

Fastgraph[®] 6.0

for Windows[®]

Reference Manual

Copyright © 1995-2000 by Ted Gruber Software, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without express written permission from Ted Gruber Software. The software described in this publication is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

This publication and its associated software are sold without warranties, either expressed or implied, regarding their merchantability or fitness for any particular application or purpose. The information in this publication is subject to change without notice and does not represent a commitment on the part of Ted Gruber Software. In no event shall Ted Gruber Software be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential, or other damages resulting from the use of or the inability to use this product, even if Ted Gruber Software has been notified of the possibility of such damages.

Fastgraph for Windows version 6.0

Fastgraph® is a registered trademark of Ted Gruber Software, Inc.

Windows®, Direct3D®, DirectDraw®, and DirectX® are registered trademarks of Microsoft Corporation.

All other brand and product names mentioned in this publication are trademarks or registered trademarks of their respective holders.

Table of Contents

Introduction	1
fg_3Dline()	2
fg_3Dpolygon()	3
fg_3Dpolygonobject()	4
fg_3Dpov()	5
fg_3Dproject()	6
fg_3Drenderstate()	7
fg_3Dsetobject()	8
fg_3Dsetzclip()	9
fg_3Dshade()	10
fg_3Dshadeobject()	11
fg_3Dtexturemap()	12
fg_3Dtexturemapobject()	13
fg_3Dtransform()	14
fg_3Dtransformobject()	15
fg_3Dviewport()	16
fg_3Dzclip()	17
fg_3Dzcliprgb()	18
fg_3Dzcliptm()	19
fg_arc()	20
fg_arcw()	21
fg_avidone()	22
fg_aviframe()	23
fg_avihead()	24
fg_avimake()	25
fg_aviopen()	27
fg_avipal()	28
fg_avisplay()	29
fg_avisize()	30
fg_aviskip()	31
fg_blend()	32
fg_blend50()	33
fg_blenddcdb()	34
fg_blendvar()	35
fg_blendvb()	36
fg_blendvbw()	37
fg_bmphead()	38
fg_bmppal()	39
fg_bmpsize()	40
fg_box()	41
fg_boxdepth()	42
fg_boxw()	43
fg_boxx()	44
fg_boxxw()	45
fg_circle()	46
fg_circlef()	47

<code>fg_circlefw()</code>	48
<code>fg_circlew()</code>	49
<code>fg_clip2vb()</code>	50
<code>fg_clipdcb()</code>	51
<code>fg_clipmap()</code>	52
<code>fg_clipmask()</code>	53
<code>fg_clpimage()</code>	54
<code>fg_clprect()</code>	55
<code>fg_clprectw()</code>	56
<code>fg_clprectx()</code>	57
<code>fg_colors()</code>	58
<code>fg_contdcb()</code>	59
<code>fg_contrgb()</code>	60
<code>fg_contvb()</code>	61
<code>fg_copypage()</code>	62
<code>fg_cut()</code>	63
<code>fg_cutdcb()</code>	64
<code>fg_dash()</code>	65
<code>fg_dashrel()</code>	66
<code>fg_dashrw()</code>	67
<code>fg_dashw()</code>	68
<code>fg_ddapply()</code>	69
<code>fg_ddflip()</code>	70
<code>fg_ddflipnw()</code>	71
<code>fg_ddframe()</code>	72
<code>fg_ddfreedc()</code>	73
<code>fg_ddgetdc()</code>	74
<code>fg_ddgetobj()</code>	75
<code>fg_ddlock()</code>	76
<code>fg_ddsetobj()</code>	77
<code>fg_ddsetup()</code>	78
<code>fg_ddstatus()</code>	79
<code>fg_ddunlock()</code>	80
<code>fg_ddusage()</code>	81
<code>fg_defcolor()</code>	82
<code>fg_defpal()</code>	83
<code>fg_dispfile()</code>	84
<code>fg_display()</code>	85
<code>fg_displayp()</code>	86
<code>fg_draw()</code>	87
<code>fg_drawdcb()</code>	88
<code>fg_drawmap()</code>	89
<code>fg_drawmask()</code>	90
<code>fg_drawrel()</code>	91
<code>fg_drawrelx()</code>	92
<code>fg_drawrw()</code>	93
<code>fg_drawrxw()</code>	94
<code>fg_drawww()</code>	95
<code>fg_drawwx()</code>	96

fg_drawxw()	97
fg_drawz()	98
fg_direct()	99
fg_drectw()	100
fg_drwimage()	101
fg_ellipse()	102
fg_ellipsef()	103
fg_ellipsew()	104
fg_ellipsfw()	105
fg_erase()	106
fg_fillpage()	107
fg_findrgb()	108
fg_fixdiv()	109
fg_fixed()	110
fg_fixmul()	111
fg_fixtrig()	112
fg_flicdone()	113
fg_flichead()	114
fg_flicopen()	115
fg_flicplay()	116
fg_flicsize()	117
fg_flicskip()	118
fg_flipdcb()	119
fg_flipmask()	120
fg_float()	121
fg_flood()	122
fg_floodw()	123
fg_flpimage()	124
fg_fontdc()	125
fg_fontload()	126
fg_gammadcb()	127
fg_gammargb()	128
fg_gammavb()	129
fg_gdiflip()	130
fg_getblock()	131
fg_getclip()	132
fg_getclock()	133
fg_getcolor()	134
fg_getdacs()	135
fg_getdc()	136
fg_getdcb()	137
fg_getdepth()	138
fg_gethcbpp()	139
fg_gethpage()	140
fg_getimage()	141
fg_getindex()	142
fg_getline()	143
fg_getlines()	144
fg_getmap()	145

fg_getmaxx()	146
fg_getmaxy()	147
fg_getpage()	148
fg_getpixel()	149
fg_getrgb()	150
fg_getview()	151
fg_getworld()	152
fg_getxbox()	153
fg_getxjust()	154
fg_getxpos()	155
fg_getybox()	156
fg_getyjust()	157
fg_getypos()	158
fg_gouraud()	159
fg_gouraudz()	160
fg_graydcb()	161
fg_grayrgb()	162
fg_grayvb()	163
fg_imagebuf()	164
fg_imagesiz()	165
fg_initw()	166
fg_inside()	167
fg_invdcb()	168
fg_invert()	169
fg_jpegbuf()	170
fg_jpeghead()	171
fg_jpegmem()	172
fg_jpegsize()	173
fg_justify()	174
fg_kbtest()	175
fg_loadpcx()	176
fg_locate()	177
fg_logfont()	178
fg_logpal()	179
fg_makebmp()	180
fg_makepcx()	181
fg_makeppr()	182
fg_makespr()	183
fg_mapdacs()	184
fg_maprgb()	185
fg_measure()	186
fg_memavail()	187
fg_modeset()	188
fg_modetest()	189
fg_mousecur()	190
fg_mouseini()	191
fg_mouselim()	192
fg_mousemov()	193
fg_mousepos()	194

fg_mouseptr()	195
fg_mousesiz()	196
fg_mousevis()	197
fg_move()	198
fg_move3d()	199
fg_moverel()	200
fg_moverw()	201
fg_movew()	202
fg_opacity()	203
fg_pack()	204
fg_pagesize()	205
fg_paint()	206
fg_paintw()	207
fg_paste()	208
fg_pastedcb()	209
fg_pcxhead()	210
fg_pcxpal()	211
fg_pcxrange()	212
fg_pcxsize()	213
fg_photodcb()	214
fg_photorgb()	215
fg_photovb()	216
fg_point()	217
fg_pointw()	218
fg_pointx()	219
fg_pointxw()	220
fg_polyedge()	221
fg_polyfill()	222
fg_polyfilz()	223
fg_polygon()	224
fg_polygonw()	225
fg_polyline()	226
fg_polyoff()	227
fg_print()	228
fg_printer()	229
fg_project()	230
fg_putblock()	231
fg_putdcb()	232
fg_putimage()	233
fg_putpixel()	234
fg_realize()	235
fg_rect()	236
fg_rectw()	237
fg_rectx()	238
fg_reduce()	239
fg_restore()	240
fg_restorew()	241
fg_revdcdb()	242
fg_revimage()	243

fg_revmask()	244
fg_rotate()	245
fg_rotate3d()	246
fg_rotddb()	247
fg_rotsize()	248
fg_save()	249
fg_savew()	250
fg_scale()	251
fg_scaledcb()	252
fg_scroll()	253
fg_setalpha()	254
fg_setangle()	255
fg_setclip()	256
fg_setclipw()	257
fg_setcolor()	258
fg_setdacs()	259
fg_setdc()	260
fg_sethpage()	261
fg_setpage()	262
fg_setratio()	263
fg_setrgb()	264
fg_setsize()	265
fg_setsizew()	266
fg_setview()	267
fg_setworld()	268
fg_shear()	269
fg_sheardcb()	270
fg_showavi()	271
fg_showbmp()	272
fg_showflic()	273
fg_showjpeg()	274
fg_showpcx()	275
fg_showppr()	276
fg_showspr()	277
fg_stall()	278
fg_swchar()	279
fg_swlength()	280
fg_swtext()	281
fg_tcdefine()	282
fg_tcmask()	283
fg_tcxfer()	284
fg_texmap()	285
fg_texmapp()	286
fg_texmappz()	287
fg_texmapz()	288
fg_text()	289
fg_texture()	290
fg_tmdefine()	291
fg_tmfree()	292

fg_tminit()	293
fg_tmselect()	294
fg_tmspan()	295
fg_tmtransparency()	296
fg_transdcb()	297
fg_transfer()	298
fg_trig()	299
fg_unmaprgb()	300
fg_unpack()	301
fg_vb2clip()	302
fg_vbaddr()	303
fg_vballoc()	304
fg_vbclose()	305
fg_vbcolors()	306
fg_vbcopy()	307
fg_vbdefine()	308
fg_vbdepth()	309
fg_vbfin()	310
fg_vbfree()	311
fg_vbhandle()	312
fg_vbinit()	313
fg_vbopen()	314
fg_vbpaste()	315
fg_vbprint()	316
fg_vbscale()	317
fg_vbsize()	318
fg_vbtccopy()	319
fg_vbtzcopy()	320
fg_vbundef()	321
fg_version()	322
fg_view3d()	323
fg_waitfor()	324
fg_where()	325
fg_xalpha()	326
fg_xclient()	327
fg_xconvert()	328
fg_xscreen()	329
fg_xvb()	330
fg_xview()	331
fg_xworld()	332
fg_yalpha()	333
fg_yclient()	334
fg_yconvert()	335
fg_yscreen()	336
fg_yvb()	337
fg_yview()	338
fg_yworld()	339
fg_zballoc()	340
fg_zbframe()	341

fg_zbfree()	342
fg_zbopen()	343

Introduction

The *Fastgraph 6.0 Reference Manual* provides an alphabetical summary of Fastgraph functions, with the following information presented for each function:

- function prototypes or declarations for each supported language
- a description of the function itself
- the number of parameters, their purpose, and their data types
- the meaning and data type of the function's return value (if any)
- information about important restrictions pertaining to the function
- references to similar functions, or other functions that affect the function
- example programs in the *Fastgraph 6.0 User's Guide* that use the function

This manual includes information about Fastgraph *legacy functions* in case you encounter them in programs developed with earlier versions of Fastgraph. Legacy functions are still included in Fastgraph, but they have been replaced by other functions and may not be supported in future releases. We therefore recommend that new applications avoid using the legacy functions, and where possible, you eliminate them from existing programs. Legacy functions are identified as such in the function descriptions; they are also listed in Appendix D of the *Fastgraph 6.0 User's Guide*. For legacy functions, the "Replaced by" section lists the individual function or group of functions with the same or enhanced functionality.

fg_3Dline()

Prototype

```
void fg_3Dline (double x1, double y1, double z1, double x2, double y2,
double z2);

Sub fg_3Dline (ByVal x1 As Double, ByVal y1 As Double, ByVal z1 As
Double, ByVal x2 As Double, ByVal y2 As Double, ByVal z2 As Double)

procedure fg_3Dline (x1, y1, z1, x2, y2, z2 : double);
```

Description

The **fg_3Dline()** function draws a line defined in 3D world space, with optional z-buffering and 3D clipping.

Parameters

x1, *y1*, and *z1* are the 3D world space coordinates for one of the line's end points.

x2, *y2*, and *z2* are the 3D world space coordinates for the line's other end point.

Return value

none

Restrictions

none

See also

fg_3Drenderstate()

Examples

Geometry

fg_3Dpolygon()

Prototype

```
void fg_3Dpolygon (double *xyz_array, int n);  
Sub fg_3Dpolygon (xyz_array() As Double, ByVal n As Long)  
procedure fg_3Dpolygon (var xyz_array : double; n : integer);
```

Description

The **fg_3Dpolygon()** function draws a filled or unfilled convex polygon defined in 3D world space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon.

Parameters

xyz_array is the array containing the 3D world space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

n is the number of vertices in *xyz_array*.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dpolygon()**, only a part of it will be filled.

See also

fg_3Dpolygonobject(), fg_3Drenderstate(), fg_3Dshade(), fg_3Dtexturemap(), fg_polyoff()

Examples

Geometry

fg_3Dpolygonobject()

Prototype

```
void fg_3Dpolygonobject (double *xyz_array, int n);  
Sub fg_3Dpolygonobject (xyz_array() As Double, ByVal n As Long)  
procedure fg_3Dpolygonobject (var xyz_array : double; n : integer);
```

Description

The **fg_3Dpolygonobject()** function draws a filled or unfilled convex polygon defined in 3D object space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon. The polygon is drawn in 3D world space at the position and orientation specified in the most recent call to **fg_3Dsetobject()**.

Parameters

xyz_array is the array containing the 3D object space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

n is the number of vertices in *xyz_array*.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dpolygonobject()**, only a part of it will be filled.

See also

fg_3Dpolygon(), fg_3Drenderstate(), fg_3Dsetobject(), fg_3Dshadeobject(),
fg_3Dtexturemapobject(), fg_polyoff()

Examples

Cube, Geometry

fg_3Dpov()

Prototype

```
void fg_3Dpov (double x, double y, double z, int x_angle, int y_angle,  
int z_angle);
```

```
Sub fg_3Dpov (ByVal x As Double, ByVal y As Double, ByVal z As Double,  
ByVal x_angle As Long, ByVal y_angle As Long, ByVal z_angle As Long)
```

```
procedure fg_3Dpov (x, y, z : double; x_angle, y_angle, z_angle :  
integer);
```

Description

The **fg_3Dpov()** function defines the viewer's position and orientation in 3D world space.

Parameters

x is the x coordinate of the viewer's position in 3D world space.

y is the y coordinate of the viewer's position in 3D world space.

z is the z coordinate of the viewer's position in 3D world space.

x_angle is the viewer's counterclockwise orientation about the world space x axis.

y_angle is the viewer's counterclockwise orientation about the world space y axis.

z_angle is the viewer's counterclockwise orientation about the world space z axis.

Return value

none

Restrictions

none

Examples

Geometry, Tunnel

fg_3Dproject()

Prototype

```
void fg_3Dproject (double *source, int *dest, int n);  
Sub fg_3Dproject (source() As Double, dest() As Long, ByVal n As Long)  
procedure fg_3Dproject (var source : double; var dest : integer; n :  
integer);
```

Description

The **fg_3Dproject()** function projects a series of transformed 3D (x,y,z) points to 2D (x,y) screen space points. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

source is the name of the array containing the transformed 3D (x,y,z) coordinates to project to screen space. The first three elements of the *source* array contain the coordinates for the first point, the next three elements are for the next point, and so on.

dest is the name of the array that receives the projected 2D (x,y) screen space coordinates. The first two elements of the *dest* array will contain the coordinates for the first point, the next two elements will contain the next point, and so on. The size of *dest* must be large enough to hold $2/3*n$ integer values.

n is the number of 3D points to project.

Return value

none

Restrictions

Before using this function, you must set up a 3D viewport with **fg_3Dviewport()**.

See also

fg_3Dtransform(), fg_3Dtransformobject()

fg_3Drenderstate()

Prototype

```
void fg_3Drenderstate (int flags);  
Sub fg_3Drenderstate (ByVal flags As Long)  
procedure fg_3Drenderstate (flags : integer);
```

Description

The **fg_3Drenderstate()** function defines the render state for the 3D line drawing and 3D polygon drawing functions.

Parameters

flags is one or more of the following flags:

Flag	Applies to
FG_LINEAR_TM	fg_3Dtexturemap() and fg_3Dtexturemapobject()
FG_PERSPECTIVE_TM	fg_3Dtexturemap() and fg_3Dtexturemapobject()
FG_WIREFRAME	fg_3Dpolygon() and fg_3Dpolygonobject()
FG_ZBUFFER	all
FG_ZCLIP	all

Return value

none

Restrictions

none

See also

fg_3Dline(), **fg_3Dpolygon()**, **fg_3Dpolygonobject()**, **fg_3Dshade()**, **fg_3Dshadeobject()**, **fg_3Dtexturemap()**, **fg_3Dtexturemapobject()**

Examples

Cube, Geometry, TMcube, TMcubeX, Tunnel

fg_3Dsetobject()

Prototype

```
void fg_3Dsetobject (double x, double y, double z, int x_angle, int
y_angle, int z_angle);

Sub fg_3Dsetobject (ByVal x As Double, ByVal y As Double, ByVal z As
Double, ByVal x_angle As Long, ByVal y_angle As Long, ByVal z_angle As
Long)

procedure fg_3Dsetobject (x, y, z : double; x_angle, y_angle, z_angle :
integer);
```

Description

The **fg_3Dsetobject()** function defines an object's position in 3D world space and its rotation about its own object space axes (center of gravity).

Parameters

x is the x position of the object's center of gravity in 3D world space.

y is the y position of the object's center of gravity in 3D world space.

z is the z position of the object's center of gravity in 3D world space.

x_angle is the counterclockwise rotation about the object's x axis, expressed in tenths of degrees.

y_angle is the counterclockwise rotation about the object's y axis, expressed in tenths of degrees.

z_angle is the counterclockwise rotation about the object's z axis, expressed in tenths of degrees.

Return value

none

Restrictions

none

See also

fg_3Dpolygonobject(), fg_3Dshadeobject(), fg_3Dtexturemapobject()

Examples

Cube, Geometry, TMcube, TMcubeX

fg_3Dsetzclip()

Prototype

```
void fg_3Dsetzclip (double z_near, double z_far);  
Sub fg_3Dsetzclip (ByVal z_near As Double, ByVal z_far As Double)  
procedure fg_3Dsetzclip (z_near, z_far : double);
```

Description

The **fg_3Dsetzclip()** function defines the near and far z clipping planes for 3D clipping. If you do not call **fg_3Dsetzclip()**, Fastgraph will use its default near and far clipping planes of 1 and 1000 respectively.

Parameters

z_near is the z coordinate of the near clipping plane. It must be greater than zero, or if z-buffering is used, greater than or equal to 1.

z_far is the z coordinate of the far clipping plane. It must be greater or equal to *z_near*.

Return value

none

Restrictions

3D clipping is not supported for right-handed 3D coordinate systems.

Examples

TMcube, TMcubeX

fg_3Dshade()

Prototype

```
void fg_3Dshade (double *xyz_array, char *rgb_array, int n);  
Sub fg_3Dshade (xyz_array() As Double, rgb_array() As Byte, ByVal n As Long)  
procedure fg_3Dshade (var xyz_array : double; var rgb_array : byte; n : integer);
```

Description

The **fg_3Dshade()** function draws a Gouraud-shaded convex polygon defined in 3D world space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon.

Parameters

xyz_array is an array containing the 3D world space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

rgb_array is an array containing the RGB color components for each polygon vertex. The first three *rgb_array* elements represent the RGB color values at the polygon's first vertex, the next three *rgb_array* elements are for the second vertex, and so forth. Each RGB color component is a value between 0 and 255.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dshade()**, only a part of it will be filled.

See also

fg_3Dpolygon(), fg_3Drenderstate(), fg_3Dshadeobject(), fg_3Dtexturemap(), fg_polyoff()

Examples

Tunnel

fg_3Dshadeobject()

Prototype

```
void fg_3Dshadeobject (double *xyz_array, char *rgb_array, int n);
Sub fg_3Dshadeobject (xyz_array() As Double, rgb_array() As Byte, ByVal
n As Long)
procedure fg_3Dshadeobject (var xyz_array : double; var rgb_array :
byte; n : integer);
```

Description

The **fg_3Dshadeobject()** function draws a Gouraud-shaded convex polygon defined in 3D object space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon. The polygon is drawn in 3D world space at the position and orientation specified in the most recent call to **fg_3Dsetobject()**.

Parameters

xyz_array is an array containing the 3D object space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

rgb_array is an array containing the RGB color components for each polygon vertex. The first three *rgb_array* elements represent the RGB color values at the polygon's first vertex, the next three *rgb_array* elements are for the second vertex, and so forth. Each RGB color component is a value between 0 and 255.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dshadeobject()**, only a part of it will be filled.

See also

fg_3Dpolygonobject(), fg_3Drenderstate(), fg_3Dsetobject(), fg_3Dshade(),
fg_3Dtexturemapobject(), fg_polyoff()

fg_3Dtexturemap()

Prototype

```
void fg_3Dtexturemap (double *xyz_array, int *uv_array, int n);  
Sub fg_3Dtexturemap (xyz_array() As Double, uv_array() As Long, ByVal n  
As Long)  
procedure fg_3Dtexturemap (var xyz_array : double; var uv_array :  
integer; n : integer);
```

Description

The **fg_3Dtexturemap()** function draws a linear or perspective texture-mapped polygon defined in 3D world space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon.

Parameters

xyz_array is an array containing the 3D world space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

uv_array is an array containing the (u,v) texture map coordinates for each polygon vertex. The first two *uv_array* elements represent the (u,v) values at the polygon's first vertex, the next two *uv_array* elements are for the second vertex, and so forth.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dtexturemap()**, only a part of it will be filled.

See also

fg_3Dpolygon(), fg_3Drenderstate(), fg_3Dshade(), fg_3Dtexturemapobject(), fg_polyoff(),
fg_tmdefine(), fg_tmselect(), fg_tmspan(), fg_tmtransparency()

fg_3Dtexturemapobject()

Prototype

```
void fg_3Dtexturemapobject (double *xyz_array, int *uv_array, int n);
Sub fg_3Dtexturemapobject (xyz_array() As Double, uv_array() As Long,
    ByVal n As Long)
procedure fg_3Dtexturemapobject (var xyz_array : double; var uv_array :
    integer; n : integer);
```

Description

The **fg_3Dtexturemapobject()** function draws a linear or perspective texture-mapped polygon defined in 3D object space, with optional z-buffering and 3D clipping. Backface removal is performed unless drawing a z-buffered polygon. The polygon is drawn in 3D world space at the position and orientation specified in the most recent call to **fg_3Dsetobject()**.

Parameters

xyz_array is an array containing the 3D object space (x,y,z) coordinates for each polygon vertex. The first three *xyz_array* elements represent the (x,y,z) values at the polygon's first vertex, the next three *xyz_array* elements are for the second vertex, and so forth.

uv_array is an array containing the (u,v) texture map coordinates for each polygon vertex. The first two *uv_array* elements represent the (u,v) values at the polygon's first vertex, the next two *uv_array* elements are for the second vertex, and so forth.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_3Dtexturemapobject()**, only a part of it will be filled.

See also

fg_3Dpolygonobject(), **fg_3Drenderstate()**, **fg_3Dsetobject()**, **fg_3Dshadeobject()**,
fg_3Dtexturemap(), **fg_polyoff()**, **fg_tmdefine()**, **fg_tmselect()**, **fg_tmspan()**, **fg_tmtransparency()**

Examples

TMcube, TMcubeX

fg_3Dtransform()

Prototype

```
void fg_3Dtransform (double *source, double *dest, int n);  
Sub fg_3Dtransform (source() As Double, dest() As Double, ByVal n As  
Long)  
procedure fg_3Dtransform (var source, dest : double; n : integer);
```

Description

The **fg_3Dtransform()** function transforms a series of 3D (x,y,z) coordinates from world space to view space. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

source is the name of the array containing the 3D world space (x,y,z) coordinates to transform. The first three elements of the *source* array contain the coordinates for the first point, the next three elements are for the next point, and so on.

dest is the name of the array that receives the transformed 3D view space (x,y,z) coordinates. The first three elements of the *dest* array will contain the coordinates for the first point, the next three elements will contain the next point, and so on. The size of *dest* must be at least as large as the *source* array.

n is the number of 3D points to transform.

Return value

none

Restrictions

none

See also

fg_3Dproject(), fg_3Dtransformobject()

fg_3Dtransformobject()

Prototype

```
void fg_3Dtransformobject (double *source, double *dest, int n);  
Sub fg_3Dtransformobject (source() As Double, dest() As Double, ByVal n  
As Long)  
procedure fg_3Dtransformobject (var source, dest : double; n :  
integer);
```

Description

The **fg_3Dtransformobject()** function transforms a series of 3D (x,y,z) coordinates from object space to view space. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

source is the name of the array containing the 3D object space (x,y,z) coordinates to transform. The first three elements of the *source* array contain the coordinates for the first point, the next three elements are for the next point, and so on.

dest is the name of the array that receives the transformed 3D view space (x,y,z) coordinates. The first three elements of the *dest* array will contain the coordinates for the first point, the next three elements will contain the next point, and so on. The size of *dest* must be at least as large as the *source* array.

n is the number of 3D points to transform.

Return value

none

Restrictions

none

See also

fg_3Dproject(), fg_3Dsetobject(), fg_3Dtransform()

fg_3Dviewport()

Prototype

```
void fg_3Dviewport (int minx, int maxx, int miny, int maxy, double  
  ratio);  
  
Sub fg_3Dviewport (ByVal minx As Long, ByVal maxx As Long, ByVal miny  
  As Long, ByVal maxy As Long, ByVal ratio As Double)  
  
procedure fg_3Dviewport (minx, maxx, miny, maxy : integer; ratio :  
  double);
```

Description

The **fg_3Dviewport()** function defines the 3D viewport in screen space and the corresponding projection ratio. You must set up a 3D viewport before using any 3D drawing functions or **fg_3Dproject()**.

Parameters

minx is the x coordinate of the viewport's left edge.

maxx is the x coordinate of the viewport's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the viewport's top edge.

maxy is the y coordinate of the viewport's bottom edge. It must be greater than or equal to the value of *miny*.

ratio is the projection ratio. It must be greater than zero for a left-handed 3D coordinate system, and less than zero for a right-handed system.

Return value

none

Restrictions

3D clipping cannot be used with a right-handed coordinate system.

Examples

Cube, Geometry, TMcube, TMcubeX, Tunnel

fg_3Dzclip()

Prototype

```
int fg_3Dzclip (double *source, double *dest, int n);  
  
Function fg_3Dzclip (source() As Double, dest() As Double, ByVal n As  
Long) As Long  
  
function fg_3Dzclip (var source, dest : double; n : integer) : integer;
```

Description

The **fg_3Dzclip()** function clips a series of 3D (x,y,z) polygon vertices against the z plane clipping values defined by **fg_3Dsetzclip()**. Near clipping uses true clipping, but far clipping occurs only if the entire polygon lies beyond the far clipping plane. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

source is the name of the array containing the 3D (x,y,z) vertices to clip. The first three elements of the *source* array contain the coordinates for the first vertex, the next three elements are for the next vertex, and so on.

dest is the name of the array that receives the clipped 3D (x,y,z) vertices. The first three elements of the *dest* array will contain the coordinates for the first vertex, the next three elements will contain the next vertex, and so on. The size of *dest* must be at least as large as the *source* array.

n is the number of vertices in the *source* array.

Return value

The resulting number of vertices in the *dest* array.

Restrictions

3D clipping is not supported for right-handed 3D coordinate systems.

See also

fg_3Dtransform(), fg_3Dtransformobject(), fg_3Dzcliprgb(), fg_3Dzcliptm()

fg_3Dzcliprgb()

Prototype

```
int fg_3Dzcliprgb (double *xyz_source, double *xyz_dest, char  
*rgb_source, char *rgb_dest, int n);
```

```
Function fg_3Dzcliprgb (xyz_source() As Double, xyz_dest() As Double,  
rgb_source() As Byte, rgb_dest() As Byte, ByVal n As Long) As Long
```

```
function fg_3Dzcliprgb (var xyz_source, xyz_dest : double; var  
rgb_source, rgb_dest : byte; n : integer) : integer;
```

Description

The **fg_3Dzcliprgb()** function clips a series of 3D (x,y,z) polygon vertices and corresponding RGB shading values against the z plane clipping values defined by **fg_3Dsetzclip()**. Near clipping is true clipping, but far clipping occurs only if the entire polygon lies beyond the far clipping plane. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

xyz_source is the name of the array containing the 3D (x,y,z) vertices to clip. The first three elements of the array contain the coordinates for the first vertex, the next three elements are for the next vertex, and so on.

xyz_dest is the name of the array that receives the clipped 3D (x,y,z) vertices. The first three elements of the array will contain the coordinates for the first vertex, the next three elements will contain the next vertex, and so on. The *xyz_dest* array must be at least as large as the *xyz_source* array.

rgb_source is the name of the array containing the RGB shading values for each (x,y,z) coordinate triple in *xyz_source*. The first three *rgb_source* elements represent the shading values at the first vertex in *xyz_source*, the next three *rgb_source* elements are for the second vertex, and so forth.

rgb_dest is the name of the array that receives the clipped RGB shading values. The first three elements of the *rgb_dest* array will contain the shading values for the first vertex, the next three elements will be for the second vertex, and so on. The *rgb_dest* array must be at least as large as the *rgb_source* array.

n is the number of vertices in the *xyz_source* and *rgb_source* arrays.

Return value

The resulting number of vertices in the *xyz_dest* and *rgb_dest* arrays.

Restrictions

RGB clipping is not supported for right-handed 3D coordinate systems.

See also

fg_3Dtransform(), fg_3Dtransformobject(), fg_3Dzclip(), fg_3Dzcliptm()

fg_3Dzcliptm()

Prototype

```
int fg_3Dzcliptm (double *xyz_source, double *xyz_dest, int *uv_source,
int *uv_dest, int n);
```

```
Function fg_3Dzcliptm (xyz_source() As Double, xyz_dest() As Double,
uv_source() As Long, uv_dest() As Long, ByVal n As Long) As Long
```

```
function fg_3Dzcliptm (var xyz_source, xyz_dest : double; var
uv_source, uv_dest : integer; n : integer) : integer;
```

Description

The **fg_3Dzcliptm()** function clips a series of 3D (x,y,z) polygon vertices and corresponding 2D (u,v) texture map coordinates against the z plane clipping values defined by **fg_3Dsetzclip()**. Near clipping uses true clipping, but far clipping occurs only if the entire polygon lies beyond the far clipping plane. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

xyz_source is the name of the array containing the 3D (x,y,z) vertices to clip. The first three elements of the array contain the coordinates for the first vertex, the next three elements are for the next vertex, and so on.

xyz_dest is the name of the array that receives the clipped 3D (x,y,z) vertices. The first three elements of the array will contain the coordinates for the first vertex, the next three elements will contain the next vertex, and so on. The *xyz_dest* array must be at least as large as the *xyz_source* array.

uv_source is the name of the array containing the (u,v) texture map coordinates for each (x,y,z) coordinate triple in *xyz_source*. The first two *uv_source* elements represent the (x,y,z) values at the first vertex in *xyz_source*, the next two *uv_source* elements are for the second vertex, and so forth.

uv_dest is the name of the array that receives the clipped 2D (u,v) coordinates. The first two elements of the *uv_dest* array will contain the coordinates for the first vertex, the next two elements will be for the next vertex, and so on. The *uv_dest* array must be at least as large as the *uv_source* array.

n is the number of vertices in the *xyz_source* and *uv_source* arrays.

Return value

The resulting number of vertices in the *xyz_dest* and *uv_dest* arrays.

Restrictions

Texture clipping is not supported for right-handed 3D coordinate systems.

See also

fg_3Dtransform(), fg_3Dtransformobject(), fg_3Dzclip(), fg_3Dzcliprgb()

fg_arc()

Prototype

```
void fg_arc (int radius, int start_angle, int end_angle);  
Sub fg_arc (ByVal radius As Long, ByVal start_angle As Long, ByVal  
end_angle As Long)  
procedure fg_arc (radius, start_angle, end_angle : integer);
```

Description

The **fg_arc()** function draws a circular arc in screen space, centered at the current graphics position, with clipping. The arc is drawn counterclockwise from the starting angle to the ending angle.

Parameters

radius is the arc's radius in horizontal screen space units. Its value must be greater than zero. The radius defines the arc's horizontal distance (at zero degrees) from the current graphics position.

start_angle is the starting point of the arc, expressed in tenths of degrees counterclockwise from the horizontal.

end_angle is the ending point of the arc, expressed in tenths of degrees counterclockwise from the horizontal.

Return value

none

Restrictions

none

See also

fg_arcw(), fg_circle(), fg_ellipse()

fg_arcw()

Prototype

```
void fg_arcw (double radius, int start_angle, int end_angle);  
Sub fg_arcw (ByVal radius As Double, ByVal start_angle As Long, ByVal  
end_angle As Long)  
procedure fg_arcw (radius : real; start_angle, end_angle : integer);
```

Description

The **fg_arcw()** function draws a circular arc in 2D world space, centered at the current graphics position, with clipping. The arc is drawn counterclockwise from the starting angle to the ending angle.

Parameters

radius is the arc's radius in horizontal world space units. Its value must be greater than zero. The radius defines the arc's horizontal distance (at zero degrees) from the current graphics position.

start_angle is the starting point of the arc, expressed in tenths of degrees counterclockwise from the horizontal.

end_angle is the ending point of the arc, expressed in tenths of degrees counterclockwise from the horizontal.

Return value

none

Restrictions

none

See also

fg_arc(), fg_circlew(), fg_ellipsew()

fg_avidone()

Prototype

```
void fg_avidone (void *context);  
Sub fg_avidone (context() As Any)  
procedure fg_avidone (var context);
```

Description

The **fg_avidone()** function closes the AVI file associated with the specified context descriptor.

