### Table of Contents

Table of Contents .................................................................................................................. i
INTRODUCTION............................................................................................................. III

Base Types ....................................................................................................................... iii
Predefined Constants........................................................................................................ iv
API Reference ....................................................................................................................... v

grAADrawTriangle ................................................................................................................. 1
grAlphaBlendFunction .......................................................................................................... 3
grAlphaCombine ..................................................................................................................... 5
grAlphaControlsITRGBLighting ............................................................................................. 9
grAlphaTestFunction ............................................................................................................ 11
grAlphaTestReferenceValue ................................................................................................. 13
grBufferClear ....................................................................................................................... 15
grBufferSwap ....................................................................................................................... 17
grChromaKeyMode ............................................................................................................. 19
grChromaKeyValue .............................................................................................................. 21
grChromaRangeExt .............................................................................................................. 23
grChromaRangeModeExt ....................................................................................................... 25
grClipWindow ...................................................................................................................... 27
grColorCombine ................................................................................................................... 29
grColorMask ........................................................................................................................ 31
grConstantColorValue ......................................................................................................... 33
grCoordinateSpace .............................................................................................................. 35
grCullMode .......................................................................................................................... 37
grDepthBiasLevel ............................................................................................................... 39
grDepthBufferFunction ........................................................................................................ 41
grDepthBufferMode ............................................................................................................. 43
grDepthMask ........................................................................................................................ 45
grDepthRange ...................................................................................................................... 47
grDisable ............................................................................................................................. 49
grDisableAllEffects ............................................................................................................. 51
grDitherMode ....................................................................................................................... 53
grDrawLine .......................................................................................................................... 55
grDrawPoint ......................................................................................................................... 57
grDrawTriangle .................................................................................................................... 59
grDrawVertexArray .............................................................................................................. 61
grDrawVertexArrayContiguous ........................................................................................... 63
grEnable ............................................................................................................................... 65
grErrorSetCallback ............................................................................................................. 67
grFinish ............................................................................................................................... 69
grFlush ................................................................................................................................. 71
grFogColorValue ................................................................................................................ 73
grFogMode .......................................................................................................................... 75
grFogTable .......................................................................................................................... 77
grGet .................................................................................................................................... 79
grGetProcAddress .............................................................................................................. 83
grGetString .......................................................................................................................... 85
grGlideGetState .................................................................................................................. 87
grGlideGetVertexLayout ...................................................................................................... 89
**Introduction**

This document is the official programming reference for version 3.0 of 3Dfx Interactive Glide Rasterization Library. The Glide Library is a low-level rendering and state management subroutine library that serves as a thin layer over the register level interface to the 3Dfx Interactive family of graphics accelerators. Glide permits easy and efficient implementation of 3D rendering libraries, games, and drivers on the graphics hardware. Glide only implements operations that are natively supported by the hardware. Higher level operations are located in the Glide Utility Library, which is currently part of Glide.

Glide serves three primary purposes:

- It relieves programmers from hardware specific issues such as timing, maintaining register shadows, and working with hard-coded register constants and offsets.
- It defines an abstraction of the graphics hardware to facilitate ease of software porting.
- It acts as a delivery vehicle for sample source code providing in-depth optimizations.

By abstracting the low level details of interfacing with the graphics hardware into a set of C-callable APIs, Glide allows developers targeting the graphics hardware to avoid working with hardware registers and memory directly, enabling faster development and lower probability of bugs. Glide also handles mundane and error prone chores, such as initialization and shutdown.

Glide currently consists of Glide APIs as well as Glide Utility APIs. All Glide APIs begin with the `gr` prefix, all Glide Utility APIs being with the `gu` prefix. Glide Utility APIs do not directly communicate with hardware registers; they are strictly layered on Glide APIs. Therefore, their functionality could be performed just as easily by application code. Glide Utility APIs are included in Glide for convenience.

A 3Dfx interactive graphics accelerator consists of a Pixelfx chip that performs pixel rendering operations and manages the video frame buffer, and one or more Texelfx chips that perform texture mapping operations. Each Texelfx chip contains one Texture Mapping Unit, or TMU. The term TMU is used throughout the rest of this manual.

The internal name for the graphics subsystem is “SST”. Some function names, type definitions, and constants within Glide reflect this internal name, which is easier to type than some variation of Voodoo Graphics, Voodoo Rush, or Voodoo².

**Base Types**

All 3Dfx Interactive programming libraries use a common set of platform-independent signed and unsigned types. These types can be found in `3dfx.h` and are described in the table below:

<table>
<thead>
<tr>
<th>type name</th>
<th>format</th>
<th>ANSI C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FxU8</td>
<td>8-bit unsigned</td>
<td>unsigned char</td>
</tr>
<tr>
<td>FxI8</td>
<td>8-bit signed</td>
<td>signed char</td>
</tr>
<tr>
<td>FxU16</td>
<td>16-bit unsigned</td>
<td>unsigned short int</td>
</tr>
<tr>
<td>FxI16</td>
<td>16-bit signed</td>
<td>signed short int</td>
</tr>
<tr>
<td>FxU32</td>
<td>32-bit unsigned</td>
<td>unsigned long int</td>
</tr>
<tr>
<td>FxI32</td>
<td>32-bit signed value</td>
<td>signed long int</td>
</tr>
<tr>
<td>FxFloat</td>
<td>32-bit floating point value</td>
<td>float</td>
</tr>
<tr>
<td>FxBool</td>
<td>32-bit signed</td>
<td>long int</td>
</tr>
</tbody>
</table>
**Glide Types and Structures**

Glide overlays the base types with a collection of type definitions whose names convey the kind of values that variables of that type will assume. Each type is associated with a character string prefix that becomes part of each predefined constant associated with the type. The table below lists the types, the underlying base type, and the prefix for constants that can be used to establish a value for a parameter of the associated type. In a few cases, the unifying prefix is not used and the table provides the list of associated constants instead.

<table>
<thead>
<tr>
<th>enumerated type</th>
<th>underlying base type</th>
<th>valid values or prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrChipID_t</td>
<td>FxI32</td>
<td>GR_FBI, GR_TMU0, GR_TMU1, GR_TMU2</td>
</tr>
<tr>
<td>GrCombineFunction_t</td>
<td>FxI32</td>
<td>GR_COMBINE_FUNCTION_</td>
</tr>
<tr>
<td>GrCombineFactor_t</td>
<td>FxI32</td>
<td>GR_COMBINE_FACTOR_</td>
</tr>
<tr>
<td>GrCombineLocal_t</td>
<td>FxI32</td>
<td>GR_COMBINE_LOCAL_</td>
</tr>
<tr>
<td>GrCombineOther_t</td>
<td>FxI32</td>
<td>GR_COMBINE_OTHER_</td>
</tr>
<tr>
<td>GrAlphaSource_t</td>
<td>FxI32</td>
<td>GR_ALPHA_SOURCE_</td>
</tr>
<tr>
<td>GrColorCombineFnc_t</td>
<td>FxI32</td>
<td>GR_COLORCOMBINE_</td>
</tr>
<tr>
<td>GrAlphaBlendFnc_t</td>
<td>FxI32</td>
<td>GR_BLEND_</td>
</tr>
<tr>
<td>GrAspectRatio_t</td>
<td>FxI32</td>
<td>GR_ASPECT_LOG2_</td>
</tr>
<tr>
<td>GrBuffer_t</td>
<td>FxI32</td>
<td>GR_BUFFER_</td>
</tr>
<tr>
<td>GrChromakeyMode_t</td>
<td>FxI32</td>
<td>GR_CHROMAKEY_</td>
</tr>
<tr>
<td>GrChromaRangeMode_t</td>
<td>FxI32</td>
<td>GR_CHROMARANGE_</td>
</tr>
<tr>
<td>GrTexChromakeyMode_t</td>
<td>FxI32</td>
<td>GR_TEXCHROMA_</td>
</tr>
<tr>
<td>GrCmpFnc_t</td>
<td>FxI32</td>
<td>GR_CMP_</td>
</tr>
<tr>
<td>GrColorFormat_t</td>
<td>FxI32</td>
<td>GR_COLORFORMAT_</td>
</tr>
<tr>
<td>GrCullMode_t</td>
<td>FxI32</td>
<td>GR_CULL_</td>
</tr>
<tr>
<td>GrDepthBufferMode_t</td>
<td>FxI32</td>
<td>GR_DEPTHBUFFER_</td>
</tr>
<tr>
<td>GrDitherMode_t</td>
<td>FxI32</td>
<td>GR_DITHER_</td>
</tr>
<tr>
<td>GrFogMode_t</td>
<td>FxI32</td>
<td>GR_FOG_</td>
</tr>
<tr>
<td>GrLock_t</td>
<td>FxU32</td>
<td>GR_LFB_</td>
</tr>
<tr>
<td>GrLfbBypassMode_t</td>
<td>FxI32</td>
<td>GR_LFBBYPASS_</td>
</tr>
<tr>
<td>GrLfbWriteMode_t</td>
<td>FxI32</td>
<td>GR_LFBWRITEMODE_</td>
</tr>
<tr>
<td>GrOriginLocation_t</td>
<td>FxI32</td>
<td>GR_ORIGIN_</td>
</tr>
<tr>
<td>GrLOD_t</td>
<td>FxI32</td>
<td>GR_LOD_LOG2_</td>
</tr>
<tr>
<td>GrMipMapMode_t</td>
<td>FxI32</td>
<td>GR_MIPMAP_</td>
</tr>
<tr>
<td>GrSmoothingMode_t</td>
<td>FxI32</td>
<td>GR_SMOOTHING_</td>
</tr>
<tr>
<td>GrTextureClampMode_t</td>
<td>FxI32</td>
<td>GR_TEXTURECLAMP_</td>
</tr>
<tr>
<td>GrTextureCombineFnc_t</td>
<td>FxI32</td>
<td>GR_TEXTURECOMBINE_</td>
</tr>
<tr>
<td>GrTextureFilterMode_t</td>
<td>FxI32</td>
<td>GR_TEXTUREFILTER_</td>
</tr>
<tr>
<td>GrTextureFormat_t</td>
<td>FxI32</td>
<td>GR_TEXFMT_</td>
</tr>
<tr>
<td>GrTexImage_table_t</td>
<td>FxU32</td>
<td>GR_TEXTABLE_</td>
</tr>
<tr>
<td>GrNCCTable_t</td>
<td>FxU32</td>
<td>GR_NCCTABLE_</td>
</tr>
<tr>
<td>GrTexBaseRange_t</td>
<td>FxU32</td>
<td>GR_TEXBASE_</td>
</tr>
<tr>
<td>GrEnableMode_t</td>
<td>FxU32</td>
<td>GR_MODE_</td>
</tr>
<tr>
<td>GrCoordinateSpaceMode_t</td>
<td>FxU32</td>
<td>GR_WINDOW_COORDS, GR_CLIP_COORDS</td>
</tr>
<tr>
<td>GrLfbSrcFmt_t</td>
<td>FxU32</td>
<td></td>
</tr>
</tbody>
</table>
Glide also defines a small set of structures, as shown below.

<table>
<thead>
<tr>
<th>structure name</th>
<th>definition</th>
<th>description and usage</th>
</tr>
</thead>
</table>
| GrLfbInfo_t    | typedef struct {
|                | int size;
|                | void *lfbPtr;
|                | FxU32 strideInBytes;
|                | GrLfbWriteMode_t writeMode;
|                | GrOriginLocation_t origin;
|                } GrLfbInfo_t; | Contains state information about the linear frame buffer. Used with grLfbLock. |
| GrTexInfo      | typedef struct {
|                | GrLOD_t smallLodLog2;
|                | GrLOD_t largeLodLog2;
|                | GrAspectRatio_t aspectRatioLog2;
|                | GrTextureFormat_t format;
|                | Void *data;
|                } GrTexInfo; | Contains mipmap parameters. Used with grTexDownloadMipMap, grTexMultibaseAddress, grTexSource, and grTexTextureMemRequired. |
| GrResolution   | typedef struct {
|                | GrScreenResolution_t resolution;
|                | GrScreenRefresh_t refresh;
|                | int numColorBuffers;
|                | int numAuxBuffers;
|                } GlideResolution; | Contains display screen parameters. Used with grQueryResolutions. |

Predefined Constants
Glide predefines several constants and is also dependent upon several externally defined constants. These are documented in full in glide.h.

API Reference
The following is an API reference that lists the APIs provided by Glide, their purpose, usage, parameters, and notes describing their implementation.
grAADrawTriangle

NAME

grAADrawTriangle – draw an anti-aliased triangle

C SPECIFICATION

```c
void grAADrawTriangle( const void *a,
                        const void *b,
                        const void *c,
                        FxBool antialiasAB,
                        FxBool antialiasBC,
                        FxBool antialiasCA )
```

PARAMETERS

- **a, b, c**: Pointers to the vertices defining the triangle.
- **antialiasAB**: If **FXTRUE**, anti-alias the AB edge.
- **antialiasBC**: If **FXTRUE**, anti-alias the BC edge.
- **antialiasCA**: If **FXTRUE**, anti-alias the CA edge.

DESCRIPTION

Glide draws a triangle with the specified edges anti-aliased by setting up the alpha iterator so that it represents pixel coverage. `grAlphaCombine` must select iterated alpha and `grAlphaBlendFunction` should select `GR_BLEND_SRC_ALPHA, GR_BLEND_ONE_MINUS_SCR_ALPHA` as the RGB blend functions and `GR_BLEND_ZERO, GR_BLEND_ZERO` as the alpha blend functions if sorting from back to front and `GR_BLEND_ALPHA_SATURATE, GR_BLEND_ONE` as the RGB blend functions and `GR_BLEND_SATURATE, GR_BLEND_ONE` as the alpha blend functions if sorting from front to back. Opaque anti-aliased primitives must set alpha=255 in the vertex data. Transparent anti-aliased primitives are drawn by setting alpha to values less than 255; this alpha value is multiplied by the pixel coverage to obtain the final alpha value for alpha blending.

NOTES

If there is a steep gradient in a particular color space (i.e., green goes from 255.0 to 0.0 in a small number of pixels), then there will be visual anomalies at the edges of the resultant anti-aliased triangle. The workaround for this ‘feature’ is to reduce the gradient by increasing small color components and decreasing large ones. This can be demonstrated by changing the values of `maxColor` and `minColor` in `test25` of the Glide distribution. Note that this ‘feature’ is only present when the color combine mode includes iterated RGB or alpha as one of the parameters in the final color.

`grAADrawTriangle` is independent of the GR_AA_ORDERED mode that enables anti-aliasing.

SEE ALSO

`grAlphaBlendFunction, grAlphaCombine, grDrawTriangle, grEnable`
NAME

`grAlphaBlendFunction` – specify the alpha blending function

C SPECIFICATION

```c
void grAlphaBlendFunction( GrAlphaBlendFnc_t rgb_sf,
                           GrAlphaBlendFnc_t rgb_df,
                           GrAlphaBlendFnc_t alpha_sf,
                           GrAlphaBlendFnc_t alpha_df);
```

PARAMETERS

- `rgb_sf` Specifies the red, green, and blue source blending factors. The following symbolic constants are accepted:
  - `GR_BLEND_ZERO`
  - `GR_BLEND_ONE`
  - `GR_BLEND_DST_COLOR`
  - `GR_BLEND_ONE_MINUS_DST_COLOR`
  - `GR_BLEND_SRC_ALPHA`
  - `GR_BLEND_ONE_MINUS_SRC_ALPHA`
  - `GR_BLEND_DST_ALPHA`
  - `GR_BLEND_ONE_MINUS_DST_ALPHA`
  - `GR_BLEND_ALPHA_SATURATE`

- `rgb_df` Specifies the red, green, and blue destination blending factors. The following symbolic constants are accepted:
  - `GR_BLEND_ZERO`
  - `GR_BLEND_ONE`
  - `GR_BLEND_SRC_COLOR`
  - `GR_BLEND_ONE_MINUS_SRC_COLOR`
  - `GR_BLEND_SRC_ALPHA`
  - `GR_BLEND_ONE_MINUS_SRC_ALPHA`
  - `GR_BLEND_DST_ALPHA`
  - `GR_BLEND_ONE_MINUS_DST_ALPHA`
  - `GR_BLEND_PREFOG_COLOR`

- `alpha_sf` Specifies the alpha source blending factor. The following symbolic constants are accepted:
  - `GR_BLEND_ZERO`
  - `GR_BLEND_ONE`

- `alpha_df` Specifies the alpha destination blending factor. The following symbolic constants are accepted:
  - `GR_BLEND_ZERO`
  - `GR_BLEND_ONE`

DESCRIPTION

Alpha blending blends the RGBA values for rendered pixels (source) with the RGBA values that are already in the frame buffer (destination). `grAlphaBlendFunction` defines the operation of blending. `rgb_sf` and `alpha_sf` specify which of nine methods is used to scale the source color and alpha components. `rgb_df` and `alpha_df` specify which of eight methods is used to scale the destination color and alpha components.

Alpha blending is defined by the equations:
\[ R = \min(255, R_s + R_d d_d) \]
\[ G = \min(255, G_s + G_d d_g) \]
\[ B = \min(255, B_s + B_d d_b) \]
\[ A = \min(255, A_s + A_d d_a) \]

where \(R_s, G_s, B_s, A_s\) are the source color and alpha components, \(R_d, G_d, B_d, A_d\) are the destination color and alpha components, \(s_R, s_G, s_B, s_A\) are the source blending factors, and \(d_R, d_G, d_B, d_A\) are the destination blending factors.

The blending factors are as follows:

<table>
<thead>
<tr>
<th>blending factor</th>
<th>component blend factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_BLEND_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_BLEND_ONE</td>
<td>1</td>
</tr>
<tr>
<td>GR_BLEND_SRC_COLOR</td>
<td>( C_s / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_ONE_MINUS_SRC_COLOR</td>
<td>( 1 - C_s / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_DST_COLOR</td>
<td>( C_d / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_ONE_MINUS_DST_COLOR</td>
<td>( 1 - C_d / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_SRC_ALPHA</td>
<td>( A_s / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_ONE_MINUS_SRC_ALPHA</td>
<td>( 1 - A_s / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_DST_ALPHA</td>
<td>( A_d / 255 )</td>
</tr>
<tr>
<td>GR_BLEND_ALPHA_SATURATE</td>
<td>( \min(A_s / 255, 1 - A_d / 255) )</td>
</tr>
<tr>
<td>GR_BLEND_PREFOG_COLOR</td>
<td>color before fog is applied</td>
</tr>
</tbody>
</table>

where \(C_s\) and \(C_d\) are the corresponding \(R_s, G_s, B_s, A_s\) and \(R_d, G_d, B_d, A_d\) components respectively.

To disable alpha blending, call
\[
\text{grAlphaBlendFunction}(\text{GR_BLEND_ONE, GR_BLEND_ZERO, GR_BLEND_ONE, GR_BLEND_ZERO})
\]

**NOTES**

The source of incoming alpha and color are determined by \text{grAlphaCombine} and \text{grColorCombine} respectively.

Alpha blending that requires a destination alpha is mutually exclusive with depth buffering. Attempting to use \text{GR_BLEND_DST_ALPHA}, \text{GR_BLEND_ONE_MINUS_DST_ALPHA}, or \text{GR_BLEND_ALPHA_SATURATE} when depth buffering is enabled will have undefined results.

On Voodoo Graphics systems, alpha blending and triple buffering are mutually exclusive as well.

\text{GR_BLEND_PREFOG_COLOR} is useful when applying fog to a scene generated in multiple passes. See the \textit{Glide Programming Guide} for more information.

**SEE ALSO**

\text{grAADrawTriangle, grAlphaCombine, grColorCombine}
**grAlphaCombine**

**NAME**

`grAlphaCombine` – configure the alpha combine unit

**C SPECIFICATION**

```c
void grAlphaCombine( GrCombineFunction_t func,
                   GrCombineFactor_t factor,
                   GrCombineLocal_t local,
                   GrCombineOther_t other,
                   FxBool invert )
```

**PARAMETERS**

`func` Specifies the function used in source alpha generation. Valid parameters are described below. The combine function names are prefixed with the string “GR_COMBINE_FUNCTION_”: e.g. `GR_COMBINE_FUNCTION_ZERO` or `GR_COMBINE_FUNCTION_BLEND_LOCAL`.

<table>
<thead>
<tr>
<th>combine function <code>func</code></th>
<th>computed alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>0</td>
</tr>
<tr>
<td>LOCAL</td>
<td>$A_{local}$</td>
</tr>
<tr>
<td>LOCAL_ALPHA</td>
<td>$A_{local}$</td>
</tr>
<tr>
<td>SCALE_OTHER</td>
<td>$f \times A_{other}$</td>
</tr>
<tr>
<td>BLEND_OTHER</td>
<td>$f \times A_{other} + A_{local}$</td>
</tr>
<tr>
<td>SCALE_OTHER_ADD_LOCAL</td>
<td>$f \times (A_{other} - A_{local})$</td>
</tr>
<tr>
<td>SCALE_OTHER_ADD_LOCAL_ALPHA</td>
<td>$f \times (A_{other} - A_{local}) + A_{local}$</td>
</tr>
<tr>
<td>SCALE_OTHER_MINUS_LOCAL</td>
<td>$(f \times A_{other}) - A_{local}$</td>
</tr>
<tr>
<td>BLEND</td>
<td>$(f \times A_{other}) + (1-f) \times A_{local}$</td>
</tr>
<tr>
<td>SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL</td>
<td>$(f \times (A_{other} - A_{local}) + A_{local})$</td>
</tr>
<tr>
<td>BLEND_LOCAL</td>
<td>$(f \times (-A_{local}) + A_{local})$</td>
</tr>
<tr>
<td>SCALE_MINUS_LOCAL_ADD_LOCAL</td>
<td>$(1-f) \times A_{local}$</td>
</tr>
<tr>
<td>BLEND_LOCAL</td>
<td>$(f \times (-A_{local}) + A_{local})$</td>
</tr>
</tbody>
</table>
**factor**

Specifies the scaling factor used in alpha generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combination factor</th>
<th>scale factor ((f))</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_COMBINE_FACTOR_NONE</code></td>
<td>Unspecified</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ZERO</code></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_LOCAL</code></td>
<td>(A_{\text{local}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_OTHER_ALPHA</code></td>
<td>(A_{\text{other}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_LOCAL_ALPHA</code></td>
<td>(A_{\text{local}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_TEXTURE_ALPHA</code></td>
<td>(A_{\text{textrue}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ONE</code></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL</code></td>
<td>(1 - A_{\text{local}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA</code></td>
<td>(1 - A_{\text{other}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA</code></td>
<td>(1 - A_{\text{local}} / 255)</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_FACTOR_ONE_MINUS_TEXTURE_ALPHA</code></td>
<td>(1 - A_{\text{textrue}} / 255)</td>
<td></td>
</tr>
</tbody>
</table>

**local**

Specifies the local alpha used in source alpha generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>local combine source</th>
<th>local alpha ((A_{\text{local}}))</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_COMBINE_LOCAL_NONE</code></td>
<td>Unspecified alpha.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_LOCAL_ITERATED</code></td>
<td>Iterated vertex alpha.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_LOCAL_CONSTANT</code></td>
<td>Constant alpha.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_LOCAL_DEPTH</code></td>
<td>High 8 bits from iterated vertex (z).</td>
<td></td>
</tr>
</tbody>
</table>

**other**

Specifies the other alpha used in source alpha generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>other combine source</th>
<th>other alpha ((A_{\text{other}}))</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_COMBINE_OTHER_NONE</code></td>
<td>Unspecified alpha.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_OTHER_ITERATED</code></td>
<td>Iterated vertex alpha.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_OTHER_TEXTURE</code></td>
<td>Alpha from texture map.</td>
<td></td>
</tr>
<tr>
<td><code>GR_COMBINE_OTHER_CONSTANT</code></td>
<td>Constant alpha.</td>
<td></td>
</tr>
</tbody>
</table>

**invert**

Specifies whether the generated alpha should be bitwise inverted as a final step.

