gfxlib - A 2D Game Graphics Library for BBCSDL

0. Contents

1. Summary

This library contains a set of routines which are designed to emulate, partially, the functionality of David Williams's GFXLIB & GFXLIB2 libraries for BB4W. The hope is that it can be used both for conversions from BB4W, and for new programs written for BBCSDL. Therefore whilst it sticks quite closely to the interfaces used in GFXLIB2 it does not slavishly do so if performance would be compromised.

2. Coordinate system

Specifically, the library uses top-down Y coordinates (zero is at the top) so that it is not necessary for the routines to do a conversion. Although this makes more work for anybody converting a program from BB4W (typically it will be necessary to change 'y' to 'WinH% - y - h - 1' whenever calling one of the library routines) it should help performance in new programs.

3. Limitations

Whilst SDL2 can efficiently perform many of the transformations supported by GFXLIB (scaling, rotating, flipping, blending, tinting etc.) there are a few which would be very difficult to achieve, at least efficiently and across all platforms. These include OR/AND/XOR plotting, RGB rearrangement and colour replacement.

One thing that SDL2 is particularly badly suited to is pixel-accurate collision detection. In GFXLIB this is usually achieved by borrowing the alpha channel as a collision mask, which is convenient and efficient, but there is nothing similar available in SDL2 (even if the alpha channel were usable for this purpose, reading it back from the GPU would be unacceptably slow).

So this library provides two workarounds. Firstly there *is* support for pixel-accurate collision detection, using SDL Surfaces, but this is quite slow and is unlikely to be usable in games with a large number of objects with which accurate collisions need to be detected (e.g. 'baddies'). But when speed is not critical, or the number of objects is small, it does provide a good solution.

Secondly the library supports rectangle-based collision detection, in which the bounding rectangle, rather than the object itself, is the target for a collision. This is much faster but does mean that a collision will be detected if a 'bullet' passes close enough to an object to intersect with its bounding rectangle but not to hit the object itself.

It is straightforward to mix both methods in the same program, so you could use the fast method for objects which approximate to a rectangular shape and the pixel-accurate approach for those that don't.

4. Performance tuning

Although the library routines may be called in-situ, the overhead of the procedure calls and the saving of LOCAL variables, creation of LOCAL structures etc. may be significant in critical applications. In that event transfer the contents of the library routines into your program inline, and relocate the structure declarations, LOCAL statements etc. to the parent procedure.

Additionally, consider storing the position and size of your objects in structures, to eliminate the overhead of transferring them into 'rect' structures every time a graphics operation is performed. It is acceptable for them to be the first four (integer) members of a larger structure, containing other properties of the object.

If using collision detection, and if you don't need to use the 'alpha' value to determine which object has been collided with (for example because you have other means of discovering that), set the stored 'alpha' value to 255 (&FF) since this will result in a speed improvement.

5. Core routines

5.1 PROC_gfxInit

This routine simply declares a number of global variables. It should be called during initialisation of your program, and may safely be called more than once.

5.2 PROC_gfxCreateCanvas(W%,H%)

This routine must be called if your program uses collision detection. It sets up a global canvas to contain the 'alpha' values. It should be called once during initialisation, and must be freed before exit by calling PROC_gfxFreeCanvas().

     W%,H%: The canvas dimensions, in pixels. Usually the size of your window.

5.3 PROC_gfxClr(R%,G%,B%)
     similar to GFXLIB_Clr

Clears the screen (render target) to the specified (opaque) background colour;

also zeros the 'alpha' values in the global canvas if one has been created.

     R%,G%,B%: Background colour (red, green, blue: 0-255)

5.4 PROC_gfxClrX(R%,G%,B%,A%)
     similar to GFXLIB_ClrX

Clears the render target to the specified background colour and opacity (doesn't affect the global canvas). Typically used to clear to a transparent background.