Parameters

context is the name of a 48-byte (for playing an AVI) or a 24-byte (for creating an AVI) buffer containing the AVI file context descriptor.

Return value

none

Restrictions

none

See also

fg_aviopen()

Examples

AVImake, Image

fg_aviframe()

Prototype

```
int fg_aviframe (void *context, void *bitmap);  
Function fg_aviframe (context() As Any, bitmap() As Any) As Long  
function fg_aviframe (var context, bitmap) : integer;
```

Description

The **fg_aviframe()** function writes one frame to an AVI file previously opened with **fg_avimake()**.

Parameters

context is the name of a 24-byte buffer containing the AVI file context descriptor.

bitmap is a bitmap containing the AVI frame data. The bitmap width and height must be the same as specified in the **fg_avimake()** call for this context descriptor. If creating a 256-color AVI, *bitmap* must be a 256-color bitmap. If creating a high color or true color AVI, *bitmap* must be a 24-bit direct color bitmap.

Return value

0 = Frame written successfully
-1 = Error writing the frame

Restrictions

none

See also

fg_avimake(), fg_transdcb()

Examples

AVImake

fg_avihead()

Prototype

```
int fg_avihead (char *filename, void *header);
```

```
Function fg_avihead (ByVal filename As String, header() As Any) As Long
```

```
function fg_avihead (filename : string; var header) : integer;
```

Description

The **fg_avihead()** function reads an AVI file header into a 56-byte buffer. Refer to Appendix E of the *Fastgraph 6.0 User's Guide* for details about the AVI header.

Parameters

filename is the name of the AVI file. It may include a path specification and must be terminated by a zero byte.

header is the name of the buffer to receive the AVI file header. Its size must be at least 56 bytes.

Return value

0 = Success

-1 = The specified file does not exist

-2 = The specified file is not an AVI file

Restrictions

none

See also

fg_avipal(), fg_avisplay(), fg_avisize(), fg_showavi()

Examples

Image

fg_avimake()

Prototype

```
int fg_avimake (char *filename, void *context, int compressor, int
width, int height, int depth, int quality, int rate);
```

Function fg_avimake (ByVal filename As String, context() As Any, ByVal compressor As Long, ByVal width As Long, ByVal height As Long, ByVal depth As Long, ByVal quality As Long, ByVal rate As Long) As Long

```
function fg_avimake (filename : string; var context; compressor, width,
height, depth, quality, rate : integer) : integer;
```

Description

The **fg_avimake()** function creates an empty AVI file that will be built with **fg_aviframe()**.

Parameters

filename is the name of the AVI file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

context is the name of a 24-byte buffer that will receive the AVI file context descriptor. The descriptor values will only be meaningful if the return value is zero.

compressor is the four-character code (FourCC) for the compressor that will be used to compress the video data. It can instead be zero for no compression, or -1 to display a dialog box to select compression options at run time. Here are the FourCC values for some of the more popular codecs supplied with Video for Windows:

Codec name	FourCC	Hex equivalent
Intel Indeo 3.2	IV32	49563332
Microsoft Video 1	MSVC	4D535643
Microsoft Run Length Encoding	MRLE	4D524C45
Radius Cinepak	CVID	43564944

width is the AVI image width in pixels.

height is the AVI image height in pixels.

depth is the suggested AVI color depth in bits per pixel.

quality is the compression quality, between 0 and 10,000. A value of 10,000 indicates lossless compression; lower values result in increasing degrees of lossy compression. The quality value is not used if compressor is 0 or -1.

rate is the video rate in frames per second.

Return value

- 0 = AVI file created successfully
- 1 = Error creating the AVI file
- 2 = Error creating the AVI video stream
- 3 = User pressed Cancel on the compress options dialog box

Restrictions

none

fg_avimake() (continued)

See also

`fg_avidone()`, `fg_aviframe()`, `fg_aviopen()`

Examples

`AVImake`

fg_aviopen()

Prototype

```
int fg_aviopen (char *filename, void *context);  
Function fg_aviopen (ByVal filename As String, context() As Any) As Long  
function fg_aviopen (filename : string; var context) : integer;
```

Description

The **fg_aviopen()** function opens an AVI file for reading.

Parameters

filename is the name of the AVI file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

context is the name of a 48-byte buffer that will receive the AVI file context descriptor. The descriptor values will only be meaningful if the return value is zero.

Return value

- 0 = AVI file opened successfully
- 1 = The specified file does not exist
- 2 = The specified file is not an AVI file
- 3 = Error initializing the AVI video stream
- 4 = Error allocating memory
- 5 = The codec needed for the specified AVI file is not available

Restrictions

none

See also

fg_avidone(), fg_avimake()

Examples

Image

fg_avipal()

Prototype

```
int fg_avipal (char *filename, void *palette);
```

```
Function fg_avipal (ByVal filename As String, palette() As Any) As Long
```

```
function fg_avipal (filename : string; var palette) : integer;
```

Description

The **fg_avipal()** function retrieves the palette of an image stored in an AVI file. The palette values are returned as RGB color components, each between 0 and 255.

Parameters

filename is the name of the AVI file. The file name must be terminated by a zero byte.

palette is the name of the array that will receive the AVI palette values. The palette values are returned as RGB color components, each between 0 and 255. The first three bytes of *palette* will contain the RGB values for color 0, the next three for color 1, and so forth. The size of the *palette* array must be at least three times the number of colors in the AVI image. You can also specify NULL for the *palette* parameter, in which case **fg_avipal()** will return the AVI color depth but no palette values.

Return value

>0 = The number of colors in the AVI palette

0 = The AVI file does not have a palette (probably a high color or true color AVI file)

-1 = The specified file does not exist

-2 = The specified file is not an AVI file

Restrictions

none

See also

fg_avihead(), fg_setdacs(), fg_showavi()

Examples

Image

fg_avisplay()

Prototype

```
int fg_avisplay (void *context, int frames, int flags);

Function fg_avisplay (context() As Any, ByVal frames As Long, ByVal
flags As Long) As Long

function fg_avisplay (var context; frames, flags : integer) : integer;
```

Description

The **fg_avisplay()** function plays the next one or more individual frames in an AVI file that was previously opened with **fg_avioopen()**.

Parameters

context is the name of a 48-byte buffer containing the AVI file context descriptor.

frames is the number of frames to play from the AVI file, starting from the current file position.

flags is a series of flags that controls how the AVI is played:

Flag	Meaning
FG_AT_XY	If specified, play the AVI file relative to the current graphics position. If not, play it relative to (0,0).
FG_IGNOREAVIPALETTE	If specified, play the AVI file using the current palette. If not, use the palette values stored in the AVI file. Not meaningful for high color or true color AVI files.
FG_NODELAY	If specified, play the AVI file with no delay between frames. If not, delay between frames as specified in the AVI header.

Return value

The number of frames played. This value may be less than the value of *frames* if the end-of-file is reached before playing the requested number of frames.

Restrictions

none

See also

fg_avioopen(), fg_aviskip(), fg_showavi()

Examples

Image

fg_avisize()

Prototype

```
void fg_avisize (void *header, int *width, int *height);  
Sub fg_avisize (header() As Any, width As Long, height As Long)  
procedure fg_avisize (var header; var width, height : integer);
```

Description

The **fg_avisize()** function returns the dimensions for the AVI image associated with the specified AVI file header.

Parameters

header is the name of the buffer containing the 56-byte AVI file header.

width receives the AVI image width in pixels.

height receives the AVI image height in pixels.

Return value

none

Restrictions

none

See also

fg_avihead(), fg_showavi()

Examples

Image

fg_aviskip()

Prototype

```
int fg_aviskip (void *context, int frames);  
Function fg_aviskip (context() As Any, ByVal frames As Long) As Long  
function fg_aviskip (var context; frames : integer) : integer;
```

Description

The **fg_aviskip()** function advances one or more frames in an AVI file that was previously opened with **fg_aviopen()**.

Parameters

context is the name of a 48-byte buffer containing the AVI file context descriptor.

frames is the number of frames to skip in the AVI file, starting from the current file position. If *frames* is negative, the AVI file position will be set to the first frame.

Return value

The number of frames skipped. This value may be less than the value of *frames* if the end-of-file is reached before skipping the requested number of frames. If *frames* is negative, the return value will be zero.

Restrictions

none

See also

fg_aviopen(), fg_aviplay()

Examples

Image

fg_blend()

Prototype

```
int fg_blend (int foreground, int background);  
  
Function fg_blend (ByVal foreground As Long, ByVal background As Long)  
As Long  
  
function fg_blend (foreground, background : integer) : integer;
```

Description

The **fg_blend()** function computes an alpha-blended color value from the specified foreground and background colors using the current **fg_opacity()** setting.

Parameters

foreground is the foreground color value. For high color virtual buffers, *foreground* is encoded using the 5/6/5 or 5/5/5 RGB color scheme. For true color virtual buffers, *foreground* is encoded as four 8-bit color components (xRGB).

background is the background color value, encoded the same way as the foreground color.

Return value

The alpha-blended color value, encoded the same way as the foreground and background colors.

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend50(), fg_blenddcb(), fg_blendvar(), fg_blendvb(), fg_blendvbw(), fg_opacity()

fg_blend50()

Prototype

```
int fg_blend50 (int foreground, int background);  
Function fg_blend50 (ByVal foreground As Long, ByVal background As  
Long) As Long  
function fg_blend50 (foreground, background : integer) : integer;
```

Description

The **fg_blend50()** function computes a 50% alpha-blended color value from the specified foreground and background colors.

Parameters

foreground is the foreground color value. For high color virtual buffers, *foreground* is encoded using the 5/6/5 or 5/5/5 RGB color scheme. For true color virtual buffers, *foreground* is encoded as four 8-bit color components (xRGB).

background is the background color value, encoded the same way as the foreground color.

Return value

The 50% alpha-blended color value, encoded the same way as the foreground and background colors.

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend(), fg_blenddcb(), fg_blendvar(), fg_blendvb(), fg_blendvbw()

fg_blenddcb()

Prototype

```
void fg_blenddcb (void *foreground, void *background, void *blended,  
int size);  
  
Sub fg_blenddcb (foreground() As Any, background() As Any, blended() As  
Any, ByVal size As Long)  
  
procedure fg_blenddcb (var foreground, background, blended; size :  
integer);
```

Description

The **fg_blenddcb()** function computes an alpha-blended bitmap from the specified foreground and background bitmaps using the current **fg_opacity()** setting.

Parameters

foreground is the name of the array containing the foreground direct color bitmap.

background is the name of the array containing the background direct color bitmap.

blended is the name of the array that will receive the resulting alpha blended direct color bitmap. It must be at least as large as the *foreground* and *background* arrays.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend(), fg_blend50(), fg_blendvar(), fg_blendvb(), fg_blendvbv(), fg_opacity()

Examples

Blend

fg_blendvar()

Prototype

```
void fg_blendvar (void *foreground, void *background, void *opacity,  
void *blended, int size);  
  
Sub fg_blendvar (foreground() As Any, background() As Any, opacity() As  
Any, blended() As Any, ByVal size As Long)  
  
procedure fg_blendvar (var foreground, background, opacity, blended;  
size : integer);
```

Description

The **fg_blendvar()** function computes an alpha-blended bitmap from the specified foreground and background bitmaps using the specified opacity bitmap.

Parameters

foreground is the name of the array containing the foreground direct color bitmap.

background is the name of the array containing the background direct color bitmap.

opacity is the name of the array containing the opacity values, represented as a 256-color bitmap.

blended is the name of the array that will receive the resulting alpha blended direct color bitmap. It must be at least as large as the *foreground* and *background* arrays.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend(), fg_blend50(), fg_blenddcb(), fg_blendvb(), fg_blendvbv()

Examples

Blend

fg_blendvb()

Prototype

```
void fg_blendvb (void *foreground, void *background, int width, int height);  
  
Sub fg_blendvb (foreground() As Any, background() As Any, ByVal width As Long, ByVal height As Long)  
  
procedure fg_blendvb (var foreground, background; width, height : integer);
```

Description

The **fg_blendvb()** function computes an alpha-blended image from the specified foreground and background bitmaps using the current **fg_opacity()** setting. The resulting alpha-blended image is written to the active virtual buffer with its lower left corner at the current graphics position.

Parameters

foreground is the name of the array containing the foreground direct color bitmap.

background is the name of the array containing the background direct color bitmap. If *background* is NULL, **fg_blendvb()** gets the background pixels from the same area of the active virtual buffer where the blended pixels will be written.

width is the width of each direct color bitmap in pixels.

height is the height of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend(), fg_blend50(), fg_blenddcbb(), fg_blendvar(), fg_blendvbv(), fg_opacity()

fg_blendvbv()

Prototype

```
void fg_blendvbv (void *foreground, void *background, void *opacity,
int width, int height);

Sub fg_blendvbv (foreground() As Any, background() As Any, opacity() As
Any, ByVal width As Long, ByVal height As Long)

procedure fg_blendvbv (var foreground, background, opacity; width,
height : integer);
```

Description

The **fg_blendvbv()** function computes an alpha-blended image from the specified foreground and background bitmaps using the specified opacity bitmap. The resulting alpha-blended image is written to the active virtual buffer with its lower left corner at the current graphics position.

Parameters

foreground is the name of the array containing the foreground direct color bitmap.

background is the name of the array containing the background direct color bitmap. If *background* is NULL, **fg_blendvbv()** gets the background pixels from the same area of the active virtual buffer where the blended pixels will be written.

opacity is the name of the array containing the opacity values, represented as a 256-color bitmap.

width is the width of each direct color bitmap in pixels.

height is the height of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_blend(), fg_blend50(), fg_blenddcb(), fg_blendvar(), fg_blendvb()

fg_bmphead()

Prototype

```
int fg_bmphead (char *filename, void *header);
```

```
Function fg_bmphead (ByVal filename As String, header() As Any) As Long
```

```
function fg_bmphead (filename : string; var header) : integer;
```

Description

The **fg_bmphead()** function reads a BMP file header into a 54-byte buffer. Refer to Appendix E of the *Fastgraph 6.0 User's Guide* for details about the BMP header.

Parameters

filename is the name of the BMP file. It may include a path specification and must be terminated by a zero byte.

header is the name of the buffer to receive the BMP file header. Its size must be at least 54 bytes.

Return value

0 = Success

-1 = The specified file does not exist

-2 = The specified file is not a BMP file

Restrictions

none

See also

fg_bmppal(), fg_bmpsize(), fg_showbmp()

Examples

Image, ImgProc

fg_bmppal()

Prototype

```
int fg_bmppal (char *filename, void *palette);  
Function fg_bmppal (ByVal filename As String, palette() As Any) As Long  
function fg_bmppal (filename : string; var palette) : integer;
```

Description

The **fg_bmppal()** function retrieves the palette of an image stored in a BMP file. The palette values are returned as RGB color components, each between 0 and 255.

Parameters

filename is name of the BMP file. The file name must be terminated by a zero byte.

palette is the name of the array that will receive the BMP palette values. The palette values are returned as RGB color components, each between 0 and 255. The first three bytes of *palette* will contain the RGB values for color 0, the next three for color 1, and so forth. The size of the *palette* array must be at least three times the number of colors in the BMP palette. You can also specify NULL for the *palette* parameter, in which case **fg_bmppal()** will return the image's color depth but no palette values.

Return value

- >0 = The number of colors in the BMP palette
- 0 = The BMP file does not have a palette (probably a 24-bit BMP file)
- 1 = The specified file does not exist
- 2 = The specified file is not a BMP file

Restrictions

none

See also

fg_bmphead(), fg_setdacs(), fg_showbmp()

Examples

Image, ImgProc

fg_bmpsize()

Prototype

```
void fg_bmpsize (void *header, int *width, int *height);  
Sub fg_bmpsize (header() As Any, width As Long, height As Long)  
procedure fg_bmpsize (var header; var width, height : integer);
```

Description

The **fg_bmpsize()** function returns the dimensions for the BMP image associated with the specified BMP file header.

Parameters

header is the name of the buffer containing the 54-byte BMP file header.

width receives the BMP image width in pixels. If *header* does not contain a valid BMP file header, *width* will be set to -1.

height receives the BMP image height in pixels. If *header* does not contain a valid BMP file header, *height* will be set to -1.

Return value

none

Restrictions

none

See also

fg_bmphead(), fg_showbmp()

Examples

Image, ImgProc

fg_box()

Prototype

```
void fg_box (int minx, int maxx, int miny, int maxy);  
Sub fg_box (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long,  
ByVal maxy As Long)  
procedure fg_box (minx, maxx, miny, maxy : integer);
```

Description

The **fg_box()** function draws an unfilled rectangle in screen space, with clipping. The width of the rectangle's edges is one pixel unless changed with **fg_boxdepth()**.

Parameters

minx is the x coordinate of the rectangle's left edge.

maxx is the x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the rectangle's top edge.

maxy is the y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_boxdepth(), fg_boxw(), fg_boxx(), fg_rect()

Examples

Graphics

fg_boxdepth()

Prototype

```
void fg_boxdepth (int xdepth, int ydepth);  
Sub fg_boxdepth (ByVal xdepth As Long, ByVal ydepth As Long)  
procedure fg_boxdepth (xdepth, ydepth : integer);
```

Description

The **fg_boxdepth()** function defines the depth of rectangles drawn with the box display functions. The **fg_vbinit()** function initializes the box depth to one pixel in each direction.

Parameters

xdepth is the width in pixels of the rectangle's left and right sides. It must be greater than zero.

ydepth is the height in pixels of the rectangle's top and bottom sides. It must be greater than zero.

Return value

none

Restrictions

none

See also

fg_box(), fg_boxw(), fg_boxx(), fg_boxxw()

fg_boxw()

Prototype

```
void fg_boxw (double xmin, double xmax, double ymin, double ymax);  
Sub fg_boxw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin As  
Double, ByVal ymax As Double)  
procedure fg_boxw (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_boxw()** function draws an unfilled rectangle in 2D world space, with clipping. The width of the rectangle's edges is one pixel unless changed with **fg_boxdepth()**.

Parameters

xmin is the world space x coordinate of the rectangle's left edge.

xmax is the world space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the rectangle's bottom edge.

ymax is the world space y coordinate of the rectangle's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

See also

fg_box(), fg_boxdepth(), fg_boxxw(), fg_rectw()

fg_boxx()

Prototype

```
void fg_boxx (int minx, int maxx, int miny, int maxy);  
Sub fg_boxx (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_boxx (minx, maxx, miny, maxy : integer);
```

Description

The **fg_boxx()** function draws an unfilled rectangle in "exclusive or" mode in screen space, with clipping. The width of the rectangle's edges is one pixel unless changed with **fg_boxdepth()**.

Parameters

minx is the x coordinate of the rectangle's left edge.

maxx is the x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the rectangle's top edge.

maxy is the y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_box(), fg_boxdepth(), fg_boxxw(), fg_rectx()

fg_boxxw()

Prototype

```
void fg_boxxw (double xmin, double xmax, double ymin, double ymax);  
Sub fg_boxxw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin As Double, ByVal ymax As Double)  
procedure fg_boxxw (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_boxxw()** function draws an unfilled rectangle in "exclusive or" mode in 2D world space, with clipping. The width of the rectangle's edges is one pixel unless changed with **fg_boxdepth()**.

Parameters

xmin is the world space x coordinate of the rectangle's left edge.

xmax is the world space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the rectangle's bottom edge.

ymax is the world space y coordinate of the rectangle's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

See also

fg_boxdepth(), fg_boxw(), fg_boxx()

fg_circle()

Prototype

```
void fg_circle (int radius);  
Sub fg_circle (ByVal radius As Long)  
procedure fg_circle (radius : integer);
```

Description

The **fg_circle()** function draws an unfilled circle in screen space. The circle is centered at the current graphics cursor position.

Parameters

radius defines the circle's radius in screen space units. It must be greater than zero.

Return value

none

Restrictions

none

See also

fg_arc(), fg_circlef(), fg_circlew(), fg_ellipse()

Examples

Graphics

fg_circlef()

Prototype

```
void fg_circlef (int radius);  
Sub fg_circlef (ByVal radius As Long)  
procedure fg_circlef (radius : integer);
```

Description

The **fg_circlef()** function draws a filled circle in screen space. The circle is centered at the current graphics cursor position and is filled with pixels of the current color.

Parameters

radius defines the circle's radius in screen space units. It must be greater than zero.

Return value

none

Restrictions

none

See also

fg_circle(), fg_circlefw(), fg_ellipsef()

fg_circlefw()

Prototype

```
void fg_circlefw (double radius);  
Sub fg_circlefw (ByVal radius As Double)  
procedure fg_circlefw (radius : real);
```

Description

The **fg_circlefw()** function draws a filled circle in 2D world space. The circle is centered at the current graphics cursor position and is filled with pixels of the current color.

Parameters

radius defines the circle's radius in horizontal world space units. It must be greater than zero.

Return value

none

Restrictions

none

See also

fg_circlef(), fg_circlew(), fg_ellipsfw()

fg_circlew()

Prototype

```
void fg_circlew (double radius);  
Sub fg_circlew (ByVal radius As Double)  
procedure fg_circlew (radius : real);
```

Description

The **fg_circlew()** function draws an unfilled circle in 2D world space. The circle is centered at the current graphics cursor position.

Parameters

radius defines the circle's radius in horizontal world space units. It must be greater than zero.

Return value

none

Restrictions

none

See also

fg_arcw(), fg_circle(), fg_circlefw(), fg_ellipsew()

fg_clip2vb()

Prototype

```
int fg_clip2vb (int minx, int maxx, int miny, int maxy, int flags);  
  
Function fg_clip2vb (ByVal minx As Long, ByVal maxx As Long, ByVal miny  
As Long, ByVal maxy As Long, ByVal flags As Long) As Long  
  
function fg_clip2vb (minx, maxx, miny, maxy, flags : integer) :  
integer;
```

Description

The **fg_clip2vb()** function copies a rectangular region from the Windows clipboard to the active virtual buffer. The region's extremes are expressed in screen space units.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*. If the clipboard image width is less than the width of the specified transfer region, *maxx* is reduced to *minx*+width-1.

miny is the y coordinate of the source region's top edge. If the clipboard image height is less than the height of the transfer region, *miny* is reduced to *maxy*-height+1.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

flags is a bit mask that controls how the resulting image is displayed and whether we cut or copy the clipboard contents, as shown here:

Bit	Value	Meaning
0	0	Update the logical palette with the clipboard palette data
0	1	Ignore the clipboard palette data
1	0	Empty clipboard after copying its contents (cut)
1	1	Keep clipboard contents intact (copy)

All other bits are reserved for future use and should be zero.

Return value

0 = Success

-1 = Another application has control of the clipboard

-2 = The clipboard does not contain a DIB image

-3 = The color depth of the clipboard image does not match the virtual buffer color depth

Restrictions

If the clipboard image width or height exceeds the size of the transfer region, the clipboard image is truncated at the boundaries of the transfer region.

See also

fg_vb2clip()

Examples

CBdemo

fg_clipdcb()

Prototype

```
void fg_clipdcb (void *bitmap, int width, int height);  
Sub fg_clipdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_clipdcb (var bitmap; width, height : integer);
```

Description

The **fg_clipdcb()** function displays a direct color bitmap, with clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Color 0 pixels will be considered transparent.

For high color virtual buffers, each pixel in the bitmap is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipmap(), fg_clpimage(), fg_drawdcb(), fg_flipdcb(), fg_getdcb(), fg_invdcb(), fg_putdcb(), fg_revdcdb()

Examples

Dcb

fg_clipmap()

Prototype

```
void fg_clipmap (void *bitmap, int width, int height);  
Sub fg_clipmap (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_clipmap (var bitmap; width, height : integer);
```

Description

The **fg_clipmap()** function displays a monochrome bitmap, with clipping. The bitmap will be displayed so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about monochrome bitmaps.

Parameters

bitmap is the name of the array containing the bitmap. Each byte of *bitmap* represents eight pixels. Bits that are set (1) result in the corresponding pixel being displayed in the current color. Bits that are reset (0) leave the corresponding pixel unchanged.

width is the bitmap width in bytes.

height is the bitmap height in bytes.

Return value

none

Restrictions

none

See also

fg_clipdcb(), fg_clpimage(), fg_drawmap(), fg_getmap(), fg_invert()

Examples

Monomap

fg_clipmask()

Prototype

```
void fg_clipmask (void *bitmap, int runs, int width);  
Sub fg_clipmask (bitmap() As Any, ByVal runs As Long, ByVal width As Long)  
procedure fg_clipmask (var bitmap; runs, width : integer);
```

Description

The **fg_clipmask()** legacy function displays a masking map, with clipping. The masking map will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the masking map. The masking map is a series of alternating "protect" and "zero" pixel runs. The "protect" runs leave the corresponding virtual buffer pixels unchanged, while the "zero" runs set them to color zero. The length of each run must be between 0 and 255.

runs is the number of pixel runs in the masking map.

width is the masking map width in pixels.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions.

fg_clpimage()

Prototype

```
void fg_clpimage (void *bitmap, int width, int height);  
Sub fg_clpimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_clpimage (var bitmap; width, height : integer);
```

Description

The **fg_clpimage()** function displays a 256-color bitmap, with clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clipdcb(), fg_clipmap(), fg_drwimage(), fg_flpimage(), fg_getimage(), fg_invert(),
fg_putimage(), fg_revimage(), fg_setclip()

Examples

Bitmap, Fishtank

fg_clprect()

Prototype

```
void fg_clprect (int minx, int maxx, int miny, int maxy);  
Sub fg_clprect (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_clprect (minx, maxx, miny, maxy : integer);
```

Description

The **fg_clprect()** function draws a solid (filled) rectangle in screen space, with clipping.

Parameters

minx is the screen space x coordinate of the rectangle's left edge.

maxx is the screen space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the screen space y coordinate of the rectangle's top edge.

maxy is the screen space y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_clprectw(), fg_clprectx(), fg_rect(), fg_rectw(), fg_setclip()

fg_clprectw()

Prototype

```
void fg_clprectw (double xmin, double xmax, double ymin, double ymax);  
Sub fg_clprectw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin  
As Double, ByVal ymax As Double)  
procedure fg_clprectw (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_clprectw()** function draws a solid (filled) rectangle in 2D world space, with clipping.

Parameters

xmin is the world space x coordinate of the rectangle's left edge.

xmax is the world space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the rectangle's bottom edge.

ymax is the world space y coordinate of the rectangle's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

See also

fg_clprect(), fg_rect(), fg_rectw(), fg_setclipw()

fg_clprectx()

Prototype

```
void fg_clprectx (int minx, int maxx, int miny, int maxy);  
Sub fg_clprectx (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_clprectx (minx, maxx, miny, maxy : integer);
```

Description

The **fg_clprectx()** function draws a solid (filled) rectangle in “exclusive or” mode in screen space, with clipping.

Parameters

minx is the x coordinate of the rectangle's left edge.

maxx is the x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the rectangle's top edge.

maxy is the y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_clprect(), fg_rectw(), fg_setclip()

fg_colors()

Prototype

```
int fg_colors (void);  
Function fg_colors () As Long  
function fg_colors : integer;
```

Description

The **fg_colors()** function returns the display driver's color depth in bits per pixel.

Parameters

none

Return value

The display driver's color depth in bits per pixel. For example, a return value of 8 means the display driver is currently set to 8 bits per pixel, or 256 colors.

Restrictions

none

See also

[fg_getdepth\(\)](#)

Examples

Colors, Cube, Display, Fade, FirstDD, Rainbow, TMcubeX, Tunnel

fg_contdcb()

Prototype

```
void fg_contdcb (void *source, void *dest, int lower, int upper, int
size);

Sub fg_contdcb (source() As Any, dest() As Any, ByVal lower As Long,
ByVal upper As Long, ByVal size As Long)

procedure fg_contdcb (var source, dest; lower, upper, size : integer);
```

Description

The **fg_contdcb()** function applies a contrast enhancement transform to a direct color bitmap.

Parameters

source is the name of the array containing the direct color bitmap to be transformed.

dest is the name of the array that will receive the resulting transformed bitmap.

lower is the lower bound. All color components below this value will be set to zero. It must be between 0 and 254.

upper is the upper bound. All color components above this value will be set to 255. It must be between 1 and 255 and greater than the lower bound.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_contrgb(), fg_contvb()

fg_contrgb()

Prototype

```
void fg_contrgb (void *values, int lower, int upper, int count);  
Sub fg_contrgb (values() As Any, ByVal lower As Long, ByVal upper As  
Long, ByVal count As Long)  
procedure fg_contrgb (var values; lower, upper, count : integer);
```

Description

The **fg_contrgb()** function applies a contrast enhancement transform to a series of RGB color triples.

Parameters

values is the name of the array containing the RGB color components, arranged as three-byte RGB triples. Each RGB color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least $3 \times \text{count}$ bytes.

lower is the lower bound. All color components below this value will be set to zero. It must be between 0 and 254.

upper is the upper bound. All color components above this value will be set to 255. It must be between 1 and 255 and greater than the lower bound.

count is the number of RGB color triples to transform.

Return value

none

Restrictions

none

See also

fg_contdcb(), fg_contvb()

fg_contvb()

Prototype

```
void fg_contvb (int lower, int upper, int width, int height);  
Sub fg_contvb (ByVal lower As Long, ByVal upper As Long, ByVal width As Long, ByVal height As Long)  
procedure fg_contvb (lower, upper, width, height : integer);
```

Description

The **fg_contvb()** function applies a contrast enhancement transform to a rectangular region of the active virtual buffer. The region's lower left corner is at the current graphics position.

Parameters

lower is the lower bound. All color components below this value will be set to zero. It must be between 0 and 254.

upper is the upper bound. All color components above this value will be set to 255. It must be between 1 and 255 and greater than the lower bound.

width is the region's width in pixels.

height is the region's height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_contdcb(), fg_contrgb()

Examples

ImgProc

fg_copypage()

Prototype

```
void fg_copypage (int source, int dest);  
Sub fg_copypage (ByVal source As Long, ByVal dest As Long)  
procedure fg_copypage (source, dest : integer);
```

Description

The **fg_copypage()** function transfers the contents of one virtual buffer to another virtual buffer with the same dimensions. Assuming *maxx* and *maxy* represent the maximum x and y coordinates of the virtual buffers, the call

```
fg_copypage(source,dest);
```

is equivalent to

```
fg_vbcopy(0, maxx, 0, maxy, 0, maxy, source, dest);
```

Parameters

source is the source virtual buffer handle.

dest is the destination virtual buffer handle.

Return value

none

Restrictions

The source and destination virtual buffers *must* have the same color depth and same dimensions. If they have the same color depth but not the same dimensions, use **fg_vbcopy()** instead of **fg_copypage()**.

When using DirectX, the source or destination virtual buffers must not be locked.

See also

fg_vbcopy(), fg_vbopen()

Examples

Fishtank, ImgProc

fg_cut()

Prototype

```
void fg_cut (void *bitmap, void *section, int xpos, int ypos, int
width, int sec_width, int sec_height);

Sub fg_cut (bitmap() As Any, section() As Any, ByVal xpos As Long,
ByVal ypos As Long, ByVal width As Long, ByVal sec_width As Long, ByVal
sec_height As Long)

procedure fg_cut (var bitmap, section; xpos, ypos, width, sec_width,
sec_height : integer);
```

Description

The **fg_cut()** function extracts a 256-color bitmap section from a 256-color bitmap.

Parameters

bitmap is the name of the array containing the source bitmap. This bitmap is not modified in any way.

section is the name of the array to receive the bitmap section. Its size must be at least *sec_width* * *sec_height* bytes.

xpos is the x coordinate within the source bitmap that defines the left edge of the bitmap section. It must be greater than or equal to zero, but less than *width*.

ypos is the y coordinate within the source bitmap that defines the bottom edge of the bitmap section. It must be greater than or equal to zero, but less than *height*. The bottom row of a bitmap is considered row zero.

width is the source bitmap width in pixels.

sec_width is the bitmap section width in pixels.

sec_height is the bitmap section height in pixels.

Return value

none

Restrictions

none

See also

fg_cutdcb(), fg_paste()

fg_cutdcb()

Prototype

```
void fg_cutdcb (void *bitmap, void *section, int xpos, int ypos, int
width, int sec_width, int sec_height);

Sub fg_cutdcb (bitmap() As Any, section() As Any, ByVal xpos As Long,
ByVal ypos As Long, ByVal width As Long, ByVal sec_width As Long, ByVal
sec_height As Long)

procedure fg_cutdcb (var bitmap, section; xpos, ypos, width, sec_width,
sec_height : integer);
```

Description

The **fg_cutdcb()** function extracts a direct color bitmap section from a direct color bitmap.

Parameters

bitmap is the name of the array containing the source bitmap. This bitmap is not modified in any way.

section is the name of the array to receive the bitmap section. It must be large enough to hold a DCB of *sec_width* * *sec_height* pixels.

xpos is the x coordinate within the source bitmap that defines the left edge of the bitmap section. It must be greater than or equal to zero, but less than *width*.

ypos is the y coordinate within the source bitmap that defines the bottom edge of the bitmap section. It must be greater than or equal to zero, but less than *height*. The bottom row of a bitmap is considered row zero.

width is the source bitmap width in pixels.

sec_width is the bitmap section width in pixels.

sec_height is the bitmap section height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_cut(), fg_pastedcb()

fg_dash()

Prototype

```
void fg_dash (int ix, int iy, int pattern);  
Sub fg_dash (ByVal ix As Long, ByVal iy As Long, ByVal pattern As Long)  
procedure fg_dash (ix, iy, pattern : integer);
```

Description

The **fg_dash()** function draws a dashed line from the graphics cursor position to an absolute screen space position. It also makes the destination position the new graphics cursor position.

Parameters

ix is the screen space x coordinate of the destination position.

iy is the screen space y coordinate of the destination position.

pattern represents a 16-bit cyclic dash pattern. Bits that are 1 will result in a pixel being drawn; bits that are 0 will result in a pixel being skipped.

Return value

none

Restrictions

none

See also

fg_dashrel(), fg_dashrw(), fg_dashw(), fg_move()

fg_dashrel()

Prototype