**DESCRIPTION**

`grAlphaCombine` configures the alpha combine unit of the graphics subsystem’s hardware pipeline. This provides a low level mechanism for controlling all rendering modes within the hardware without manipulating individual register bits. The alpha combine unit computes the source alpha for the remainder of the rendering pipeline. The default mode is

```
grAlphaCombine( GR_COMBINE_FUNCTION_SCALE_OTHER, GR_COMBINE_FACTOR_ONE,  
GR_COMBINE_LOCAL_NONE, GR_COMBINE_OTHER_CONSTANT, FXFALSE); 
```

The alpha combine unit computes the function specified by the combine function on the inputs specified by the local combine source, other combine source, and the combine scale factor. The result is clamped to \([0..255]\), and then a bitwise inversion may be applied, controlled by the `invert` parameter.

The constant color parameters are the colors passed to `grConstantColorValue`. If the texture has no alpha component, then texture alpha is 255.

`grAlphaCombine` also keeps track of required vertex parameters for the rendering routines.

`GR_COMBINE_FACTOR_NONE`, `GR_COMBINE_LOCAL_NONE`, and `GR_COMBINE_OTHER_NONE` are provided
to indicate that no parameters are required. Currently they are the same as \texttt{GR\_COMBINE\_FACTOR\_ZERO}, \texttt{GR\_COMBINE\_LOCAL\_CONSTANT}, and \texttt{GR\_COMBINE\_OTHER\_CONSTANT} respectively.

\textbf{NOTES}

The local alpha value specified by the \textit{local} parameter and the other alpha value specified by the \textit{other} parameter are used by the color combine unit.

Inverting the bits in a color is the same as computing $1.0 - \text{color}$ for floating point color values in the range $[0..1]$ or $255 - \text{color}$ for 8-bit color values in the range $[0..255]$.

\textbf{SEE ALSO}

\texttt{grColorCombine}, \texttt{grConstantColorValue}, \texttt{grDrawTriangle}
**NAME**

`grAlphaControlsITRGBLighting` – enables/disables alpha controlled lighting

**C SPECIFICATION**

```c
void grAlphaControlsITRGBLighting(FxBool enable)
```

**PARAMETERS**

- `enable` Specifies whether the mode is enabled or disabled.

**DESCRIPTION**

When enabled, the normal color combine controls for local color ($C_{local}$) are overridden, and the most significant bit of texture alpha ($A_{texture}$) selects between iterated vertex RGB and the constant color set by `grConstantColorValue`. By default, alpha controlled lighting mode is disabled.

<table>
<thead>
<tr>
<th><code>value of enable</code></th>
<th><code>high order bit of alpha channel</code></th>
<th><code>color combine local color</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>FXTRUE</td>
<td>0</td>
<td>iterated RGB</td>
</tr>
<tr>
<td>FXTRUE</td>
<td>1</td>
<td><code>grConstantColorValue</code></td>
</tr>
<tr>
<td>FXFALSE</td>
<td>0</td>
<td>set by <code>grColorCombine</code></td>
</tr>
<tr>
<td>FXFALSE</td>
<td>1</td>
<td>set by <code>grColorCombine</code></td>
</tr>
</tbody>
</table>

**NOTES**

Some possible uses for this mode are self-lit texels and specular paint. If a texture contains texels that represent self-luminous areas, such as windows, then multiplicative lighting can be disabled for these texels as follows. Choose a texture format that contains one bit of alpha and set the alpha for each texel to 1 if the texel is self-lit. Set the Glide constant color to white and enabled alpha controlled lighting mode. Finally, set up texture lighting by multiplying the texture color by iterated RGB where iterated RGB is the local color in the color combine unit. When a texel’s alpha is 0, the texture color will be multiplied by the local color which is iterated RGB. This applies lighting to the texture. When a texel’s alpha is 1, the texture color will be multiplied by the Glide constant color which was previously set to white, so no lighting is applied.

If the color combine unit is configured to add iterated RGB to a texture for the purpose of a specular highlight, then texture alpha can be used as specular paint. In this example, the Glide constant color is set to black and iterated RGB iterates the specular lighting. If a texel’s alpha is 0, the texture color will be added to iterated RGB, and specular lighting is applied to the texture. If the texel’s alpha is 1, the texture color will be added to the Glide constant color which was previously set to black, so no lighting is applied. The result is that the alpha channel in the texture controls where specular lighting is applied to the texture and specularity can be painted onto the texture in the alpha channel.

**SEE ALSO**

`grColorCombine`, `grConstantColorValue`
**grAlphaTestFunction**

**NAME**

*grAlphaTestFunction* – specify the alpha test function

**C SPECIFICATION**

```c
void grAlphaTestFunction( GrCmpFnc_t function )
```

**PARAMETERS**

- `function` The new alpha comparison function.

**DESCRIPTION**

The alpha test discards pixels depending on the outcome of a comparison between the incoming alpha value and a constant reference value. *grAlphaTestFunction* specifies the comparison function and *grAlphaTestReferenceValue* specifies the constant reference value.

The incoming alpha value is compared to the constant alpha test reference value using the function specified by *function*. If the comparison passes, the pixel is drawn, conditional on subsequent tests, such as depth buffer and chroma-key. If the comparison fails, the pixel is not drawn. The default function is *GR_CMP_ALWAYS*.

The comparison functions are as follows:

<table>
<thead>
<tr>
<th>function</th>
<th>comparison function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_CMP_NEVER</td>
<td>Never passes.</td>
</tr>
<tr>
<td>GR_CMP_LESS</td>
<td>Passes if the incoming alpha value is less than the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_EQUAL</td>
<td>Passes if the incoming alpha value is equal to the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_LEQUAL</td>
<td>Passes if the incoming alpha value is less than or equal to the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_GREATER</td>
<td>Passes if the incoming alpha value is greater than the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_NOTEQUAL</td>
<td>Passes if the incoming alpha value is not equal to the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_GEQUAL</td>
<td>Passes if the incoming alpha value is greater than or equal to the constant alpha reference value.</td>
</tr>
<tr>
<td>GR_CMP_ALWAYS</td>
<td>Always passes.</td>
</tr>
</tbody>
</table>

Alpha testing is performed on all pixel writes, including those resulting from scan conversion of points, lines, and triangles, and from direct linear frame buffer writes. Alpha testing is implicitly disabled during linear frame buffer writes if the pixel pipeline is disabled (see *grLfbLock* and *grLfbWriteRegion*).

**NOTES**

- The incoming alpha is the output of the alpha combine unit that is configured with *grAlphaCombine*.

**SEE ALSO**

- *grAlphaCombine*, *grAlphaTestReferenceValue*, *grLfbLock*, *grLfbWriteRegion*
grAlphaTestReferenceValue

NAME

grAlphaTestReferenceValue – specify the alpha test reference value

C SPECIFICATION

void grAlphaTestReferenceValue ( GrAlpha_t value )

PARAMETERS

value The new alpha test reference value.

DESCRIPTION

The alpha test discards pixels depending on the outcome of a comparison between the pixel’s incoming alpha value and a constant reference value. grAlphaTestFunction specifies the comparison function and grAlphaTestReferenceValue specifies the constant reference value. The default reference value is 0x00.

The incoming alpha value is compared to the value using the function specified by grAlphaTestFunction. If the comparison passes, the pixel is drawn, conditional on subsequent tests such as depth buffer and chroma-key. If the comparison fails, the pixel is not drawn.

Alpha testing is performed on all pixel writes, including those resulting from scan conversion of points, lines, and triangles, and from direct linear frame buffer writes. Alpha testing is implicitly disabled during linear frame buffer writes if the pixel pipeline is disabled (see grLfbLock and grLfbWriteRegion).

NOTES

The incoming alpha is the output of the alpha combine unit that is configured with grAlphaCombine.

SEE ALSO

grAlphaCombine, grAlphaTestReferenceValue, grLfbLock, grLfbWriteRegion
NAME

gBufferClear – clear the buffers to the specified values

C SPECIFICATION

void gBufferClear ( GrColor_t color, GrAlpha_t alpha, FxU32 depth )

PARAMETERS

color
The color value used for clearing the draw buffer.

alpha
The alpha value used for clearing the alpha buffer (ignored if alpha buffering is not enabled, i.e. a destination alpha is not specified in gAlphaBlendFunction).

depth
An unsigned value used for clearing the depth buffer (ignored if depth buffering is not enabled).

DESCRIPTION

Clears the appropriate buffers with the given values. gClipWindow defines the area within the buffer to be cleared. Any buffers that are enabled are cleared by gBufferClear. For example, if depth buffering is enabled, the depth buffer will be cleared. If an application does not want a buffer to be cleared, then it should mask off writes to the buffer using gDepthMask and/or gColorMask as appropriate.

Although color, alpha, and depth parameters are always specified, the parameters actually used will depend on the current configuration of the hardware; the irrelevant parameters are ignored.

The minimum and maximum values for depth buffer data can be retrieved by calling gGet (GR_ZDEPTH_MIN_MAX,...) or gGet (GR_W_EPTH_MIN_MAX,...). The depth parameter can be one of these values or a direct representation of a value in the depth buffer. In the latter case the value is either a 1/z value (for GR_DEPTHBUFFER_ZBUFFER mode) or a 16-bit floating point format q value (for GR_DEPTHBUFFER_WBUFFER mode). The 16-bit floating point format is described in detail in the Glide Programming Manual.

NOTES

A buffer clear fills pixels much faster than triangle rendering; thus clearing the buffer is much faster than rendering a screen-sized rectangle. Clearing buffers is not always necessary, however, and should be avoided if possible. When depth buffering is disabled and every visible pixel is rendered each frame, simply draw each frame on top of whatever was previously in the frame buffer. When depth buffering is enabled, a sorted background that covers the entire area can be drawn with the depth buffer compare function set to GR_CMP_ALWAYS so that all pixel colors and depth values are replaced, and then normal depth buffering can be resumed.

The constants GR_ZDEPTHVALUE_NEAREST and GR_ZDEPTHVALUE_FARTHEST assume that depth values decrease as they get further away from the eye. However, any linear function of 1/z can be used for computing depth buffer values and therefore they can either increase or decrease with distance from the eye.

SEE ALSO

gClipWindow, gColorMask, gDepthMask, gGet, gRenderBuffer
NAME

gbufferswap – exchange front and back buffers

C SPECIFICATION

void gbufferswap(int swap_interval)

PARAMETERS

swap_interval The number of vertical retraces to wait before swapping the front and back buffers.

DESCRIPTION

gbufferswap exchanges the front and back buffers in the graphics subsystem after swap_interval vertical retraces. If the swap_interval is 0, then the buffer swap does not wait for vertical retrace. Otherwise, the buffers are swapped after swap_interval vertical retraces. For example, if the monitor frequency is 60 Hz, a swap_interval of 3 results in a maximum frame rate of 20 Hz.

The exchange takes place during the next vertical retrace of the monitor, rather than immediately after gbufferswap is called. If the application is double buffering, the graphics subsystem will stop rendering and wait until the swap occurs before executing more commands. If the application is triple buffering and the third rendering buffer is available, rendering commands will take place immediately in the third buffer.

NOTES

A swap_interval of 0 may result in visual artifacts, such as ‘tearing’, since a buffer swap can occur during the middle of a screen refresh cycle. This setting is very useful in performance monitoring situations, as true rendering performance can be measured without including the time buffer swaps spend waiting for vertical retrace.

gbufferswap waits until there are fewer than 7 pending buffer swap requests in the graphics command FIFO before returning.

SEE ALSO

gget
**grChromakeyMode**

**NAME**

grChromakeyMode – enable/disable hardware chroma-keying

**C SPECIFICATION**

```c
void grChromakeyMode( GrChromakeyMode_t mode )
```

**PARAMETERS**

- **mode** specifies whether chroma-keying should be enabled or disabled. Valid values are GR_CHROMAKEY_ENABLE and GR_CHROMAKEY_DISABLE.

**DESCRIPTION**

Enables and disables chroma-keying. When chroma-keying is enabled, color values are compared to a global chroma-key reference value (set by `grChromakeyValue`). If the pixel’s color is the same as the chroma-key reference value, the pixel is discarded. The chroma-key comparison takes place before the color combine function. By default, chroma-keying is disabled.

**NOTES**

The chroma-key comparison compares the chroma-key reference value to the other color specified in the configuration of the color combine unit.

**SEE ALSO**

grColorCombine, grChromakeyValue, grChromaRangeModeExt
**grChromakeyValue**

**NAME**

grChromakeyValue – set the global chroma-key reference value

**C SPECIFICATION**

void grChromakeyValue( GrColor_t value )

**PARAMETERS**

value The new chroma-key reference value.

**DESCRIPTION**

Sets the global chroma-key reference value as a packed RGBA value. The color format should be in the same format as specified in the cFormat parameter to grSstWinOpen.

**NOTES**

The chroma-key comparison compares the chroma-key reference value to the other color specified in the configuration of the color combine unit. The comparison is performed between colors with 24-bit precision; thus value must be set accordingly. See Table 10.1 in the Glide Programming Guide for details on how colors formats are expanded to 24 bits.

**SEE ALSO**

grColorCombine, grChromakeyMode, grChromaRangeExt
**NAME**

`grChromaRangeExt` – set the global chroma-range bounds and match criteria

**C SPECIFICATION**

```c
void grChromaRangeExt(GrColor_t color0, GrColor_t color1, FxU32 mode)
```

**PARAMETERS**

- `color0, color1` Independent range values for red, green, and blue.
- `mode` Chroma-range match criteria. Only one mode value is currently supported: `GR_CHROMARANGE_RGB_ALL_EXT`.

**DESCRIPTION**

`grChromaRangeExt` sets the global chroma-range reference values as order-insensitive packed RGBA values. The color format for `color0` and `color1` should be the same one as specified in the `cFormat` parameter to `grSstWinOpen`. The order in which range values are specified for a particular color component is irrelevant, i.e. the `{color0, color1}` pairs `{(130,36,87), (150,38,92)}` and `{(150,36,92), (130,38,87)}` are equivalent.

The `mode` parameter determines the way the color ranges are used in the chroma test. Only one value is currently supported, `GR_CHROMARANGE_RGB_ALL_EXT`. In this mode, each color component pair defines an inclusive range such that \( \text{lower bound} \leq \text{color} \leq \text{upper bound} \). If all components of the incoming pixel color fall within their ranges, the chroma test succeeds and the pixel is invalidated.

**NOTES**

Glide 3.0 is the first release to support `grChromaRangeExt`. The API is available only with hardware support. Use `grGetString(GR_EXTENSION,...)` and search for the sub-string “CHROMARANGE” to query for availability of this extension. If the extension is present, the entry point may be retrieved via `grGetProcAddr`.

The chroma-key comparison compares the chroma-key reference value to the other color specified in the configuration of the color combine unit. The comparison is performed between colors with 24-bit precision; thus `value` must be set accordingly. See Table 10.1 in the *Glide Programming Guide* for details on how colors formats are expanded to 24 bits.

Chroma ranging must be enabled before use. See `grChromaRangeModeExt`.

**SEE ALSO**

`grColorCombine, grChromakeyMode, grChromaRangeModeExt, grGetProcAddr`
**grChromaRangeModeExt**

**NAME**

`grChromaRangeModeExt` – enable or disable chroma range checking

**C SPECIFICATION**

```c
void grChromaRangeModeExt ( GrChromakeyMode_t mode )
```

**PARAMETERS**

- **mode** Chroma range checking enable flag. One of the two values:
  - GR_CHROMARANGE_DISABLE_EXT
  - GR_CHROMARANGE_ENABLE_EXT

**DESCRIPTION**

`grChromaRangeModeExt` enables and disables chroma range checking. Chroma ranging is independent of chroma-keying, but must be enabled before use.

**NOTES**

Glide 3.0 is the first release to support `grChromaRangeModeExt`. The API is available only with hardware support. Use `grGetString(GR_EXTENSION,...)` and search for the sub-string “CHROMARANGE” to query for availability of this extension. If the extension is present, the entry point may be retrieved via `grGetProcAddr`.

**SEE ALSO**

- `grColorCombine`, `grChromakeyMode`, `grChromaRangeExt`, `grGetString`, `grGetProcAddr`
grClipWindow

NAME

grClipWindow – set the size and location of the hardware clipping window

C SPECIFICATION

void grClipWindow(FxU32 minx, FxU32 miny, FxU32 maxx, FxU32 maxy)

PARAMETERS

minx The lower x screen coordinate of the clipping window.

miny The lower y screen coordinate of the clipping window.

maxx The upper x screen coordinate of the clipping window.

maxy The upper y screen coordinate of the clipping window.

DESCRIPTION

grClipWindow specifies the hardware clipping window. Any pixels outside the clipping window are rejected. Values are inclusive for minimum x and y values and exclusive for maximum x and y values. The clipping window also specifies the area grBufferClear clears.

At startup the default values for the clip window are the full size of the screen, e.g. (0,0,640,480) for 640x480 mode and (0,0,800,600) for 800x600 mode. To disable clipping simply sets the size of the clip window to the screen size. The clipping window should not be used for general purpose primitive clipping; since clipped pixels are processed but discarded; proper geometric clipping should be done by the application for best performance. The clipping window should be used to prevent stray pixels that appear from imprecise geometric clipping. Note that if the pixel pipeline is disabled (see grLfbLock), clipping is not performed on linear frame buffer writes.

NOTES

SEE ALSO

grBufferClear, grLfbLock
**NAME**

```
grColorCombine` – configure the color combine unit
```

**C SPECIFICATION**

```
void grColorCombine( GrCombineFunction_t func,
                    GrCombineFactor_t factor,
                    GrCombineLocal_t local,
                    GrCombineOther_t other,
                    FxBool invert)
```

**PARAMETERS**

- `func` Specifies the function used in source color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combine function func</th>
<th>computed color</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FUNCTION_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL</td>
<td>C local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL_ALPHA</td>
<td>A local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER</td>
<td>f * C other</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_BLEND_OTHER</td>
<td>f * C other + A other</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL</td>
<td>f * C other + C local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL</td>
<td>f * (C other - C local)</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL</td>
<td>f * (C other - C local) + C local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_BLEND_LOCAL</td>
<td>f * (– C local) + C local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA</td>
<td>f * (– C local) + A local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_TEXTURE_ALPHA</td>
<td>f * (– C local) + A texture</td>
</tr>
</tbody>
</table>

- `factor` Specifies the scaling factor f used in source color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>value for factor</th>
<th>scale factor used (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FACTOR_NONE</td>
<td>unspecified</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL</td>
<td>C local / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ALPHA</td>
<td>A other / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL_ALPHA</td>
<td>A local / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_TEXTURE_ALPHA</td>
<td>A texture / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE</td>
<td>1</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL</td>
<td>1 – C local / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_ALPHA</td>
<td>1 – A other / 255</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_TEXTURE_ALPHA</td>
<td>1 – A texture / 255</td>
</tr>
</tbody>
</table>
grColorCombine

Specifies the local color used in source color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>value for local</th>
<th>color used (C&lt;sub&gt;local&lt;/sub&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_LOCAL_NONE</td>
<td>Unspecified color.</td>
</tr>
<tr>
<td>GR_COMBINE_LOCAL_ITERATED</td>
<td>Iterated vertex color (Gouraud shading).</td>
</tr>
<tr>
<td>GR_COMBINE_LOCAL_CONSTANT</td>
<td>Constant color.</td>
</tr>
</tbody>
</table>

other

Specifies the other color used in source color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>value for other</th>
<th>color used (C&lt;sub&gt;other&lt;/sub&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_OTHER_NONE</td>
<td>Unspecified color.</td>
</tr>
<tr>
<td>GR_COMBINE_OTHER_ITERATED</td>
<td>Iterated vertex color (Gouraud shading).</td>
</tr>
<tr>
<td>GR_COMBINE_OTHER_TEXTURE</td>
<td>Color from texture map.</td>
</tr>
<tr>
<td>GR_COMBINE_OTHER_CONSTANT</td>
<td>Constant color.</td>
</tr>
</tbody>
</table>

invert

Specifies whether the generated source color should be bitwise inverted as a final step.

DESCRIPTION

gColorCombine configures the color combine unit of the graphics subsystem’s hardware pipeline. This provides a low level mechanism for controlling all modes of the color combine unit without manipulating individual register bits.

The color combine unit computes the function specified by the combine function on the inputs specified by the local combine source, other combine source, and the combine scale factor. The result is clamped to [0..255], and then a bitwise inversion may be applied, controlled by the invert parameter. The resulting color goes to the alpha and depth units.

The default color combine mode is

```c
grColorCombine( GR_COMBINE_FUNCTION_SCALE_OTHER,
                GR_COMBINE_FACTOR_ONE,
                GR_COMBINE_LOCAL_ITERATED,
                GR_COMBINE_OTHER_ITERATED,
                FXFALSE
              )
```

GR_COMBINE_LOCAL_CONSTANT and GR_COMBINE_OTHER_CONSTANT select the constant color specified in a previous call to grConstantColorValue. The iterated color selected by GR_COMBINE_LOCAL_ITERATED or GR_COMBINE_OTHER_ITERATED are the red, green, blue, and alpha values associated with a drawing primitive’s vertices.

grColorCombine also keeps track of required vertex parameters for the rendering routines. GR_COMBINE_FACTOR_NONE, GR_COMBINE_LOCAL_NONE, and GR_COMBINE_OTHER_NONE are provided to indicate that no parameters are required. Currently they are the same as GR_COMBINE_FACTOR_ZERO, GR_COMBINE_LOCAL_CONSTANT, and GR_COMBINE_OTHER_CONSTANT respectively.

