Largest desired value

Return Value

Random 16-bit number between 0 and max.

See Also

MGL_randoml, *MGL_srand*

MGL_randoml

Generate a random 32-bit number between 0 and max.

Declaration

```
ulong ASMAPI MGL_randoml(  
    ulong max)
```

Prototype In

mgraph.h

Parameters

max Largest desired value

Return Value

Random 32-bit number between 0 and max.

See Also

MGL_random, *MGL_srand*

Returns the real packed MGL color for a color index.

Declaration

```
color_t MGLAPI MGL_realColor(  
    MGLDC *dc,  
    int color)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to map color with
<i>color</i>	Color to map

Return Value

Real packed MGL color value for the color index.

Description

This function returns a packed MGL color value appropriate for the specified device context given a color index. This routine works with all device contexts, including RGB device contexts. For the color index devices, the value is simply returned unchanged. However for RGB devices, the color index is translated via the current color palette for the device to find the appropriate packed MGL color value for that device. Thus you can still write code for RGB devices that works with color indexes (although you cannot do things like hardware palette fades and rotates as the palette is implemented in software).

See Also

MGL_setColorCI, *MGL_setColorRGB*, *MGL_setPalette*, *MGL_getPalette*

MGL_realizePalette

Realizes the hardware color palette for the display device context.

Declaration

```
void MGLAPI MGL_realizePalette(  
    MGLDC *dc,  
    int numColors,  
    int startIndex,  
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to realize palette for
<i>numColors</i>	Number of colors to realize
<i>startIndex</i>	Starting index of first color to realize
<i>waitVRT</i>	True if routine should sync to vertical retrace, false if not.

Description

This function realizes the hardware palette associated with a display device context. Calls to *MGL_setPalette* only update the palette values in the color palette for the device context structure, but do not actually program the hardware palette for display device contexts in 4 and 8 bits per pixel modes. In order to program the hardware palette you must call this routine.

When the hardware palette is realized, you normally need to sync to the vertical retrace to ensure that the palette values are programmed without the onset of snow (see *MGL_setPaletteSnowLevel* to adjust the number of colors programmed per retrace period). If however you wish to perform double buffered animation and change the hardware color palette at the same time, you should call this routine immediately after calling either *MGL_setVisualPage* or *MGL_swapBuffers* with the waitVRT flag set to false.

See Also

MGL_setPalette, *MGL_setVisualPage*, *MGL_swapBuffers*,
MGL_setPaletteSnowLevel

MGL_rect

Draws a rectangle outline.

Declaration

```
void MGL_rect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>r</i>	Rectangle to draw
-----------------	-------------------

Description

This function is the same as *MGL_rectCoord*, however it takes an entire rectangle as the parameter instead of coordinates.

See Also

MGL_rectCoord, MGL_rectPt

Draws a rectangle outline.

Declaration

```
void MGLAPI MGL_rectCoord(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left coordinate of the rectangle
<i>top</i>	Top coordinate of the rectangle
<i>right</i>	Right coordinate of the rectangle
<i>bottom</i>	Bottom coordinate of the rectangle
<i>r</i>	Rectangle to draw
<i>leftTop</i>	Point for upper left corner of rectangle
<i>rightBottom</i>	Point containing lower right corner of rectangle

Description

This function draws a rectangle outline in the current drawing attributes at the specified location.

See Also

MGL_rect, *MGL_rectPt*

MGL_rectPt

Draws a rectangle outline.

Declaration

```
void MGL_rectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Parameters

r Rectangle to draw

Description

This function is the same as *MGL_rectCoord*, however it takes the top left and bottom right coordinates of the rectangle as two points instead of four coordinates.

See Also

MGL_rectCoord, *MGL_rect*

MGL_registerAllDispDrivers

Registers all available display drivers to be linked in and used for detection.

Declaration

```
void MGLAPI MGL_registerAllDispDrivers(  
    bool useLinear,  
    bool useDirectDraw,  
    bool useWinDirect)
```

Prototype In

mgraph.h

Parameters

<i>useLinear</i>	True if linear drivers should be registered
<i>useWinDirect</i>	True if WinDirect drivers should be registered
<i>useDirectDraw</i>	True if DirectDraw drivers should be registered

Description

This function is a backwards compatibility function for older versions of the library, and simply calls *MGL_registerAllDispDriversExt* to achieve the same functionality. Please see the new function for more information.

See Also

MGL_registerAllDispDriversExt

MGL_registerAllDispDriversExt

Registers all available display drivers to be linked in and used for detection.

Declaration

```
void MGLAPI MGL_registerAllDispDriversExt(  
    bool useWinDirect,  
    bool useDirectDraw,  
    bool useVGA,  
    bool useVGAX,  
    bool useVBE,  
    bool useLinear,  
    bool useVBEAF,  
    bool useFullscreenDIB)
```

Prototype In

mgraph.h

Parameters

useWinDirect	True if WinDirect drivers should be registered
useDirectDraw	True if DirectDraw drivers should be registered
useVGA	True if VGA drivers should be registered
useVGAX	True if VGAX drivers should be registered
useVBE	True if VBE drivers should be registered
useLinear	True if linear drivers should be registered
useVBEAF	True if VBEAF drivers should be registered
useFullscreenDIB	True if fullscreen DIB drivers should be registered

Description

This function registers all available display drivers to be linked in, and also registers those drivers in the device detection chain. If you pass a value of true for useDirectDraw, all the DirectDraw drivers will be registered for Windows (this parameter is ignored under DOS). If you pass a value of true for useWinDirect, all the WinDirect drivers will be registered. These parameters are useful for allowing the user to interactively force the use of DirectDraw or WinDirect dynamically from within your application (see the sample programs which all provide this support under Windows and the *MGL_detectGraph* function for more information).

The useVGA, useVGAX, useVBE, useLinear and useVBEAF parameters all control the use of WinDirect device drivers, so if you pass a value of false to useWinDirect, it is the same as passing a value of false in all of the above parameters.

This is a useful function to get code going quickly, but for release code you should only link in those display drivers for the video modes that your application will be using to reduce the size of the resulting executable.

See Also

MGL_registerAllDispDrivers, MGL_registerDriver, MGL_detectGraph

MGL_registerAllMemDrivers

Registers all known packed pixel memory drivers to be linked.

Declaration

```
void MGLAPI MGL_registerAllMemDrivers(void)
```

Prototype In

mgraph.h

Description

This function registers all available memory device context drivers to be linked in, and also registers those drivers in the device detection chain. This is a useful function to get code up and running quickly. However, in order to minimize the size of the resulting executable for release code, you should only link in those drivers which will actually be used by your application.

See Also

MGL_registerDriver

MGL_registerAllOpenGLDrivers

Registers all OpenGL drivers to be linked in and used for detection

Declaration

```
void MGLAPI MGL_registerAllOpenGLDrivers(void)
```

Prototype In

mgraph.h

Description

This function registers all available OpenGL hardware display drivers to be linked in, and also registers those drivers in the device detection chain. This is a useful function to get code up and running quickly. However, in order to minimize the size of the resulting executable for release code, you should only link in those drivers which will actually be used by your application.

See Also

MGL_registerDriver

MGL_registerDriver

Registers a device driver to be linked in.

Declaration

```
int MGLAPI MGL_registerDriver(  
    const char *name,  
    void *driver)
```

Prototype In

mgraph.h

Return Value

grOK on success, grBadDriver if driver was invalid.

Description

This function registers a specific device driver to be linked into the application code, and to be used in the automatic device detection process. This function is used to link in both display device drivers and packed pixel memory device drivers. If you do not have the proper device driver loaded you will not be able to create a device context that requires that driver. For instance if you do not register the 8 bit packed pixel device, and attempt to create an 8 bit memory device context, the function will fail.

The names of the standard fullscreen display device drivers currently supported are:

<i>Driver</i>	Description
<i>MGL_VGA4NAME</i>	Standard VGA 4 bit display driver
<i>MGL_VGA8NAME</i>	Standard VGA 8 bit display driver (32 bit only)
<i>MGL_VGAXNAME</i>	Standard VGA ModeX 8 bit display driver
<i>MGL_SVGA4NAME</i>	VBE 1.x 4 bit display driver
<i>MGL_SVGA8NAME</i>	VBE 1.x 8 bit display driver
<i>MGL_SVGA16NAME</i>	VBE 1.x 15/16 bit display driver
<i>MGL_SVGA24NAME</i>	VBE 1.x 24 bit display driver
<i>MGL_SVGA32NAME</i>	VBE 1.x 32 bit display driver
<i>MGL_LINEAR8NAME</i>	VBE 2.0 8 bit linear framebuffer display driver
<i>MGL_LINEAR16NAME</i>	VBE 2.0 15/16 bit linear framebuffer display driver
<i>MGL_LINEAR24NAME</i>	VBE 2.0 24 bit linear framebuffer display driver
<i>MGL_LINEAR32NAME</i>	VBE 2.0 32 bit linear framebuffer display driver
<i>MGL_ACCEL8NAME</i>	VBE/AF 8 bit accelerated display driver
<i>MGL_ACCEL16NAME</i>	VBE/AF 15/16 bit accelerated display driver
<i>MGL_ACCEL24NAME</i>	VBE/AF 24 bit accelerated display driver
<i>MGL_ACCEL32NAME</i>	VBE/AF 32 bit accelerated display driver
<i>MGL_DDRAW8NAME</i>	DirectDraw 8 bit accelerated display driver
<i>MGL_DDRAW16NAME</i>	DirectDraw 15/16 bit accelerated display driver
<i>MGL_DDRAW24NAME</i>	DirectDraw 24 bit accelerated display driver
<i>MGL_DDRAW32NAME</i>	DirectDraw 32 bit accelerated display driver
<i>MGL_FULLDIB8NAME</i>	Fullscreen DIB 8 bit display driver
<i>MGL_FULLDIB16NAME</i>	Fullscreen DIB 15/16 bit display driver
<i>MGL_FULLDIB24NAME</i>	Fullscreen DIB 24 bit display driver
<i>MGL_FULLDIB32NAME</i>	Fullscreen DIB 32 bit display driver
<i>MGL_OPENGLNAME</i>	Hardware Microsoft OpenGL display driver
<i>MGL_FSOGL8NAME</i>	Hardware 8 bit fullscreen OpenGL display driver
<i>MGL_FSOGL16NAME</i>	Hardware 15/16 bit fullscreen OpenGL display driver
<i>MGL_FSOGL24NAME</i>	Hardware 24 bit fullscreen OpenGL display driver
<i>MGL_FSOGL32NAME</i>	Hardware 32 bit fullscreen OpenGL display driver

The names of the standard packed pixel memory device drivers currently supported are:

<i>Driver</i>	Description
<i>MGL_PACKED1NAME</i>	Packed pixel 1 bit memory driver
<i>MGL_PACKED4NAME</i>	Packed pixel 4 bit memory driver
<i>MGL_PACKED8NAME</i>	Packed pixel 8 bit memory driver
<i>MGL_PACKED16NAME</i>	Packed pixel 15/16 bit memory driver
<i>MGL_PACKED24NAME</i>	Packed pixel 24 bit memory driver
<i>MGL_PACKED32NAME</i>	Packed pixel 32 bit memory driver

You must also pass the address of the driver, which is the type and color depth of the driver but with '_driver' added to the end. For instance to register just the 8 bit VBE SuperVGA driver, you would use the following call:

```
MGL_registerDriver(MGL_SVGA8NAME, SVGA8_driver);
```

See Also

MGL_init, MGL_detectGraph, MGL_unregisterAllDrivers

MGL_registerEventProc

Registers a user supplied window procedure for event handling.

Declaration

```
void MGLAPI MGL_registerEventProc(  
    WNDPROC userWndProc)
```

Prototype In

mglwin.h

Parameters

userWndProc Point to user supplied Window Procedure

Description

This function registers a user supplied window procedure with MGL that will be used for event handling purposes. By default MGL applications can simply use the `EVT_*` event handling functions that are common to both the DOS and Windows versions of MGL. However developers porting Windows specific code from DirectDraw may wish to use their existing window specific event handling code. This function allows you to do this by telling MGL to use your window procedure for all event processing.

See Also

EVT_getNext

MGL_registerFullScreenWindow

Registers a user window with SciTech MGL to be used for fullscreen modes

Declaration

```
void MGLAPI MGL_registerFullScreenWindow(  
    HWND hwnd)
```

Prototype In

mglwin.c

Parameters

hwnd

Handle to user window to use for fullscreen modes.

Description

This function allows the application to create the window used for fullscreen modes, and let SciTech MGL know about the window so it will use that window instead of creating it's own fullscreen window. By default when you create a fullscreen device context, SciTech MGL will create a fullscreen window that covers the entire desktop that is used to capture Windows events such as keyboard events, mouse events and activation events. However in some situations it is beneficial to have only a single window that is used for all fullscreen graphics modes, as well as windows modes (primarily to be able to properly support DirectSound via a single window).

If you have registered a fullscreen window with SciTech MGL, it will take that window, zoom it fullscreen and modify the window styles and attributes to remove the title bar and other window decorations. When SciTech MGL then returns from fullscreen mode back to GDI mode, it will restore the window back to the original position, style and state it was in before the fullscreen mode was created. This allows you to create a single window with SciTech MGL and use it for both windowed modes and fullscreen modes.

Note: *When SciTech MGL exits via a call to MGL exit, the fullscreen window that you passed in will be destroyed (necessary to work around bugs in DirectDraw).*

MGL_resizeWinDC

Resizes the windowed device context after a WM_SIZE message has been captured.

Declaration

```
void MGLAPI MGL_resizeWinDC(  
    MGLDC *dc)
```

Parameters

dc MGL device context to resize

Description

This is more efficient than re-creating the windowed device context for every WM_SIZE message (especially if OpenGL has been initialized for the device context).

MGL_restoreAttributes

Restores a previously saved set of rasterizing attributes.

Declaration

```
void MGLAPI MGL_restoreAttributes(  
    attributes_t *attr)
```

Prototype In

mgraph.h

Parameters

attr Pointer to the attribute list to restore

Description

This function restores a set of attributes that were saved with the *MGL_getAttributes* routine. The attributes list represents the current state of MGL. The value of the color palette is not changed by this routine.

See Also

MGL_getAttributes

MGL_result

Returns result code of the last graphics operation.

Declaration

```
int MGLAPI MGL_result(void)
```

Prototype In

mgraph.h

Return Value

Result code of the last graphics operation

Description

This function 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. Error codes returned by this function are enumerated in *MGL_errorType*.

See Also

MGL_setResult

MGL_resume

Resume low level event handling code.

Declaration

```
void MGL_resume(void)
```

Prototype In

mgldos.h

Description

Resumes the event handling code for MGL. This function should be used to re-enable the MGL event handling code after shelling out to DOS from your application code or running another application.

See Also

MGL_suspend, *MGL_init*

MGL_rgbBlue

Extract the blue component from a packed 24 bit RGB tuple.

Declaration

```
uchar  MGL_rgbBlue(  
    color_t c)
```

Prototype In

mgraph.h

Parameters

c Packed 24 bit color value to extract value from

Return Value

Blue component from the packed 24 bit color value.

See Also

MGL_rgbRed, MGL_rgbGreen

MGL_rgbColor

Computes a packed MGL color from a 24 bit RGB tuple.

Declaration

```
color_t MGLAPI MGL_rgbColor(  
    MGLDC *dc,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	Device context to map color with
<i>R</i>	Red component of color to map (0 - 255)
<i>G</i>	Green component of color to map (0 - 255)
<i>B</i>	Blue component of color to map (0 - 255)

Return Value

Packed MGL color closest to specified RGB tuple.

Description

This function computes the packed MGL color value from a specific 24 bit RGB tuple for a device context. If the device context is an RGB device context or an 8 bit device in RGB dithered mode, this value simply returns the proper packed MGL pixel value representing this color (the same as *MGL_packColor* would). However if the device context is a color index device, the color palette is searched for the color value that is the closest to the specified color. This function allows you to specify a color given an RGB tuple, and will work in color index modes as well with any color palette.

See Also

MGL_realColor, *MGL_setColorCI*, *MGL_setColorRGB*

MGL_rgbGreen

Extract the green component from a packed 24 bit RGB tuple.

Declaration

```
uchar  MGL_rgbGreen(  
    color_t c)
```

Prototype In

mgraph.h

Parameters

c Packed 24 bit color value to extract value from

Return Value

Green component from the packed 24 bit color value.

See Also

MGL_rgbRed, *MGL_rgbBlue*

MGL_rgbRed

Extract the red component from a packed 24 bit RGB tuple.

Declaration

```
uchar  MGL_rgbRed(  
    color_t c)
```

Prototype In

mgraph.h

Parameters

c Packed 24 bit color value to extract value from

Return Value

Red component from the packed 24 bit color value.

See Also

MGL_rgbGreen, MGL_rgbBlue

MGL_rgnEllipse

Generate an ellipse outline as a region.

Declaration

```
region_t * MGLAPI MGL_rgnEllipse(  
    rect_t extentRect,  
    const region_t *_pen)
```

Prototype In

mgraph.h

Parameters

extentRect

Bounding rectangle for the ellipse

pen

Region to use as the pen when drawing the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates the outline of an ellipse as a complex region by dragging the specified pen region around the perimeter of the ellipse. The pen used can be any arbitrary shape, however rectangular pens are special cased to provide fast region generation.

See Also

MGL_rgnEllipseArc

MGL_rgnEllipseArc

Generate an elliptical arc outline as a region.

Declaration

```
region_t * MGLAPI MGL_rgnEllipseArc(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle,  
    const region_t *_pen)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle for the ellipse
<i>startAngle</i>	Starting angle for the elliptical arc
<i>endAngle</i>	Ending angle for the elliptical arc
<i>pen</i>	Region to use as the pen when drawing the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates the outline of an elliptical arc as a complex region by dragging the specified pen region around the perimeter of the ellipse. The pen used can be any arbitrary shape, however rectangular pens are special cased to provide fast region generation.

See Also

MGL_rgnEllipse, *MGL_rgnGetArcCoords*

MGL_rgnGetArcCoords

Returns the real arc coordinates for an elliptical arc region.

Declaration

```
void MGLAPI MGL_rgnGetArcCoords(  
    arc_coords_t *coords)
```

Prototype In

mgraph.h

Parameters

coords Pointer to structure to store coordinates

Description

This function returns the center coordinate, and starting and ending points on the ellipse that defines the last elliptical arc region that was generated. You can then use these coordinates to draw a line from the center of the ellipse to the starting and ending points to complete the outline of an elliptical wedge.

Note that you must call this routine immediately after calling the *MGL_rgnEllipseArc* routine.

See Also

MGL_rgnEllipseArc

MGL_rgnLine

Generate a line as a region.

Declaration

```
region_t *MGL_rgnLine(  
    point_t p1,  
    point_t p2,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint
<i>p2</i>	Second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

This function is the same as *MGL_rgnLine* but takes the parameters for the line as two points instead of coordinates.

See Also

MGL_rgnLineCoord, *MGL_rgnLineFX*

MGL_rgnLineCoord

Generate a line as a region.

Declaration

```
region_t * MGLAPI MGL_rgnLineCoord(  
    int x1,  
    int y1,  
    int x2,  
    int y2,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

Generates a region as a line starting at the point (x1,y1) and ending at the point (x2,y2). Note that this function takes the coordinates of the lines in integer format. MGL draws all lines internally in 16.16 fixed point format, so this routine always converts the coordinates and then calls the *MGL_rgnLineFX* routine. If you need maximum performance, you should call the fixed point line drawing functions instead.

See Also

MGL_rgnLine, *MGL_rgnLineFX*

MGL_rgnLineCoordFX

Generate a fixed point line as a region.

Declaration

```
region_t * MGLAPI MGL_rgnLineCoordFX(  
    fix32_t x1,  
    fix32_t y1,  
    fix32_t x2,  
    fix32_t y2,  
    const region_t *_pen)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

Generates a region as a line starting at the point (x1,y1) and ending at the point (x2,y2).

See Also

MGL_rgnLineFX, *MGL_rgnLine*

MGL_rgnLineFX

Generate a fixed point line as a region.

Declaration

```
region_t *MGL_rgnLineFX(  
    fxpoint_t p1,  
    fxpoint_t p2,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint
<i>p2</i>	Second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

This function is the same as *MGL_rgnLineFX* but takes the parameters for the line as two points instead of coordinates.

See Also

MGL_rgnLineCoordFX, *MGL_rgnLine*

MGL_rgnSolidEllipse

Generates a solid ellipse as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidEllipse(  
    rect_t extentRect)
```

Prototype In

mgraph.h

Parameters

extentRect Bounding rectangle for the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid ellipse as a complex region

See Also

MGL_rgnSolidEllipseArc

MGL_rgnSolidEllipseArc

Generates a solid elliptical arc as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidEllipseArc(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle for the ellipse
<i>startAngle</i>	Starting angle for the elliptical arc
<i>endAngle</i>	Ending angle for the elliptical arc

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid elliptical arc as a region.

See Also

MGL_rgnEllipse, *MGL_rgnGetArcCoords*

MGL_rgnSolidRect

Generate a solid rectangle as a region from two points.

Declaration

```
void MGL_rgnSolidRect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle the coordinates of the region

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRectCoord, *MGL_rgnSolidRectPt*

MGL_rgnSolidRectCoord

Generate a solid rectangle as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidRectCoord(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left coordinate of the rectangle
<i>top</i>	Top coordinate of the rectangle
<i>right</i>	Right coordinate of the rectangle
<i>bottom</i>	Bottom coordinate of the rectangle
<i>r</i>	Rectangle to generate as a region
<i>lt</i>	Point containing left-top coordinates of the region
<i>rb</i>	Point containing right-bottom coordinates of the region

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRect, *MGL_rgnSolidRectPt*

MGL_rgnSolidRectPt

Generate a solid rectangle as a region from two points.

Declaration

```
region_t MGL_rgnSolidRectPt(  
    point_t lt,  
    point_t rb)
```

Prototype In

mgraph.h

Parameters

lt
rb

Point containing left-top coordinates of the region
Point containing right-bottom coordinates of the region

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRectCoord, *MGL_rgnSolidRect*

MGL_rightBottom

Returns the bottom right coordinate from a rectangle.

Declaration

```
point_t MGL_rightBottom(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to get coordinate from

Return Value

The bottom right coordinate of the rectangle.

Description

Returns the bottom right coordinates from a rectangle.

See Also

MGL_leftTop

MGL_rotateGlyph

Declaration

```
void MGLAPI MGL_rotateGlyph(  
    uchar *dst,  
    uchar *src,  
    int *byteWidth,  
    int *height,  
    int rotation)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination glyph buffer
<i>byteWidth</i>	Width of the glyph in bytes
<i>height</i>	Height of the glyph in scanlines
<i>rotation</i>	Rotation direction for the glyph

Description

This function computes the rotated image glyph of the source glyph, and stores the value in the destination buffer. The source buffer is not modified, and the rotated image glyph may possibly be larger than the source glyph. The resulting width and height of the destination glyph is returned. Supported directions are enumerated in *MGL_textDirType*

Note: *You must preallocate enough space to hold the rotated glyph in the destination buffer, as this may actually be larger than the source glyph.*

The final size will be the following:

$$(\text{height} + 7) / 8 + \text{byteWidth} * 8$$

MGL_rotatePalette

Rotates the palette values for a device context.

Declaration

```
void WINAPI MGL_rotatePalette(  
    MGLDC *dc,  
    int numColors,  
    int startIndex,  
    int direction)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context containing palette
<i>numColors</i>	Number of colors to rotate
<i>startIndex</i>	Starting index for colors to rotate
<i>direction</i>	Direction to rotate the palette entries

Description

This function rotates the palette values in the device context in the specified direction. Note that this routine does not effect the currently active hardware palette, and you must call *MGL_realizePalette* in order to make the program the rotated palette to the hardware.

Supported directions of rotation are enumerated in *MGL_palRotateType*.

When the direction specified is *MGL_ROTATE_UP*, the first entry in the palette is moved to the last position in the palette, and all the remaining entries are moved one position up in the array. If the direction specified is *MGL_ROTATE_DOWN*, the last entry is moved into the first entry of the palette, and the remaining entries are all moved one position down in the array.

See Also

MGL_setPalette, *MGL_getPalette*, *MGL_fadePalette*

MGL_saveBitmapFromDC

Save a portion of a device context to bitmap file on disk.

Declaration

```
bool MGLAPI MGL_saveBitmapFromDC(  
    MGLDC *dc,  
    const char *bitmapName,  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>bitmapName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_loadBitmap, *MGL_loadBitmapIntoDC*

MGL_saveJPEGFromDC

Save a portion of a device context to JPEG on disk.

Declaration

```
bool MGLAPI MGL_saveJPEGFromDC(  
    MGLDC *dc,  
    const char *JPEGName,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int quality)
```

Prototype In mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>JPEGName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save
<i>quality</i>	Quality factor for compression (1-100)

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a JPEG format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

Note that MGL currently only supports saving bitmap data to JPEG files from 8 bits per pixel device contexts.

Note: *In order to use this function you must link with the separate JPEG.LIB library for your compiler.*

See Also

MGL_LoadJPEG, MGL_loadJPEGIntoDC

MGL_savePCXFromDC

Save a portion of a device context to PCX on disk.

Declaration

```
bool MGLAPI MGL_savePCXFromDC(  
    MGLDC *dc,  
    const char *PCXName,  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>PCXName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a PCX format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

Note that MGL currently only supports saving bitmap data to PCX files from 8 bits per pixel device contexts.

See Also

`MGL_LoadPCX`, *`MGL_loadPCXIntoDC`*

MGL_scanLeftForColor

Scans left in viewport surface for a specified color.

Declaration

```
int WINAPI MGL_scanLeftForColor(  
    int x,  
    int y,  
    color_t color)
```

Prototype In

mgraph.h

Parameters

x	Starting x coordinate to scan from
y	Starting y coordinate to scan from
color	Color value to scan for

Return Value

x coordinate of pixel if found, or -1 if search hit left edge of viewport.

Description

This function begins scanning in the viewport 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. If a pixel is found that matches the color, the x coordinate of the pixel is returned. If the search went beyond the left edge of the viewport -1 is returned.

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.

See Also

MGL_scanRightForColor, MGL_scanLeftWhileColor, MGL_scanRightWhileColor

MGL_scanLeftWhileColor

Scans left in viewport surface for any color but the specified color.

Declaration

```
int ASMAPI MGL_scanLeftWhileColor(  
    int x,  
    int y,  
    color_t color)
```

Prototype In

mgraph.h

Parameters

x	Starting x coordinate to scan from
y	Starting y coordinate to scan from
color	Color value to scan on

Return Value

x coordinate of pixel if found, -1 if search hit left edge of viewport.

Description

This function begins scanning in the viewport 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 viewport.

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.

See Also

MGL_scanLeftForColor, MGL_scanRightForColor, MGL_scanRightWhileColor

MGL_scanLine

Fills a specified scanline.

Declaration

```
void WINAPI 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

Description

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_penStyleType, *MGL_setPenBitmapPattern*, *MGL_setPenPixmapPattern*

MGL_scanRightForColor

Scans right in viewport for a specified color.

Declaration

```
int ASMAPI MGL_scanRightForColor(  
    int x,  
    int y,  
    color_t color)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	Starting x coordinate to scan from
<i>y</i>	Starting y coordinate to scan from
<i>color</i>	Color value to scan for

Return Value

x coordinate of pixel if found, or maxx+1 if search hit right edge of viewport

Description

This function begins scanning in the viewport 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 device context.

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.

See Also

MGL_scanLeftForColor, *MGL_scanLeftWhileColor*, *MGL_scanRightWhileColor*

MGL_scanRightWhileColor

Scans right in viewport for any color but the specified color.