     R%,G%,B%: Background colour (red, green, blue: 0-255)
     A%: Background opacity, from 0 (transparent) to 255 (fully opaque)

5.5 PROC_gfxCopy(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_Copy

Copies a region of the window (or render target) to a texture/bitmap.

     t%%: Destination texture; must have been created with FN_gfxCreateTexture
     W%,H%,X%,Y%: Width, height, x-position, y-position (pixels, top-down)

5.6 PROC_gfxLine(x1%,y1%,x2%,y2%,R%,G%,B%)
     similar to GFXLIB_Line

Draws a one-pixel thick straight line between two points, in the specified colour.

     x1%,y1%,x2%,y2%: endpoints (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255)

5.7 PROC_gfxPlotPixel(X%,Y%,R%,G%,B%)
     similar to GFXLIB_PlotPixel

Plots a single pixel in the specified colour.

     X%,Y%: Coordinates (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255)

5.8 PROC_gfxPlotPoint(T%,X%,Y%,R%,G%,B%)
     similar to GFXLIB_PlotPoint

Plots a pre-defined shape in the specified colour, as in GFXLIB_PlotPoint.

     T%: Point type (0-26)
     X%,Y%: Coordinates (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255)

5.9 PROC_gfxRectangleSolid(X%,Y%,W%,H%,R%,G%,B%)
     similar to GFXLIB_RectangleSolid

Plots a solid rectangle in the specified colour.

     X%,Y%,W%,H%: X-position, y-position, width, height (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255)

6. Plotting routines that perform scaling but not rotation or flipping

```
6.1  PROC_gfxPlotScale(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_Plot and GFXLIB_PlotScale
```

Plots a scaled texture/bitmap/sprite at the specified (top-left) coordinates.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture()
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

If you do not want any scaling pass the original width and height of the bitmap.

```
6.2  PROC_gfxPlotScaleAvg(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_PlotScaleAvg
```

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) and averages the pixels' RGB values with those of the background (50% blend).

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

```
6.3  PROC_gfxPlotScaleBlend(t%%,W%,H%,X%,Y%,O%)
     similar to GFXLIB_PlotScaleBlend
```

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the specified blending between the pixels' RGB values and those of the background.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     O%: Opacity of the scaled texture/bitmap, 0 (transparent) to 255 (opaque)
```

```
6.4  PROC_gfxPlotScaleFade(t%%,W%,H%,X%,Y%,A%)
     similar to GFXLIB_PlotScaleFade
```

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the pixels' RGB values multiplied by the specified factor.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     A%: Multiplication factor, 0 (0.0) to 255 (1.0); this is opposite to GFXLIB2
```

```
6.5  PROC_gfxPlotScaleMultiply(t%%,W%,H%,X%,Y%)
```

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the pixels' RGB values multiplied by those of the background (&FF = 1.0).

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

```
6.6  PROC_gfxPlotScaleAddSaturate(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_PlotAddARGBSaturate
```

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the pixels' RGB values summed with those of the background, clipping at peak white.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

6.7 PROC_gfxPlotScaleTint(t%%,w%%,W%,H%,X%,Y%,R%,G%,B%,S%)
     similar to GFXLIB_PlotScaleTint

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the pixels' RGB values colour-blended with the supplied tinting colour.

- t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
- w%%: Opaque white texture, returned from FN_gfxLoadWhiteTexture
- W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
- R%,G%,B%: Tint colour (red, green, blue: 0-255)
- S%: Strength of the tint in the range 0 (none) to 255 (maximum)

6.8 PROC_gfxPlotScaleTintBlend(t%%,w%%,W%,H%,X%,Y%,R%,G%,B%,S%,O%)
     similar to GFXLIB_PlotScaleTintBlend

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the pixels' RGB values colour-blended with the supplied tinting colour and then blended with the background (approximately).

- t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
- w%%: Opaque white texture, returned from FN_gfxLoadWhiteTexture
- W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
- R%,G%,B%: Tint colour (red, green, blue: 0-255)
- S%: Strength of the tint in the range 0 (none) to 255 (maximum)
- O%: Opacity of the tinted pixels (0 = transparent, 255 = opaque)

6.9 PROC_gfxReversePlotScale(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_ReversePlot2

Plots a scaled texture/bitmap/sprite at the specified coordinates (top-left) with the destination alpha channel, rather than the source alpha, determining the key.