NOTES

In the tables above, A<sub>local</sub> is the local alpha value selected by grAlphaCombine and A<sub>other</sub> is the other alpha value selected by grAlphaCombine.

Inverting the bits in a color is the same as computing (1.0 – color) for floating point color values in the range [0..1] or (255 – color) for 8-bit color values in the range [0..255].

SEE ALSO

grAlphaCombine, grConstantColorValue, grDrawTriangle
**NAME**

`grColorMask` – enable/disable writing into the color and alpha buffers

**C SPECIFICATION**

```c
void grColorMask( FxBool rgb, FxBool alpha )
```

**PARAMETERS**

- `rgb` The new color buffer mask.
- `alpha` The new alpha buffer mask.

**DESCRIPTION**

`grColorMask` specifies whether the color and/or alpha buffers can or cannot be written to during rendering operations. If `rgb` is `FXFALSE`, for example, no change is made to the color buffer regardless of the drawing operation attempted. The `alpha` parameter is ignored if depth buffering is enabled since the alpha and depth buffers share memory.

The value of `grColorMask` is ignored during linear frame buffer writes if the pixel pipeline is disabled (see `grLfbLock`). The default values are all `FXTRUE`, indicating that the associated buffers are writable.

**NOTES**

**SEE ALSO**

- `grBufferClear`, `grDepthMask`, `grLfbLock`
**grConstantColorValue**

**NAME**

`grConstantColorValue` – set the global constant color

**C SPECIFICATION**

```c
void grConstantColorValue( GrColor_t color )
```

**PARAMETERS**

- `color` The new constant color.

**DESCRIPTION**

Glide refers to a global constant color in the color combine unit and alpha combine unit if
`GR_COMBINE_LOCAL_CONSTANT` or `GR_COMBINE_OTHER_CONSTANT` are specified. This constant color is
set with `grConstantColorValue`. The color format should be in the same format as specified in the
cFormat parameter to `grSstWinOpen`. The default value is `0xFFFFFFFF`.

**NOTES**

**SEE ALSO**

- `grAlphaCombine`, `grColorCombine`
grCoordinateSpace

NAME

grCoordinateSpace – specify the coordinate space for vertices

C SPECIFICATION

void grCoordinateSpace( GrCoordinateSpaceMode_t mode )

PARAMETERS

mode Specifies the coordinate space for vertices. One of the following values:

<table>
<thead>
<tr>
<th>mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_CLIP_COORDS</td>
<td>Vertices are specified in clip coordinates.</td>
</tr>
<tr>
<td>GR_WINDOW_COORDS</td>
<td>Vertices are specified in native hardware device coordinates relative to the window origin.</td>
</tr>
</tbody>
</table>

DESCRIPTION

This API specifies the coordinate space for vertex coordinates. The mode parameter is one of the following:

When window coordinates are used, the application performs the coordinate divisions by w, providing x/w, y/w, z/w, 1/w, s/w, t/w, and q/w as necessary in the vertex structure (only x/w and y/w are mandatory). Window coordinates may be less than optimal on future hardware that can perform perspective division and viewport transformations.

When clip coordinates are used, the division by w is performed automatically. The minimal vertex specifies x, y, and w. If z buffering is enabled, z should be in the range [−w..+w]; otherwise, z data need not be given. Glide will automatically compute x/w, y/w, z/w, and 1/w, perform vertex snapping on the results, and then apply the viewport transformation to get window coordinates. Texture coordinates s and t are in the range [0..1] for all texture sizes and aspect ratios. Glide automatically computes s/w, t/w, and q/w.

NOTES

Glide 3.0 is the first release to support grCoordinateSpace.

SEE ALSO

grClipWindow, grDepthRange, grSstOrigin
**grCullMode**

**NAME**

`grCullMode` – set the cull mode

**C SPECIFICATION**

```c
void grCullMode ( GrCullMode_t mode )
```

**PARAMETERS**

`mode` The new culling mode. Valid parameters are `GR_CULL_DISABLE`, `GR_CULL_NEGATIVE`, and `GR_CULL_POSITIVE`.

**DESCRIPTION**

Specifies the type of backface culling, if any, that Glide performs when rendering a triangle. Glide computes the signed area of a triangle prior to rendering, and the sign of this area can be used for backface culling operations. If the sign of the area matches the `mode`, then the triangle is rejected. `grCullMode` assumes that `GR_CULL_POSITIVE` corresponds to a counter-clockwise oriented triangle when the origin is `GR_ORIGIN_LOWER_LEFT` and a clockwise oriented triangle when the origin is `GR_ORIGIN_TOP_LEFT`.

<table>
<thead>
<tr>
<th>location of the origin</th>
<th>triangle orientation</th>
<th>sign of area of culled triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_ORIGIN_LOWERLEFT</code></td>
<td>clockwise</td>
<td>negative</td>
</tr>
<tr>
<td><code>GR_ORIGIN_LOWERLEFT</code></td>
<td>counter-clockwise</td>
<td>positive</td>
</tr>
<tr>
<td><code>GR_ORIGIN_UPPERLEFT</code></td>
<td>clockwise</td>
<td>positive</td>
</tr>
<tr>
<td><code>GR_ORIGIN_UPPERLEFT</code></td>
<td>counter-clockwise</td>
<td>negative</td>
</tr>
</tbody>
</table>

**NOTES**

`grCullMode` has no effect on points and lines, but does effect all triangle rendering primitives including polygons.

**SEE ALSO**

`grDrawTriangle`, `grDrawVertexArray`, `grDrawVertexArrayContiguous`
**grDepthBiasLevel**

**NAME**

`grDepthBiasLevel` – set the depth bias level

**C SPECIFICATION**

```c
void grDepthBiasLevel(FxU32 level)
```

**PARAMETERS**

- `level` The new depth bias level.

**DESCRIPTION**

`grDepthBiasLevel` allows an application to specify a depth bias used when rendering coplanar polygons. Specifically, if two polygons are coplanar but do not share vertices, e.g. a surface detail polygon sits on top of a larger polygon, artifacts such as “poke through” may result. To remedy such artifacts an application should increment or decrement the depth bias level, as appropriate for the depth buffer mode and function, per coplanar polygon. For left handed coordinate systems where 0x0000 corresponds to “nearest to viewer” and 0xFFFF corresponds “farthest from viewer” depth bias levels should be decremented on successive rendering of coplanar polygons.

Depth biasing is mutually exclusive of linear frame buffer writes.

**NOTES**

When using a floating point depth buffer (a w buffer), the bias is added after a floating point depth is computed, and the add is an integer add.

In depth buffering modes `GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS` and `GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS`, the depth bias level specifies the value to compare depth buffer values against, and is not added to the source depth value when writing to the depth buffer. See `grDepthBufferMode` for more information.

**SEE ALSO**

- `grDepthBufferMode`, `grDepthMask`
**grDepthBufferFunction**

**NAME**

`grDepthBufferFunction` – specify the depth buffer comparison function

**C SPECIFICATION**

```c
void grDepthBufferFunction( GrCmpFnc_t func )
```

**PARAMETERS**

`func` The new depth comparison function.

**DESCRIPTION**

`grDepthBufferFunction` specifies the function used to compare each rendered pixel’s depth value with the depth value present in the depth buffer. The comparison is performed only if depth testing is enabled with `grDepthBufferMode`. The choice of depth buffer function is typically dependent upon the depth buffer mode currently active.

The valid comparison functions are as follows:

<table>
<thead>
<tr>
<th><code>func</code></th>
<th>comparison function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_CMP_NEVER</td>
<td>Never passes.</td>
</tr>
<tr>
<td>GR_CMP_LESS</td>
<td>Passes if the pixel’s depth value is less than the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_EQUAL</td>
<td>Passes if the pixel’s depth value is equal to the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_LEQUAL</td>
<td>Passes if the pixel’s depth value is less than or equal to the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_GREATER</td>
<td>Passes if the pixel’s depth value is greater than the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_NOTEQUAL</td>
<td>Passes if the pixel’s depth value is not equal to the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_GEQUAL</td>
<td>Passes if the pixel’s depth value is greater than or equal to the stored depth value.</td>
</tr>
<tr>
<td>GR_CMP_ALWAYS</td>
<td>Always passes.</td>
</tr>
</tbody>
</table>

The default comparison function is **GR_CMP_LESS**.

**NOTES**

**SEE ALSO**

`grDepthBufferMode, grDepthMask, grDepthBiasLevel, grLfbConstantDepth`
**NAME**

`grDepthBufferMode` – set the depth buffering mode

**C SPECIFICATION**

```c
void grDepthBufferMode( GrDepthBufferMode_t mode )
```

**PARAMETERS**

- **mode**
  - The new depth buffering mode.

**DESCRIPTION**

`grDepthBufferMode` specifies the type of depth buffering to be performed. Valid modes are `GR_DEPTHBUFFER_DISABLE`, `GR_DEPTHBUFFER_ZBUFFER`, `GR_DEPTHBUFFER_WBUFFER`, `GR_DEPTHBUFFER_ZBUFFERCOMPARE_TO_BIAS`, or `GR_DEPTHBUFFER_WBUFFERCOMPARE_TO_BIAS`.

If `GR_DEPTHBUFFER_ZBUFFER` or `GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS` is selected, then the graphics subsystem will perform 16-bit fixed point `z` buffering. If `GR_DEPTHBUFFER_WBUFFER` or `GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS` is selected, then the graphics subsystem will perform 16-bit floating point `w` buffering. By default the depth buffer mode is `GR_DEPTHBUFFER_DISABLE`. Refer to the Glide Programming Guide for more information about `w` and `z` buffering.

If `GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS` or `GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS` is selected, then the bias specified with `grDepthBiasLevel` is used as a pixel’s depth value for comparison purposes only. Depth buffer values are compared against the depth bias level and if the compare passes and the depth buffer mask is enabled, the pixel’s unbiased depth value is written to the depth buffer. This mode is useful for clearing beneath cockpits and other types of overlays without effecting either the color or depth values for the cockpit or overlay.

Consider the following example: the depth buffer is cleared to `0xFFFF` and a cockpit is drawn with a depth value of zero. Next, the scene beneath the cockpit is drawn with depth buffer compare function of `GR_CMP_LESS` rendering pixels only where the cockpit is not drawn. To clear the color and depth buffers underneath the cockpit without disturbing the cockpit, the area to be cleared is rendered using triangles (not `grBufferClear`) with the depth bias level set to zero, a depth buffer compare function of `GR_CMP_NOTEQUAL` and a depth buffer mode of `GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS` or `GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS`. All pixels whose previous depth buffer values are not equal to zero will be rendered and the depth buffer will be set to either unbiased `z` or `w` depending on the mode. Using this method, the color and depth buffers can be cleared to any desired value beneath a cockpit or overlay without effecting the cockpit or overlay. Or more desirably, sorted background polygons from the scene to be rendered that cover all of the visible area can be rendered in this mode, saving the time consuming clearing operation. After the depth buffer is cleared beneath the cockpit, the depth buffer mode is returned to either `GR_DEPTHBUFFER_ZBUFFER` or `GR_DEPTHBUFFER_WBUFFER` and the compare function is returned to its normal setting (`GR_CMP_LESS` in this example). Note that since this mode of clearing is performed using triangle rendering, the fill rate is one half that of a rectangle clear using `grBufferClear`. In the case where sorted background polygons are used to clear underneath the cockpit, this method should always be faster than the alternative of calling `grBufferClear` and then drawing the background polygons. In the case where background polygons are not used, both methods:

1. Clearing the buffers with `grBufferClear` and then repainting the cockpit.
2. Clearing beneath the cockpit with triangles and not repainting the cockpit.
should be compared and the faster method chosen. Avoiding a cockpit repaint is important; cockpits are
typically rendered with linear frame buffer writes and while the writes are individually fast, the process can
be lengthy if the cockpit covers many pixels.

Notes
Since alpha, depth, and triple buffering are mutually exclusive of each other, enabling depth buffering when
using either the alpha or triple buffer will have undefined results.

`GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS` and `GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS`
modes are not available in revision 1 of the Pixelfx chip (use `grGet` to obtain the revision number).

See Also

`grDepthBufferFunction`, `grDepthMask`, `grDepthBiasLevel`, `grGet`, `grLfbConstantDepth`
**grDepthMask**

**NAME**

`grDepthMask` – enable/disable writing into the depth buffer

**C SPECIFICATION**

```c
void grDepthMask(FxBool enable)
```

**PARAMETERS**

- `enable` The new depth buffer mask.

**DESCRIPTION**

`grDepthMask` specifies whether the depth buffer is enabled for writing. If `enable` is `FXFALSE`, depth buffer writing is disabled. Otherwise, it is enabled. Initially, depth buffer writing is disabled.

**NOTES**

Since the alpha and depth buffers share the same memory, `grDepthMask` should be called only if depth buffering is being used.

`grDepthMask` is ignored during linear frame buffer writes if the pixel pipeline is bypassed (see `grLfbLock`).

**SEE ALSO**

- `grBufferClear`, `grDepthBufferFunction`, `grDepthBufferMode`, `grDepthBiasLevel`, `grLfbConstantDepth`, `grLfbLock`
**grDepthRange**

**NAME**

`grDepthRange` – specify viewport depth range

**C SPECIFICATION**

```c
void grDepthRange(FxFloat near, FxFloat far)
```

**PARAMETERS**

- `near, far` range for depth values for z buffering

**DESCRIPTION**

`grDepthRange` specifies the viewport parameters for the depth component for z buffering. It is useful only when vertices are specified in clip coordinates.

If z buffering, clip-space z is in the range [-w, +w]. After division by w, z is in the range [-1, 1] which is mapped to the depth buffer according to `[near, far]`, where `[near=0, far=1]` represents the entire range of the depth buffer, regardless of how many bits are in the depth buffer.

If w buffering, `grDepthRange` is ignored.

**NOTES**

Glide 3.0 is the first release to support `grDepthRange`.

**SEE ALSO**

- `grCoordinateSpace`
**NAME**

`grDisable` – disable Glide operating modes

**C SPECIFICATION**

```c
void grDisable (GrEnableMode_t mode)
```

**PARAMETERS**

- `mode`: Glide operating mode. Valid values are `GR_AA_ORDERED`, `GR_ALLOW_MIPMAP_DITHER`, `GR_PASSTHRU`, `GR_SHAMELESS_PLUG`, and `GR_VIDEO_SMOOTHING`.

**DESCRIPTION**

`grDisable` disables various Glide operating modes. The `mode` parameter is one of the following:

<table>
<thead>
<tr>
<th><code>mode</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_AA_ORDERED</code></td>
<td>An anti-aliasing method that requires objects to be sorted by depth. This mode applies to all primitives except strips and fans.</td>
</tr>
<tr>
<td><code>GR_ALLOW_MIPMAP_DITHER</code></td>
<td><code>GR_MIPMAP_NEAREST_DITHER</code> mode. By default, this mode cannot be enabled with <code>grTexMipMapMode</code> because of the performance impact. Note that this does not actually set mipmap dithering; <code>grTexMipMapMode</code> must still be called.</td>
</tr>
<tr>
<td><code>GR_PASSTHRU</code></td>
<td>Pass through mode. When enabled, the graphics frame buffer will be displayed. When disabled, the VGA frame buffer will be displayed. (This feature replaces the now-obsolete <code>grSstControl</code> API). Pass through mode is not supported by all hardware configurations. Use <code>grGet (GR_SUPPORTS_PASSTHRU, ...)</code> to determine whether or not pass through mode is supported on the current system.</td>
</tr>
<tr>
<td><code>GR_VIDEO_SMOOTHING</code></td>
<td>Video smoothing mode. Enabling smoothing reduces dithering artifacts but may result in a slightly blurrier image. If the hardware does not support video smoothing, this function is a no-op.</td>
</tr>
<tr>
<td><code>GR_SHAMELESS_PLUG</code></td>
<td>The 3Dfx power shield shameless plug that is blended into each displayed frame.</td>
</tr>
</tbody>
</table>

**NOTES**

Glide 3.0 is the first release to support `grDisable`.

**SEE ALSO**

`grAADrawTriangle`, `grEnable`, `grGet`, `grTexMipMapMode`
**NAME**

_`grDisableAllEffects`_ – disable all special effects in the graphics subsystem

**C SPECIFICATION**

```c
void grDisableAllEffects(void)
```

**PARAMETERS**

none

**DESCRIPTION**

`grDisableAllEffects` disables all special effects (alpha blending, alpha testing, chroma-keying, fog, depth buffering) in the graphics subsystem with the exception of clipping, dithering, and the color/depth masks. Effects must be re-enabled individually.

**NOTES**

**SEE ALSO**

`grAlphaBlendFunction`, `grAlphaTestFunction`, `grChromakeyMode`, `grDepthBufferMode`, `grFogMode`
**grDitherMode**

**NAME**

`grDitherMode` – sets the dithering mode

**C SPECIFICATION**

```c
void grDitherMode( GrDitherMode_t mode )
```

**PARAMETERS**

`mode` The new dithering mode.

**DESCRIPTION**

`grDitherMode` selects the form of dithering used when converting 24-bit RGB values to the 16-bit RGB color buffer format. Valid values are `GR_DITHER_DISABLE`, `GR_DITHER_2x2`, and `GR_DITHER_4x4`.

- `GR_DITHER_DISABLE` forces a simple truncation, which may result in noticeable banding.
- `GR_DITHER_2x2` uses a 2x2 ordered dither matrix, and `GR_DITHER_4x4` uses a 4x4 ordered dither matrix.

The default dithering `mode` is `GR_DITHER_4x4`. `grDitherMode` is *not* affected by `grDisableAllEffects`.

**NOTES**

**SEE ALSO**
grDrawLine

NAME

grDrawLine – draw a one-pixel-wide arbitrarily oriented line

C SPECIFICATION

void grDrawLine( const void *a, const void *b )

PARAMETERS

a, b     Endpoints and attributes of the line.

DESCRIPTION

grDrawLine renders a one-pixel-wide, arbitrarily oriented line with the given endpoints. All current Glide attributes will affect the appearance of the line. The drawing mode set by the most recent call to grEnable determines the kind of line that will be drawn. For example, if GR_AA_ORDERED has been enabled, grDrawLine will draw an anti-aliased line. All current Glide attributes will affect the appearance of the line.

NOTES

The Glide 2.x routine grAADrawLine is obsolete in Glide 3.0. To draw an anti-aliased point, enable anti-aliasing by calling grEnable(GR_AA_ORDERED) before drawing the point with grDrawLine.

SEE ALSO

grDrawPoint, grDrawTriangle, grEnable
**grDrawPoint**

**NAME**

`grDrawPoint` – draw a point

**C SPECIFICATION**

```c
void grDrawPoint ( const void *pt )
```

**PARAMETERS**

`pt` Location and attributes of the point.

**DESCRIPTION**

`grDrawPoint` renders a single point. The drawing mode set by the most recent call to `grEnable` determines the kind of point that will be drawn. For example, if `GR_AA_ORDERED` has been enabled, `grDrawPoint` will draw an anti-aliased point. All current Glide attributes will affect the appearance of the point. If many points need to be rendered to the screen, e.g. a sprite, use linear frame buffer writes instead.

`pt` is a pointer to a vertex structure whose internal layout has been specified by calls to `grVertexLayout`.

**NOTES**

The Glide 2.x routine `grAADrawPoint` is obsolete in Glide 3.0. To draw an anti-aliased point, enable anti-aliasing by calling `grEnable(GR_AA_ORDERED)` before drawing the point with `grDrawPoint`.

**SEE ALSO**

`grDrawLine, grDrawTriangle, grEnable, grLfbLock, grVertexLayout`
**NAME**

`grDrawTriangle` – draw a triangle

**C SPECIFICATION**

```c
void grDrawTriangle(const void *a, const void *b, const void *c)
```

**PARAMETERS**

- `a`, `b`, `c` Location and attributes of the vertices defining the triangle.

**DESCRIPTION**

`grDrawTriangle` renders an arbitrarily oriented triangle. The drawing mode set by the most recent call to `grEnable` determines the kind of point that will be drawn. For example, if `GR_AA.ORDERED` has been enabled, `grDrawTriangle` will draw an anti-aliased triangle. All current Glide attributes will affect the appearance of the triangle.

The parameters `a`, `b`, and `c` are pointers to vertex structures whose internal layout has been specified by calls to `grVertexLayout`.

Triangles are rendered with the following filling rules:

1. Zero area triangles render zero pixels.
2. Pixels are rendered if and only if their center lies within the triangle.

A pixel center is within a triangle if it is inside all three of the edges. If a pixel center lies exactly on an edge, it is considered to be inside for the left and horizontal bottom (lower y coordinate) edges and outside for the right and horizontal top (higher y coordinate) edges. If a pixel is outside any edge, it is considered to be outside the triangle.

In the following picture, a pixel whose center is at the intersection of the 8 triangles is rendered only by triangle D. The center pixel lies on a right edge in triangles A, B, E, F, G, and H. In triangle C and H, the pixel lies exactly on a top edge (high Y). But in triangle D, the pixel lies exactly on the bottom and left edges and is therefore considered to be inside the triangle.

![Diagram of triangles](image)

These filling rules guarantee that perfect meshes will draw every pixel within the mesh once and only once.
NOTES

If GR_AA_ORDERED has been enabled, grDrawTriangle will anti-alias all three edges of the triangle. To selectively anti-alias edges, use grAADrawTriangle.

SEE ALSO

grAADrawTriangle, grDrawLine, grDrawPoint, grEnable, grVertexLayout
**grDrawVertexArray**

**NAME**

*grDrawVertexArray* – draw a list of by-vertex vertices

**C SPECIFICATION**

```c
void grDrawVertexArray (FxU32 mode, FxU32 count, void *pointers[])
```

**PARAMETERS**

- **mode**
  Type of vertices. Valid values are `GR_POINTS`, `GR_LINE_STRIP`, `GR_LINES`, `GR_POLYGON`, `GR_TRIANGLE_STRIP`, `GR_TRIANGLE_FAN`, `GR_TRIANGLES`, `GR_TRIANGLE_STRIP_CONTINUE`, and `GR_TRIANGLE_FAN_CONTINUE`.

- **count**
  Number of vertices to draw.

- **pointers**
  Array of pointers to the vertex data.

**DESCRIPTION**

*grDrawVertexArray* draws the vertices identified by `pointers`, according to `mode`. It assumes the by-vertex format introduced in Glide 3.0, whereby the parametric values associated with a single vertex are bundled together. The vertex layout has been established by a series of calls to `grVertexLayout`.

The mode argument determines the shape that will be drawn with the vertices, as shown below.