```
void fg_dashrel (int ix, int iy, int pattern);  
Sub fg_dashrel (ByVal ix As Long, ByVal iy As Long, ByVal pattern As Long)  
procedure fg_dashrel (ix, iy, pattern : integer);
```

Description

The **fg_dashrel()** function draws a dashed line from the graphics cursor position to a screen space position relative to it. It also makes the destination position the new graphics cursor position.

Parameters

ix is the screen space x offset of the destination position.

iy is the screen space y offset of the destination position.

pattern represents a 16-bit cyclic dash pattern. Bits that are 1 will result in a pixel being drawn; bits that are 0 will result in a pixel being skipped.

Return value

none

Restrictions

none

See also

fg_dash(), fg_dashrw(), fg_dashw(), fg_moverel()

fg_dashrw()

Prototype

```
void fg_dashrw (double x, double y, int pattern);  
Sub fg_dashrw (ByVal x As Double, ByVal y As Double, ByVal pattern As  
Long)  
procedure fg_dashrw (x, y : real; pattern : integer);
```

Description

The **fg_dashrw()** function draws a dashed line from the graphics cursor position to a 2D world space position relative to it. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space x offset of the destination position.

y is the world space y offset of the destination position.

pattern represents a 16-bit cyclic dash pattern. Bits that are 1 will result in a pixel being drawn; bits that are 0 will result in a pixel being skipped.

Return value

none

Restrictions

none

See also

fg_dash(), fg_dashrel(), fg_dashw(), fg_moverw()

fg_dashw()

Prototype

```
void fg_dashw (double x, double y, int pattern);  
Sub fg_dashw (ByVal x As Double, ByVal y As Double, ByVal pattern As  
Long)  
procedure fg_dashw (x, y : real; pattern : integer);
```

Description

The **fg_dashw()** function draws a dashed line from the graphics cursor position to an absolute 2D world space position. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space x coordinate of the destination position.

y is the world space y coordinate of the destination position.

pattern represents a 16-bit cyclic dash pattern. Bits that are 1 will result in a pixel being drawn; bits that are 0 will result in a pixel being skipped.

Return value

none

Restrictions

none

See also

fg_dash(), fg_dashrel(), fg_dashrw(), fg_movew()

fg_ddapply()

Prototype

```
void fg_ddapply (void);  
Sub fg_ddapply ()  
procedure fg_ddapply;
```

Description

The **fg_ddapply()** function activates Fastgraph's DirectX settings defined through **fg_ddsetobj()**.

Parameters

none

Return value

none

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddsetobj()

Examples

SetupD3D, SetupDD

fg_ddflip()

Prototype

```
int fg_ddflip (void);  
Function fg_ddflip () As Long  
function fg_ddflip : integer;
```

Description

The **fg_ddflip()** function moves or "flips" the DirectDraw back buffer surface to the screen. It is meaningful only for programs that specify the FG_DX_FLIP flag through **fg_ddsetup()**.

Parameters

none

Return value

The **fg_ddflip()** return value will be zero if successful. Otherwise, the return value will be a standard error code listed in Microsoft's DirectX documentation.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddflipnw(), fg_ddsetup(), fg_gdflip()

Examples

FrameDD, TMcubeX

fg_ddflipnw()

Prototype

```
int fg_ddflipnw (void);  
Function fg_ddflipnw () As Long  
function fg_ddflipnw : integer;
```

Description

The **fg_ddflipnw()** function moves or "flips" the DirectDraw back buffer surface to the screen, without waiting for the display hardware to finish drawing if it is busy. It is meaningful only for programs that specify the FG_DX_FLIP flag through **fg_ddsetup()**.

Parameters

none

Return value

The **fg_ddflipnw()** return value will be zero if successful, or the DirectDraw error DDERR_WASSTILLDRAWING if the flip operation could not be performed because the display hardware was busy. Otherwise, the return value will be a standard error code listed in Microsoft's DirectX documentation.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddflip(), fg_ddsetup()

fg_ddframe()

Prototype

```
void fg_ddframe (int state);  
Sub fg_ddframe (ByVal state As Long)  
procedure fg_ddframe (state : integer);
```

Description

The **fg_ddframe()** function begins or ends a Direct3D rendering sequence. It does nothing unless Direct3D is being used.

Parameters

state defines whether we are beginning or ending a Direct3D rendering sequence. If *state* is 0, **fg_ddframe()** begins a rendering sequence. If *state* is 1, **fg_ddframe()** ends a rendering sequence.

Return value

none

Restrictions

The **fg_ddframe()** function does nothing if Direct3D is not being used.

This function is available only in Fastgraph's DirectX libraries.

Examples

TMcubeX

fg_ddfreedc()

Prototype

```
void fg_ddfreedc (HDC hdc);  
Sub fg_ddfreedc (ByVal hdc As Long)  
procedure fg_ddfreedc (hdc : HDC);
```

Description

The **fg_ddfreedc()** function releases the device context for the active virtual buffer, and unlocks the DirectDraw surface associated with that virtual buffer.

Parameters

hdc is the handle of the device context to release, as returned by **fg_ddgetdc()**.

Return value

none

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

[fg_ddgetdc\(\)](#)

fg_ddgetdc()

Prototype

```
HDC fg_ddgetdc (void);  
Function fg_ddgetdc () As Long  
function fg_ddgetdc : HDC;
```

Description

The **fg_ddgetdc()** function creates a device context for the active virtual buffer, and locks the DirectDraw surface associated with that virtual buffer.

Parameters

none

Return value

A handle to the device context.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddfreedc(), fg_fontdc()

fg_ddgetobj()

Prototype

```
int fg_ddgetobj (int code);  
Function fg_ddgetobj (ByVal code As Long) As Long  
function fg_ddgetobj (code : integer) : integer;
```

Description

The **fg_ddgetobj()** function returns a pointer to the requested DirectX object.

Parameters

code specifies the DirectX object:

- 0 DirectDraw2 object
- 1 DirectDrawSurface object for the primary surface
- 2 DirectDrawClipper object
- 3 DirectDrawPalette object
- 4 Direct3D2 object
- 5 Direct3DDevice2 object (if using hardware acceleration)
- 6 Direct3DDevice2 object (if using software rendering)
- 7 Direct3DViewport2 object
- 8 DirectDrawSurface object for the Direct3D z-buffer

Return value

A pointer to the requested DirectX object. If the *code* parameter does not specify one of the values listed above, the return value is zero.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

The DirectDrawClipper object is meaningful only for windowed programs.

The DirectDrawPalette object is meaningful only for full screen programs.

See also

fg_ddsetobj()

fg_ddlock()

Prototype

```
int fg_ddlock (void);  
Function fg_ddlock () As Long  
function fg_ddlock : integer;
```

Description

The **fg_ddlock()** function locks the DirectDraw surface associated with the active virtual buffer and returns its address. The address will be valid until you unlock the surface.

Parameters

none

Return value

The address of the DirectDraw surface associated with the active virtual buffer. If no virtual buffer is active, the return value will be zero.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddunlock()

fg_ddsetobj()

Prototype

```
void fg_ddsetobj (void *object, int code);  
Sub fg_ddsetobj (object As Any, ByVal code As Long)  
procedure fg_ddsetobj (var object; code : integer);
```

Description

The **fg_ddsetobj()** function defines a pointer to an externally created DirectX object. Normally you will only use this function in specialized applications, or when using Fastgraph with third party DirectX tools.

Parameters

object is a pointer to the DirectX object.

code specifies which DirectX object to define:

- 0 DirectDraw2 object
- 1 DirectDrawSurface object for the primary surface
- 2 DirectDrawClipper object
- 3 DirectDrawPalette object
- 4 Direct3D2 object
- 5 Direct3DDevice2 object (if using hardware acceleration)
- 6 Direct3DDevice2 object (if using software rendering)
- 7 Direct3DViewport2 object
- 8 DirectDrawSurface object for the Direct3D z-buffer

Return value

none

Restrictions

The DirectDrawClipper object is used only in windowed programs.

The DirectDrawPalette object is used only in full screen programs.

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddapply(), fg_ddgetobj()

Examples

SetupD3D, SetupDD

fg_ddsetup()

Prototype

```
void fg_ddsetup (int width, int height, int depth, int flags);  
Sub fg_ddsetup (ByVal width As Long, ByVal height As Long, ByVal depth  
As Long, ByVal flags As Long)  
procedure fg_ddsetup (width, height, depth, flags : integer);
```

Description

The **fg_ddsetup()** function defines the display resolution, color depth, and other DirectX information for full screen programs. Calling **fg_ddsetup()** does not actually set the requested display mode but merely defines how **fg_vbinit()** will initialize DirectX.

Parameters

width is the horizontal resolution in pixels.

height is the vertical resolution in pixels.

depth is the color depth in bits per pixel. It must be 8, 16, 24, or 32. You can also specify a *depth* of zero, which is used in hybrid programs that switch between full screen and windowed DirectDraw. If *depth* is zero, **fg_ddsetup()** ignores the other three parameters.

flags is a series of flags that specifies how Fastgraph will use DirectX, as summarized here:

Flag	Meaning
FG_DX_BLIT	Use DirectDraw blitting
FG_DX_FLIP	Use DirectDraw page flipping (required for Direct3D)
FG_DX_RENDER_FG	Use Fastgraph rendering for 3D functions
FG_DX_RENDER_SW	Use Direct3D software rendering if available
FG_DX_RENDER_HW	Use Direct3D hardware acceleration if available
FG_DX_ZBUFFER	Create a Direct3D z-buffer
FG_DX_TCDEPTH	Use specified true color bit depth only

If FG_DX_BLIT is specified, FG_DX_RENDER_xx and FG_DX_ZBUFFER are ignored because these flags are specific to Direct3D programs. FG_DX_TCDEPTH is ignored unless the *depth* parameter is 24 or 32.

Return value

none

Restrictions

The combination of *width*, *height*, and *depth* parameters must define a valid DirectDraw screen resolution and color depth.

For Direct3D programs, you must specify the FG_DX_FLIP flag, and the *depth* parameter must be 16, 24, or 32.

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddflip(), fg_ddstatus(), fg_ddusage(), fg_vbinit()

Examples

FrameDD, FullScr, TMcubeX

fg_ddstatus()

Prototype

```
int fg_ddstatus (void);  
Function fg_ddstatus () As Long  
function fg_ddstatus : integer;
```

Description

The **fg_ddstatus()** function returns error code resulting from the most recent DirectDraw or Direct3D operation.

Parameters

none

Return value

If no DirectDraw or Direct3D error occurred, the return value will be zero. Otherwise, the return value will be one of the error codes listed in Microsoft's DirectX documentation.

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddsetup(), fg_ddusage(), fg_vbinit()

fg_ddunlock()

Prototype

```
void fg_ddunlock (void);  
Sub fg_ddunlock ()  
procedure fg_ddunlock;
```

Description

The **fg_ddunlock()** function unlocks the DirectDraw surface associated with the active virtual buffer.

Parameters

none

Return value

none

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

[fg_ddlock\(\)](#)

fg_ddusage()

Prototype

```
int fg_ddusage (void);  
Function fg_ddusage () As Long  
function fg_ddusage : integer;
```

Description

The **fg_ddusage()** function returns a value indicating how Fastgraph is using DirectX.

Parameters

none

Return value

- 1 = DirectX not available
- 0 = Using DirectX with Fastgraph's 3D rendering
- 1 = Using DirectX with Direct3D software rendering
- 2 = Using DirectX with Direct3D hardware acceleration

Restrictions

This function is meaningful only after calling **fg_vbinit()**.

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddsetup(), fg_ddstatus(), fg_vbinit()

fg_defcolor()

Prototype

```
void fg_defcolor (int index, int value);  
Sub fg_defcolor (ByVal index As Long, ByVal value As Long)  
procedure fg_defcolor (index, value : integer);
```

Description

The **fg_defcolor()** legacy function assigns a color value to a virtual color index.

Parameters

index is the virtual color index to define, between 0 and 255.

value is the color value to assign to the specified color index, between 0 and 255.

Return value

none

Restrictions

none

fg_defpal()

Prototype

```
HPALETTE fg_defpal (void);  
Function fg_defpal () As Long  
function fg_defpal : HPALETTE;
```

Description

The **fg_defpal()** function creates a logical palette containing Fastgraph's default colors. It often appears in the WM_CREATE message handler. See Chapter 3 of the *Fastgraph 6.0 User's Guide* for a list of colors in the default logical palette.

Parameters

none

Return value

A Windows handle to the new logical palette. If an error occurs in creating the logical palette, **fg_defpal()** returns zero.

Restrictions

none

See also

fg_getdacs(), fg_getrgb(), fg_logpal(), fg_realize(), fg_setdacs(), fg_setrgb()

Examples

Nearly all the example programs use this function.

fg_dispfile()

Prototype

```
void fg_dispfile (char *filename, int width, int format);  
Sub fg_dispfile (ByVal filename As String, ByVal width As Long, ByVal  
format As Long)  
procedure fg_dispfile (filename : string; width, format : integer);
```

Description

The **fg_dispfile()** legacy function displays an image from a standard or packed pixel run file. The image will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

filename is the name of the PPR or SPR file. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte).

width is the image width in pixels. It must be greater than zero.

format specifies the image format. The value of *format* must be 0 if the image is in standard pixel run format, and 1 if the image is in packed pixel run format.

Return value

none

Restrictions

none

Replaced by

BMP and PCX display functions

fg_display()

Prototype

```
void fg_display (void *bitmap, int runs, int width);
```

```
Sub fg_display (bitmap() As Any, ByVal runs As Long, ByVal width As Long)
```

```
procedure fg_display (var bitmap; runs, width : integer);
```

Description

The **fg_display()** legacy function displays an image stored in Fastgraph's standard pixel run format, where the image resides in an array. The image will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the pixel run map. The pixel run map is a sequence of (color,count) pairs. Each "color" element is a value between 0 and 255 specifying the color for that pixel run. Each "count" element is a value between 0 and 255 specifying the length in pixels of that pixel run.

runs is the number of pixel runs to display from the pixel run map. It is normally one-half the size of the *bitmap* array.

width is the image width in pixels. It must be greater than zero.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_displayp()

Prototype

```
void fg_displayp (void *bitmap, int runs, int width);  
Sub fg_displayp (bitmap() As Any, ByVal runs As Long, ByVal width As Long)  
procedure fg_displayp (var bitmap; runs, width : integer);
```

Description

The **fg_displayp()** legacy function displays an image stored in Fastgraph's packed pixel run format, where the image resides in an array. The image will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the pixel run map. The pixel run map is a sequence of (color,count) pairs. Each "color" element is a value between 0 and 15 specifying the color for that pixel run. Each "count" element is a value between 0 and 255 specifying the length in pixels of that pixel run.

runs is the number of pixel runs to display from the pixel run map. It is normally two-thirds the size of the *bitmap* array.

width is the image width in pixels. It must be greater than zero.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_draw()

Prototype

```
void fg_draw (int ix, int iy);  
Sub fg_draw (ByVal ix As Long, ByVal iy As Long)  
procedure fg_draw (ix, iy : integer);
```

Description

The **fg_draw()** function draws a solid line from the graphics cursor position to an absolute screen space position. It also makes the destination position the new graphics cursor position.

Parameters

ix is the screen space x coordinate of the destination position.

iy is the screen space y coordinate of the destination position.

Return value

none

Restrictions

none

See also

fg_3Dline(), fg_drawrel(), fg_drawww(), fg_drawx(), fg_move()

Examples

Graphics, Scroller

fg_drawdcb()

Prototype

```
void fg_drawdcb (void *bitmap, int width, int height);  
Sub fg_drawdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_drawdcb (var bitmap; width, height : integer);
```

Description

The **fg_drawdcb()** function displays a direct color bitmap, without clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Color 0 pixels will be considered transparent.

For high color virtual buffers, each pixel in the bitmap is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipdcb(), fg_drawmap(), fg_drwimage(), fg_flipdcb(), fg_getdcb(), fg_invdcb(), fg_putdcb(), fg_revdcdb()

Examples

Dcb

fg_drawmap()

Prototype

```
void fg_drawmap (void *bitmap, int width, int height);  
Sub fg_drawmap (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_drawmap (var bitmap; width, height : integer);
```

Description

The **fg_drawmap()** function displays a monochrome bitmap. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about monochrome bitmaps.

Parameters

bitmap is the name of the array containing the bitmap. Each byte of *bitmap* represents eight pixels. Bits that are set (1) result in the corresponding pixel being displayed in the current color. Bits that are reset (0) leave the corresponding pixel unchanged.

width is the bitmap width in bytes.

height is the bitmap height in bytes.

Return value

none

Restrictions

none

See also

fg_clipmap(), fg_drawdcb(), fg_drwimage(), fg_getmap(), fg_invert()

Examples

Monomap

fg_drawmask()

Prototype

```
void fg_drawmask (void *bitmap, int runs, int width);  
Sub fg_drawmask (bitmap() As Any, ByVal runs As Long, ByVal width As Long)  
procedure fg_drawmask (var bitmap; runs, width : integer);
```

Description

The **fg_drawmask()** legacy function displays a masking map. The masking map will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the masking map. The masking map is a series of alternating "protect" and "zero" pixel runs. The "protect" runs leave the corresponding virtual buffer pixels unchanged, while the "zero" runs set them to color zero. The length of each run must be between 0 and 255.

runs is the number of pixel runs in the masking map.

width is the masking map width in pixels.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_drawrel()

Prototype

```
void fg_drawrel (int ix, int iy);  
Sub fg_drawrel (ByVal ix As Long, ByVal iy As Long)  
procedure fg_drawrel (ix, iy : integer);
```

Description

The **fg_drawrel()** function draws a solid line from the graphics cursor position to a screen space position relative to it. It also makes the destination position the new graphics cursor position.

Parameters

ix is the screen space x offset of the destination position.

iy is the screen space y offset of the destination position.

Return value

none

Restrictions

none

See also

fg_draw(), fg_drawrelx(), fg_drawrw(), fg_moverel()

fg_drawrelx()

Prototype

```
void fg_drawrelx (int ix, int iy);  
Sub fg_drawrelx (ByVal ix As Long, ByVal iy As Long)  
procedure fg_drawrelx (ix, iy : integer);
```

Description

The **fg_drawrelx()** function draws a solid line in "exclusive or" mode from the graphics cursor position to a screen space position relative to it. The destination position becomes the new graphics cursor position.

Parameters

ix is the screen space x offset of the destination position.
iy is the screen space y offset of the destination position.

Return value

none

Restrictions

none

See also

fg_drawrel(), fg_drawrxw(), fg_drawx(), fg_moverel()

fg_drawrw()

Prototype

```
void fg_drawrw (double x, double y);  
Sub fg_drawrw (ByVal x As Double, ByVal y As Double)  
procedure fg_drawrw (x, y : real);
```

Description

The **fg_drawrw()** function draws a solid line from the graphics cursor position to a 2D world space position relative to it. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space *x* offset of the destination position.
y is the world space *y* offset of the destination position.

Return value

none

Restrictions

none

See also

[fg_drawrel\(\)](#), [fg_drawrxw\(\)](#), [fg_draww\(\)](#), [fg_moverw\(\)](#)

fg_drawrxw()

Prototype

```
void fg_drawrxw (double x, double y);  
Sub fg_drawrxw (ByVal x As Double, ByVal y As Double)  
procedure fg_drawrxw (x, y : real);
```

Description

The **fg_drawrxw()** function draws a solid line in "exclusive or" mode from the graphics cursor position to a 2D world space position relative to it. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space x offset of the destination position.
y is the world space y offset of the destination position.

Return value

none

Restrictions

none

See also

fg_drawrelx(), fg_drawrw(), fg_drawxw(), fg_moverw()

fg_draww()

Prototype

```
void fg_draww (double x, double y);  
Sub fg_draww (ByVal x As Double, ByVal y As Double)  
procedure fg_draww (x, y : real);
```

Description

The **fg_draww()** function draws a solid line from the graphics cursor position to an absolute 2D world space position. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space x coordinate of the destination position.

y is the world space y coordinate of the destination position.

Return value

none

Restrictions

none

See also

fg_draw(), fg_drawrw(), fg_drawxw(), fg_movew()

Examples

SWchars

fg_drawx()

Prototype

```
void fg_drawx (int ix, int iy);  
Sub fg_drawx (ByVal ix As Long, ByVal iy As Long)  
procedure fg_drawx (ix, iy : integer);
```

Description

The **fg_drawx()** function draws a solid line in "exclusive or" mode from the graphics cursor position to an absolute screen space position. It also makes the destination position the new graphics cursor position.

Parameters

ix is the screen space x coordinate of the destination position.
iy is the screen space y coordinate of the destination position.

Return value

none

Restrictions

none

See also

fg_draw(), fg_drawrelx(), fg_drawxw(), fg_move()

fg_drawxw()

Prototype

```
void fg_drawxw (double x, double y);  
Sub fg_drawxw (ByVal x As Double, ByVal y As Double)  
procedure fg_drawxw (x, y : real);
```

Description

The **fg_drawxw()** function draws a solid line in "exclusive or" mode from the graphics cursor position to an absolute 2D world space position. It also makes the destination position the new graphics cursor position.

Parameters

x is the world space x coordinate of the destination position.
y is the world space y coordinate of the destination position.

Return value

none

Restrictions

none

See also

fg_drawrxw(), fg_drawww(), fg_drawx(), fg_movew()

fg_drawz()

Prototype

```
void fg_drawz (int ix, int iy, double z_start, double z_end);  
Sub fg_drawz (ByVal ix As Long, ByVal iy As Long, ByVal z_start As  
Double, ByVal z_end As Double)  
procedure fg_drawz (ix, iy : integer; z_start, z_end : double);
```

Description

The **fg_drawz()** function draws a projected z-buffered line from the graphics cursor position to an absolute screen space position, with 2D clipping. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

ix is the screen space x coordinate of the destination position.
iy is the screen space y coordinate of the destination position.
z_start is the 3D world space z coordinate at the graphics cursor position.
z_end is the 3D world space z coordinate at the destination position.

Return value

none

Restrictions

none

See also

fg_3Dline(), fg_3Drenderstate()

fg_drect()

Prototype

```
void fg_drect (int minx, int maxx, int miny, int maxy, void *matrix);  
Sub fg_drect (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long, matrix() As Any)  
procedure fg_drect (minx, maxx, miny, maxy : integer; var matrix);
```

Description

The **fg_drect()** function draws a dithered rectangle in screen space, without clipping.

Parameters

minx is the screen space x coordinate of the rectangle's left edge.

maxx is the screen space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the screen space y coordinate of the rectangle's top edge.

maxy is the screen space y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

matrix is an eight-byte array that defines the 4x2 dithering matrix. See Chapter 5 of the *Fastgraph 6.0 User's Guide* for more information about the dithering matrix.

Return value

none

Restrictions

none

See also

fg_drectw(), fg_rect()

fg_drectw()

Prototype

```
void fg_drectw (double xmin, double xmax, double ymin, double ymax,  
void *matrix);  
  
Sub fg_drectw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin  
As Double, ByVal ymax As Double, matrix() As Any)  
  
procedure fg_drectw (xmin, xmax, ymin, ymax : real; var matrix);
```

Description

The **fg_drectw()** function draws a dithered rectangle in 2D world space, without clipping.

Parameters

xmin is the world space x coordinate of the rectangle's left edge.

xmax is the world space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the rectangle's bottom edge.

ymax is the world space y coordinate of the rectangle's top edge. It must be greater than or equal to the value of *ymin*.

matrix is an eight-byte array that defines the 4x2 dithering matrix. See Chapter 5 of the *Fastgraph 6.0 User's Guide* for more information about the dithering matrix.

Return value

none

Restrictions

none

See also

fg_direct(), fg_rectw()

fg_drwimage()

Prototype

```
void fg_drwimage (void *bitmap, int width, int height);  
Sub fg_drwimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_drwimage (var bitmap; width, height : integer);
```

Description

The **fg_drwimage()** function displays a 256-color bitmap, without clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clpimage(), fg_drawdcb(), fg_drawmap(), fg_flpimage(), fg_getimage(), fg_invert(), fg_putimage(), fg_revimage()

Examples

Bitmap, Rotate, Scale

fg_ellipse()

Prototype

```
void fg_ellipse (int horiz, int vert);  
Sub fg_ellipse (ByVal horiz As Long, ByVal vert As Long)  
procedure fg_ellipse (horiz, vert : integer);
```

Description

The **fg_ellipse()** function draws an unfilled ellipse in screen space. The ellipse is centered at the current graphics cursor position, and its size is determined by the specified lengths of its semi-axes.

Parameters

horiz is the length of the ellipse's horizontal semi-axis (the absolute screen space distance from the center of the ellipse to its horizontal extremity).

vert is the length of the ellipse's vertical semi-axis (the absolute screen space distance from the center of the ellipse to its vertical extremity).

Return value

none

Restrictions

none

See also

fg_arc(), fg_circle(), fg_ellipsef(), fg_ellipsew()

Examples

Graphics

fg_ellipsef()

Prototype

```
void fg_ellipsef (int horiz, int vert);  
Sub fg_ellipsef (ByVal horiz As Long, ByVal vert As Long)  
procedure fg_ellipsef (horiz, vert : integer);
```

Description

The **fg_ellipsef()** function draws a filled ellipse in screen space. The ellipse is centered at the current graphics cursor position, and its size is determined by the specified lengths of its semi-axes. The ellipse is filled with pixels of the current color.

Parameters

horiz is the length of the ellipse's horizontal semi-axis (the absolute screen space distance from the center of the ellipse to its horizontal extremity).

vert is the length of the ellipse's vertical semi-axis (the absolute screen space distance from the center of the ellipse to its vertical extremity).

Return value

none

Restrictions

none

See also

fg_circlef(), fg_ellipse(), fg_ellipsefw()

Examples

Rainbow

fg_ellipsew()

Prototype

```
void fg_ellipsew (double horiz, double vert);  
Sub fg_ellipsew (ByVal horiz As Double, ByVal vert As Double)  
procedure fg_ellipsew (horiz, vert : real);
```

Description

The **fg_ellipsew()** function draws an unfilled ellipse in 2D world space. The ellipse is centered at the current graphics cursor position, and its size is determined by the specified lengths of its semi-axes.

Parameters

horiz defines the horizontal semi-axis of the ellipse (the absolute world space distance from the center of the ellipse to its horizontal extremity).

vert defines the vertical semi-axis of the ellipse (the absolute world space distance from the center of the ellipse to its vertical extremity).

Return value

none

Restrictions

none

See also

fg_arcw(), fg_circlew(), fg_ellipse(), fg_ellipsfw()

fg_ellipsfw()

Prototype

```
void fg_ellipsfw (double horiz, double vert);  
Sub fg_ellipsfw (ByVal horiz As Double, ByVal vert As Double)  
procedure fg_ellipsfw (horiz, vert : real);
```

Description

The **fg_ellipsfw()** function draws a filled ellipse in 2D world space. The ellipse is centered at the current graphics cursor position, and its size is determined by the specified lengths of its semi-axes. The ellipse is filled with pixels of the current color.

Parameters

horiz defines the horizontal semi-axis of the ellipse (the absolute world space distance from the center of the ellipse to its horizontal extremity).

vert defines the vertical semi-axis of the ellipse (the absolute world space distance from the center of the ellipse to its vertical extremity).

Return value

none

Restrictions

none

See also

fg_circlefw(), fg_ellipsew()

fg_erase()

Prototype

```
void fg_erase (void);  
Sub fg_erase ()  
procedure fg_erase;
```

Description

The **fg_erase()** function fills the active virtual buffer with color 0 pixels.

Parameters

none

Return value

none

Restrictions

When using DirectX, the active virtual buffer must not be locked.

See also

[fg_fillpage\(\)](#)

Examples

CBdemo, Fishtank, VBdemo

fg_fillpage()

Prototype

```
void fg_fillpage (void);  
Sub fg_fillpage ()  
procedure fg_fillpage;
```

Description

The **fg_fillpage()** function fills the active virtual buffer with pixels of the current color.

Parameters

none

Return value

none

Restrictions

When using DirectX, the active virtual buffer must not be locked.

See also

`fg_erase()`, `fg_setcolor()`

Examples

Nearly all the example programs use this function.

fg_findrgb()

Prototype

```
int fg_findrgb (int red, int green, int blue);  
Function fg_findrgb (ByVal red As Long, ByVal green As Long, ByVal blue  
As Long) As Long  
function fg_findrgb (red, green, blue : integer) : integer;
```

Description

The **fg_findrgb()** legacy function finds the color closest to the specified color in the active logical palette.

Parameters

red, *green*, and *blue* respectively define the red, green, and blue components of the target color.

Return value

The logical palette index of the closest matching color, between 0 and 255.

Restrictions

Before calling **fg_findrgb()**, a logical palette must be defined and realized.

Replaced by

fg_maprgb()

fg_fixdiv()

Prototype

```
long fg_fixdiv (long n1, long n2);  
Function fg_fixdiv (ByVal n1 As Long, ByVal n2 As Long) As Long  
function fg_fixdiv (n1, n2 : longint) : longint;
```

Description

The **fg_fixdiv()** legacy function returns the fixed point quotient of two fixed point numbers.

Parameters

n1 is the fixed point dividend (numerator).

n2 is the fixed point divisor (denominator).

Return value

The fixed point quotient of $n1/n2$.

Restrictions

The integral portion of the divisor must not be zero.

Replaced by

Floating point 3D geometry system

fg_fixed()

Prototype

```
long fg_fixed (double n);  
Function fg_fixed (ByVal n As Double) As Long  
function fg_fixed (n : real) : longint;
```

Description

The **fg_fixed()** legacy function translates a floating point quantity to its fixed point equivalent.

Parameters

n is the floating point value to translate.

Return value

The fixed point equivalent of the specified floating point value.

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_fixmul()

Prototype

```
long fg_fixmul (long n1, long n2);  
Function fg_fixmul (ByVal n1 As Long, ByVal n2 As Long) As Long  
function fg_fixmul (n1, n2 : longint) : longint;
```

Description

The **fg_fixmul()** legacy function returns the fixed point product of two fixed point numbers.

Parameters

n1 and *n2* are the fixed point numbers to multiply.

Return value

The fixed point product of $n1 \cdot n2$.

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_fixtrig()

Prototype

```
void fg_fixtrig (int angle, long *cosine, long *sine);  
Sub fg_fixtrig (ByVal angle As Long, cosine As Long, sine As Long)  
procedure fg_fixtrig (angle: integer; var cosine, sine : longint);
```

Description

The **fg_fixtrig()** legacy function returns the fixed point cosine and fixed point sine of a given angle.

Parameters

angle is the desired angle, expressed in tenths of degrees counterclockwise from the positive x axis. For example, the value 900 represents 90 degrees.

cosine receives the fixed point cosine of the angle.

sine receives the fixed point sine of the angle.

Return value

none

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_flicdone()

Prototype

```
void fg_flicdone (void *context);  
Sub fg_flicdone (context() As Any)  
procedure fg_flicdone (var context);
```

Description

The **fg_flicdone()** function closes the flic file associated with the specified context descriptor.

Parameters

context is the name of a 16-byte buffer containing the flic file context descriptor.

Return value

none

Restrictions

none

See also

fg_flicopen()

Examples

AVImake, Image

fg_flichead()

Prototype

```
int fg_flichead (char *filename, void *header);  
Function fg_flichead (ByVal filename As String, header() As Any) As Long  
function fg_flichead (filename : string; var header) : integer;
```

Description

The **fg_flichead()** function reads an FLI or FLC file header into a 128-byte buffer. Refer to Appendix E of the *Fastgraph 6.0 User's Guide* for details about the FLI/FLC header.

Parameters

filename is the name of the FLI/FLC file, terminated by a zero byte.

header is the name of the buffer to receive the FLI/FLC file header. Its size must be at least 128 bytes.

Return value

- 0 = Success
- 1 = The specified file does not exist
- 2 = The specified file is not an FLI or FLC file

Restrictions

none

See also

fg_flicplay(), fg_flicsize(), fg_showflic()

Examples

AVImake, Image

fg_flicopen()

Prototype

```
int fg_flicopen (char *filename, void *context);
```

```
Function fg_flicopen (ByVal filename As String, context() As Any) As Long
```

```
function fg_flicopen (filename : string; var context) : integer;
```

Description

The **fg_flicopen()** function opens an FLI or FLC file (collectively called flic files) for subsequent processing by Fastgraph's other low-level flic file support functions. If successful, the file pointer will be positioned at the beginning of the first frame.

Parameters

filename is the name of the flic file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

context is the name of a 16-byte buffer that will receive the flic file context descriptor. The descriptor values will only be meaningful if the return value is zero.

Return value

0 = FLI/FLC file opened successfully

-1 = The specified file does not exist

-2 = The specified file is not an FLI or FLC file

Restrictions

none

See also

fg_flicdone(), fg_flicplay(), fg_flicskip(), fg_showflic()

Examples

AVImake, Image

fg_flicplay()

Prototype

```
int fg_flicplay (void *context, int frames, int flags);  
  
Function fg_flicplay (context() As Any, ByVal frames As Long, ByVal  
flags As Long) As Long  
  
function fg_flicplay (var context; frames, flags : integer) : integer;
```

Description

The **fg_flicplay()** function plays the next one or more individual frames in a flic file that was previously opened with **fg_flicopen()**.

Parameters

context is the name of a 16-byte buffer containing the flic file context descriptor.

frames is the number of frames to play from the flic file, starting from the current file position.

flags is a series of flags that controls how the frames are played. Refer to the description of **fg_showflic()** for the meanings of the flags.

Return value

The number of frames played. This value may be less than *frames* if the end-of-file is reached before playing the requested number of frames.

Restrictions

none

See also

fg_flicopen(), fg_flicskip(), fg_showflic()

Examples

AVImake, Image

fg_flicsize()

Prototype