- t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
- W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)

7. Plotting routines which perform scaling and rotation or flipping

7.1 PROC_gfxPlotRotateScale(t%%,W%,H%,X%,Y%,a#)
     similar to GFXLIB_PlotRotateScaleBilinear

Plots a rotated and scaled texture/bitmap/sprite at the specified coordinates. Note that the supplied coordinates are of the CENTRE of the sprite (centre of rotation) not the top-left corner as is the case for the non-rotating routines.

- t%%: Texture/sprite e.g. as returned from FN_gfxLoadTexture
- W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)
- a#: Angle of rotation, in degrees clockwise

7.2   PROC_gfxPlotRotateScaleBlend(t%%,W%,H%,X%,Y%,a#,O%)

Plots a rotated and scaled texture/bitmap/sprite at the specified coordinates, with blending between the pixels' RGB values and those of the background.
Note that the supplied coordinates are of the CENTRE of the sprite.

- t%%: Texture/sprite e.g. as returned from FN_gfxLoadTexture
- W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)

```
     a#: Angle of rotation, in degrees clockwise
     O%: Opacity of the resulting pixels, 0 (transparent) to 255 (opaque)
```

7.3 PROC_gfxPlotRotateScaleTint(t%%,w%%,W%,H%,X%,Y%,a#,R%,G%,B%,S%)

Plots a rotated and scaled texture/bitmap/sprite at the specified coordinates, with the pixels' RGB values colour-blended with the supplied tinting colour. Note that the supplied coordinates are of the CENTRE of the sprite.

```
     t%%: Texture/sprite e.g. as returned from FN_gfxLoadTexture
     w%%: Opaque white texture returned from FN_gfxLoadWhiteTexture
     W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)
     a#: Angle of rotation, in degrees clockwise
     R%,G%,B%: Tint colour (red, green, blue: 0-255)
     S%: Strength of the tint in the range 0 (none) to 255 (maximum)
```

7.4 PROC_gfxPlotFlipScale(t%%,W%,H%,X%,Y%,F%)

     similar to GFXLIB_BPlotFlip

Plots a flipped and scaled texture/bitmap/sprite at the specified coordinates.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
```

7.5 PROC_gfxPlotFlipScaleBlend(t%%,W%,H%,X%,Y%,F%,O%)

Plots a flipped and scaled texture/bitmap/sprite at the specified coordinates, with blending between the pixels' RGB values and those of the background.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
     O%: Opacity of the flipped bitmap, 0 (transparent) to 255 (opaque)
```

7.6 PROC_gfxPlotFlipScaleTint(t%%,w%%,W%,H%,X%,Y%,F%,R%,G%,B%,S%)

Plots a filpped and scaled texture/bitmap/sprite at the specified coordinates, with the pixels' RGB values colour-blended with the supplied tinting colour.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
     R%,G%,B%: Tint colour (red, green, blue: 0-255)
     S%: Strength of the tint in the range 0 (none) to 255 (maximum)
```

8. Plotting routines which plot only the shape (silhouette), not the colour

These routines are typically used to plot a shadow.

8.1 PROC_gfxPlotScaleShapeHalfIntensity(t%%,W%,H%,X%,Y%)

     similar to GFXLIB_PlotScaleShapeHalfIntensity

Plots a scaled shape (silhouette) at the specified coordinates (top-left) within which the background pixels are attenuated to half their original values.

```
     t%%: Texture e.g. as returned from FN_gfxLoadTexture; only the shape is used
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

8.2 PROC_gfxPlotScaleShapeQuarterIntensity(t%%,W%,H%,X%,Y%)
     similar to GFXLIB_PlotScaleShapeQuarterIntensity

Plots a scaled shape (silhouette) at the specified coordinates (top-left) within which the background pixels are attenuated to one quarter of their original values.

```
     t%%: Texture e.g. as returned from FN_gfxLoadTexture; only the shape is used
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
```

8.3 PROC_gfxPlotScaleShape(w%%,W%,H%,X%,Y%,R%,G%,B%)
     similar to GFXLIB_PlotScaleShape

Plots a scaled shape (silhouette) at the specified coordinates (top-left) with the supplied RGB colour. Note that this requires an opaque white source texture.

```
     w%%: Opaque white texture, returned from FN_gfxLoadWhiteTexture (see below)
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     R%,G%,B%: Colour multiplication factors, 0 (0.0) to 255 (1.0)
```

This routine may also be used to modulate the RGB values of a non-white texture.