Declaration

```
int WINAPI MGL_scanRightWhileColor(  
    int x,  
    int y,  
    color_t color)
```

Prototype In

mgraph.h

Parameters

x	Starting x coordinate to scan from
y	Starting y coordinate to scan from
color	Color value to scan on

Return Value

x coordinate of pixel if found, maxx+1 if search hit right edge of viewport

Description

This function begins scanning in the viewport 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 viewport.

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.

See Also

MGL_scanLeftForColor, MGL_scanRightForColor, MGL_scanLeftWhileColor

Compute the intersection between two rectangles.

Declaration

```
bool MGLAPI MGL_sectRect(  
    rect_t r1,  
    rect_t r2,  
    rect_t *d)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	First rectangle to intersect
<i>r2</i>	Second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

Return Value

True if the rectangles intersect, false if not.

Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

See Also

MGL_sectRectFast, *MGL_sectRectCoord*, *MGL_unionRect*

Compute the intersection between two rectangles.

Declaration

```
bool MGLAPI MGL_sectRectCoord(  
    int left1,  
    int top1,  
    int right1,  
    int bottom1,  
    int left2,  
    int top2,  
    int right2,  
    int bottom2,  
    rect_t *d)
```

Prototype In
mgraph.h

Parameters

<i>left1</i>	Left coordinate of first rectangle to intersect
<i>top1</i>	Top coordinate of first rectangle to intersect
<i>right1</i>	Right coordinate of first rectangle to intersect
<i>bottom1</i>	Bottom coordinate of first rectangle to intersect
<i>left2</i>	Left coordinate of second rectangle to intersect
<i>top2</i>	Top coordinate of second rectangle to intersect
<i>right2</i>	Right coordinate of second rectangle to intersect
<i>bottom2</i>	Bottom coordinate of second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

Return Value

True if the rectangles intersect, false if not.

Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

See Also

MGL_sectRectFastCoord, *MGL_sectRect*, *MGL_unionRect*

MGL_sectRectFast

Compute the intersection between two rectangles.

Declaration

```
void MGL_sectRectFast(  
    rect_t s1,  
    rect_t s2,  
    rect_t *d)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	First rectangle to intersect
<i>r2</i>	Second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

Return Value

True if the rectangles intersect, false if not.

Description

This is the same as *MGL_sectRect* but it is implemented as a macro and does not test the rectangle for intersection.

See Also

MGL_sectRect, *MGL_sectRectCoord*, *MGL_unionRect*

Compute the intersection between two rectangles.

Declaration

```
bool MGL_sectRectFastCoord(  
    int left1,  
    int top1,  
    int right1,  
    int bottom1,  
    int left2,  
    int top2,  
    int right2,  
    int bottom2,  
    rect_t *d)
```

Prototype In

mgraph.h

Parameters

<i>left1</i>	Left coordinate of first rectangle to intersect
<i>top1</i>	Top coordinate of first rectangle to intersect
<i>right1</i>	Right coordinate of first rectangle to intersect
<i>bottom1</i>	Bottom coordinate of first rectangle to intersect
<i>left2</i>	Left coordinate of second rectangle to intersect
<i>top2</i>	Top coordinate of second rectangle to intersect
<i>right2</i>	Right coordinate of second rectangle to intersect
<i>bottom2</i>	Bottom coordinate of second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

Return Value

True if the rectangles intersect, false if not.

Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

See Also

MGL_sectRectFastCoord, *MGL_sectRect*, *MGL_unionRect*

MGL_sectRegion

Compute the Boolean intersection between two regions.

Declaration

```
region_t * MGLAPI MGL_sectRegion(  
    const region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.h

Parameters

r1
r2

First region to compute intersection with
Second region to compute intersection with

Return Value

Resulting intersection region, or NULL if out of memory.

Description

Computes the Boolean intersection of two regions, returning the result in a new region. The region may actually be an empty region, in which case the bounding rectangle for the region will be an empty rectangle.

See Also

MGL_diffRegion, *MGL_unionRegion*, *MGL_sectRegionRect*

MGL_sectRegionRect

Compute the Boolean intersection between a region and a rectangle.

Declaration

```
region_t * MGLAPI MGL_sectRegionRect(  
    const region_t *r1,  
    const rect_t *r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	Region to compute intersection with
<i>r2</i>	Rectangle to compute intersection with

Return Value

Resulting intersection region, or NULL if out of memory.

Description

Computes the Boolean intersection of a region and a rectangle, returning the result in a new region. The region may actually be an empty region, in which case the bounding rectangle for the region will be an empty rectangle. Note that this routine will compute the intersection faster than calling *MGL_sectRegion* with a simple region as the second region to intersect.

See Also

MGL_sectRegion, *MGL_diffRegion*, *MGL_unionRegion*

MGL_setACCELDriver

Sets the internal VBE/AF accelerator functions driver used by SciTech MGL.

Declaration

```
void MGLAPI MGL_setACCELDriver(  
    void *driver)
```

Prototype In

mgraph.h

Parameters

driver Pointer to the VBE/AF driver for SciTech MGL to use

Description

This function allows the application developer to load a VESA VBE/AF Accelerator Functions driver from disk, and let SciTech MGL know about it so that it will use the pre-loaded version instead of attempting to load the version itself. The main purpose of this function is to allow customers using the SciTech UVBELib/Accel libraries to let SciTech MGL know about the driver generated by these device support libraries and use it instead of the one on the end users system.

MGL_setActivePage

Sets the currently active hardware display page for a display device context.

Declaration

```
void MGLAPI MGL_setActivePage(  
    MGLDC *dc,  
    int page)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context of interest
<i>page</i>	Number of active hardware display page to use

Description

This function sets the currently active hardware video page to which all output from 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_swapBuffers

MGL_setAppInstance

Sets the application instance handle.

Declaration

```
void MGLAPI MGL_setAppInstance(  
    MGL_HINSTANCE hInstApp)
```

Prototype In

mglwin.h

Parameters

hInstApp Handle to the application instance

Description

This function sets the application instance handle for MGL, which is necessary for some internal MGL operations. You must call this function under Windows before you call *MGL_init*, and if you do not MGL will exit with an error message requesting that you do so.

MGL_setAspectRatio

Declaration

```
void MGLAPI MGL_setAspectRatio(  
    int aspectRatio)
```

Prototype In
mgraph.h

Parameters

aspectRatio New value for the aspect ratio

Description

This function sets the aspect ratio for the device context to a new value. This ratio is equal to:

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

The device context aspect ratio can be used to display circles and squares on the 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 code:

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

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

See Also

MGL_getAspectRatio

MGL_setBackColor

Sets the currently active background color.

Declaration

```
void ASMAPI MGL_setBackColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New background color value

Description

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 MGL_BITMAP_OPAQUE fill mode.

Note that the value passed to this routine is either a color index or a color value in the correct packed pixel format for the device context. Use the *MGL_packColor* routine to pack 24 bit RGB values for RGB device contexts.

See Also

MGL_getBackColor, *MGL_setColor*, *MGL_getColor*, *MGL_packColor*

MGL_setBlueCodeIndex

Sets the color index used for the blue code stereo sync mechanism

Declaration

```
void MGLAPI MGL_setBlueCodeIndex(  
    int index)
```

Prototype In

mgraph.h

Description

This function sets the color index used in the blue code stereo sync mechanism. The blue code system as pioneered by StereoGraphics, requires SciTech MGL to draw pure blue sync lines at the bottom of the screen of differing lengths to signal to the LC glasses which is the left and right eye images. In order to do this in 8bpp color index modes, SciTech MGL must take up a single palette entry for drawing the blue codes, which by default is set to index 255. You can use this function to set the blue code index to another value other than 255 to suit your applications palette arrangement.

Note: *You must call this function before you create a stereo display device context if you wish to change the blue code index.*

See Also

*MGL_startStereo, MGL_stopStereo, MGL_createStereoDisplayDC,
MGL_setStereoSyncType*

MGL_setBorderColors

Sets the border colors for the current device context.

Declaration

```
void MGLAPI MGL_setBorderColors(  
    color_t bright,  
    color_t dark)
```

Prototype In

mgraph.h

Parameters

bright
dark

Color for the bright component of the border color
Color for the dark component of the border color

Description

The border colors are the two colors used to draw horizontal, vertical and rectangular borders with the *MGL_drawBorder* function.

MGL_setBufSize

Sets the size of the internal MGL buffer.

Declaration

```
void MGLAPI MGL_setBufSize(  
    unsigned size)
```

Prototype In

mgraph.h

Parameters

size New size of the internal MGL buffer

Description

This function sets the size of the internal MGL scratch buffer, which MGL uses for local scratch space while rasterizing the primitives. The default size of this buffer is 6Kb for 16 bit code and 12Kb for 32 bit code, which is adequate for most needs. If however you attempt to rasterize some primitives and MGL runs out of local storage space you will need to increase the size of this internal buffer.

Note that this routine must be called before *MGL_init* or *MGL_initWindowed* is called for the first time.

See Also

MGL_init, *MGL_initWindowed*

MGL_setClipMode

Sets the clipping mode.

Declaration

```
void MGLAPI MGL_setClipMode(  
    bool mode)
```

Prototype In

mgraph.h

Parameters

mode True for clipping to be turned on, false for no clipping

Description

Sets the current clipping mode. You can selectively turn clipping on and off for 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 rasterizing and perform your own clipping you may want to turn this off for extra performance from MGL.

See Also

MGL_setClipModeDC, *MGL_getClipMode*

MGL_setClipModeDC

Sets the clipping mode for a specific DC.

Declaration

```
void MGLAPI MGL_setClipModeDC(  
    MGLDC *dc,  
    bool mode)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context of the target clipping mode
<i>mode</i>	True for clipping to be turned on, false for no clipping

Description

This function is the same as *MGL_setClipMode*, however the device context does not have to be the current device context.

See Also

MGL_setClipMode, *MGL_getClipMode*

MGL_setClipRect

Sets the current clipping rectangle.

Declaration

```
void MGLAPI MGL_setClipRect(  
    rect_t clip)
```

Prototype In

mgraph.h

Parameters

clip

New clipping rectangle to be used

Description

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 is made to allow clipping outside of the active viewport.

See Also

MGL_setClipRectDC, MGL_getClipRect, MGL_setViewport, MGL_getViewport

MGL_setClipRectDC

Sets the current clipping rectangle for a specific DC.

Declaration

```
void MGLAPI MGL_setClipRectDC(  
    MGLDC *dc,  
    rect_t clip)
```

Prototype In

mgraph.h

Parameters

dc

Display device context in which the rectangle is located .

clip

New clipping rectangle to be used

Description

This function is the same as *MGL_setClipRect*, however the device context does not have to be the current device context.

See Also

MGL_setClipRect, *MGL_getClipRect*, *MGL_setViewport*, *MGL_getViewport*

MGL_setColor

Sets the current foreground color.

Declaration

```
void ASMAPI MGL_setColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New foreground color value

Description

Sets the current foreground color values. The foreground color value is used to draw all primitives.

Note that 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_setColorCI

Sets the current foreground color given a color index.

Declaration

```
void MGLAPI MGL_setColorCI(  
    int index)
```

Prototype In

mgraph.h

Parameters

index Color index of color to set

Description

Sets the current foreground color value given a color index. This routine works with all device contexts, including RGB device contexts. For the color index devices, the value for the foreground color is simply set unchanged. However for RGB devices, the color index is translated via the current color palette for the device to find the appropriate packed MGL color value for that device. Thus you can still write code for RGB devices that works with color indexes.

See Also

MGL_setColor, *MGL_setColorRGB*, *MGL_realColor*

MGL_setColorMapMode

Sets the current color map mode.

Declaration

```
void MGLAPI MGL_setColorMapMode(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode New color map mode to set

Description

This function sets the current color map mode for the device context. Supported modes are enumerated in *MGL_colorModes*.

The current color map mode only affects 8 bit display modes.

See Also

MGL_getColorMapMode

MGL_setColorRGB

Sets the current foreground color given a 24 bit RGB tuple.

Declaration

```
void MGLAPI MGL_setColorRGB(  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>R</i>	Red component of color (0 - 255)
<i>G</i>	Green component of color (0 - 255)
<i>B</i>	Blue component of color (0 - 255)

Description

This function sets the foreground color to a specific 24 bit RGB tuple. If the device context is an RGB device context or an 8 bit device in RGB dithered mode, this value simply sets the proper packed MGL pixel value representing this color (the same as *MGL_packColor* would). However if the device context is a color index device, the color palette is searched for the color value that is the closest to the specified color. This function allows you to specify a color given an RGB tuple, and will work in color index modes as well with any color palette.

See Also

MGL_setColor, *MGL_setColorCI*, *MGL_rgbColor*

MGL_setDefaultPalette

Resets the palette to the default values.

Declaration

```
void MGLAPI MGL_setDefaultPalette(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Description

Sets the palette to the current MGL default values for the device context. This can be used to reset the palette to the original default values that the palette is programmed with when MGL is initialized.

See Also

MGL_getDefaultPalette, *MGL_setPalette*, *MGL_getPalette*

MGL_setDisplayStart

Changes the display start address for virtual scrolling.

Declaration

```
void MGLAPI MGL_setDisplayStart(  
    MGLDC *dc,  
    int x,  
    int y,  
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Scrolling display device context to change
<i>x</i>	New display start x coordinate
<i>y</i>	New display start y coordinate
<i>waitFlag</i>	True if we should wait for the vertical retrace

Description

This function sets the CRTC display starting address for the hardware scrolling device context to the specified (x,y) coordinate. You can use this routine to implement hardware scrolling or panning by moving the display start address coordinates.

The waitVRT flag is used for synchronizing with the vertical retrace and can be one of the following values:

<i>waitVRT</i>	Meaning
<i>0</i>	Set coordinates and update hardware, but do not wait for a vertical retrace when changing the hardware start address.
<i>1</i>	Set coordinates and update hardware, waiting for a vertical retrace during the update for flicker free panning.
<i>-1</i>	Set coordinates but don't update hardware display start.

Passing a waitVRT flag of -1 can be used to implement double buffering and hardware scrolling at the same time. To do this you would call this function first to set the display start x and y coordinates without updating the hardware, and then call *MGL_setVisualPage* to swap display pages and the

new hardware start address will then be programmed.

See Also

MGL_getDisplayStart, MGL_createScrollingDC

Overrides the default file I/O functions used by MGL.

Declaration

```
void MGLAPI MGL_setFileIO(  
    fileio_t *fio)
```

Prototype In

mgraph.h

Parameters

fio

Structure containing new file I/O functions

Description

This function allows the programmer to override the default file I/O functions used by all the MGL functions that access files (bitmap, font, icon and cursor loading). By default the standard C I/O functions are used and you can reset back to the standard C I/O functions by calling this function with the *fio* parameter set to NULL.

This function is useful for creating your own file system, such as storing all the bitmaps, fonts and icons that your application requires in a large file of your own format. This way end users browsing your program's data files will not be able to view any of the data (game developers may wish to keep the bitmaps used for levels in the game secret to make it harder for the user to cheat when playing the game).

This function allows you to overload the *fopen*, *fclose*, *fseek*, *ftell*, *fread* and *fwrite* functions used by MGL. See the *fileio_t* structure for more information.

MGL_setLineStipple

Set the current line stipple pattern.

Declaration

```
void ASMAPI MGL_setLineStipple(  
    ushort stipple)
```

Prototype In

mgraph.h

Parameters

stipple New 16 - bit stipple pattern to set.

Description

Sets the current line stipple pattern. The line stipple pattern is used to determine which pixels in the line get drawn depending on which bits in the pattern are set. The stipple pattern is a 16-bit value, and everywhere that a bit is set to a 1 in the pattern, a pixel will be drawn in the line. Everywhere that a bit is a 0, the pixel will be skipped in the line. Note that bit 0 in the stipple pattern corresponds to pixel 0,16,32,... in the line, bit 1 is pixel 1,17,33 etc. To create a line that is drawn as a 'dot dot dash dash' you would use the following value:

0011100111001001b or 0x39C9

Note that to enable stippled line mode you must call *MGL_setLineStyle*, with the *MGL_LINE_STIPPLE* parameter. Also note that stippled lines can only be 1 pixel wide, and the pen size will be ignored when drawing a stippled line.

See Also

MGL_setLineStyle, *MGL_setLineStippleCount*, *MGL_getLineStipple*

MGL_setLineStippleCount

Declaration

```
void ASMAPI MGL_setLineStippleCount(  
    uint stippleCount)
```

Prototype In
mgraph.h

Parameters

stippleCount New line stipple counter to use

Description

Sets the current line stipple counter to a specific value. The line stipple counter is used to count the number of pixels that have been drawn in the line, and is updated after the line has been drawn. The purpose of this counter is to allow you to draw connected lines using a stipple pattern and the stippling will be continuous across the break in the lines. You can use this function to reset the stipple counter to a known value before drawing lines to force the stipple pattern to start at a specific bit position (usually resetting it to 0 before drawing a group of lines is sufficient).

Note that VBE/AF 1.0 accelerated devices do not support the line stipple counter, so this counter is essentially reset to 0 every time that a line is drawn using the hardware. VBE/AF 2.0 will rectify this problem in the future.

See Also

MGL_setLineStyle, MGL_setLineStipple, MGL_getLineStippleCount

MGL_setLineStyle

Sets the current line style.

Declaration

```
void MGLAPI MGL_setLineStyle(  
    int style)
```

Prototype In

mgraph.h

Parameters

style New line style to use

Description

Sets the current line style. MGL supports two different line styles, either pen style patterned lines (MGL_LINE_PENSTYLE) or stippled lines (MGL_LINE_STIPPLE). Pen style patterned lines are similar to those provided by QuickDraw for the Macintosh where lines are drawing using a rectangular pen that can have an arbitrary size and can be filled with an arbitrary pattern. Pen style patterned lines are the default. Stippled lines are similar to those used by CAD programs on the PC, and are 1-pixel wide lines that can be drawn using a 16-bit stipple mask. Stippled lines can be drawn very fast in hardware using the VBE/AF accelerator drivers.

In stippled line mode the line stipple pattern is used to determine which pixels in the line get drawn depending on which bits in the pattern are set. The stipple pattern is a 16-bit value, and everywhere that a bit is set to a 1 in the pattern, a pixel will be drawn in the line. Everywhere that a bit is a 0, the pixel will be skipped in the line. Note that bit 0 in the stipple pattern corresponds to pixel 0,16,32,... in the line, bit 1 is pixel 1,17,33 etc. To create a line that is drawn as a 'dot dot dash dash' you would use the following value:

```
0011100111001001b or 0x39C9
```

In stippled line mode the line stipple counter is used to count the number of pixels that have been drawn in the line, and is updated after the line has been drawn. The purpose of this counter is to allow you to draw connected lines using a stipple pattern and the stippling will be continuous across the break in the lines. You can use this function to reset the stipple counter to a known value before drawing lines to force the stipple pattern to start at a

specific bit position (usually resetting it to 0 before drawing a group of lines is sufficient).

Note that VBE/AF 1.0 accelerated devices do not support the line stipple counter, so this counter is essentially reset to 0 every time that a line is drawn using the hardware. VBE/AF 2.0 will rectify this problem in the future.

See Also

MGL_setLineStipple, MGL_setLineStippleCount, MGL_getLineStyle

MGL_setMainWindow

Sets the application main window handle.

Declaration

```
void MGLAPI MGL_setMainWindow(  
    MGL_HWND hwnd)
```

Prototype In

mglwin.h

Parameters

hwndMain New main window handle to set

Description

This function informs MGL what your application's main window handle is. If you have a dialog box or main window for your application and you want to start a fullscreen graphics mode under Windows without destroying your main window, you can pass the window handle to MGL using this function so it will automatically ensure that the main application window is removed from the focus chain while running in fullscreen mode. The main purpose of this function is to ensure that your main window cannot become active while running in fullscreen modes, which will cause the system to switch out of the fullscreen mode back to normal GDI mode.

If you don't call this function, you should ensure that when you start a fullscreen graphics mode that your application currently has no windows open on the desktop. If you don't wish to destroy the windows you should call this function, or ensure that the windows which have been hidden are disabled (`ShowWindow(hwnd,FALSE)` and `EnableWindow(false)`) before you start the fullscreen mode.

MGL_setMarkerColor

Sets the current marker color value.

Declaration

```
void MGLAPI MGL_setMarkerColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New marker color to set

Description

Sets the current marker color value. The marker color is used when drawing markers with the *MGL_marker* routine.

Note that 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

Sets the current marker size value.

Declaration

```
void MGLAPI MGL_setMarkerSize(  
    int size)
```

Prototype In

mgraph.h

Parameters

<i>size</i>	New marker size
--------------------	-----------------

Description

Sets the current marker size. The marker size is used to determine how big to draw the markers 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. A 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

Sets the current marker style.

Declaration

```
void MGLAPI MGL_setMarkerStyle(  
    int style)
```

Prototype In

mgraph.h

Parameters

<i>style</i>	New marker style value
---------------------	------------------------

Description

Sets the current marker style value. The marker style defines the type of marker to be rasterized. MGL Marker styles are defined in *MGL_markerStyleType*.

See Also

MGL_getMarkerStyle, *MGL_marker*, *MGL_polyMarker*

MGL_setOpenGLFuncs

Set the OpenGL rendering functions table pointer

Declaration

```
void MGLAPI MGL_setOpenGLFuncs(  
    MGL_glFuncs *glFuncs)
```

Prototype In

gl\gl.h

Parameters

glFuncs Pointer to MGL_glFuncs structure to fill in

Description

This function is called by the user application if the MGL libraries are stored in a DLL. By letting SciTech MGL know about the OpenGL function pointer table, when the OpenGL implementation is swapped by SciTech MGL it automatically updates the table in the user DLL to point to the newly loaded OpenGL entry points. This allows the code in the DLL to run with maximum performance for calls to OpenGL via function pointers.

MGL_setPalette

Sets the palette values for a device context.

Declaration

```
void MGLAPI MGL_setPalette(  
    MGLDC *dc,  
    palette_t *pal,  
    int numColors,  
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette values for
<i>pal</i>	Pointer to array of palette values to set
<i>numColors</i>	Number of color values to set
<i>startIndex</i>	Starting index of first color value to set

Description

This function sets part or all of the color palette for the device context. You can specify only a subset of the palette values to be modified with the `startIndex` and `numColors` arguments. Thus:

```
MGL_setPalette(dc, pal, 10, 50);
```

will program the 10 color indices from 50-60 with the values stored in the palette buffer 'pal'.

Note: *This routine does not actually change the value of the hardware palette. If you wish to change the hardware palette to reflect the new values, you will need to call the MGL_realizePalette function to update the hardware palette.*

Note: *You must ensure that you do not attempt to program invalid color indices! Use MGL_maxColor() to find the largest color index in color index modes.*

Note: *This function is also valid for RGB device contexts, and will simply set the color translation tables for these devices (used for drawing color index bitmaps and translating color index color values to RGB values).*

See Also

MGL_getPalette, MGL_setPaletteEntry, MGL_realizePalette

MGL_setPaletteEntry

Sets a single palette entry.

Declaration

```
void MGLAPI MGL_setPaletteEntry(  
    MGLDC *dc,  
    int entry,  
    uchar red,  
    uchar green,  
    uchar blue)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette entry in
<i>entry</i>	Palette index to program
<i>red</i>	Red component for palette entry
<i>green</i>	Green component for palette entry
<i>blue</i>	Blue component for palette entry

Description

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. Note that this routine does not actually change the value of the hardware palette, and if you wish to change the hardware palette to reflect the new values, you will need to call the *MGL_realizePalette* function to update the hardware palette.

This function is also valid for RGB device contexts, and will simply set the color translation tables for these devices (used for drawing color index bitmaps and translating color index color values to RGB values).

See Also

MGL_getPaletteEntry, *MGL_setPalette*, *MGL_getPalette*, *MGL_realizePalette*

MGL_setPaletteSnowLevel

Sets the current palette snow level for a display device context.

Declaration

```
void MGLAPI MGL_setPaletteSnowLevel(  
    MGLDC *dc,  
    int level)
```

Prototype In

mgraph.h

Parameters

dc
level

Display device context to set snow level for
New snow level to set

Description

This function sets the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. By default MGL programs all 256 entries per retrace, but you may need to slow this down on systems with slower hardware that causes snow during multiple palette realization commands.

See Also

MGL_getPaletteSnowLevel, *MGL_setPalette*

MGL_setPenBitmapPattern

Sets the currently active bitmap pattern.

Declaration

```
void ASMAPI MGL_setPenBitmapPattern(  
    const pattern_t *pat)
```

Prototype In

mgraph.h

Parameters

pat New bitmap pattern to use

Description

This function sets the currently active bitmap pattern used when rasterizing patterned primitive in the MGL_BITMAP_ TRANSPARENT and MGL_BITMAP_ OPQAUE pen styles. A bitmap pattern is defined as an 8 x 8 pixel monochrome pattern stored as an array of 8 bytes.

When filling in the MGL_BITMAP_ 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 MGL_BITMAP_ 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_setPenPixmapPattern, MGL_setPenStyle

MGL_setPenPixmapPattern

Sets the currently active pixmap pattern.

Declaration

```
void ASMAPI MGL_setPenPixmapPattern(  
    const pixpattern_t *pat)
```

Prototype In

mgraph.h

Parameters

pat New pixmap pattern to use

Description

This function sets the currently active pixmap pattern used when rasterizing patterned primitive in the MGL_PIXMAP pen style. A pixmap pattern is defined as an 8 x 8 pixel color pattern stored as an 8 x 8 array of MGL color values. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_getPenPixmapPattern, *MGL_setPenBitmapPattern*, *MGL_setPenStyle*

MGL_setPenSize

Declaration

```
void MGLAPI MGL_setPenSize(  
    int h,  
    int w)
```

Prototype In

mgraph.h

Parameters

<i>h</i>	Height of the pen in pixels
<i>w</i>	Width of the pen in pixels

Description

Sets the size of the current pen size in pixels. The default pen is 1 pixel by 1 pixel in dimensions, however you can change this to whatever value you like. When primitives are rasterized 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

Sets the current pen style.

Declaration

```
void ASMAPI MGL_setPenStyle(  
    int style)
```

Prototype In

mgraph.h

Parameters

style New pen style to use

Description

Returns the currently active pen style. Pen styles supported by the SciTech MGL are enumerated in *MGL_penStyleType*.

When filling in the MGL_BITMAP_ 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 MGL_BITMAP_ OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_getPenStyle, *MGL_setPenBitmapPattern*

MGL_setPolygonType

Sets the current polygon type.

Declaration

```
void MGLAPI MGL_setPolygonType(  
    int type)
```

Prototype In

mgraph.h

Parameters

<i>type</i>	New polygon type
--------------------	------------------

Description

Sets the current polygon type. You can change this value to force MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). Polygon types supported by the SciTech MGL are enumerated in *MGL_polygonType*.

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rasterizing. Note that this setting does not affect the specialized triangle and quadrilateral rasterizing routines.

See Also

MGL_getPolygonType, *MGL_fillPolygon*

MGL_setRelViewport

Sets a viewport relative to the current viewport.

Declaration

```
void MGLAPI MGL_setRelViewport(  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

view Bounding rectangle for the new viewport

Description

Sets the current viewport to the viewport specified by *view*, relative to the currently active viewport. The new viewport is restricted to fall within the bounds of the currently active viewport. Note that when the viewport is changing, the viewport origin is always reset back to (0,0).

All output in 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_setViewportOrg*

MGL_setRelViewportDC

Sets a viewport relative to the current viewport for a specific DC.