```
void fg_flicsize (void *header, int *width, int *height);  
Sub fg_flicsize (header() As Any, width As Long, height As Long)  
procedure fg_flicsize (var header; var width, height : integer);
```

Description

The **fg_flicsize()** function returns the dimensions for the flic image associated with the specified flic file header.

Parameters

header is the name of the buffer containing the 128-byte FLI/FLC file header.

width receives the flic image width in pixels. If *header* does not contain a valid FLI/FLC file header, *width* will be set to -1.

height receives the flic image height in pixels. If *header* does not contain a valid FLI/FLC file header, *height* will be set to -1.

Return value

none

Restrictions

none

See also

fg_flichead(), fg_showflic()

Examples

AVImake, Image

fg_flicskip()

Prototype

```
int fg_flicskip (void *context, int frames);  
Function fg_flicskip (context() As Any, ByVal frames As Long) As Long  
function fg_flicskip (var context; frames : integer) : integer;
```

Description

The **fg_flicskip()** function advances one or more frames in a flic file that was previously opened with **fg_flicopen()**. If the last frame played by **fg_flicplay()** played the frame from the **fg_imagebuf()** buffer, the frame position will be adjusted in the **fg_imagebuf()** buffer. Otherwise, the flic file position itself will be adjusted.

Parameters

context is the name of a 16-byte buffer containing the flic file context descriptor.

frames is the number of frames to skip in the flic file, starting from the current file position. If *frames* is negative, the flic file position will be set to the first frame.

Return value

The number of frames skipped. This value may be less than *frames* if the end-of-file is reached before skipping the requested number of frames. If *frames* is negative, the return value will be zero.

Restrictions

none

See also

fg_flicopen(), fg_flicplay()

Examples

Image

fg_flipdcb()

Prototype

```
void fg_flipdcb (void *bitmap, int width, int height);  
Sub fg_flipdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_flipdcb (var bitmap; width, height : integer);
```

Description

The **fg_flipdcb()** function displays a reversed direct color bitmap, with clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Color 0 pixels will be considered transparent.

For high color virtual buffers, each pixel in the bitmap is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipdcb(), fg_drawdcb(), fg_flpimage(), fg_getdcb(), fg_invdcb(), fg_putdcb(), fg_revdcdb()

Examples

Dcb

fg_flipmask()

Prototype

```
void fg_flipmask (void *bitmap, int runs, int width);  
Sub fg_flipmask (bitmap() As Any, ByVal runs As Long, ByVal width As Long)  
procedure fg_flipmask (var bitmap; runs, width : integer);
```

Description

The **fg_flipmask()** legacy function displays a reversed masking map, with clipping. The masking map will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the masking map. The masking map is a series of alternating "protect" and "zero" pixel runs. The "protect" runs leave the corresponding virtual buffer pixels unchanged, while the "zero" runs set them to color zero. The length of each run must be between 0 and 255.

runs is the number of pixel runs in the masking map.

width is the masking map width in pixels.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_float()

Prototype

```
double fg_float (long n);  
Function fg_float (ByVal n As Long) As Double  
function fg_float (n : longint) : real;
```

Description

The **fg_float()** legacy function translates a fixed point quantity to its floating point equivalent.

Parameters

n is the fixed point value to translate.

Return value

The floating point equivalent of the specified fixed point value.

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_flood()

Prototype

```
void fg_flood (int ix, int iy);  
Sub fg_flood (ByVal ix As Long, ByVal iy As Long)  
procedure fg_flood (ix, iy : integer);
```

Description

The **fg_flood()** function fills an arbitrary closed region with pixels of the current color, with clipping. The region is defined by specifying a screen space point within its interior. If this point lies outside the clipping region, no fill is performed.

Parameters

ix is the screen space x coordinate of the interior point.

iy is the screen space y coordinate of the interior point.

Return value

none

Restrictions

none

See also

fg_floodw(), fg_paint()

fg_floodw()

Prototype

```
void fg_floodw (double x, double y);  
Sub fg_floodw (ByVal x As Double, ByVal y As Double)  
procedure fg_floodw (x, y : real);
```

Description

The **fg_floodw()** function fills an arbitrary closed region with pixels of the current color, with clipping. The region is defined by specifying a 2D world space point within its interior. If this point lies outside the clipping region, no fill is performed.

Parameters

x is the world space x coordinate of the interior point.
y is the world space y coordinate of the interior point.

Return value

none

Restrictions

none

See also

fg_flood(), fg_paintw()

fg_flpimage()

Prototype

```
void fg_flpimage (void *bitmap, int width, int height);  
Sub fg_flpimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_flpimage (var bitmap; width, height : integer);
```

Description

The **fg_flpimage()** function displays a reversed 256-color bitmap, with clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clpimage(), fg_drwimage(), fg_flipdcb(), fg_getimage(), fg_invert(), fg_putimage(),
fg_revimage(), fg_setclip()

Examples

Bitmap, Fishtank

fg_fontdc()

Prototype

```
void fg_fontdc (HDC hdc);  
Sub fg_fontdc (ByVal hdc As Long)  
procedure fg_fontdc (hdc : HDC);
```

Description

The **fg_fontdc()** function defines the text device context. By default, **fg_print()** directs strings to the window's client area, but **fg_fontdc()** can redirect strings to the active virtual buffer.

Parameters

hdc is a Windows handle to the device context for text output. To direct strings to the client area, set *hdc* to the same device context passed to **fg_setdc()**. To direct strings to the active virtual buffer, set *hdc* to the device context returned by **fg_getdc()**.

Return value

none

Restrictions

none

See also

fg_ddgetdc(), **fg_getdc()**, **fg_print()**, **fg_setdc()**

Examples

Strings2

fg_fontload()

Prototype

```
void fg_fontload (int font_id);  
Sub fg_fontload (ByVal font_id As Long)  
procedure fg_fontload (font_id : integer);
```

Description

The **fg_fontload()** function makes the requested Windows stock font the current font.

Parameters

font_id is a code identifying which font to load. It must reference one of the six stock fonts supplied with Windows, as shown here:

Value	Symbolic Equivalent
10	OEM_FIXED_FONT
11	ANSI_FIXED_FONT
12	ANSI_VAR_FONT
13	SYSTEM_FONT
14	DEVICE_DEFAULT_FONT
16	SYSTEM_FIXED_FONT

Return value

none

Restrictions

none

See also

fg_fontdc(), fg_logfont(), fg_print()

Examples

Fontdemo

fg_gammadb()

Prototype

```
void fg_gammadb (void *source, void *dest, double gamma, int size);  
Sub fg_gammadb (source() As Any, dest() As Any, ByVal gamma As Double,  
ByVal size As Long)  
procedure fg_gammadb (var source, dest; gamma : double; size :  
integer);
```

Description

The **fg_gammadb()** function applies a gamma correction transform to a direct color bitmap.

Parameters

source is the name of the array containing the direct color bitmap to be transformed.

dest is the name of the array that will receive the resulting transformed bitmap.

gamma is the gamma correction value. Values between 0.0 and 1.0 will lighten the image, while values greater than 1.0 will darken the image.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_gammargb(), fg_gammavb()

fg_gammargb()

Prototype

```
void fg_gammargb (void *values, double gamma, int count);  
Sub fg_gammargb (values() As Any, ByVal gamma As Double, ByVal count As Long)  
procedure fg_gammargb (var values; gamma : double; count : integer);
```

Description

The **fg_gammargb()** function applies a gamma correction transform to a series of RGB color triples.

Parameters

values is the name of the array containing the RGB color components, arranged as three-byte RGB triples. Each RGB color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least $3 \times \text{count}$ bytes.

gamma is the gamma correction value. Values between 0.0 and 1.0 will lighten the image, while values greater than 1.0 will darken the image.

count is the number of RGB color triples to transform.

Return value

none

Restrictions

none

See also

fg_gammadcb(), fg_gammavb()

fg_gammavb()

Prototype

```
void fg_gammavb (double gamma, int width, int height);  
Sub fg_gammavb (ByVal gamma As Double, ByVal width As Long, ByVal  
height As Long)  
procedure fg_gammavb (gamma : double; width, height : integer);
```

Description

The **fg_gammavb()** function applies a gamma correction transform to a rectangular region of the active virtual buffer. The region's lower left corner is at the current graphics position.

Parameters

gamma is the gamma correction value. Values between 0.0 and 1.0 will lighten the image, while values greater than 1.0 will darken the image.

width is the region's width in pixels.

height is the region's height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_gammadcb(), fg_gammargb()

Examples

ImgProc

fg_gdiflip()

Prototype

```
void fg_gdiflip (void);  
Sub fg_gdiflip ()  
procedure fg_gdiflip;
```

Description

The **fg_gdiflip()** function makes the GDI drawing surface the visible surface in a DirectX page flipping environment.

Parameters

none

Return value

none

Restrictions

This function is available only in Fastgraph's DirectX libraries.

See also

fg_ddflip(), fg_ddsetup()

fg_getblock()

Prototype

```
void fg_getblock (void *buffer, int minx, int maxx, int miny, int
maxy);

Sub fg_getblock (buffer() As Any, ByVal minx As Long, ByVal maxx As
Long, ByVal miny As Long, ByVal maxy As Long)

procedure fg_getblock (var buffer; minx, maxx, miny, maxy : integer);
```

Description

The **fg_getblock()** legacy function retrieves a block (for later display with the **fg_putblock()** function) from the specified position in the active virtual buffer. The block extremes are defined in screen space. Use **fg_imagesiz()** to determine the array size required to hold the block.

Parameters

buffer is the name of the array to receive the block.

minx is the screen space x coordinate of the block's left edge.

maxx is the x coordinate of the block's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the block's top edge.

maxy is the y coordinate of the block's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

The **fg_imagesiz()** function provides an easy way to determine the array size required to store the block. However, in true color virtual buffers that use a 32bpp memory architecture, **fg_imagesiz()** does not work because it returns bitmap sizes, which are always 24bpp when using a true color virtual buffer.

Replaced by

fg_getdcb(), fg_getimage()

fg_getclip()

Prototype

```
void fg_getclip (int *minx, int *maxx, int *miny, int *maxy);  
Sub fg_getclip (minx As Long, maxx As Long, miny As Long, maxy As Long)  
procedure fg_getclip (var minx, maxx, miny, maxy : integer);
```

Description

The **fg_getclip()** function returns the clipping region extremes in screen space. The clipping region is a rectangular area outside of which certain graphics are suppressed. By default, the clipping region is set to the virtual buffer extents.

Parameters

minx receives the x coordinate of the clipping region's left edge.

maxx receives the x coordinate of the clipping region's right edge.

miny receives the y coordinate of the clipping region's top edge.

maxy receives the y coordinate of the clipping region's bottom edge.

Return value

none

Restrictions

none

See also

fg_setclip(), fg_setclipw()

fg_getclock()

Prototype

```
long fg_getclock (void);  
Function fg_getclock () As Long  
function fg_getclock : longint;
```

Description

The **fg_getclock()** function returns the number of clock ticks since Windows started running.

Parameters

none

Return value

The number of clock ticks since Windows started running. There are approximately 18.2 clock ticks per second.

Restrictions

none

fg_getcolor()

Prototype

```
int fg_getcolor (void);  
Function fg_getcolor () As Long  
function fg_getcolor : integer;
```

Description

The **fg_getcolor()** function returns the current color, as defined by the most recent call **fg_setcolor()**.

Parameters

none

Return value

The current color value.

Restrictions

none

See also

fg_setcolor(), fg_unmaprgb()

Examples

FrameDD

fg_getdacs()

Prototype

```
void fg_getdacs (int start, int count, void *values);  
Sub fg_getdacs (ByVal start As Long, ByVal count As Long, values() As  
Any)  
procedure fg_getdacs (start, count : integer; var values);
```

Description

The **fg_getdacs()** function retrieves the red, green, and blue color components of a consecutive group of colors in the active logical palette or virtual palette. Retrieving many colors with **fg_getdacs()** is faster than doing so individually with **fg_getrgb()**.

Parameters

start is the starting color number, between 0 and 255.

count is the number of consecutive colors to retrieve, between 1 and 256. The sum of *start* and *count* cannot exceed 256.

values is the name of the array that will receive the color components. The first three bytes of this array receive the red, green, and blue components for color *start*, the next three bytes receive the components for color *start+1*, and so forth. Each color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least $3*count$ bytes.

Return value

none

Restrictions

Before calling **fg_getdacs()**, a logical palette must be defined and realized.

See also

fg_getrgb(), **fg_realize()**, **fg_setdacs()**

Examples

Fade

fg_getdc()

Prototype

```
HDC fg_getdc (void);  
Function fg_getdc () As Long  
function fg_getdc : HDC;
```

Description

The **fg_getdc()** function returns the handle to the DIB section device context associated with the active virtual buffer.

Parameters

none

Return value

A Windows handle to the DIB section device context.

Restrictions

Before using **fg_getdc()**, you must have already called **fg_vbinit()**.

See also

fg_vbinit()

Examples

GDIdemo, Strings2

fg_getdcb()

Prototype

```
void fg_getdcb (void *bitmap, int width, int height);  
Sub fg_getdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_getdcb (var bitmap; width, height : integer);
```

Description

The **fg_getdcb()** function retrieves a direct color bitmap. The graphics cursor defines the bitmap's lower left corner.

For high color virtual buffers, each pixel in the bitmap will be a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel will be a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array that will receive the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipdcb(), fg_drawdcb(), fg_flipdcb(), fg_getimage(), fg_getmap(), fg_invdcb(), fg_putdcb(), fg_revdcdb()

Examples

Blend, TMcube, TMcubeX

fg_getdepth()

Prototype

```
int fg_getdepth (void);  
Function fg_getdepth () As Long  
function fg_getdepth : integer;
```

Description

The **fg_getdepth()** function returns the active virtual buffer's color depth in bits per pixel.

Parameters

none

Return value

The color depth in bits per pixel.

Restrictions

none

See also

fg_colors(), fg_vbdepth()

fg_gethcbpp()

Prototype

```
int fg_gethcbpp (void);  
Function fg_gethcbpp () As Long  
function fg_gethcbpp : integer;
```

Description

The **fg_gethcbpp()** function returns the color depth for high color virtual buffers.

Parameters

none

Return value

The color depth in bits per pixel. For Fastgraph's native libraries and windowed DirectX programs, the color depth will always be 16. For full screen DirectX programs, the color depth will be 15 for the 5/5/5 high color pixel format, or 16 for the 5/6/5 high color format.

Restrictions

For full screen DirectX programs, the **fg_gethcbpp()** return value is meaningful only after calling **fg_vbinit()** and only if the **fg_ddsetup()** color depth parameter was 16.

See also

fg_colors(), fg_transdcb()

fg_gethpage()

Prototype

```
int fg_gethpage (void);  
Function fg_gethpage () As Long  
function fg_gethpage : integer;
```

Description

The **fg_gethpage()** function returns the background virtual buffer handle, as set in the most recent call to **fg_sethpage()**.

Parameters

none

Return value

The background virtual buffer handle.

Restrictions

none

See also

fg_sethpage(), fg_vbopen()

fg_getimage()

Prototype

```
void fg_getimage (void *bitmap, int width, int height);  
Sub fg_getimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_getimage (var bitmap; width, height : integer);
```

Description

The **fg_getimage()** function retrieves a 256-color bitmap. The graphics cursor defines the bitmap's lower left corner. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array that will receive the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clpimage(), fg_drwimage(), fg_flpimage(), fg_getdcb(), fg_getmap(), fg_putimage(), fg_revimage()

Examples

AVImake, Fishtank, PCXflip, TMcube, TMcubeX

fg_getindex()

Prototype

```
int fg_getindex (int index);  
Function fg_getindex (ByVal index As Long) As Long  
function fg_getindex (index : integer) : integer;
```

Description

The **fg_getindex()** legacy function returns the color value assigned to a virtual color index.

Parameters

index is the virtual color index to retrieve, between 0 and 255.

Return value

The color value assigned to the specified virtual index.

Restrictions

none

fg_getline()

Prototype

```
long fg_getline (int row);  
Function fg_getline (ByVal row As Long) As Long  
function fg_getline (row : integer) : longint;
```

Description

The **fg_getline()** function returns the address of the specified scan line in the active virtual buffer.

Parameters

row is the scan line number, between 0 and **fg_getmaxy()**.

Return value

The address of the first pixel in the requested scan line.

Restrictions

none

See also

fg_ddlock(), **fg_vbaddr()**

fg_getlines()

Prototype

```
int fg_getlines (void);  
Function fg_getlines () As Long  
function fg_getlines : integer;
```

Description

The **fg_getlines()** legacy function returns the number of text rows in the active virtual buffer when using the current font.

Parameters

none

Return value

The number of text rows for the current virtual buffer and font.

Restrictions

none

Replaced by

Screen space

fg_getmap()

Prototype

```
void fg_getmap (void *bitmap, int width, int height);  
Sub fg_getmap (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_getmap (var bitmap; width, height : integer);
```

Description

The **fg_getmap()** function retrieves a monochrome bitmap. The graphics cursor defines the bitmap's lower left corner. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about monochrome bitmaps.

Parameters

bitmap is the name of the array that will receive the bitmap. Each byte of *bitmap* represents eight pixels. Pixels of the current color set the corresponding bits in *bitmap*. Pixels of other colors make the corresponding *bitmap* bits zero.

width is the bitmap width in bytes.

height is the bitmap height in bytes.

Return value

none

Restrictions

none

See also

fg_clipmap(), fg_drawmap(), fg_getdcb(), fg_getimage()

fg_getmaxx()

Prototype

```
int fg_getmaxx (void);  
Function fg_getmaxx () As Long  
function fg_getmaxx : integer;
```

Description

The **fg_getmaxx()** function returns the maximum x coordinate in screen space. The maximum x coordinate is one less than the active virtual buffer's horizontal resolution.

Parameters

none

Return value

The maximum x coordinate.

Restrictions

none

See also

fg_getmaxy()

Examples

AVImake, Colors, First, FirstDD, Fishtank, GDIdemo, Image, ImgProc, Rainbow

fg_getmaxy()

Prototype

```
int fg_getmaxy (void);  
Function fg_getmaxy () As Long  
function fg_getmaxy : integer;
```

Description

The **fg_getmaxy()** function returns the maximum y coordinate in screen space. The maximum y coordinate is one less than the active virtual buffer's vertical resolution.

Parameters

none

Return value

The maximum y coordinate.

Restrictions

none

See also

fg_getmaxx()

Examples

AVImake, Colors, First, FirstDD, Fishtank, GDIdemo, Image, ImgProc, Rainbow

fg_getpage()

Prototype

```
int fg_getpage (void);  
Function fg_getpage () As Long  
function fg_getpage : integer;
```

Description

The **fg_getpage()** legacy function returns the foreground virtual buffer handle, as set in the most recent call to **fg_setpage()**.

Parameters

none

Return value

The foreground virtual buffer handle.

Restrictions

none

Replaced by

fg_vbcopy()

fg_getpixel()

Prototype

```
int fg_getpixel (int ix, int iy);  
Function fg_getpixel (ByVal ix As Long, ByVal iy As Long) As Long  
function fg_getpixel (ix, iy : integer) : integer;
```

Description

The **fg_getpixel()** function returns the color value of a specified pixel.

Parameters

ix is the pixel's screen space x coordinate.

iy is the pixel's screen space y coordinate.

Return value

The color value of the pixel. For 256-color virtual buffers, it will be between 0 and 255. For high color virtual buffers, it will be a 16-bit encoded RGB value. For true color virtual buffers, it will be a 32-bit xRGB value.

Restrictions

none

See also

fg_point(), fg_pointw(), fg_putpixel()

fg_getrgb()

Prototype

```
void fg_getrgb (int number, int *red, int *green, int *blue);  
Sub fg_getrgb (ByVal number As Long, red As Long, green As Long, blue  
As Long)  
procedure fg_getrgb (number : integer; var red, green, blue : integer);
```

Description

The **fg_getrgb()** function returns the red, green, and blue color components for a specified color in the active logical palette or virtual palette.

Parameters

number is the color number, between 0 and 255.

red, *green*, and *blue* respectively receive the red, green, and blue components of the specified color. Each color component is a value between 0 and 255; increasing values produce more intense colors.

Return value

none

Restrictions

Before calling **fg_getrgb()**, a logical palette must be defined and realized.

See also

fg_getdacs(), fg_realize(), fg_setrgb()

Examples

Cube

fg_getview()

Prototype

```
void fg_getview (int *view_minx, int *view_maxx, int *view_miny, int  
*view_maxy, int *minx, int *maxx, int *miny, int *maxy);
```

```
Sub fg_getview (view_minx As Long, view_maxx As Long, view_miny As  
Long, view_maxy As Long, minx As Long, maxx As Long, miny As Long, maxy  
As Long)
```

```
procedure fg_getview (var view_minx, view_maxx, view_miny, view_maxy,  
minx, maxx, miny, maxy : integer);
```

Description

The **fg_getview()** function returns the 2D viewport extremes in viewport units and screen space units.

Parameters

view_minx receives the viewport's left edge in viewport units.

view_maxx receives the viewport's right edge in viewport units.

view_miny receives the viewport's top edge in viewport units.

view_maxy receives the viewport's bottom edge in viewport units.

minx receives the viewport's left edge in screen space units.

maxx receives the viewport's right edge in screen space units.

miny receives the viewport's top edge in screen space units.

maxy receives the viewport's bottom edge in screen space units.

Return value

none

Restrictions

none

See also

[fg_setview\(\)](#)

fg_getworld()

Prototype

```
void fg_getworld (double *xmin, double *xmax, double *ymin, double
*ymin);

Sub fg_getworld (xmin As Double, xmax As Double, ymin As Double, ymax
As Double)

procedure fg_getworld (var xmin, xmax, ymin, ymax : real);
```

Description

The **fg_getworld()** function returns the current 2D world space limits, as defined in the most recent call to **fg_setworld()**.

Parameters

xmin receives the world space coordinate of the virtual buffer's left edge.

xmax receives the world space coordinate of the virtual buffer's right edge.

ymin receives the world space coordinate of the virtual buffer's bottom edge.

ymax receives the world space coordinate of the virtual buffer's top edge.

Return value

none

Restrictions

none

See also

fg_setworld()

fg_getxbox()

Prototype

```
int fg_getxbox (void);  
Function fg_getxbox () As Long  
function fg_getxbox : integer;
```

Description

The **fg_getxbox()** function returns the width in pixels of the left and right edges of rectangles drawn with the **fg_box()** family of functions. By default, the width is one pixel, but this can be changed by calling **fg_boxdepth()**.

Parameters

none

Return value

The width in pixels of the left and right sides (that is, the vertical edges) of rectangles drawn with the **fg_box()** family of functions.

Restrictions

none

See also

fg_boxdepth(), fg_getybox()

fg_getxjust()

Prototype

```
int fg_getxjust (void);  
Function fg_getxjust () As Long  
function fg_getxjust : integer;
```

Description

The **fg_getxjust()** function returns the horizontal justification setting used by **fg_print()**. The **fg_vbinit()** function sets the default justification to -1, and its value may be changed with **fg_justify()**.

Parameters

none

Return value

-1 = Strings are left justified relative to the current graphics x position
0 = Strings are centered about the current graphics x position
1 = Strings are right justified relative to the current graphics x position

Restrictions

none

See also

fg_getyjust(), fg_justify()

fg_getxpos()

Prototype

```
int fg_getxpos (void);  
Function fg_getxpos () As Long  
function fg_getxpos : integer;
```

Description

The **fg_getxpos()** function returns the screen space x coordinate of the graphics cursor position.

Parameters

none

Return value

The x coordinate of graphics cursor position.

Restrictions

none

See also

fg_getypos()

fg_getybox()

Prototype

```
int fg_getybox (void);  
Function fg_getybox () As Long  
function fg_getybox : integer;
```

Description

The **fg_getybox()** function returns the width in pixels of the top and bottom edges of rectangles drawn with the **fg_box()** family of functions. By default, the width is one pixel, but this can be changed by calling **fg_boxdepth()**.

Parameters

none

Return value

The width in pixels of the top and bottom sides (that is, the horizontal edges) of rectangles drawn with the **fg_box()** family of functions.

Restrictions

none

See also

fg_boxdepth(), fg_getxbox()

fg_getyjust()

Prototype

```
int fg_getyjust (void);  
Function fg_getyjust () As Long  
function fg_getyjust : integer;
```

Description

The **fg_getyjust()** function returns the vertical justification setting used by **fg_print()**. The **fg_vbinit()** function sets the default justification to -1, and its value may be changed with **fg_justify()**.

Parameters

none

Return value

- 1 = Strings will have their bottom edge at the current graphics y position
- 0 = Strings are centered about the current graphics y position
- 1 = Strings will have their top edge at the current graphics y position

Restrictions

none

See also

fg_getxjust(), fg_justify()

fg_getypos()

Prototype

```
int fg_getypos (void);  
Function fg_getypos () As Long  
function fg_getypos : integer;
```

Description

The **fg_getypos()** function returns the screen space y coordinate of the graphics cursor position.

Parameters

none

Return value

The y coordinate of graphics cursor position.

Restrictions

none

See also

fg_getxpos()

fg_gouraud()

Prototype

```
void fg_gouraud (int *vertex_array, char *rgb_array, int n);
Sub fg_gouraud (vertex_array() As Long, rgb_array() As Byte, ByVal n As Long)
procedure fg_gouraud (var vertex_array : integer; var rgb_array : byte;
n : integer);
```

Description

The **fg_gouraud()** function draws a projected Gouraud-shaded convex polygon in screen space, with 2D clipping and automatic backface removal. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth. The vertices must be stored in clockwise order, meaning you would travel clockwise along the polygon edge to go from one vertex to the next when facing the front of the polygon.

rgb_array is the name of the array containing the RGB color components for each (x,y) coordinate pair in *vertex_array*. The first three *rgb_array* elements represent the RGB color values at the first vertex in *vertex_array*, the next three *rgb_array* elements are for the second vertex, and so forth. Each RGB color component is a value between 0 and 255.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

If you attempt to fill a non-convex polygon with **fg_gouraud()**, or if the vertices are not stored in clockwise order, only a portion of the polygon will be filled.

See also

fg_3Dshade(), fg_3Dshadeobject(), fg_gouraudz(), fg_inside(), fg_polyoff()

fg_gouraudz()

Prototype

```
void fg_gouraudz (int *vertex_array, char *rgb_array, double
*xyz_array, int n);

Sub fg_gouraudz (vertex_array() As Long, rgb_array() As Byte,
xyz_array() As Double, ByVal n As Long)

procedure fg_gouraudz (var vertex_array : integer; var rgb_array :
byte; var xyz_array : double; n : integer);
```

Description

The **fg_gouraudz()** function draws a projected z-buffered Gouraud-shaded convex polygon in screen space, with 2D clipping. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

rgb_array is the name of the array containing the RGB color components for each (x,y) coordinate pair in *vertex_array*. The first three *rgb_array* elements represent the RGB color values at the first vertex in *vertex_array*, the next three *rgb_array* elements are for the second vertex, and so forth. Each RGB color component is a value between 0 and 255.

xyz_array is the name of the array containing the 3D (x,y,z) coordinates for each (x,y) coordinate pair in *vertex_array*. The first three *xyz_array* elements represent the (x,y,z) values at the first vertex in *vertex_array*, the next three *xyz_array* elements are for the second vertex, and so forth. Only the z coordinates are meaningful in this function.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

If you attempt to fill a non-convex polygon with **fg_gouraudz()**, only a portion of the polygon will be filled.

See also

fg_3Dshade(), fg_3Dshadeobject(), fg_gouraud(), fg_inside(), fg_polyoff(), fg_zbopen()

fg_graydcb()

Prototype

```
void fg_graydcb (void *source, void *dest, int size);  
Sub fg_graydcb (source() As Any, dest() As Any, ByVal size As Long)  
procedure fg_graydcb (var source, dest; size : integer);
```

Description

The **fg_graydcb()** function applies a grayscale transform to a direct color bitmap.

Parameters

source is the name of the array containing the direct color bitmap to be transformed.

dest is the name of the array that will receive the resulting transformed bitmap.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_grayrgb(), fg_grayvb()

fg_grayrgb()

Prototype

```
void fg_grayrgb (void *values, int count);  
Sub fg_grayrgb (values() As Any, ByVal count As Long)  
procedure fg_grayrgb (var values; count : integer);
```

Description

The **fg_grayrgb()** function applies a grayscale transform to a series of RGB color triples.

Parameters

values is the name of the array containing the RGB color components, arranged as three-byte RGB triples. Each RGB color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least $3*count$ bytes.

count is the number of RGB color triples to transform.

Return value

none

Restrictions

none

See also

fg_graydcb(), fg_grayvb()

fg_grayvb()

Prototype

```
void fg_grayvb (int width, int height);  
Sub fg_grayvb (ByVal width As Long, ByVal height As Long)  
procedure fg_grayvb (width, height : integer);
```

Description

The **fg_grayvb()** function applies a grayscale transform to a rectangular region of the active virtual buffer. The region's lower left corner is at the current graphics position.

Parameters

width is the region's width in pixels.

height is the region's height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_graydcb(), fg_grayrgb()

Examples

ImgProc

fg_imagebuf()

Prototype

```
void fg_imagebuf (void *buffer, unsigned int size);  
Sub fg_imagebuf (buffer() As Any, ByVal size As Long)  
procedure fg_imagebuf (var buffer; size : integer);
```

Description

The **fg_imagebuf()** function specifies the size and address of the buffer used internally when creating or displaying image files. The default internal buffer size is 4,096 bytes. Image display or creation is typically faster when a larger buffer is used.

Parameters

buffer is the address of the internal buffer.

size is the buffer size in bytes. If *size* is zero, Fastgraph will use its own internal buffers when creating or displaying image files.

Return value

none

Restrictions

none

See also

fg_flicplay(), fg_makebmp(), fg_makepcx(), fg_showbmp(), fg_showflic(), fg_showjpeg(), fg_showpcx()

fg_imagesiz()

Prototype

```
long fg_imagesiz (int width, int height);  
  
Function fg_imagesiz (ByVal width As Long, ByVal height As Long) As Long  
  
function fg_imagesiz (width, height : integer) : longint;
```

Description

The **fg_imagesiz()** function determines the number of bytes required to store a bitmap of the specified dimensions for the active virtual buffer's color depth.

Parameters

width specifies the bitmap width in pixels.
height specifies the bitmap height in pixels.

Return value

The number of bytes required to store a bitmap of the specified size, based on the active virtual buffer's color depth.

Restrictions

none

See also

fg_clipdcb(), fg_clpimage(), fg_drawdcb(), fg_drwimage(), fg_flipdcb(), fg_flpimage(), fg_getdcb(), fg_getimage(), fg_putdcb(), fg_putimage(), fg_revdcdb(), fg_revimage()

Examples

AVImake, PCXflip

fg_initw()

Prototype

```
void fg_initw (void);  
Sub fg_initw ()  
procedure fg_initw;
```

Description

The **fg_initw()** function initializes Fastgraph's 2D world space internal parameters. This function must be called once, before any other function that uses 2D world space coordinates, usually in the WM_CREATE message handler.

Parameters

none

Return value

none

Restrictions

none

Examples

SWchars

fg_inside()

Prototype

```
int fg_inside (int *vertex_array, int n, int ix, int iy);  
  
Function fg_inside (vertex_array() As Long, ByVal n As Long, ByVal ix  
As Long, ByVal iy As Long) As Long  
  
function fg_inside (var vertex_array : integer; n, ix, iy : integer) :  
integer;
```

Description

The **fg_inside()** function determines if the specified point is inside a convex polygon. The **fg_polyoff()** offsets are applied to the polygon vertices but not to the test point.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

n is the number of vertices in the polygon. Normally, it is one-half the size of *vertex_array*.

ix is the screen space x coordinate of the test point.

iy is the screen space y coordinate of the test point.

Return value

0 = The test point is outside the polygon

1 = The test point is inside the polygon

Restrictions

If *vertex_array* does not define a convex polygon, the return value is undefined.

See also

fg_polyedge(), fg_polyfill(), fg_polyline(), fg_polyoff()

fg_invdcb()

Prototype

```
void fg_invdcb (void *bitmap, int width, int height);  
Sub fg_invdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_invdcb (var bitmap; width, height : integer);
```

Description

The **fg_invdcb()** function inverts the orientation of a direct color bitmap. Fastgraph's bitmapped image display functions expect the bitmap to be stored starting with the bottom row and proceeding toward the top. The **fg_invdcb()** function will reverse the row order of such bitmaps, so that a "top to bottom" image becomes a "bottom to top" image, or vice versa.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clipdcb(), fg_drawdcb(), fg_flipdcb(), fg_getdcb(), fg_invert(), fg_putdcb(), fg_revdcdb()

Examples

TMcube, TMcubeX

fg_invert()

Prototype

```
void fg_invert (void *bitmap, int width, int height);  
Sub fg_invert (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_invert (var bitmap; width, height : integer);
```

Description

The **fg_invert()** function inverts the orientation of a monochrome, 256-color, or direct color bitmap. Fastgraph's bitmapped image display functions expect the bitmap to be stored starting with the bottom row and proceeding toward the top. The **fg_invert()** function will reverse the row order of such bitmaps, so that a "top to bottom" image becomes a "bottom to top" image, or vice versa.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in bytes.

height is the bitmap height in bytes.

Return value

none

Restrictions

none

See also

fg_clipdcb(), fg_clipmap(), fg_clpimage(), fg_drawdcb(), fg_drawmap(), fg_drwimage(),
fg_flipdcb(), fg_flpimage(), fg_getdcb(), fg_getimage(), fg_getmap(), fg_invdcb(), fg_putimage(),
fg_revdcb(), fg_revimage()

Examples

TMcube, TMcubeX

fg_jpegbuf()

Prototype

```
void fg_jpegbuf (void *buffer, unsigned int size);  
Sub fg_jpegbuf (buffer() As Any, ByVal size As Long)  
procedure fg_jpegbuf (buffer : pointer; size : integer);
```

Description

The **fg_jpegbuf()** legacy function is obsolete. A "do nothing" version of **fg_jpegbuf()** is provided for source code compatibility with earlier versions of Fastgraph.

Parameters

buffer is the address of the JPEG buffer.

size is the buffer size in bytes.

Return value

none

Restrictions

none

fg_jpeghead()

Prototype