8.4 PROC_gfxPlotScaleShapeBlend(w%%,W%,H%,X%,Y%,R%,G%,B%,O%)
     similar to GFXLIB_PlotShapeBlend

Plots a scaled shape (silhouette) at the specified coordinates (top-left) with the supplied RGB colour and colour-blended with the background.

```
     w%%: Opaque white texture, returned from FN_gfxLoadWhiteTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255); requires a white texture
     O%: Opacity of the shape (silhouette), 0 (transparent) to 255 (opaque)
```

8.5 PROC_gfxPlotRotateShapeHalfIntensity(t%%,W%,H%,X%,Y%,a#)
     similar to GFXLIB_PlotRotateShapeHalfIntensity

Plots a rotated and scaled shape (silhouette) centred at the specified coordinates within which the background pixels are attenuated to half their original values.

```
     t%%: Texture e.g. as returned from FN_gfxLoadTexture; only the shape is used
     W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)
     a#: Angle of rotation, in degrees clockwise
```

8.6 PROC_gfxPlotFlipShapeHalfIntensity(t%%,W%,H%,X%,Y%,F%)

Plots a scaled and flipped shape (silhouette) at the specified top-left coordinates within which the background pixels are attenuated to half their original values.

```
     t%%: Texture e.g. as returned from FN_gfxLoadTexture; only the shape is used
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
```

9. Plotting routines which also set the 'alpha' value in the bounding rectangle

These routines are fast, compared with those that set pixel-accurate 'alpha'. If objects overlap, the 'alpha' value corresponds to the last (top) object plotted.

```
9.1  PROC_gfxRectangleSolidSetAlpha(X%,Y%,W%,H%,R%,G%,B%,A%)
```

Plots a solid rectangle in the specified colour and sets the supplied 'alpha' value in the global canvas.

```
     X%,Y%,W%,H%: X-position, y-position, width, height (pixels, top-down)
     R%,G%,B%: Colour (red, green, blue: 0-255)
     A%: The 'alpha' value to set in the canvas created by PROC_gfxCreateCanvas
```

```
9.2  PROC_gfxPlotScaleSetAlphaRect(t%%,W%,H%,X%,Y%,A%)
     similar to GFXLIB_PlotScaleSetAlphaValue
```

Plots a scaled texture/bitmap/sprite at the specified (top-left) coordinates and sets the supplied 'alpha' value within the bounding rectangle in the global canvas.

```
     t%%: Texture/bitmap e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     A%: The 'alpha' value to set in the canvas created by PROC_gfxCreateCanvas
```

```
9.3  PROC_gfxPlotRotateScaleSetAlphaRect(t%%,W%,H%,X%,Y%,a#,A%)
     similar to GFXLIB_PlotRotateSetAlphaValue
```

Plots a rotated and scaled texture/bitmap/sprite at the supplied centre coordinates and sets the supplied 'alpha' value within the bounding rectangle in the canvas.

```
     t%%: Texture/bitmap/sprite e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)
     a#: Angle of rotation, in degrees clockwise
     A%: The `alpha' value to set in the canvas created by PROC_gfxCreateCanvas
```

```
9.4  PROC_gfxPlotFlipScaleSetAlphaRect(t%%,W%,H%,X%,Y%,F%,A%)
```

Plots a flipped and scaled texture/bitmap/sprite at the supplied corner coordinates and sets the supplied 'alpha' value within the bounding rectangle in the canvas.

```
     t%%: Texture/bitmap/sprite e.g. as returned from FN_gfxLoadTexture
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
     A%: The `alpha' value to set in the canvas created by PROC_gfxCreateCanvas
```

10. Plotting routines which also set the 'alpha' value to pixel-accuracy

These routines are slow, compared with those that set the 'alpha' within the entire bounding rectangle. If objects overlap, the 'alpha' values combine additively, saturating at 255 (although if scaling is involved the addition may be inaccurate).