Declaration

```
void MGLAPI MGL_setRelViewportDC(  
    MGLDC *dc,  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	Bounding rectangle for the new viewport

Description

This function is the same as *MGL_setRelViewport*, however the device context does not have to be the current device context.

See Also

MGL_setRelViewport, *MGL_getViewport*, *MGL_setViewport*,
MGL_clearViewport, *MGL_setClipRect*, *MGL_setViewportOrg*

MGL_setResult

Sets the internal MGL result flag.

Declaration

```
void MGLAPI MGL_setResult(  
    int result)
```

Prototype In

mgraph.h

Return Value

New internal result flag

Description

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 MGL that will return result codes in the same manner as MGL.

See Also

MGL_result, *MGL_errorType*

MGL_setSpaceExtra

Sets the current space extra value.

Declaration

```
void MGLAPI MGL_setSpaceExtra(  
    int extra)
```

Prototype In

mgraph.h

Parameters

extra New space extra value

Description

Sets 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. When this value is positive, it will insert extra space between the characters in a font and making this value negative will make the characters run on top of each other.

See Also

MGL_getSpaceExtra, *MGL_drawStr*

MGL_setStereoSyncType

Sets the stereo sync type for LC shutter glasses

Declaration

```
void MGLAPI MGL_setStereoSyncType(  
    int type)
```

Prototype In

mgraph.h

Description

This function sets the stereo sync type that SciTech MGL uses to communicate with the LC shutter glasses, letting them know when the left and right eye images are being displayed on the screen. By default SciTech MGL starts in MGL_STEREO_AUTO mode and if there is no hardware stereo sync available it will enable all supported sync mechanisms at the same time. This provides the simplest and most powerful support, since the user can plug in any of the supported LC glasses and correctly view and image. However you may wish to force the value to one of the supported mechanisms listed in the *MGL_stereoSyncType* enumeration.

Hardware stereo sync is supported by many new graphics controllers, and requires a physical connector from the back of the graphics adapter (usually a VESA MINI DIN-3 connector or VESA EVC connector) that passes the sync information directly from the graphics chip to the LC shutter glasses. Many older graphics adapters don't support this and SciTech MGL will use a software sync mechanism on these adapters. If the user has problems with the hardware stereo sync (such as if they have older glasses that don't support this), you can disable this and force software syncing with the MGL_STEREO_IGNORE_HARDWARE flag.

Note: *You may call this function while a stereo image is being displayed to change the stereo sync mechanism on the fly.*

See Also

MGL_startStereo, MGL_stopStereo, MGL_createStereoDisplayDC, MGL_setBlueCodeIndex

MGL_setSuspendAppCallback

Sets the fullscreen suspend application callback function.

Declaration

```
void MGLAPI MGL_setSuspendAppCallback(  
    MGL_suspend_cb_t saveState)  
typedef int (ASMAPI *MGL_suspend_cb_t)(MGLDC *dc, int  
flags)
```

Prototype In

mgraph.h

Parameters

saveState New suspend app callback to be used.

Description

This function is used to register an application suspend callback function. This is used in fullscreen modes under Windows and is called by MGL when the application's fullscreen window has lost the input focus and the system has returned to the normal GDI desktop. The focus can be lost due to the user hitting a System Key combination such as Alt-Tab or Ctrl-Esc which forces your fullscreen application into the background. MGL takes care of all the important details such as saving and restoring the state of the hardware, so all your suspend application callback needs to do is save the current state of your program so that when the request is made to re-activate your application, you can redraw the screen and continue from where you left off.

When MGL detects that your application has been suspended it will call the registered callback with a combination of the following flags:

<i>Flag</i>	Meaning
<i>MGL_DEACTIVATE</i>	This flag will be sent when your application has lost the input focus and has been suspended.
<i>MGL_REACTIVATE</i>	This flag will be sent when the user re-activates your fullscreen application again indicating that the fullscreen mode has now been restored and the application must redraw the display ready to continue on.

By default if you have not installed a suspend callback handler, MGL will simply restore the display to the original state with the screen cleared to black when the application is re-activated. If your application is a game or animation that is continuously updating the screen, you won't need to do anything as the next frame in the animation will re-draw the screen correctly. If however your application is caching bitmaps in offscreen video memory, all of the bitmaps will need to be restored to the offscreen display device context when the application is restored (the offscreen memory will be cleared to black also).

Note: *By the time your callback is called, the display memory may have already been lost under DirectDraw so you cannot save and restore the contents of the display memory, but must be prepared to redraw the entire display when the callback is called with the MGL_REACTIVATE flag set.*

MGL_setTextDirection

Sets the current text direction.

Declaration

```
void MGLAPI MGL_setTextDirection(  
    int direction)
```

Prototype In

mgraph.h

Parameters

direction New text direction value

Description

Sets the current text direction. Directions supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_getTextDirection, *MGL_drawStr*

MGL_setTextJustify

Sets the current text horizontal and vertical justification.

Declaration

```
void MGLAPI MGL_setTextJustify(  
    int horiz,  
    int vert)
```

Prototype In

mgraph.h

Parameters

horiz
vert

New horizontal text justification value
New vertical text justification value

Description

Sets the current text justification values. Horizontal and vertical justification type supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_getTextJustify

MGL_setTextSettings

Restores the current text settings.

Declaration

```
void MGLAPI MGL_setTextSettings(  
    text_settings_t *settings)
```

Prototype In

mgraph.h

Parameters

settings Text settings to restore

Description

Restores a set of previously saved text settings. This routine provides a way to save and restore all the values relating to the rasterizing of text in MGL with a single function call.

See Also

MGL_getTextSettings

MGL_setTextSize

Sets the current text scaling factors

Declaration

```
void MGLAPI 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

Description

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

$$\text{scaled} = \frac{\text{unscaled} * \text{numer}}{\text{denom}}$$

Note: *MGL can only scale vectored fonts.*

See Also

MGL_getTextSize

MGL_setUserEventFilter

Installs a user supplied event filter callback for DOS event handling.

Declaration

```
void MGLAPI MGL_setUserEventFilter(  
    bool (*userEventFilter)(event_t *evt))
```

Prototype In

mgldos.h

Parameters

userEventFilter Address of user supplied event filter callback

Description

This function allows the application programmer to install an event filter callback for DOS event handling. Once you install your callback, the MGL event handling routines will call your callback with a pointer to the new event that will be placed into the event queue. Your callback can the modify the contents of the event before it is placed into the queue (for instance adding custom information or perhaps high precision timing information).

If your callback returns FALSE, the event will be ignore and will not be posted to the event queue. You should always return true from your event callback unless you plan to use the events immediately that they are recieved.

Note: *Your event callback will be called in response to a hardware interrupt and will be executing in the context of the hardware interrupt handler (i.e.: keyboard interrupt, mouse interrupt or timer interrupt). For this reason SciTech MGL automatically locks down the code pages for the callback that you register with the PM_lockCodePages function. If your filter callback calls any other functions, either place those function immediately **after** your filter callback, or explicitly call PM_lockCodePages for all the functions that might be called from your handler. You should also ensure that your filter callback runs as fast as possible to ensure that the system is ready to recieve the next event function.*

Note: *You can also use this filter callback to process events at the time they are activated by the user (i.e.: when the user hits the key or moves the mouse), but make sure your code runs as fast as possible as it will be executing inside*

the context of an interrupt handler.

See Also

EVT_getNext, EVT_peekNext

MGL_setViewport

Sets the currently active viewport.

Declaration

```
void MGLAPI MGL_setViewport(  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

view New global viewport bounding rectangle

Description

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. Note that when the viewport is changing, the viewport origin is always reset back to (0,0).

All output in 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_setViewportOrg*

MGL_setViewportDC

Sets the currently active viewport for a specific DC.

Declaration

```
void MGLAPI MGL_setViewportDC(  
    MGLDC *dc,  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	New global viewport bounding rectangle

Description

This function is the same as *MGL_setViewport*, however the device context does not have to be the current device context.

See Also

MGL_setViewport, *MGL_getViewport*, *MGL_setRelViewport*,
MGL_clearViewport, *MGL_setClipRect*, *MGL_setViewportOrg*

MGL_setViewportOrg

Sets the logical viewport origin.

Declaration

```
void MGLAPI MGL_setViewportOrg(  
    point_t org)
```

Prototype In

mgraph.h

Parameters

org New logical viewport origin.

Description

This function sets the currently active viewport origin. When a new viewport is set with the *MGL_setViewport* function, the viewport origin is reset to (0,0), which means that any primitives drawn at pixel location (0,0) will appear at the top left hand corner of the viewport.

You can change the logical coordinate of the viewport origin to any value you please, which will effectively offset all drawing within the currently active viewport. Hence if you set the viewport origin to (10,10), drawing a pixel at (10,10) would make it appear at the top left hand corner of the viewport.

See Also

MGL_setViewportOrgDC, *MGL_getViewportOrg*, *MGL_setViewport*

MGL_setViewportOrgDC

Sets the logical viewport origin for a specific DC.

Declaration

```
void MGLAPI MGL_setViewportOrgDC(  
    MGLDC *dc,  
    point_t org)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>org</i>	New logical viewport origin.

Description

This function is the same as *MGL_setViewportOrg*, however the device context does not have to be the current device context.

See Also

MGL_setViewportOrg, *MGL_getViewportOrg*, *MGL_setViewport*

MGL_setVisualPage

Sets the currently visible hardware video page for a display device context.

Declaration

```
void MGLAPI MGL_setVisualPage(  
    MGLDC *dc,  
    int page,  
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context
<i>page</i>	New hardware display page
<i>waitVRT</i>	True if we should sync to the vertical retrace

Description

This function sets the currently visible hardware video page for a display device context. 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.

When the visible display page is changed, you should normally allow MGL to sync to the vertical retrace to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display. You may however turn off the vertical retrace synching if you are synching up with the retrace period using some other means.

Note that if you wish to implement both double buffering and hardware scrolling or panning, you should call the *MGL_setDisplayStart* function first with waitVRT set to -1, and then call this function with waitVRT set to true to actually update the hardware. The first call to *MGL_setDisplayStart* simply updates the internal display start variables but does not program the hardware. For more information please see the *MGL_setDisplayStart* function.

See Also

MGL_getVisualPage, *MGL_getActivePage*, *MGL_setActivePage*,
MGL_swapBuffers, *MGL_setDisplayStart*

MGL_setWinDC

Associate a window manager device context with an MGL device context.

Declaration

```
bool MGLAPI MGL_setWinDC(  
    MGLDC *dc,  
    MGL_HDC hdc)
```

Prototype In

mglwin.h

Parameters

<i>dc</i>	MGL windowed device context to use
<i>hdc</i>	Handle to window manager device context to associate

Return Value

True if the application's palette has changed, false if not.

Description

This function is used to tell MGL what window manager device context should actually be used for all subsequent BitBlt calls to transfer data from memory device contexts to the windowed device context. Because the window manager device contexts are usually allocated on the fly for every window repaint operation, this function is usually called immediately after the window gets a repaint event and before the windowed DC is used to transfer the memory device context data to the window. A typical Windows WM_PAINT handler would be coded as follows:

```
case WM_PAINT:  
    hdc = BeginPaint(hwnd,&ps);  
    MGL_setWinDC(winDC,hdc);  
    // Do rasterizing code in here //  
    MGL_bitBlt(winDC,memDC,r,0,0,MGL_REPLACE_MODE);  
    EndPaint(hwnd,&ps);  
    return 0;
```

See Also

MGL_activatePalette

MGL_setWriteMode

Sets the current write mode operation.

Declaration

```
void ASMAPI MGL_setWriteMode(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode New write mode operation to use

Description

Sets the currently active write mode. Write mode operations supported by the SciTech MGL for all output primitives are enumerated in *MGL_writeModeType*.

See Also

MGL_getWriteMode

MGL_singleBuffer

Returns the display device context single buffered mode.

Declaration

```
void MGLAPI MGL_singleBuffer(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Description

This function puts the display device context 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

Returns the total device x coordinate dimensions.

Declaration

```
int MGLAPI MGL_sizeX(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Return Value

Number of pixels in x direction for entire device - 1

Description

Returns the total number of pixels available along the x coordinate axis for the currently active device context. This is different than the *MGL_maxx* routine which returns the dimensions of the currently active viewport.

See Also

MGL_sizeY, *MGL_maxx*, *MGL_maxy*

MGL_sizey

Returns the total device y coordinate dimensions.

Declaration

```
int MGLAPI MGL_sizey(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Return Value

Number of pixels in y direction for entire device - 1

Description

Returns the total number of pixels available along the y coordinate axis for the currently active device context. This is different than the *MGL_maxy* routine which returns the dimensions of the currently active viewport.

See Also

MGL_sizex, *MGL_maxx*

MGL_srand

Reseed MGL random number generator.

Declaration

```
void ASMAPI MGL_srand(  
    uint seed)
```

Prototype In

mgraph.h

Parameters

seed New seed value for the random number generator

Description

This function reseeds the random number generator to start generating a new sequence of numbers. Generally this function is used to randomize the generator by seeding it with the value obtained from the *MGL_getTicks* function.

See Also

MGL_random, *MGL_randoml*

MGL_startStereo

Enables free running stereo display mode

Declaration

```
void MGLAPI MGL_startStereo(MGLDC *dc)
```

Prototype In

mgraph.h

Description

This function enables the free running stereo display mode for LC shutter glasses. This function only works if the display device context you created was a stereo display device context created with *MGL_createStereoDisplayDC*. By default free running stereo mode is on when you create the stereo display device context.

See Also

MGL_stopStereo, *MGL_createStereoDisplayDC*

MGL_stopStereo

Disables free running stereo display mode

Declaration

```
void MGLAPI MGL_stopStereo(MGLDC *dc)
```

Prototype In

mgraph.h

Description

This function disables the free running stereo display mode for LC shutter glasses. This function only works if the display device context you created was a stereo display device context created with *MGL_createStereoDisplayDC*. By default free running stereo mode is on when you create the stereo display device context, and you can use this function to disable it in parts of your application that don't require stereo (such as when navigating the menu system etc). Note that when stereo mode is disabled, SciTech MGL always displays from the left eye buffer.

See Also

MGL_startStereo, *MGL_createStereoDisplayDC*

MGL_stretchBLT

Stretches a block of image data from one device context to another.

Declaration

```
void MGL_stretchBLT(  
    MGLDC dst,  
    MGLDC src,  
    rect_t srcRect,  
    rect_t destRect)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstRect</i>	Rectangle defining destination image

Description

This function is the same as *MGL_stretchBltCoord*, however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_stretchBltCoord, *MGL_bitBlt*, *MGL_bitBltCoord*

MGL_stretchBitmap

Stretches a lightweight bitmap to the specified rectangle.

Declaration

```
void MGLAPI MGL_stretchBitmap(  
    MGLDC *dc,  
    int dstLeft,  
    int dstTop,  
    int dstRight,  
    int dstBottom,  
    const bitmap_t *bitmap)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display

Description

Stretches a lightweight bitmap to the specified destination rectangle on the device context. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

Note that for maximum performance when displaying 8 bit bitmaps, you should ensure that the color palette for the bitmap is identical to the device context, otherwise the pixels in the bitmap will be translated during the draw operation.

See Also

MGL_loadBitmap

MGL_stretchBitmapSection

Stretches a section of a lightweight bitmap to the specified device context.

Declaration

```
void MGLAPI MGL_stretchBitmapSection(  
    MGLDC *dc,  
    int dstLeft,  
    int dstTop,  
    int dstRight,  
    int dstBottom,  
    const bitmap_t *bitmap)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to stretch
<i>top</i>	Top coordinate of section to stretch
<i>right</i>	Right coordinate of section to stretch
<i>bottom</i>	Bottom coordinate of section to stretch
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display

Description

Stretches a section of a lightweight bitmap to the specified destination rectangle on the device context. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

Note: *For maximum performance when displaying 8 bit bitmaps, you should ensure that the color palette for the bitmap is identical to the device context, otherwise the pixels in the bitmap will be translated during the draw operation.*

See Also

MGL_loadBitmap

Declaration

```
void MGLAPI MGL_stretchBltCoord(  
    MGLDC *dst,  
    MGLDC *src,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int dstLeft,  
    int dstTop,  
    int dstRight,  
    int dstBottom)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination image
<i>dstTop</i>	Top coordinate of destination image
<i>dstRight</i>	Right coordinate of destination image
<i>dstBottom</i>	Bottom coordinate of destination image

Description

Copies a block of bitmap data form one device context to another, stretching or shrinking the image as necessary to fit the destination rectangle for the destination device context.

The source and destination device context may not be the same. This routine has been highly optimized for absolute maximum performance, so it will provide the fastest method of stretching bitmap data between device contexts, and can also be used to stretch bitmap data from a memory device context to a windowed device context.

This function will correctly handle StretchBlts across device contexts with differing pixel depths, and will perform the necessary pixel format

translation to convert from the source device to the destination device. Note that although the code to implement this is highly optimized, this can be a time consuming operation so you should attempt to pre-convert all bitmaps to the current display device pixel format for maximum performance if possible.

MGL does however have special case code to specifically handle translation of 24 bit RGB format bitmaps (the standard RGB DIB format used by Video for Windows) to all 8 bit and above pixel formats. When converting from 24 bit to 8 bit, MGL will dither bitmaps in real time from 24 bit to the 8 bit halftone palette. This provides a solid foundation to build real time 24 bit motion video playback in all supported video modes in MGL.

Note that when *MGL_bitBlt* is called for 4 and 8 bit source bitmaps MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for Blting bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device, or you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively, however the zoom factor is determined using the unclipped source and destination rectangles.

See Also

MGL_stretchBlt, *MGL_bitBlt*, *MGL_bitBltCoord*

MGL_surfaceAccessType

Return the direct surface access flags.

Declaration

```
int MGLAPI MGL_surfaceAccessType(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Direct surface access flags for the device context.

Description

This function returns the direct surface access flags for the specified device context, which can be used to determine if the surface for the device context is directly accessible, and if the surface has been virtualized in software. The access flags returned are enumerated in *MGL_surfaceAccessFlagsType*.

If the surface access flags is *MGL_VIRTUAL_ACCESS*, this means that the surface for the device can be directly accessed, however the surface is actually virtualized in software using a page fault handler for SuperVGA devices that do not have a real hardware linear framebuffer. If the surface is virtualized, you must ensure that when you directly access the surface you do so on BYTE, WORD and DWORD aligned boundaries. If you access it on a non-aligned boundary across a page fault, you will cause an infinite page fault loop to occur. If the surface access flags is *MGL_NO_ACCESS* the framebuffer will be banked and if you wish to rasterize directly to it you will need to use the *SVGA_setBank* functions to change banks. In banked modes the surface pointer points to the start of the banked framebuffer window (i.e.: 0xA0000).

MGL_suspend

Suspend low level interrupt handling.

Declaration

```
void MGLAPI MGL_suspend(void)
```

Prototype In

mgldos.h

Description

This function suspends the low level interrupt handling code used by the SciTech MGL when it is initialized since MGL takes over the keyboard and mouse interrupt handlers to manage it's own event queue. If you wish to shell out to DOS or to spawn another application program temporarily, you must call this function to suspend interrupt handling or else the spawned application will not be able to access the keyboard and mouse correctly.

See Also

MGL_resume

MGL_swapBuffers

Swaps the currently active front and back buffers for a display device context.

Declaration

```
void MGLAPI MGL_swapBuffers(  
    MGLDC *dc,  
    int waitVRT)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	Display device context
<i>waitVRT</i>	Should we wait for the vertical retrace?

Description

This function swaps the currently active front and back buffers. This routine should only be called after the *MGL_doubleBuffer* has been called to initialize the double buffering for the device context. Once double buffering has been set up, all output from MGL will go to the current offscreen buffer, and the output can be made visible by calling this routine. This routine is the standard technique used to achieve smooth animation.

When the visible display buffer is changed, you should normally allow MGL to sync to the vertical retrace to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display. You may however turn off the vertical retrace synching if you are synching up with the retrace period using some other means.

See Also

MGL_doubleBuffer, *MGL_singleBuffer*

Compute the bounding box for a text string.

Declaration

```
void MGLAPI MGL_textBounds(  
    int x,  
    int y,  
    const char *str,  
    rect_t *bounds)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate string would be drawn at
<i>y</i>	y coordinate string would be drawn at
<i>str</i>	String to measure
<i>bounds</i>	Place to store the computed bounds

Description

This function computes the bounding box that fits tightly around a text string drawn at a specified location on the current device context. This routine correctly computes the bounding rectangle for the string given the current text justification, size and direction settings.

See Also

MGL_textHeight, *MGL_textWidth*

MGL_textHeight

Returns the height of the current font in pixels.

Declaration

```
int MGLAPI MGL_textHeight(void)
```

Prototype In

mgraph.h

Return Value

Height of the current font in pixels

Description

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.

See Also

MGL_textWidth, MGL_drawStr, MGL_getCharMetrics, MGL_getFontMetrics

MGL_textWidth

Returns the width of the character string in pixels.

Declaration

```
int MGLAPI MGL_textWidth(  
    const char *str)
```

Prototype In

mgraph.h

Parameters

str Character string to measure

Return Value

Width of the character string in pixels

Description

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.

See Also

MGL_textHeight, *MGL_drawStr*, *MGL_getCharMetrics*, *MGL_getFontMetrics*

Copies a block of image data with source transparency.

Declaration

```
void MGL_transBlt(  
    MGLDC *dst,  
    MGLDC *src,  
    rect_t srcRect,  
    int dstLeft,  
    int dstTop,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

This function is the same as *MGL_transBltCoord*, however it takes a rectangle as a parameter instead of the four coordinates of a rectangle.

See Also

MGL_transBltCoord, *MGL_bitBlt*

Copies a block of image data with source transparency.

Declaration

```
void MGLAPI MGL_transBltCoord(  
    MGLDC *dst,  
    MGLDC *src,  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int dstLeft,  
    int dstTop,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

Copies a block of bitmap data from one device context to another with either source or destination transparency. When transferring the data with source transparency, pixels in the destination image that are equal to the specified transparent color will not be updated, and those pixels that are not the same will be updated. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and

those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

The device contexts must not be the same and must be in the same pixel depth and the same pixel format, or this routine will simply do nothing. This routine also only works for device contexts with pixel depths that are at least 8 bits deep.

This routine has been highly optimized for maximum performance in all pixel depths, so will provide a very fast method for performing transparent sprite animation. However you may find that if you can use alternative techniques to pre- compile the sprites (like using run length encoding etc.) you will be able to build faster software based sprite animation code that can directly access the device context surface. However this routine can also be used to perform hardware accelerated Blts between offscreen memory device's and the display device when running in fullscreen modes, providing the hardware accelerator (if present) can support this operation. If you have a hardware accelerator capable of this, this will provide the ultimate performance for transparent sprite animation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

See Also

MGL_transBlt, MGL_bitBlt

MGL_transBltLin

Copies a block of image data with source transparency.

Declaration

```
void MGL_transBltLin(  
    MGLDC dst,  
    MGLDC src,  
    ulong srcOfs,  
    rect_t dstRect,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context (must be a display device)
<i>src</i>	Source device context (must be a linear offscreen device)
<i>srcOfs</i>	Starting offset of bitmap in the source device surface
<i>dstRect</i>	Rectangle defining source image
<i>transparent</i>	Transparent color to skip in source image
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

This function is the same as *MGL_transBltLin*, however it takes the entire rectangle as a parameter instead of the individual coordinates.

See Also

MGL_transBltLinCoord, *MGL_bitBltLin*

MGL_transBltLinCoord

Copies a block of image data with source transparency.

Declaration

```
void MGLAPI MGL_transBltLinCoord(  
    MGLDC *dst,  
    MGLDC *src,  
    ulong srcOfs,  
    int dstLeft,  
    int dstTop,  
    int dstRight,  
    int dstBottom,  
    color_t transparent,  
    bool sourceTrans)
```

Prototype In
mgraph.h

Parameters

<i>dst</i>	Destination device context (must be a display device)
<i>src</i>	Source device context (must be a linear offscreen device)
<i>srcOfs</i>	Starting offset of bitmap in the source device surface
<i>dstLeft</i>	Left coordinate of destination image
<i>dstTop</i>	Top coordinate of destination image
<i>dstRight</i>	Right coordinate of destination image
<i>dstBottom</i>	Bottom coordinate of destination image
<i>transparent</i>	Transparent color to skip in source image
<i>sourceTrans</i>	True for source transparency, false for destination transparency

Description

This function is similar to *MGL_transBlt* except that the source device context must be for a linear offscreen device and the destination device context must be for a display device. The difference is that this function copies a bitmap as a linear chunk of memory from the offscreen memory to the display memory using the hardware accelerator. Hence you can store bitmaps in the offscreen memory device contiguously, which provides for much more efficient utilization of the hardware video memory for storing sprites and bitmaps.

Note however that not all hardware accelerators can support linear offscreen memory, in which case you would not be able to create a linear offscreen device context. Also some hardware accelerators cannot support transparent BitBlt operations.

The destination rectangle is clipped according to the current clipping rectangles for the destination device context.

See Also

MGL_transBltLin, *MGL_bitBltLin*

MGL_traverseRegion

Traverses a region for all rectangles in definition.

Declaration

```
void MGLAPI MGL_traverseRegion(  
    region_t *rgn,  
    rgncallback_t doRect)  
typedef void (ASMAPI *rgncallback_t)(const rect_t *r)
```

Prototype In

mgraph.h

Parameters

<i>rgn</i>	Region to traverse
<i>doRect</i>	Callback function to call for every rectangle processed

Description

This function traverses the definition of the region, calling the supplied callback function once for every rectangle in union of rectangles that make up the complex region.

See Also

MGL_diffRegion, *MGL_unionRegion*, *MGL_sectRegion*

MGL_underScoreLocation

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

Declaration

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

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to be passed to <i>MGL_drawStrXY</i>
<i>y</i>	y coordinate to be passed to <i>MGL_drawStrXY</i>
<i>str</i>	String to measure

Description

This function takes an (x,y) location that would normally 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_unionRect

Computes the union of two rectangles.

Declaration

```
void MGLAPI MGL_unionRect(  
    rect_t r1,  
    rect_t r2,  
    rect_t *d)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	First rectangle to compute union of
<i>r2</i>	Second rectangle to compute union of
<i>d</i>	Place to store resulting union rectangle

Description

This function computes the union of two rectangles, and stores the result in a third rectangle.

See Also

MGL_unionRectCoord, *MGL_sectRect*

MGL_unionRectCoord

Computes the union of two rectangles.

Declaration

```
void MGLAPI MGL_unionRectCoord(  
    int left1,  
    int top1,  
    int right1,  
    int bottom1,  
    int left2,  
    int top2,  
    int right2,  
    int bottom2,  
    rect_t *d)
```

Prototype In

mgraph.h

Parameters

<i>left1</i>	Left coordinate of first rectangle to compute union of
<i>top1</i>	Top coordinate of first rectangle to compute union of
<i>right1</i>	Right coordinate of first rectangle to compute union of
<i>bottom1</i>	Bottom coordinate of first rectangle to compute union of
<i>left2</i>	Left coordinate of second rectangle to compute union of
<i>top2</i>	Top coordinate of second rectangle to compute union of
<i>right2</i>	Right coordinate of second rectangle to compute union of
<i>bottom2</i>	Bottom coordinate of second rectangle to compute union of
<i>d</i>	Place to store resulting union rectangle

Description

This function computes the union of two rectangles, and stores the result in a third rectangle.

See Also

MGL_sectRect

MGL_unionRegion

Computes the Boolean union of two regions.

Declaration

```
bool MGLAPI MGL_unionRegion(  
    region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	Region with which r2 is unioned, and also becomes the result region.
<i>r2</i>	Region to be unioned with r1

Return Value

True if the union is valid, false if an empty region was created.