```
int fg_jpeghead (char *filename, void *header);
```

```
Function fg_jpeghead (ByVal filename As String, header() As Any) As Long
```

```
function fg_jpeghead (filename : string; var header) : integer;
```

Description

The **fg_jpeghead()** function reads a JPEG file header into a 10-byte buffer. Strictly speaking, JPEG files do not have formal headers, but **fg_jpeghead()** returns relevant information from the file's start of frame segment. We call it a header for consistency with other image file formats. Refer to Appendix E of the *Fastgraph 6.0 User's Guide* for details about the JPEG header.

Parameters

filename is the name of the JPEG file. It may include a path specification and must be terminated by a zero byte.

header is the name of the 10-byte buffer to receive the JPEG file header.

Return value

0 = Success

-1 = The specified file does not exist

-2 = The specified file is not a baseline JPEG file

Restrictions

none

See also

fg_jpegsize(), fg_showjpeg()

Examples

Image, ImgProc

fg_jpegmem()

Prototype

```
int fg_jpegmem (void *header);  
Function fg_jpegmem (header() As Any) As Long  
function fg_jpegmem (var header) : integer;
```

Description

The **fg_jpegmem()** legacy function is obsolete. A "do nothing" version of **fg_jpegmem()** is provided for source code compatibility with earlier versions of Fastgraph.

Parameters

header is the name of the buffer containing the 10-byte JPEG file header.

Return value

The "do nothing" version of **fg_jpegmem()** always returns a nominal buffer size of 256 bytes. If *header* does not contain a valid JPEG file header, **fg_jpegmem()** returns zero.

Restrictions

none

fg_jpegsize()

Prototype

```
void fg_jpegsize (void *header, int *width, int *height);  
Sub fg_jpegsize (header() As Any, width As Long, height As Long)  
procedure fg_jpegsize (var header; var width, height : integer);
```

Description

The **fg_jpegsize()** function returns the image dimensions for the JPEG image associated with the specified JPEG file header.

Parameters

header is the name of the buffer containing the 10-byte JPEG file header.

width receives the JPEG image width in pixels. If *header* does not contain a valid JPEG file header, *width* will be set to -1.

height receives the JPEG image height in pixels. If *header* does not contain a valid JPEG file header, *height* will be set to -1.

Return value

none

Restrictions

none

See also

fg_jpeghead(), fg_showjpeg()

Examples

Image, ImgProc

fg_justify()

Prototype

```
void fg_justify (int xjust, int yjust);  
Sub fg_justify (ByVal xjust As Long, ByVal yjust As Long)  
procedure fg_justify (xjust, yjust : integer);
```

Description

The **fg_justify()** function defines the horizontal and vertical justification settings for strings displayed with **fg_print()**.

Parameters

xjust defines the horizontal justification. If *xjust* is -1, strings will be displayed left justified relative to the current graphics x position. If *xjust* is 0, strings will be centered about the x position. If *xjust* is 1, strings will be right justified.

yjust defines the vertical justification. If *yjust* is -1, the bottom of the characters will be the current graphics y position. If *yjust* is 0, strings will be centered about the y position. If *yjust* is 1, the top of the characters will be at the y position.

Return value

none

Restrictions

The values of *xjust* and *yjust* must be -1, 0, or 1.

See also

fg_getxjust(), fg_getyjust(), fg_print()

Examples

Strings1, Strings2

fg_kbtest()

Prototype

```
int fg_kbtest (int scan_code);  
Function fg_kbtest (ByVal scan_code As Long) As Long  
function fg_kbtest (scan_code : integer) : integer;
```

Description

The **fg_kbtest()** function determines if the key having the specified scan code is now pressed or released.

Parameters

scan_code is the scan code of the key to check, or zero. If *scan_code* is zero, **fg_kbtest()** reports whether any key is pressed. Refer to Chapter 11 of the *Fastgraph 6.0 User's Guide* for a list of scan codes.

Return value

0 = The key is released
1 = The key is pressed

Restrictions

none

Examples

Cube, KBdemo, TMcube, TMcubeX, Tunnel

fg_loadpcx()

Prototype

```
int fg_loadpcx (char *filename, int flags);
```

```
Function fg_loadpcx (ByVal filename As String, ByVal flags As Long) As Long
```

```
function fg_loadpcx (filename : string; flags : integer) : integer;
```

Description

The **fg_loadpcx()** legacy function displays a PCX file. It is equivalent to the **fg_showpcx()** function.

For 256-color virtual buffers, 256-color PCX files are reduced to the 236 non-system colors if color reduction is enabled. 16-color and monochrome PCX files are always remapped to colors 10 to 25 to avoid conflicts with the system colors.

Parameters

filename is the name of the PCX file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

flags is a series of flags that controls how the image is displayed. Refer to the description of **fg_showpcx()** for the meanings of the flags.

Return value

0 = Success

1 = The specified file does not exist

2 = The specified file is not a PCX file

3 = The PCX file cannot be loaded into the active virtual buffer

4 = Error allocating memory

Restrictions

A logical palette must be defined and realized in order to use the palette values stored in the PCX file.

24-bit PCX files can only be loaded into direct color virtual buffers.

Replaced by

fg_showpcx()

fg_locate()

Prototype

```
void fg_locate (int row, int column);  
Sub fg_locate (ByVal row As Long, ByVal column As Long)  
procedure fg_locate (row, column : integer);
```

Description

The **fg_locate()** legacy function changes the text cursor position for strings displayed with **fg_text()**. The **fg_vbinit()** function sets the text cursor position to (0,0).

Parameters

row is the text cursor's destination row number, between 0 and one less than the number of character rows available.

column is text cursor's destination column number, between 0 and one less than the number of character columns available.

Return value

none

Restrictions

When using a proportional font, **fg_locate()** assumes the width of a character column is equal to font's average character width.

Replaced by

Screen space

fg_logfont()

Prototype

```
void fg_logfont (HFONT hfont);  
Sub fg_logfont (ByVal hfont As Long)  
procedure fg_logfont (hfont : HFONT);
```

Description

The **fg_logfont()** function makes the requested logical font the current font.

Parameters

hfont is a Windows handle to the logical font, as returned by the Windows API functions **CreateFont()** or **CreateFontIndirect()**.

Return value

none

Restrictions

none

See also

fg_fontdc(), fg_fontload(), fg_print()

Examples

Fontdemo

fg_logpal()

Prototype

```
HPALETTE fg_logpal (int start, int count, void *values);  
  
Function fg_logpal (ByVal start As Long, ByVal count As Long, values()  
As Any) As Long  
  
function fg_logpal (start, count : integer; var values) : HPALETTE;
```

Description

The **fg_logpal()** function creates a 256-color logical palette containing the specified colors. The logical palette will be set up with the system colors (colors 0-9 and 246-255) set to their default values, unless explicitly changed.

Parameters

start is the starting color number to define in the logical palette, between 0 and 255.

count is the number of colors to define, between 0 and 256. The sum of *start* and *count* cannot exceed 256. If *count* is zero, the logical palette will contain the Windows system colors as its first and last ten entries, with the remaining 236 colors undefined.

values is the name of the array containing the color components. The first three bytes of this array must contain the red, green, and blue components for color *start*, the next three bytes contain the components for color *start*+1, and so forth. Each RGB color component is a value between 0 and 255. The size of the *values* array must be at least 3**count* bytes.

Return value

A Windows handle to the new logical palette. If an error occurs in creating the logical palette, **fg_logpal()** returns zero.

Restrictions

none

See also

fg_defpal(), fg_realize(), fg_setdacs(), fg_setrgb()

Examples

Rainbow

fg_makebmp()

Prototype

```
int fg_makebmp (int minx, int maxx, int miny, int maxy, int depth, char
*filename);
```

Function fg_makebmp (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long, ByVal depth As Long, ByVal filename As String) As Long

```
function fg_makebmp (minx, maxx, miny, maxy, depth : integer; filename
: string) : integer;
```

Description

The **fg_makebmp()** function creates a BMP file from the specified rectangular region of the active virtual buffer. The region's extremes are expressed in screen space units.

For 16-color BMP files, **fg_makebmp()** assumes the color values have been mapped to the non-system colors. It thus subtracts 10 from each pixel value when creating the BMP file. For monochrome (2-color) BMP files, **fg_makebmp()** treats the low-order bit of each pixel as the actual color.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to *miny*.

depth is the new BMP file's color depth in bits per pixel. Valid values are 1 (monochrome), 4 (16-color), 8 (256-color), and 24 (direct color RGB).

filename is the name of the BMP file to create. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte). If an identically named file already exists, it is overwritten.

Return value

0 = Success

1 = The BMP file was not created

2 = Error allocating memory

Restrictions

Monochrome, 16-color, and 256-color BMP files can only be created with a 256-color virtual buffer is active.

24-bit BMP files can only be created when a direct color virtual buffer is active.

See also

fg_imagebuf(), fg_makepcx(), fg_showbmp()

Examples

Image, ImgProc

fg_makepcx()

Prototype

```
int fg_makepcx (int minx, int maxx, int miny, int maxy, char
*filename);
```

```
Function fg_makepcx (ByVal minx As Long, ByVal maxx As Long, ByVal miny
As Long, ByVal maxy As Long, ByVal filename As String) As Long
```

```
function fg_makepcx (minx, maxx, miny, maxy : integer; filename :
string) : integer;
```

Description

The **fg_makepcx()** function creates a PCX file from the specified rectangular region of the active virtual buffer. The region's extremes are expressed in screen space units. For 256-color virtual buffers, **fg_makepcx()** creates 256-color PCX files; for direct color virtual buffers, it creates 24-bit PCX files.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to *miny*.

filename is the name of the PCX file to create. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte). If an identically named file already exists, it is overwritten.

Return value

0 = Success

1 = The PCX file was not created

2 = Error allocating memory

Restrictions

none

See also

fg_imagebuf(), fg_makebmp(), fg_showpcx()

Examples

Image, ImgProc

fg_makeppr()

Prototype

```
int fg_makeppr (int minx, int maxx, int miny, int maxy, char  
*filename);
```

```
Function fg_makeppr (ByVal minx As Long, ByVal maxx As Long, ByVal miny  
As Long, ByVal maxy As Long, ByVal filename As String) As Long
```

```
function fg_makeppr (minx, maxx, miny, maxy : integer; filename :  
string) : integer;
```

Description

The **fg_makeppr()** legacy function creates a packed pixel run (PPR) file from the specified rectangular region of the active virtual buffer. The region's extremes are expressed in screen space units.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to *miny*.

filename is the name of the PPR file to create. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte). If an identically named file already exists, it is overwritten.

Return value

0 = Success

1 = The PPR file was not created

Restrictions

none

Replaced by

BMP and PCX creation functions

fg_makespr()

Prototype

```
int fg_makespr (int minx, int maxx, int miny, int maxy, char
*filename);
```

```
Function fg_makespr (ByVal minx As Long, ByVal maxx As Long, ByVal miny
As Long, ByVal maxy As Long, ByVal filename As String) As Long
```

```
function fg_makespr (minx, maxx, miny, maxy : integer; filename :
string) : integer;
```

Description

The **fg_makespr()** legacy function creates a standard pixel run (SPR) file from the specified rectangular region of the active virtual buffer.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to *miny*.

filename is the name of the SPR file to create. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte). If an identically named file already exists, it is overwritten.

Return value

0 = Success

1 = The SPR file was not created

Restrictions

none

Replaced by

BMP and PCX creation functions

fg_mapdacs()

Prototype

```
void fg_mapdacs (void *source, void *dest, int count);  
Sub fg_mapdacs (source() As Any, dest() As Any, ByVal count As Long)  
procedure fg_mapdacs (var source, dest; count : integer);
```

Description

The **fg_mapdacs()** function translates RGB color components from the “0 to 63” range used in Fastgraph for DOS to the “0 to 255” range used in Fastgraph for Windows.

Parameters

source is the name of the array containing the “0 to 63” color components. The first three bytes of this array must contain the red, green, and blue components for the first color, the next three bytes contain the components for the second color, and so forth. The size of the *source* array must be at least $3*count$ bytes.

dest is the name of the array to receive the “0 to 255” color components. On return, the first three bytes of this array will contain the red, green, and blue components for the first color, the next three bytes will contain the components for the second color, and so forth. The size of the *dest* array must be at least $3*count$ bytes.

count is the number of sets of RGB color values to translate.

Return value

none

Restrictions

none

See also

fg_setdacs(), fg_setrgb()

fg_maprgb()

Prototype

```
int fg_maprgb (int red, int green, int blue);  
  
Function fg_maprgb (ByVal red As Long, ByVal green As Long, ByVal blue  
As Long) As Long  
  
function fg_maprgb (red, green, blue : integer) : integer;
```

Description

When used with a direct color virtual buffer, **fg_maprgb()** maps 8-bit red, green, and blue color components into a suitable 16-bit or 24-bit color value, depending on the color depth of the active virtual buffer. When used with a 256-color virtual buffer, **fg_maprgb()** returns the index of the closest matching color in the logical palette.

Parameters

red, *green*, and *blue* respectively specify the color's red, green, and blue components. These values must each be between 0 and 255; increasing values produce more intense colors.

Return value

The color value for the specified RGB components.

Restrictions

none

See also

fg_setcolor(), fg_setrgb(), fg_unmaprgb()

Examples

Colors, Cube, Dcb, FirstDD

fg_measure()

Prototype

```
unsigned int fg_measure (void);  
Function fg_measure () As Long  
function fg_measure : integer;
```

Description

The **fg_measure()** function returns the approximate number of delay units per clock tick. This quantity is proportional to the system's processor speed. Delay units are used by **fg_stall()**.

Parameters

none

Return value

The approximate number of delay units per clock tick.

Restrictions

none

See also

fg_stall()

fg_memavail()

Prototype

```
long fg_memavail (void);  
Function fg_memavail () As Long  
function fg_memavail : longint;
```

Description

The **fg_memavail()** legacy function determines the amount of free global memory available to Windows.

Parameters

none

Return value

The amount of free global memory (in bytes) available to Windows.

Restrictions

none

fg_modeset()

Prototype

```
int fg_modeset (int width, int height, int depth, int style);  
  
Function fg_modeset (ByVal width As Long, ByVal height As Long, ByVal  
depth As Long, ByVal style As Long) As Long  
  
function fg_modeset (width, height, depth, style : integer) :integer;
```

Description

The **fg_modeset()** function defines the desktop resolution and color depth when using Fastgraph's native libraries. We recommend that programs that change the display mode with **fg_modeset()** do so *before* setting up the logical palette, as some Windows display drivers redefine the palette values when changing display modes.

Parameters

width is the new horizontal resolution in pixels.

height is the new vertical resolution in pixels.

depth is the new color depth in bits per pixel. It must be 0, 8, 16, 24, or 32. If zero, **fg_modeset()** ignores the other parameters and restores the default display resolution and color depth as defined in the registry.

style controls the taskbar visibility. If *style* is zero, the display will include a taskbar; if *style* is any other value, the taskbar will be invisible.

Return value

<0 = The mode change failed

0 = The mode change was successful

1 = Windows must be restarted before the new mode takes effect

Restrictions

For some older display drivers, if the specified color depth does not match the display's current color depth, Windows must be restarted before the new mode takes effect.

See also

fg_colors(), fg_ddsetup(), fg_modetest()

Examples

Display, TMcubeX

fg_modetest()

Prototype

```
int fg_modetest (int width, int height, int depth);
```

```
Function fg_modetest (ByVal width As Long, ByVal height As Long, ByVal  
depth As Long) As Long
```

```
function fg_modetest (width, height, depth : integer) : integer;
```

Description

The **fg_modetest()** function checks if the requested display resolution and color depth are supported when using Fastgraph's native libraries.

Parameters

width is the requested horizontal resolution in pixels.

height is the requested vertical resolution in pixels.

depth is requested new color depth in bits per pixel. It must be 8, 16, 24, or 32.

Return value

<0 = The requested mode is not available

0 = The requested mode is available

1 = The requested mode is available, but Windows must first be restarted

Restrictions

none

See also

fg_colors(), fg_modeset()

Examples

Display

fg_mousecur()

Prototype

```
void fg_mousecur (HCURSOR hcursor);  
Sub fg_mousecur (ByVal hcursor As Long)  
procedure fg_mousecur (HCURSOR hcursor);
```

Description

The **fg_mousecur()** function activates a mouse cursor previously created with **fg_mouseptr()**. It is usually called in the WM_SETCURSOR message handler when the mouse cursor enters the window's client area.

Parameters

hcursor is a Windows handle to the mouse cursor, usually created with **fg_mouseptr()** or the Windows API function **LoadCursor()**.

Return value

none

Restrictions

none

See also

fg_mouseini(), fg_mouseptr()

Examples

MCdemo

fg_mouseini()

Prototype

```
int fg_mouseini (void);  
Function fg_mouseini () As Long  
function fg_mouseini : integer;
```

Description

The **fg_mouseini()** function initializes Fastgraph's mouse environment. It must be called before any of Fastgraph's other mouse support functions, usually in the WM_CREATE message handler.

Parameters

none

Return value

1 = The mouse initialization was successful
-1 = The mouse initialization failed

Restrictions

none

See also

fg_mousecur(), fg_mouselim(), fg_mousemov(), fg_mousepos(), fg_mouseptr(), fg_mousevis()

Examples

FrameDD, FullScr, MCdemo

fg_mouselim()

Prototype

```
void fg_mouselim (int minx, int maxx, int miny, int maxy);  
Sub fg_mouselim (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_mouselim (minx, maxx, miny, maxy : integer);
```

Description

The **fg_mouselim()** function defines the rectangular area in which the mouse cursor may move. The area is defined in virtual buffer coordinates and is automatically translated to the equivalent client coordinates.

Parameters

minx is the x coordinate of the area's left edge.

maxx is the x coordinate of the area's right edge. If *maxx* is less than *minx*, the mouse cursor can move anywhere on the screen.

miny is the y coordinate of the area's top edge.

maxy is the y coordinate of the area's bottom edge. If *maxy* is less than *miny*, the mouse cursor can move anywhere on the screen.

Return value

none

Restrictions

none

See also

fg_mouseini(), fg_mousemov()

fg_mousemov()

Prototype

```
void fg_mousemov (int ix, int iy);  
Sub fg_mousemov (ByVal ix As Long, ByVal iy As Long)  
procedure fg_mousemov (ix, iy : integer);
```

Description

The **fg_mousemov()** function moves the mouse cursor to the specified screen space position. The position is defined in virtual buffer coordinates and is automatically translated to the equivalent client coordinates. The mouse cursor is moved whether or not it is currently visible.

Parameters

ix is the x coordinate of the new mouse cursor position.

iy is the y coordinate of the new mouse cursor position.

Return value

none

Restrictions

If you attempt to move the mouse cursor outside the area defined by **fg_mouselim()**, **fg_mousemov()** positions the cursor at the nearest point possible within that area.

See also

fg_mouseini(), fg_mouselim(), fg_mousepos()

fg_mousepos()

Prototype

```
void fg_mousepos (int *ix, int *iy);  
Sub fg_mousepos (ix As Long, iy As Long)  
procedure fg_mousepos (var ix, iy : integer);
```

Description

The **fg_mousepos()** function returns the current mouse position. The position is automatically translated from client coordinates to the equivalent virtual buffer coordinates.

Parameters

ix receives the x coordinate of the mouse cursor position. If the cursor is outside the client area, *ix* will be set to -1.

iy receives the y coordinate of the mouse cursor position. If the cursor is outside the client area, *iy* will be set to -1.

Return value

none

Restrictions

none

See also

fg_mouseini(), fg_mousemov()

fg_mouseptr()

Prototype

```
HCURSOR fg_mouseptr (void *masks, int xoffset, int yoffset);
```

```
Function fg_mouseptr (masks() As Any, ByVal xoffset As Long, ByVal  
yoffset As Long) As Long
```

```
function fg_mouseptr (var masks; xoffset, yoffset : integer) : HCURSOR;
```

Description

The **fg_mouseptr()** function creates a user-defined mouse cursor. Refer to Chapter 11 of the *Fastgraph 6.0 User's Guide* for complete information about defining the mouse cursor.

Parameters

masks is a array containing the 16-element or 32-element screen mask followed by the 16-element or 32-element cursor mask. The mouse driver displays the mouse cursor by logically ANDing the screen contents with the screen mask, and then XORing that result with the cursor mask. The first item of each mask corresponds to the top row of the mouse cursor. The following table summarizes the cursor appearance for all possible combinations of mask bits.

Screen Mask Bit	Cursor Mask Bit	Resulting Cursor Pixel
0	0	black
0	1	white
1	0	unchanged
1	1	inverted

xoffset is the x coordinate of the "hot spot" relative to the upper left corner of the mouse cursor.

yoffset is the y coordinate of the "hot spot" relative to the upper left corner of the mouse cursor.

Return value

A Windows handle to the user-defined mouse cursor.

Restrictions

none

See also

fg_mousecur(), fg_mouseini(), fg_mousesiz(), fg_mousevis()

Examples

MCdemo

fg_mousesiz()

Prototype

```
void fg_mousesiz (int pixels);  
Sub fg_mousesiz (ByVal pixels As Long)  
procedure fg_mousesiz (pixels : integer);
```

Description

The **fg_mousesiz()** function defines the mouse cursor size in pixels. Valid sizes are 16x16 (the default) and 32x32.

Parameters

pixels is the mouse cursor width and height in pixels. It must be either 16 or 32.

Return value

none

Restrictions

The new cursor size does not take effect until the next **fg_mouseptr()** call.

See also

fg_mouseini(), fg_mouseptr(), fg_mousevis()

fg_mousevis()

Prototype

```
void fg_mousevis (int state);  
Sub fg_mousevis (ByVal state As Long)  
procedure fg_mousevis (state : integer);
```

Description

The **fg_mousevis()** function makes the mouse cursor visible or invisible.

Parameters

state defines the mouse cursor visibility. If *state* is 0, the mouse cursor is made invisible. If it is 1, the mouse cursor is made visible.

Return value

none

Restrictions

none

See also

fg_mouseini()

Examples

FrameDD, FullScr

fg_move()

Prototype

```
void fg_move (int ix, int iy);  
Sub fg_move (ByVal ix As Long, ByVal iy As Long)  
procedure fg_move (ix, iy : integer);
```

Description

The **fg_move()** function establishes the graphics cursor position at an absolute screen space point. The **fg_vbinit()** function sets the graphics cursor position to (0,0).

Parameters

ix is the screen space x coordinate of the graphics cursor's new position.

iy is the screen space y coordinate of the graphics cursor's new position.

Return value

none

Restrictions

none

See also

fg_moverel(), fg_moverw(), fg_movew()

Examples

Most of the example programs use this function.

fg_move3d()

Prototype

```
void fg_move3d (long *transform, long x, long y, long z, int flag);  
Sub fg_move3d (transform() As Long, ByVal x As Long, ByVal y As Long,  
ByVal z As Long, ByVal flag As Long)  
procedure fg_move3d (var transform : longint; x, y, z : longint; flag :  
integer);
```

Description

The **fg_move3d()** legacy function builds a 3D transformation matrix representing an absolute or relative move by the specified 3D coordinates.

Parameters

transform is the name of the 12-element transformation matrix containing fixed point 3D rotation and 3D translation values.

x, *y*, and *z* are the fixed point values defining the new (x,y,z) position in 3D space.

flag controls if the move is absolute or relative. If *flag* is zero, the move is absolute and the specified coordinates become the new translation values in the transformation matrix. If *flag* is not zero, the move is relative and the coordinates are added to the translation values in the transformation matrix.

Return value

none

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_moverel()

Prototype

```
void fg_moverel (int ix, int iy);  
Sub fg_moverel (ByVal ix As Long, ByVal iy As Long)  
procedure fg_moverel (ix, iy : integer);
```

Description

The **fg_moverel()** function establishes the graphics cursor position at a screen space point relative to the current position.

Parameters

ix is the screen space x offset of the graphics cursor's new position.

iy is the screen space y offset of the graphics cursor's new position.

Return value

none

Restrictions

none

See also

fg_move(), fg_moverw(), fg_movew()

Examples

Rotate, TMcube, TMcubeX

fg_moverw()

Prototype

```
void fg_moverw (double x, double y);  
Sub fg_moverw (ByVal x As Double, ByVal y As Double)  
procedure fg_moverw (x, y : real);
```

Description

The **fg_moverw()** function establishes the graphics cursor position at a 2D world space point relative to the current position.

Parameters

x is the world space *x* offset of the graphics cursor's new position.

y is the world space *y* offset of the graphics cursor's new position.

Return value

none

Restrictions

none

See also

fg_move(), fg_moverel(), fg_movew()

fg_movew()

Prototype

```
void fg_movew (double x, double y);  
Sub fg_movew (ByVal x As Double, ByVal y As Double)  
procedure fg_movew (x, y : real);
```

Description

The **fg_movew()** function establishes the graphics cursor position at an absolute 2D world space point. The **fg_initw()** function sets the graphics cursor position to (0.0,0.0).

Parameters

x is the world space x coordinate of the graphics cursor's new position.

y is the world space y coordinate of the graphics cursor's new position.

Return value

none

Restrictions

none

See also

fg_move(), fg_moverel(), fg_moverw()

Examples

SWchars

fg_opacity()

Prototype

```
void fg_opacity (int opacity);  
Sub fg_opacity (ByVal opacity As Long)  
procedure fg_opacity (opacity : integer);
```

Description

The **fg_opacity()** function defines the foreground opacity value used by the alpha blending functions.

Parameters

opacity is the opacity value. It must be between 0 (foreground is transparent) and 255 (foreground is completely opaque).

Return value

none

Restrictions

none

See also

fg_blend(), fg_blenddcb(), fg_blendvb()

Examples

Blend

fg_pack()

Prototype

```
void fg_pack (void *source, void *dest, int width, int height);  
Sub fg_pack (source() As Any, dest() As Any, ByVal width As Long, ByVal  
height As Long)  
procedure fg_pack (var source, dest; width, height : integer);
```

Description

The **fg_pack()** legacy function converts a 256-color bitmap a 16-color bitmap.

Parameters

source is the name of the array containing the 256-color bitmap to convert.

dest is the name of the array that will receive the converted 16-color bitmap.

width is the *source* bitmap width in pixels.

height is the *source* bitmap height in pixels.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_pagesize()

Prototype

```
long fg_pagesize (int width, int height);  
Function fg_pagesize (ByVal width As Long, ByVal height As Long) As Long  
function fg_pagesize (width, height : integer) : longint;
```

Description

The **fg_pagesize()** legacy function returns the number of bytes needed for a virtual buffer of the specified dimensions, using the current color depth. It is equivalent to the **fg_vbsize()** function.

Parameters

width specifies the virtual buffer width in pixels.

height specifies the virtual buffer height in pixels.

Return value

The virtual buffer size in bytes.

Restrictions

none

Replaced by

fg_vbsize()

fg_paint()

Prototype

```
void fg_paint (int ix, int iy);  
Sub fg_paint (ByVal ix As Long, ByVal iy As Long)  
procedure fg_paint (ix, iy : integer);
```

Description

The **fg_paint()** function fills an arbitrary closed region with pixels of the current color. The region is defined by specifying a screen space point within its interior.

Parameters

ix is the screen space x coordinate of the interior point.

iy is the screen space y coordinate of the interior point.

Return value

none

Restrictions

The virtual buffer edges are *not* considered region boundaries, and filling an open region will cause **fg_paint()** to behave unpredictably.

See also

fg_flood(), fg_paintw()

Examples

Graphics, Scroller

fg_paintw()

Prototype

```
void fg_paintw (double x, double y);  
Sub fg_paintw (ByVal x As Double, ByVal y As Double)  
procedure fg_paintw (x, y : real);
```

Description

The **fg_paintw()** function fills an arbitrary closed region with pixels of the current color. The region is defined by specifying a 2D world space point within its interior.

Parameters

x is the world space *x* coordinate of the interior point.

y is the world space *y* coordinate of the interior point.

Return value

none

Restrictions

The virtual buffer edges are *not* considered region boundaries, and filling an open region will cause **fg_paintw()** to behave unpredictably.

See also

fg_floodw(), fg_paint()

fg_paste()

Prototype

```
void fg_paste (void *bitmap, void *section, int xpos, int ypos, int
width, int sec_width, int sec_height);

Sub fg_paste (bitmap() As Any, section() As Any, ByVal xpos As Long,
ByVal ypos As Long, ByVal width As Long, ByVal sec_width As Long, ByVal
sec_height As Long)

procedure fg_paste (var bitmap, section; xpos, ypos, width, sec_width,
sec_height : integer);
```

Description

The **fg_paste()** function inserts a 256-color bitmap section into a 256-color bitmap.

Parameters

bitmap is the name of the array containing the bitmap into which **fg_paste()** will insert the bitmap section.

section is the name of the array containing the bitmap section. This bitmap is not modified in any way.

xpos is the x coordinate that defines where to insert the bitmap section within *bitmap*. The bitmap section's left edge will be at this position.

ypos is the y coordinate that defines where to insert the bitmap section within *bitmap*. The bitmap section's bottom edge will be at this position.

width is the width of *bitmap* in pixels.

sec_width is the bitmap section width in pixels.

sec_height is the bitmap section height in pixels.

Return value

none

Restrictions

none

See also

fg_cut(), fg_pastedcb()

fg_pastedcb()

Prototype

```
void fg_pastedcb (void *bitmap, void *section, int xpos, int ypos, int
width, int sec_width, int sec_height);

Sub fg_pastedcb (bitmap() As Any, section() As Any, ByVal xpos As Long,
ByVal ypos As Long, ByVal width As Long, ByVal sec_width As Long, ByVal
sec_height As Long)

procedure fg_pastedcb (var bitmap, section; xpos, ypos, width,
sec_width, sec_height : integer);
```

Description

The **fg_pastedcb()** function inserts a direct color bitmap section into a direct color bitmap.

Parameters

bitmap is the name of the array containing the bitmap into which **fg_pastedcb()** will insert the bitmap section.

section is the name of the array containing the bitmap section. This bitmap is not modified in any way.

xpos is the x coordinate that defines where to insert the bitmap section within *bitmap*. The bitmap section's left edge will be at this position.

ypos is the y coordinate that defines where to insert the bitmap section within *bitmap*. The bitmap section's bottom edge will be at this position.

width is the width of *bitmap* in pixels.

sec_width is the bitmap section width in pixels.

sec_height is the bitmap section height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_cutdcb(), fg_paste()

fg_pcxhead()

Prototype

```
int fg_pcxhead (char *filename, void *header);
```

```
Function fg_pcxhead (ByVal filename As String, header() As Any) As Long
```

```
function fg_pcxhead (filename : string; var header) : integer;
```

Description

The **fg_pcxhead()** function reads a PCX file header into a 128-byte buffer. Refer to Appendix E of the *Fastgraph 6.0 User's Guide* for details about the PCX header.

Parameters

filename is the name of the PCX file. It may include a path specification and must be terminated by a zero byte.

header is the name of the buffer to receive the PCX file header. Its size must be at least 128 bytes.

Return value

0 = Success

-1 = The specified file does not exist

-2 = The specified file is not a PCX file

Restrictions

none

See also

fg_pcxpal(), fg_pcxrange(), fg_pcxsize(), fg_showpcx()

Examples

Image, ImgProc

fg_pcxpal()

Prototype

```
int fg_pcxpal (char *filename, void *palette);
```

```
Function fg_pcxpal (ByVal filename As String, palette() As Any) As Long  
function fg_pcxpal (filename : string; var palette) : integer;
```

Description

The **fg_pcxpal()** function retrieves the palette of an image stored in a PCX file. The palette values are returned as RGB color components, each between 0 and 255.

If the PCX file includes an extended (256-color) palette, **fg_pcxpal()** will return the values in the extended palette. Otherwise, **fg_pcxpal()** will return the values from the 16-color palette in the PCX header.

Parameters

filename is name of the PCX file. The file name must be terminated by a zero byte.

palette is the name of the array that will receive the PCX palette values. The palette values are returned as RGB color components, each between 0 and 255. The first three bytes of *palette* will contain the RGB values for color 0, the next three for color 1, and so forth. The size of the *palette* array must be at least three times the number of colors in the PCX palette. You can also specify NULL for the *palette* parameter, in which case **fg_pcxpal()** will return the image's color depth but no palette values.

Return value

- >0 = The number of colors in the PCX palette (16 or 256)
- 0 = The PCX file does not have a palette (24-bit PCX file)
- 1 = The specified file does not exist
- 2 = The specified file is not a PCX file

Restrictions

none

See also

fg_pcxhead(), fg_setdacs(), fg_showpcx()

Examples

Image, ImgProc

fg_pcxrange()

Prototype

```
void fg_pcxrange (void *header, int *minx, int *maxx, int *miny, int *maxy);
```

```
Sub fg_pcxrange (header() As Any, minx As Long, maxx As Long, miny As Long, maxy As Long)
```

```
procedure fg_pcxrange (var header; var minx, maxx, miny, maxy : integer);
```

Description

The **fg_pcxrange()** function returns the image extents for the PCX image associated with the specified PCX file header.

Parameters

header is the name of the buffer containing the 128-byte PCX file header.

minx receives the x coordinate of the image's left edge. If *header* does not contain a valid PCX file header, *minx* will be set to -1.

maxx receives the x coordinate of the image's right edge. If *header* does not contain a valid PCX file header, *maxx* will be set to -1.

miny receives the y coordinate of the image's top edge. If *header* does not contain a valid PCX file header, *miny* will be set to -1.

maxy receives the y coordinate of the image's bottom edge. If *header* does not contain a valid PCX file header, *maxy* will be set to -1.

Return value

none

Restrictions

none

See also

fg_pcxhead(), fg_pcxsize(), fg_showpcx()

fg_pcxsize()

Prototype

```
void fg_pcxsize (void *header, int *width, int *height);  
Sub fg_pcxsize (header() As Any, width As Long, height As Long)  
procedure fg_pcxsize (var header; var width, height : integer);
```

Description

The **fg_pcxsize()** function returns the image dimensions for the PCX image associated with the specified PCX file header.

Parameters

header is the name of the buffer containing the 128-byte PCX file header.

width receives the PCX image width in pixels. If *header* does not contain a valid PCX file header, *width* will be set to -1.

height receives the PCX image height in pixels. If *header* does not contain a valid PCX file header, *height* will be set to -1.

Return value

none

Restrictions

none

See also

fg_pcxhead(), fg_pcxrange(), fg_showpcx()

Examples

Image, ImgProc

fg_photodcb()

Prototype