If you don't need to use the 'alpha' value to identify which object was collided with (for example because you have other means of determining that) and only need it to be non-zero, set it to 255 since that will result in a speed improvement.

```
10.1 PROC_gfxShapeSetAlpha(s%%,X%,Y%,A%)
     similar to GFXLIB_ShapeSetAlphaValue
```

Sets the 'alpha' value within the shape of the specified object, no plotting takes place (so no texture needs to be supplied, just a surface).

```
     s%%: Surface/shape returned from FN_gfxLoadTextureAndSurface()
     X%,Y%: X-position, y-position of bounding rectangle (pixels, top-down)
     A%: The 'alpha' value to set in the canvas (use &FF if you don't care)
```

10.2 PROC_gfxPlotSetAlpha(t%%,s%%,X%,Y%,A%)
     similar to GFXLIB_PlotSetAlphaValue

Plots a texture/bitmap/sprite at the specified coordinates (top-left) and sets the 'alpha' value, of the non-transparent pixels only, in the global canvas.
If you want scaling, use PROC_gfxPlotFlipScaleSetAlpha() with the flip set to zero.

```
     t%%: Texture/bitmap/sprite e.g. as returned from FN_gfxLoadTexture
     s%%: Surface/shape returned from FN_gfxLoadTextureAndSurface()
     X%,Y%: X-position, y-position of bounding rectangle (pixels, top-down)
     A%: The 'alpha' value to set in the canvas (use &FF if you don't care)
```

10.3 PROC_gfxPlotRotateScaleSetAlpha(t%%,s%%,W%,H%,X%,Y%,a#,A%)
     similar to GFXLIB_PlotRotateScaleAlphaBlendSetAlphaValue

Plots a rotated texture/bitmap/sprite at the supplied center coordinates and sets the 'alpha' value, of the non-transparent pixels only, in the global canvas.

```
     t%%: Texture/bitmap/sprite e.g. as returned from FN_gfxLoadTexture
     s%%: Surface/shape returned from FN_gfxLoadTextureAndSurface()
     W%,H%,X%,Y%: Scaled width, height, x-centre, y-centre (pixels, top-down)
     a#: Angle of rotation, in degrees clockwise
     A%: The 'alpha' value to set in the canvas (use &FF if you don't care)
```

10.4 PROC_gfxPlotFlipScaleSetAlpha(t%%,s%%,W%,H%,X%,Y%,F%,A%)

Plots a flipped texture/bitmap/sprite at the supplied corner coordinates and sets the 'alpha' value, of the non-transparent pixels only, in the global canvas.

```
     t%%: Texture/bitmap/sprite e.g. as returned from FN_gfxLoadTexture
     s%%: Surface/shape returned from FN_gfxLoadTextureAndSurface()
     W%,H%,X%,Y%: Scaled width, height, x-position, y-position (pixels, top-down)
     F%: 1 to flip horizontally, 2 to flip vertically and 3 for both
     A%: The 'alpha' value to set in the canvas (use &FF if you don't care)
```

11. Routines used in collision detection

11.1 FN_gfxReadAlpha(X%,Y%)
     similar to GFXLIB_ReadAlphaValue

This function returns the 'alpha' value at the specified coordinates in the global canvas created by PROC_gfxCreateCanvas.

```
     X%,Y%: The coordinates at which the 'alpha' is to be read (pixels, top-down)
```

11.2 FN_gfxDetectCollision(x,y,u,v,M%,X%,Y%)
     similar to GFXLIB_DetectCollision

This function detects a collision between a 'bullet' and an object, taking into account the speed and direction of the bullet (so a collision will still be detected if the bullet is travelling so fast that it passes 'through' the object). If a collision was detected it returns the 'alpha' value at that point, else zero.

```
     x,y: Coordinates of test point (pixels, top-down)
     u,v: Velocities of test point (pixels per frame period)
     M%: Mask against which the 'alpha' value is tested (collision if non-zero)
     X%,Y%: Point at which the collision was detected (if any)
```