Description

Computes the Boolean union of two regions for the area covered by region r1 and region r2, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

See Also

MGL_diffRegion, *MGL_sectRegion*, *MGL_unionRegionOfs*,
MGL_unionRegionRect

MGL_unionRegionOfs

Computes the Boolean union of two regions and offsets the result.

Declaration

```
bool MGLAPI MGL_unionRegionOfs(  
    region_t *r1,  
    const region_t *r2,  
    int xOffset,  
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	Region with which r2 is unioned, and also becomes the result region.
<i>r2</i>	Region to be unioned with r1
<i>xOffset</i>	Offset to add to all x coordinates in region 2
<i>yOffset</i>	Offset to add to all y coordinates in region 2

Return Value

True if the union is valid, false if an empty region was created.

Description

Computes the Boolean union of two regions for the area covered by region r1 and region r2, computing the resulting region in r1, which may result in an empty region. This routine also adds the specified (x,y) offset value to all the coordinates in region 2 before they are unioned with region 1, which allows you to quickly do a union with a translated region without needing to explicitly translate the region itself. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

See Also

MGL_diffRegion, MGL_sectRegion, MGL_unionRegion, MGL_unionRegionRect

MGL_unionRegionRect

Computes the Boolean union of a region and a rectangle.

Declaration

```
bool MGLAPI MGL_unionRegionRect(  
    region_t *r1,  
    const rect_t *r2)
```

Prototype In

mgraph.h

Parameters

<i>r1</i>	Region with which r2 is unioned, and also becomes the result region.
<i>r2</i>	Rectangle to be unioned with r1

Return Value

True if the union is valid, false if an empty region was created.

Description

Computes the Boolean union of a region *r1* and rectangle *r2*, computing the resulting region in *r1*, which may result in an empty region. If you need to retain the value of *r1*, you need to first copy *r1* to a temporary region.

This routine is faster than using *MGL_unionRegion* if the region to be unioned is a simple rectangle rather than a complex region.

See Also

MGL_diffRegion, *MGL_sectRegion*, *MGL_unionRegion*, *MGL_unionRegionOfs*

MGL_unloadBitmap

Unloads a bitmap file from memory.

Declaration

```
void MGLAPI MGL_unloadBitmap(  
    bitmap_t *bitmap)
```

Prototype In

mgraph.h

Parameters

bitmap Pointer to bitmap to unload

Description

Unloads the specified bitmap file from memory, and frees up all the system resources associated with this bitmap.

See Also

MGL_loadBitmap

MGL_unloadCursor

Unloads a cursor file from memory.

Declaration

```
void MGLAPI MGL_unloadCursor(  
    cursor_t *cursor)
```

Prototype In

mgraph.h

Parameters

cursor Pointer to cursor to unload

Description

Unloads the specified cursor file from memory, and frees up all the system resources associated with this cursor.

See Also

MGL_loadCursor

MGL_unloadFont

Unloads a font file from memory.

Declaration

```
void MGLAPI MGL_unloadFont(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font Pointer to font to unload

Description

Unloads the specified font file from memory, and frees up all the system resources associated with this font.

See Also

MGL_loadCursorExt

MGL_unloadIcon

Unloads an icon file from memory.

Declaration

```
void MGLAPI MGL_unloadIcon(  
    icon_t *icon)
```

Prototype In

mgraph.h

Parameters

<i>icon</i>	Pointer to icon to unload
--------------------	---------------------------

Description

Unloads the specified icon file from memory, and frees up all the system resources associated with this icon.

See Also

MGL_loadIcon

MGL_unpackColor

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGLAPI MGL_unpackColor(  
    pixel_format_t *pf,  
    color_t color,  
    uchar *R,  
    uchar *G,  
    uchar *B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>R</i>	Place to store red extracted component
<i>G</i>	Place to store green extracted component
<i>B</i>	Place to store blue extracted component

Description

This function takes a packed color value in the correct format for the specified pixel format 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 pixel format is a 15 or 16 bit format because of loss of precision. The values are scaled back into the normal 24 bit RGB space.

See Also

MGL_unpackColorFast, *MGL_packColor*, *MGL_getPixelFormat*

MGL_unpackColorFast

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGL_unpackColorFast(  
    pixel_format_t *pf,  
    color_t color,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>R</i>	Place to store red extracted component
<i>G</i>	Place to store green extracted component
<i>B</i>	Place to store blue extracted component

Description

This function is the same as *MGL_unpackColor*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_unpackColor, *MGL_packColor*, *MGL_getPixelFormat*

MGL_unpackColorRGB

Unpacks a packed 24 bit color value into RGB components.

Declaration

```
void MGLAPI MGL_unpackColorRGB(  
    color_t color,  
    uchar *R,  
    uchar *G,  
    uchar *B)
```

Prototype In

mgraph.h

Parameters

<i>color</i>	Color to unpack
<i>R</i>	Place to store red extracted component
<i>G</i>	Place to store green extracted component
<i>B</i>	Place to store blue extracted component

Description

This function takes a packed 24 bit color value and extracts the red, green and blue components.

See Also

MGL_unpackColorRGBFast, MGL_packColorRGB, MGL_packColor

MGL_unpackColorRGBFast

Unpacks a packed 24 bit color value into RGB components.

Declaration

```
void MGL_unpackColorRGBFast(  
    color_t color,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>color</i>	Color to unpack
<i>R</i>	Place to store red extracted component
<i>G</i>	Place to store green extracted component
<i>B</i>	Place to store blue extracted component

Description

This function is the same as *MGL_unpackColorRGB*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_unpackColorRGB, *MGL_packColorRGB*, *MGL_packColor*

MGL_unregisterAllDrivers

Unregisters all currently registered display and memory drivers.

Declaration

```
void MGLAPI MGL_unregisterAllDrivers(void)
```

Prototype In

mgraph.h

Description

This function unregisters all currently registered display drivers and allows you to dynamically change the set of registered drivers at runtime. This function is most useful for allowing the user to select at runtime whether to use WinDirect VBE 2.0 and VBE/AF drivers or to use the Microsoft DirectDraw drivers. To switch between WinDirect and DirectDraw you can use this function to unregister all the drivers, then re-register only the DirectDraw drivers or only the WinDirect drivers.

Note that this function also unregisters all the registered memory device drivers, so you will need to re-register the packed pixel memory drivers that you require when you re-register the display drivers.

MGL_useEvents

Enables or disables event handling functions.

Declaration

```
void MGLAPI MGL_useEvents(  
    bool use)
```

Prototype In

mgldos.h

Parameters

use True to enable events, false to disable.

Description

This function allows the application programmer to enable or disable the use of the MGL event handling functions for DOS applications. If you wish to use MGL for graphics output, but you already have your own functions for handling keyboard and mouse input, you can call this function before you call *MGL_init* for the first time and it will disable the built in MGL event handling code.

See Also

MGL_suspend, *MGL_resume*

MGL_useFont

Sets the currently active font.

Declaration

```
bool MGLAPI MGL_useFont(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font New font to use

Return Value

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

Description

Selects the specified font as the currently active font for the active device context. If the font data is invalid, the MGL result flag is set and the routine will return false.

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

See Also

MGL_drawStr, MGL_loadFont, MGL_unloadFont

MGL_useLocalMalloc

Use local memory allocation routines.

Declaration

```
void MGL_useLocalMalloc(  
    void _HUGE * (*malloc)(long size),  
    void (*free)(void _HUGE *p))
```

Prototype In

mgraph.h

Parameters

~~*malloc*~~
~~*free*~~

Pointer to new malloc routine to use
Pointer to new free routine to use

Description

Tells MGL to use a set of user specified memory allocation routines instead of using the normal malloc/free standard library functions. This is useful if you wish to use a third party debugging malloc library or perhaps a set of faster memory allocation functions with MGL. Once you have registered your memory allocation routines, all calls to *MGL_malloc*, *MGL_calloc* and *MGL_free* will be revectorized to your local memory allocation routines.

This is also useful if you need to keep track of just how much physical memory your program has been using. You can use the *MGL_availableMemory* function to find out how much physical memory is available when the program starts, and then you can use your own local memory allocation routines to keep track of how much memory has been used and freed.

See Also

MGL_availableMemory, *MGL_malloc*, *MGL_calloc*, *MGL_free*

MGL_vSync

Waits for the vertical sync for a display device context.

Declaration

```
void MGLAPI MGL_vSync(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest

Description

This function will wait until the next vertical sync comes along for the current fullscreen graphics mode, before returning.

Generates the commands to draw a vectored font.

Declaration

```
bool MGLAPI MGL_vecFontEngine(  
    int x,  
    int y,  
    const char *str,  
    void (ASMAPI *move)(int x,int y),  
    void (ASMAPI *draw)(int x,int y))
```

Prototype In

mgraph.h

Parameters

<i>x</i>	<i>x</i> coordinate to start drawing text at
<i>str</i>	Character string to draw
<i>move</i>	Routine to call to perform a move operation
<i>draw</i>	Routine to call to perform a draw operation

Return Value

True if the string is correctly rasterized, false if font is not a vector font.

Description

This function calls a set of user supplied routines to rasterize 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 coordinates required to draw each character by any arbitrary transformation, 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 (i.e. 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 (i.e. inverse of normal device coordinate y-axis values).

This routine does not honor the standard scaling factors, but simply draws the characters with a 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.

See Also

MGL_drawStr, MGL_useFont

MGL_zbufferAccessType

Gets the Z-buffer surface direct access flags for a device context.

Declaration

```
int MGLAPI MGL_zbufferAccessType(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to get Z-buffer access flags for

Return Value

Direct surface access flag for the device context.

Description

This function returns the direct Z-buffer surface access flags for the specified device context, which can be used to determine if the Z-buffer surface for the device context is directly accessible, and if the surface has been virtualized in software. The access flags returned are enumerated in *MGL_surfaceAccessFlagsType*.

If the surface access flag is MGL_VIRTUAL_ZACCESS, this means that the surface for the device can be directly accessed, however the surface is actually virtualized in software using a page fault handler for SuperVGA devices that do not have real hardware linear Z-buffer access. If the surface is virtualized, you must ensure that when you directly access the surface you do so on BYTE, WORD and DWORD aligned boundaries. If you access it on a non-aligned boundary across a page fault, you will cause an infinite page fault loop to occur.

MS_available

Determines if a mouse is attached and functioning.

Declaration

```
bool MGLAPI MS_available(void)
```

Prototype In

mgraph.h

Return Value

True if a mouse is attached and can be used, false if mouse is not available.

Description

This function can be used to determine if a mouse pointer is installed and functioning, so that application software can determine if a mouse can be used or not. If this function returns false, it is up to the application software to provide full control via the keyboard or inform users that they will need to install a mouse.

MS_drawCursor

Draws the mouse cursor at the current location.

Declaration

```
void MGLAPI MS_drawCursor(void)
```

Prototype In

mgraph.h

Description

This function draws the mouse cursor at the current location regardless of it's hidden status. This function does not save the video memory below the mouse cursor, and is primarily intended to implement mouse cursors when performing double buffered or multibuffered animation. You simply call this function to draw the mouse cursor on the active display page after you have rasterized the frame. Note that when performing animation and drawing the mouse cursor, you should leave the normal mouse cursor hidden with a call to *MS_hide* and use this function to draw the cursor on the rasterized frame when necessary.

Note that this is different to the normal automatic mouse cursor drawing functions which always draw the mouse cursor onto the currently visible display page, which wont work with double or multibuffered animation techniques.

See Also

MS_show, *MS_hide*

MS_getPos

Returns the current mouse cursor location.

Declaration

```
void MGLAPI MS_getPos(  
    int *x,  
    int *y)
```

Prototype In

mgraph.h

Parameters

x	Place to store value for mouse x coordinate (screen coordinates)
y	Place to store value for mouse y coordinate (screen coordinates)

Description

Obtains the current mouse cursor position in screen coordinates. Normally the mouse cursor location is tracked using the mouse movement events that are posted to the event queue when the mouse moves, however this routine provides an alternative method of polling the mouse cursor location.

See Also

MS_moveTo

MS_hide

Hides the mouse cursor.

Declaration

```
void MGLAPI MS_hide(void)
```

Prototype In

mgraph.h

Description

Decrements the internal mouse cursor display counter, and hides the cursor if the counter was previously set to zero. Calls to *MS_show* increment the counter, allowing the *MS_show* and *MS_hide* calls to be nested.

See Also

MS_show, *MS_obscure*

MS_moveTo

Moves the mouse cursor to a new location.

Declaration

```
void MGLAPI MS_moveTo(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

x	New mouse x coordinate (screen coordinates)
y	New mouse y coordinate (screen coordinates)

Description

Moves the mouse cursor to the specified location in screen coordinates.

Note that it is not usually a good idea to move the mouse cursor around while the user is interacting with the application, but this can be used to restore the mouse cursor to a known location if it has been hidden for a long period of time.

See Also

MS_getPos

Hides the mouse cursor from view during graphics output.

Declaration

```
void MGLAPI MS_obscure(void)
```

Prototype In

mgraph.h

Description

Hides the mouse cursor from view in order to perform graphics output using 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 mouse cursor during graphics output as the *MS_hide* routine will always hide the cursor, regardless of whether the system has a hardware mouse cursor or not.

See Also

MS_show, *MS_hide*

MS_setCursor

Sets the mouse cursor shape.

Declaration

```
void MGLAPI MS_setCursor(  
    cursor_t *curs)
```

Prototype In

mgraph.h

Parameters

curs Pointer to new mouse cursor shape

Description

Sets the graphics mouse cursor shape, passed in the *cursor_t* structure. The *cursor_t* structure contains a mouse cursor and mask and a mouse cursor xor mask that is used to display the cursor on the screen, along with the mouse cursor hotspot location. Refer to the *cursor_t* structure definition for more information.

MS_setCursorColor

Sets the current mouse cursor color.

Declaration

```
void MGLAPI MS_setCursorColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color

New mouse cursor color, in current display mode format.

Description

Sets the color for the mouse cursor to the specified color, which is passed in as a packed MGL color in the proper format for the current display mode (either a color index or a packed RGB color value). By default the mouse cursor is set to white, which is a color index of 15 by default in MGL. If you re-program the color palette in 4 or 8 bit modes, you will need to reset the mouse cursor value to the value that represents white.

MS_show

Displays the mouse cursor.

Declaration

```
void MGLAPI MS_show(void)
```

Prototype In

mgraph.h

Description

Increments the internal mouse cursor display counter, and displays the cursor when the counter gets to zero. Calls to *MS_hide* decrement the counter, and this call effectively cancels a single *MS_hide* call, allowing the *MS_show* and *MS_hide* calls to be nested.

If the mouse was obscured with the *MS_obscure* function, this reverses the effect and will redisplay the mouse cursor again. On systems with a hardware mouse cursor, the *MS_obscure* function effectively does nothing, while on systems using a software mouse cursor, the *MS_obscure* function simply calls *MS_hide*.

Note that the mouse cursor display counter is reset to -1 by default when an MGL fullscreen mode is started, so a single *MS_show* will display the mouse cursor after the mode has been started.

See Also

MS_hide, *MS_obscure*

SPR_draw

Draws the sprite object at the specified location.

Declaration

```
void MGLAPI SPR_draw(  
    SPR_bitmap *bmp,  
    int x,  
    int y)
```

Prototype In

gm\sprite.h

Parameters

<i> bmp </i>	Sprite object to draw
<i> x </i>	X coordinate to draw the sprite at
<i> y </i>	Y coordinate to draw the sprite at

Description

This function draws the sprite object at the specified (x,y) location on the device context currently bound to the Sprite Manager (i.e.: the device context you used when you initialized it with *SPR_mgrInit*). The sprite is drawn using the attributes of the sprite when you added it (i.e.: opaque or source transparent).

See Also

SPR_mgrAddOpaqueBitmap, *SPR_mgrAddTransparentBitmap*,

SPR_mgrAddOpaqueBitmap

Adds an opaque or non-transparent bitmap to the Sprite Manager

Declaration

```
SPR_bitmap * MGLAPI SPR_mgrAddOpaqueBitmap(  
    bitmap_t *bmp)
```

Prototype In

gm\sprite.h

Parameters

bmp MGL bitmap to add to the Sprite Manager

Return Value

Pointer to the loaded sprite object

Description

This function adds a new opaque or non-transparent bitmap to the Sprite Manager. When you add the bitmap, the Sprite Manager takes over ownership of the memory allocated to the bitmap and you *must* not call *MGL_unloadBitmap* on the bitmap to free the memory. Once the bitmap has been added, you can then draw the bitmap by calling *SPR_draw* and pass in the pointer to the bitmap returned by this function.

Note: *The bitmap added to the Sprite Manager must be in the same format at the device context that the bitmaps will be copied to. Hence you should add code to your program to do any necessary conversions to the destination pixel format for the bitmaps (see the Fox & Bear sample program which contains code to do this).*

Note: *The Sprite Manager also maintains ownership of the SPR_bitmap object that is returned by this function, and this object will be destroyed automatically when you empty or exit the Sprite Manager.*

See Also

SPR_mgrAddTransparentBitmap, SPR_draw, SPR_mgrEmpty

SPR_mgrAddTransparentBitmap

Adds a source transparent bitmap to the Sprite Manager

Declaration

```
SPR_bitmap * MGLAPI SPR_mgrAddTransparentBitmap(  
    bitmap_t *bmp,  
    color_t transparent)
```

Prototype In

gm\sprite.h

Parameters

bmp

transparent

MGL bitmap to add to the Sprite Manager
Transparent color for the bitmap

Return Value

Pointer to the loaded sprite object

Description

This function adds a new transparent bitmap to the Sprite Manager. When you add the bitmap, the Sprite Manager takes over ownership of the memory allocated to the bitmap and you *must* not call *MGL_unloadBitmap* on the bitmap to free the memory. Once the bitmap has been added, you can then draw the bitmap by calling *SPR_draw* and pass in the pointer to the bitmap returned by this function.

Note that the transparent color you pass in is a *source transparent* color, which means that pixels in the source bitmap that match the transparent color will not be drawn when you call *SPR_draw* for the returned sprite object (i.e.: they are transparent).

Note: *The bitmap added to the Sprite Manager must be in the same format at the device context that the bitmaps will be copied to. Hence you should add code to your program to do any necessary conversions to the destination pixel format for the bitmaps (see the Fox & Bear sample program which contains code to do this).*

Note: *The Sprite Manager also maintains ownership of the SPR_bitmap object that is returned by this function, and this object will be destroyed automatically when you empty or exit the Sprite Manager.*

See Also

MGL Library Reference: *SPR_mgrAddOpaqueBitmap*, *SPR_draw*, *SPR_mgrEmpty*

SPR_mgrEmpty

Empties Sprite Manager of all loaded bitmaps

Declaration

```
void MGLAPI SPR_mgrEmpty(void)
```

Prototype In

gm\sprite.h

Description

This function empties the Sprite Manager of all currently loaded bitmaps and deallocates the memory owned by those bitmaps. This function is most useful to clearing the Sprite Manager of all bitmaps when moving from one level to another in your game.

See Also

SPR_mgrInit, SPR_mgrEmpty

SPR_mgrExit

Shuts down the Game Framework Sprite Manager

Declaration

```
void MGLAPI SPR_mgrExit(void)
```

Prototype In

gm\sprite.h

Description

This function shuts down the Sprite Manager, and destroys all bitmaps currently own by the Sprite Manager.

See Also

SPR_mgrInit, *SPR_mgrEmpty*

Initializes the Game Framework Sprite Manager

Declaration

```
bool MGLAPI SPR_mgrInit(  
    MGLDC *dc,  
    bool useRLE)
```

Prototype In

gm\sprite.h

Parameters

<i>dc</i>	MGL device context to use
<i>useRLE</i>	True if Run Length Encoding should be used for system bitmaps

Return Value

True on success, false on failure to initialize.

Description

This function initializes the Sprite Manager library and sets up for storing bitmaps with the sprite manager. The device context you pass in can be any MGL device context, but usually it should be an MGL display device context or an MGL memory device context. The Sprite Manager will automatically interrogate the capabilities of the device context, and if the device context supports the creation of an offscreen device context for storing sprites in video memory, an offscreen device context will be created and managed by the sprite manager.

The sprite manager automatically does all the work to keep track of which bitmaps are loaded in video memory and which are loaded in system memory. If there is not enough video memory left to store a sprite when you add it to the Sprite Manager, that sprite is stored in system memory automatically. When the sprites are loaded in system memory, if you pass in true to the useRLE parameter the Sprite Manager will compress all the system memory sprites using a Run Length Encoding algorithm which can lead to faster performance when blitting from system memory to video memory. If you pass in false to this parameter, the Sprite Manager will simply store the bitmaps in system memory and display them using

MGL_putBitmap and *MGL_putTransparentBitmap*.

Note: *When you add bitmaps to the Sprite Manager, those bitmaps are then owned by the Sprite Manager and will be automatically destroyed with *MGL_unloadBitmap* when the Sprite Manager is emptied of all bitmaps.*

Note: *If you switch fullscreen graphics modes or you switch from fullscreen modes to windowed modes, you must call *SPR_mgrExit* to destroy the bitmap manager and re-initialize it with the newly created device contexts. This is necessary because the capabilities of the new device context and the amount of available offscreen video memory (if any) will be different in the new graphics mode and all the sprites will have to be re-loaded into the Sprite Manager.*

See Also

SPR_mgrExit

SPR_mgrOffscreenCacheFull

Determines if the Sprite Manager offscreen device context is full.

Declaration

```
bool MGLAPI SPR_mgrOffscreenCacheFull(void)
```

Prototype In

gm\sprite.h

Return Value

True if Sprite Manager offscreen device context is full.

Description

This function is used to determine if the Sprite Manager's offscreen device context (if it is using one) is full and cannot accept any more bitmaps. This function is most useful for determining if the offscreen video memory is full and that the Sprite Manager has started overflowing the bitmaps into system memory. In this case you can expect the performance of your application to drop when the offscreen video memory is full.

See Also

SPR_mgrUsingOffscreenDC

SPR_mgrReloadHW

Reloads all offscreen sprites to the hardware

Declaration

```
void MGLAPI SPR_mgrReloadHW(void)
```

Prototype In

gm\sprite.h

Description

This function re-downloads all sprites that are stored in offscreen video memory to the offscreen device context. This function is usually called in the Game Framework suspend application callback, since on re-activation and switching back to GDI mode all the offscreen memory has been lost and needs to be reloaded.

SPR_mgrUsingOffscreenDC

Determines if the Sprite Manager is using a hardware offscreen DC.

Declaration

```
bool MGLAPI SPR_mgrUsingOffscreenDC(void)
```

Prototype In

gm\sprite.h

Return Value

True if Sprite Manager is using a hardware offscreen device context.

See Also

SPR_mgrOffscreenCacheFull

SVGA_setBank

Changes the active bank on banked framebuffer devices.

Declaration

```
void ASMAPI SVGA_setBank(void)
```

Prototype In

mgraph.h

Description

This assembler callable function changes the active SuperVGA bank in banked framebuffer modes, so that you can implement your own high performance direct framebuffer rasterizing code for banked modes. To call this function you simply load the new bank value into the DL register and then call it from your assembler rasterizing code. All registers including the DL register will be preserved across the call.

Note that if you are doing direct rasterizing in a bank framebuffer, the device context surface pointer will be a pointer to the start of the banked framebuffer and will never change.

See Also

SVGA_setBankC

SVGA_setBankC

Changes the active bank on banked framebuffer devices.

Declaration

```
void ASMAPI SVGA_setBankC(int bank)
```

Prototype In

mgraph.h

Parameters

bank New bank number to change to

Description

This C callable function changes the active SuperVGA bank in banked framebuffer modes, so that you can implement your own high performance direct framebuffer rasterizing code for banked modes. To call this function you simply pass the new bank value then call it from your C rasterizing code.

Note that if you are doing direct rasterizing in a bank framebuffer, the device context surface pointer will be a pointer to the start of the banked framebuffer and will never change.

See Also

SVGA_setBank

ULZElapsedTime

Compute the elapsed time between two timer counts.

Declaration

```
ulong ULZElapsedTime(ulong start,ulong finish)
```

Prototype In

ztimer.h

Parameters

start
finish

Starting time for elapsed count
Ending time for elapsed count

Return Value

Elapsed timer in resolution counts.

Description

Returns the elapsed time for the Ultra Long Period Zen Timer in units of the timers resolution (1/18th of a second under DOS). This function correctly computes the difference even if a midnight boundary has been crossed during the timing period.

See Also

ULZReadTime, *ULZTimerResolution*

ULZReadTime

Reads the current time from the Ultra Long Period Zen Timer.

Declaration

```
ulong ULZReadTime(void)
```

Prototype In

ztimer.h

Return Value

Current timer value in resolution counts.

Description

Reads the current Ultra Long Period Zen Timer and returns it's current count. You can use the *ULZElapsedTime* function to find the elapsed time between two timer count readings.

See Also

ULZElapsedTime, *ULZTimerResolution*

ULZTimerCount

Returns the current count for the Ultra Long Period Zen Timer.

Declaration

```
ulong ULZTimerCount(void)
```

Prototype In

ztimer.h

Return Value

Count that has elapsed in resolution counts.

Description

Returns the current count that has elapsed between calls to *ULZTimerOn* and *ULZTimerOff* in resolution counts.

See Also

ULZTimerOn, *ULZTimerOff*, *ULZTimerLap*, *ULZTimerResolution*

ULZTimerLap

Returns the current count for the Ultra Long Period Zen Timer and keeps it running.

Declaration

```
ulong ULZTimerLap(void)
```

Prototype In

ztimer.h

Return Value

Count that has elapsed in resolution counts.

Description

Returns the current count that has elapsed since the last call to *ULZTimerOn* in microseconds. The time continues to run after this function is called so you can call this function repeatedly.

See Also

ULZTimerOn, *ULZTimerOff*, *ULZTimerCount*

ULZTimerOff

Stops the Long Period Zen Timer counting.

Declaration

```
void ULZTimerOff(void)
```

Prototype In

ztimer.h

Description

Stops the Ultra Long Period Zen Timer counting and latches the count. Once you have stopped the timer you can read the count with *ULZTimerCount*.

See Also

ULZTimerOn, *ULZTimerLap*, *ULZTimerCount*

Starts the Ultra Long Period Zen Timer counting.

Declaration

```
void ULZTimerOn(void)
```

Prototype In

ztimer.h

Description

Starts the Ultra Long Period Zen Timer counting. Once you have started the timer, you can stop it with *ULZTimerOff* or you can latch the current count with *ULZTimerLap*.

The Ultra Long Period Zen Timer uses the available operating system services to obtain accurate timings results with as much precision as the operating system provides, but with enough granularity to time longer periods of time than the Long Period Zen Timer. Note that the resolution of the timer ticks is not constant between different platforms, and you should use the *ULZTimerResolution* function to determine the number of seconds in a single tick of the timer, and use this to convert the timer counts to seconds.

Under 32-bit Windows, we use the *timeGetTime* function which provides a resolution of 1 millisecond (0.001 of a second). Given that the timer count is returned as an unsigned 32-bit integer, this we can time intervals that are a maximum of 2^{32} milliseconds in length (or about 1,200 hours or 50 days!).

Under 32-bit DOS, we use the system timer tick which runs at 18.2 times per second. Given that the timer count is returned as an unsigned 32-bit integer, this we can time intervals that are a maximum of $2^{32} * (1/18.2)$ in length (or about 65,550 hours or 2731 days!).

See Also

ULZTimerOff, *ULZTimerLap*, *ULZTimerCount*, *ULZElapsedTime*, *ULZReadTime*

ULZTimerResolution

Returns the resolution of the Ultra Long Period Zen Timer.

Declaration

```
float ULZTimerResolution(void)
```

Prototype In

ztimer.h

Return Value

Resolution of the timer in seconds per timer count.

Description

Returns the resolution of the Ultra Long Period Zen Timer as a floating point value measured in seconds per timer count.