- **GR_POINTS**
- **GR_LINES**
- **GR_TRIANGLES**
- **GR_LINE_STRIP**
- **GR_TRIANGLE_STRIP**
- **GR_TRIANGLE_FAN**

Mode **GR_POLYGON** is inappropriate with the example vertex list, since they produce a non-convex polygon. If **GR_POLYGON** is specified, the points are drawn as a triangle fan and produce a different polygon than the one that results from connecting the vertices in sequence.

Mode **GR_LINES** expects a vertex list with an even number of vertices, and will ignore the last vertex in the example. Similarly, **GR_TRIANGLE** expects vertex triples and will ignore the last vertex in the example.

Two additional values are accepted for mode: **GR_TRIANGLE_STRIP_CONTINUE** and **GR_TRIANGLE_FAN_CONTINUE**. These two modes can be used in strictly limited circumstances: `grDrawVertexArray(GR_TRIANGLE_STRIP_CONTINUE, ...)` must follow immediately after a command to draw or continue a triangle strip. Likewise, `grDrawVertexArray(GR_TRIANGLE_FAN_CONTINUE, ...)` must follow immediately after a command to draw or continue a triangle fan.
**NOTES**

Glide 3.0 is the first release to support `grDrawVertexArray`. Six old routines for drawing polygons have been made obsolete by `grDrawVertexArray`: `grDrawPolygon`, `grDrawPlanarPolygon`; `grDrawPolygonVertexList`; `grDrawPlanarPolygonVertexList`; `grAADrawPolygon`; and `grAADrawPolygonVertexList`. The table below shows how to convert calls to the obsolete routines with calls to `grDrawVertexArray`. It assumes that the old `GrVertex` structure has been defined both syntactically and with calls to `grVertexLayout`.

<table>
<thead>
<tr>
<th>Old Method</th>
<th>New Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>grDrawPlanarPolygon(nVerts, ilist, vlist)</code></td>
<td><code>grDrawVertexArray(GR_POLYGON, nVerts, vlist sorted according to ilist)</code></td>
</tr>
<tr>
<td><code>grDrawPolygon(nVerts, ilist, vlist)</code></td>
<td><code>grDrawVertexArray(GR_POLYGON, nVerts, vlist sorted according to ilist)</code></td>
</tr>
<tr>
<td><code>grDrawPlanarPolygonVertexList(nVerts, vlist)</code></td>
<td><code>grDrawVertexArrayContiguous(GR_POLYGON, nVerts, vlist, sizeof(GrVertex))</code></td>
</tr>
<tr>
<td><code>grDrawPolygonVertexList(nVerts, vlist)</code></td>
<td><code>grDrawVertexArrayContiguous(GR_POLYGON, nVerts, vlist, sizeof(GrVertex))</code></td>
</tr>
<tr>
<td><code>grAADrawPolygon(nVerts, vlist)</code></td>
<td><code>grEnable(AA_ORDERED); grDrawVertexArray(GR_POLYGON, nVerts, vlist sorted according to ilist)</code></td>
</tr>
<tr>
<td><code>grAADrawPolygonVertexList(nVerts, vlist)</code></td>
<td><code>grEnable(AA_ORDERED); grDrawVertexArrayContiguous(GR_POLYGON, nVerts, vlist, sizeof(GrVertex))</code></td>
</tr>
</tbody>
</table>

**SEE ALSO**

`grDrawVertexArrayContiguous`, `grEnable`, `grVertexLayout`
grDrawVertexArrayContiguous

NAME

grDrawVertexArrayContiguous – draw a contiguous list of by-vertex vertices

C SPECIFICATION

void grDrawVertexArrayContiguous (FxU32 mode, FxU32 count, void *vertex, FxU32 stride)

PARAMETERS

mode Type of vertices. Valid values are GR_POINTS, GR_LINE_STRIP, GR_LINES,
GR_POLYGON, GR_TRIANGLE_STRIP, GR_TRIANGLE_FAN, GR_TRIANGLES,
GR_TRIANGLE_STRIP_CONTINUE, and GR_TRIANGLE_FAN_CONTINUE.

count Number of vertices to draw.

vertex Pointer to the first element of an array of vertex data.

stride Size of a single vertex in the array.

DESCRIPTION

This is a specialized variant of grDrawVertexArray to account for the common case of all the vertices
being in a linear array. The vertices are processed in order.

NOTES

Glide 3.0 is the first release to support grDrawVertexArrayContiguous. It is a generalization of the
now-obsolete grDrawPolygonVertexList.

SEE ALSO

grDrawVertexArray
**grEnable**

**NAME**

*grEnable* – enable Glide operating modes

**C SPECIFICATION**

```c
void grEnable ( GrEnableMode_t mode )
```

**PARAMETERS**

- **mode**
  Glide operating mode. Valid values are `GR_AA_ORDERED`, `GR_ALLOW_MIPMAP_DITHER`, `GR_PASSTHRU`, `GR_SHAMELESS_PLUG`, and `GR_VIDEO_SMOOTHING`.

**DESCRIPTION**

*grEnable* enables various Glide operating modes. The *mode* parameter is one of the following:

<table>
<thead>
<tr>
<th>mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_AA_ORDERED</code></td>
<td>Anti-aliasing method that requires objects to be sorted by depth. This mode applies to all primitives except strips and fans.</td>
</tr>
<tr>
<td><code>GR_ALLOW_MIPMAP_DITHER</code></td>
<td>Allow <code>GR_MIPMAP_NEAREST_DITHER</code> mode. By default, this mode cannot be enabled with <code>grTexMipMapMode</code> because of the performance impact. Note that this does not actually set mipmap dithering; <code>grTexMipMapMode</code> must still be called.</td>
</tr>
<tr>
<td><code>GR_PASSTHRU</code></td>
<td>Pass through mode. When enabled, the graphics frame buffer will be displayed. When disabled, the VGA frame buffer will be displayed. (This feature replaces the now-obsolete <code>grSstControl</code> API). Pass through mode is not supported by all hardware configurations. Use <code>grGet( GR_SUPPORTS_PASSTHRU, ... )</code> to determine whether or not pass through mode is supported on the current system.</td>
</tr>
<tr>
<td><code>GR_VIDEO_SMOOTHING</code></td>
<td>Video smoothing mode. Enabling smoothing reduces dithering artifacts but may result in a slightly blurrier image. If the hardware does not support video smoothing, this mode is a no-op.</td>
</tr>
<tr>
<td><code>GR_SHAMELESS_PLUG</code></td>
<td>The 3Dfx power shield shameless plug is blended into each displayed frame if this mode is enabled.</td>
</tr>
</tbody>
</table>

**NOTES**

Glide 3.0 is the first release to support *grEnable*.

**SEE ALSO**

*grAADrawTriangle, grDisable, grGet, grTexMipMapMode*
**grErrorSetCallback**

**NAME**

`grErrorSetCallback` – install a user-defined error-handler

**C SPECIFICATION**

```c
typedef void (*GrErrorCallbackFnc_t)( const char *string, FxBool fatal );
void grErrorSetCallback( GrErrorCallbackFnc_t fnc );
```

**PARAMETERS**

- `function` Pointer to a function to be called with all future errors.

**DESCRIPTION**

`grErrorSetCallback` allows an application to install a callback function to handle error messages generated internally by Glide. The callback function accepts a string describing the error and a flag indicating if the error is fatal or recoverable. `grErrorSetCallback` is relevant only for the debug build of Glide; the release build of Glide removes all internal parameter validation and error checking, thus the user installed callback will never be called.

**NOTES**

**SEE ALSO**
**grFinish**

**NAME**

`grFinish` – force completion of all outstanding graphics commands

**C SPECIFICATION**

```c
void grFinish ( void )
```

**PARAMETERS**

none

**DESCRIPTION**

Calling `grFinish` forces all previously issued Glide commands to complete: it does not return until all effects from previous commands are fully realized on the screen. `grFinish` should be used judiciously as it can have severe performance impacts if called too frequently.

**NOTES**

Glide 3.0 is the first release to support `grFinish`. It replaces the now-obsolete `grSstIdle`.

**SEE ALSO**

`grFlush`
**grFlush**

**NAME**

grFlush – flush the graphics FIFO

**C SPECIFICATION**

```c
void grFlush ( void )
```

**PARAMETERS**

none

**DESCRIPTION**

Calling **grFlush** forces all previously issued commands to begin execution, guaranteeing they will complete in finite time. However, they may not all be completed upon return. Use **grFinish** to guarantee command completion upon return.

**NOTES**

Glide 3.0 is the first release to support **grFlush**. It is a no-op in current hardware because commands are not buffered (they are FIFOed, and the FIFO is guaranteed to drain). Future hardware designs may utilize a buffer rather than a FIFO; in that case, this command will become necessary. Developers interested in writing upward-compatible software should start using **grFlush** now.

**SEE ALSO**

**grFinish**
**grFogColorValue**

**NAME**

`grFogColorValue` – set the global fog color

**C SPECIFICATION**

```c
void grFogColorValue( GrColor_t value )
```

**PARAMETERS**

- `value` The new global fog color.

**DESCRIPTION**

`grFogColorValue` specifies the global fog color to be used during fog blending operations. The color format should be in the same format as specified in the `cFormat` parameter to `grSstWinOpen`.

The fog operation blends the fog color ($C_{fog}$) with each rasterized pixel’s color ($C_{in}$) using a blending factor $f$. Factor $f$ is derived from a user downloaded fog table based on either the pixels’ $q$ or $f$ component, depending on the current `grFogMode`.

The new color is computed as follows:

$$C_{out} = f \cdot C_{fog} + (1-f) \cdot C_{in}$$

**NOTES**

Fog is applied after color combining and before alpha blending.

**SEE ALSO**

- `grDisableAllEffects`
- `grFogMode`
- `grFogTable`
**grFogMode**

**NAME**

`grFogMode` – enable/disable per-pixel fog blending operations

**C SPECIFICATION**

```c
void grFogMode( GrFogMode_t mode )
```

**PARAMETERS**

`mode` The new fog mode.

**DESCRIPTION**

`grFogMode` enables/disables fog blending operations. Valid parameters are `GR_FOG_DISABLE`, `GR_FOG_WITH_TABLE_ON_Q`, `GR_FOG_ADD2`, and `GR_FOG_MULT2`. The last two modes can be used in conjunction with the others to tailor the fog equation, as shown below. If the “FOGCOORD” extension is supported, then `GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT` is also permitted.

The fog operation blends the fog color ($c_{fog}$) with each rasterized pixel’s color ($c_{in}$) using a blending factor $f$. A value of $f = 0$ indicates minimum fog density and a value of $f = 255$ indicates maximum fog density. The new color is computed as follows:

$$c_{out} = f c_{fog} + (1-f)c_{in}$$

Factor $f$ is determined by `mode`. If `mode` is `GR_FOG_WITH_TABLE_ON_Q`, then $f$ is computed by interpolating between fog table entries, where the fog table is indexed with a floating point representation of the pixel’s $q$ (if using clip coordinates) or $w$ (if using window coordinates) component. If `mode` is `GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT`, then the fog table is indexed with a special vertex parameter, `GR_PARAM_FOG_EXT`. 
### if mode sets

<table>
<thead>
<tr>
<th>Mode</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_FOG_DISABLE</td>
<td>( c_{\text{out}} = c_{\text{in}} )</td>
</tr>
<tr>
<td>GR_FOG_WITH_TABLE_ON_Q</td>
<td>( c_{\text{out}} = f_{\text{fog}(q)} \cdot c_{\text{fog}} + (1 - f_{\text{fog}(q)}) \cdot c_{\text{in}} )</td>
</tr>
<tr>
<td>GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT</td>
<td>( c_{\text{out}} = f_{\text{fog}(v.fog)} \cdot c_{\text{fog}} + (1 - f_{\text{fog}(v.fog)}) \cdot c_{\text{in}} )</td>
</tr>
<tr>
<td>GR_FOG_MULT2</td>
<td>( c_{\text{out}} = f c_{\text{fog}} )</td>
</tr>
<tr>
<td>GR_FOG_ADD2</td>
<td>( c_{\text{out}} = (1 - f) c_{\text{in}} )</td>
</tr>
</tbody>
</table>

### Notes

Fog is applied after color combining and before alpha blending.

Mode modifiers GR_FOG_ADD2 and GR_FOG_MULT2 are useful when applying fog to scenes that require several passes to generate. See the Glide Programming Guide for more information.

### See Also

grFogColorValue, grFogTable
grFogTable

NAME

grFogTable – download a fog table

C SPECIFICATION

```c
void grFogTable(const GrFog_t table[])
```

PARAMETERS

`table` The new fog table.

DESCRIPTION

`grFogTable` downloads a new table of 8-bit values that are logically viewed as fog opacity values corresponding to various depths. The table entries control the amount of blending between the fog color and the pixel’s color. A value of 0x00 indicates no fog blending and a value of 0xFF indicates complete fog.

The fog operation blends the fog color (C\textsubscript{fog}) with each rasterized pixel’s color (C\textsubscript{in}) using a blending factor \( f \). Factor \( f \) depends upon the most recent call to \texttt{grFogMode}. If the \texttt{grFogMode} is set to \texttt{GR_FOG_WITH_TABLE_ON_Q}, the factor \( f \) is computed by interpolating between fog table entries, where the fog table is indexed with a floating point representation of the pixel’s \( q \) (if using clip coordinates) or \( w \) (if using window coordinates) component. The order of the entries within the fog table correspond roughly to their distance from the viewer. The exact world \( w \) corresponding to fog table entry \( i \) can be found by calling \texttt{guFogTableIndexToW(i)} or by computing:

\[
pow(2.0, 3.0 + (\text{double})(i >> 2)) / (8 - (i & 3));
\]

The new color is computed as follows:

\[
C_{out} = f \cdot C_{fog} + (1 - f) \cdot C_{in}
\]

An exponential fog table can be generated by computing \((1 - e^{-kw}) \cdot 255\) where \( k \) is the fog density and \( w \) is world distance. It is usually best to normalize the fog table so that the last entry is 255.

NOTES

In previous versions of Glide, fog tables had a fixed size of 64 entries. With Glide 3.0, the fog table size is variable and hardware dependent. Use \texttt{grGet(GR_FOG_TABLE_ENTRIES, \ldots)} to retrieve the size for your system configuration.

The difference between consecutive entries in the fog table must be less than 64.

Fog is applied after color combining and before alpha blending.

There are several Glide Utility APIs for generating fog tables.

SEE ALSO

\texttt{grFogMode, grFogColorValue, grGet, guFogTableIndexToW}
**NAME**

**grGet** – return Glide state

**C SPECIFICATION**

```c
FxU32 grGet (FxU32 *params)
```

**PARAMETERS**

- **pname**: Encoded selectors for the environmental parameters to be returned.
- **plength**: Length of the return buffer, in bytes.
- **params**: Pointer to a buffer in which the parameters will be returned.

**DESCRIPTION**

**grGet** retrieves the values of selected Glide state variables. If successful, it returns the number of bytes written into the **params** buffer. If **grGet** fails (invalid context, invalid **pname** or NULL **params**), 0 is returned and the contents of the **params** array is unchanged.

<table>
<thead>
<tr>
<th>selector encoded in <strong>pname</strong></th>
<th>number of values returned</th>
<th>number of bytes returned</th>
<th>description of value(s) returned in <strong>params</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_BITS_DEPTH</td>
<td>1</td>
<td>4</td>
<td>The number of bits of depth (z or w) in the frame buffer.</td>
</tr>
<tr>
<td>GR_BITS_RGBA</td>
<td>4</td>
<td>16</td>
<td>The number of bits each of red, green, blue, alpha in the frame buffer. If there is no separate alpha buffer (e.g. on Voodoo2, the depth buffer can be used as an alpha buffer), 0 will be returned for alpha bits.</td>
</tr>
<tr>
<td>GR_BITS_GAMMA</td>
<td>1</td>
<td>4</td>
<td>The number of bits for each channel in the gamma table. If gamma correction is not available, <strong>grGet</strong> will fail, and the <strong>params</strong> array will be unmodified.</td>
</tr>
<tr>
<td>GR_FIFO_FULLNESS</td>
<td>2</td>
<td>8</td>
<td>How full the FIFO is, as a percentage. The value is returned in two forms: 1.24 fixed point and a hardware-specific format.</td>
</tr>
<tr>
<td>GR_FOG_TABLE_ENTRIES</td>
<td>1</td>
<td>4</td>
<td>The number of entries in the hardware fog table.</td>
</tr>
<tr>
<td>GR_GAMMA_TABLE_ENTRIES</td>
<td>1</td>
<td>4</td>
<td>The number of entries in the hardware gamma table. Returns <strong>FXFALSE</strong> if it is not possible to manipulate gamma (e.g. on a Macronix card, or in windowed mode).</td>
</tr>
<tr>
<td>GR_GLIDE_STATE_SIZE</td>
<td>1</td>
<td>4</td>
<td>Size of buffer, in bytes, needed to save Glide state. See <strong>grGlideGetState</strong>.</td>
</tr>
<tr>
<td>GR_GLIDE_VERTEX_LAYOUT_SIZE</td>
<td>1</td>
<td>4</td>
<td>Size of buffer, in bytes, needed to save the current vertex layout. See <strong>grGlideGetVertexLayout</strong>.</td>
</tr>
<tr>
<td>GR_IS_BUSY</td>
<td>1</td>
<td>4</td>
<td>Returns <strong>FXFALSE</strong> if idle, <strong>FXTRUE</strong> if busy.</td>
</tr>
<tr>
<td>GR_LFB_PIXEL_PIPE</td>
<td>1</td>
<td>4</td>
<td>Returns <strong>FXTRUE</strong> if LFB writes can go through the 3D pixel pipe, <strong>FXFALSE</strong> otherwise.</td>
</tr>
<tr>
<td>selector encoded in <code>pname</code></td>
<td>number of values returned</td>
<td>number of bytes returned</td>
<td>description of value(s) returned in <code>params</code></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td><code>GR_MAX_TEXTURE_SIZE</code></td>
<td>1</td>
<td>4</td>
<td>The width of the largest texture supported on this configuration (e.g. Voodoo Graphics returns 256).</td>
</tr>
<tr>
<td><code>GR_MAX_TEXTURE_ASPECT_RATIO</code></td>
<td>1</td>
<td>4</td>
<td>The logarithm base 2 of the maximum aspect ratio supported for power-of-two, mipmap-able textures (e.g. Voodoo Graphics returns 3).</td>
</tr>
<tr>
<td><code>GR_MEMORY_FB</code></td>
<td>1</td>
<td>4</td>
<td>The total number of bytes per Pixelgfx chip if a non-UMA configuration is used, else 0. In non-UMA configurations, the total FB memory is <code>GR_MEMORY_FB * GR_NUM_FB</code>.</td>
</tr>
<tr>
<td><code>GR_MEMORY_TMU</code></td>
<td>1</td>
<td>4</td>
<td>The total number of bytes per Texelgfx chip if a non-UMA configuration is used, else <code>FXFALSE</code>. In non-UMA configurations, the total usable texture memory is <code>GR_MEMORY_TMU * GR_NUM_TMU</code>.</td>
</tr>
<tr>
<td><code>GR_MEMORY_UMA</code></td>
<td>1</td>
<td>4</td>
<td>The total number of bytes if a UMA configuration, else 0.</td>
</tr>
<tr>
<td><code>GR_NON_POWER_OF_TWO_TEXTURES</code></td>
<td>1</td>
<td>4</td>
<td>Returns <code>FXTRUE</code> if this configuration supports textures with arbitrary width and height (up to the maximum). Note that only power-of-two textures may be mipmapmed. Not implemented in the initial release of Glide 3.0.</td>
</tr>
<tr>
<td><code>GR_NUM_BOARDS</code></td>
<td>1</td>
<td>4</td>
<td>The number of installed boards supported by Glide. Valid before a call to <code>grSstWinOpen</code>.</td>
</tr>
<tr>
<td><code>GR_NUM_FB</code></td>
<td>1</td>
<td>4</td>
<td>The number of Pixelgfx chips present. This number will always be 1 except for SLI configurations.</td>
</tr>
<tr>
<td><code>GR_NUM_SWAP_HISTORY_BUFFER</code></td>
<td>1</td>
<td>4</td>
<td>Number of entries in the swap history buffer. Each entry is 4 bytes long.</td>
</tr>
<tr>
<td><code>GR_NUM_TMU</code></td>
<td>1</td>
<td>4</td>
<td>The number of Texelgfx chips per Pixelgfx chip. For integrated chips, the number of TMUs will be returned.</td>
</tr>
<tr>
<td><code>GR_PENDING_BUFFERSWAPS</code></td>
<td>1</td>
<td>4</td>
<td>The number of buffer swaps pending.</td>
</tr>
<tr>
<td><code>GR_REVISION_FB</code></td>
<td>1</td>
<td>4</td>
<td>The revision of the Pixelgfx chip(s).</td>
</tr>
<tr>
<td><code>GR_REVISION_TMU</code></td>
<td>1</td>
<td>4</td>
<td>The revision of the Texelgfx chip(s).</td>
</tr>
<tr>
<td><code>GR_STATS_LINES</code></td>
<td>1</td>
<td>4</td>
<td>The number of lines drawn.</td>
</tr>
<tr>
<td><code>GR_STATS_PIXELS_AFUNC_FAIL</code></td>
<td>1</td>
<td>4</td>
<td>The number of pixels that failed the alpha function test.</td>
</tr>
<tr>
<td><code>GR_STATS_PIXELS_CHROMA_FAIL</code></td>
<td>1</td>
<td>4</td>
<td>The number of pixels that failed the chroma key (or range) test.</td>
</tr>
<tr>
<td><code>GR_STATS_PIXELS_DEPTHFUNC_FAIL</code></td>
<td>1</td>
<td>4</td>
<td>The number of pixels that failed the depth buffer test.</td>
</tr>
<tr>
<td><code>GR_STATS_PIXELS_IN</code></td>
<td>1</td>
<td>4</td>
<td>The number of pixels that went into the pixel pipe, excluding buffer clears.</td>
</tr>
<tr>
<td><code>GR_STATS_PIXELS_OUT</code></td>
<td>1</td>
<td>4</td>
<td>The number of pixels that went out of the pixel pipe, including buffer clears.</td>
</tr>
<tr>
<td><code>GR_STATS_POINTS</code></td>
<td>1</td>
<td>4</td>
<td>The number of points drawn.</td>
</tr>
<tr>
<td>selector encoded in <code>pname</code></td>
<td>number of values returned</td>
<td>number of bytes returned</td>
<td>description of value(s) returned in <code>params</code></td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------</td>
<td>-------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>GR_STATS_TRIANGLES_IN</td>
<td>1</td>
<td>4</td>
<td>The number of triangles received.</td>
</tr>
<tr>
<td>GR_STATS_TRIANGLES_OUT</td>
<td>1</td>
<td>4</td>
<td>The number of triangles drawn.</td>
</tr>
<tr>
<td>GR_SWAP_HISTORY</td>
<td>(n)</td>
<td>(4n)</td>
<td>The <code>swapHistory</code> buffer contents. The (i^{th}) 4-byte entry counts the number of vertical syncs between the ((current\ frame - i)^{th}) frame and its predecessor. If <code>swapHistory</code> is not implemented (e.g. on Voodoo Graphics and Voodoo Rush), <code>grGet</code> will fail, and the <code>params</code> array will be unmodified. Use <code>grGet(GR_NUM_SWAP_HISTORY_BUFFER, ...)</code> to determine the number of entries in the buffer.</td>
</tr>
<tr>
<td>GR_SUPPORTS_PASSTHRU</td>
<td>1</td>
<td>4</td>
<td>Returns FXTRUE if pass through mode is supported. See <code>grEnable</code>.</td>
</tr>
<tr>
<td>GR_TEXTURE_ALIGN</td>
<td>1</td>
<td>4</td>
<td>The alignment boundary for textures. For example, if textures must be 16-byte aligned, 0x10 would be returned.</td>
</tr>
<tr>
<td>GR_VIDEO_POSITION</td>
<td>2</td>
<td>8</td>
<td>Vertical and horizontal beam location. Vertical retrace is indicated by (y == 0).</td>
</tr>
<tr>
<td>GR_VIEWPORT</td>
<td>4</td>
<td>16</td>
<td>(x, y, width, height).</td>
</tr>
<tr>
<td>GR_WDEPTH_MIN_MAX</td>
<td>2</td>
<td>8</td>
<td>The minimum and maximum allowable (w) buffer values.</td>
</tr>
<tr>
<td>GR_ZDEPTH_MIN_MAX</td>
<td>2</td>
<td>8</td>
<td>The minimum and maximum allowable (z) buffer values.</td>
</tr>
</tbody>
</table>

**NOTES**

Glide 3.0 is the first release to support `grGet`.