11.3 FN_gfxGetCumulativeAlpha(s%%,X%,Y%,M%)
     similar to GFXLIB_PlotGetCumulativeAlphaBits

Returns a non-zero result if any of the pixels 'covered' by the supplied surface (shape) have 'alpha' values with bit(s) set corresponding to the supplied mask.

```
     s%%: Surface/shape returned from FN_gfxLoadTextureAndSurface()
     X%,Y%: X-position, y-position of bounding rectangle (pixels, top-down)
     M%: Mask which is ANDed with the 'alpha' values before being summed
```

11.4 FN_gfxGetCumulativeAlphaRect(W%,H%,X%,Y%,M%)

Returns a non-zero result if any of the pixels within the supplied rectangle have 'alpha' values with bit(s) set corresponding to the supplied mask.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position (pixels, top-down)
     M%: Mask which is ANDed with the 'alpha' values before being summed
```

12. Plotting routines which take arrays of coordinates

12.1 PROC_gfxPlotPixelList(xy%(),N%,O%,R%,G%,B%,X%,Y%)
     similar to GFXLIB_PlotPixelList2

Plots multiple pixels, with optional coordinate offsets. All the pixels are plotted in the same colour, this is a limitation of SDL 2.0; if you need to plot multiple colours sort your array by colour and call this function once for each differently-coloured group.
Important: the supplied coordinate array must be aligned, so declare it as LOCAL.

```
     xy%(): Array of x,y coordinates, in pixels, declared as DIM xy%(N%-1,1)
     N%: Number of pixels to plot
     O%: First array index to plot (indexes O% to O% + N% - 1 are plotted)
     R%,G%,B%: Colour (red, green, blue: 0-255)
     X%,Y%: Optional offsets, in pixels, from the coordinates in the array
```

12.2 PROC_gfxPlotRectangleList(xywh%(),N%,O%,R%,G%,B%)

Plots multiple solid rectangles; all the rectangles are plotted in the same colour, this is a limitation of SDL 2.0; if you need to plot multiple colours sort your array by colour and call this function once for each differently-coloured group.
Important: the supplied coordinate array must be aligned, so declare it as LOCAL.

```
     xywh%(): Array of xpos, ypos, width, height; declared as DIM xywh%(N%-1,3)
     N%: Number of rectangles to plot
     O%: First array index to plot (indexes O% to O% + N% - 1 are plotted)
     R%,G%,B%: Colour (red, green, blue: 0-255)
```

```
12.3 PROC_gfxPlotColumnList(t%%,W%,H%,N%,i%(),x%(),y%())
     similar to GFXLIB_PlotBMColumnList
```

Plots multiple columns of a texture/bitmap/sprite with independent coordinates, so that you can achieve effects such as warping.

```
     t%%: Texture/sprite e.g. as returned from FN_gfxLoadTexture
     W%,H%: Width and height of the texture/sprite, in pixels
     N%: Number of columns to plot
     i%(): Array of column indexes (each 0 to W% - 1)
     x%(),y%(): Arrays of destination coordinates (pixels, top-down)
```

```
12.4 PROC_gfxPlotRowList(t%%,W%,H%,N%,i%(),x%(),y%())
     similar to GFXLIB_PlotBMRowList
```

Plots multiple rows of a texture/bitmap/sprite with independent coordinates, so that you can achieve effects such as warping.

```
     t%%: Texture/sprite e.g. as returned from FN_gfxLoadTexture
     W%,H%: Width and height of the texture/sprite, in pixels
     N%: Number of rows to plot
     i%(): Array of row indexes (each 0 to H% - 1)
     x%(),y%(): Arrays of destination coordinates (pixels, top-down)
```

13. Routines which operate on a rectangular region of the window or render target

```
13.1 PROC_gfxAdd(W%,H%,X%,Y%,S%)
     similar to GFXLIB_MMXAdd64
```

Adds a specified amount to a rectangular region of the window or render target.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position in pixels (top-down)
     S%: RGB summation factor, 0 to 255 added to each of red, green and blue
```

```
13.2 PROC_gfxSubtract(W%,H%,X%,Y%,S%)
     similar to GFXLIB_MMXSubtract64
```