See Also

ULZReadTime, *ULZElapsedTime*, *ULZTimerCount*

ZTimerInit

Initializes the Zen Timer library.

Declaration

```
void ZTimerInit(void)
```

Prototype In

ztimer.h

Description

This function initializes the Zen Timer library, and *must* be called before any of the remaining Zen Timer library functions are called.

Data Structure Reference

CPU_largeInteger

Declaration

```
typedef struct {  
    ulong    low;  
    ulong    high;  
} CPU_largeInteger
```

Prototype In

ztimer.h

Description

Defines the structure for holding 64-bit integers used for storing the values returned by the Intel RDTSC instruction.

Members

low

Low 32 bits of the 64-bit integer

CPU_processorType

Declaration

```
typedef enum {  
    CPU_unknown      = 0,  
    CPU_i386         = 1,  
    CPU_i486         = 2,  
    CPU_Pentium       = 3,  
    CPU_PentiumPro    = 4,  
    CPU_PentiumII     = 5,  
    CPU_UnkPentium    = 6,  
    CPU_Alpha         = 100,  
    CPU_Mips          = 200,  
    CPU_PowerPC       = 300,  
    CPU_mask          = 0x7FFF,  
    CPU_IntelClone    = 0x8000,  
} CPU_processorType
```

Prototype In

ztimer.h

Description

Defines the types of processors returned by CPU_getType.

Members

<i>CPU_i386</i>	Intel 80386 processor
<i>CPU_i486</i>	Intel 80486 processor
<i>CPU_Pentium</i>	Intel Pentium(R) processor
<i>CPU_PentiumPro</i>	Intel PentiumPro(R) processor
<i>CPU_PentiumII</i>	Intel PentiumII(R) processor
<i>CPU_UnkPentium</i>	Unknown Intel Pentium family processor
<i>CPU_Alpha</i>	DEC Alpha processor
<i>CPU_Mips</i>	MIPS processor
<i>CPU_PowerPC</i>	PowerPC processor
<i>CPU_mask</i>	Mask to remove flags and get CPU type
<i>CPU_IntelClone</i>	This bit is set if the processor is an Intel clone

Declaration

```
typedef struct {
    MGLDC          *dc;
    MGLDC          *dispc;
    MGLDC          *memdc;
    int             driver;
    int             numModes;
    int             numFullscreenModes;
    unsigned long   modeFlags;
    GM_modeInfo     modeList[grMAXMODE+1];
    MGL_HWND        mainWindow;
} GMDC
```

Prototype In

gm\gm.h

Description

Main structure for maintaining the state information for the Game Framework. The application program always does all drawing and rendering to the *GMDC* *dc* member, which will draw directly to the framebuffer or to a system memory buffer depending on the hardware and the initialization information. The *modeFlags* field contains the original mode flags information passed to *GM_init*, which defines which color depths your game will support. The *modeList* contains a complete list of all the available graphics modes supported by the Game Framework, including *psuedo* modes that are modes that include auto-stretching.

The *dispc* and *memdc* field are primarily for internal use by the Game Framework, and you should not use those fields unless you are clear what they are used for.

Members

<i>dc</i>	DC for drawing
<i>dispdc</i>	Main display DC
<i>memdc</i>	Memory back buffer if necessary
<i>driver</i>	Driver ID detected by SciTech MGL
<i>numModes</i>	Number of modes in the mode list
<i>numFullscreenMo</i>	Number of fullscreen capable modes in the mode list
<i>modeFlags</i>	Mode flags for current graphics mode
<i>modeList</i>	List of all available modes supported by the Game Framework
<i>mainWindow</i>	Handle to main window (Windows only)

GM_driverOptions

Declaration

```
typedef struct {  
    bool                useWinDirect;  
    bool                useDirectDraw;  
    bool                useVGA;  
    bool                useVGAX;  
    bool                useVBE;  
    bool                useLinear;  
    bool                useVBEAF;  
    bool                useFullscreenDIB;  
    bool                useHWOpenGL;  
    MGL_glOpenGLType    openGLType;  
    GM_modeFlagsType    modeFlags;  
} GM_driverOptions
```

Prototype In

gm\gm.h

Description

This structure contains the flags for letting the Game Framework know which drivers should be registered with SciTech MGL to enable support for different device driver technologies. By default all these drivers are enabled, and you can change the values of these flags by calling *GM_setDriverOptions* before calling *GM_init*.

Members

<i>useWinDirect</i>	True to enable WinDirect support
<i>useDirectDraw</i>	True to enable DirectDraw support
<i>useVGA</i>	True to enable Standard VGA support
<i>useVGAX</i>	True to enable VGA ModeX support
<i>useVBE</i>	True to enable VBE 2.0 support
<i>useLinear</i>	True to enable linear framebuffer support
<i>useVBEAF</i>	True to enable VBE/AF support
<i>useFullscreenDIB</i>	True to enable fullscreen DIB support
<i>useHWOpenGL</i>	True to enable OpenGL hardware support
<i>openGLType</i>	OpenGL type to be used (defaults to MGL_GL_AUTO)
<i>modeFlags</i>	Mode flags for supported color depths

Declaration

```
typedef enum {  
    kbGrayEnter      = 0xE00D,  
    kbGrayMinus      = 0x4A2D,  
    kbGrayPlus       = 0x4E2B,  
    kbGrayTimes      = 0x372A,  
    kbGrayDivide     = 0x36E0,  
    kbF1              = 0x3B00,  
    kbF2              = 0x3C00,  
    kbF3              = 0x3D00,  
    kbF4              = 0x3E00,  
    kbF5              = 0x3F00,  
    kbF6              = 0x4000,  
    kbF7              = 0x4100,  
    kbF8              = 0x4200,  
    kbF9              = 0x4300,  
    kbF10             = 0x4400,  
    kbF11             = 0x8500,  
    kbF12             = 0x8600,  
    kbLeft            = 0x4B00,  
    kbRight           = 0x4D00,  
    kbUp              = 0x4800,  
    kbDown            = 0x5000,  
    kbIns             = 0x5200,  
    kbDel             = 0x5300,  
    kbHome            = 0x4700,  
    kbEnd             = 0x4F00,  
    kbPgUp            = 0x4900,  
    kbPgDn            = 0x5100,  
} GM_keyCodes
```

Description

The following is a set of useful constants for the keyboard message passed in the *event_t* message field by SciTech MGL's event handling functions. Use *EVT_asciiCode* to extract the ASCII codes for regular keys.

GM_modeFlagsType

Declaration

```
typedef enum {  
    GM_MODE_8BPP          = 0x01,  
    GM_MODE_16BPP         = 0x02,  
    GM_MODE_24BPP         = 0x04,  
    GM_MODE_32BPP         = 0x08,  
    GM_ONLY_2D_ACCEL      = 0x10,  
    GM_ONLY_3D_ACCEL      = 0x20,  
    GM_ALLOW_WINDOWED     = 0x40,  
    GM_MODE_ALLBPP        = 0x0F,  
} GM_modeFlagsType
```

Prototype In

gm\gm.h

Description

Mode flags to pass to *GM_init*. These flags inform the Game Framework which color depths you want to support in your game, and can be a logical OR combination of all the available flags. Hence if you game only supports 8bpp modes then you would pass in *GM_MODE_8BPP*. If you game only supports 8bpp and 16bpp, then you would pass in *GM_MODE_8BPP | GM_MODE_16BPP*.

Note: *GM_MODE_16BPP* includes support for both 15bpp (5:5:5) and 16bpp (5:6:5) MGL modes and if you support *GM_MODE_16BPP* then you will have to be able to support either format on the end users system.

GM_MODE_16BPP	Include support for 15bpp and 16bpp modes
GM_MODE_8BPP	Include support for 8bpp modes (3 bytes per pixel)
GM_MODE_32BPP	Include support for 32bpp modes (4 bytes per pixel)
GM_ONLY_2D_ACCEL	Only include modes with 2D hardware support
GM_ONLY_3D_ACCEL	Only include modes with 3D hardware support
GM_ALLOW_WINDOWED	Allow for windowed only modes in mode list
GM_MODE_ALLBPP	Include support for all color depths

Declaration

```
typedef struct {
    int         mode;
    int         xRes;
    int         yRes;
    int         bitsPerPixel;
    int         pages;
    unsigned long flags;
    char        driverName[13];
    GM_stretchType stretch;
    GM_stretchType windowedStretch;
} GM_modeInfo
```

Prototype In

gm\gm.h

Description

The structure maintains information about the graphics modes that are supported by the game framework and is passed to *GM_setMode* to specify the mode to be initialized. Note that the xRes and yRes values are the logical resolution for the mode which may be different to the physical resolution, since the Game Framework also enumerates *pseudo* modes that use stretching. Hence even if the hardware does not have native support for a 320x240 mode, it may appear in the list using 320x480 as the real mode and a stretch factor of 1x2 or using 640x480 as the real mode and a stretch factor of 2x2.

If you wish to set a windowed mode directly set the mode parameter to grWINDOWED and the mode will start as a windowed mode.

Members

<i>mode</i>	Fullscreen MGL mode number (-1 means windowed mode)
<i>xRes</i>	Logical X resolution for mode (not physical!)
<i>yRes</i>	Logical Y resolution for mode (not physical!)
<i>bitsPerPixel</i>	Color depth for mode. Note 16bpp includes 15bpp (5:5:5)
<i>pages</i>	Number of hardware display pages for mode
<i>flags</i>	Mode flags for the mode
<i>driverName</i>	Name of driver that will be used in fullscreen modes
<i>stretch</i>	Stretch factor for the mode
<i>windowedStretch</i>	Stretch factor to use in windowed modes

GM_stretchType

Declaration

```
typedef enum {  
    GM_STRETCH_1x1,  
    GM_STRETCH_1x2,  
    GM_STRETCH_2x2,  
} GM_stretchType
```

Prototype In

gm\gm.h

Description

Stretch flags to pass to *GM_setMode*. The 1x2 stretch is useful for rendering in 320x200/240 modes on hardware that can't do double scanning but does support 320x400/480 modes natively. 1x2 stretching in SciTech MGL is very efficient so this is better than rendering at 320x400 natively if each pixel takes a lot of computation time.

Members

GM_STRETCH_1x1	No stretching
GM_STRETCH_1x2	1x2 stretch (2x vertical stretch)
GM_STRETCH_2x2	2x2 stretch (2x stretch in both X and Y)

Declaration

```
typedef struct {
    attributes_t a;
    void        *surface;
    void        *zbuffer;
    int         zbits;
    int         zwidth;
    gmode_t     mi;
    pixel_format_t pf;
    color_t     *colorTab;
    color_t     *shadeTab;
    int         bankOffset;

#ifdef 0
    } MGLDC
#endif
```

Prototype In

mgraph.h

Description

Structure representing the public structure of all MGL device contexts. You can use the information in this structure to directly access the device surface for developing your own custom rendering code with SciTech MGL.

Members

a

Current device attributes

surface

Pointer to device surface. In banked modes this pointer will point to the start of the banked framebuffer memory (generally the 0xA0000 physical memory address), and if you plan to draw directly to video memory you will have to use the `SVGA_bank` function to change banks during drawing. In hardware (or virtual) linear framebuffer modes, this pointer will be a pointer to the start of the linear framebuffer memory, and you can render directly to it as a single block of memory. Note that you should also if the device surface is virtualised for display device contexts by calling (`MGL_surfaceAccessType`). If the device surface is virtualised in software, you will need to ensure that you only access the surface on BYTE, WORD or DWORD boundaries.

zbuffer

Pointer to device Z-buffer for 3D rendering if allocated, or NULL if the Z-buffer has not been allocated, or if the Z-buffer cannot be directly accessed.

zbits

Number of bits per Z-buffer coordinate. Z-buffers in SciTech MGL can currently be either 16 or 32-bits deep.

zwidth

Width of the Z-buffer in Z-buffer coordinates. This is usually equal to the device resolution, but may be more if the Z-buffer width is rounded by the device driver.

mi

Mode attribute information for the device

pf

Current pixel format for the device context.

colorTab

Color lookup table cache for the device context. In color map modes this is an array of 256 *palette_t* structures that represent the color palette for the device context. In 15-bits per pixel and higher modes, this is an array of 256 *color_t* values which contains a lookup table with pre-packed color values for the current display mode. This is used by SciTech MGL when translating color index bitmaps and drawing them in 15-bit and higher modes.

shadeTab

Shade table for the device context, or NULL if no shade table has been allocated. SciTech MGL's 3D rendering functions allow you to render smooth shaded, color index primitives even in 15-bits per pixel and higher modes. In these modes the color indexes that you pass to SciTech MGL are actually indexes into this shade table, and this table is used as a final translation to get the real color for a pixel.

bankOffset

Current offset of starting bank number for the current display page. This value is useful if you are implementing your own custom banked rendering routines, and you will need to add this value to get to the first bank in video memory for the currently active display page.

Declaration

```
typedef struct {
    ibool    rgb_flag;
    ibool    alpha_flag;
    ibool    db_flag;
    int      depth_size;
    int      stencil_size;
    int      accum_size;
} MGLVisual
```

Prototype In mgraph.h

Description

Structure representing the information passed to SciTech MGL's OpenGL binding functions to both choose an appropriate visual that is supported by the OpenGL implementation and to pass in the information for the visual when a rendering context is created. Application code will fill in the structure and call *MGL_glChooseVisual* to find out a valid set of visual information that is close to what they requested, then call *MGL_glSetVisual* to make that the current visual for a specific MGL device context. The next call to *MGL_glCreateContext* will use the visual information previously requested with the call to *MGL_glSetVisual*.

Members

<i>alpha_flag</i>	True for alpha buffers (8-bits deep)
<i>db_flag</i>	True for an RGB mode, false for color index modes
<i>depth_size</i>	True for double buffered, false for single buffered
<i>stencil_size</i>	Size of depth buffer in bits
<i>accum_size</i>	Size of stencil buffer in bits
	Size of accumulation buffer in bits

MGL_COLORS

Declaration

```
typedef enum {  
    MGL_BLACK,  
    MGL_BLUE,  
    MGL_GREEN,  
    MGL_CYAN,  
    MGL_RED,  
    MGL_MAGENTA,  
    MGL_BROWN,  
    MGL_LIGHTGRAY,  
    MGL_DARKGRAY,  
    MGL_LIGHTBLUE,  
    MGL_LIGHTGREEN,  
    MGL_LIGHTCYAN,  
    MGL_LIGHTRED,  
    MGL_LIGHTMAGENTA,  
    MGL_YELLOW,  
    MGL_WHITE,  
} MGL_COLORS
```

Prototype In

mgraph.h

Description

Defines the MGL standard colors. This is the standard set of colors for the IBM PC in DOS graphics modes. The default palette will have been programmed to contain these values by SciTech MGL when a graphics mode is started. If the palette has been changed, the colors on the screen will not correspond to the names defined here. Under a Windowing system (like Windows, OS/2 PM or X Windows) these colors will not correspond to the default colors. For Windows see the *MGL_WIN_COLORS* enumeration which defines the default colors in a windowed mode.

MGL_WIN_COLORS

Declaration

```
typedef enum {
    MGL_WIN_BLACK           = 0,
    MGL_WIN_DARKRED         = 1,
    MGL_WIN_DARKGREEN       = 2,
    MGL_WIN_DARKYELLOW      = 3,
    MGL_WIN_DARKBLUE        = 4,
    MGL_WIN_DARKMAGENTA     = 5,
    MGL_WIN_DARKCYAN        = 6,
    MGL_WIN_LIGHTGRAY       = 7,
    MGL_WIN_TURQUOISE       = 8,
    MGL_WIN_SKYBLUE         = 9,
    MGL_WIN_CREAM           = 246,
    MGL_WIN_MEDIUMGRAY      = 247,
    MGL_WIN_DARKGRAY        = 248,
    MGL_WIN_LIGHTRED        = 249,
    MGL_WIN_LIGHTGREEN      = 250,
    MGL_WIN_LIGHTYELLOW     = 251,
    MGL_WIN_LIGHTBLUE       = 252,
    MGL_WIN_LIGHTMAGENTA    = 253,
    MGL_WIN_LIGHTCYAN       = 254,
    MGL_WIN_WHITE           = 255,
} MGL_WIN_COLORS
```

Prototype In

mgraph.h

Description

Defines the Windows standard colors for 256 color graphics modes when in a window. 8,9,246,247 are reserved and you should not count on these colors always being the same. For 16 color bitmaps, colors 248-255 are mapped to colors 8-15.

MGL_bdrStyleType

Declaration

```
typedef enum {  
    MGL_BDR_INSET,  
    MGL_BDR_OUTSET,  
    MGL_BDR_OUTLINE,  
} MGL_bdrStyleType
```

Prototype In

mgraph.h

Description

Defines the border drawing styles passed to *MGL_drawBorderCoord*.

Members

<i>MGL_BDR_INSET</i>	Interior is inset into screen
<i>MGL_BDR_OUTSET</i>	Interior is outset from screen
<i>MGL_BDR_OUTLINE</i>	Border is 3d outline

MGL_clipType

Declaration

```
typedef enum {  
    MGL_CLIPOFF      = 0,  
    MGL_CLIPON       = 1,  
} MGL_clipType
```

Prototype In

mgraph.h

Description

Defines flags to enable and disable clipping via *MGL_setClipMode*.

MGL_colorModes

Declaration

```
typedef enum {  
    MGL_CMAP_MODE,  
    MGL_DITHER_RGB_MODE,  
} MGL_colorModes
```

Prototype In

mgraph.h

Description

Defines the color mapped modes. By default SciTech MGL starts up in the default mode, however you can change the color map mode to use RGB halftone dithering with the *MGL_setColorMapMode* function. Note that if you do this, you must also re-program the color palette to use the RGB halftone palette returned by *MGL_getHalfTonePalette*.

Members

<i>MGL_CMAP_MODE</i>	Normal Color mapped mode
<i>MGL_DITHER_RGB_MODE</i>	24 bit RGB halftone dithered

Declaration

```
typedef enum {  
    grDETECT          = -1,  
    grNONE             = 0,  
    grVGA,  
    grFULLDIB,  
    grVESA,  
    grSVGA,  
    grACCEL,  
    grDDRAW,  
    grDDRAWACCEL,  
    grDDRAW3D,  
    grOPENGL,  
    grOPENGL_MGL_MINI,  
    grOPENGL_MGL,  
    grMAXDRIVER,  
} MGL_driverType
```

Prototype In

mgraph.h

Description

Defines the graphics subsystems available. This is the value returned in the driver parameter for *MGL_detectGraph* and *MGL_init*. Note that if a driver returns a value of *grOPENGL_MGL_MINI*, it is an MGL OpenGL mini-driver and does not implement the entire OpenGL API. It is up to the application developer to find out from the hardware vendor who supplied the driver exactly what features are supported and to make use of those features in their application.

Members

<i>grDETECT</i>	Auto detect the graphics subsystem
<i>grNONE</i>	No graphics hardware detected
<i>grVGA</i>	Standard VGA
<i>grFULLDIB</i>	Fullscreen DIBSection's on Win95/NT
<i>grVESA</i>	VESA VBE compliant SuperVGA
<i>grSVGA</i>	Unaccelerated SuperVGA
<i>grACCEL</i>	Accelerated SuperVGA
<i>grDDRAW</i>	Unaccelerated DirectDraw
<i>grDDRAWACCEL</i>	Accelerated DirectDraw
<i>grDDRAW3D</i>	3D Accelerated DirectDraw
<i>grOPENGL</i>	Hardware Accelerated OpenGL (ICD/MCD)
<i>grOPENGL_MGL_MINI</i>	MGL specific hardware OpenGL mini-driver
<i>grOPENGL_MGL</i>	MGL specific hardware OpenGL driver
<i>grMAXDRIVER</i>	Maximum driver number

MGL_errorType

Declaration

```
typedef enum {  
    grOK = 0,  
    grNoInit = -1,  
    grNotDetected = -2,  
    grDriverNotFound = -3,  
    grBadDriver = -4,  
    grLoadMem = -5,  
    grInvalidMode = -6,  
    grError = -8,  
    grInvalidName = -9,  
    grNoMem = -10,  
    grNoModeSupport = -11,  
    grInvalidFont = -12,  
    grBadFontFile = -13,  
    grFontNotFound = -14,  
    grOldDriver = -15,  
    grInvalidDevice = -16,  
    grInvalidDC = -17,  
    grInvalidCursor = -18,  
    grCursorNotFound = -19,  
    grInvalidIcon = -20,  
    grIconNotFound = -21,  
    grInvalidBitmap = -22,  
    grBitmapNotFound = -23,  
    grZbufferTooBig = -24,  
    grNewFontFile = -25,  
    grNoDoubleBuff = -26,  
    grNoHardwareBlt = -28,  
    grNoOffscreenMem = -29,  
    grInvalidPF = -30,  
    grLastError = -31,  
} MGL_errorType
```

Prototype In

mgraph.h

Description

Defines the error codes returned by *MGL_result*. If a function fails for any reason, you can check the error code return by *MGL_result* to determine the cause of the failure, or display an error message to the user with *MGL_errorMsg*.

Members

<i>grOK</i>	No error
<i>grNoInit</i>	Graphics driver has not been installed
<i>grNotDetected</i>	Graphics hardware was not detected
<i>grDriverNotFound</i>	Graphics driver file not found
<i>grBadDriver</i>	File loaded was not a graphics driver
<i>grLoadMem</i>	Not enough memory to load graphics driver
<i>grInvalidMode</i>	Invalid graphics mode for selected driver
<i>grError</i>	General graphics error
<i>grInvalidName</i>	Invalid driver name
<i>grNoMem</i>	Not enough memory to perform operation
<i>grNoModeSupport</i>	Select video mode not supported by hard.
<i>grInvalidFont</i>	Invalid font data
<i>grBadFontFile</i>	File loaded was not a font file
<i>grFontNotFound</i>	Font file was not found
<i>grOldDriver</i>	Driver file is an old version
<i>grInvalidDevice</i>	Invalid device for selected operation
<i>grInvalidDC</i>	Invalid device context
<i>grInvalidCursor</i>	Invalid cursor file
<i>grCursorNotFound</i>	Cursor file was not found
<i>grInvalidIcon</i>	Invalid icon file
<i>grIconNotFound</i>	Icon file was not found
<i>grInvalidBitmap</i>	Invalid bitmap file
<i>grBitmapNotFound</i>	Bitmap file was not found
<i>grZbufferTooBig</i>	Zbuffer allocation is too large
<i>grNewFontFile</i>	Only Windows 2.x font files supported
<i>grNoDoubleBuff</i>	Double buffering is not available
<i>grNoHardwareBlit</i>	No hardware bitBlt for OffscreenDC
<i>grNoOffscreenMem</i>	No available offscreen memory
<i>grInvalidPF</i>	Invalid pixel format for memory DC
<i>grLastError</i>	Last error number in list

MGL_eventMaskType

Declaration

```
typedef enum {  
    EVT_KEYEVT          = (EVT_KEYDOWN | EVT_KEYREPEAT |  
    EVT_KEYUP),  
    EVT_MOUSEEVT        = (EVT_MOUSEDOWN | EVT_MOUSEUP |  
    EVT_MOUSEMOVE),  
    EVT_MOUSECLICK      = (EVT_MOUSEDOWN | EVT_MOUSEUP),  
    EVT_EVERYEVT        = 0xFFFF,  
} MGL_eventMaskType
```

Prototype In

mgraph.h

Description

Defines the event code masks you can use to test for multiple types of events, since the event codes are mutually exclusive bit fields.

Members

<i>EVT_MOUSEEVT</i>	Mask for any mouse event
<i>EVT_KEYEVT</i>	Mask for any key event
<i>EVT_MOUSECLICK</i>	Mask for any mouse click event
<i>EVT_EVERYEVT</i>	Mask for any event

MGL_eventModMaskType

Declaration

```
typedef enum {  
    EVT_LEFTBUT      = 0x0001,  
    EVT_RIGHTBUT     = 0x0002,  
    EVT_RIGHTSHIFT   = 0x0008,  
    EVT_LEFTSHIFT    = 0x0010,  
    EVT_CTRLSTATE    = 0x0020,  
    EVT_ALTSTATE     = 0x0040,  
    EVT_LEFTCTRL     = 0x0080,  
    EVT_LEFTALT      = 0x0100,  
    EVT_SHIFTKEY     = 0x0018,  
} MGL_eventModMaskType
```

Prototype In

mgraph.h

Description

Defines the event modifier masks. These are the masks used to extract the modifier information from the modifiers field of the *event_t* structure. Note that the values in the modifiers field represent the values of these modifier keys at the time the event occurred, not the time you decided to process the event.

Members

<i>EVT_LEFTBUT</i>	Set if left mouse button was down
<i>EVT_RIGHTBUT</i>	Set if right mouse button was down
<i>EVT_RIGHTSHIFT</i>	Set if right shift was down
<i>EVT_LEFTSHIFT</i>	Set if left shift was down
<i>EVT_CTRLSTATE</i>	Set if ctrl key was down
<i>EVT_ALTSTATE</i>	Set if alt key was down
<i>EVT_LEFTCTRL</i>	Set if left ctrl key was down
<i>EVT_LEFTALT</i>	Set if left alt key was down
<i>EVT_SHIFTKEY</i>	Mask for any shift key down

MGL_eventMsgMaskType

Declaration

```
typedef enum {  
    EVT_LEFTBMask    = 0x0001,  
    EVT_RIGHTBMask   = 0x0004,  
    EVT_BOTHBMASK    = 0x0005,  
    EVT_ALLBMASK     = 0x0005,  
} MGL_eventMsgMaskType
```

Prototype In

mgraph.h

Description

Defines the event message masks for mouse events

Members

EVT_RIGHTBMask	Right button is held down
EVT_BOTHBMASK	Both left and right held down together
EVT_ALLBMASK	All buttons pressed

MGL_eventType

Declaration

```
typedef enum {  
    EVT_NULLEVT          = 0x0000,  
    EVT_KEYDOWN          = 0x0001,  
    EVT_KEYREPEAT        = 0x0002,  
    EVT_KEYUP            = 0x0004,  
    EVT_MOUSEDOWN        = 0x0008,  
    EVT_MOUSEUP          = 0x0010,  
    EVT_MOUSEMOVE        = 0x0020,  
    EVT_TIMERTICK        = 0x0040,  
    EVT_USEREVT          = 0x0080,  
} MGL_eventType
```

Prototype In

mgraph.h

Description

Defines the event codes returned in the *event_t* structures what field. Note that these are defined as a set of mutually exclusive bit fields, so you can test for multiple event types using the combined event masks defined in the *MGL_eventMaskType* enumeration.

Members

<i>EVT_NULLEVT</i>	A null event
<i>EVT_KEYDOWN</i>	Key down event
<i>EVT_KEYREPEAT</i>	Key repeat event
<i>EVT_KEYUP</i>	Key up event
<i>EVT_MOUSEDOWN</i>	Mouse down event
<i>EVT_MOUSEUP</i>	Mouse up event
<i>EVT_MOUSEMOVE</i>	Mouse movement event
<i>EVT_TIMERTICK</i>	Timer tick event
<i>EVT_USEREVT</i>	First user event

MGL_fontType

Declaration

```
typedef enum {  
    MGL_VECTORFONT = 1,  
    MGL_FIXEDFONT,  
    MGL_PROPFONT,  
} MGL_fontType
```

Prototype In

mgraph.h

Description

Defines the different font types

Members

MGL_VECTORFONT	Fixed width bitmap font
MGL_PROPFONT	Proportional width bitmap font

MGL_glContextFlagsType

Declaration

```
typedef enum {  
    MGL_GL_VISUAL           = 0x8000,  
    MGL_GL_FORCEMEM        = 0x4000,  
    MGL_GL_RGB              = 0x0000,  
    MGL_GL_INDEX            = 0x0001,  
    MGL_GL_SINGLE           = 0x0000,  
    MGL_GL_DOUBLE           = 0x0002,  
    MGL_GL_ACCUM             = 0x0004,  
    MGL_GL_ALPHA            = 0x0008,  
    MGL_GL_DEPTH            = 0x0010,  
    MGL_GL_STENCIL          = 0x0020,  
} MGL_glContextFlagsType
```

Prototype In

mgraph.h

Description

MGL_glCreateContext flags to initialize the pixel format used by the OpenGL rendering context. If you pass in *MGL_GL_VISUAL*, the visual used will be the one currently selected by the previous call to *MGL_glSetVisual*, and provides the application programmer with complete control over the pixel formats used.