```
void fg_photodcb (void *source, void *dest, int size);  
Sub fg_photodcb (source() As Any, dest() As Any, ByVal size As Long)  
procedure fg_photodcb (var source, dest; size : integer);
```

Description

The **fg_photodcb()** function applies a photo-inversion transform to a direct color bitmap.

Parameters

source is the name of the array containing the direct color bitmap to be transformed.

dest is the name of the array that will receive the resulting transformed bitmap.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_photorgb(), fg_photovb()

fg_photorgb()

Prototype

```
void fg_photorgb (void *values, int count);  
Sub fg_photorgb (values() As Any, ByVal count As Long)  
procedure fg_photorgb (var values; count : integer);
```

Description

The **fg_photorgb()** function applies a photo-inversion transform to a series of RGB color triples.

Parameters

values is the name of the array containing the RGB color components, arranged as three-byte RGB triples. Each RGB color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least 3**count* bytes.

count is the number of RGB color triples to transform.

Return value

none

Restrictions

none

See also

fg_photodcb(), fg_photovb()

fg_photovb()

Prototype

```
void fg_photovb (int width, int height);  
Sub fg_photovb (ByVal width As Long, ByVal height As Long)  
procedure fg_photovb (width, height : integer);
```

Description

The **fg_photovb()** function applies a photo-inversion transform to a rectangular region of the active virtual buffer. The region's lower left corner is at the current graphics position.

Parameters

width is the region's width in pixels.

height is the region's height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_photodcb(), fg_photorgb()

Examples

ImgProc

fg_point()

Prototype

```
void fg_point (int ix, int iy);  
Sub fg_point (ByVal ix As Long, ByVal iy As Long)  
procedure fg_point (ix, iy : integer);
```

Description

The **fg_point()** function draws a point (displays a pixel) in screen space, with clipping.

Parameters

ix is the point's screen space x coordinate.

iy is the point's screen space y coordinate.

Return value

none

Restrictions

none

See also

fg_pointw(), fg_pointx(), fg_putpixel()

Examples

Graphics

fg_pointw()

Prototype

```
void fg_pointw (double x, double y);  
Sub fg_pointw (ByVal x As Double, ByVal y As Double)  
procedure fg_pointw (x, y : real);
```

Description

The **fg_pointw()** function draws a point (displays a pixel) in 2D world space, with clipping.

Parameters

x is the point's world space x coordinate.

y is the point's world space y coordinate.

Return value

none

Restrictions

none

See also

fg_point(), fg_pointxw()

fg_pointx()

Prototype

```
void fg_pointx (int ix, int iy);  
Sub fg_pointx (ByVal ix As Long, ByVal iy As Long)  
procedure fg_pointx (ix, iy : integer);
```

Description

The **fg_pointx()** function draws a point (displays a pixel) in "exclusive or" mode in screen space, with clipping.

Parameters

ix is the point's screen space x coordinate.

iy is the point's screen space y coordinate.

Return value

none

Restrictions

none

See also

fg_point(), fg_pointxw()

fg_pointxw()

Prototype

```
void fg_pointxw (double x, double y);  
Sub fg_pointxw (ByVal x As Double, ByVal y As Double)  
procedure fg_pointxw (x, y : real);
```

Description

The **fg_pointxw()** function draws a point (displays a pixel) in "exclusive or" mode in 2D world space, with clipping.

Parameters

x is the point's world space x coordinate.

y is the point's world space y coordinate.

Return value

none

Restrictions

none

See also

fg_pointw(), fg_pointx()

fg_polyedge()

Prototype

```
void fg_polyedge (int edge_flag);  
Sub fg_polyedge (ByVal edge_flag As Long)  
procedure fg_polyedge (edge_flag : integer);
```

Description

The **fg_polyedge()** function specifies if polygons drawn with **fg_polyfill()** will have their right and bottom edge pixels included. By default, such pixels are excluded.

Parameters

edge_flag controls the drawing of a filled polygon's right and bottom edge pixels. If *edge_flag* is 0, these pixels are included. If it is 1, these pixels are excluded.

Return value

none

Restrictions

This function has no effect when using Direct3D (edge pixels are always excluded with Direct3D).

See also

fg_inside(), fg_polyfill()

fg_polyfill()

Prototype

```
void fg_polyfill (int *vertex_array, void *unused, int n);  
  
Sub fg_polyfill (vertex_array() As Long, unused() As Any, ByVal n As  
Long)  
  
procedure fg_polyfill (var vertex_array : integer; var unused; n :  
integer);
```

Description

The **fg_polyfill()** function draws a filled convex polygon in screen space, with clipping and optional backface removal. The polygon is filled with pixels of the current color. By default, pixels along the polygon's right and bottom edges are excluded to improve polygon meshing. This feature can be disabled through **fg_polyedge()**.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth. If backface removal is specified, the vertices must be stored in clockwise order, meaning you would travel clockwise along the polygon edge to go from one vertex to the next when facing the front of the polygon.

unused is an unused parameter. In previous versions of Fastgraph, it was a work array used internally by **fg_polyfill()**.

n is the number of vertices in the polygon. To specify backface removal, negate *n*. For example, to draw a four-vertex polygon with backface removal, set *n* to -4.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon, only a portion of it will be filled.

If *n* is negative and the vertices are not stored in clockwise order, the polygon will not be drawn correctly.

See also

fg_inside(), fg_polyedge(), fg_polyline(), fg_polyoff()

Examples

Graphics

fg_polyfilz()

Prototype

```
void fg_polyfilz (int *vertex_array, double *xyz_array, int n);  
Sub fg_polyfilz (vertex_array() As Long, xyz_array() As Double, ByVal n  
As Long)  
procedure fg_polyfilz (var vertex_array : integer; var xyz_array :  
byte; n : integer);
```

Description

The **fg_polyfilz()** function draws a projected z-buffered filled convex polygon in screen space, with 2D clipping. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

xyz_array is the name of the array containing the 3D (x,y,z) coordinates for each (x,y) coordinate pair in *vertex_array*. The first three *xyz_array* elements represent the (x,y,z) values at the first vertex in *vertex_array*, the next three *xyz_array* elements are for the second vertex, and so forth. Only the z coordinates are meaningful in this function.

n is the number of vertices each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_polyfilz()**, only a portion of the polygon will be filled.

See also

fg_inside(), fg_polyfill(), fg_polyoff(), fg_zbopen()

fg_polygon()

Prototype

```
void fg_polygon (int *ix_array, int *iy_array, int n);  
Sub fg_polygon (ix_array() As Long, iy_array() As Long, ByVal n As  
Long)  
procedure fg_polygon (var ix_array, iy_array : integer; n : integer);
```

Description

The **fg_polygon()** function draws an unfilled polygon in screen space, using two coordinate arrays to define the polygon vertices. The drawing of the polygon begins at the first vertex defined in the coordinate arrays, through the remaining vertices in sequence, and finally back to the first vertex if necessary.

Parameters

ix_array is the name of the array containing the screen space x coordinates of the polygon vertices.

iy_array is the name of the array containing the screen space y coordinates of the polygon vertices.

n is the number of vertices in the polygon.

Return value

none

Restrictions

none

See also

fg_polyline(), fg_polygonw()

fg_polygonw()

Prototype

```
void fg_polygonw (double *x_array, double *y_array, int n);  
Sub fg_polygonw (x_array() As Double, y_array() As Double, ByVal n As Long)  
procedure fg_polygonw (var x_array, y_array : real; n : integer);
```

Description

The **fg_polygonw()** function draws an unfilled polygon in 2D world space, using two coordinate arrays to define the polygon vertices. The drawing of the polygon begins at the first vertex defined in the coordinate arrays, through the remaining vertices in sequence, and finally back to the first vertex if necessary.

Parameters

x_array is the name of the array containing the world space x coordinates of the polygon vertices.

y_array is the name of the array containing the world space y coordinates of the polygon vertices.

n is the number of vertices in the polygon.

Return value

none

Restrictions

none

See also

fg_polygon()

fg_polyline()

Prototype

```
void fg_polyline (int *vertex_array, int n);  
Sub fg_polyline (vertex_array() As Long, ByVal n As Long)  
procedure fg_polyline (var vertex_array : integer; n : integer);
```

Description

The **fg_polyline()** function draws an unfilled polygon in screen space, using a single array to define the polygon vertices. Compare this to **fg_polygon()**, which uses separate arrays for the x and y components of each vertex.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

n is the number of vertices in the polygon.

Return value

none

Restrictions

none

See also

fg_polyfill(), fg_polygon(), fg_polyoff()

fg_polyoff()

Prototype

```
void fg_polyoff (int ix, int iy);  
Sub fg_polyoff (ByVal ix As Long, ByVal iy As Long)  
procedure fg_polyoff (ix, iy : integer);
```

Description

The **fg_polyoff()** function defines the screen space offset applied to each polygon vertex in all 2D and 3D polygon drawing functions except **fg_polygon()** and **fg_polyline()**. By default, the polygon drawing functions use an offset of zero, meaning their vertex arrays specify the actual vertex coordinates.

Parameters

ix is the horizontal screen space offset applied to the x component of all vertices.

iy is the vertical screen space offset applied to the y component of all vertices.

Return value

none

Restrictions

none

See also

fg_3Dpolygon(), fg_3Dpolygonobject(), fg_3Dshade(), fg_3Dshadeobject(), fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_gouraud(), fg_gouraudz(), fg_inside(), fg_polyfill(), fg_polyfilz(), fg_polyline(), fg_texmap(), fg_textmapp(), fg_textmappz(), fg_texmapz()

Examples

Graphics

fg_print()

Prototype

```
void fg_print (char *string, int n);  
Sub fg_print (ByVal string As String, ByVal n As Long)  
procedure fg_print (string : string; n : integer);
```

Description

The **fg_print()** function displays a character string relative to the current graphics position using the current color. The characters are clipped at the client area or virtual buffer edges, not the area defined with **fg_setclip()**. By default, strings are displayed such that the bottom row of the first character is at the current graphics position. On return, the graphics cursor is positioned just to the right of the last character displayed.

By default, **fg_print()** displays strings directly in the window's client area, but strings can be redirected to the active virtual buffer with **fg_fontdc()**.

Parameters

string is the sequence of characters to display.

n is the number of characters to display from *string*.

Return value

none

Restrictions

You cannot direct strings to virtual buffers created with **fg_vbdefine()**.

See also

fg_fontdc(), fg_fontload(), fg_justify(), fg_logfont()

Examples

Fontdemo, Strings1, Strings2

fg_printer()

Prototype

```
int fg_printer (int message);
Function fg_printer (ByVal message As Long) As Long
function fg_printer (message : integer) : integer;
```

Description

The **fg_printer()** function issues a printer request. It works together with **fg_vbprint()** to print the contents of a virtual buffer.

Parameters

message is a code specifying one of the following printer functions:

Symbol	Value	Meaning
ABORTDOC	2	Abort the current print job
ENDDOC	11	End a print job
NEWFRAME	1	Issue a page eject
STARTDOC	10	Start a print job

Return value

For STARTDOC requests, the return value will be one of the following:

- 0 = Success
- 1 = No default printer is defined
- 2 = The printer is off-line or otherwise unavailable
- 3 = The printer does not support the STRETCHDIB function
- 4 = A printer device context could not be created

For all other requests, the return value will be zero.

Restrictions

none

See also

fg_vbprint()

Examples

Prdemo

fg_project()

Prototype

```
void fg_project (long *transform, long *source, int *dest, int n);  
Sub fg_project (transform() As Long, source() As Long, dest() As Long,  
ByVal n As Long)  
procedure fg_project (var transform, source : longint; var dest :  
integer; n : integer);
```

Description

The **fg_project()** legacy function projects a series of 3D (x,y,z) points to 2D (x,y) screen space points, using the specified transformation matrix.

Parameters

transform is the name of the 12-element transformation matrix containing fixed point 3D rotation and 3D translation values.

source is the name of the array containing the fixed point 3D (x,y,z) coordinates to project to screen space. The first three elements of the *source* array contain the coordinates for the first point, the next three elements are for the next point, and so on.

dest is the name of the array that receives the projected 2D (x,y) screen space coordinates. The first two elements of the *dest* array will contain the coordinates for the first point, the next two elements will contain the next point, and so on. The size of *dest* must be large enough to hold 2/3*n integer values.

n is the number of 3D points to project.

Return value

none

Restrictions

Before using this function, you must set up a 3D viewport with **fg_view3d()**.

Replaced by

Floating point 3D geometry system

fg_putblock()

Prototype

```
void fg_putblock (void *buffer, int minx, int maxx, int miny, int
maxy);

Sub fg_putblock (buffer() As Any, ByVal minx As Long, ByVal maxx As
Long, ByVal miny As Long, ByVal maxy As Long)

procedure fg_putblock (var buffer; minx, maxx, miny, maxy : integer);
```

Description

The **fg_putblock()** legacy function displays a block (previously obtained with the **fg_getblock()** function) at the specified position in the active virtual buffer. The block extremes are defined in screen space.

Parameters

buffer is the name of the array containing the block.

minx is the x coordinate of the block's left edge.

maxx is the x coordinate of the block's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the block's top edge.

maxy is the y coordinate of the block's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

Replaced by

fg_putdcb(), fg_putimage()

fg_putdcb()

Prototype

```
void fg_putdcb (void *bitmap, int width, int height);  
Sub fg_putdcb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_putdcb (var bitmap; width, height : integer);
```

Description

The **fg_putdcb()** function displays a direct color bitmap, without clipping. It is similar to **fg_drawdcb()** but does not treat color 0 pixels as transparent. The bitmap will be positioned so that its lower left corner is at the graphics cursor position.

For high color virtual buffers, each pixel in the bitmap is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipdcb(), fg_drawdcb(), fg_flipdcb(), fg_getdcb(), fg_invdcb(), fg_putimage(), fg_revdcdb()

Examples

Blend, Dcb

fg_putimage()

Prototype

```
void fg_putimage (void *bitmap, int width, int height);  
Sub fg_putimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_putimage (var bitmap; width, height : integer);
```

Description

The **fg_putimage()** function displays a 256-color bitmap, without clipping. It is similar to **fg_drwimage()** but does not treat color 0 pixels as transparent. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_drwimage(), fg_getimage(), fg_invert()

Examples

Bitmap

fg_putpixel()

Prototype

```
void fg_putpixel (int ix, int iy);  
Sub fg_putpixel (ByVal ix As Long, ByVal iy As Long)  
procedure fg_putpixel (ix, iy : integer);
```

Description

The **fg_putpixel()** function displays a pixel in screen space, without clipping.

Parameters

ix is the pixel's screen space x coordinate.

iy is the pixel's screen space y coordinate.

Return value

none

Restrictions

none

See also

fg_point(), fg_pointx()

fg_realize()

Prototype

```
void fg_realize (HPALETTE hpal);  
Sub fg_realize (ByVal hpal As Long)  
procedure fg_realize (hpal : HPALETTE);
```

Description

The **fg_realize()** function activates or “realizes” a logical palette created with **fg_defpal()** or **fg_logpal()**. The colors defined in a logical palette determine the colors in which pixels are displayed in the client area. The WM_SETFOCUS message handler usually calls **fg_realize()**.

Parameters

hpal is a Windows handle to the logical palette, as returned by **fg_defpal()** or **fg_logpal()**.

Return value

none

Restrictions

none

See also

fg_defpal(), **fg_getdacs()**, **fg_getrgb()**, **fg_logpal()**, **fg_setdacs()**, **fg_setrgb()**

Examples

All the example programs use this function.

fg_rect()

Prototype

```
void fg_rect (int minx, int maxx, int miny, int maxy);  
Sub fg_rect (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long)  
procedure fg_rect (minx, maxx, miny, maxy : integer);
```

Description

The **fg_rect()** function draws a solid (filled) rectangle in screen space, without clipping.

Parameters

minx is the x coordinate of the rectangle's left edge.

maxx is the x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the rectangle's top edge.

maxy is the y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_box(), fg_clprect(), fg_drect(), fg_rectw(), fg_rectx()

Examples

Graphics, Scroller, VBdemo

fg_rectw()

Prototype

```
void fg_rectw (double xmin, double xmax, double ymin, double ymax);  
Sub fg_rectw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin As  
Double, ByVal ymax As Double)  
procedure fg_rectw (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_rectw()** function draws a solid (filled) rectangle in 2D world space, without clipping.

Parameters

xmin is the world space x coordinate of the rectangle's left edge.

xmax is the world space x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the rectangle's bottom edge.

ymax is the world space y coordinate of the rectangle's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

See also

fg_boxw(), fg_clprectw(), fg_drectw(), fg_rect()

fg_rectx()

Prototype

```
void fg_rectx (int minx, int maxx, int miny, int maxy);  
Sub fg_rectx (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_rectx (minx, maxx, miny, maxy : integer);
```

Description

The **fg_rectx()** function draws a solid (filled) rectangle in “exclusive or” mode in screen space, without clipping.

Parameters

minx is the x coordinate of the rectangle's left edge.

maxx is the x coordinate of the rectangle's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the rectangle's top edge.

maxy is the y coordinate of the rectangle's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_clprectx(), fg_rect()

fg_reduce()

Prototype

```
void fg_reduce (int offset, int colors, void *values);  
Sub fg_reduce (ByVal offset As Long, ByVal colors As Long, values() As  
Any)  
procedure fg_reduce (offset, colors : integer; var values);
```

Description

The **fg_reduce()** function reduces a 256-color palette to the specified number of colors, and optionally applies a color offset to the resulting palette. The pixel data within the current clipping region of the active 256-color virtual buffer determines the color occurrences used in the color reduction process.

On return, the pixels in the clipping region are translated to the colors defined by the resulting palette, but the palette itself is not made active.

Parameters

offset is the color offset applied to the resulting palette, between 0 and 255.

colors is the number of unique colors desired in the resulting palette, between 1 and 256.

values is the name of a 768-byte array containing 256 sets of RGB color components for the pixel data in the active virtual buffer. On return, the entries for colors *offset* through *offset+colors-1* will contain the RGB color components for the new palette. Any resulting unused entries at the beginning or end of the *values* array are set to zero.

Return value

none

Restrictions

This function is meaningful only for 256-color virtual buffers.

See also

fg_getdacs(), fg_setclip(), fg_setdacs()

fg_restore()

Prototype

```
void fg_restore (int minx, int maxx, int miny, int maxy);  
  
Sub fg_restore (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
  
procedure fg_restore (minx, maxx, miny, maxy : integer);
```

Description

The **fg_restore()** legacy function copies a rectangular region, defined in screen space, from the background virtual buffer to the same position in the foreground virtual buffer. As with Fastgraph's other block transfer functions, no clipping is performed.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

When using DirectX, the background and foreground virtual buffers must not be locked.

Restrictions

none

Replaced by

fg_vbcopy()

fg_restorew()

Prototype

```
void fg_restorew (double xmin, double xmax, double ymin, double ymax);  
Sub fg_restorew (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin  
As Double, ByVal ymax As Double)  
procedure fg_restorew (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_restorew()** legacy function copies a rectangular region, defined in 2D world space, from the background virtual buffer to the same position in the foreground virtual buffer. As with Fastgraph's other block transfer functions, no clipping is performed.

Parameters

xmin is the world space x coordinate of the region's left edge.

xmax is the world space x coordinate of the region's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the region's bottom edge.

ymax is the world space y coordinate of the region's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

Replaced by

fg_vbcopy()

fg_revdcdb()

Prototype

```
void fg_revdcdb (void *bitmap, int width, int height);  
Sub fg_revdcdb (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_revdcdb (var bitmap; width, height : integer);
```

Description

The **fg_revdcdb()** function displays a reversed direct color bitmap, without clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Color 0 pixels will be considered transparent.

For high color virtual buffers, each pixel in the bitmap is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about direct color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_clipdcdb(), fg_drawdcdb(), fg_flipdcdb(), fg_getdcdb(), fg_invdcdb(), fg_putdcdb(), fg_revimage()

Examples

Dcb

fg_revimage()

Prototype

```
void fg_revimage (void *bitmap, int width, int height);  
Sub fg_revimage (bitmap() As Any, ByVal width As Long, ByVal height As Long)  
procedure fg_revimage (var bitmap; width, height : integer);
```

Description

The **fg_revimage()** function displays a reversed 256-color bitmap, without clipping. The bitmap will be positioned so that its lower left corner is at the graphics cursor position. Refer to Chapter 8 of the *Fastgraph 6.0 User's Guide* for complete information about 256-color bitmaps.

Parameters

bitmap is the name of the array containing the bitmap.

width is the bitmap width in pixels.

height is the bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_clpimage(), fg_drwimage(), fg_flpimage(), fg_getimage(), fg_invert(), fg_putimage(), fg_revdcbb()

Examples

Bitmap, PCXflip

fg_revmask()

Prototype

```
void fg_revmask (void *bitmap, int runs, int width);  
Sub fg_revmask (bitmap() As Any, ByVal runs As Long, ByVal width As Long)  
procedure fg_revmask (var bitmap; runs, width : integer);
```

Description

The **fg_revmask()** legacy function displays a reversed masking map. The masking map will be positioned so that its lower left corner is at the graphics cursor position.

Parameters

bitmap is the name of the array containing the masking map. The masking map is a series of alternating "protect" and "zero" pixel runs. The "protect" runs leave the corresponding virtual buffer pixels unchanged, while the "zero" runs set them to color zero. The length of each run must be between 0 and 255.

runs is the number of pixel runs in the masking map.

width is the masking map width in pixels.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_rotate()

Prototype

```
void fg_rotate (void *source, void *dest, int width, int height, int
angle);

Sub fg_rotate (source() As Any, dest() As Any, ByVal width As Long,
ByVal height As Long, ByVal angle As Long)

procedure fg_rotate (var source, dest; width, height, angle : integer);
```

Description

The **fg_rotate()** function rotates a 256-color bitmap (around its center) by a specified angle.

Parameters

source is the name of the array containing the 256-color bitmap to be rotated.

dest is the name of the array that will receive the resulting rotated 256-color bitmap.

width is the *source* bitmap width in pixels. It must be greater than zero.

height is the *source* bitmap height in pixels. It must be greater than zero.

angle is the rotation angle, expressed in tenths of degrees and measured counterclockwise from the positive x axis.

Return value

none

Restrictions

none

See also

fg_rotddb(), fg_rotsize()

Examples

Rotate

fg_rotate3d()

Prototype

```
void fg_rotate3d (long *transform, int pitch, int yaw, int roll, int
flag);

Sub fg_rotate3d (transform() As Long, ByVal pitch As Long, ByVal yaw As
Long, ByVal roll As Long, ByVal flag As Long)

procedure fg_rotate3d (var transform : longint; pitch, yaw, roll, flag
: integer);
```

Description

The **fg_rotate3d()** legacy function builds a 3D transformation matrix representing an absolute or relative rotation by the specified 3D coordinates.

Parameters

transform is the name of the 12-element transformation matrix containing fixed point 3D rotation and 3D translation values.

pitch is the counterclockwise rotation about the x axis, expressed in tenths of degrees.

yaw is the counterclockwise rotation about the y axis, expressed in tenths of degrees.

roll is the counterclockwise rotation about the z axis, expressed in tenths of degrees.

flag controls if the rotation is absolute or relative. If *flag* is zero, the rotation is absolute and the specified rotations become the new rotation values in the transformation matrix. If *flag* is not zero, the rotation is relative and the rotation angles are added to the rotation values in the transformation matrix.

Return value

none

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_rotddb()

Prototype

```
void fg_rotddb (void *source, void *dest, int width, int height, int
angle);

Sub fg_rotddb (source() As Any, dest() As Any, ByVal width As Long,
ByVal height As Long, ByVal angle As Long)

procedure fg_rotddb (var source, dest; width, height, angle : integer);
```

Description

The **fg_rotddb()** function rotates a direct color bitmap (around its center) by a specified angle.

For high color virtual buffers, each pixel in the bitmaps is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte.

Parameters

source is the name of the array containing the direct color bitmap to be rotated.

dest is the name of the array that will receive the resulting rotated direct color bitmap.

width is the *source* bitmap width in pixels. It must be greater than zero.

height is the *source* bitmap height in pixels. It must be greater than zero.

angle is the rotation angle, expressed in tenths of degrees and measured counterclockwise from the positive x axis.

Return value

none

Restrictions

none

See also

fg_rotate(), fg_rotsize()

fg_rotsize()

Prototype

```
void fg_rotsize (int width, int height, int angle, int *new_width, int *new_height);
```

```
Sub fg_rotsize (ByVal width As Long, ByVal height As Long, ByVal angle As Long, new_width As Long, new_height As Long)
```

```
procedure fg_rotsize (width, height, angle : integer; var new_width, new_height : integer);
```

Description

The **fg_rotsize()** function determines the resulting dimensions of a bitmap when rotated by a given angle.

Parameters

width is the unrotated bitmap width in pixels. It must be greater than zero.

height is the unrotated bitmap height in pixels. It must be greater than zero.

angle is the rotation angle, expressed in tenths of degrees and measured counterclockwise from the positive x axis.

new_width receives the rotated bitmap width in pixels.

new_height receives the rotated bitmap height in pixels.

Return value

none

Restrictions

none

See also

fg_rotate(), fg_rotddb()

Examples

Rotate

fg_save()

Prototype

```
void fg_save (int minx, int maxx, int miny, int maxy);  
Sub fg_save (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long)  
procedure fg_save (minx, maxx, miny, maxy : integer);
```

Description

The **fg_save()** legacy function copies a rectangular region, defined in screen space, from the foreground virtual buffer to the same position in the background virtual buffer. As with Fastgraph's other block transfer functions, no clipping is performed.

Parameters

minx is the x coordinate of the region's left edge.

maxx is the x coordinate of the region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the region's top edge.

maxy is the y coordinate of the region's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

When using DirectX, the background and foreground virtual buffers must not be locked.

Replaced by

fg_vbcopy()

fg_savew()

Prototype

```
void fg_savew (double xmin, double xmax, double ymin, double ymax);  
Sub fg_savew (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin As Double, ByVal ymax As Double)  
procedure fg_savew (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_savew()** legacy function copies a rectangular region, defined in 2D world space, from the foreground virtual buffer to the same position in the background virtual buffer. As with Fastgraph's other block transfer functions, no clipping is performed.

Parameters

xmin is the world space x coordinate of the region's left edge.

xmax is the world space x coordinate of the region's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the region's bottom edge.

ymax is the world space y coordinate of the region's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

Replaced by

fg_vbcopy()

fg_scale()

Prototype

```
void fg_scale (void *source, void *dest, int sw, int sh, int dw, int dh);  
  
Sub fg_scale (source() As Any, dest() As Any, ByVal sw As Long, ByVal sh As Long, ByVal dw As Long, ByVal dh As Long)  
  
procedure fg_scale (var source, dest; sw, sh, dw, dh : integer);
```

Description

The **fg_scale()** function scales a 256-color bitmap.

Parameters

source is the name of the array containing the 256-color bitmap to be scaled.

dest is the name of the array that will receive the resulting scaled bitmap.

sw is the *source* bitmap width in pixels. It must be greater than zero.

sh is the *source* bitmap height in pixels. It must be greater than zero.

dw is the *dest* bitmap width in pixels. It must be greater than zero.

dh is the *dest* bitmap height in pixels. It must be greater than zero.

Return value

none

Restrictions

The maximum allowable width or height of *source* and *dest* is 8,192 pixels.

See also

fg_scaledcb()

Examples

Scale

fg_scaledcb()

Prototype

```
void fg_scaledcb (void *source, void *dest, int sw, int sh, int dw, int dh);
```

```
Sub fg_scaledcb (source() As Any, dest() As Any, ByVal sw As Long, ByVal sh As Long, ByVal dw As Long, ByVal dh As Long)
```

```
procedure fg_scaledcb (var source, dest; sw, sh, dw, dh : integer);
```

Description

The **fg_scaledcb()** function scales a direct color bitmap.

For high color virtual buffers, each pixel in the bitmaps is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte.

Parameters

source is the name of the array containing the direct color bitmap to be scaled.

dest is the name of the array that will receive the resulting scaled direct color bitmap.

sw is the *source* bitmap width in pixels. It must be greater than zero.

sh is the *source* bitmap height in pixels. It must be greater than zero.

dw is the *dest* bitmap width in pixels. It must be greater than zero.

dh is the *dest* bitmap height in pixels. It must be greater than zero.

Return value

none

Restrictions

The maximum allowable width or height of *source* and *dest* is 8,192 pixels.

See also

fg_scale()

fg_scroll()

Prototype

```
void fg_scroll (int minx, int maxx, int miny, int maxy, int jump, int
type);

Sub fg_scroll (ByVal minx As Long, ByVal maxx As Long, ByVal miny As
Long, ByVal maxy As Long, ByVal jump As Long, ByVal type As Long)

procedure fg_scroll (minx, maxx, miny, maxy, jump, type : integer);
```

Description

The **fg_scroll()** function vertically scrolls a region of the active virtual buffer. The scrolling may be done either up or down, using either an end-off or circular method. The region is defined in screen space.

Parameters

minx is the x coordinate of the scrolling region's left edge.

maxx is the x coordinate of the scrolling region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the scrolling region's top edge.

maxy is the y coordinate of the scrolling region's bottom edge. It must be greater than or equal to the value of *miny*.

jump is the number of pixels to jump between each scrolling iteration. If *jump* is negative, the region will scroll toward the top of the virtual buffer. If *jump* is positive, the region will scroll toward the bottom.

type specifies the type of scroll. If *type* is zero, rows that scroll off one edge appear at the opposite edge, thus producing a circular scrolling effect. If *type* is any other value, rows that scroll off one edge will be replaced at the opposite edge by lines of the current color.

Return value

none

Restrictions

Circular scrolling uses part of the background virtual buffer, as defined in the most recent call to **fg_sethpage()**, as a temporary workspace.

When using DirectX, the active virtual buffer must not be locked. The same applies to the background virtual buffer if using circular scrolling.

See also

fg_setcolor(), fg_sethpage()

Examples

Scroller

fg_setalpha()

Prototype

```
void fg_setalpha (int alpha);  
Sub fg_setalpha (ByVal alpha As Long)  
procedure fg_setalpha (alpha : integer);
```

Description

The **fg_setalpha()** legacy function defines the alpha channel value that **fg_setdacs()** and **fg_setrgb()** use when updating a 32-bit true color virtual palette. The default alpha channel value is zero.

Parameters

alpha is the alpha channel value. Its value must be between 0 and 255.

Return value

none

Restrictions

The alpha channel is meaningful only with a 32-bit true color virtual palette.

fg_setangle()

Prototype

```
void fg_setangle (double angle);  
Sub fg_setangle (ByVal angle As Double)  
procedure fg_setangle (angle : real);
```

Description

The **fg_setangle()** function defines the angle of rotation at which software characters are displayed. If a program draws software characters before calling **fg_setangle()**, Fastgraph will use its default angle of zero degrees (that is, horizontal).

Parameters

angle is the angle of rotation, expressed in degrees and measured counterclockwise from the positive x axis.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setratio(), fg_setsize(), fg_setsizew(), fg_setworld(), fg_swchar(), fg_swlength(), fg_swtext()

fg_setclip()

Prototype

```
void fg_setclip (int minx, int maxx, int miny, int maxy);  
Sub fg_setclip (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long)  
procedure fg_setclip (minx, maxx, miny, maxy : integer);
```

Description

The **fg_setclip()** function defines the active virtual buffer's clipping region in screen space. The clipping region is a rectangular area outside of which certain graphics are suppressed.