**SEE ALSO**

`grEnable`, `grGetProcAddress`, `grGetString`
**grGetProcAddress**

**NAME**

*grGetProcAddress* – get a pointer to an extension function

**C SPECIFICATION**

GrProc grGetProcAddress (char *procName)

**PARAMETERS**

procName The name of the desired procedure.

**DESCRIPTION**

*grGetProcAddress* retrieves a pointer to the function named in procName. Use *grGetString(GR_EXTENSION, ...)* to retrieve a list of supported extensions. If an extension is supported on the calling system, the extension name in the first column of the table can be found in the string returned by *grGetString*, and the procedure names found in the third column can be used as the procName argument to *grGetProcAddress*.

<table>
<thead>
<tr>
<th>extension name</th>
<th>description</th>
<th>associated procedure names</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHROMARANGE</td>
<td>Chroma range feature in the pixel pipeline is supported.</td>
<td>grChromaRangeModeExt, grChromaRangeExt</td>
</tr>
<tr>
<td>TEXCHROMA</td>
<td>Chroma range feature in the texture mapping unit is supported.</td>
<td>grTexChromaModeExt, grChromaRangeExt</td>
</tr>
<tr>
<td>TEXMIRROR</td>
<td><strong>GR_TEXTURE_CLAMP_MIRROR_EXT</strong> mode in grTexClampMode is supported.</td>
<td></td>
</tr>
<tr>
<td>PALETTE6666</td>
<td><strong>GR_TEXTURE_PALETTE_6666_EXT</strong> format is supported in grTexDownloadTable and grTexDownloadTablePartial.</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td><strong>GR_PARAM_FOG_EXT</strong> vertex parameter in grVertexLayout is supported and <strong>GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT</strong> is supported in grFogMode.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**

Glide 3.0 is the first release to support *grGetProcAddress*.

**SEE ALSO**

*grGetString*
**grGetString**

**NAME**

`grGetString` – return environment information

**C SPECIFICATION**

```c
const char *grGetString ( FxU32 name )
```

**PARAMETERS**

- **name**
  A selector that determines which string will be returned. Valid values are `GR_EXTENSION`, `GR_HARDWARE`, `GR_RENDERER`, `GR_VENDOR`, or `GR_VERSION`.

**DESCRIPTION**

`grGetString` returns a pointer to a text string, or NULL on error (invalid `name`).

<table>
<thead>
<tr>
<th>selector</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_EXTENSION</code></td>
<td>Returns a space-separated list of Glide extension names (the extension names themselves do not contain spaces). If no extensions are supported, a single space “ ” is returned.</td>
</tr>
<tr>
<td><code>GR_HARDWARE</code></td>
<td>Returns one of “Voodoo Graphics”, “Voodoo Rush”, “Voodoo2”, or “Banshee”. Other types may be added in the future.</td>
</tr>
<tr>
<td><code>GR_RENDERER</code></td>
<td>“Glide”.</td>
</tr>
<tr>
<td><code>GR_VENDOR</code></td>
<td>The vendor, “3Dfx Interactive”.</td>
</tr>
<tr>
<td><code>GR_VERSION</code></td>
<td>The Glide version. For example, “3.0”.</td>
</tr>
</tbody>
</table>

The following extensions are potentially available in Glide 3.0, with support dependent on the hardware capabilities of the system Glide is running on:

<table>
<thead>
<tr>
<th>extension name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHROMARANGE</td>
<td>Chroma range feature in the pixel pipeline is supported.</td>
</tr>
<tr>
<td>TEXCHROMA</td>
<td>Chroma range feature in the texture mapping unit is supported.</td>
</tr>
<tr>
<td>TEXMIRROR</td>
<td>GR_TEXTURECLAMP_MIRROR_EXT mode in grTexClampMode is supported.</td>
</tr>
<tr>
<td>PALETTE6666</td>
<td>GR_TEXTURE_PALETTE_6666_EXT format is supported in grTexDownloadTable and grTexDownloadTablePartial.</td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>GR_PARAM_FOG_EXT vertex parameter in grVertexLayout is supported and GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT is supported in grFogMode.</td>
</tr>
</tbody>
</table>

**NOTES**

Glide 3.0 is the first release to support `grGetString`.

**SEE ALSO**

`grGet`, `grGetProcAddress`
grGlideGetState

NAME

grGlideGetState – get the current state of the current graphics subsystem

C SPECIFICATION

void grGlideGetState( void *state )

PARAMETERS

state Pointer to a buffer where the state is to be stored.

DESCRIPTION

grGlideGetState makes a copy of the current state of the current graphics subsystem. This allows an application to save the state and then restore it later using grGlideSetState. Use

grGet (GR_GLIDE_STATE_SIZE, …) to determine how much space will be needed (and hence, how big the state buffer should be).

NOTES

SEE ALSO

grGet, grGlideSetState
**NAME**

grGlideGetVertexLayout – get the current vertex layout

**SPECIFICATION**

```c
void grGlideGetVertexLayout ( void *layout )
```

**PARAMETERS**

- `layout` Pointer to a buffer where the state is to be stored.

**DESCRIPTION**

grGlideGetVertexLayout makes a copy of the current vertex layout established by calls to grVertexLayout. This allows an application to save the layout and then restore it later using grGlideSetVertexLayout. Use `grGet(GR_GLIDE_VERTEXLAYOUT_SIZE,...)` to determine how much space will be needed (and hence, how big the layout buffer should be).

**NOTES**

Glide 3.0 is the first release to support grGlideGetVertexLayout.

**SEE ALSO**

grGet, grGlideSetVertexLayout, grVertexLayout
**NAME**

`grGlideInit` – initialize the Glide library

**C SPECIFICATION**

```c
void grGlideInit ( void )
```

**PARAMETERS**

none

**DESCRIPTION**

`grGlideInit` initializes the Glide library, performing tasks such as finding any installed graphics subsystems, allocating memory, and initializing state variables. `grGlideInit` must be called before any other Glide routines are called (the one exception is noted below).

**NOTES**

`grGet(GR_NUM_BOARDS,...)` is the only API that can be called before `grGlideInit`.

**SEE ALSO**

`grGet`, `grGlideShutdown`, `grSstWinOpen`, `grSstSelect`
grGlideSetState

NAME

grGlideSetState – set the state of the currently active graphics subsystem

C SPECIFICATION

void grGlideSetState( const void *state )

PARAMETERS

state Pointer to a buffer containing the new state.

DESCRIPTION

grGlideSetState sets the state of the currently active graphics subsystem. This API is typically paired with calls to grGlideGetState so that an application can save and restore the state.

SEE ALSO

grGlideGetState
grGlideSetVertexLayout

NAME

grGlideSetVertexLayout – set the current vertex layout

C SPECIFICATION

void grGlideSetVertexLayout ( const void *layout )

PARAMETERS

layout Pointer to a buffer containing a vertex layout previously saved by grGlideGetVertexLayout.

DESCRIPTION

grGlideSetVertexLayout restores the vertex layout previously saved by grGlideGetVertexLayout.

SEE ALSO

grGlideGetVertexLayout
**NAME**

`grGlideShutdown` – shut down the Glide library

**C SPECIFICATION**

```c
void grGlideShutdown(void)
```

**PARAMETERS**

none

**DESCRIPTION**

`grGlideShutdown` frees up any system resources allocated by Glide, including memory, and interrupt vectors. `grGlideShutdown` should be called immediately before program termination.

**NOTES**

**SEE ALSO**

`grGlideInit`
NAME

grLfbConstantAlpha – set the constant alpha value for linear frame buffer writes

C SPECIFICATION

void grLfbConstantAlpha ( GrAlpha_t alpha )

PARAMETERS

alpha                     The new constant alpha value.

DESCRIPTION

Some linear frame buffer write modes, specifically GR_LFBWRITEMODE_555, GR_LFBWRITEMODE_565, GR_LFBWRITEMODE_888, GR_LFBWRITEMODE_555_DEPTH, and GR_LFBWRITEMODE_565_DEPTH, do not contain alpha information. grLfbConstantAlpha specifies the alpha value for these linear frame buffer write modes. This alpha value is used if alpha testing and blending operations are performed during linear frame buffer writes. The default constant alpha value is 0xFF.

NOTES

If a linear frame buffer format contains alpha information, then the alpha supplied with the linear frame buffer write is used, and the constant alpha value set with grLfbConstantAlpha is ignored.

SEE ALSO

grAlphaTestFunction, grAlphaBlendFunction
**NAME**

`grLfbConstantDepth` – set the constant depth value for linear frame buffer writes

**C SPECIFICATION**

```c
void grLfbConstantDepth (FxU32 depth)
```

**PARAMETERS**

- `depth` The new constant depth value.

**DESCRIPTION**

Some linear frame buffer write modes, specifically `GR_LFBWRITEMODE_555`, `GR_LFBWRITEMODE_565`, `GR_LFBWRITEMODE_1555`, `GR_LFBWRITEMODE_888`, and `GR_LFBWRITEMODE_8888` do not include depth information. `grLfbConstantDepth` specifies the depth value for these linear frame buffer write modes. This depth value is used for depth buffering and fog operations and is assumed to be in a format suitable for the current depth buffering mode. The default constant depth value is `0x00`.

**NOTES**

If a linear frame buffer format contains depth information, then the depth supplied with the linear frame buffer write is used, and the constant depth value set with `grLfbConstantDepth` is ignored.

**SEE ALSO**

`grDepthBufferMode`, `grFogMode`
NAME

grlfbLock – lock a frame buffer in preparation for direct linear frame buffer accesses

C SPECIFICATION

FxBool grLfbLock( GrLock_t type, GrBuffer_t buffer, GrLfbWriteMode_t writeMode, GrOriginLocation_t origin, FxBool pixelPipeline, GrLfbInfo_t *info )

PARAMETERS

type Lock type.
buffer Buffer to lock.
writeMode Requested destination pixel format.
origin Requested y origin of linear frame buffer.
pixelPipeline If FXTRUE, send linear frame buffer writes through the pixel pipeline.
info Structure to be filled with pointer and stride info.

DESCRIPTION

When a Glide application desires direct access to a color or auxiliary buffer, it must lock that buffer in order to gain access to a pointer to the frame buffer data. This lock may assert a critical code section which effects process scheduling and precludes the use of GUI debuggers; therefore, time spent doing direct accesses should be minimized and the lock should be released as soon as possible using the grLfbUnlock API. An application may hold multiple simultaneous locks to various buffers, depending on the underlying hardware. Application software should always check the return value of grLfbLock and take into account the possibility that a lock may fail.

A lock type is a bit field created by the bit-wise OR of one read/write flag and an optional idle request flag. The read/write flag can be one of:

<table>
<thead>
<tr>
<th>read/write flag</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_LFB_READ_ONLY</td>
<td>info.lfbPtr should only be used for read access; writing to this pointer will have undefined effects on the graphics subsystem.</td>
</tr>
<tr>
<td>GR_LFB_WRITE_ONLY</td>
<td>info.lfbPtr should only be used for write access; reading from this pointer will yield undefined data.</td>
</tr>
</tbody>
</table>
The idle request flag can be one of:

<table>
<thead>
<tr>
<th>idle request flag</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_LFB_IDLE</td>
<td>The 3D engine will be idled before <code>grLfbLock</code> returns. This is the default behavior if no idle request flag is specified.</td>
</tr>
<tr>
<td>GR_LFB_NOIDLE</td>
<td>The 3D engine will not be idled; there is no guarantee of serialization of linear frame buffer accesses and triangle rendering or buffer clearing operations.</td>
</tr>
</tbody>
</table>

An application may attempt to lock any Glide buffer. Currently supported buffer designations are `GR_BUFFER_FRONTBUFFER`, `GR_BUFFER_BACKBUFFER`, and `GR_BUFFER_AUXBUFFER`.

Some 3Dfx hardware supports multiple write formats to the linear frame buffer space. An application may request a particular write format by passing a `writeMode` argument other than `GR_LFB_WRITEMODE_ANY`. If the destination pixel format specified is not supported on the target hardware, then the lock will fail. Supported pixel formats are:

<table>
<thead>
<tr>
<th>writeMode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_LFBWRITEMODE_565</td>
<td>Frame buffer accepts 16-bit RGB 565 pixel data.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_555</td>
<td>Frame buffer accepts 16-bit RGB-555 pixel data. The MSB of each pixel is ignored.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_1555</td>
<td>Frame buffer accepts 16-bit ARGB-1555 pixel data. The alpha component is replicated to 8-bits and copied to the alpha buffer if the alpha buffer has been enabled with <code>grColorMask</code>.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_888</td>
<td>Frame buffer accepts 24-bit RGB 888 pixel data packed into 32-bit words. The most significant byte of each word is ignored. If dithering is enabled, then color will be dithered down to the real frame buffer storage format if necessary.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_8888</td>
<td>Frame buffer accepts 32-bit ARGB 8888 pixel data. The alpha component is copied into the alpha buffer if the alpha buffer has been enabled with <code>grColorMask</code>. If dithering is enabled, then color will be dithered down to the real frame buffer storage format if necessary.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_565_DEPTH</td>
<td>Frame buffer accepts 32-bit pixels where the two most significant bytes contain 565 RGB data, and the two least significant bytes contain 16-bit depth data.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_555_DEPTH</td>
<td>Frame buffer accepts 32-bit pixels where the two most significant bytes contain 555 RGB data, the most significant bit is ignored, and the two least significant bytes contain 16-bit depth data.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_1555_DEPTH</td>
<td>Frame buffer accepts 32-bit pixels where the two most significant bytes contain 1555 ARGB data. The alpha component is replicated to 8-bits and copied to the alpha buffer, if alpha buffering has been enabled with <code>grColorMask</code>.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_2A16</td>
<td>Frame buffer accepts 16-bit auxiliary buffer values. This is the only <code>writeMode</code> that is valid when locking the auxiliary buffer. Alpha buffer values are taken from the 8 least significant bits of each sixteen bit word.</td>
</tr>
<tr>
<td>GR_LFBWRITEMODE_ANY</td>
<td>Lock will return the pixel format that most closely matches the true frame buffer storage format in the <code>info.writeMode</code>.</td>
</tr>
</tbody>
</table>

If the application specifies `GR_LFB_WRITEMODE_ANY` and the lock succeeds, the destination pixel format will be returned in `info.writeMode`. This default destination pixel format will always be the pixel format that most closely matches the true pixel storage format in the frame buffer. On Voodoo Graphics and
Voodoo Rush, this will always be `GR_LFBWRITEMODE_565` for color buffers and `GR_LFBWRITEMODE_ZA16` for the auxiliary buffer. The `writeMode` argument is ignored for read-only locks.

Some 3Dfx hardware supports a user specified y origin for LFB writes. An application may request a particular y origin by passing an `origin` argument other than `GR_ORIGIN_ANY`. If the `origin` specified is not supported on the target hardware, then the lock will fail. If the application specifies `GR_ORIGIN_ANY` and the lock succeeds, the LFB y origin will be returned in `info.origin`. The default y origin will always be `GR_ORIGIN_UPPER_LEFT` for LFB writes. Currently supported y origin values are:

<table>
<thead>
<tr>
<th>y origin</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_ORIGIN_UPPER_LEFT</td>
<td>Addressing originates in the upper left hand corner of the screen.</td>
</tr>
<tr>
<td>GR_ORIGIN_LOWER_LEFT</td>
<td>Addressing originates in the lower left hand corner of the screen.</td>
</tr>
<tr>
<td>GR_ORIGIN_ANY</td>
<td>Lock will always choose GR_ORIGIN_UPPER_LEFT.</td>
</tr>
</tbody>
</table>

Some 3Dfx hardware allows linear frame buffer writes to be processed through the same set of functions as those pixels generated by the triangle rasterizer. This feature is enabled by passing a value of `FXTRUE` in the `pixelPipeline` argument of `grLfbLock`. If the underlying hardware is incapable of processing pixels through the pixel pipeline, then the lock will fail. Use `grGet(GR_FLB_PIXEL_PIPE, …)` to determine whether or not LFB writes can utilize the pixel pipeline.

When enabled, color, alpha, and depth data from the linear frame buffer write will be processed as if they were generated by the triangle iterators. If the selected `writeMode` lacks depth information, then the value is derived from `grLfbConstantDepth`. If the `writeMode` lacks alpha information, then the value is derived from `grLfbConstantAlpha`. Linear frame buffer writes through the pixel pipeline may not be enabled for auxiliary buffer locks. The `pixelPipeline` argument is ignored for read only locks.

Upon successful completion, the user provided `GrLfbInfo_t` structure will be filled in with information pertaining to the locked buffer. The `GrLfbInfo_t` structure is currently defined as:

```c
typedef struct {
    int size;
    void *lfbPtr;
    FxU32 strideInBytes;
    GrLfbWriteMode_t writeMode;
    GrOriginLocation_t origin;
} GrLfbInfo_t;
```

The `size` element must be initialized by the user to the size of the `GrLfbInfo_t` structure, e.g.:

```c
info.size = sizeof( GrLfbInfo_t );
```

This `size` element will be used to provide backward compatibility for future revisions of the API. An unrecognized size will cause the lock to fail. The `lfbPtr` element is assigned a valid linear pointer to be used for accessing the requested buffer. The `strideInBytes` element is assigned the byte distance between scan lines.

**NOTES**

An application may not call any Glide routines other than `grLfbLock` and `grLfbUnlock` while a lock is active.

The `info.lfbptr` is invalid and may not be used outside a `grLfbLock/grLfbUnlock` pair.

Using `GR_LFB_NOIDLE` may interfere with sound generation.

**SEE ALSO**

`grGet, grLfbUnlock, grLfbConstantAlpha, grLfbConstantDepth, grLfbReadRegion, grLfbWriteRegion`
**NAME**

`grLfbReadRegion` – efficiently copy a pixel rectangle from a linear frame buffer

**C SPECIFICATION**

```c
FxBool grLfbReadRegion( GrBuffer_t src_buffer,
                        FxU32  src_x,
                        FxU32  src_y,
                        FxU32  src_width,
                        FxU32  src_height,
                        FxU32  dst_stride,
                        void  *dst_data)
```

**PARAMETERS**

- `src_buffer` Source frame buffer. Valid values are `GR_BUFFER_FRONTBUFFER`, `GR_BUFFER_BACKBUFFER`, and `GR_BUFFER_AUXBUFFER`.
- `src_x`, `src_y` Source x and y coordinates. The y origin is always assumed to be at the upper left.
- `src_width`, `src_height` Width and height of source rectangle to be copied from the frame buffer.
- `dst_stride` Stride, in bytes, of destination user memory buffer.
- `dst_data` Pointer to destination user memory buffer.

**DESCRIPTION**

This API copies a rectangle from a region of a frame buffer into a buffer in user memory; this is the only way to read back from the frame buffer on Scanline Interleaved systems.

A `src_width` by `src_height` rectangle of pixels is copied from the buffer specified by `src_buffer`, starting at the location `(src_x, src_y)`. The pixels are copied to user memory starting at `dst_data`, with a stride in bytes defined by `dst_stride`.

The frame buffer y origin is always assumed to be at the upper left. The pixel data read will always be 16-bit 565 RGB.

The `dst_stride` must be greater than or equal to `src_width` * 2.

**NOTES**

On Voodoo² systems, `grLfbReadRegion` may not be used in conjunction with triple buffering.

**SEE ALSO**

`grLfbLock`, `grLfbUnlock`, `grLfbConstantAlpha`, `grLfbConstantDepth`, `grLfbWriteRegion`
grLfbUnlock

NAME

ggLfbUnlock – unlock a frame buffer previously locked with ggLfbLock

C SPECIFICATION

FxBOOL grLfbUnlock ( GrLock_t type, GrBuffer_t buffer )

PARAMETERS

  type        Lock type. Valid values are GR_LFB_READ_ONLY and GR_LFB_WRITE_ONLY.
  buffer      Buffer to unlock. Valid values are GR_BUFFER_FRONTBUFFER, GR_BUFFER_BACKBUFFER, and GR_BUFFER_AUXBUFFER.

DESCRIPTION

When an application desires direct access to a color or auxiliary buffer, it must lock that buffer in order to gain access to a pointer to the frame buffer data. When the application has completed its direct access transactions and would like restore 3D and GUI engine access to the buffer, then it must call grLfbUnlock.

NOTES

An application may not call any Glide routines other than grLfbLock and grLfbUnlock while a lock is active.

The info.lfbptr is invalid and may not be used outside a grLfbLock / grLfbUnlock pair.

SEE ALSO

grLfbLock, grLfbConstantAlpha, grLfbConstantDepth
**grLfbWriteRegion**

**NAME**

grLfbWriteRegion – efficiently copy a pixel rectangle into a linear frame buffer

**C SPECIFICATION**

```c
FxBool grLfbWriteRegion( GrBuffer_t dst_buffer,
                        FxU32 dst_x,
                        FxU32 dst_y,
                        GrLfbSrcFmt_t src_format,
                        FxU32 src_width,
                        FxU32 src_height,
                        FxBool pixelPipeline,
                        FxI32 src_stride,
                        void *src_data)
```

**PARAMETERS**

- **dst_buffer**
  Destination frame buffer. Valid values are GR_BUFFER_FRONTBUFFER, GR_BUFFER_BACKBUFFER, and GR_BUFFER_AUXBUFFER.