Subtracts a specified amount from a rectangular region of the window or render target. This can use a lot of memory so HIMEM may need to be raised.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position in pixels (top-down)
     S%: RGB subtraction factor, 0 to 128 (for larger values call twice)
```

```
13.3 PROC_gfxAttenuate(W%,H%,X%,Y%,R%,G%,B%)
     similar to GFXLIB_ColourTransform
```

Attenuates (multiplies by a factor) a rectangular region of the window or render target. The red, green and blue components can be independently attenuated.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position in pixels (top-down)
     R%,G%,B%: RGB multiplying factors, 0 (0.0) to 255 (1.0)
```

```
13.4 PROC_gfxDesaturate(W%,H%,X%,Y%,S%)
     similar to GFXLIB_DesaturateColour
```

Desaturates (attenuates the chrominance of) a rectangular region of the window or render target. This can use a lot of memory so HIMEM may need to be raised.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position in pixels (top-down)
     S%: Saturation, from 0 (greyscale) to 255 (unchanged)
```

13.5 PROC_gfxBoxBlurNxN(W%,H%,X%,Y%,N%)
     similar to GFXLIB_BoxBlurNxNR

Blurs (spatially filters) a rectangular region of the window or render target by a specified amount. Only the smallest (3x3) filter is an accurate 'box' filter.

```
     W%,H%,X%,Y%: Width, height, x-position, y-position in pixels (top-down)
     N%: Aperture of the filter; must be 3, 7, 15 or 31
```

14. Miscellaneous routines

14.1 PROC_gfxDrawText(t$,X%,Y%,R%,G%,B%)

Draws text at the specified coordinates, in the currently-selected font. Note that changing the font is slow, so consider pre-rendering constant text into a texture using FN_gfxCreateTextureFromText rather than changing the font on-the-fly.

```
     t$: The text string to draw
     X%,Y%: The coordinates of the top-left corner of the text (pixels, top-down)
     R%,G%,B%: Text colour (red, green, blue: 0-255)
```

14.2 PROC_gfxDrawCentredText(t$,Y%,R%,G%,B%)

Draws horizontally-centred text at the specified coordinates, in the currently-selected font. Note that changing the font is slow, so consider pre-rendering constant text into a texture using FN_gfxCreateTextureFromText.

```
     t$: The text string to draw
     Y%: Vertical coordinate of the top of the text (pixels, top-down)
     R%,G%,B%: Text colour (red, green, blue: 0-255)
```

14.3 PROC_gfxSaveAndSetDispVars(g{},t%%)
     similar to GFXLIB_SaveAndSetDispVars

Saves the current render target and sets the supplied texture as the new target; subsequent plotting operations will plot to that texture rather than to the screen (although you should use *REFRESH OFF to prevent the screen being affected). Note that if an error occurs before restoring, the system may be in an unstable state; consider using ON ERROR LOCAL to avoid this.

```
     g{}: An opaque, undeclared, structure
     t%%: The new render target, must have been created using FN_gfxCreateTexture
```

14.4 PROC_gfxRestoreDispVars(g{})
     similar to GFXLIB_RestoreDispVars

Restores the render target to what it was prior to PROC_gfxSaveAndSetDispVars().

```
     g{}: The opaque structure originally passed to PROC_gfxSaveAndSetDispVars
```

14.5 PROC_gfxGetCubicBezierCurvePoint(x#(),y#(),t#,x#,y#)

similar to GFXLIB_GetQuadraticBezierCurvePoint

Returns a point on a cubic Bézier curve, defined by four control points. The point returned depends on a supplied parameter in the range 0.0 (start) to 1.0 (end). The supplied coordinate arrays *must* be aligned, so declare them as LOCAL.

```
     x#(),y#(): Arrays (# suffix) of the x- and y-coordinates of 4 control points
     t#: A parameter in the range 0.0 to 1.0 returning a point on the curve
     x#,y#: The returned point: x#(0),y#(0) if t = 0.0; x#(3),y#(3) if t = 1.0
```

14.6 PROC_gfxGetQuarticBezierCurvePoint(x#(),y#(),t#,x#,y#)
     similar to GFXLIB_GetQuarticBezierCurvePoint

Returns a point on a quartic Bézier curve, defined by five control points. The point returned depends on a supplied parameter in the range 0.0 (start) to 1.0 (end). The supplied coordinate arrays *must* be aligned, so declare them as LOCAL.