Parameters

minx is the screen space x coordinate of the clipping region's left edge.

maxx is the screen space x coordinate of the clipping region's right edge. It must be greater than or equal to the value of *minx*.

miny is the screen space y coordinate of the clipping region's top edge.

maxy is the screen space y coordinate of the clipping region's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

none

Restrictions

none

See also

fg_getclip(), fg_setclipw()

Examples

Rainbow

fg_setclipw()

Prototype

```
void fg_setclipw (double xmin, double xmax, double ymin, double ymax);  
Sub fg_setclipw (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin  
As Double, ByVal ymax As Double)  
procedure fg_setclipw (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_setclipw()** function defines the active virtual buffer's clipping region in 2D world space. The clipping region is a rectangular area outside of which certain graphics are suppressed.

Parameters

xmin is the world space x coordinate of the clipping region's left edge.

xmax is the world space x coordinate of the clipping region's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the world space y coordinate of the clipping region's bottom edge.

ymax is the world space y coordinate of the clipping region's top edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

none

See also

fg_getclip(), fg_setclip()

fg_setcolor()

Prototype

```
void fg_setcolor (int color);  
Sub fg_setcolor (ByVal color As Long)  
procedure fg_setcolor (color : integer);
```

Description

The **fg_setcolor()** function establishes the current color.

Parameters

color defines the current color. For 256-color virtual buffers, *color* must be between 0 and 255. For high color virtual buffers, *color* must be a 16-bit encoded RGB value; for true color virtual buffers, it must be a 32-bit xRGB value.

Return value

none

Restrictions

none

See also

fg_colors(), fg_getcolor(), fg_maprgb(), fg_realize()

Examples

Nearly all the example programs use this function.

fg_setdacs()

Prototype

```
void fg_setdacs (int start, int count, void *values);  
Sub fg_setdacs (ByVal start As Long, ByVal count As Long, values() As  
Any)  
procedure fg_setdacs (start, count : integer; var values);
```

Description

The **fg_setdacs()** function defines the red, green, and blue color components of a consecutive group of colors in the active logical palette or virtual palette. Defining many colors with **fg_setdacs()** is faster than doing so individually with **fg_setrgb()**.

Parameters

start is the starting color number, between 0 and 255.

count is the number of consecutive colors to define, between 1 and 256. The sum of *start* and *count* cannot exceed 256.

values is the name of the array containing the color components. The first three bytes of this array must contain the red, green, and blue components for color *start*, the next three bytes contain the components for color *start+1*, and so forth. Each color component is a value between 0 and 255; increasing values produce more intense colors. The size of the *values* array must be at least $3*count$ bytes.

Return value

none

Restrictions

Before calling **fg_setdacs()**, a logical palette must be defined and realized.

See also

fg_getdacs(), fg_mapdacs(), fg_realize(), fg_setrgb()

Examples

Fade, Rainbow

fg_setdc()

Prototype

```
void fg_setdc (HDC hdc);  
Sub fg_setdc (ByVal hdc As Long)  
procedure fg_setdc (hdc : HDC);
```

Description

The **fg_setdc()** function establishes a device context for the window's client area. It is usually one of the first Fastgraph functions called in the WM_CREATE message handler.

Parameters

hdc is a Windows handle to the client area device context, usually obtained with the **GetDC()** function from the Windows API (or the hDC property in Visual Basic programs).

Return value

none

Restrictions

none

Examples

All the example programs use this function.

fg_sethpage()

Prototype

```
void fg_sethpage (int handle);  
Sub fg_sethpage (ByVal handle As Long)  
procedure fg_sethpage (handle : integer);
```

Description

The **fg_sethpage()** function establishes the background virtual buffer for save, restore, and circular scroll operations.

Parameters

handle is the virtual buffer handle.

Return value

none

Restrictions

none

See also

fg_gethpage(), fg_scroll(), fg_vbopen()

Examples

Scroller

fg_setpage()

Prototype

```
void fg_setpage (int handle);  
Sub fg_setpage (ByVal handle As Long)  
procedure fg_setpage (handle : integer);
```

Description

The **fg_setpage()** legacy function establishes the foreground virtual buffer for save and restore operations.

Parameters

handle is the virtual buffer handle.

Return value

none

Restrictions

none

Replaced by

fg_vbcopy()

fg_setratio()

Prototype

```
void fg_setratio (double ratio);  
Sub fg_setratio (ByVal ratio As Double)  
procedure fg_setratio (ratio : real);
```

Description

The **fg_setratio()** function defines the aspect ratio for software characters. The aspect ratio is the ratio of character width to character height. If a program draws software characters before calling **fg_setratio()**, Fastgraph will use its default aspect ratio of 1.

Parameters

ratio is the aspect ratio. It must be greater than zero.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setsize(), fg_setsizew(), fg_setworld(), fg_swchar(), fg_swlength(), fg_swtext()

Examples

SWchars

fg_setrgb()

Prototype

```
void fg_setrgb (int number, int red, int green, int blue);  
Sub fg_setrgb (ByVal number As Long, ByVal red As Long, ByVal green As  
Long, ByVal blue As Long)  
procedure fg_setrgb (number, red, green, blue : integer);
```

Description

The **fg_setrgb()** function defines the red, green, and blue color components for a specified color in the active logical palette or virtual palette.

Parameters

number is the color number, between 0 and 255.

red, *green*, and *blue* respectively define the red, green, and blue components of the specified color. Each color component is a value between 0 and 255; increasing values produce more intense colors.

Return value

none

Restrictions

Before calling **fg_setrgb()**, a logical palette must be defined and realized.

See also

fg_getrgb(), fg_mapdacs(), fg_maprgb(), fg_realize(), fg_setdacs()

fg_setsize()

Prototype

```
void fg_setsize (int size);  
Sub fg_setsize (ByVal size As Long)  
procedure fg_setsize (size : integer);
```

Description

The **fg_setsize()** function defines the height of software characters in screen space units. If neither **fg_setsize()** nor **fg_setsizew()** is called, Fastgraph will use its default character height of one 2D world space unit.

Parameters

size is the character height in screen space units.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setratio(), fg_setsizew(), fg_setworld(), fg_swchar(), fg_swlength(), fg_swtext()

fg_setsizew()

Prototype

```
void fg_setsizew (double size);  
Sub fg_setsizew (ByVal size As Double)  
procedure fg_setsizew (size : real);
```

Description

The **fg_setsizew()** function defines the height of software characters in 2D world space units. If neither **fg_setsize()** nor **fg_setsizew()** is called, Fastgraph will use its default character height of one world space unit.

Parameters

size is the character height in world space units.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setratio(), fg_setsize(), fg_setworld(), fg_swchar(), fg_swlength(), fg_swtext()

Examples

SWchars

fg_setview()

Prototype

```
void fg_setview (int view_minx, int view_maxx, int view_miny, int
view_maxy, int minx, int maxx, int miny, int maxy);

Sub fg_setview (ByVal view_minx As Long, ByVal view_maxx As Long, ByVal
view_miny As Long, ByVal view_maxy As Long, ByVal minx As Long, ByVal
maxx As Long, ByVal miny As Long, ByVal maxy As Long)

procedure fg_setview (view_minx, view_maxx, view_miny, view_maxy, minx,
maxx, miny, maxy : integer);
```

Description

The **fg_setview()** function defines a 2D viewport with the specified extremes at the specified screen space position.

Parameters

view_minx is the viewport's left edge in viewport units.

view_maxx is the viewport's right edge in viewport units. Its value must be greater than *view_minx*.

view_miny is the viewport's top edge in viewport units.

view_maxy is the viewport's bottom edge in viewport units. Its value must be greater than *view_miny*.

minx is the screen space x coordinate corresponding to the viewport's left edge.

maxx is the screen space x coordinate corresponding to the viewport's right edge. It must be greater than *minx*.

miny is the screen space y coordinate corresponding to the viewport's top edge.

maxy is the screen space y coordinate corresponding to the viewport's bottom edge. It must be greater than *miny*.

Return value

none

Restrictions

none

See also

fg_getview(), fg_xview(), fg_yview()

fg_setworld()

Prototype

```
void fg_setworld (double xmin, double xmax, double ymin, double ymax);  
Sub fg_setworld (ByVal xmin As Double, ByVal xmax As Double, ByVal ymin  
As Double, ByVal ymax As Double)  
procedure fg_setworld (xmin, xmax, ymin, ymax : real);
```

Description

The **fg_setworld()** function defines the 2D world space coordinates that correspond to the edges of the active virtual buffer.

Parameters

xmin is the world space coordinate of the screen's left edge.

xmax is the world space coordinate of the screen's right edge. It must be greater than the value of *xmin*.

ymin is the world space coordinate of the screen's bottom edge.

ymax is the world space coordinate of the screen's top edge. It must be greater than the value of *ymin*.

Return value

none

Restrictions

Before using this function, you must call **fg_initw()** to initialize Fastgraph's 2D world space parameters.

See also

fg_getworld(), fg_initw(), fg_setview()

Examples

SWchars

fg_shear()

Prototype

```
void fg_shear (void *source, void *dest, int width, int height, int
new_size, int type);

Sub fg_shear (source() As Any, dest() As Any, ByVal width As Long,
ByVal height As Long, ByVal new_size As Long, ByVal type As Long)

procedure fg_shear (var source, dest; width, height, new_size, type :
integer);
```

Description

The **fg_shear()** function shears a 256-color bitmap. Shearing may be thought of as anchoring one corner of an image and stretching the opposite corner horizontally or vertically.

Parameters

source is the name of the array containing the 256-color bitmap to be sheared.

dest is the name of the array that will receive the resulting sheared bitmap.

width is the *source* bitmap width in pixels. It must be greater than zero.

height is the *source* bitmap height in pixels. It must be greater than zero.

new_size is the width in pixels (for horizontal shears) or height in pixels (for vertical shears) of the resulting *dest* bitmap. It must be at least as large as the corresponding dimension in the *source* bitmap.

type is a code indicating the shear type and direction, as shown here:

- 0 = Horizontal shear to the left (bottom edge is stretched to the right)
- 1 = Horizontal shear to the right (top edge is stretched to the right)
- 2 = Vertical shear to the left (left edge is stretched up)
- 3 = Vertical shear to the right (right edge is stretched up)

Return value

none

Restrictions

The maximum allowable width or height of *source* and *dest* is 8,192 pixels.

See also

fg_sheardcb()

Examples

Scale

fg_sheardcb()

Prototype

```
void fg_sheardcb (void *source, void *dest, int width, int height, int new_size, int type);
```

```
Sub fg_sheardcb (source() As Any, dest() As Any, ByVal width As Long, ByVal height As Long, ByVal new_size As Long, ByVal type As Long)
```

```
procedure fg_sheardcb (var source, dest; width, height, new_size, type : integer);
```

Description

The **fg_sheardcb()** function shears a direct color bitmap. Shearing may be thought of as anchoring one corner of an image and stretching the opposite corner horizontally or vertically.

For high color virtual buffers, each pixel in the bitmaps is a 16-bit (two byte) encoded RGB value. For true color virtual buffers, each pixel is a 24-bit (three byte) RGB value, stored blue byte first, then green byte, then red byte.

Parameters

source is the name of the array containing the direct color bitmap to be sheared.

dest is the name of the array that will receive the resulting sheared direct color bitmap.

width is the *source* bitmap width in pixels. It must be greater than zero.

height is the *source* bitmap height in pixels. It must be greater than zero.

new_size is the width in pixels (for horizontal shears) or height in pixels (for vertical shears) of the resulting *dest* bitmap. It must be at least as large as the corresponding dimension in the *source* bitmap..

type is a code indicating the shear type and direction, as shown here:

0 = Horizontal shear to the left (bottom edge is stretched to the right)

1 = Horizontal shear to the right (top edge is stretched to the right)

2 = Vertical shear to the left (left edge is stretched up)

3 = Vertical shear to the right (right edge is stretched up)

Return value

none

Restrictions

The maximum allowable width or height of *source* and *dest* is 8,192 pixels.

See also

fg_shear()

fg_showavi()

Prototype

```
int fg_showavi (char *filename, int count, int flags);

Function fg_showavi (ByVal filename As String, ByVal count As Long,
ByVal flags As Long) As Long

function fg_showavi (filename : string; count, flags : integer) :
integer;
```

Description

The **fg_showavi()** function plays an animation sequence from an AVI file. Each frame in the AVI file is played in the active virtual buffer and then scaled to fit the client area.

Parameters

filename is the name of the AVI file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

count is the number of times to play the AVI image. If count is zero, the AVI plays continuously. You can stop the AVI play at any time by pressing the Escape key.

flags is a series of flags that controls how the AVI is played:

Flag	Meaning
FG_AT_XY	If specified, play the AVI file relative to the current graphics position. If not, play it relative to (0,0).
FG_IGNOREAVIPALETTE	If specified, play the AVI file using the current palette. If not, use the palette values stored in the AVI file. Not meaningful for high color or true color AVI files.
FG_NODELAY	If specified, play the AVI file with no delay between frames. If not, delay between frames as specified in the AVI header.

Return value

- 0 = AVI file opened successfully
- 1 = The specified file does not exist
- 2 = The specified file is not an AVI file
- 3 = Error initializing the AVI video stream
- 4 = Error allocating memory
- 5 = The codec needed for the specified AVI file is not available

Restrictions

When using DirectX, the active virtual buffer must not be locked.

See also

fg_avihead(), fg_avisplay(), fg_avisize(), fg_realize()

Examples

Image

fg_showbmp()

Prototype

```
int fg_showbmp (char *filename, int flags);
```

Function fg_showbmp (ByVal filename As String, ByVal flags As Long) As Long

```
function fg_showbmp (filename : string; flags : integer) : integer;
```

Description

The **fg_showbmp()** function displays a BMP file.

For 256-color virtual buffers, 256-color BMP files are reduced to the 236 non-system colors if color reduction is enabled. 16-color and monochrome BMP files are always remapped to colors 10 to 25 to avoid conflicts with the system colors.

Parameters

filename is the name of the BMP file. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte).

flags is a series of flags that controls how the image is displayed:

Flag	Meaning
FG_AT_XY	If specified, display the image relative to the current graphics position. If not, display the image relative to (0,0).
FG_IGNOREPALETTE	If specified, display the image using the current palette. If not, use the palette values stored in the BMP file. Not meaningful for 24-bit BMP files.
FG_FROMBUFFER	If specified, get the image data from the fg_imagebuf() buffer. If not, get the image data from the BMP file.
FG_KEEPCOLORS	If specified, disable color reduction and remapping. If not, enable color reduction and remapping to avoid using the Windows system colors. Not meaningful for 24-bit BMP files.

Return value

0 = Success

1 = The specified file does not exist

2 = The specified file is not a BMP file

3 = The BMP file cannot be loaded into the active virtual buffer

4 = The BMP file is an unsupported RLE BMP file

5 = Error allocating memory

Restrictions

A logical palette must be defined and realized in order to use the palette values stored in the BMP file.

24-bit BMP files can only be loaded into direct color virtual buffers.

See also

fg_bmphead(), fg_bmppal(), fg_bmpsize(), fg_imagebuf(), fg_makebmp(), fg_realize()

Examples

Blend, Image, ImgProc, KBdemo, Panner, Prdemo

fg_showflic()

Prototype

```
int fg_showflic (char *filename, int count, int flags);

Function fg_showflic (ByVal filename As String, ByVal count As Long,
ByVal flags As Long) As Long

function fg_showflic (filename : string; count, flags : integer) :
integer;
```

Description

The **fg_showflic()** function plays an animation sequence from an FLI or FLC file (collectively called flic files). Each frame in the flic file is played in the active virtual buffer and then scaled to fit the client area. For 256-color virtual buffers, flic files are reduced to the 236 non-system colors if color reduction is enabled.

Parameters

filename is the name of the flic file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

count is the number of times to play the flic image. If count is zero, the flic plays continuously. You can stop the flic play at any time by pressing the Escape key.

flags is a series of flags that controls how the image is played:

Flag	Meaning
FG_AT_XY	If specified, play the flic file relative to the current graphics position. If not, play it relative to (0,0).
FG_IGNOREFLICPALETTE	If specified, play the flic file using the current palette. If not, use the palette values stored in the flic file.
FG_FROMBUFFER	If specified, get the image data from the fg_imagebuf() buffer. If not, get the image data from the flic file.
FG_KEEPCOLORS	If specified, disable color reduction and remapping. If not, enable color reduction and remapping to avoid using the Windows system colors.
FG_NODELAY	If specified, play the flic file with no delay between frames. If not, delay between frames as specified in the flic header.

Return value

0 = Success

1 = The specified file does not exist

2 = The specified file is not a flic file

Restrictions

A logical palette must be defined and realized in order to use the palette values stored in the flic file.

When using DirectX, the active virtual buffer must not be locked.

See also

fg_flichead(), fg_flicplay(), fg_flicsize(), fg_imagebuf(), fg_realize()

Examples

Image

fg_showjpeg()

Prototype

```
int fg_showjpeg (char *filename, int flags);
```

Function fg_showjpeg (ByVal filename As String, ByVal flags As Long) As Long

```
function fg_showjpeg (filename : string; flags : integer) : integer;
```

Description

The **fg_showjpeg()** function displays a JPEG file. The JPEG image can be either grayscale or color, but it must be a baseline JPEG file. Baseline JPEG files use Huffman encoding and cannot use the progressive, hierarchical, or lossless compression and storage modes.

Parameters

filename is the name of the JPEG file. It may include a path specification and must be terminated by a zero byte.

flags is a series of flags that controls how the image is displayed:

Flag	Meaning
FG_AT_XY	If specified, display the image relative to the current graphics position. If not, display the image relative to (0,0).
FG_FROMBUFFER	If specified, get the image data from the fg_imagebuf() buffer. If not, get the image data from the JPEG file.

Return value

0 = Success

1 = The specified file does not exist

2 = The specified file is not a JPEG file

3 = The JPEG file does not have a valid structure (probably not baseline JPEG)

4 = Error allocating memory

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_imagebuf(), fg_jpeghead(), fg_jpegsize()

Examples

Image, ImgProc

fg_showpcx()

Prototype

```
int fg_showpcx (char *filename, int flags);
```

```
Function fg_showpcx (ByVal filename As String, ByVal flags As Long) As Long
```

```
function fg_showpcx (filename : string; flags : integer) : integer;
```

Description

The **fg_showpcx()** function displays a PCX file.

For 256-color virtual buffers, 256-color PCX files are reduced to the 236 non-system colors if color reduction is enabled. 16-color and monochrome PCX files are always remapped to colors 10 to 25 to avoid conflicts with the system colors.

Parameters

filename is the name of the PCX file. A device and path name may be included as part of the file name. The file name must be terminated by a null character (that is, a zero byte).

flags is a series of flags that controls how the image is displayed:

Flag	Meaning
FG_AT_XY	If specified, display the image relative to the current graphics position. If not, display the image at the position indicated in PCX header.
FG_IGNOREPALETTE	If specified, display the image using the current palette. If not, use the palette values stored in the PCX file. Not meaningful for 24-bit PCX files.
FG_FROMBUFFER	If specified, get the image data from the fg_imagebuf() buffer. If not, get the image data from the PCX file.
FG_KEEPCOLORS	If specified, disable color reduction and remapping. If not, enable color reduction and remapping to avoid using the Windows system colors. Not meaningful for 24-bit PCX files.

Return value

0 = Success

1 = The specified file does not exist

2 = The specified file is not a PCX file

3 = The PCX file cannot be loaded into the active virtual buffer

4 = Error allocating memory

Restrictions

A logical palette must be defined and realized in order to use the palette values stored in the PCX file.

24-bit PCX files can only be loaded into direct color virtual buffers.

See also

fg_imagebuf(), fg_makepcx(), fg_pcxhead(), fg_pcxpal(), fg_pcxrange(), fg_pcxsize(), fg_realize()

Examples

CBdemo, Fade, Fishtank, Image, ImgProc, PCXflip, TMcube, TMcubeX

fg_showppr()

Prototype

```
int fg_showppr (char *filename, int width);
```

```
Function fg_showppr (ByVal filename As String, ByVal width As Long) As Long
```

```
function fg_showppr (filename : string; width : integer) : integer;
```

Description

The **fg_showppr()** legacy function displays a packed pixel run (PPR) file. The image will be positioned so that its lower left corner is at the graphics cursor position in the active virtual buffer.

Parameters

filename specifies the name of the PPR file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

width is the image width in pixels. It must be greater than zero.

Return value

0 = Success

1 = The specified file does not exist

Restrictions

none

Replaced by

BMP and PCX display functions

fg_showspr()

Prototype

```
int fg_showspr (char *filename, int width);  
  
Function fg_showspr (ByVal filename As String, ByVal width As Long) As Long  
  
function fg_showspr (filename : string; width : integer) : integer;
```

Description

The **fg_showspr()** legacy function displays a standard pixel run (SPR) file. The image will be positioned so that its lower left corner is at the graphics cursor position in the active virtual buffer.

Parameters

filename specifies the name of the SPR file. A device and path name may be included as part of the file name. The file name must be terminated by a zero byte.

width is the image width in pixels. It must be greater than zero.

Return value

0 = Success

1 = The specified file does not exist

Restrictions

none

Replaced by

BMP and PCX display functions

fg_stall()

Prototype

```
void fg_stall (int delay);  
Sub fg_stall (ByVal delay As Long)  
procedure fg_stall (delay : integer);
```

Description

The **fg_stall()** function delays a program's execution for a given number of processor-specific delay units. You can use **fg_measure()** to obtain the number of delay units per clock tick for the system being used.

Parameters

delay is the number of delay units to wait.

Return value

none

Restrictions

none

See also

fg_measure(), fg_waitfor()

fg_swchar()

Prototype

```
void fg_swchar (char *string, int n, int justify);
Sub fg_swchar (ByVal string As String, ByVal n As Long, ByVal justify
As Long)
procedure fg_swchar (string : string; n, justify : integer);
```

Description

The **fg_swchar()** function displays a string of software characters in the current color. The string may be left justified, centered, or right justified relative to the graphics cursor.

Parameters

string is the sequence of characters to display. It may contain special operators, as summarized in the following table.

Operator	Meaning
\	Switch to other font
\^	Superscript the next character
\v	Subscript the next character
—	Begin underlining characters until another underscore character is encountered

n is the number of characters in *string*, including any special operator characters.

justify determines how *string* is positioned relative to the current position. If *justify* is negative, *string* is left justified; if it is zero, *string* is centered; if it is positive, *string* is right justified.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setratio(), fg_setsize(), fg_setsizew(), fg_setworld(), fg_swlength(), fg_swtext()

Examples

SWchars

fg_swlength()

Prototype

```
double fg_swlength (char *string, int n);
```

```
Function fg_swlength (ByVal string As String, ByVal n As Long) As Double
```

```
function fg_swlength (string : string; n : integer) : real;
```

Description

The **fg_swlength()** function computes the length of a string of software characters.

Parameters

string is the sequence of characters for which to compute the length. It may contain special operators used by the **fg_swchar()** and **fg_swtext()** functions.

n is the number of characters in *string*, including any special operator characters.

Return value

The length of *string*, in 2D world space units.

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setratio(), fg_setsize(), fg_setsizew(), fg_setworld(), fg_swchar(), fg_swtext()

fg_swtext()

Prototype

```
void fg_swtext (char *string, int n, int justify);
Sub fg_swtext (ByVal string As String, ByVal n As Long, ByVal justify
As Long)
procedure fg_swtext (string : string; n, justify : integer);
```

Description

The **fg_swtext()** function is a scaled down version of **fg_swchar()**. It does not include the alternate font character definitions and thus requires less memory than **fg_swchar()**.

Parameters

string is the sequence of characters to display. It may contain special operators, as summarized in the following table.

Operator	Meaning
\^	Superscript the next character
\v	Subscript the next character
—	Begin underlining characters until another underscore character is encountered

n is the number of characters in *string*, including any special operator characters.

justify determines how *string* is positioned relative to the current position. If *justify* is negative, *string* is left justified; if it is zero, *string* is centered; if it is positive, *string* is right justified.

Return value

none

Restrictions

Before using this function, you must use **fg_initw()** and **fg_setworld()** to establish a 2D world space coordinate system.

See also

fg_initw(), fg_setangle(), fg_setratio(), fg_setsize(), fg_setsizew(), fg_setworld(), fg_swchar(), fg_swlength()

fg_tcdefine()

Prototype

```
void fg_tcdefine (int index, int attribute);  
Sub fg_tcdefine (ByVal index As Long, ByVal attribute As Long)  
procedure fg_tcdefine (index, attribute : integer);
```

Description

The **fg_tcdefine()** function defines the transparency attribute of a color for use with **fg_vbtccopy()**.

Parameters

index is the color being defined, between 0 and 255.

attribute is the transparency attribute for the color. If the attribute is 0, the specified color will be opaque (non-transparent). If it is any other value, **fg_vbtccopy()** will treat the color as transparent.

Return value

none

Restrictions

none

See also

fg_vbtccopy(), fg_vbtzcopy()

Examples

VBdemo

fg_tcmask()

Prototype

```
void fg_tcmask (int mask);  
Sub fg_tcmask (ByVal mask As Long)  
procedure fg_tcmask (mask : integer);
```

Description

The **fg_tcmask()** legacy function defines which of the first 16 color values **fg_tcxfer()** and **fg_vbtccopy()** will consider transparent. You must use **fg_tcdefine()** to control the transparency of colors 17 to 255.

Parameters

mask is a 16-bit mask, where each bit indicates whether or not the corresponding color value is transparent. For example, if bit 0 (the rightmost bit) is 1, then color 0 will be transparent. If bit 0 is 0, color 0 will not be transparent.

Return value

none

Restrictions

none

Replaced by

fg_tcdefine()

fg_tcxfer()

Prototype

```
void fg_tcxfer (int minx, int maxx, int miny, int maxy, int newx, int
newy, int source, int dest);

Sub fg_tcxfer (ByVal minx As Long, ByVal maxx As Long, ByVal miny As
Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long, ByVal
source As Long, ByVal dest As Long)

procedure fg_tcxfer (minx, maxx, miny, maxy, newx, newy, source, dest :
integer);
```

Description

The **fg_tcxfer()** legacy function is identical to **fg_vbtccopy()**. It copies a rectangular region from one virtual buffer to another, or to a non-overlapping region in the same virtual buffer. It excludes any pixels whose color is transparent. As with Fastgraph's other block transfer functions, no clipping is performed. The **fg_tcdefine()** and **fg_tcmask()** functions define which colors are transparent.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge.

newy is the y coordinate of the destination region's bottom edge.

source is the handle of the virtual buffer containing the source region.

dest is the handle of the virtual buffer for the destination region.

Return value

none

Restrictions

If *source* and *dest* reference the same virtual buffer, the source and destination regions must not overlap.

Replaced by

fg_vbtccopy()

fg_texmap()

Prototype

```
void fg_texmap (int *vertex_array, int *uv_array, int n);

Sub fg_texmap (vertex_array() As Long, uv_array() As Long, ByVal n As Long)

procedure fg_texmap (var vertex_array, uv_array : integer; n : integer);
```

Description

The **fg_texmap()** function draws a projected linear texture-mapped convex polygon in screen space, with 2D clipping and automatic backface removal. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each vertex in the destination polygon. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth. The vertices must be stored in clockwise order, meaning you would travel clockwise along the polygon edge to go from one vertex to the next.

uv_array is the name of the array containing the (u,v) texture map coordinates for each (x,y) coordinate pair in *vertex_array*. The first two *uv_array* elements represent the (x,y) values at the first vertex in *vertex_array*, the next two *uv_array* elements are for the second vertex, and so forth.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_texmap()**, or if the vertices are not stored in clockwise order, only a portion of the polygon will be filled.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_inside(), fg_polyoff(), fg_texmapp(), fg_texmappz(), fg_texmapz(), fg_tmdefine(), fg_tmselect(), fg_tmtransparency(),

fg_texmapp()

Prototype

```
void fg_texmapp (int *vertex_array, int *uv_array, double *xyz_array,  
int n);  
  
Sub fg_texmapp (vertex_array() As Long, uv_array() As Long, xyz_array()  
As Double, ByVal n As Long)  
  
procedure fg_texmapp (var vertex_array, uv_array : integer; var  
xyz_array : double; n : integer);
```

Description

The **fg_texmapp()** function draws a projected perspective corrected texture-mapped convex polygon in screen space, with 2D clipping and automatic backface removal. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth. The vertices must be stored in clockwise order, meaning you would travel clockwise along the polygon edge to go from one vertex to the next.

uv_array is the name of the array containing the (u,v) texture map coordinates for each (x,y) coordinate pair in *vertex_array*. The first two *uv_array* elements represent the (x,y) values at the first vertex in *vertex_array*, the next two *uv_array* elements are for the second vertex, and so forth.

xyz_array is the name of the array containing the 3D (x,y,z) coordinates for each (x,y) coordinate pair in *vertex_array*. The first three *xyz_array* elements represent the (x,y,z) values at the first vertex in *vertex_array*, the next three *xyz_array* elements are for the second vertex, and so forth. Only the z coordinates are meaningful in this function.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_texmapp()**, or if the vertices are not stored in clockwise order, only a portion of the polygon will be filled.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_inside(), fg_polyoff(), fg_texmap(),
fg_texmappz(), fg_texmapz(), fg_tmdefine(), fg_tmselect(), fg_tmspan(), fg_tmtransparency()

fg_texmappz()

Prototype

```
void fg_texmappz (int *vertex_array, int *uv_array, double *xyz_array,
int n);

Sub fg_texmappz (vertex_array() As Long, uv_array() As Long,
xyz_array() As Double, ByVal n As Long)

procedure fg_texmappz (var vertex_array, uv_array : integer; var
xyz_array : double; n : integer);
```

Description

The **fg_texmappz()** function draws a projected z-buffered perspective corrected texture-mapped convex polygon in screen space, with 2D clipping. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

uv_array is the name of the array containing the (u,v) texture map coordinates for each (x,y) coordinate pair in *vertex_array*. The first two *uv_array* elements represent the (x,y) values at the first vertex in *vertex_array*, the next two *uv_array* elements are for the second vertex, and so forth.

xyz_array is the name of the array containing the 3D (x,y,z) coordinates for each (x,y) coordinate pair in *vertex_array*. The first three *xyz_array* elements represent the (x,y,z) values at the first vertex in *vertex_array*, the next three *xyz_array* elements are for the second vertex, and so forth. Only the z coordinates are meaningful in this function.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_texmappz()**, only a portion of the polygon will be filled.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_inside(), fg_polyoff(), fg_texmap(), fg_texmapp(), fg_texmapz(), fg_tmdefine(), fg_tmselect(), fg_tmspan(), fg_tmtransparency(), fg_zbopen()

fg_texmapz()

Prototype

```
void fg_texmapz (int *vertex_array, int *uv_array, double *xyz_array,
int n);

Sub fg_texmapz (vertex_array() As Long, uv_array() As Long, xyz_array()
As Double, ByVal n As Long)

procedure fg_texmapz (var vertex_array, uv_array : integer; var
xyz_array : double; n : integer);
```

Description

The **fg_texmapz()** function draws a projected z-buffered linear texture-mapped convex polygon in screen space, with 2D clipping. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

vertex_array is the name of the array containing the (x,y) coordinate pairs of each polygon vertex. The first array element is the x component of the first vertex, the second element is the y component of the first vertex, the third element is the x component of the second vertex, and so forth.

uv_array is the name of the array containing the (u,v) texture map coordinates for each (x,y) coordinate pair in *vertex_array*. The first two *uv_array* elements represent the (x,y) values at the first vertex in *vertex_array*, the next two *uv_array* elements are for the second vertex, and so forth.

xyz_array is the name of the array containing the 3D (x,y,z) coordinates for each (x,y) coordinate pair in *vertex_array*. The first three *xyz_array* elements represent the (x,y,z) values at the first vertex in *vertex_array*, the next three *xyz_array* elements are for the second vertex, and so forth. Only the z coordinates are meaningful in this function.

n is the number of vertices in each of the above arrays.

Return value

none

Restrictions

If you attempt to fill a non-convex polygon with **fg_texmapz()**, only a portion of the polygon will be filled.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_inside(), fg_polyoff(), fg_texmap(), fg_texmapp(), fg_texmappz(), fg_tmdefine(), fg_tmselect(), fg_tmtransparency(), fg_zbopen()

fg_text()

Prototype

```
void fg_text (char *string, int n);  
Sub fg_text (ByVal string As String, ByVal n As Long)  
procedure fg_text (string : string; n : integer);
```

Description

The **fg_text()** legacy function displays a character string, starting at the text cursor position, using the current color. The characters are clipped at the client area or virtual buffer edges, not the area defined with **fg_setclip()**. On return, the text cursor is positioned just to the right of the last character displayed.

By default, **fg_text()** displays strings directly in the window's client area, but you can redirect strings to the active virtual buffer with **fg_fontdc()**.

Parameters

string is the sequence of characters to display.

n is the number of characters to display from *string*.

Return value

You cannot direct strings to virtual buffers created with **fg_vbdefine()**.

Restrictions

none

Replaced by

fg_print()

fg_texture()

Prototype

```
void fg_texture (void *texture, int width);  
Sub fg_texture (texture() As Any, ByVal width As Long)  
procedure fg_texture (var texture; width : integer);
```

Description

The **fg_texture()** legacy function defines the texture map applied to the destination polygon in Fastgraph's texture-mapped polygon drawing functions.

Parameters

texture is the name of the array containing the texture map applied to the destination polygon. For 256-color virtual buffers, the *texture* array is a 256-color bitmap, but stored top-down. For direct color virtual buffers, it is a direct color bitmap, but again stored top-down.

width is the *texture* array width in pixels.

Return value

none

Restrictions

none

Replaced by

fg_tmdefine(), fg_tmselect()

fg_tmdefine()

Prototype

```
int fg_tmdefine (void *texture, int width, int height);  
  
Function fg_tmdefine (texture As Any, ByVal width As Long, ByVal height  
As Long) As Long  
  
function fg_tmdefine (var texture; width, height : integer) : integer;
```

Description

The **fg_tmdefine()** function assigns a handle to a texture map, and if using Direct3D, loads the texture map into a DirectDraw texture surface.

Parameters

texture is an array containing the texture map. For 256-color virtual buffers, *texture* is a 256-color bitmap, but stored top-down. For direct color virtual buffers, it is a direct color bitmap, but again stored top-down.

width is the *texture* array width in pixels.

height is the *texture* array height in pixels.

Return value

If successful, **fg_tmdefine()** returns a handle by which the texture map is referenced (greater than or equal to 0). If unsuccessful, possible return codes are -1 (maximum number of textures exceeded) or -2 (cannot make texture available to Direct3D).

Restrictions

If using Direct3D, the texture map width and height must be powers of two.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_tmfree(), fg_tminit(), fg_tmselect()

Examples

TMcube, TMcubeX

fg_tmfree()

Prototype

```
void fg_tmfree (int handle);  
Sub fg_tmfree (ByVal handle As Long)  
procedure fg_tmfree (handle : integer);
```

Description

The **fg_tmfree()** function releases a texture handle, and if using Direct3D, releases the DirectDraw surface associated with the texture map.

Parameters

handle is the texture handle to release. If *handle* is negative, **fg_tmfree()** will release all texture handles.

Return value

none

Restrictions

none

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_tmdefine(), fg_tminit(), fg_tmselect()

Examples

TMcube, TMcubeX

fg_tminit()

Prototype

```
int fg_tminit (int count);  
Function fg_tminit (ByVal count As Long) As Long  
function fg_tminit (count : integer) : integer;
```

Description

The **fg_tminit()** function initializes Fastgraph's texture map manager.

Parameters

count is the maximum number of texture maps that will be in use at the same time.

Return value

0 = Success
-1 = Error allocating memory

Restrictions

none

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_tmdefine(), fg_tmfree(), fg_tmselect()

Examples

TMcube, TMcubeX

fg_tmselect()

Prototype

```
void fg_tmselect (int handle);  
Sub fg_tmselect (ByVal handle As Long)  
procedure fg_tmselect (handle : integer);
```

Description

The **fg_tmselect()** function makes the specified texture map the active texture map.

Parameters

handle is the texture map handle returned by **fg_tmdefine()**.

Return value

none

Restrictions

none

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject(), fg_tmdefine(), fg_tmfree(), fg_tminit()

Examples

TMcube, TMcubeX

fg_tmspan()

Prototype