- **dst_x, dst_y**
  Destination x and y coordinates. The y origin is always assumed to be at the upper left.

- **src_format**
  Format of source image.

- **src_width, src_height**
  Width and height of source image.

- **pixelPipeline**
  Enable pixel pipeline processing for source data.

- **src_stride**
  Stride of source image.

- **src_data**
  Pointer to image data.

**DESCRIPTION**

This API copies a rectangle from a region of memory pointed to by src_data into the linear frame buffer as efficiently as possible. The image may be in one of the following source formats:

- **GR_LFB_SRC_FMT_565**
  RGB 565 color image.

- **GR_LFB_SRC_FMT_555**
  RGB 555 color image.

- **GR_LFB_SRC_FMT_1555**
  RGB 1555 color image.

- **GR_LFB_SRC_FMT_888**
  RGB 888 color image each pixel padded to 32-bits with RGB in low order 24-bits.

- **GR_LFB_SRC_FMT_8888**
  ARGB 8888 color image.

- **GR_LFB_SRC_FMT_565_DEPTH**
  RGB 565 and 16-bit depth value packed into each 32-bit element of image.

- **GR_LFB_SRC_FMT_555_DEPTH**
  RGB 555 and 16-bit depth value packed into each 32-bit element of image.

- **GR_LFB_SRC_FMT_1555_DEPTH**
  RGB 1555 and 16-bit depth value packed into each 32-bit element of image.
**GR_LFB_SRC_FMT_ZA16**

Two 16-bit depth or alpha values. Alpha values are stored into odd bytes.

**GR_LFB_SRC_FMT_RLE16**

16 BPP RLE Encoded image - see notes.

Not all 3Dfx graphics subsystems will support all source image formats. The function will fail, returning `FXFALSE`, if the source format specified is not supported by the 3D hardware.

The `src_data` pointer must point to the starting pixel of the rectangle to be copied. A rectangle in memory defined by `src_width`, `src_height`, and `src_stride` will be copied into the buffer designated by `dst_buffer` at the location `(dst_x, dst_y)`. `src_stride` is defined as bytes per scan line in the source image.

If `pixelPipeline` is set to `FXTRUE`, source data will be processed through the pixel pipeline before being written into the linear frame buffer. Not all 3Dfx graphics subsystems will support pixel pipeline processing on LFB writes (e.g. Voodoo Rush). The `grLfbWriteRegion` function will fail, returning `FXFALSE`, if pixel pipeline processing is enabled on a system that doesn’t support it. Use `grGet(GR_FLB_PIXEL_PIPE, ...)` to determine whether or not LFB writes can utilize the pixel pipeline.

The frame buffer y origin is always assumed to be at the upper left.

**NOTES**

The `GR_LFB_SRC_FMT_RLE16` format is a two-word format consisting of one 16-bit count word and one 16-bit color word. The count word should be treated as a signed 16-bit integer. Negative values are currently ignored.

The Glide 3.0 release is the first release to include the `pixelPipeline` argument to `grLfbWriteRegion`.

**SEE ALSO**

`grGet`, `grLfbLock`, `grLfbUnlock`, `grLfbConstantAlpha`, `grLfbConstantDepth`, `grLfbReadRegion`
### NAME

**grLoadGammaTable** – load the hardware gamma correction table

### C Specification

```c
void grLoadGammaTable ( FxU32 nEntries, FxU32 *r, FxU32 *g, FxU32 *b )
```

### PARAMETERS

- **nEntries**
  - The number of elements in the color parameter arrays.

- **r, g, b**
  - Arrays of RGB values that will interpolated to generate an output gamma value.

### DESCRIPTION

**grLoadGammaTable** loads the hardware gamma correction table with \( nEntries \) red, green, and blue values that will be interpolated to generate an output gamma value. If \( nEntries \) is less than the size of the hardware-dependent gamma table, the first part of the table will be overwritten by the new values. If \( nEntries \) is greater than the gamma table size, the extra values are discarded. The size of the gamma table may be obtained by calling `grGet(GR_GAMMA_TABLE_ENTRIES)` . The entries in the gamma table must be monotonically increasing in each color component.; otherwise, the results are undefined.

### NOTES

It is strongly recommended that `guGammaCorrectionRGB` be used instead of `grLoadGammaTable`.

Glide 3.0 is the first release to support `grLoadGammaTable`.

### SEE ALSO

- `grGet`, `guGammaCorrectionRGB`
**NAME**

*grQueryResolutions* – query for possible frame buffer configurations

**C SPECIFICATION**

```c
typedef struct {
    GrScreenResolution_t resolution;
    GrScreenRefresh_t refresh;
    int numColorBuffers;
    int numAuxBuffers;
} GrResolution;

FxI32 grQueryResolutions ( const GrResolution * resTemplate,
                           GrResolution *output )
```

**PARAMETERS**

*resTemplate*  Template containing query constraints.

*output*  Pointer to an array of *GrResolution* structures that will receive the resolution information.

**DESCRIPTION**

*grQueryResolutions* returns all available frame buffer configurations that match the constraints specified in the template *resTemplate*. The constraints are specified as either *GR_QUERY_ANY* or a specific value in each of the four fields in the *GrResolution* structure. *grQueryResolutions* returns the number of bytes required to contain the available resolution information. Typically, a Glide application program calls *grQueryResolutions* with a NULL *output* parameter initially and uses the return value to allocate space, then calls *grQueryResolutions* again to return the information. This process is demonstrated in the code fragment below:

```c
GrResolution query;
GrResolution *list;
Int listSize;
/* find all possible modes that include a z-buffer */
query.resolution    = GR_QUERY_ANY;
query.refresh       = GR_QUERY_ANY;
query.numColorBuffers = GR_QUERY_ANY;
query.numAuxBuffers  = 1;

listSize = grQueryResolutions( &query, NULL );
list = malloc( listSize );
grQueryResolutions( &query, list );
```

**NOTES**

Glide 3.0 is the first release to support *grQueryResolutions*.

**SEE ALSO**

*grGet, grGetString, grStWinOpen*
**grRenderBuffer**

NAME

grRenderBuffer – selects the current color buffer for drawing and clearing

C SPECIFICATION

```c
void grRenderBuffer( GrBuffer_t buffer )
```

PARAMETERS

- `buffer` Selects the current color buffer. Valid values are GR_BUFFER_FRONTBUFFER and GR_BUFFER_BACKBUFFER.

DESCRIPTION

grRenderBuffer selects the buffer for primitive drawing and buffer clears. The default is GR_BUFFER_BACKBUFFER.

NOTES

SEE ALSO

- grBufferClear, grDrawLine, grDrawPoint, grDrawTriangle, grDrawVertexArray, grDrawVertexArrayContiguous
NAME

grReset – reset selected Glide state

C SPECIFICATION

FxBool grReset (FxU32 what)

PARAMETERS

what The piece of state to reset.

DESCRIPTION

grReset resets statistic counters and vertex layouts. The argument what is one of the selectors listed below.

<table>
<thead>
<tr>
<th>selector</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_STATS_PIXELS</td>
<td>Reset all the pixel statistic counters.</td>
</tr>
<tr>
<td>GR_STATS_POINTS</td>
<td>Reset all the point statistic counters.</td>
</tr>
<tr>
<td>GR_STATS_LINES</td>
<td>Reset all the line statistic counters.</td>
</tr>
<tr>
<td>GR_STATS_TRIANGLES</td>
<td>Reset all the triangle statistic counters.</td>
</tr>
<tr>
<td>GR_VERTEX_PARAMETERS</td>
<td>Reset grVertexLayout parameter offset to zero, and all parameter modes to GR_PARAM_DISABLE.</td>
</tr>
</tbody>
</table>

The current values of the statistics counters and the vertex layout size can be retrieved with calls to grGet.

NOTES

Glide 3.0 is the first release to support grReset. It replaces the now-obsolete API grSstResetPerfStats.

SEE ALSO

grGet
**grSelectContext**

NAME

    grSelectContext – activate a context

C SPECIFICATION

    FxBool grSelectContext ( GrContext_t context )

PARAMETERS

    context    A context handle.

DESCRIPTION

When programming a full screen Glide application, the developer has complete ownership of the 3D hardware and texture ram. Many applications will be developed to run under Windows 95, however, and must be prepared to restore the graphics state after a period of inactivity.

To gracefully handle the loss of resources (e.g. to another 3D application being scheduled by the Windows 95 operating system), an application is required to periodically (typically once per frame) query with grSelectContext to determine if Glide’s resources have be reallocated by the system. Context is a context handle returned from a successful call to grSstWinOpen.

If none of the rendering context’s state and resources have been disturbed since the last call, grSelectContext will return FXTRUE. In this case no special actions by the application are required. If it returns FXFALSE, then the application must assume that the rendering state has changed and must be reestablished (by re-downloading textures, explicitly resetting the rendering state, etc.) before further rendering commands are issued.

NOTES

    Glide 3.0 is the first release to support grSelectContext.

SEE ALSO

    grSstWinOpen
**NAME**

`grSttOrigin` – establishes a y origin

**C SPECIFICATION**

```c
void grSttOrigin( GrOriginLocation_t origin )
```

**PARAMETERS**

- `origin` Specifies the direction of the y coordinate axis. `GR_ORIGIN_UPPER_LEFT` places the screen space origin at the upper left corner of the screen with positive y going down. `GR_ORIGIN_LOWER_LEFT` places the screen space origin at the lower left corner of the screen with positive y going up.

**DESCRIPTION**

`grSttOrigin` sets the y origin for all triangle operations, fast fill, and clipping rectangles.

**NOTES**

- `grSttOrigin` overrides the y origin specified in `grSttWinOpen`.

**SEE ALSO**

- `grSttWinOpen`
NAME

gSstSelect – make a graphics subsystem current

C SPECIFICATION

void gSstSelect ( int which_sst )

PARAMETERS

which_sst The ordinal number of the graphics subsystem to make current. This value must be
between 0 and the number of installed subsystems returned by
gGet ( GR_NUM_BOARDS, ... ).

DESCRIPTION

gSstSelect selects a particular installed graphics subsystem as active. If the value passed is greater
than the number of installed graphics subsystems and you are using the debug build of Glide, a run-time error
will be generated.

NOTES

SEE ALSO

get, gSstWinOpen
NAME

`grSstWinClose` – close a graphics context

C SPECIFICATION

```c
FxBool grSstWinClose ( GrContext_t context )
```

PARAMETERS

- `context` : The graphics context to close.

DESCRIPTION

`grSstWinClose` closes a graphics context. It will fail, returning `FXFALSE`, if `context` is not a valid handle to a graphics context. Otherwise, it returns the state of Glide to the one following `grGlideInit`, so that `grSstWinOpen` can be called to open a new context.

NOTES

Glide 3.0 is the first release to require the `context` parameter for `grSstWinClose`.

SEE ALSO

`grSstWinOpen`
**NAME**

`grSstWinOpen` – opens the graphics display device

**C SPECIFICATION**

```c
define GrContext_t grSstWinOpen( FxU32 hWin, 
                               GrScreenResolution_t res, 
                               GrScreenRefresh_t ref, 
                               GrColorFormat_t cFormat, 
                               GrOriginLocation_t org_loc, 
                               int num_buffers, 
                               int num_aux_buffers )
```

**PARAMETERS**

- **hWin** Specifies a handle to the window. The interpretation of this value depends on the system environment. Applications run on Voodoo Graphics or Voodoo 2 hardware must specify `NULL` as well. Win32 full screen applications running on Voodoo Rush system must specify a window handle; a `NULL` value for `hWin` will cause the application’s real window handle (i.e. what is returned by `GetActiveWindow`) to be used. Since Win32 pure console applications do not have a window handle, they can be used only with Voodoo Graphics or Voodoo 2 systems and a `NULL` window handle is required.

<table>
<thead>
<tr>
<th>System environment</th>
<th>hWin value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win32 Full Screen</td>
<td></td>
</tr>
<tr>
<td>Win32 Pure Console</td>
<td><code>NULL</code> (Voodoo Graphics and Voodoo 2 only).</td>
</tr>
</tbody>
</table>

- **res** Specifies which screen resolution to use. Refer to `sst1vid.h` for available video resolutions, e.g., `GR_RESOLUTION_640x480` and `GR_RESOLUTION_800x600`. In addition, the resolution `GR_RESOLUTION_NONE` is permitted (but not recommended) for Voodoo Rush. This signals Glide to use the user specified window (see the `hWin` parameter). Specifying `GR_RESOLUTION_NONE` on a Voodoo Graphics or Voodoo 2 system will cause the call to fail.

- **ref** Specifies the refresh rate to use. Refer to `sst1vid.h` for available video resolutions, e.g., `GR_REFRESH_60Hz` and `GR_REFRESH_72Hz`. The `ref` parameter is ignored when a Win32 application is running in a window (Voodoo Rush systems only).
**cFormat**

Specifies the packed color RGBA ordering for linear frame buffer writes and parameters for the following APIs: `grBufferClear`, `grChromakeyValue`, `grConstantColorValue`, and `grFogColorValue`. The following table illustrates the available formats:

<table>
<thead>
<tr>
<th>color format</th>
<th>hex variable organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COLORFORMAT_RGBA</td>
<td>0xRRGGBBAA</td>
</tr>
<tr>
<td>GR_COLORFORMAT_ARGB</td>
<td>0xAARRGGBB</td>
</tr>
<tr>
<td>GR_COLORFORMAT_BGRA</td>
<td>0xBBGGRRAA</td>
</tr>
<tr>
<td>GR_COLORFORMAT_ABGR</td>
<td>0xAABBGGRR</td>
</tr>
</tbody>
</table>

**org_loc**

Specifies the direction of the y coordinate axis. `GR_ORIGIN_UPPER_LEFT` places the screen space origin at the upper left corner of the screen with positive y going down (a la IBM VGA). `GR_ORIGIN_LOWER_LEFT` places the screen space origin at the lower left corner of the screen with positive y going up (a la SGI GL).

**num_buffers**

Specifies the number of rendering buffers to use. Supported values 2 (double-buffering) or 3 (triple buffering). If there is not enough memory to support the desired resolution (e.g. 800×600 triple buffered on a 2MB system), an error will occur.

**num_aux_buffers**

Specifies the number of auxiliary buffers required by an application. The auxiliary buffers are used either for depth or alpha buffering. Permitted values are 0 or 1. For full screen applications, this parameter allows Voodoo Graphics, Voodoo², and Voodoo Rush systems to validate whether the available video memory will support the application’s requirements for color and auxiliary buffers at a specified screen resolution. For a windowed application running on Voodoo Rush, this parameter allows an application to run in a larger 3D window if a depth buffer is not necessary (depth and back buffers share the same off-screen video memory).

**DESCRIPTION**

`grSstWinOpen` initializes the graphics to a known state using the given parameters. It supports Voodoo graphics, Voodoo², and Voodoo Rush, and either full-screen or windowed operation in the latter. By default all special effects of the hardware (depth buffering, fog, chroma-key, alpha blending, alpha testing, etc.) are disabled and must be individually enabled. All global state constants (chroma-key value, alpha test reference, constant depth value, constant alpha value, etc.) and pixel rendering statistic counters are initialized to 0x00. Upon success, an opaque graphics context handle is returned; otherwise zero is returned. If `grSstWinOpen` is called on an already open window, FXFALSE will be returned.

**NOTES**

Glide 3.0 is the first release to introduce graphics window context handles. Only one graphics context may be created and active at a time. This restriction may be relaxed in future releases.

**SEE ALSO**

`grSstWinClose`
grTexCalcMemRequired

NAME

grTexCalcMemRequired – return the texture memory consumed by a texture

C SPECIFICATION

FxU32 grTexCalcMemRequired( GrLOD_t smallLOD,
                              GrLOD_t largeLOD,
                              GrAspectRatio_t aspect,
                              GrTextureFormat_t format )

PARAMETERS

smallLOD, largeLOD The smallest and largest LODs in the mipmap. Values are chosen from the following:
GR_LOD_LOG2_1, GR_LOD_LOG2_2, GR_LOD_LOG2_4, GR_LOD_LOG2_8, GR_LOD_LOG2_16, GR_LOD_LOG2_32, GR_LOD_LOG2_64, GR_LOD_LOG2_128, and GR_LOD_LOG2_256.

aspect Aspect ratio of the mipmap. Values are chosen from the following:
GR_ASPECT_LOG2_8x1, GR_ASPECT_LOG2_4x1, GR_ASPECT_LOG2_2x1, GR_ASPECT_LOG2_1x2, GR_ASPECT_LOG2_1x4, and GR_ASPECT_LOG2_1x8.

format Format of the mipmap. Values are chosen from the following:

DESCRIPTION

grTexCalcMemRequired calculates and returns the amount of memory a mipmap of the specified LOD range, aspect ratio, and format requires. Because of the packing requirements of some texture formats the number returned may reflect padding bytes required to properly align the mipmap in memory. See grGet(GR_TEXTUREALIGN,...).

NOTES

The value returned includes memory for both the even and odd mipmap levels. In the case where a mipmap is split across two TMUs with the even levels in one TMU and the odd levels in the other TMU, use grTexTextureMemRequired to compute the memory requirements of each TMU.

It is possible that memory required for a mipmap is less than the sum of the memory required for its individual mipmap levels. When multiple mipmap levels are packed into one mipmap, they will be loaded into contiguous memory.

SEE ALSO

grGet, grTexTextureMemRequired
**grTexChromaModeExt**

**NAME**

`grTexChromaModeExt` – set the chroma-key mode for a given TMU

**C SPECIFICATION**

```c
void grTexChromaModeExt ( GrChipID_t tmu, GrChromakeyMode_t mode )
```

**PARAMETERS**

- `tmu`  
  The TMU. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- `mode`  
  One of the following:

<table>
<thead>
<tr>
<th><code>mode</code></th>
<th><code>description</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_TEXCHROMA_DISABLE_EXT</code></td>
<td>Disable texture chroma keying.</td>
</tr>
<tr>
<td><code>GR_TEXCHROMA_ENABLE_EXT</code></td>
<td>Enables texture chroma keying. Chroma texels obtain 0x00 for RGBA.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

`grTexChromaModeExt` sets the chroma range mode for the specified TMU. It is independent of the chroma-key and chroma range modes in the pixel pipeline.

**NOTES**

Glide 3.0 is the first release to introduce `grTexChromaModeExt`. The API is available only with hardware support. Use `grGetString(GR_EXTENSION,...)` and search for the sub-string "TEXCHROMA" to query for availability of this extension. If the extension is present, the entry point may be retrieved via `grGetProcAddr`.

The TEXCHROMA extension is independent of the chroma-key and chroma range modes in the pixel pipeline.

**SEE ALSO**

- `grChromakeyMode`, `grChromaRangeModeExt`, `grGetProcAddress`, `grGetString`, `grTexChromaRangeExt`
**NAME**

`grTexChromaRangeExt` – set chroma-range values for a given TMU

**C SPECIFICATION**

```c
void grTexChromaRangeExt ( GrChipID_t tmu, 
             GrColor_t color0, 
             GrColor_t color1, 
             GrTexChromakeyMode_t mode
)
```

**PARAMETERS**

- **tmu**
  The TMU. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.

- **color0**, **color1**
  Independent range values for red, green, and blue.

- **mode**
  Chroma-range match criteria. Only one mode value is currently supported: `GR_TEXCHROMARANGE_RGB_ALL_EXT`.

**DESCRIPTION**

`grTexChromaRangeExt` sets the chroma range values for the TMU specified by `tmu`.

The color format for `color0` and `color1` should be the same one as specified in the `cFormat` parameter to `grSstWinOpen`. The order in which range values are specified for a particular color component is irrelevant, i.e. the `{color0, color1}` pairs `{{130,36,87}, {150,38,92}}` and `{{150,36,92}, {130,38,87}}` are equivalent.

The `mode` parameter determines the way the color ranges are used in the chroma test. Only one value is currently supported, `GR_TEXCHROMARANGE_RGB_ALL_EXT`. In this mode, each color component pair defines an inclusive range such that `lower bound ≤ color ≤ upper bound`. If **all** components of the incoming pixel color fall within their ranges, the chroma test succeeds and the pixel is invalidated.

**NOTES**

Glide 3.0 is the first release to introduce `grTexChromaRangeExt`. The API is available only with hardware support. Use `grGetString(GR_EXTENSION,...)` and search for the sub-string “TEXCHROMA” to query for availability of this extension. If the extension is present, the entry point may be retrieved via `grGetProcAddr`.

The TEXCHROMA extension is independent of the chroma-key and chroma range modes in the pixel pipeline.

**SEE ALSO**

- `grChromakeyValue`, `grChromaRangeExt`, `grGetProcAddress`, `grGetString`, `grTexChromaModeExt`
**Name**

`grTexClampMode` – set the texture map clamping/wrapping mode

**C Specification**

```c
void grTexClampMode( GrChipID_t tmu,
                      GrTextureClampMode_t sClampMode,
                      GrTextureClampMode_t tClampMode );
```

**Parameters**

- `tmu` Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- `sClampMode` The new mode for the `s` direction, either `GR_TEXTURECLAMP_CLAMP` or `GR_TEXTURECLAMP_WRAP`.
- `tClampMode` The new mode for the `t` direction, either `GR_TEXTURECLAMP_CLAMP` or `GR_TEXTURECLAMP_WRAP`.

**Description**

`grTexClampMode` sets the texture mapping clamping/wrapping mode for both the `s` and `t` directions. If wrapping is enabled, then texture maps will tile, i.e. values greater than 255 for window coordinate systems, or 1.0 for clip coordinate system, will wrap around to 0. If clamping is enabled, then texture map indices will be clamped to 0 and 255 (window coordinates) or 0.0 and 1.0 (clip coordinates).

If the TEXMIRROR extension is supported, then either `sClampMode` or `tClampMode` or both may be set to `GR_TEXTURECLAMP_MIRROR_EXT`. See the Glide 3.0 Programming Manual for a description of extensions in general and the TEXMIRROR one in particular.

**Notes**

Both modes should always be set to `GR_TEXTURECLAMP_CLAMP` for perspectively projected textures.

**See Also**

`grGetString`, `grTexSource`
grTexCombine

NAME

grTexCombine – configure a texture combine unit

C SPECIFICATION

```c
void grTexCombine( GrChipID_t tmu,
                   GrCombineFunction_t rgb_function,
                   GrCombineFactor_t rgb_factor,
                   GrCombineFunction_t alpha_function,
                   GrCombineFactor_t alpha_factor,
                   FxBool rgb_invert,
                   FxBool alpha_invert)
```

PARAMETERS

- **tmu**
  Texture Mapping Unit to modify. Valid values are GR_TMU0, GR_TMU1, and GR_TMU2.