```
     x#(),y#(): Arrays (# suffix) of the x- and y-coordinates of 5 control points
     t#: A parameter in the range 0.0 to 1.0 returning a point on the curve
     x#,y#: The returned point: x#(0),y#(0) if t = 0.0; x#(4),y#(4) if t = 1.0
```

15. Texture and surface support routines

15.1 FN_gfxCreateTexture(W%,H%)

Creates an empty texture/bitmap, suitable for use as a new render target.

```
     W%,H%: The dimensions of the created texture, in pixels
```

The returned value is a pointer so must be stored in a 64-bit integer (e.g. t%%) or variant (e.g. t) variable. The texture must be destroyed before exit by calling PROC_gfxDestroyTexture() otherwise a memory leak will result.

15.2 FN_gfxCreateTextureFromText(t$,R%,G%,B%,W%,H%)

Creates a new texture containing pre-rendered text, in the current font, on a transparent background. Changing the font on-the-fly is slow so pre-render your constant text into textures if possible.

```
     t$; The text string to be rendered to the texture
     R%,G%,B%: Text colour (red, green, blue: 0-255)
     W%,H%: The returned width and height of the created texture
```

The returned value is a pointer so must be stored in a 64-bit integer (e.g. t%%) or variant (e.g. t) variable. The texture must be destroyed before exit by calling PROC_gfxDestroyTexture() otherwise a memory leak will result.

15.3 FN_gfxLoadTexture(file$,key%)

Loads a texture from an image file in BMP, GIF, JPG, PIC, PNG, PNM or TGA format, optionally specifying an RGB colour that will be treated as 'transparent'. If no colour key is supplied the transparency will be taken from the alpha channel of the image file (e.g. GIF or PNG), if any. For black pass &FF000000 as the key.

```
     file$: The (preferably absolute) path and filename of the image file.
     key%: A 24-bit colour (&BBGGRR) which will be treated as transparent.
```

The returned value is a pointer so must be stored in a 64-bit integer (e.g. t%%) or variant (e.g. t) variable. The texture must be destroyed before exit by calling PROC_gfxDestroyTexture() otherwise a memory leak will result.

15.4 FN_gfxLoadWhiteTexture(file$,key%)

As FN_gfxLoadTexture() except that non-transparent pixels are forced to peak white.

    file$: The (preferably absolute) path and filename of the image file.
    key%: A 24-bit colour (&BBGGRR) which will be treated as transparent.

The returned value is a pointer so must be stored in a 64-bit integer (e.g. t%%) or variant (e.g. t) variable. The texture must be destroyed before exit by calling PROC_gfxDestroyTexture() otherwise a memory leak will result.

15.5 FN_gfxLoadTextureAndSurface(file$,key%,s%%)

As FN_gfxLoadTexture() except that a surface is returned as well as a texture.

    file$: The (preferably absolute) path and filename of the image file.
    key%: A 24-bit colour (&BBGGRR) which will be treated as transparent.
    s%%: The returned surface, typically for pixel-accurate collision detection.

The returned values are pointers so must be stored in 64-bit integer (e.g. t%%) or variant (e.g. t) variables. The texture must be destroyed before exit by calling PROC_gfxDestroyTexture() and the surface freed by caling PROC_gfxFreeSurface().

15.6 PROC_gfxDestroyTexture(t%%)

Destroys a texture created by FN_gfxCreateTexture(), FN_gfxCreateTextureFromText(), FN_gfxLoadTexture(), FN_gfxLoadWhiteTexture() or FN_gfxLoadTextureAndSurface().

    t%%: The texture to be destroyed (must be a 64-bit integer or variant).

15.7 PROC_gfxFreeSurface(s%%)

Frees a surface created by FN_gfxLoadTextureAndSurface().

    s%%: The surface to be destroyed (must be a 64-bit integer or variant).

15.8 PROC_gfxFreeCanvas

Frees the canvas created by PROC_gfxCreateCanvas().