You can pass in a combination of any of the other flags (i.e.: *MGL_GL_RGB* | *MGL_GL_DOUBLE* | *MGL_GL_DEPTH*) to let SciTech MGL know what you want and to have it automatically select an appropriate visual for you. This provides a quick and simple way to get application code up and running.

Members

<i>MGL_GL_VISUAL</i>	Use curently assigned visual from call to <i>MGL_glSetVisual</i>
<i>MGL_GL_FORCEMEM</i>	Force system memory back buffer for all rendering
<i>MGL_GL_RGB</i>	Select RGB rendering mode (/default/)
<i>MGL_GL_INDEX</i>	Select color index display mode
<i>MGL_GL_SINGLE</i>	Select single buffered display mode (/default/)
<i>MGL_GL_DOUBLE</i>	Select double buffered display mode
<i>MGL_GL_ACCUM</i>	Enable accumulation buffer (16 bits)
<i>MGL_GL_ALPHA</i>	Enable alpha buffer (8 bit)
<i>MGL_GL_DEPTH</i>	Enable depth buffer (16 bits)
<i>MGL_GL_STENCIL</i>	Enable stencil buffer (8 bits)

MGL_glOpenGLType

Declaration

```
typedef enum {  
    MGL_GL_AUTO,  
    MGL_GL_MICROSOFT,  
    MGL_GL_SGI,  
    MGL_GL_MESA,  
    MGL_GL_HWMGL,  
} MGL_glOpenGLType
```

Prototype In

mgraph.h

Description

MGL_glSetOpenGL flags to select the OpenGL implementation. In the AUTO mode we automatically determine which version of OpenGL to use depending on the target runtime system. For Win32 unless there is hardware acceleration available we choose Silicon Graphic's OpenGL, but if hardware acceleration is present we use the regular Microsoft OpenGL implementation so we can utilize the hardware. For DOS we currently use the Mesa implementation, but you can also force Mesa to be used for the Windows environment if you wish.

Members

<i>MGL_GL_AUTO</i>	Automatically choose OpenGL implementation
<i>MGL_GL_MICROSOFT</i>	Force Microsoft OpenGL implementation
<i>MGL_GL_SGI</i>	Force SGI OpenGL implementation
<i>MGL_GL_MESA</i>	Force Mesa OpenGL implementation
<i>MGL_GL_HWMGL</i>	Force MGL specific hardware OpenGL implementation

MGL_hardwareFlagsType

Declaration

```
typedef enum {  
    MGL_HW_NONE           = 0x0000,  
    MGL_HW_LINE           = 0x0010,  
    MGL_HW_STIPPLE_LINE   = 0x0020,  
    MGL_HW_POLY           = 0x0040,  
    MGL_HW_RECT           = 0x0080,  
    MGL_HW_PATT_RECT      = 0x0100,  
    MGL_HW_CLRPATT_RECT   = 0x0200,  
    MGL_HW_SCR_BLT        = 0x0400,  
    MGL_HW_SRCTRANS_BLT   = 0x0800,  
    MGL_HW_DSTTRANS_BLT   = 0x1000,  
    MGL_HW_MONO_BLT       = 0x2000,  
    MGL_HW_CLIP           = 0x4000,  
    MGL_HW_FLAGS          = 0xFFFF,  
} MGL_hardwareFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags for the types of hardware acceleration supported by the device context. This will allow the application to tailor the use of MGL functions depending upon whether specific hardware support is available. Hence applications can use specialised software rendering support if the desired hardware support is not available on the end user system.

Note: *If the hardware flags are not MGL_HW_NONE, you must call the MGL_beginDirectAccess and MGL_endDirectAccess functions before and after any custom code that does direct framebuffer rendering!! This is not necessary for non-accelerated device contexts, so you might want to optimise these calls out if there is no hardware acceleration support.*

Members

<i>MGL_HW_NONE</i>	No hardware acceleration
<i>MGL_HW_LINE</i>	Hardware line drawing
<i>MGL_HW_STIPPLE_LINE</i>	Hardware stippled line drawing
<i>MGL_HW_POLY</i>	Hardware polygon filling
<i>MGL_HW_RECT</i>	Hardware rectangle fill
<i>MGL_HW_PATT_RECT</i>	Hardware pattern rectangle fill
<i>MGL_HW_CLRPATT_RECT</i>	Hardware color pattern fill
<i>MGL_HW_SCR_BLT</i>	Hardware screen/screen bitBlt
<i>MGL_HW_SRCTRANS_BLT</i>	Hardware source transparent Blt
<i>MGL_HW_DSTTRANS_BLT</i>	Hardware dest. transparent Blt
<i>MGL_HW_MONO_BLT</i>	Hardware monochrome Blt
<i>MGL_HW_CLIP</i>	Hardware clipping

MGL_lineStyleType

Declaration

```
typedef enum {  
    MGL_LINE_PENSTYLE,  
    MGL_LINE_STIPPLE,  
} MGL_lineStyleType
```

Prototype In

mgraph.h

Description

Defines the line styles passed to *MGL_setLineStyle*.

Members

MGL_LINE_PENSTYLE	Line drawn with current pen style.
MGL_LINE_STIPPLE	Line drawn with current line stipple pattern. The line stipple pattern is a 16x1 pattern that defines which pixels should be drawn in the line. Where a bit is a 1 in the pattern, a pixel will be drawn and where a bit is a 0 in the pattern, the pixel will be left untouched on the screen.

MGL_markerStyleType

Declaration

```
typedef enum {  
    MGL_MARKER_SQUARE,  
    MGL_MARKER_CIRCLE,  
    MGL_MARKER_X,  
} MGL_markerStyleType
```

Prototype In

mgraph.h

Description

Defines the marker types passed to *MGL_setMarkerStyle*

Members

<i>MGL_MARKER_SQUARE</i>	Draw a solid square at the marker location
<i>MGL_MARKER_CIRCLE</i>	Draw a solid circle at the marker location
<i>MGL_MARKER_X</i>	Draw a cross (X) at the marker location

MGL_modeFlagsType

Declaration

```
typedef enum {  
    MGL_HAVE_LINEAR           = 0x00000001,  
    MGL_HAVE_REFRESH_CTRL    = 0x00000002,  
    MGL_HAVE_INTERLACED      = 0x00000004,  
    MGL_HAVE_DOUBLE_SCAN     = 0x00000008,  
    MGL_HAVE_TRIPLEBUFFER    = 0x00000010,  
    MGL_HAVE_STEREO          = 0x00000020,  
    MGL_HAVE_STEREO_DUAL     = 0x00000040,  
    MGL_HAVE_STEREO_HWSYNC   = 0x00000080,  
    MGL_HAVE_STEREO_EVCSYNC  = 0x00000100,  
    MGL_HAVE_HWCURSOR        = 0x00000200,  
    MGL_HAVE_ACCEL_2D        = 0x00000400,  
    MGL_HAVE_ACCEL_3D        = 0x00000800,  
    MGL_HAVE_ACCEL_VIDEO     = 0x00001000,  
    MGL_HAVE_VIDEO_XINTERP   = 0x00002000,  
    MGL_HAVE_VIDEO_YINTERP   = 0x00004000,  
} MGL_modeFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags returned by the *MGL_modeFlags* functions. This function allows you to enumerate and detect support for different types of hardware features for a specific graphics mode after calling *MGL_detectGraph*, but before you actually initialise the desired mode. This will allow your application to search for fullscreen graphics modes that have the features that you desire (such as 2D or 3D acceleration).

Members

MGL_HAVE_LINEAR

Graphics mode supports a hardware linear framebuffer.

MGL_HAVE_REFRESH_CTRL

Graphics mode supports refresh rate control, allowing you to increase the refresh rate to a desired value (such as high refresh rates for stereo LC shutter glasses support).

MGL_HAVE_INTERLACED

Graphics mode supports interlaced operation, and you can request and interlaced mode via the refresh rate control mechanism in SciTech MGL.

MGL_HAVE_DOUBLE_SCAN

Graphics mode supports double scan operation.

MGL_HAVE_TRIPLEBUFFER

Graphics mode supports hardware triple buffering, allowing your application to use true triple buffering without any visible flickering.

MGL_HAVE_STEREO

Graphics mode supports hardware stereo page flipping, providing hardware support for stereo LC shutter glasses.

MGL_HAVE_STEREO_DUAL

Graphics mode supports hardware stereo page flipping, with dual display start addresses.

MGL_HAVE_STEREO_HWSYNC

Graphics mode provides hardware stereo sync support via an external connector for stereo LC shutter glasses.

MGL_HAVE_STEREO_EVCSYNC

Graphics mode provides support for the EVC stereo connector. If this bit is set, the above bit will also be set.

MGL_HAVE_HWCURSOR

Graphics mode supports a hardware cursor.

MGL_HAVE_ACCEL_2D

Graphics mode supports 2D hardware acceleration. 2D acceleration may be provided either by WinDirect and a VESA VBE/AF driver, or via DirectDraw.

MGL_HAVE_ACCEL_3D

Graphics mode supports 3D hardware acceleration. Hardware 3D acceleration is always provided in the form of an OpenGL hardware driver of some form.

MGL_HAVE_ACCEL_VIDEO

Graphics mode supports hardware video

MGL_HAVE_VIDEO_XINTERP

MGL_HAVE_VIDEO_YINTERP

acceleration, either via WinDirect and a VESA VBE/AF driver, or via DirectDraw. Graphics mode supports hardware video with interpolation along the X axis. Graphics mode supports hardware video with interpolation along the Y axis.

Declaration

```
typedef enum {  
    grVGA_320x200x16,  
    grVGA_640x200x16,  
    grVGA_640x350x16,  
    grVGA_640x400x16,  
    grVGA_640x480x16,  
    grSVGA_800x600x16,  
  
    grVGA_320x200x256,  
  
    grVGAX_320x200x256,  
    grVGAX_320x240x256,  
    grVGAX_320x400x256,  
    grVGAX_320x480x256,  
  
    grSVGA_320x200x256,  
    grSVGA_320x240x256,  
    grSVGA_320x400x256,  
    grSVGA_320x480x256,  
    grSVGA_400x300x256,  
    grSVGA_512x384x256,  
    grSVGA_640x350x256,  
    grSVGA_640x400x256,  
    grSVGA_640x480x256,  
    grSVGA_800x600x256,  
    grSVGA_1024x768x256,  
    grSVGA_1152x864x256,  
    grSVGA_1280x960x256,  
    grSVGA_1280x1024x256,  
    grSVGA_1600x1200x256,  
  
    grSVGA_320x200x32k,  
    grSVGA_320x240x32k,  
    grSVGA_320x400x32k,  
    grSVGA_320x480x32k,  
    grSVGA_400x300x32k,  
    grSVGA_512x384x32k,  
    grSVGA_640x350x32k,  
    grSVGA_640x400x32k,  
    grSVGA_640x480x32k,  
    grSVGA_800x600x32k,  
    grSVGA_1024x768x32k,  
    grSVGA_1152x864x32k,
```

grSVGA_1280x960x32k,
grSVGA_1280x1024x32k,
grSVGA_1600x1200x32k,

grSVGA_320x200x64k,
grSVGA_320x240x64k,
grSVGA_320x400x64k,
grSVGA_320x480x64k,
grSVGA_400x300x64k,
grSVGA_512x384x64k,
grSVGA_640x350x64k,
grSVGA_640x400x64k,
grSVGA_640x480x64k,
grSVGA_800x600x64k,
grSVGA_1024x768x64k,
grSVGA_1152x864x64k,
grSVGA_1280x960x64k,
grSVGA_1280x1024x64k,
grSVGA_1600x1200x64k,

grSVGA_320x200x16m,
grSVGA_320x240x16m,
grSVGA_320x400x16m,
grSVGA_320x480x16m,
grSVGA_400x300x16m,
grSVGA_512x384x16m,
grSVGA_640x350x16m,
grSVGA_640x400x16m,
grSVGA_640x480x16m,
grSVGA_800x600x16m,
grSVGA_1024x768x16m,
grSVGA_1152x864x16m,
grSVGA_1280x960x16m,
grSVGA_1280x1024x16m,
grSVGA_1600x1200x16m,

grSVGA_320x200x4G,
grSVGA_320x240x4G,
grSVGA_320x400x4G,
grSVGA_320x480x4G,
grSVGA_400x300x4G,
grSVGA_512x384x4G,
grSVGA_640x350x4G,
grSVGA_640x400x4G,
grSVGA_640x480x4G,
grSVGA_800x600x4G,
grSVGA_1024x768x4G,
grSVGA_1152x864x4G,

```
grSVGA_1280x960x4G,  
grSVGA_1280x1024x4G,  
grSVGA_1600x1200x4G,  
  
grWINDOWED,  
  
grMAXMODE,  
} MGL_modeType
```

Description

Defines the supported graphics modes. This is the value passed in the mode parameter to *MGL_init* and *MGL_changeDisplayMode*. Note that although we define symbolic constants for all the available graphics modes supported by this version of the library, for maximum compatibility with future versions of the library (which may define new modes as they become available), the best way to search for a supported mode is to search the list of modes returned by *MGL_availableModes*, and look for one that has the desired color depth and resolution by calling *MGL_modeResolution*. Check out how this is done in the SciTech Game Framework source code for more information.

Note: *The only video modes supported by this graphics library are those that support at least 16 colors per pixel.*

MGL_palRotateType

Declaration

```
typedef enum {  
    MGL_ROTATE_UP,  
    MGL_ROTATE_DOWN,  
} MGL_palRotateType
```

Prototype In

mgraph.h

Description

Defines the different palette rotation directions

Members

~~MGL_ROTATE_DOWN~~ Rotate the palette values up

MGL_penStyleType

Declaration

```
typedef enum {  
    MGL_BITMAP_SOLID,  
    MGL_BITMAP_OPAQUE,  
    MGL_BITMAP_TRANSPARENT,  
    MGL_PIXMAP,  
} MGL_penStyleType
```

Prototype In

mgraph.h

Description

Defines the pen styles passed to *MGL_setPenStyle*. These styles define the different fill styles that can be used when the filling the interior of filled primitives and also the outline of non-filled primitives.

Members

MGL_BITMAP_SOLID

Fill with a solid color

MGL_BITMAP_OPAQUE

Fill with an opaque bitmap pattern. Where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the background color is used. The pattern itself is defined as an 8x8 monochrome bitmap.

MGL_BITMAP_TRANSPARENT

Fill with a transparent bitmap pattern. Where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the pixel is left unmodified on the screen. The pattern itself is defined as an 8x8 monochrome bitmap.

MGL_PIXMAP

Fill with a color pixmap pattern. The pixmap pattern is defined as an 8x8 array of *color_t* values, where each entry corresponds to the color values packed for the appropriate color mode (i.e.: a color index in color map modes and a packed RGB value in HiColor and TrueColor modes).

MGL_polygonType

Declaration

```
typedef enum {  
    MGL_CONVEX_POLYGON,  
    MGL_COMPLEX_POLYGON,  
    MGL_AUTO_POLYGON,  
} MGL_polygonType
```

Prototype In

mgraph.h

Description

Defines the polygon types passed to *MGL_setPolygonType*.

Members

MGL_CONVEX_POLYGON

Monotone vertical polygon (includes convex polygons). A monotone vertical polygon is one whereby there will never be a horizontal line that can intersect the polygon at more than two edges at a time. Note that if you set the polygon type to this value and you pass it a non-monotone vertical polygon, the output results are undefined.

MGL_COMPLEX_POLYGON

Non-Simple polygons. When set to this mode SciTech MGL will correctly rasterise all polygon types that you pass to it, however the drawing will be slower.

MGL_AUTO_POLYGON

Auto detect the polygon type. In this mode SciTech MGL will examine the polygon vertices passed in and will automatically draw it with the faster routines if it is monotone vertical. Note that this does incur an overhead for the checking code, so if you know all your polygons are monotone vertical or convex, then you should set the type to *MGL_CONVEX_POLYGON*.

MGL_refreshRateType

Declaration

```
typedef enum {  
    MGL_DEFAULT_REFRESH = -1,  
    MGL_INTERLACED_MODE = 0x4000,  
} MGL_refreshRateType
```

Prototype In

mgraph.h

Description

Defines the flags passed to MGL_setRefreshRate. You can pass the value of MGL_DEFAULT_REFRESH to set the refresh rate to the adapter default.

Members

<i>MGL_DEFAULT_REFRESH</i>	Use the default refresh rate for the graphics card
<i>MGL_INTERLACED_MODE</i>	Set the mode to be interlaced (not always supported)

MGL_stereoBufType

Declaration

```
typedef enum {  
    MGL_LEFT_BUFFER           = 0x0000,  
    MGL_RIGHT_BUFFER          = 0x8000,  
} MGL_stereoBufType
```

Prototype In

mgraph.h

Description

Defines the flags passed to *MGL_setActivePage* to let SciTech MGL know which buffer you wish to draw to when running in stereo mode (i.e.: after a display device context created with *MGL_createStereoDisplayDC*). This value is logical 'or'ed with the page parameter to *MGL_setActivePage*.

Members

<i>MGL_LEFT_BUFFER</i>	Draw to the left buffer in stereo modes
<i>MGL_RIGHT_BUFFER</i>	Draw to the right buffer in stereo modes

MGL_stereoSyncType

Declaration

```
typedef enum {  
    MGL_STEREO_BLUE_CODE           = 0,  
    MGL_STEREO_PARALLEL_PORT       = 1,  
    MGL_STEREO_SERIAL_PORT         = 2,  
    MGL_STEREO_IGNORE_HW_STEREO    = 0x8000,  
} MGL_stereoSyncType
```

Prototype In

mgraph.h

Description

Defines the flags passed to *MGL_setStereoSyncType* to let SciTech MGL know what type of stereo synchronisation method should be used when running on a system without hardware stereo sync for LC shutter glasses. By default SciTech MGL will assume the hardware stereo sync works properly if the BIOS/Drivers report that this feature is available, however in cases where the BIOS mis-reports this or where the user has glasses that don't support this, you can disable automatic use of hardware stereo sync by 'or'ing in *MGL_STEREO_IGNORE_HARDWARE* when you call *MGL_setStereoSyncType*.

Members

MGL_STEREO_BLUE_CODE

Use the blue code synchronisation methods as supported by

MGL_STEREO_PARALLEL_PORT

Stereo Parallel Port shutter glasses. Synchronisation method as supported by NuVision LC shutter glasses.

MGL_STEREO_SERIAL_PORT

Use the serial port synchronisation method as supported by WooBoo CyberBoy LC shutter glasses.

MGL_STEREO_IGNORE_HARDWARE

Tell SciTech MGL to ignore hardware stereo sync and use the specified software stereo sync mechanism.

MGL_surfaceAccessFlagsType

Declaration

```
typedef enum {  
    MGL_NO_ACCESS          = 0x0 ,  
    MGL_VIRTUAL_ACCESS     = 0x1 ,  
    MGL_LINEAR_ACCESS      = 0x2 ,  
    MGL_SURFACE_FLAGS      = 0x3 ,  
} MGL_surfaceAccessFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags for the types of direct surface access provided.

Members

<i>MGL_NO_ACCESS</i>	Surface cannot be accessed
<i>MGL_VIRTUAL_ACCESS</i>	Surface is virtualised
<i>MGL_LINEAR_ACCESS</i>	Surface can be linearly accessed

MGL_suspendAppCodesType

Declaration

```
typedef enum {  
    MGL_NO_DEACTIVATE      = 0 ,  
    MGL_SUSPEND_APP        = 1 ,  
    MGL_NO_SUSPEND_APP     = 2 ,  
} MGL_suspendAppCodesType
```

Prototype In

mgraph.h

Description

Defines the return codes that the application can return from the suspend application callback registered with SciTech MGL. The default value to be returned is MGL_SUSPEND_APP and this will cause the application execution to be suspended until the application is re-activated again by the user. During this time the application will exist on the task bar under Windows 95 and Windows NT in minimised form.

MGL_NO_DEACTIVATE signals to SciTech MGL that the application does not want to allow switching to occur, and the switch request will be ignored and the app will remain in fullscreen mode. This is valid in both WinDirect and DirectX modes under Windows 3.1 and Windows 95, and allows you to give power users the option of disabling the ability to switch away from the application back to the desktop. Note however that under Windows NT this is not possible, so even if you return this under NT the switch will still occur.

MGL_NO_SUSPEND_APP can be used to tell SciTech MGL to switch back to the Windows desktop, but not to suspend the applications execution. This must be used with care as the suspend application callback is then responsible for setting a flag in the application that will *stop* the application from doing any rendering directly to the framebuffer while the application is minimised on the task bar (since the application no longer owns the screen!). This return value is most useful for networked games that need to maintain network connectivity while the user has temporarily switched back to the Windows desktop. Hence you can ensure that your main loop continues to run, including networking and AI code, but no drawing occurs to the screen.

Note: *SciTech MGL ensures that your application will never be switched away from outside of a message processing loop. Hence as long as you do not*

process messages inside your drawing loops, you will never lose the active focus (and your surfaces) while drawing, but only during event processing.

Members

MGL_NO_DEACTIVATE

Don't allow app to be deactivated

MGL_SUSPEND_APP

Suspend application execution until restored

MGL_NO_SUSPEND_APP

Don't suspend execution, but allow switch

MGL_suspendAppFlagsType

Declaration

```
typedef enum {  
    MGL_DEACTIVATE    = 0x0001,  
    MGL_REACTIVATE    = 0x0002,  
} MGL_suspendAppFlagsType
```

Prototype In

mgraph.h

Description

Defines the suspend application callback flags, passed to the suspend application callback registered with SciTech MGL. This callback is called when the user presses one of the system key sequences indicating that they wish to change the active application. SciTech MGL will catch these events and if you have registered a callback, will call the callback to save the state of the application so that it can be properly restored when the user switches back to your application. SciTech MGL takes care of all the details about saving and restoring the state of the hardware, and all your application needs to do is save its own state so that you can re-draw the application screen upon re-activation.

Note: *Your application suspend callback may get called twice with the MGL_DEACTIVATE flag in order to test whether the switch should occur (under both DirectX and WinDirect fullscreen modes).*

Note: *When your callback is called with the MGL_DEACTIVATE flag, you cannot assume that you have access to the display memory surfaces as they may have been lost by the time your callback has been called.*

MGL_REACTIVATE	Application regaining active focus
MGL_DEACTIVATE	Application losing active focus

MGL_textDirType

Declaration

```
typedef enum {  
    MGL_LEFT_DIR      = 0,  
    MGL_UP_DIR        = 1,  
    MGL_RIGHT_DIR     = 2,  
    MGL_DOWN_DIR      = 3,  
} MGL_textDirType
```

Prototype In

mgraph.h

Description

Defines the text direction styles passed to *MGL_setTextDirection*

Members

<i>MGL_LEFT_DIR</i>	Text goes to left
<i>MGL_UP_DIR</i>	Text goes up
<i>MGL_RIGHT_DIR</i>	Text goes right
<i>MGL_DOWN_DIR</i>	Text goes down

MGL_textJustType

Declaration

```
typedef enum {  
    MGL_LEFT_TEXT      = 0,  
    MGL_TOP_TEXT       = 0,  
    MGL_CENTER_TEXT    = 1,  
    MGL_RIGHT_TEXT     = 2,  
    MGL_BOTTOM_TEXT    = 2,  
    MGL_BASELINE_TEXT  = 3,  
} MGL_textJustType
```

Prototype In

mgraph.h

Description

Defines the text justification styles passed to *MGL_setTextJustify*.

Members

MGL_TOP_TEXT	Justify from top
MGL_CENTER_TEXT	Center the text
MGL_RIGHT_TEXT	Justify from right
MGL_BOTTOM_TEXT	Justify from bottom
MGL_BASELINE_TEXT	Justify from the baseline

MGL_waitVRTFlagType

Declaration

```
typedef enum {  
    MGL_tripleBuffer      = 0,  
    MGL_waitVRT           = 1,  
    MGL_dontWait          = 2,  
} MGL_waitVRTFlagType
```

Prototype In

mgraph.h

Description

Defines for waitVRT flag for *MGL_setVisualPage*, *MGL_swapBuffers* and *MGL_glSwapBuffers*.

Members

- MGL_tripleBuffer*** Triple buffer. This mode enables hardware or software triple buffering if available on the target system. In this case when triple buffering is available SciTech MGL will ensure that there is no flicker when flipping pages, however your frame rate will run at the maximum rate until you get to the physical refresh rate of the screen (i.e.: 60fps or higher). Note that if there is no hardware or software triple buffering available, this function will work like regular double buffering. Note also that you *must* have at least 3 pages available for triple buffering to work.
- MGL_waitVRT*** Wait for vertical retrace. This mode always waits for the vertical retrace when swapping display pages, and is required if only have two pages available to avoid flicker during animation.
- MGL_dontWait*** Don't wait for retrace. This mode simply programs the display start address change and returns. This may cause flicker on the screen during animation, and is mostly useful for debugging and testing purposes to see what the raw framerate of an animation is. Also, if you don't have hardware or software triple buffering available, and you allocate at least 3 pages, you can achieve the effect of triple buffering (if you know that the frame rate of your animation will not exceed the refresh rate of the screen).

MGL_writeModeType

Declaration

```
typedef enum {  
    MGL_REPLACE_MODE,  
    MGL_AND_MODE,  
    MGL_OR_MODE,  
    MGL_XOR_MODE,  
} MGL_writeModeType
```

Prototype In

mgraph.h

Description

Defines the Windows standard colors for 256 color graphics modes when in a window. 8,9,246,247 are reserved and you should not count on these colors always being the same. For 16 color bitmaps, colors 248-255 are mapped to colors 8-15.

Members

MGL_REPLACE_MODE	Replace mode
MGL_AND_MODE	AND mode
MGL_OR_MODE	OR mode
MGL_XOR_MODE	XOR mode

Declaration

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

Prototype In

mgraph.h

Description

Structure used to return elliptical arc starting and ending coordinates. This structure is used to obtain the exact center, starting and ending coordinates after an elliptical arc has been rasterized, so that you can properly turn the arc into a pie slice for example.

Members

<i>x</i>	x coordinate of the center of the elliptical arc
<i>y</i>	y coordinate of the center of the elliptical arc
<i>startX</i>	x coordinate of the starting pixel on the elliptical arc
<i>startY</i>	y coordinate of the starting pixel on the elliptical arc
<i>endX</i>	x coordinate of the ending pixel on the elliptical arc
<i>endY</i>	y coordinate of the ending pixel on the elliptical arc

Declaration