- **rgb_function**
  Specifies the function used in texture color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combine function</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FUNCTION_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL</td>
<td>C_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL_ALPHA</td>
<td>A_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER</td>
<td>f * C_other</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_BLEND_OTHER</td>
<td></td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL</td>
<td>f * C_other + C_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL_ALPHA</td>
<td>f * C_other + A_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL</td>
<td>f * (C_other - C_local)</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_BLEND</td>
<td><img src="equation" alt="Equation" /></td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL</td>
<td>f * (C_other - C_local) + A_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL</td>
<td>f * (C_other - C_local) + C_local</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_BLEND_LOCAL</td>
<td><img src="equation" alt="Equation" /></td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA</td>
<td><img src="equation" alt="Equation" /></td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA</td>
<td><img src="equation" alt="Equation" /></td>
</tr>
</tbody>
</table>
**rgb_factor**

Specifies the scaling factor \( f \) used in texture color generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combine factor</th>
<th>scale factor ((f))</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FACTOR_NONE</td>
<td>unspecified</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL</td>
<td>( C_{\text{local}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_OTHER_ALPHA</td>
<td>( A_{\text{other}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL_ALPHA</td>
<td>( A_{\text{local}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_DETAIL_FACTOR</td>
<td>( \beta )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOD_FRACTION</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE</td>
<td>1</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL</td>
<td>1 – ( C_{\text{local}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA</td>
<td>1 – ( A_{\text{other}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA</td>
<td>1 – ( A_{\text{local}} / 255 )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_DETAIL_FACTOR</td>
<td>1 – ( \beta )</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOD_FRACTION</td>
<td>1 – ( \lambda )</td>
</tr>
</tbody>
</table>

**alpha_function**

Specifies the function used in texture alpha generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combine function</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FUNCTION_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL</td>
<td>( A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_LOCAL_ALPHA</td>
<td>( A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER</td>
<td>( f * A_{\text{other}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL</td>
<td>( f * A_{\text{other}} + A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL_ALPHA</td>
<td>( f * A_{\text{other}} + A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL</td>
<td>( f * (A_{\text{other}} – A_{\text{local}}) ) = ( f * A_{\text{other}} + (1 – f) * A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL</td>
<td>( f * (A_{\text{other}} – A_{\text{local}}) + A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_BLEND</td>
<td>( f * (A_{\text{other}} – A_{\text{local}}) + A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_BLEND_LOCAL</td>
<td>( f * (-A_{\text{local}}) + A_{\text{local}} ) = ( (1 – f) * A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL</td>
<td>( f * (-A_{\text{local}}) + A_{\text{local}} ) = ( (1 – f) * A_{\text{local}} )</td>
</tr>
<tr>
<td>GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA</td>
<td>( f * (-C_{\text{local}}) + A_{\text{local}} ) = ( (1 – f) * A_{\text{local}} )</td>
</tr>
</tbody>
</table>
alpha_factor  Specifies the scaling factor $f$ used in texture alpha generation. Valid parameters are described below:

<table>
<thead>
<tr>
<th>combine factor</th>
<th>scale factor ($f$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_COMBINE_FACTOR_NONE</td>
<td>unspecified</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL</td>
<td>$A_{local} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_OTHER_ALPHA</td>
<td>$A_{other} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOCAL_ALPHA</td>
<td>$A_{local} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_DETAIL_FACTOR</td>
<td>$\beta$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_LOD_FRACTION</td>
<td>1</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE</td>
<td>1</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL</td>
<td>$1 - A_{local} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA</td>
<td>$1 - A_{other} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA</td>
<td>$1 - A_{local} / 255$</td>
</tr>
<tr>
<td>GR_COMBINE_FACTOR_ONE_MINUS_DETAIL_FACTOR</td>
<td>$1 - \beta$</td>
</tr>
</tbody>
</table>

rgb_invert  Specifies whether the generated texture color should be bitwise inverted as a final step.

alpha_invert  Specifies whether the generated texture alpha should be bitwise inverted as a final step.

DESCRIPTION  

grTexCombine configures the color and alpha texture combine units of the graphics hardware pipeline. This provides a low level mechanism for controlling all the modes of the texture combine unit without manipulating individual register bits.

The texture combine unit computes the function specified by rgb_function and alpha_function using rgb_factor and alpha_factor as scale factors on the local ($C_{local}$ and $A_{local}$) and upstream ($C_{other}$ and $A_{other}$) filtered texels. The result is clamped to $[0..255]$, and then a bitwise inversion may be applied, controlled by the rgb_invert and alpha_invert parameters. The final result is then passed downstream, to either another TMU or the Pixelfx chip.

In the rgb_factor and alpha_factor tables, $\beta$ is the detail blend factor that is computed as a function of LOD, and $\lambda$ is the fractional part of the LOD. See grTexDetailControl for further information.

grTexCombine also tracks required vertex parameters for the rendering routines. GR_COMBINE_FACTOR_NONE indicates that no parameters are required; it is functionally equivalent to GR_COMBINE_FACTOR_ZERO.

NOTES  

$C_{local}$ and $A_{local}$ are the color components generated by indexing and filtering from the mipmap stored on the selected TMU; $C_{other}$ and $A_{other}$ are the incoming color components from the neighboring TMU.

Inverting the bits in a color is the same as computing $(1.0 – \text{color})$ for floating point color values in the range $[0..1]$ or $(255 – \text{color})$ for 8-bit color values in the range $[0..255]$.

The TMU closest to the Pixelfx chip is GR_TMU0. If a TMU exists upstream from GR_TMU0, it is GR_TMU1. If a TMU exists upstream from GR_TMU1, it is GR_TMU2.

SEE ALSO  

grDrawTriangle, grTexLodBiasValue, grTexDetailControl
NAME

grTexDetailControl – set the detail texturing controls

C SPECIFICATION

void grTexDetailControl( GrChipID_t tmu,
                         int lodBias,
                         FxU8 detailScale,
                         float detailMax
                     )

PARAMETERS

tmu

Texture Mapping Unit to modify. Valid values are GR_TMU0, GR_TMU1, and
GR_TMU2.

lodBias

Controls where the blending between the two textures begins. This value is an LOD
bias value in the range [-32.. +31].

detailScale

Controls the steepness of the blend. Values are in the range [0..7] are valid. The scale
is computed as 2^{detailScale}.

detailMax

Controls the maximum blending that occurs. Values in the range [0.0..1.0] are valid.

DESCRIPTION

Detail texturing refers to the effect where the blend between two textures in a texture combine unit is a
function of the LOD calculated for each pixel. grTexDetailControl controls how the detail blending
factor, $\beta$, is computed from LOD. The lodBias parameter controls where the blending begins, the
detailScale parameter controls the steepness of the blend (how fast the detail pops in), and the detailMax
parameter controls the maximum blending that occurs. Detail blending factor $\beta$ is calculated as

$$\beta = \min( \text{detailMax}, \max( 0, (\text{lodBias} - \text{LOD}) \ll \text{detailScale} ) / 255.0 )$$

where LOD is the calculated LOD before grTexLodBiasValue is added. The detail blending factor is
typically used by calling grTexCombine with an rgb_function of GR_COMBINE_FUNCTION_BLEND and an
rgb_factor of GR_COMBINE_FACTOR_DETAIL_FACTOR to compute:

$$C_{out} = \beta*\text{detail_texture} + (1 - \beta)*\text{main_texture}$$

NOTES

An LOD of $n$ is calculated when a pixel covers approximately $2^{2n}$ texels. For example, when a pixel covers
approximately 1 texel, the LOD is 0. When a pixel covers 4 texels, the LOD is 1, and when a pixel covers
16 texels, the LOD is 2.

Detail blending occurs in the downstream TMU. Since the detail texture and main texture typically have
very different computed LODs, the detail texturing control settings depend on which texture is in the
downstream TMU.

SEE ALSO

grTexCombine, grTexLodBiasValue
**NAME**

`grTexDownloadMipMap` – download a complete mipmap to texture memory

**C SPECIFICATION**

```c
void grTexDownloadMipMap ( GrChipID_t tmu,
                        FxU32 startAddress,
                        FxU32 evenOdd,
                        GrTexInfo *info )
```

**PARAMETERS**

- `tmu` Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- `startAddress` Offset into texture memory where the texture will be loaded.
- `evenOdd` Which mipmap levels to download. Valid values are `GR_MIPMAPLEVELMASK_EVEN`, `GR_MIPMAPLEVELMASK_ODD`, and `GR_MIPMAPLEVELMASK_BOTH`.
- `info` Format, dimensions, and image data for texture.

**DESCRIPTION**

`grTexDownloadMipMap` downloads an entire mipmap to an area of texture memory specified by `startAddress`. Valid values for `startAddress` must be between the values returned by `grTexMinAddress` and `grTexMaxAddress`.

**NOTES**

The `GrTexInfo` structure has changed in Glide 3.0. See `glide.h` for more information.

An error will occur if the mipmap is loaded into an area that crosses a 2MB boundary. See the *Glide Programming Manual* for more information.

**SEE ALSO**

- `grTexDownloadMipMapLevel`, `grTexMinAddress`, `grTexMaxAddress`,
- `grTexTextureMemRequired`, `grTexSource`
grTexDownloadMipMapLevel

NAME

grTexDownloadMipMapLevel – download a single mipmap level to texture memory

C SPECIFICATION

```c
void grTexDownloadMipMapLevel ( GrChipID_t tmu,
                              FxU32 startAddress,
                              GrLOD_t thisLod,
                              GrLOD_t largeLod,
                              GrAspectRatio_t aspectRatio,
                              GrTextureFormat_t format,
                              FxU32 evenOdd,
                              void *data)
```

PARAMETERS

- `tmu` Texture Mapping Unit to modify. Valid values are GR_TMU0, GR_TMU1, and GR_TMU2.
- `startAddress` Starting address in texture memory of the largest level of the mipmap.
- `thisLod` Constant describing LOD to be downloaded. Values are chosen from the following: GR_LOD_LOG2_1, GR_LOD_LOG2_2, GR_LOD_LOG2, GR_LOD_LOG2_4, GR_LOD_LOG2_8, GR_LOD_LOG2_16, GR_LOD_LOG2_32, GR_LOD_LOG2_64, GR_LOD_LOG2_128, and GR_LOD_LOG2_256.
- `largeLod` Constant describing largest LOD in the complete mipmap of which `thisLod` is a part. Values are chosen from the following: GR_LOD_LOG2_1, GR_LOD_LOG2_2, GR_LOD_LOG2, GR_LOD_LOG2_4, GR_LOD_LOG2_8, GR_LOD_LOG2_16, GR_LOD_LOG2_32, GR_LOD_LOG2_64, GR_LOD_LOG2_128, and GR_LOD_LOG2_256.
- `aspectRatio` Constant describing aspect ratio of texture image. Values are chosen from the following: GR_ASPECT_LOG2_8x1, GR_ASPECT_LOG2_4x1, GR_ASPECT_LOG2_2x1, GR_ASPECT_LOG2_1x1, GR_ASPECT_LOG2_1x2, GR_ASPECT_LOG2_1x4, and GR_ASPECT_LOG2_1x8.
- `format` Constant describing format of color data in texture image. Values are chosen from the following: GR_TEXFMT_RGB_332, GR_TEXFMT_YUV_422, GR_TEXFMT_ALPHA_8, GR_TEXFMT_INTENSITY_8, GR_TEXFMT_ALPHA_INTENSITY_44, GR_TEXFMT_P_8, GR_TEXFMT_8332, GR_TEXFMT_AYIQ_8422, GR_TEXFMT_RGB_565, GR_TEXFMT_ARGB_1555, GR_TEXFMT_ARGB_4444, GR_TEXFMT_ALPHA_INTENSITY_88, and GR_TEXFMT_AP_88.
- `evenOdd` Which mipmap levels to download. Valid values are GR_MIPMAPLEVELMASK_EVEN, GR_MIPMAPLEVELMASK_ODD, and GR_MIPMAPLEVELMASK_BOTH.
- `data` Raw texture image data.
grTexDownloadMipMapLevel downloads a single mipmap level. startAddress points to the beginning of the mipmap; Glide will calculate an offset from startAddress to the beginning of the level to be replaced.

startAddress must lie between the values returned by grTexMinAddress and grTexMaxAddress.

An error will occur if the mipmap level is loaded into an area that crosses a 2MB boundary. See the Glide Programming Manual for more information.

NOTES

SEE ALSO

grTexDownloadMipMap, grTexDownloadMipMapLevelPartial, grTexMinAddress, grTexMaxAddress, grTexTextureMemRequired, grTexSource
**NAME**

`grTexDownloadMipMapLevelPartial` – download part of a single mipmap level to texture memory

**C SPECIFICATION**

```c
void grTexDownloadMipMapLevelPartial( GrChipID_t tmu,
    FxU32 startAddress,
    GrLOD_t thisLod,
    GrLOD_t largeLod,
    GrAspectRatio_t aspectRatio,
    GrTextureFormat_t format,
    FxU32 evenOdd
    void *data,
    int start,
    int end )
```

**PARAMETERS**

- `tmu`  
  Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.

- `startAddress`  
  Starting address in texture memory of the largest level of the mipmap.

- `thisLod`  
  Constant describing LOD to be downloaded. Values are chosen from the following:  
  `GR_LOD_LOG2_1`, `GR_LOD_LOG2_2`, `GR_LOD_LOG2`, `GR_LOD_LOG2_4`, `GR_LOD_LOG2_8`, `GR_LOD_LOG2_16`, `GR_LOD_LOG2_32`, `GR_LOD_LOG2_64`, `GR_LOD_LOG2_128`, and `GR_LOD_LOG2_256`.

- `largeLod`  
  Constant describing largest LOD in the complete mipmap of which `thisLod` is a part.  
  Values are chosen from the following:  
  `GR_LOD_LOG2_1`, `GR_LOD_LOG2_2`, `GR_LOD_LOG2_4`, `GR_LOD_LOG2_8`, `GR_LOD_LOG2_16`, `GR_LOD_LOG2_32`, `GR_LOD_LOG2_64`, and `GR_LOD_LOG2_128`.

- `aspectRatio`  
  Constant describing aspect ratio of texture image. Values are chosen from the following:  
  `GR_ASPECT_LOG2_8x1`, `GR_ASPECT_LOG2_4x1`, `GR_ASPECT_LOG2_2x1`, `GR_ASPECT_LOG2_1x1`, `GR_ASPECT_LOG2_1x2`, `GR_ASPECT_LOG2_1x4`, and `GR_ASPECT_LOG2_1x8`.

- `format`  
  Constant describing format of color data in texture image. Values are chosen from the following:  

- `evenOdd`  
  Which mipmap levels to download. Valid values are `GR_MIPMAPLEVELMASK_EVEN`, `GR_MIPMAPLEVELMASK_ODD`, and `GR_MIPMAPLEVELMASK_BOTH`.

- `data`  
  Raw texture image data.
**DESCRIPTION**

`grTexDownloadMipMapLevelPartial` downloads part of a single mipmap level. `startAddress` points to the beginning of the mipmap; Glide will calculate an offset from `startAddress` to the beginning of the level to be replaced, with `start` and `end` determining the specific rows of the mipmap level to replace.

Valid values for `startAddress` must be between the values returned by `grTexMinAddress` and `grTexMaxAddress`. `startAddress` should point to the beginning of the mipmap even if the starting row to be downloaded is not the first row in the texture.

**NOTES**

To download one row of the texture, use the same value for `start` and `end`.

An error will occur if the mipmap is loaded into an area that crosses a 2MB boundary. See the *Glide Programming Manual* for more information.

**SEE ALSO**

`grTexDownloadMipMap`, `grTexDownloadMipMapLevel`, `grTexMinAddress`, `grTexMaxAddress`, `grTexTextureMemRequired`, `grTexSource`
**grTexDownloadTable**

**NAME**

`grTexDownloadTable` – download an NCC table or color palette

**C SPECIFICATION**

```c
void grTexDownloadTable( GrTexTable_t type, void *data )
```

**PARAMETERS**

- `type` Type of texture table. The valid values are:
  - `GR_TEXTABLE_NCC0` – Narrow-channel compression table 0,
  - `GR_TEXTABLE_NCC1` – Narrow-channel compression table 1,
  - `GR_TEXTABLE_PALETTE` – 256 entry color palette containing 8-bit RGB.
  - `GR_TEXTABLE_PALETTE_6666_EXT` – 256 entry color palette containing 6-bit ARGB. *This is an extension and may not be supported on all hardware.*

- `data` Table data, either of type `GuNccTable` or `GuTexPalette`.

**DESCRIPTION**

`grTexDownloadTable` downloads either an NCC table or a 256-entry color palette. The color palette is referenced when rendering texture formats `GR_TEXFMT_P_8` or `GR_TEXFMT_AP_88`. One of two NCC tables is used when decompressing texture formats `GR_TEXFMT_YIQ_422` or `GR_TEXFMT_AYIQ_8422`. Use `grTexNCCTable` to select one of the two NCC tables.

**NOTES**

- `grTexSource` does not download a texture’s table – this must be done separately using `grTexDownloadTable`.

- `grTexDownloadTable` does not download an NCC table if the table address is the same as the last table downloaded. Therefore, if the table’s data has changed, it must be copied to a new address before downloading.

Glide 3.0 implements one global palette and two global NCC tables. Previous versions of Glide allowed palettes and NCC tables to differ on each TMU. In Glide 3.0, however, if one TMU is using palette textures, then the others cannot be in NCC mode. Similarly, if one TMU is using a compressed texture, the palette is off limits to the other TMUs.

**SEE ALSO**

- `grGetString`, `grTexDownloadTablePartial`, `grTexNCCTable`, `grTexSource`
**grTexDownloadTablePartial**

**NAME**

`grTexDownloadTablePartial` – download a subset of an NCC table or color palette

**C SPECIFICATION**

```c
void grTexDownloadTablePartial( GrTexTable_t type,
                  void *data,
                  int start,
                  int end)
```

**PARAMETERS**

- **type**
  Type of texture table. Valid values are:
  - `GR_TEXTABLE_PALETTE` – 256-entry color palette containing 8-bit RGB.
  - `GR_TEXTABLE_PALETTE_6666_EXT` – 256-entry color palette containing 6-bit ARGB. This is an extension and may not be supported on all hardware.

- **data**
  Table data, either of type `GuNccTable` or `GuTexPalette`.

- **start, end**
  Starting and ending entries to download.

**DESCRIPTION**

`grTexDownloadTablePartial` downloads part of a 256-entry color palette to a TMU. Entries from `start` up to and including `end` are downloaded. The color palette is referenced when rendering texture formats `GR_TEXFMT_P_8` or `GR_TEXFMT_AP_88`.

**NOTES**

To download one entry, use the same value for `start` and `end`.

Partial downloading of NCC tables is not supported at this time.

**SEE ALSO**

- `grGetString`, `grTexDownloadTable`, `grTexNCCTable`, `grTexSource`
grTexFilterMode

NAME

grTexFilterMode – specify the texture minification and magnification filters

C SPECIFICATION

```c
void grTexFilterMode( GrChipID_t tmu,
                      GrTextureFilterMode_t minFilterMode,
                      GrTextureFilterMode_t magFilterMode );
```

PARAMETERS

- **tmu**
  Texture Mapping Unit to modify. Valid values are GR_TMU0, GR_TMU1, and GR_TMU2.

- **minFilterMode**
  The minification filter, either GR_TEXTUREFILTER_POINT_SAMPLED or GR_TEXTUREFILTER_BILINEAR.

- **magFilterMode**
  The magnification filter, either GR_TEXTUREFILTER_POINT_SAMPLED or GR_TEXTUREFILTER_BILINEAR.

DESCRIPTION

**grTexFilterMode** specifies the texture filters for minification and magnification. The magnification filter is used when the LOD calculated for a pixel indicates that the pixel covers less than one texel. Otherwise, the minification filter is used.

NOTES

SEE ALSO

- grTexSource
**grTexLodBiasValue**

**NAME**

`grTexLodBiasValue` – set the LOD bias value

**C SPECIFICATION**

```c
void grTexLodBiasValue ( GrChipID_t tmu, float bias )
```

**PARAMETERS**

- `tmu` Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- `bias` The new LOD bias value, a signed floating point value in the range [-8..7.75].

**DESCRIPTION**

`grTexLodBiasValue` changes the current LOD bias value, which allows an application to maintain fine grain control over the effects of mipmapping, specifically when mipmap levels change. The LOD bias value is added to the LOD calculated for a pixel and the result determines which mipmap level to use. Smaller LOD values make increasingly sharper images which may suffer from aliasing and moiré effects. Larger LOD values make increasingly smooth images which may suffer from becoming too blurry. The default LOD bias value is 0.0.

During some special effects, an LOD bias may help image quality. If an application is not performing texture mapping with trilinear filtering or dithered mipmapping, then an LOD bias of 0.5 generally improves image quality by rounding to the nearest LOD. If an application is performing dithered mipmapping (i.e., `grTexMipMapMode` is `GR_MIPMAP_NEAREST_DITHER`), then an LOD bias of 0.0 or 0.25 generally improves image quality. An LOD bias value of 0.0 is usually best with trilinear filtering.

**NOTES**

The `bias` parameter is rounded to the nearest quarter increment.

**SEE ALSO**

`grTexMipMapMode`, `grTexSource`
grTexMaxAddress

NAME

grTexMaxAddress – return the highest start address for texture downloads

C SPECIFICATION

FxU32 grTexMaxAddress (GrChipID_t tmu)

PARAMETERS

tmu
Texture Mapping Unit to query. Valid values are GR_TMU0, GR_TMU1, and GR_TMU2.

DESCRIPTION

grTexMaxAddress returns the upper bound on texture memory addresses for a specific TMU.

NOTES

The returned address is the highest valid texture start address and is valid only for the smallest mipmap level GR_LOD_LOG2_1.

SEE ALSO

grTexMinAddress, grTexDownloadMipMap, grTexDownloadMipMapLevel, grTexSource
**grTexMinAddress**

**NAME**

`grTexMinAddress` – return the lowest start address for texture downloads

**C SPECIFICATION**

```c
FxU32 grTexMinAddress ( GrChipID_t tmu )
```

**PARAMETERS**

`tmu`  
Texture Mapping Unit to query. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.

**DESCRIPTION**

`grTexMinAddress` returns the lower bound on texture memory addresses for a specific TMU.