```
void fg_tmspan (int npixels);  
Sub fg_tmspan (ByVal npixels As Long)  
procedure fg_tmspan (npixels : integer);
```

Description

The **fg_tmspan()** function defines the span size in pixels for perspective texture mapping. The span size is the pixel interval at which the perspective texture mapping functions calculate the true u and v texture coordinates when drawing horizontal polygon rows. Smaller span sizes result in texture mapping that is more perspective correct, but larger span sizes are faster. The default span size is 32 pixels.

Parameters

npixels defines the span size in pixels. If *npixels* is 1, the perspective texture mapping functions will draw perspective correct texture-mapped polygons. If *npixels* is greater than 1, these functions will draw perspective corrected texture-mapped polygons.

Return value

none

Restrictions

This function has no effect when using Direct3D hardware acceleration or Direct3D software rendering.

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject()

fg_tmtransparency()

Prototype

```
void fg_tmtransparency (int state);  
Sub fg_tmtransparency (ByVal state As Long)  
procedure fg_tmtransparency (state : integer);
```

Description

The **fg_tmtransparency()** function defines the transparency state of zero-value pixels in texture maps.

Parameters

state defines the transparency state of zero-value pixels in texture maps. If *state* is zero, zero-value pixels will not be transparent (this is the default behavior). If *state* is any other value, zero-value pixels will be transparent.

Return value

none

Restrictions

none

See also

fg_3Dtexturemap(), fg_3Dtexturemapobject()

fg_transdcb()

Prototype

```
void fg_transdcb (void *source, void *dest, int source_depth, int
dest_depth, int size);

Sub fg_transdcb (source() As Any, dest() As Any, ByVal source_depth As
Long, ByVal dest_depth As Long, ByVal size As Long)

procedure fg_transdcb (var source, dest; source_depth, dest_depth, size
: integer);
```

Description

The **fg_transdcb()** function translates a direct color bitmap to another color depth.

Parameters

source is the name of the array containing the direct color bitmap to be converted.

dest is the name of the array that will receive the resulting converted bitmap.

source_depth is the color depth of the source bitmap. It must be 15, 16, or 24.

dest_depth is the color depth of the dest bitmap. It must be 15, 16, or 24.

size is the size of each direct color bitmap in pixels.

Return value

none

Restrictions

If *source_depth* and *dest_depth* are not 15, 16, or 24, **fg_transdcb()** does nothing.

If translating a high color bitmap to true color, the *source* and *dest* bitmaps must be different bitmaps.

See also

fg_getdepth(), fg_gethcbpp()

fg_transfer()

Prototype

```
void fg_transfer (int minx, int maxx, int miny, int maxy, int newx, int newy, int source, int dest);
```

```
Sub fg_transfer (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long, ByVal source As Long, ByVal dest As Long)
```

```
procedure fg_transfer (minx, maxx, miny, maxy, newx, newy, source, dest : integer);
```

Description

The **fg_transfer()** legacy function copies a rectangular region from one virtual buffer to another, or to a non-overlapping position within the same virtual buffer. It is equivalent to **fg_vbcopy()**.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge.

newy is the y coordinate of the destination region's bottom edge.

source is the handle for the virtual buffer containing the source region.

dest is the handle for the virtual buffer containing the destination region.

Return value

none

Restrictions

If *source* and *dest* reference the same virtual buffer, the source and destination regions must not overlap.

When using DirectX, the source and destination virtual buffers must not be locked.

Replaced by

fg_vbcopy()

fg_trig()

Prototype

```
void fg_trig (int angle, double *cosine, double *sine);  
Sub fg_trig (ByVal angle As Long, cosine As Double, sine As Double)  
procedure fg_trig (angle : integer; var cosine, sine : double);
```

Description

The **fg_trig()** function returns the floating point cosine and sine of a given angle. This function is called internally by Fastgraph's 3D functions and is not usually called directly by applications.

Parameters

angle is the angle, expressed in tenths of degrees.

cosine receives the cosine of the angle.

sine receives the sine of the angle.

Return value

none

Restrictions

none

fg_unmaprgb()

Prototype

```
void fg_unmaprgb (int color, int *red, int *green, int *blue);  
Sub fg_unmaprgb (ByVal color As Long, red As Long, green As Long, blue  
As Long)  
procedure fg_unmaprgb (color : integer; var red, green, blue :  
integer);
```

Description

The **fg_unmaprgb()** function extracts the red, green, and blue color components from an encoded 16-bit or 32-bit color value.

Parameters

color is the encoded color value. For high color virtual buffers, *color* is a 16-bit encoded RGB value. For true color virtual buffers, *color* is a 32-bit value encoded as four 8-bit color components (xRGB).

red receives the encoded color value's red component, between 0 and 255.

green receives the encoded color value's green component, between 0 and 255.

blue receives the encoded color value's blue component, between 0 and 255.

Return value

none

Restrictions

This function is meaningful only with direct color virtual buffers.

See also

fg_maprgb(), fg_setrgb()

fg_unpack()

Prototype

```
void fg_unpack (void *source, void *dest, int size);  
Sub fg_unpack (source() As Any, dest() As Any, ByVal size As Long)  
procedure fg_unpack (var source, dest; size : integer);
```

Description

The **fg_unpack()** legacy function converts a 16-color bitmap to a 256-color bitmap.

Parameters

source is the name of the array containing the 16-color bitmap to convert.

dest is the name of the array that will receive the converted 256-color bitmap.

size is the size of the *source* array in bytes.

Return value

none

Restrictions

none

Replaced by

256-color bitmap functions

fg_vb2clip()

Prototype

```
int fg_vb2clip (int minx, int maxx, int miny, int maxy);
```

```
Function fg_vb2clip (ByVal minx As Long, ByVal maxx As Long, ByVal miny  
As Long, ByVal maxy As Long) As Long
```

```
function fg_vb2clip (minx, maxx, miny, maxy : integer) : integer;
```

Description

The **fg_vb2clip()** function copies a rectangular region from the active virtual buffer to the Windows clipboard. A DIB is constructed from the specified region, and the DIB along with its palette data are written to the clipboard. The region's extremes are expressed in screen space units.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

Return value

0 = Success

-1 = Another application has control of the clipboard

-2 = There is not enough free global memory to create the DIB

Restrictions

none

See also

fg_clip2vb(), fg_vbpaste()

Examples

CBdemo

fg_vbaddr()

Prototype

```
long fg_vbaddr (int handle);  
Function fg_vbaddr (ByVal handle As Long) As Long  
function fg_vbaddr (handle : integer) : pointer;
```

Description

For the native libraries, **fg_vbaddr()** returns the address of the specified virtual buffer. For the DirectX libraries, it returns a pointer to the DirectDrawSurface object for the specified virtual buffer. Use **fg_ddlock()** to obtain the virtual buffer address when using the DirectX libraries.

Parameters

handle is the virtual buffer handle.

Return value

The address of, or a pointer to the DirectDrawSurface object for, the specified virtual buffer.

Restrictions

If *handle* does not reference a valid virtual buffer handle, the return value will be undefined.

See also

fg_ddlock(), fg_getline(), fg_vbopen()

Examples

SetupD3D

fg_vballoc()

Prototype

```
int fg_vballoc (int width, int height);
```

```
Function fg_vballoc (ByVal width As Long, ByVal height As Long) As Long
```

```
function fg_vballoc (width, height : integer) : integer;
```

Description

The **fg_vballoc()** function creates a virtual buffer of the specified size. The memory for the virtual buffer is allocated automatically. Virtual buffers are usually created in the WM_CREATE message handler.

Parameters

width is the virtual buffer width in pixels. If necessary, the width is extended to a multiple of four bytes.

height is the virtual buffer height in pixels.

Return value

If successful, **fg_vballoc()** returns a handle by which the virtual buffer is referenced (between 0 and 255). If unsuccessful, possible return codes are -1 (virtual buffer table full) or -2 (cannot allocate memory for the virtual buffer).

Restrictions

none

See also

fg_vbdefine(), fg_vbdepth(), fg_vbfree(), fg_vbinit(), fg_vbopen()

Examples

Nearly all the example programs use this function.

fg_vbclose()

Prototype

```
void fg_vbclose (void);  
Sub fg_vbclose ()  
procedure fg_vbclose;
```

Description

The **fg_vbclose()** function closes the active virtual buffer. It is usually called in the WM_DESTROY message handler.

Parameters

none

Return value

none

Restrictions

none

See also

fg_vbopen()

Examples

All the example programs use this function.

fg_vbcolors()

Prototype

```
void fg_vbcolors (void);  
Sub fg_vbcolors ()  
procedure fg_vbcolors;
```

Description

The **fg_vbcolors()** function copies the colors from the current logical palette into the active virtual buffer's color table. It is usually called immediately after opening each 256-color virtual buffer for the first time (you do not need to use **fg_vbcolors()** with direct color virtual buffers).

Parameters

none

Return value

none

Restrictions

none

See also

fg_defpal(), fg_logpal(), fg_vbopen()

Examples

Nearly all the example programs use this function.

fg_vbcopy()

Prototype

```
void fg_vbcopy (int minx, int maxx, int miny, int maxy, int newx, int
newy, int source, int dest);

Sub fg_vbcopy (ByVal minx As Long, ByVal maxx As Long, ByVal miny As
Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long, ByVal
source As Long, ByVal dest As Long)

procedure fg_vbcopy (minx, maxx, miny, maxy, newx, newy, source, dest :
integer);
```

Description

The **fg_vbcopy()** function copies a rectangular region from one virtual buffer to another, or to a non-overlapping position within the same virtual buffer.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge.

newy is the y coordinate of the destination region's bottom edge.

source is the handle for the virtual buffer containing the source region.

dest is the handle for the virtual buffer containing the destination region.

Return value

none

Restrictions

If *source* and *dest* reference the same virtual buffer, the source and destination regions must not overlap.

When using DirectX, the source and destination virtual buffers must not be locked.

See also

fg_vbopen(), fg_vbtccopy(), fg_vbtzcopy()

Examples

VBdemo

fg_vbdefine()

Prototype

```
int fg_vbdefine (void *buffer, int width, int height);
```

Function fg_vbdefine (buffer() As Any, ByVal width As Long, ByVal height As Long) As Long

```
function fg_vbdefine (buffer : pointer; width, height : integer) : integer;
```

Description

The **fg_vbdefine()** function creates a virtual buffer with the specified dimensions from previously allocated memory. Virtual buffers are usually created in the WM_CREATE message handler. Use **fg_vbsize()** to determine the amount of memory needed for the virtual buffer. Refer to Chapter 10 of the *Fastgraph 6.0 User's Guide* for details about allocating blocks of memory suitable for virtual buffers.

Parameters

buffer is the address of the virtual buffer.

width is the virtual buffer width in pixels. If necessary, the width is extended to a multiple of four bytes.

height is the virtual buffer height in pixels.

Return value

If successful, **fg_vbdefine()** returns a handle by which the virtual buffer is referenced (between 0 and 255). If unsuccessful, the function returns -1.

Restrictions

This function cannot be used to create virtual buffers when using Fastgraph's DirectX libraries (use **fg_vballoc()** instead).

See also

fg_vballoc(), fg_vbdepth(), fg_vbinit(), fg_vbopen(), fg_vbsize(), fg_vbundef()

fg_vbdepth()

Prototype

```
void fg_vbdepth (int bpp);  
Sub fg_vbdepth (ByVal bpp As Long)  
procedure fg_vbdepth (bpp : integer);
```

Description

The **fg_vbdepth()** function defines the default virtual buffer color depth in bits per pixel.

Parameters

bpp is the new default virtual buffer color depth in bits per pixel. It must be either 8, 16, 24, or 32.

Return value

none

Restrictions

none

See also

fg_vballoc(), fg_vbdefine(), fg_vbinit()

Examples

Blend, Colors, Cube, Dcb, FirstDD, Image, ImgProc, TMcube, TMcubeX, Tunnel

fg_vbfin()

Prototype

```
void fg_vbfin (void);  
Sub fg_vbfin ()  
procedure fg_vbfin;
```

Description

The **fg_vbfin()** function terminates virtual buffer processing. It is usually the last Fastgraph function called in the WM_DESTROY message handler.

Parameters

none

Return value

none

Restrictions

Before calling this function, all virtual buffers created with **fg_vballoc()** must be released with **fg_vbfree()**. In addition, any virtual buffer handles set up through **fg_vbdefine()** must be released with **fg_vbundef()**.

See also

fg_vbfree(), fg_vbinit(), fg_vbundef()

Examples

All the example programs use this function.

fg_vbfree()

Prototype

```
void fg_vbfree (int handle);  
Sub fg_vbfree (ByVal handle As Long)  
procedure fg_vbfree (handle : integer);
```

Description

The **fg_vbfree()** function releases a virtual buffer's handle and frees the memory allocated to the virtual buffer. It is usually called in the WM_DESTROY message handler.

Parameters

handle is the handle that references the virtual buffer to free. Its value must be a valid virtual buffer handle and must not reference the active virtual buffer. If *handle* is negative and no virtual buffer is active, **fg_vbfree()** frees all virtual buffers.

Return value

none

Restrictions

You should use **fg_vbfree()** only with virtual buffers created with **fg_vballoc()**.

This function has no effect if *handle* references the active virtual buffer.

See also

fg_vballoc(), fg_vbfin()

Examples

Nearly all the example programs use this function.

fg_vbhandle()

Prototype

```
int fg_vbhandle (void);  
Function fg_vbhandle () As Long  
function fg_vbhandle : integer;
```

Description

The **fg_vbhandle()** function returns the handle of the active virtual buffer.

Parameters

none

Return value

The active virtual buffer handle, between 0 and 255. If no virtual buffer is active, the return value is -1.

Restrictions

none

See also

fg_vbopen()

fg_vbinit()

Prototype

```
int fg_vbinit (void);  
Function fg_vbinit () As Long  
function fg_vbinit : integer;
```

Description

The **fg_vbinit()** function initializes Fastgraph's virtual buffer environment. This function must be called once, before any other functions that reference virtual buffers, usually in the WM_CREATE message handler.

In Fastgraph's DirectX libraries, **fg_vbinit()** also initializes DirectDraw and Direct3D.

Parameters

none

Return value

For Fastgraph's native libraries, the return value will always be zero.

For Fastgraph's DirectX libraries, the possible return values are:

- 1 = DirectX not available
- 0 = Using DirectX with Fastgraph's 3D rendering
- 1 = Using DirectX with Direct3D software rendering
- 2 = Using DirectX with Direct3D hardware acceleration

Restrictions

none

See also

fg_ddsetup(), fg_vballoc(), fg_vbdefine(), fg_vbdepth(), fg_vbfin(), fg_vbfree(), fg_vbopen()

Examples

All the example programs use this function.

fg_vbopen()

Prototype

```
int fg_vbopen (int handle);  
Function fg_vbopen (ByVal handle As Long) As Long  
function fg_vbopen (handle : integer) : integer;
```

Description

The **fg_vbopen()** function makes an existing virtual buffer the active virtual buffer. If another virtual buffer is open when you call **fg_vbopen()**, that virtual buffer is first closed.

Parameters

handle is the handle of the desired virtual buffer, as returned by **fg_vballoc()** or **fg_vbdefine()**. Its value must be a valid virtual buffer handle.

Return value

- 0 = Virtual buffer opened successfully
- 1 = Invalid virtual buffer handle
- 2 = No virtual buffer yet defined for specified handle

Restrictions

none

See also

fg_vballoc(), fg_vbclose(), fg_vbdefine(), fg_vbinit()

Examples

All the example programs use this function.

fg_vbpaste()

Prototype

```
void fg_vbpaste (int minx, int maxx, int miny, int maxy, int newx, int newy);
```

```
Sub fg_vbpaste (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long)
```

```
procedure fg_vbpaste (minx, maxx, miny, maxy, newx, newy : integer);
```

Description

The **fg_vbpaste()** function copies a rectangular region from the active virtual buffer to the window's client area. The WM_PAINT message handler usually calls **fg_vbpaste()** or **fg_vbscale()**.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge, expressed in client units.

newy is the y coordinate of the destination region's bottom edge, expressed in client units.

Return value

none

Restrictions

When using DirectX, the active virtual buffer must not be locked.

See also

fg_vb2clip(), fg_vbscale()

Examples

Display, FrameDD, FullScr, Graphics, KBdemo, Panner, SWchars, TMcubeX

fg_vbprint()

Prototype

```
void fg_vbprint (int minx, int maxx, int miny, int maxy, int xmin, int xmax, int ymin, int ymax, int units);
```

```
Sub fg_vbprint (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long, ByVal xmin As Long, ByVal xmax As Long, ByVal ymin As Long, ByVal ymax As Long, ByVal units As Long)
```

```
procedure fg_vbprint (minx, maxx, miny, maxy, xmin, xmax, ymin, ymax, units : integer);
```

Description

The **fg_vbprint()** function prints a rectangular region from the active virtual buffer on the default printer, scaling the printed image as requested. Source coordinates are expressed in screen space, while destination coordinates are expressed in the specified units.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

xmin is the x coordinate of the destination region's left edge.

xmax is the x coordinate of the destination region's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the y coordinate of the destination region's top edge.

ymax is the y coordinate of the destination region's bottom edge. It must be greater than or equal to the value of *ymin*.

units specifies the units of the destination coordinates, as shown here:

Value	Meaning	Value	Meaning
0	device units (no translation)	4	millimeters
1	percentage	5	0.1 millimeters
2	0.01 inches	6	0.01 millimeters
3	0.001 inches		

Return value

none

Restrictions

none

See also

fg_printer()

Examples

Prdemo

fg_vbscale()

Prototype

```
void fg_vbscale (int minx, int maxx, int miny, int maxy, int xmin, int
xmax, int ymin, int ymax);

Sub fg_vbscale (ByVal minx As Long, ByVal maxx As Long, ByVal miny As
Long, ByVal maxy As Long, ByVal xmin As Long, ByVal xmax As Long, ByVal
ymin As Long, ByVal ymax As Long)

procedure fg_vbprint (minx, maxx, miny, maxy, xmin, xmax, ymin, ymax :
integer);
```

Description

The **fg_vbscale()** function copies a rectangular region from the active virtual buffer to the window's client area, scaling the image to fit the client area as requested. Source coordinates are expressed in screen space, while destination coordinates are expressed as client coordinates. The WM_PAINT message handler usually calls **fg_vbpaste()** or **fg_vbscale()**.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

xmin is the x coordinate of the destination region's left edge.

xmax is the x coordinate of the destination region's right edge. It must be greater than or equal to the value of *xmin*.

ymin is the y coordinate of the destination region's top edge.

ymax is the y coordinate of the destination region's bottom edge. It must be greater than or equal to the value of *ymin*.

Return value

none

Restrictions

When using DirectX, the active virtual buffer must not be locked.

See also

fg_scale(), fg_vbpaste()

Examples

Nearly all the example programs use this function.

fg_vbsize()

Prototype

```
long fg_vbsize (int width, int height);
```

```
Function fg_vbsize (ByVal width As Long, ByVal height As Long) As Long
```

```
function fg_vbsize (width, height : integer) : longint;
```

Description

The **fg_vbsize()** function returns the number of bytes required for a virtual buffer of specified dimensions, using the current color depth.

Parameters

width specifies the virtual buffer width in pixels.

height specifies the virtual buffer height in pixels.

Return value

The number of bytes required for a virtual buffer of the specified size and current color depth.

Restrictions

none

See also

fg_vbdefine(), fg_vbdepth(), fg_vbinit()

fg_vbtccopy()

Prototype

```
void fg_vbtccopy (int minx, int maxx, int miny, int maxy, int newx, int
newy, int source, int dest);
```

```
Sub fg_vbtccopy (ByVal minx As Long, ByVal maxx As Long, ByVal miny As
Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long, ByVal
source As Long, ByVal dest As Long)
```

```
procedure fg_vbtccopy (minx, maxx, miny, maxy, newx, newy, source, dest
: integer);
```

Description

The **fg_vbtccopy()** function copies a rectangular region from one virtual buffer to another, or to a non-overlapping region in the same virtual buffer. It excludes any pixels whose color is transparent. As with Fastgraph's other block transfer functions, no clipping is performed. The **fg_tcdefine()** function defines which colors are transparent.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge.

newy is the y coordinate of the destination region's bottom edge.

source is the handle of the virtual buffer containing the source region.

dest is the handle of the virtual buffer for the destination region.

Return value

none

Restrictions

If *source* and *dest* reference the same virtual buffer, the source and destination regions must not overlap.

See also

fg_tcdefine(), fg_vbcopy(), fg_vbopen(), fg_vbtzcopy()

Examples

VBdemo

fg_vbtzcopy()

Prototype

```
void fg_vbtzcopy (int minx, int maxx, int miny, int maxy, int newx, int newy, int source, int dest);
```

```
Sub fg_vbtzcopy (ByVal minx As Long, ByVal maxx As Long, ByVal miny As Long, ByVal maxy As Long, ByVal newx As Long, ByVal newy As Long, ByVal source As Long, ByVal dest As Long)
```

```
procedure fg_vbtzcopy (minx, maxx, miny, maxy, newx, newy, source, dest : integer);
```

Description

The **fg_vbtzcopy()** function copies a rectangular region from one virtual buffer to another, or to a non-overlapping region in the same virtual buffer, excluding any pixels whose value is zero. As with Fastgraph's other block transfer routines, no clipping is performed.

Parameters

minx is the x coordinate of the source region's left edge.

maxx is the x coordinate of the source region's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the source region's top edge.

maxy is the y coordinate of the source region's bottom edge. It must be greater than or equal to the value of *miny*.

newx is the x coordinate of the destination region's left edge.

newy is the y coordinate of the destination region's bottom edge.

source is the handle for the virtual buffer containing the source region.

dest is the handle for the virtual buffer for the destination region.

Return value

none

Restrictions

If *source* and *dest* reference the same virtual buffer, the source and destination regions must not overlap.

When using DirectX, the source and destination virtual buffers must not be locked.

See also

fg_vbcopy(), fg_vbopen(), fg_vbtccopy()

fg_vbundef()

Prototype

```
void fg_vbundef (int handle);  
Sub fg_vbundef (ByVal handle As Long)  
procedure fg_vbundef (handle : integer);
```

Description

The **fg_vbundef()** function releases the handle associated with a virtual buffer.

Parameters

handle is the virtual buffer handle to release. Its value must be a valid virtual buffer handle and must not reference the active virtual buffer.

Return value

none

Restrictions

This function has no effect if *handle* references the active virtual buffer.

See also

fg_vbdefine(), fg_vbfin()

fg_version()

Prototype

```
void fg_version (int *major, int *minor);  
Sub fg_version (major As Long, minor As Long)  
procedure fg_version (var major, minor : integer);
```

Description

The **fg_version()** function returns the major and minor version numbers for your copy of Fastgraph for Windows. For example, if you are using version 6.00, the major version number is 6 and the minor version number is 0.

Parameters

major receives the major version number.

minor receives the minor version number, expressed in hundredths.

Return value

none

Restrictions

none

fg_view3d()

Prototype

```
void fg_view3d (int minx, int maxx, int miny, int maxy, long ratio);  
Sub fg_view3d (ByVal minx As Long, ByVal maxx As Long, ByVal miny As  
Long, ByVal maxy As Long, ByVal ratio As Long)  
procedure fg_view3d (minx, maxx, miny, maxy : integer; ratio :  
longint);
```

Description

The **fg_view3d()** legacy function defines the 3D viewport in screen space and the corresponding projection ratio. You must set up a 3D viewport before using **fg_project()**.

Parameters

minx is the x coordinate of the viewport's left edge.

maxx is the x coordinate of the viewport's right edge. It must be greater than or equal to the value of *minx*.

miny is the y coordinate of the viewport's top edge.

maxy is the y coordinate of the viewport's bottom edge. It must be greater than or equal to the value of *miny*.

ratio is the fixed point projection ratio. It must be greater than zero.

Return value

none

Restrictions

none

Replaced by

Floating point 3D geometry system

fg_waitfor()

Prototype

```
void fg_waitfor (int ticks);  
Sub fg_waitfor (ByVal ticks As Long)  
procedure fg_waitfor (ticks : integer);
```

Description

The **fg_waitfor()** function delays a program's execution for a given number of clock ticks. There are 18.2 clock ticks per second, regardless of the system's processor speed.

Parameters

ticks is the number of clock ticks to wait.

Return value

none

Restrictions

none

See also

fg_stall()

fg_where()

Prototype

```
void fg_where (int *row, int *column);  
Sub fg_where (row As Long, column As Long)  
procedure fg_where (row, column : integer);
```

Description

The **fg_where()** legacy function retrieves the text cursor position.

Parameters

row receives the text cursor's current row number, between 0 and one less than the number of character rows available.

column receives text cursor's current column number, between 0 and one less than the number of character columns available.

Return value

none

Restrictions

none

Replaced by

Screen space

fg_xalpha()

Prototype

```
int fg_xalpha (int ix);  
Function fg_xalpha (ByVal ix As Long) As Long  
function fg_xalpha (ix : integer) : integer;
```

Description

The **fg_xalpha()** legacy function translates a screen space x coordinate to the character space column containing that coordinate.

Parameters

ix is the screen space coordinate to translate.

Return value

The character space column containing the screen space coordinate *ix*.

Restrictions

none

Replaced by

Screen space

fg_xclient()

Prototype

```
int fg_xclient (int ix);  
Function fg_xclient (ByVal ix As Long) As Long  
function fg_xclient (ix : integer) : integer;
```

Description

The **fg_xclient()** function translates a screen space x coordinate to the corresponding client area x coordinate.

Parameters

ix is the screen space coordinate to translate.

Return value

The client x coordinate corresponding to the screen space coordinate *ix*.

Restrictions

none

See also

fg_xvb(), fg_yclient(), fg_yvb()

Examples

Fontdemo, Strings1

fg_xconvert()

Prototype

```
int fg_xconvert (int column);  
Function fg_xconvert (ByVal column As Long) As Long  
function fg_xconvert (column : integer) : integer;
```

Description

The **fg_xconvert()** legacy function translates a character space column to the screen space coordinate of its leftmost pixel. Using **fg_xconvert(1)** is an easy way to determine the width of a character cell in pixels.

Parameters

column is the character space column to translate.

Return value

The screen space x coordinate of the leftmost pixel in the character space column *column*.

Restrictions

none

Replaced by

Screen space

fg_xscreen()

Prototype

```
int fg_xscreen (double x);  
Function fg_xscreen (ByVal x As Double) As Long  
function fg_xscreen (x : real) : integer;
```

Description

The **fg_xscreen()** function translates a 2D world space x coordinate to its screen space equivalent.

Parameters

x is the world space coordinate to translate.

Return value

The screen space x coordinate equivalent to the world space coordinate x.

Restrictions

none

See also

fg_xworld(), fg_yscreen(), fg_yworld()

fg_xvb()

Prototype

```
int fg_xvb (int ix);  
Function fg_xvb (ByVal ix As Long) As Long  
function fg_xvb (ix : integer) : integer;
```

Description

The **fg_xvb()** function translates a client area x coordinate to the corresponding screen space x coordinate within the active virtual buffer.

Parameters

ix is the client coordinate to translate.

Return value

The screen space x coordinate corresponding to the client coordinate *ix*.

Restrictions

none

See also

fg_xclient(), fg_yclient(), fg_yvb()

fg_xview()

Prototype

```
int fg_xview (int vx);  
Function fg_xview (ByVal vx As Long) As Long  
function fg_xview (vx : integer) : integer;
```

Description

The **fg_xview()** function translates a horizontal viewport coordinate to the corresponding screen space x coordinate.

Parameters

`vx` is the horizontal viewport coordinate to translate.

Return value

The screen space x coordinate corresponding to the specified viewport coordinate. If no viewport has been defined, the return value will be equal to `vx`.

Restrictions

none

See also

`fg_getview()`, `fg_setview()`, `fg_yview()`

fg_xworld()

Prototype

```
double fg_xworld (int ix);  
Function fg_xworld (ByVal ix As Long) As Double  
function fg_xworld (ix : integer) : real;
```

Description

The **fg_xworld()** function translates a screen space x coordinate to its 2D world space equivalent.

Parameters

ix is the screen space coordinate to translate.

Return value

The world space x coordinate equivalent to the screen space coordinate *ix*.

Restrictions

none

See also

fg_xscreen(), fg_yscreen(), fg_yworld()

fg_yalpha()

Prototype

```
int fg_yalpha (int iy);  
Function fg_yalpha (ByVal iy As Long) As Long  
function fg_yalpha (iy : integer) : integer;
```

Description

The **fg_yalpha()** legacy function translates a screen space y coordinate to the character space row containing that coordinate.

Parameters

iy is the screen space coordinate to translate.

Return value

The character space row containing the screen space coordinate *iy*.

Restrictions

none

Replaced by

Screen space

fg_yclient()

Prototype

```
int fg_yclient (int iy);  
Function fg_yclient (ByVal iy As Long) As Long  
function fg_yclient (iy : integer) : integer;
```

Description

The **fg_yclient()** function translates a screen space y coordinate to the corresponding client area y coordinate.

Parameters

iy is the screen space coordinate to translate.

Return value

The client y coordinate corresponding to the screen space coordinate *iy*.

Restrictions

none

See also

fg_xclient(), fg_xvb(), fg_yvb()

Examples

Fontdemo, Strings1

fg_yconvert()

Prototype

```
int fg_yconvert (int row);  
Function fg_yconvert (ByVal row As Long) As Long  
function fg_yconvert (row : integer) : integer;
```

Description

The **fg_yconvert()** legacy function translates a character space row to the screen space coordinate of its top (lowest-numbered) pixel. Using **fg_yconvert(1)** is an easy way to determine the height in pixels of a character cell.

Parameters

row is the character space row to translate.

Return value

The screen space y coordinate of the top pixel in the character space row *row*.

Restrictions

none

Replaced by

Screen space

fg_yscreen()

Prototype

```
int fg_yscreen (double y);  
Function fg_yscreen (ByVal y As Double) As Long  
function fg_yscreen (y : real) : integer;
```

Description

The **fg_yscreen()** function translates a 2D world space y coordinate to its screen space equivalent.

Parameters

y is the world space coordinate to translate.

Return value

The screen space y coordinate equivalent to the world space coordinate y.

Restrictions

none

See also

fg_xscreen(), fg_xworld(), fg_yworld()

fg_yvb()

Prototype

```
int fg_yvb (int iy);  
Function fg_yvb (ByVal iy As Long) As Long  
function fg_yvb (iy : integer) : integer;
```

Description

The **fg_yvb()** function translates a client area y coordinate to the corresponding screen space y coordinate within the active virtual buffer.

Parameters

iy is the client coordinate to translate.

Return value

The screen space y coordinate corresponding to the client coordinate *iy*.

Restrictions

none

See also

fg_xclient(), fg_xvb(), fg_yclient()

fg_yview()

Prototype

```
int fg_yview (int vy);  
Function fg_yview (ByVal vy As Long) As Long  
function fg_yview (vy : integer) : integer;
```

Description

The **fg_yview()** function translates a vertical viewport coordinate to the corresponding screen space y coordinate.

Parameters

vy is the vertical viewport coordinate to translate.

Return value

The screen space y coordinate corresponding to the specified viewport coordinate. If no viewport has been defined, the return value will be equal to vy.

Restrictions

none

See also

fg_getview(), fg_setview(), fg_xview()

fg_yworld()

Prototype

```
double fg_yworld (int iy);  
Function fg_yworld (ByVal iy As Long) As Double  
function fg_yworld (iy : integer) : real;
```

Description

The **fg_yworld()** function translates a screen space y coordinate to its 2D world space equivalent.

Parameters

iy is the screen space coordinate to translate.

Return value

The world space y coordinate equivalent to the screen space coordinate *iy*.

Restrictions

none

See also

fg_xscreen(), fg_xworld(), fg_yscreen()

fg_zballoc()

Prototype

```
int fg_zballoc (int width, int height);
```

```
Function fg_zballoc (ByVal width As Long, ByVal height As Long) As Long
```

```
function fg_zballoc (width, height : integer) : integer;
```

Description

The **fg_zballoc()** function creates a z-buffer of the specified size. The memory for the z-buffer is allocated automatically.

Parameters

width is the z-buffer width in pixels.

height is the z-buffer height in pixels.

Return value

If successful, **fg_zballoc()** returns a handle by which the z-buffer is referenced. If there is not enough free memory to create the z-buffer, **fg_zballoc()** returns zero.

Restrictions

none

See also

fg_ddsetup(), fg_zbframe(), fg_zbfree(), fg_zbopen()

Examples

Geometry, TMcube, TMcubeX

fg_zbframe()

Prototype

```
void fg_zbframe (void);  
Sub fg_zbframe ()  
procedure fg_zbframe;
```

Description

The **fg_zbframe()** function prepares the active z-buffer for the next animation frame.

Parameters

none

Return value

none

Restrictions

none

See also

fg_zballoc(), fg_zbfree(), fg_zbopen()

Examples

TMcube, TMcubeX

fg_zbfree()

Prototype

```
void fg_zbfree (int handle);  
Sub fg_zbfree (ByVal handle As Long)  
procedure fg_zbfree (handle : integer);
```

Description

The **fg_zbfree()** function frees the memory allocated to the specified z-buffer.

Parameters

handle is the handle that references the z-buffer to free.

Return value

none

Restrictions

none

See also

fg_zballoc(), fg_zbframe(), fg_zbopen()

Examples

Geometry, TMcube, TMcubeX

fg_zbopen()

Prototype

```
void fg_zbopen (int handle);  
Sub fg_zbopen (ByVal handle As Long)  
procedure fg_zbopen (handle : integer);
```

Description

The **fg_zbopen()** function makes an existing z-buffer the active z-buffer. If another z-buffer is open when you call **fg_zbopen()**, that z-buffer is first closed.

Parameters

handle is the handle that references the desired z-buffer, as returned by **fg_zballoc()**. It must be a valid z-buffer handle.

Return value

none

Restrictions

The active z-buffer must be at least as large as the active virtual buffer.

See also

fg_zballoc(), fg_zbframe(), fg_zbfree()

Examples

Geometry, TMcube, TMcubeX