```
typedef struct {
    color_t      color;
    color_t      backColor;
    int          colorMode;
    int          markerSize;
    int          markerStyle;
    color_t      markerColor;
    color_t      bdrBright;
    color_t      bdrDark;
    point_t      CP;
    int          writeMode;
    int          penStyle;
    int          penHeight;
    int          penWidth;
    pattern_t    penPat;
    pixpattern_t penPixPat;
    int          lineStyle;
    uint         lineStipple;
    uint         stippleCount;
    rect_t       viewPort;
    point_t      viewPortOrg;
    rect_t       clipRect;
    int          clip;
    int          polyType;
    text_settings_t ts;
} attributes_t
```

Prototype In

mgraph.h

Description

Structure representing the current MGL rendering attributes. This structure groups all of the MGL rendering state variables, and can be used to save and restore the entire MGL rendering state for any device context as a single unit.

Note: *You should only save and restore the state to the same device context!*

Members

color	Current foreground color
backColor	Current background color
colorMode	Current color map mode. Will be either MGL_CMAP_MODE or MGL_DITHER_RGB_MODE. This value only affects 8 bit rasterizing modes, and determines whether rasterizing should be performed using straight color indexes or by dithering 24 bit RGB values on the fly to 8 bit colors.
markerSize	Current marker size
markerStyle	Current marker style. Will be one of values defined by the <i>MGL_markerStyleType</i> enumeration.
markerColor	Current marker color
bdrBright	Current border bright color
bdrDark	Current border dark color
CP	Current Position coordinate
writeMode	Current write mode. Will be one of the values defined by the <i>MGL_writeModeType</i> enumeration.
penStyle	Current pen fill style. Will be one of values defined by the <i>MGL_penStyleType</i> enumeration.
penHeight	Current pen height
penWidth	Current pen width
penPat	Current pen 8x8 monochrome bitmap pattern
penPixPat	Current pen 8x8 color pixmap pattern
lineStyle	Current line style. Will be one of the values defined by the <i>MGL_lineStyleType</i> enumeration.
lineStipple	Current 16-bit line stipple mask.
stippleCount	Current line stipple counter.
viewPort	Current viewport. All coordinates are offset relative to the current viewport when rendered, and output is always clipped to the viewport boundaries.
viewPortOrg	Current viewport logical origin. You can change the logical viewport origin independently after the viewport has been set. This effectively changes the logical coordinate of the top left corner of the viewport from the default of (0,0) to a new value (i.e. -10,-10).
clipRect	Current clip rectangle, stored in local viewport coordinates.
clip	True if clipping is currently on, false if not.
polyType	Current polygon rasterizing type. Will be one of the values defined by the <i>MGL_polygonType</i> enumeration.
ts	Current text drawing attributes

Declaration

```
typedef struct {
    int          width;
    int          height;
    int          bitsPerPixel;
    int          bytesPerLine;
    uchar        *surface;
    palette_t    *pal;
    pixel_format_t *pf;

    } bitmap_t
```

Prototype In mgraph.h

Description

Structure representing a loaded lightweight bitmap image. This is the structure of Windows .BMP files after they have been loaded from disk with the *MGL_loadBitmap* function. Lightweight bitmaps have very little memory overhead once loaded from disk, since the entire bitmap information is stored in a single contiguous block of memory (although this is not necessary; see below). However the only thing you can do with a lightweight bitmap is display it to any MGL device context, using either stretching or transparency (*MGL_putBitmap*, *MGL_stretchBitmap*, *MGL_putBitmapTransparent*). If you need to be able to draw on the bitmap surface, then you should load the bitmap into an MGL memory device context where you can call any of the standard MGL drawing functions and BitBlt operations on the bitmap. The only disadvantage of doing this is that a memory device context has a lot more memory overhead involved in maintaining the device context information.

You can build your own lightweight bitmap loading routines by creating the proper header information and loading the bitmap information into this structure. Note that although SciTech MGL loads the bitmap files from disk with the bitmap surface, pixel format information and palette information all loaded into a single memory block, this is not necessary. If you wish you can create your own lightweight bitmaps with the bitmap surface allocated in a separate memory block and then use this bitmap header to blast information from this memory block to a device context as fast as possible.

Members

width

Width of the bitmap in pixels

height

Height of the bitmap in pixels

bitsPerPixel

Pixel depth of the bitmap

bytesPerLine

Scanline width for the bitmap. The scanline width must always be aligned to a DWORD boundary, so the minimum scanline width is 4 bytes.

Surface

Pointer to the bitmap surface.

pal

Pointer to the bitmap palette. If this field is NULL, the bitmap does not have an associated palette.

pf

Pointer to the bitmap pixel format info. This field will be NULL for all bitmaps with 8 or less bits per pixel, but will always be properly filled in for bitmaps with 15 or more bits per pixel.

color_t

Declaration

```
typedef unsigned long color_t
```

Prototype In

mgraph.h

Description

Type definition for all color values used in MGL. All color values are 32 bits wide, and can be either a 4 or 8 bit color index, or a packed RGB tuple depending on the pixel format for the display mode. For packed RGB display modes, the colors may contain 15, 16, 24 or 32 bits of color information, and the format of the RGB colors is stored in the *pixel_format_t* structure. You should use the *MGL_packColor* family of functions to encode color values in RGB modes, and use the *MGL_unpackColor* family of functions to extract color values in RGB modes.

Declaration

```
typedef struct {  
    ulong    xorMask[32];  
    ulong    andMask[32];  
    int      xHotSpot;  
    int      yHotSpot;  
} cursor_t
```

Prototype In

mgraph.h

Description

Structure representing a loaded mouse cursor. This is the structure of the mouse cursor data after it has been loaded from disk by SciTech MGL, and is used to set the mouse cursor shape. You can build your own mouse cursors manually by filling in this structure.

The mouse cursor is drawn by SciTech MGL by first using the cursor AND mask to punch a hole in the background of the display surface, and then the cursor XOR mask is XOR'ed into the display surface using the currently active mouse cursor color. This method is compatible with the way that Microsoft Windows displays mouse cursors on the screen.

Note however that the AND mask for the cursor is expected to have a 0 where the background will be left alone and a 1 where the background should be set to 0. This is the opposite of the mask definition stored in the Windows .CUR cursor files, and MGL internally inverts this data when it loads the mouse cursors. The main reason for this inversion is for performance reasons, since SciTech MGL needs to work with the mask stored this way internally for faster internal drawing when drawing the cursor in software.

Members

<i>xorMask</i>	32x32 bit XOR pixel mask
<i>andMask</i>	32x32 bit AND pixel mask (see note above)
<i>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the screen
<i>yHotSpot</i>	y coordinate of the mouse hotspot location

Declaration

```
typedef struct {
    ulong    which;
    uint     what;
    ulong    when;
    int      where_x;
    int      where_y;
    ulong    message;
    ulong    modifiers;
    int      next;
    int      prev;
} event_t
```

Prototype In

mgraph.h

Description

Structure describing the information contained in an event extracted from the event queue.

Members

<i>which</i>	Window identifier for message for use by high level window manager code (i.e. MegaVision GUI or Windows API).
<i>what</i>	Type of event that occurred. Will be one of the values defined by the <i>MGL_waitVRTFlagType</i> enumeration.
<i>when</i>	Time that the event occurred in milliseconds since startup
<i>where_x</i>	X coordinate of the mouse cursor location at the time of the event (in screen coordinates)
<i>where_y</i>	Y coordinate of the mouse cursor location at the time of the event (in screen coordinates)
<i>message</i>	Event specific message for the event. For use events this can be any user specific information. For keyboard events this contains the ASCII code in bits 0-7, the keyboard scan code in bits 8-15 and the character repeat count in bits 16-30. You can use the <i>EVT_asciiCode</i> , <i>EVT_scanCode</i> and <i>EVT_repeatCount</i> macros to extract this information from the message field. For mouse events this contains information about which button was pressed, and will be a combination of the flags defined by the <i>MGL_eventMsgMaskType</i> enumeration.
<i>modifiers</i>	Contains additional information about the state of the keyboard shift modifiers (Ctrl, Alt and Shift keys) when the event occurred. For mouse events it will also contain the state of the mouse buttons. Will be a combination of the values defined by the <i>MGL_eventModMaskType</i> enumeration.
<i>next</i>	Internal use; do not use.
<i>prev</i>	Internal use; do not use.

Declaration

```
typedef struct {
    FILE *    (*fopen)(const char *filename,const char
*mode);
    int      (*fclose)(FILE *f);
    int      (*fseek)(FILE *f,long offset,int whence);
    long     (*ftell)(FILE *f);
    size_t   (*fread)(void *ptr,size_t size,size_t n,FILE
*f);
    size_t   (*fwrite)(const void *ptr,size_t size,size_t
n,FILE *f);
} fileio_t
```

Prototype In

mgraph.h

Description

Structure representing the set of file I/O functions that can be overridden in SciTech MGL. When you override the file I/O functions in SciTech MGL, you must provide a compatible function for each of the entries in this structure that behave identically to the standard C library I/O functions of similar names.

Note: *Once you have overridden the file I/O functions, you can access the overridden functions from other libraries and DLL's by calling the MGL_fopen family of functions, which are simply stubs to call the currently overridden function via the function pointers.*

<i>fclose</i>	Standard C fclose function replacement
<i>fseek</i>	Standard C fseek function replacement
<i>ftell</i>	Standard C ftell function replacement
<i>fread</i>	Standard C fread function replacement
<i>fwrite</i>	Standard C fwrite function replacement

fix32_t

Declaration

```
typedef long                fix32_t
```

Prototype In

mgraph.h

Description

Type definition for all standard 32-bit fixed point values used in MGL. Standard fixed point values are 32-bits wide, and represented in 16.16 fixed point format (16 bits of integer, 16 bits of fraction). These numbers can represent signed numbers from +32767.9 to -32768.9.

Note: *If you are doing fixed point arithmetic for screen coordinate calculations, be very careful of overflow conditions when doing multiplication operations.*

Declaration

```
typedef struct {  
    char        name[_MGL_FNAME_SIZE];  
    short       fontType;  
    short       maxWidth;  
    short       maxKern;  
    short       fontWidth;  
    short       fontHeight;  
    short       ascent;  
    short       descent;  
    short       leading;  
} font_t
```

Prototype In

mgraph.h

Description

Structure representing a loaded MGL font file. MGL font files come in two flavors, either vector fonts or bitmap fonts. Vector fonts represent the characters in the font as a set of lines that are drawn, and vector fonts can be scaled and rotated to any desired angle. Vector fonts however do not look very good when rasterized in high resolution. Bitmap fonts represent the characters in the font as small monochrome bitmaps, and can be either fixed width fonts or proportional fonts.

SciTech MGL can load both MGL 1.x style font files (vector and bitmap fonts) or Windows 2.x style bitmap font files. For creating your own font files, you should use any standard Windows font file editor and save the fonts in Windows 2.x format (which is the standard format used by Windows 3.x, Windows 95 and Windows NT for bitmap fonts).

Members

<i>width</i>	Actual width of the character in pixels
<i>fontWidth</i>	Font character width, including any extra padding between this character and the next character. This value is used to advance the current position to the start of the next character, and can be larger than the actual character width (in order to put space between the characters).
<i>FontHeight</i>	Standard height of the font (not including the leading value).
<i>ascent</i>	Font or character ascent value. The ascent value is the number of pixels that the font extends up from the font's baseline.
<i>descent</i>	Font or character descent value. The descent value is the number of pixels that the font extends down from the font's baseline.
<i>Leading</i>	Font leading value. The leading value is the number of vertical pixels of space that are usually required between two lines of text drawn with this font.
<i>kern</i>	Character kern value. The kern value for the character is the number of pixels it extends back past the character origin (such as the tail of the lowercase j character for some fonts).

fxpoint_t

Declaration

```
typedef struct {  
    fix32_t x,y;  
} fxpoint_t
```

Prototype In

mgraph.h

Description

Structure describing a 16.16 fixed point coordinate.

Members

<i>x</i>	Fixed point x coordinate
<i>y</i>	Fixed point y coordinate

Declaration

```
typedef struct {
    int      xRes;
    int      yRes;
    int      bitsPerPixel;
    int      numberOfPlanes;
    color_t  maxColor;
    int      maxPage;
    int      bytesPerLine;
    int      aspectRatio;
    long     pageSize;
    int      scratch1;
    int      scratch2;
    int      redMaskSize;
    int      redFieldPosition;
    int      greenMaskSize;
    int      greenFieldPosition;
    int      blueMaskSize;
    int      blueFieldPosition;
    int      rsvdMaskSize;
    int      rsvdFieldPosition;
    ulong     modeFlags;
} gmode_t
```

Prototype In

mgraph.h

Description

Structure representing the attributes for a specific video mode. This structure is also used to store the rendering dimensions for all device context surfaces in the *MGLDC* structure.

Members

<i>xRes</i>	Device x resolution - 1
<i>yRes</i>	Device y resolution - 1
<i>bitsPerPixel</i>	Pixel depth
<i>numberOfPlanes</i>	Number of planes (always 1 for memory devices)
<i>maxColor</i>	Maximum color for device - 1
<i>maxPage</i>	Maximum number of hardware display pages - 1
<i>bytesPerLine</i>	Number of bytes in a single device scanline
<i>aspectRatio</i>	Device pixel aspect ratio ((horiz/vert) * 1000)
<i>pageSize</i>	Number of bytes in a hardware display page
<i>scratch1</i>	Internal scratch value
<i>scratch2</i>	Internal scratch value
<i>redMaskSize</i>	Size of RGB red mask
<i>redFieldPosition</i>	Number of bits in RGB red field
<i>greenMaskSize</i>	Size of RGB green mask
<i>greenFieldPosition</i>	Number of bits in RGB green field
<i>blueMaskSize</i>	Size of RGB blue mask
<i>blueFieldPosition</i>	Number of bits in RGB blue field
<i>rsvdMaskSize</i>	Size of RGB reserved mask
<i>rsvdFieldPosition</i>	Number of bits in RGB reserved field

Declaration

```
typedef struct {  
    int        byteWidth;  
    uchar      *andMask;  
    bitmap_t   xorMask;  
  
    } icon_t
```

Prototype In

mgraph.h

Description

Structure representing a loaded icon. Icons are used by SciTech MGL to display small, transparent bitmap images that can be of any dimension. The standard Windows .ICO files can store icons in 32x32 and 64x64 formats, although SciTech MGL can load icons of any dimensions if you can find an editor that will allow you to create large icons.

Icons are always drawn by SciTech MGL by first using the icon AND mask to punch a hole in the background of the display surface, and then the icon bitmap XOR mask is XOR'ed into the display surface. This method is compatible with the way that Microsoft Windows displays icons on the screen.

Members

<i>byteWidth</i>	Width of the monochrome AND mask in bytes. Must be consistent with the bitmap width in the xorMask structure.
<i>andMask</i>	Pointer to the AND mask information, which is stored contiguously in memory after the header block. The dimensions of the AND mask is defined by the dimensions of the xorMask bitmap image.
<i>xorMask</i>	Bitmap image header block, containing information about the mask used to draw the icon image. The actual bitmap surface and palette data is stored contiguously in memory after the header block.

Declaration

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

Prototype In

mgraph.h

Description

Structure representing text metrics for a font or a single character, in the current text attributes. For bitmap fonts you can get all the metric information from the *font_t* structure, however for vector fonts, this routine will provide the proper metrics for the font after being scaled by the current font character scaling size. This structure is also used to obtain specified 'tightest fit' metrics information about any character in the font.

Members

width

Actual width of the character in pixels

fontWidth

Font character width, including any extra padding between this character and the next character. This value is used to advance the current position to the start of the next character, and can be larger than the actual character width (in order to put space between the characters).

fontHeight

Standard height of the font (not including the leading value).

ascent

Font or character ascent value. The ascent value is the number of pixels that the font extends up from the font's baseline.

descent

Font or character descent value. The descent value is the number of pixels that the font extends down from the font's baseline.

leading

Font leading value. The leading value is the number of vertical pixels of space that are usually required between two lines of text drawn with this font.

kern

Character kern value. The kern value for the character is the number of pixels it extends back past the character origin (such as the tail of the lowercase j character for some fonts).

Declaration

```
typedef struct {  
    uchar    blue;  
    uchar    green;  
    uchar    red;  
    uchar    alpha;  
} palette_t
```

Prototype In

mgraph.h

Description

Structure representing a single color palette entry. Color palette entries are used to build the color lookup tables for all the device contexts used in SciTech MGL, which are used to define the final color values for colors in color index modes (8-bits per pixel and below). Color palette information is always stored in 8-bits per primary format (i.e.: 8-bits of red, green and blue information), and will be converted by MGL to the appropriate color format used by the underlying hardware when the hardware palette is programmed. Hence in standard VGA modes (which only use 6-bits per primary) the bottom two bits of color information will be lost when the palette is programmed.

Members

<i>blue</i>	Blue channel color (0 - 255)
<i>green</i>	Green channel color (0 - 255)
<i>red</i>	Red channel color (0 - 255)
<i>alpha</i>	Alignment value (not used and should always be 0)

Declaration

```
typedef uchar  pattern_t[8]
```

Prototype In

mgraph.h

Description

Type definition for an 8x8 monochrome bitmap pattern. This is used to specify the monochrome bitmap pattern used for filling solid objects when the pen style is MGL_BITMAP_OPAQUE or MGL_BITMAP_TRANSPARENT.

When the pen style is MGL_BITMAP_OPAQUE, where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the background color is used.

When the pen style is MGL_BITMAP_TRANSPARENT, where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the pixel is left unmodified on the screen.

Declaration

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

Prototype In mgraph.h

Description

Structure representing the format of an RGB pixel. This structure is used to describe the current RGB pixel format used by all MGL device contexts with pixel depths greater than or equal to 15-bits per pixel. The pixel formats for 15 and 16-bit modes are constant and never change, however there are 2 possible pixel formats for 24 bit RGB modes and 4 possible formats for 32 bit RGB modes that are supported by SciTech MGL. The possible modes for 24-bits per pixel are:

RGB

Values are packed with Red in byte 2, Green in byte 1 and Blue in byte 0. This is the standard format used by all 24 bit Windows BMP files, and the native display format for most graphics hardware on the PC.

BGR

Values are packed with Blue in byte 2, Green in byte 1 and Red in byte 0. This format is the native display format for some graphics hardware on the PC.

The possible modes for 32-bits per pixel are:

32-bit	Description
ARGB	Values are packed with Red in byte 2, Green in byte 1 and Blue in byte 0 and alpha in byte 3.
ABGR	Values are packed with Blue in byte 2, Green in byte 1 and Red in byte 0 and alpha in byte 3.
RGBA	Values are packed with Red in byte 3, Green in byte 2 and Blue in byte 1 and alpha in byte 0.
BGRA	Values are packed with Blue in byte 3, Green in byte 2 and Red in byte 1 and alpha in byte 0.

If you intend to write your own direct rendering code for 15-bits per pixel and higher graphics modes, you will need to write your code so that it will adapt to the underlying pixel format used by the hardware to display the correct colors on the screen. SciTech MGL will perform pixel format translation on the fly for *MGL_bitBlit* operations, but this can be time consuming. The formula for packing the pixel data into the proper positions given three 8-bit RGB values is as follows:

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

Alternatively you can unpack the color values with the following code:

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

Note: For 32-bit modes, the alpha channel information is unused, but should always be set to zero. Some hardware devices interpret the alpha channel information so unless you use a value of zero, you will get some strange looking results on the screen.

Members

redMask

Unshifted 8 bit mask for the red color channel, and will be 5-bits wide for a 5-bit color channel or 8-bits wide for an 8-bit color channel.

greenMask

Unshifted 8 bit mask for the green color channel.

blueMask

Unshifted 8 bit mask for the blue color channel.

rsvdMask

Unshifted 8 bit mask for the reserved or alpha channel.

redPos

Bit position for bit 0 of the red color channel information.

redAdjust

Number of bits to shift the 8 bit red value right

greenPos

Bit position for bit 0 of the green color channel information

greenAdjust

Number of bits to shift the 8 bit green value right

bluePos

Bit position for bit 0 of the blue color channel information.

blueAdjust

Number of bits to shift the 8 bit blue value right

rsvdPost

Bit position for bit 0 of the reserved channel information

rsvdAdjust

Number of bits to shift the 32 bit reserved value right

pixpattern_t

Declaration

```
typedef color_t pixpattern_t[8][8]
```

Prototype In

mgraph.h

Description

Type definition for an 8x8 color pixmap pattern. This is used to specify the color pixmap pattern used for filling solid objects when the pen style is in MGL_PIXMAP mode. The pixmap pattern is defined as an 8x8 array of *color_t* values, where each entry corresponds to the color values packed for the appropriate color mode (i.e.: a color index in color map modes and a packed RGB value in HiColor and TrueColor modes).

point_t

Declaration

```
typedef struct {  
    int x,y;  
} point_t
```

Prototype In

mgraph.h

Description

Structure describing an integer point passed to SciTech MGL.

Members

x	x coordinate for the point
----------	-----------------------------------

Declaration

```
typedef struct {  
    int left;  
    int top;  
    int right;  
    int bottom;  
} rect_t
```

Prototype In

mgraph.h

Description

Structure describing an integer rectangle. Note that MGL defines and uses rectangles such that the bottom and right coordinates are not actually included in the pixels that define a raster coordinate rectangle. This allows for correct handling of overlapping rectangles without drawing any pixels twice.

Members

<i>left</i>	Left coordinate of the rectangle
<i>top</i>	Top coordinate of the rectangle
<i>right</i>	Right coordinate of the rectangle
<i>bottom</i>	Bottom coordinate of the rectangle

region_t

Declaration

```
typedef struct {  
    rect_t      rect;  
    span_t      *spans;  
} region_t
```

Prototype In

mgraph.h

Description

Structure representing a complex region. Complex regions are used to represent non-rectangular areas as unions of smaller rectangles (the smallest being a single pixel). You can use complex regions to build complex clipping regions for user interface library development (such as the SciTech MegaVision Library which makes extensive use of SciTech MGL's region management functions).

Members

rect

Bounding rectangle for the region. If the spans field below is NULL, then the region is a simple region and is composed of only a single rectangle. Note however that you can have a simple region that consists of only single rectangle in the span structure (usually after complex region arithmetic). You can use the *MGL_isSimpleRegion* function to determine if the region contains only a single rectangle.

spans

Pointer to the internal region span structure.

Declaration

```
typedef struct {  
    int             horizJust;  
    int             vertJust;  
    int             dir;  
    int             szNumerx;  
    int             szNumery;  
    int             szDenomx;  
    int             szDenomy;  
    int             spaceExtra;  
    font_t          *font;  
} text_settings_t
```

Prototype In

mgraph.h

Description

Structure representing the current text rasterizing settings. This structure is used to group all these settings together in SciTech MGL, and allows you to save and restore the text rendering settings as a single unit.

Members

<i>horizJust</i>	Horizontal justification value. Will be one of the values defined by the <i>MGL_textJustType</i> enumeration.
<i>vertJust</i>	Vertical justification value. Will be one of the values defined by the <i>MGL_textJustType</i> enumeration.
<i>dir</i>	Current text direction value. Will be one of the values defined by the <i>MGL_textDirType</i> enumeration.
<i>szNumerx</i>	Current text x size numerator value
<i>szNumery</i>	Current text y size numerator value
<i>szDenomx</i>	Current text x size denominator value
<i>szDenomy</i>	Current text y size denominator value
<i>spaceExtra</i>	Current text space extra value. The space extra value is the number of extra pixels to be added to every space character when rendering the line of text.
<i>Font</i>	Pointer to current active font loaded in memory.

A

arc_coords_t, 169, 206, 436, 654
attributes_t, 208, 427, 609, 655

B

bitmap_t, 111, 210, 325, 327, 338, 340, 344, 346,
399, 400, 401, 402, 404, 531, 532, 554, 580,
581, 657, 670

C

color_t, 111, 143, 209, 214, 221, 245, 259, 260,
261, 262, 358, 383, 384, 385, 386, 387, 400,
402, 404, 411, 430, 431, 432, 433, 454, 455,
457, 458, 469, 471, 477, 490, 541, 542, 544,
545, 558, 559, 560, 561, 577, 581, 609, 610,
641, 655, 659, 668, 676, 678
CPU_getProcessorSpeed, 20, 21, 22
CPU_getProcessorType, 20, 21, 22
CPU_haveMMX, 20, 21, 22
CPU_largeInteger, 600
CPU_processorType, 601
cursor_t, 331, 332, 555, 576, 660

E

event_t, 23, 25, 27, 29, 32, 59, 62, 63, 64, 71, 72,
73, 514, 605, 623, 625, 661
EVT_asciiCode, 23, 31, 32, 605, 662
EVT_flush, 24, 26, 29, 30
EVT_getNext, 24, 25, 27, 29, 30, 424, 515
EVT_halt, 24, 26, 27, 29, 30
EVT_isKeyDown, 28
EVT_peekNext, 24, 26, 27, 29, 30, 515
EVT_post, 30
EVT_repeatCount, 23, 31, 32, 662
EVT_scanCode, 23, 32, 662
EVT_setTimerTick, 26, 33

F

index, 484, 663

G

GM_chooseMode, 34
GM_cleanup, 35, 48, 49, 50, 51
GM_driverOptions, 57, 604
GM_exit, 36, 49
GM_findMode, 37
GM_getDoDraw, 38
GM_getExitMainLoop, 39
GM_getHaveWin31, 40
GM_getHaveWin95, 41
GM_getHaveWinNT, 42
GM_init, 43, 45, 46, 47, 49, 54, 55, 56, 58, 60, 61,
62, 65, 67, 68, 70, 71, 72, 73, 75, 77, 602, 604,
606
GM_initPath, 45
GM_initSysPalNoStatic, 46, 55
GM_initWindowPos, 47
GM_keyCodes, 605
GM_mainLoop, 35, 36, 48, 50, 51, 56, 59, 60, 61,
62, 63, 64, 71, 72, 73
GM_modeFlagsType, 604, 606
GM_modeInfo, 34, 37, 65, 69, 75, 602, 607
GM_processEvents, 35, 48, 49, 50, 51
GM_processEventsWin, 35, 48, 49, 50, 51
GM_realizePalette, 52, 74
GM_registerEventProc, 48, 49, 53, 54
GM_registerMainWindow, 54
GM_setAppActivate, 55
GM_setDrawFunc, 43, 44, 49, 56, 61
GM_setDriverOptions, 44, 57, 604
GM_setEventFunc, 59
GM_setExitFunc, 60
GM_setGameLogicFunc, 43, 44, 49, 61
GM_setKeyDownFunc, 43, 62, 63, 64
GM_setKeyRepeatFunc, 62, 63, 64
GM_setKeyUpFunc, 62, 63, 64
GM_setMode, 43, 44, 65, 70, 75, 78, 607, 608
GM_setModeFilterFunc, 68
GM_setModeSwitchFunc, 66, 67, 69, 75
GM_setMouseDownFunc, 71, 72, 73
GM_setMouseMoveFunc, 71, 72, 73
GM_setMouseUpFunc, 71, 72, 73
GM_setPalette, 52, 74
GM_setPreModeSwitchFunc, 75
GM_setSuspendAppCallback, 55, 76

I

icon_t, 335, 337, 407, 557, 670

L

LZTimerCount, 81, 82, 83, 84
LZTimerLap, 81, 82, 83, 84
LZTimerOff, 81, 82, 83, 84
LZTimerOn, 81, 82, 83, 84

M

metrics_t, 216, 234, 671
MGL_activatePalette, 89, 134, 139, 522
MGL_appActivate, 90
MGL_availableBitmap, 91, 326, 329
MGL_availableCursor, 92, 331
MGL_availableFont, 93, 333
MGL_availableIcon, 94, 336
MGL_availableJPEG, 95, 339, 342
MGL_availableMemory, 96, 565
MGL_availableModes, 97, 99, 148, 149, 639
MGL_availablePages, 97, 99, 126, 148, 149
MGL_availablePCX, 98, 345, 348
MGL_backfacing, 100
MGL_bdrStyleType, 615
MGL_beep, 101
MGL_beginDirectAccess, 102, 175, 630
MGL_beginPixel, 103, 176, 259, 260, 261, 262, 390, 391
MGL_bitBlt, 104, 106, 107, 109, 111, 112, 117, 126, 127, 130, 138, 522, 530, 534, 541, 543, 676
MGL_bitBltCoord, 104, 105, 530, 534
MGL_bitBltLin, 108, 110, 544, 546
MGL_bitBltLinCoord, 108, 109
MGL_buildMonoMask, 111
MGL_calloc, 113, 200, 354, 565
MGL_changeDisplayMode, 114, 125, 126, 127, 133, 134, 135, 137, 639
MGL_charWidth, 116, 357
MGL_checkIdentityPalette, 106, 117, 534
MGL_clearDevice, 118, 120, 209, 469
MGL_clearRegion, 119, 123
MGL_clearViewport, 118, 120, 209, 272, 273, 469, 503, 504, 516, 517
MGL_clipLineFX, 121
MGL_clipType, 616

MGL_colorModes, 222, 479, 617
MGL_COLORS, 613
MGL_computePixelAddr, 122
MGL_copyRegion, 119, 123, 160, 201
MGL_createCustomDC, 124
MGL_createDisplayDC, 114, 115, 125, 128, 130, 132, 133, 134, 135, 137, 139, 145, 155, 228
MGL_createLinearOffscreenDC, 128, 132, 145
MGL_createMemoryDC, 127, 129, 134, 139, 145
MGL_createOffscreenDC, 127, 128, 131, 145, 226
MGL_createScrollingDC, 133, 483
MGL_createStereoDisplayDC, 127, 135, 470, 507, 528, 529, 644
MGL_createWindowedDC, 127, 130, 138, 145
MGL_defaultAttributes, 142
MGL_defaultColor, 143
MGL_defRect, 140, 141
MGL_defRectPt, 140, 141
MGL_delay, 144
MGL_destroyDC, 114, 115, 127, 128, 130, 132, 134, 137, 139, 145
MGL_detectGraph, 97, 99, 146, 305, 306, 372, 417, 418, 423, 618, 634
MGL_diffRegion, 150, 151, 160, 174, 313, 379, 381, 382, 463, 464, 547, 551, 552, 553
MGL_diffRegionRect, 150, 151
MGL_disjointRect, 152
MGL_divotSize, 153, 154, 230, 231, 406
MGL_divotSizeCoord, 153, 154
MGL_doubleBuffer, 155, 524, 537
MGL_drawBorder, 156, 157, 214, 471
MGL_drawBorderCoord, 156, 157, 615
MGL_drawGlyph, 158, 370
MGL_drawHDivider, 159
MGL_drawRegion, 160
MGL_drawStr, 161, 162, 215, 265, 266, 506, 510, 539, 540, 564, 568
MGL_drawStrXY, 161, 162, 548
MGL_drawVDivider, 163
MGL_driverName, 164, 232, 371, 372, 373, 374
MGL_driverType, 146, 306, 618
MGL_ellipse, 165, 166, 168, 170, 186, 187, 188, 189
MGL_ellipseArc, 165, 166, 167, 168, 170, 186, 187, 188, 189, 206
MGL_ellipseArcCoord, 167
MGL_ellipseArcEngine, 169, 172, 319
MGL_ellipseCoord, 165, 170

MGL_ellipseEngine, 169, 171, 319
MGL_emptyRect, 152, 173
MGL_emptyRegion, 174
MGL_endDirectAccess, 175, 630
MGL_endPixel, 103, 176, 259, 260, 261, 262, 390, 391
MGL_equalPoint, 177, 178
MGL_equalRect, 152, 178
MGL_equalRegion, 174, 178, 179
MGL_errorMsg, 180, 306, 308, 620
MGL_errorType, 428, 505, 620
MGL_eventMaskType, 622, 625
MGL_eventModMaskType, 623, 662
MGL_eventMsgMaskType, 624, 662
MGL_eventType, 25, 29, 625
MGL_exit, 60, 115, 125, 145, 181
MGL_fadePalette, 182, 448
MGL_fatalError, 184
MGL_fclose, 166, 168, 185, 187, 188
MGL_fillEllipse, 165, 166, 168, 170, 186, 187, 188
MGL_fillEllipseArc, 165, 166, 168, 170, 186, 187, 188, 189, 206
MGL_fillEllipseArcCoord, 187, 188
MGL_fillEllipseCoord, 186, 189
MGL_fillPolygon, 190, 264, 502
MGL_fillPolygonCnvx, 191
MGL_fillPolygonCnvxFX, 191, 192
MGL_fillPolygonFX, 190, 192, 193
MGL_fillRect, 120, 195, 196, 197
MGL_fillRectCoord, 195, 196, 197
MGL_fillRectPt, 195, 196, 197
MGL_FixDiv, 85, 87, 88
MGL_FixMul, 85, 86, 88, 100
MGL_FixMulDiv, 85, 87, 88
MGL_fontType, 626, 666
MGL_fopen, 96, 198, 663
MGL_fread, 199
MGL_free, 113, 200, 354, 565
MGL_freeRegion, 119, 123, 160, 201, 379
MGL_fseek, 202, 203
MGL_ftell, 203
MGL_fwrite, 204
MGL_getActivePage, 205, 276, 466, 521
MGL_getArcCoords, 206
MGL_getAspectRatio, 207, 468
MGL_getAttributes, 142, 208, 427
MGL_getBackColor, 209, 221, 469, 477
MGL_getBitmapFromDC, 210
MGL_getBitmapSize, 211, 212, 326, 329
MGL_getBitmapSizeExt, 212
MGL_getBitsPerPixel, 213
MGL_getBorderColors, 214
MGL_getCharMetrics, 216, 234, 357, 539, 540
MGL_getClipMode, 217, 218, 219, 220, 473, 474
MGL_getClipModeDC, 217, 218
MGL_getClipRect, 217, 218, 219, 220, 475, 476
MGL_getClipRectDC, 219, 220
MGL_getColor, 143, 209, 221, 469, 477
MGL_getColorMapMode, 222, 479
MGL_getCP, 215, 279, 280
MGL_getDefaultPalette, 142, 223, 251, 481
MGL_getDirectDrawActiveSurface, 224, 228
MGL_getDirectDrawObject, 225
MGL_getDirectDrawOffscreenSurface, 224, 226, 228
MGL_getDirectDrawPalette, 227
MGL_getDirectDrawPrimarySurface, 224, 225, 226, 227, 228
MGL_getDisplayStart, 229, 483
MGL_getDivot, 153, 154, 230, 231, 406
MGL_getDivotCoord, 230, 231
MGL_getDriver, 232
MGL_getFont, 233
MGL_getFontMetrics, 216, 234, 357, 539, 540
MGL_getFullScreenWindow, 235
MGL_getGlyphHeight, 236, 237
MGL_getGlyphWidth, 236, 237
MGL_getHalfTonePalette, 238, 301, 617
MGL_getHardwareFlags, 239
MGL_getJPEGSize, 240, 241, 339, 342
MGL_getJPEGSizeExt, 241
MGL_getLineStipple, 242, 485
MGL_getLineStippleCount, 243, 486
MGL_getLineStyle, 244, 488
MGL_getMarkerColor, 245, 490
MGL_getMarkerSize, 246, 247, 356, 491
MGL_getMarkerStyle, 246, 247, 356, 492
MGL_getMode, 232, 248
MGL_getPalette, 183, 223, 251, 252, 253, 411, 448, 481, 495, 496
MGL_getPaletteEntry, 251, 252, 496
MGL_getPaletteSize, 223, 251, 253
MGL_getPaletteSnowLevel, 254, 497
MGL_getPCXSize, 249, 250, 345, 348
MGL_getPCXSizeExt, 250
MGL_getPenBitmapPattern, 255, 498
MGL_getPenPixmapPattern, 256, 499
MGL_getPenSize, 257, 500

MGL_getPenStyle, 255, 256, 258, 501
MGL_getPixel, 103, 259, 260
MGL_getPixelCoord, 259, 260
MGL_getPixelCoordFast, 261, 262
MGL_getPixelFast, 261, 262
MGL_getPixelFormat, 130, 213, 263, 558, 559
MGL_getPolygonType, 264, 502
MGL_getSpaceExtra, 265, 506
MGL_getTextDirection, 266, 510
MGL_getTextJustify, 267, 511
MGL_getTextSettings, 268, 512
MGL_getTextSize, 269, 513
MGL_getTickResolution, 270, 271
MGL_getTicks, 270, 271, 527
MGL_getViewport, 272, 273, 475, 476, 503, 504, 516, 517
MGL_getViewportDC, 272, 273
MGL_getViewportOrg, 274, 275, 518, 519
MGL_getViewportOrgDC, 274, 275
MGL_getVisualPage, 205, 276, 466, 521
MGL_getWinDC, 125, 277
MGL_getWriteMode, 278, 523
MGL_getX, 279, 280
MGL_getY, 279, 280
MGL_glChooseVisual, 78, 281, 282, 283, 296, 297, 612
MGL_glContextFlagsType, 78, 627
MGL_glCreateContext, 78, 281, 282, 284, 287, 288, 290, 296, 297, 612, 627
MGL_glDeleteContext, 284, 290
MGL_glEnumerateDrivers, 285, 293, 294
MGL_glGetProcAddress, 286
MGL_glGetVisual, 288
MGL_glHaveHWOpenGL, 289
MGL_glMakeCurrent, 78, 283, 284, 290
MGL_globalToLocal, 299, 300, 350, 351
MGL_globalToLocalDC, 299, 300
MGL_glOpenGLType, 294, 604, 629
MGL_glRealizePalette, 291, 295
MGL_glResizeBuffers, 292
MGL_glSetDriver, 285, 293, 294
MGL_glSetOpenGLType, 285, 293, 294
MGL_glSetPalette, 291, 295
MGL_glSetVisual, 78, 281, 282, 283, 288, 296, 612, 627, 628
MGL_glSwapBuffers, 298, 652
MGL_halfTonePixel, 301
MGL_halfTonePixel555, 302, 303
MGL_halfTonePixel565, 302, 303
MGL_hardwareFlagsType, 239, 630
MGL_haveWidePalette, 304
MGL_init, 97, 99, 115, 125, 127, 128, 132, 133, 135, 147, 149, 181, 248, 305, 307, 308, 325, 329, 331, 333, 335, 339, 342, 344, 348, 423, 429, 467, 472, 563, 618, 639
MGL_initWindowed, 134, 139, 181, 306, 307, 472
MGL_insetRect, 309, 380
MGL_isCurrentDC, 310, 352
MGL_isDisplayDC, 311, 312, 314
MGL_isMemoryDC, 311, 312, 314
MGL_isSimpleRegion, 313, 681
MGL_isWindowedDC, 311, 312, 314
MGL_leftTop, 315, 446
MGL_line, 316, 317
MGL_lineCoord, 316, 317
MGL_lineCoordFX, 318, 320
MGL_lineEngine, 169, 172, 319
MGL_lineFX, 317, 318, 320
MGL_lineRel, 215, 321, 322
MGL_lineRelCoord, 321, 322
MGL_lineStyleType, 632, 656
MGL_lineTo, 215, 323, 324
MGL_lineToCoord, 323, 324
MGL_loadBitmap, 45, 91, 210, 211, 325, 327, 329, 340, 346, 399, 400, 401, 403, 405, 449, 531, 532, 554, 657
MGL_loadBitmapExt, 211, 326, 327
MGL_loadBitmapIntoDC, 211, 326, 327, 328, 330, 343, 349, 449
MGL_loadBitmapIntoDCExt, 329, 330
MGL_loadCursor, 92, 331, 332, 555
MGL_loadCursorExt, 331, 332, 556
MGL_loadFont, 45, 93, 233, 333, 334, 564
MGL_loadFontExt, 334
MGL_loadIcon, 94, 335, 337, 407, 557
MGL_loadIconExt, 336, 337
MGL_loadJPEG, 240, 338, 340, 342
MGL_loadJPEGExt, 339, 340
MGL_loadJPEGIntoDC, 240, 338, 339, 341, 343, 451
MGL_loadJPEGIntoDCExt, 343
MGL_loadPCX, 249, 344, 346, 348
MGL_loadPCXExt, 345, 346
MGL_loadPCXIntoDC, 249, 345, 347, 349, 453
MGL_loadPCXIntoDCExt, 349
MGL_localToGlobal, 299, 300, 350, 351
MGL_localToGlobalDC, 350, 351
MGL_makeCurrentDC, 310, 352
MGL_makeSubDC, 353

MGL_malloc, 96, 113, 200, 230, 231, 354, 565
MGL_mapToPalette, 355
MGL_marker, 245, 246, 247, 356, 490, 491, 492
MGL_markerStyleType, 247, 356, 492, 633, 656
MGL_maxCharWidth, 357
MGL_maxColor, 358, 494
MGL_maxPage, 359
MGL_maxx, 360, 361, 362, 363, 525, 526
MGL_maxxDC, 360, 361
MGL_maxy, 360, 361, 362, 363, 525, 526
MGL_maxyDC, 362, 363
MGL_memcpy, 364, 365, 366
MGL_memcpyVIRTDEST, 364, 365, 366
MGL_memcpyVIRTsrc, 364, 365, 366
MGL_memset, 113, 200, 354, 367, 368, 369
MGL_memsetl, 367, 368, 369
MGL_memsetw, 367, 368, 369
MGL_mirrorGlyph, 158, 370
MGL_modeDriverName, 164, 371
MGL_modeFlags, 127, 372, 634
MGL_modeFlagsType, 372, 634
MGL_modeName, 164, 248, 371, 372, 373, 374
MGL_modeResolution, 149, 372, 374, 639
MGL_modeType, 97, 637
MGL_moveRel, 215, 375, 376
MGL_moveRelCoord, 321, 322, 375, 376
MGL_moveTo, 215, 377, 378
MGL_moveToCoord, 377, 378
MGL_newRegion, 119, 123, 160, 201, 379
MGL_offsetRect, 309, 380
MGL_offsetRegion, 174, 381
MGL_optimizeRegion, 382
MGL_packColor, 263, 383, 384, 431, 469, 477, 480, 490, 558, 559, 560, 561, 659
MGL_packColorFast, 383, 384
MGL_packColorRGB, 383, 384, 385, 386, 387, 560, 561
MGL_packColorRGBFast, 385, 386, 387
MGL_packColorRGBFast2, 387
MGL_palRotateType, 448, 640
MGL_penStyleType, 258, 456, 501, 641, 656
MGL_pixel, 103, 388, 389
MGL_pixelCoord, 388, 389
MGL_pixelCoordFast, 390, 391
MGL_pixelFast, 388, 390, 391
MGL_polygonType, 264, 502, 642, 656
MGL_polyLine, 392, 393, 394
MGL_polyMarker, 245, 246, 247, 356, 392, 393, 394, 490, 491, 492
MGL_polyPoint, 392, 393, 394
MGL_ptInRect, 395, 396
MGL_ptInRectCoord, 396
MGL_ptInRegion, 174, 397, 398
MGL_ptInRegionCoord, 174, 397, 398
MGL_putBitmap, 326, 339, 345, 399, 585, 657
MGL_putBitmapMask, 111, 112, 400
MGL_putBitmapSection, 401
MGL_putBitmapTransparent, 402, 405, 657
MGL_putBitmapTransparentSection, 403, 404
MGL_putDivot, 153, 154, 230, 231, 406
MGL_putIcon, 336, 407
MGL_putMonoImage, 408
MGL_random, 409, 410, 527
MGL_randoml, 409, 410, 527
MGL_realColor, 127, 130, 411, 431, 478
MGL_realizePalette, 182, 183, 238, 291, 412, 448, 494, 495, 496
MGL_rect, 195, 196, 197, 413, 414, 415
MGL_rectCoord, 413, 414, 415
MGL_rectPt, 413, 414, 415
MGL_refreshRateType, 643
MGL_registerAllDispDrivers, 305, 416, 418
MGL_registerAllDispDriversExt, 416, 417
MGL_registerAllMemDrivers, 419
MGL_registerAllOpenGLDrivers, 420
MGL_registerDriver, 130, 146, 148, 305, 418, 419, 420, 421
MGL_registerEventProc, 235, 424
MGL_registerFullScreenWindow, 425
MGL_resizeWinDC, 426
MGL_restoreAttributes, 142, 208, 427
MGL_result, 128, 131, 145, 180, 210, 306, 308, 326, 329, 331, 333, 335, 339, 342, 344, 348, 428, 449, 450, 452, 505, 620
MGL_resume, 429, 536, 563
MGL_rgbBlue, 430, 432, 433
MGL_rgbColor, 431, 480
MGL_rgbGreen, 430, 432, 433
MGL_rgbRed, 430, 432, 433
MGL_rgnEllipse, 434, 435, 442
MGL_rgnEllipseArc, 434, 435, 436
MGL_rgnGetArcCoords, 435, 436, 442
MGL_rgnLine, 437, 438, 439, 440
MGL_rgnLineCoord, 437, 438
MGL_rgnLineCoordFX, 439, 440
MGL_rgnLineFX, 437, 438, 439, 440
MGL_rgnSolidEllipse, 441
MGL_rgnSolidEllipseArc, 441, 442
MGL_rgnSolidRect, 443, 444, 445
MGL_rgnSolidRectCoord, 443, 444, 445

MGL_rgnSolidRectPt, 443, 444, 445
 MGL_rightBottom, 315, 446
 MGL_rotateGlyph, 158, 370, 447
 MGL_rotatePalette, 183, 448
 MGL_saveBitmapFromDC, 210, 326, 329, 449
 MGL_saveJPEGFromDC, 339, 342, 450
 MGL_savePCXFromDC, 345, 348, 452
 MGL_scanLeftForColor, 454, 455, 457, 458
 MGL_scanLeftWhileColor, 454, 455, 457, 458
 MGL_scanLine, 456
 MGL_scanRightForColor, 454, 455, 457, 458
 MGL_scanRightWhileColor, 454, 455, 457, 458
 MGL_sectRect, 152, 459, 460, 461, 462, 549, 550
 MGL_sectRectCoord, 459, 460, 461
 MGL_sectRectFast, 459, 461
 MGL_sectRectFastCoord, 460, 462
 MGL_sectRegion, 150, 151, 160, 174, 313, 379, 381, 382, 463, 464, 547, 551, 552, 553
 MGL_sectRegionRect, 463, 464
 MGL_setACCELDriver, 465
 MGL_setActivePage, 136, 205, 276, 466, 521, 644
 MGL_setAppInstance, 467
 MGL_setAspectRatio, 207, 468
 MGL_setBackColor, 209, 221, 469, 477
 MGL_setBlueCodeIndex, 137, 470, 507
 MGL_setBorderColors, 471
 MGL_setBufSize, 472
 MGL_setClipMode, 217, 218, 473, 474, 616
 MGL_setClipModeDC, 473, 474
 MGL_setClipRect, 217, 218, 219, 220, 272, 273, 475, 476, 503, 504, 516, 517
 MGL_setClipRectDC, 475, 476
 MGL_setColor, 143, 209, 221, 469, 477, 478, 480
 MGL_setColorCI, 127, 130, 411, 431, 478, 480
 MGL_setColorMapMode, 222, 238, 479, 617
 MGL_setColorRGB, 411, 431, 478, 480
 MGL_setDefaultPalette, 481
 MGL_setDisplayStart, 134, 229, 482, 521
 MGL_setFileIO, 484
 MGL_setLineStipple, 242, 243, 244, 485, 486, 488
 MGL_setLineStippleCount, 242, 243, 244, 485, 486, 488
 MGL_setLineStyle, 242, 243, 244, 485, 486, 487, 632
 MGL_setMainWindow, 489
 MGL_setMarkerColor, 245, 490
 MGL_setMarkerSize, 246, 247, 356, 491
 MGL_setMarkerStyle, 246, 247, 356, 492, 633

MGL_setOpenGLFuncs, 493
 MGL_setPalette, 183, 223, 238, 251, 252, 295, 355, 411, 412, 448, 481, 494, 496, 497
 MGL_setPaletteEntry, 252, 495, 496
 MGL_setPaletteSnowLevel, 254, 412, 497
 MGL_setPenBitmapPattern, 255, 256, 258, 456, 498, 499, 501
 MGL_setPenBitmapPattern, 255, 256, 258, 456, 498, 499
 MGL_setPenSize, 257, 500
 MGL_setPenStyle, 255, 256, 258, 498, 499, 501, 641
 MGL_setPolygonType, 190, 193, 194, 264, 502, 642
 MGL_setRelViewport, 272, 273, 503, 504, 516, 517
 MGL_setRelViewportDC, 504
 MGL_setResult, 428, 505
 MGL_setSpaceExtra, 265, 506
 MGL_setStereoSyncType, 137, 470, 507, 645
 MGL_setSuspendAppCallback, 76, 508
 MGL_setTextDirection, 266, 510, 650
 MGL_setTextJustify, 267, 511, 651
 MGL_setTextSettings, 268, 512
 MGL_setTextSize, 269, 513
 MGL setUserEventFilter, 514
 MGL_setViewport, 272, 273, 274, 275, 475, 476, 503, 504, 516, 517, 518, 519
 MGL_setViewportDC, 517
 MGL_setViewportOrg, 272, 273, 274, 275, 503, 504, 516, 517, 518, 519
 MGL_setViewportOrgDC, 518, 519
 MGL_setVisualPage, 205, 276, 412, 466, 482, 520, 652
 MGL_setWinDC, 89, 134, 139, 522
 MGL_setWriteMode, 278, 523
 MGL_singleBuffer, 155, 524, 537
 MGL_sizeX, 360, 361, 362, 363, 525, 526
 MGL_sizeY, 360, 361, 362, 363, 525, 526
 MGL_srand, 409, 410, 527
 MGL_startStereo, 137, 470, 507, 528, 529
 MGL_stereoBufType, 644
 MGL_stereoSyncType, 507, 645
 MGL_stopStereo, 137, 470, 507, 528, 529
 MGL_stretchBitmap, 531, 657
 MGL_stretchBitmapSection, 532
 MGL_stretchBLT, 530
 MGL_stretchBlitCoord, 530, 533
 MGL_surfaceAccessFlagsType, 535, 569, 646
 MGL_surfaceAccessType, 126, 131, 535, 610

MGL_suspend, 429, 536, 563
 MGL_suspendAppCodesType, 647
 MGL_suspendAppFlagsType, 649
 MGL_swapBuffers, 155, 412, 466, 521, 524, 537, 652
 MGL_textBounds, 538
 MGL_textDirType, 447, 650, 682
 MGL_textHeight, 116, 161, 162, 357, 538, 539, 540
 MGL_textJustType, 266, 267, 510, 511, 651, 682
 MGL_textWidth, 116, 161, 162, 357, 538, 539, 540, 548
 MGL_transBlit, 104, 107, 541, 543, 545
 MGL_transBlitCoord, 541, 542
 MGL_transBlitLin, 108, 110, 112, 544, 546
 MGL_transBlitLinCoord, 544, 545
 MGL_traverseRegion, 547
 MGL_underScoreLocation, 548
 MGL_unionRect, 152, 459, 460, 461, 462, 549
 MGL_unionRectCoord, 549, 550
 MGL_unionRegion, 150, 151, 160, 174, 313, 379, 381, 382, 463, 464, 547, 551, 552, 553
 MGL_unionRegionOfs, 551, 552, 553
 MGL_unionRegionRect, 551, 552, 553
 MGL_unloadBitmap, 326, 339, 345, 554, 580, 581, 586
 MGL_unloadCursor, 331, 555
 MGL_unloadFont, 233, 333, 556, 564
 MGL_unloadIcon, 336, 557
 MGL_unpackColor, 263, 383, 384, 558, 559, 659
 MGL_unpackColorFast, 558, 559
 MGL_unpackColorRGB, 560, 561
 MGL_unpackColorRGBFast, 560, 561
 MGL_unregisterAllDrivers, 148, 423, 562
 MGL_useEvents, 563
 MGL_useFont, 116, 161, 162, 233, 333, 564, 568
 MGL_useLocalMalloc, 113, 354, 565
 MGL_vecFontEngine, 567
 MGL_vSync, 566
 MGL_waitVRTFlagType, 79, 80, 652, 662
 MGL_WIN_COLORS, 613, 614
 MGL_writeModeType, 106, 110, 278, 399, 523, 653, 656
 MGL_zbufferAccessType, 569
 MGLDC, 77, 89, 90, 104, 105, 108, 109, 114, 122, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 135, 138, 142, 145, 153, 154, 155, 182, 205, 210, 213, 218, 220, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 239, 248, 251, 252, 253, 254, 263, 273, 275, 276, 277, 281,

282, 284, 288, 290, 291, 292, 295, 296, 298, 300, 304, 310, 311, 312, 314, 328, 330, 341, 343, 347, 349, 351, 352, 353, 355, 358, 359, 361, 363, 399, 400, 401, 402, 404, 406, 407, 408, 411, 412, 426, 431, 448, 449, 450, 452, 466, 474, 476, 481, 482, 494, 496, 497, 504, 508, 517, 519, 520, 522, 524, 525, 526, 528, 529, 530, 531, 532, 533, 535, 537, 541, 542, 544, 545, 566, 569, 585, 602, 609, 668

MGLVisual, 281, 288, 296, 612

MS_available, 570

MS_drawCursor, 571

MS_getPos, 572, 574

MS_hide, 571, 573, 575, 578

MS_moveTo, 572, 574

MS_obscure, 573, 575, 578

MS_setCursor, 331, 576

MS_setCursorColor, 577

MS_show, 571, 573, 575, 578

P

palette_t, 74, 182, 223, 238, 251, 295, 355, 494, 610, 657, 673

pattern_t, 255, 498, 655, 674

pixel_format_t, 124, 129, 211, 212, 240, 241, 263, 383, 384, 558, 559, 609, 657, 659, 675

pixmap_t, 256, 499, 655, 678

point_t, 141, 177, 190, 191, 197, 215, 259, 262, 274, 275, 299, 300, 315, 316, 321, 323, 350, 351, 356, 375, 377, 388, 391, 392, 393, 394, 395, 397, 415, 437, 445, 446, 518, 519, 655, 679

R

rect_t, 104, 108, 140, 141, 151, 152, 153, 156, 165, 166, 169, 171, 173, 178, 186, 187, 195, 219, 220, 230, 272, 273, 309, 315, 380, 395, 396, 413, 434, 435, 441, 442, 443, 446, 459, 460, 461, 462, 464, 475, 476, 503, 504, 516, 517, 530, 538, 541, 544, 547, 549, 550, 553, 655, 680, 681

region_t, 80, 119, 123, 150, 151, 160, 174, 179, 201, 313, 379, 381, 382, 397, 398, 434, 435, 437, 438, 439, 440, 441, 442, 444, 445, 463, 464, 547, 551, 552, 553, 681

S

SPR_draw, 579, 580, 581, 582
SPR_mgrAddOpaqueBitmap, 579, 580, 582
SPR_mgrAddTransparentBitmap, 579, 580,
581
SPR_mgrEmpty, 580, 582, 583, 584
SPR_mgrExit, 584, 586
SPR_mgrInit, 579, 583, 584, 585
SPR_mgrOffscreenCacheFull, 587, 589
SPR_mgrReloadHW, 588
SPR_mgrUsingOffscreenDC, 587, 589
SVGA_setBank, 126, 131, 535, 590, 591
SVGA_setBankC, 590, 591

T

text_settings_t, 268, 512, 655, 682

U

ULZElapsedTime, 592, 593, 597, 598
ULZReadTime, 592, 593, 597, 598
ULZTimerCount, 594, 595, 596, 597, 598
ULZTimerLap, 594, 595, 596, 597
ULZTimerOff, 594, 595, 596, 597
ULZTimerOn, 594, 595, 596, 597
ULZTimerResolution, 592, 593, 594, 597, 598

Z

ZTimerInit, 599