**NOTES**

**SEE ALSO**

`grTexMaxAddress`, `grTexDownloadMipMap`, `grTexDownloadMipMapLevel`, `grTexSource`
**grTexMipMapMode**

**NAME**

`grTexMipMapMode` – set the mipmapping mode

**C SPECIFICATION**

```c
void grTexMipMapMode( GrChipID_t tmu, GrMipMapMode_t mode, FxBool lodBlend)
```

**PARAMETERS**

- `tmu`  
  Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.

- `mode`  
  The new mipmapping mode. Valid values are `GR_MIPMAP_DISABLE`, `GR_MIPMAP_NEAREST`, and `GR_MIPMAP_NEAREST_DITHER`.

- `lodBlend`  
  `FXTRUE` enables blending between levels of detail for trilinear mipmapping. `FXFALSE` disables LOD blending.

**DESCRIPTION**

`grTexMipMapMode` sets the mipmapping mode for the graphics hardware. The graphics hardware performs mipmapping with no performance penalty. Either no mipmapping, nearest mipmapping, or nearest dithered mipmapping can be performed. Nearest mipmapping (`GR_MIPMAP_NEAREST`) selects the nearest mipmap based on LOD. Dithered nearest mipmapping (`GR_MIPMAP_NEAREST_DITHERED`) dithers between adjacent mipmap levels to reduce the effects of mipmap banding but without the cost of trilinear filtering with LOD blending.

**NOTES**

- `GR_MIPMAP_NEAREST_DITHERED` mode can degrade fill-rate performance by 20-30% in some applications. If this mode is used, performance should be benchmarked to determine the cost of the increased quality. In order to prevent inadvertent use of the `GR_MIPMAP_NEAREST_DITHERED` mode, it’s availability must be specifically enabled by calling `grEnable(GR_ALLOW_MIPMAP_DITHER)`. Note that `grEnable` merely make the mode available; it must still be enabled by calling `grTexMipMapMode`.

- `GR_MIPMAP_NEAREST` truncates the LOD calculated for each pixel. To round to the nearest LOD, set the LOD bias value to 0.5 with `grTexLodBiasValue`.

- `GR_MIPMAP_NEAREST` should be used when `lodBlend` is `FXTRUE`.

**SEE ALSO**

- `grEnable`, `grTexLodBiasValue`, `grTexSource`
**grTexMultibase**

**Name**

`grTexMultibase` – enables or disables multibase addressing

**C Specification**

```c
void grTexMultibase( GrChipID_t tmu, FxBool enable )
```

**Parameters**

- `tmu` Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- `enable` `FXTRUE` enables multibase addressing, `FXFALSE` disables multibase addressing.

**Description**

`grTexMultibase` enables or disables multibase addressing. Normally, mipmap levels are stored sequentially in texture memory. Multibase addressing allows mipmap levels to be loaded into different texture memory locations. Multibase addressing must be enabled before downloading a multibased texture, and before rendering using a multibased texture. Multibase addressing must be disabled before downloading or rendering from a texture with a single base address.

**Notes**

Use `grTexMultibaseAddress` to specify the multiple base addresses for a multibased texture.

An error will occur if a mipmap level is loaded into an area that crosses a 2MB boundary. See the Glide Programming Manual for more information.

**See Also**

`grTexMultibaseAddress, grTexSource`
**grTexMultibaseAddress**

**NAME**

*grTexMultibaseAddress* – specify one base address for a multibased texture

**C SPECIFICATION**

```c
void grTexMultibaseAddress(
GrChipID_t tmu,
GrTexBaseRange_t range,
FxU32 startAddress,
FxU32 evenOdd,
GrTexInfo *info
)
```

**PARAMETERS**

- **tmu**
  - Texture Mapping Unit to modify. Valid values are `GR_TMU0`, `GR_TMU1`, and `GR_TMU2`.
- **range**
  - Which base address to specify. Valid values are `GR_TEXBASE_256`, `GR_TEXBASE_128`, `GR_TEXBASE_64`, and `GR_TEXBASE_32_TO_1`.
- **startAddress**
  - Starting address in texture memory for texture.
- **evenOdd**
  - Which mipmap levels reside on this TMU for this texture. Valid values are `GR_MIPMAPLEVELMASK_EVEN`, `GR_MIPMAPLEVELMASK_ODD`, and `GR_MIPMAPLEVELMASK_BOTH`.
- **info**
  - Format and dimensions of the texture.

**DESCRIPTION**

*grTexMultibaseAddress* specifies one base address for a texture with multiple base addresses. Normally, mipmap levels are stored sequentially in texture memory. Multibase addressing allows mipmap levels to be loaded into different texture memory locations. Four different base addresses are specified for a multibased texture, one for `GR_LOD_LOG2_256`, one for `GR_LOD_LOG2_128`, one for `GR_LOD_LOG2_64`, and one for `GR_LOD_LOG2_32` through `GR_LOD_LOG2_1`. In each case, `startAddress` should point to the texture memory location for the corresponding mipmap level.

All of the base addresses for a multibased texture should be specified before downloading the texture or rendering from the texture.

**NOTES**

The `GrTexInfo` structure has changed in Glide 3.0. See `glide.h` for more information.

*grTexSource* does not restore the multiple base addresses for a multibased texture, but does set the base address for mipmap level `GR_LOD_LOG2_256`. Therefore, it is not necessary to call *grTexMultibaseAddress* with a `range` of `GR_TEXBASE_256` after a call to *grTexSource*.

If a mipmap does not include some of the larger mipmap levels, then the base addresses associated with these missing levels need not be specified.

An error will occur if a mipmap level is loaded into an area that crosses a 2MB boundary. See the *Glide Programming Manual* for more information.

**SEE ALSO**

*grTexMultibase, grTexSource*
**NAME**

`grTexNCCTable` – select an NCC table

**C Specification**

```c
void grTexNCCTable(GrNCCTable_t table )
```

**Parameters**

- `table`: NCC table to use for decompressing compressed textures. Valid values are `GR_TEXTURE_NCC0` and `GR_TEXTURE_NCC1`.

**Description**

`grTexNCCTable` selects one of the two NCC tables as the current source for NCC decompression operations. Before rendering operations commence, the appropriate NCC table should be downloaded using `grTexDownloadTable`.

**Notes**

Glide 3.0 implements two global NCC tables. Previous versions of Glide allowed different NCC tables on each TMU.

**See Also**

- `grTexDownloadTable`, `grTexSource`
**NAME**

**grTexSource** – specify the current texture source for rendering

**C SPECIFICATION**

```c
void grTexSource( GrChipID_t tmu,
                 FxU32     startAddress,
                 FxU32     evenOdd,
                 GrTexInfo *info )
```

**PARAMETERS**

- **tmu**
  Texture Mapping Unit to modify. Valid values are GR_TMU0, GR_TMU1, and GR_TMU2.

- **startAddress**
  Starting address in texture memory for texture.

- **evenOdd**
  Which mipmap levels have been downloaded at `startAddress`. Valid values are
  GR_MIPMAPLEVELMASK_EVEN, GR_MIPMAPLEVELMASK_ODD, and
  GR_MIPMAPLEVELMASK_BOTH.

- **info**
  Format and dimensions of the new texture.

**DESCRIPTION**

**grTexSource** sets up the area of texture memory that is to be used as a source for subsequent texture
mapping operations. The `startAddress` specified should be the same as the `startAddress` argument to
**grTexDownloadMipMap** or the starting address used for the largest mipmap level when using
**grTexDownloadMipMapLevel** or **grTexDownloadMipMapLevelPartial**.

**NOTES**

The **GrTexInfo** structure has changed in Glide 3.0. See **glide.h** for more information.

An error will occur if a mipmap level is loaded into an area that crosses a 2 Mbyte boundary. See the **Glide Programming Manual** for more information.

**SEE ALSO**

- **grTexDownloadMipMap**
- **grTexDownloadMipMapLevel**
- **grTexDownloadMipMapLevelPartial**
- **grTexMinAddress**
- **grTexMaxAddress**
- **grTexTextureMemRequired**
**grTexTextureMemRequired**

**NAME**

*grTexTextureMemRequired* – return the texture memory consumed by a texture

**C SPECIFICATION**

```
FxU32 grTexTextureMemRequired( FxU32 evenOdd, GrTexInfo *info )
```

**PARAMETERS**

- `evenOdd` Which mipmap levels are included: even, odd or both. Valid values are `GR_MIPMAPLEVELMASK_EVEN`, `GR_MIPMAPLEVELMASK_ODD`, and `GR_MIPMAPLEVELMASK_BOTH`.
- `info` Format and dimensions of the texture.

**DESCRIPTION**

*grTexTextureMemRequired* calculates and returns the number of bytes required to store a given texture, including any padding bytes required to properly align the mipmap in memory. See *grGet*(GR_TEXTURE_ALIGN,...). The number returned may be added to the start address for a texture download to determine the next free location in texture memory.

**NOTES**

The *GrTexInfo* structure has changed in Glide 3.0. See *glide.h* for more information.

**SEE ALSO**

*grGet, grTexCalcMemRequired, grTexDownloadMipMap, grTexDownloadMipMapLevel, grTexMinAddress, grTexMaxAddress, grTexSource*
**grVertexLayout**

**NAME**

*grVertexLayout* – specify the format of by-vertex arrays

**C SPECIFICATION**

```c
void grVertexLayout ( FxU32 param, FxI32 offset, FxU32 mode )
```

**PARAMETERS**

- **param**
  - Parameter selector.
- **offset**
  - Offset of the parameter data from the vertex pointer, in bytes.
- **mode**
  - **GR_PARAM_ENABLE** or **GR_PARAM_DISABLE**.

**DESCRIPTION**

*grVertexLayout* specifies the internal format of the vertex structure for arrays organized in the *by-vertex* format. All parameters associated with a vertex (color, texture coordinates, etc.) are grouped together in a vertex structure, and there is a single array of these vertex structures. The layout of each vertex is defined with *grVertexLayout*, and drawn with *grDrawVertexArray*.

*grVertexLayout* is called once for each value of *param*, chosen from the tables that follow. Every vertex must include **GR_PARAM_XY**; the other components depend on your choice of coordinate space and rendering modes.
When using **clip coordinates**, use these values for `param`:

<table>
<thead>
<tr>
<th>param</th>
<th>type</th>
<th>size in bytes</th>
<th>description</th>
<th>values</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_PARAM_XY</td>
<td>FxFloat</td>
<td>8</td>
<td>x and y coordinates. Vertex snapping is no longer required.</td>
<td>In the range $[-w..w]$.</td>
<td>Required. Must be at offset 0.</td>
</tr>
<tr>
<td>GR_PARAM_Z</td>
<td>FxFloat</td>
<td>4</td>
<td>z coordinate.</td>
<td>In the range $[-w..w]$.</td>
<td>When z buffering is enabled.</td>
</tr>
<tr>
<td>GR_PARAM_W</td>
<td>FxFloat</td>
<td>4</td>
<td>w coordinate.</td>
<td>In the range $[1..64K]$.</td>
<td>Required.</td>
</tr>
<tr>
<td>GR_PARAM_Q</td>
<td>FxFloat</td>
<td>4</td>
<td>Usage depends on choice of coordinate space.</td>
<td>Depth/fog iterator.</td>
<td>When using fog mode $GR_FOG_WITH_TABLE_ON_Q$ or $w$ buffering is enabled. Defaults to 1 if not defined.</td>
</tr>
<tr>
<td>GR_PARAM_STn</td>
<td>FxFloat</td>
<td>8</td>
<td>x and t coordinates for TMU n.</td>
<td>$s, t$ in range $[0,1]$ for one repeat of the texture.</td>
<td>Independent of aspect ratio.</td>
</tr>
<tr>
<td>GR_PARAM_Qn</td>
<td>FxFloat</td>
<td>4</td>
<td>q coordinate for TMU n.</td>
<td></td>
<td>When texture mapping with projected textures. Defaults to GR_PARAM_Q if not defined.</td>
</tr>
<tr>
<td>GR_PARAM_A</td>
<td>FxFloat</td>
<td>4</td>
<td>alpha value.</td>
<td>In the range $[0..1]$.</td>
<td>When using alpha blending, alpha testing, or anti-aliasing.</td>
</tr>
<tr>
<td>GR_PARAM_RGB</td>
<td>FxFloat</td>
<td>12</td>
<td>RGB triplet.</td>
<td>In the range $[0..1]$</td>
<td>Choose one of the two color formats.</td>
</tr>
<tr>
<td>GR_PARAM_PARGB</td>
<td>FxU32</td>
<td>4</td>
<td>Packed ARGB, one byte per component.</td>
<td>Each component is an integer in the range $[0..255]$</td>
<td></td>
</tr>
<tr>
<td>GR_PARAM_FOG_EXT (if FOGCOORD extension is supported)</td>
<td>FxFloat</td>
<td>4</td>
<td>Fog table index.</td>
<td>$f/w$ in the range $[0..1]$</td>
<td>When using fog mode $GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT$</td>
</tr>
</tbody>
</table>
When using window coordinates, choose from these values for `param`:

<table>
<thead>
<tr>
<th>param</th>
<th>type</th>
<th>size in bytes</th>
<th>description</th>
<th>values</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GR_PARAM_XY</code></td>
<td>FxFloat</td>
<td>8</td>
<td>x and y coordinates. Vertex snapping is no longer required.</td>
<td>x/w, y/w in the range [-2048..2047]</td>
<td>Required. Must be at offset 0.</td>
</tr>
<tr>
<td><code>GR_PARAM_Z</code></td>
<td>FxFloat</td>
<td>4</td>
<td>z coordinate.</td>
<td>Stored as 1/z. In the range [0..64K]</td>
<td>When z buffering is enabled.</td>
</tr>
<tr>
<td><code>GR_PARAM_Q</code></td>
<td>FxFloat</td>
<td>4</td>
<td>Usage depends on choice of coordinate space.</td>
<td>1/w</td>
<td>Required.</td>
</tr>
<tr>
<td><code>GR_PARAM_STn</code></td>
<td>FxFloat</td>
<td>8</td>
<td>s and t coordinates for TMU n.</td>
<td>Stored as s/lq, t/lq in the range [0..256] for one repeat of the texture. The range of the smaller dimension is limited by the aspect ratio. See Chapter 9.</td>
<td>When texture mapping.</td>
</tr>
<tr>
<td><code>GR_PARAM_Qn</code></td>
<td>FxFloat</td>
<td>4</td>
<td>q coordinate for TMU n.</td>
<td>In the range [0..255]</td>
<td>When texture mapping with projected textures. Defaults to GR_PARAM_Q if not defined or if disabled.</td>
</tr>
<tr>
<td><code>GR_PARAM_A</code></td>
<td>FxFloat</td>
<td>4</td>
<td>alpha value.</td>
<td>In the range [0..255]</td>
<td>When using alpha blending, alpha testing, or anti-aliasing.</td>
</tr>
<tr>
<td><code>GR_PARAM_RGB</code></td>
<td>FxFloat</td>
<td>12</td>
<td>RGB triplet.</td>
<td>In the range [0..255].</td>
<td>Choose one of the two color formats.</td>
</tr>
<tr>
<td><code>GR_PARAM_PARGB</code></td>
<td>FxU32</td>
<td>4</td>
<td>Packed ARGB, one byte per component.</td>
<td>Each component is an integer in the range [0..255].</td>
<td>When using fog mode [GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT]</td>
</tr>
<tr>
<td><code>GR_PARAM_FOG_EXT</code> (if FOGCOORD extension is supported)</td>
<td>FxFloat</td>
<td>4</td>
<td>Fog table index.</td>
<td>Stored as f/lq in the range [0..1].</td>
<td>When using fog mode [GR_FOG_WITH_TABLE_ON_FOGCOORD_EXT]</td>
</tr>
</tbody>
</table>

`offset` is the offset in bytes of the parameter data from the vertex pointer. The offset can be either positive or negative.

`mode` is either `GR_PARAM_ENABLE` or `GR_PARAM_DISABLE`. Disabling a parameter will potentially cause it to inherit the last known value. When a parameter is disabled the offset argument is ignored. Disabling a mandatory parameter like `GR_PARAM_XY` will cause a fatal Glide error.

`grVertexLayout` modifies the behavior of all `grDrawVertex*` APIs.
Notes

Glide 3.0 is the first release to support `grVertexLayout`.

The `GrVertex` structure has disappeared in Glide 3.0, to be replaced by a user-defined structure whose layout is communicated through calls to `grVertexLayout`. To create the old structure, define it in the application and execute the appropriate calls:

```c
typedef struct{
    float x, y, z; /* X, Y, Z */
    float r, g, b; /* R, G, B */
    float ooz; /* 65535/Z (used for Z-buffering) */
    float a; /* Alpha */
    float oow; /* 1/W (used for W-buffering, texturing) */
    GrTmuVertex tmuvtx[GLIDE_NUM_TMU];
} MyVertex; /* old GrVertex */
```

```c
grCoordinateSpace(GR_WINDOW_COORDS);
grVertexLayout(GR_PARAM_XY, 0, GR_PARAM_ENABLE);
grVertexLayout(GR_PARAM_RGB, 12, GR_PARAM_ENABLE);
grVertexLayout(GR_PARAM_Z, 24, GR_PARAM_ENABLE);
grVertexLayout(GR_PARAM_A, 28, GR_PARAM_ENABLE);
grVertexLayout(GR_PARAM_Q, 32, GR_PARAM_ENABLE);
grVertexLayout(GR_PARAM_ST0, 36, GR_PARAM_ENABLE);
```

To conserve memory, define only the parameters that will be used by the rendering mode and coordinate space settings you have chosen.

See also

`grDrawVertexArray`, `grFogMode`, `grGet`, `grGetString`, `grGlideGetVertexLayout`, `grGlideSetVertexLayout`
NAME

grViewport – define a viewport

C SPECIFICATION

void grViewport (FxI32 x, FxI32 y, F132 width, FxI32 height )

PARAMETERS

x, y The origin of the viewport, relative to the screen origin.

width, height The width and height of the viewport.

DESCRIPTION

grViewport specifies the viewport transformation. The current grSstOrigin setting determines whether
x and y specify the upper left corner or the lower left corner. Negative width and height are allowed and
mirror the image about the x or y axis. If (x, y) represent normalized screen coordinates, then the window
coordinates (x, x) are computed as:

\[ x_{\text{win}} = (x_{s} + 1)(\text{width}/2) + x \]

and

\[ y_{\text{win}} = (y_{s} + 1)(\text{height}/2) + y \]

NOTES

Glide 3.0 is the first release to support grViewport.

SEE ALSO
gu3dfGetInfo

NAME

gu3dfGetInfo – get information about the mipmap stored in a .3DF file

C SPECIFICATION

FxBool gu3dfGetInfo ( const char *filename, Gu3dfInfo *info )

PARAMETERS

filename Name of the .3DF file.
info Pointer to a Gu3dfInfo structure to fill with information about the mipmap.

DESCRIPTION

gu3dfGetInfo allows an application to determine relevant information about a .3DF file located on disk. The information is assigned to the appropriate member elements of the info structure. The Gu3dfInfo structure is defined in glide.h.

After an application has determined the characteristics of a .3DF mipmap, it is responsible for allocating system memory for the mipmap. This pointer is stored in the info->data pointer and used by gu3dfLoad.

NOTES

SEE ALSO

gu3dfLoad
gu3dfLoad

NAME

gu3dfLoad – load a .3DF file into system memory

C SPECIFICATION

FxBool gu3dfLoad( const char *filename, Gu3dfInfo *info )

PARAMETERS

filename Name of the file to load.
info Pointer to a Gu3dfInfo structure that gu3dfLoad fills in after loading the file.

DESCRIPTION

gu3dfLoad loads a .3DF file specified by filename into the pointer specified by info->data. gu3dfLoad returns FXTRUE if the file was successfully loaded; otherwise it returns FXFALSE. It is assumed the info structure passed has been appropriately configured with a call to gu3dfGetInfo.

NOTES

SEE ALSO

gu3dfGetInfo
guFogGenerateExp

NAME

guFogGenerateExp – generate an exponential fog table

C SPECIFICATION

void guFogGenerateExp( GrFog_t *fogTable, float density )

PARAMETERS

fogTable Pointer to an array that will receive the generated fog table values.
density The fog density, typically between 0.0 and 1.0.

DESCRIPTION

guFogGenerateExp generates an exponential fog table according to the equation:

\[ e^{-density \cdot w} \]

where \( w \) is the eye-space \( w \) coordinate associated with the fog table entry. The resulting fog table is copied into \( fogTable \).

NOTES

The fog table is normalized (scaled) such that the last entry is maximum fog (255).

SEE ALSO

grFogMode, grFogTable, guFogGenerateExp2, guFogGenerateLinear, guFogTableIndexToW
**guFogGenerateExp2**

**NAME**

*guFogGenerateExp2* – generate an exponential squared fog table

**C SPECIFICATION**

```c
void guFogGenerateExp2( GrFog_t *fogTable, float density )
```

**PARAMETERS**

- `fogTable` Pointer to an array that will receive the generated fog table values.
- `density` The fog density, typically between 0.0 and 1.0.

**DESCRIPTION**

*guFogGenerateExp2* generates an exponential squared fog table according to the equation:

\[
e^{-\text{density} \cdot w^2}
\]

where \(w\) is the eye-space \(w\) coordinate associated with the fog table entry. The resulting fog table is copied into `fogTable`.

**NOTES**

The fog table is normalized (scaled) such that the last entry is maximum fog (255).

**SEE ALSO**

*grFogMode, grFogTable, guFogGenerateExp, guFogGenerateLinear, guFogTableIndexToW*
guFogGenerateLinear

NAME

guFogGenerateLinear – generate a linear fog table

C SPECIFICATION

void guFogGenerateLinear( GrFog_t *fogTable,
                         float near,
                         float far )

PARAMETERS

fogTable Pointer to an array that will receive the generated fog table values.
near The eye-space w coordinate where minimum fog exists.
far The eye-space w coordinate where maximum fog exists.

DESCRIPTION

guFogGenerateLinear generates a linear (in eye-space) fog table according to the equation:

\[(w – near)/(far – near)\]

where \(w\) is the eye-space \(w\) coordinate associated with the fog table entry. The resulting fog table is copied into fogTable.

NOTES

The fog table is clamped so that all values are between minimum fog (0) and maximum fog (255).

guFogGenerateLinear fog is linear in eye-space \(w\), not in screen-space.

SEE ALSO

grFogMode, grFogTable, guFogGenerateExp, guFogGenerateExp2, guFogTableIndexToW
guFogTableIndexToW

NAME

guFogTableIndexToW – convert a fog table index to a floating point eye-space w value

C SPECIFICATION

float guFogTableIndexToW(int i)

PARAMETERS

i The fog table index, between 0 and GR_FOG_TABLE_SIZE.

DESCRIPTION

guFogTableIndexToW returns the floating point fog coordinate value associated with entry i in a fog table. Because fog table entries are non-linear, it is not straight forward to initialize a fog table. guFogTableIndexToW assists by converting fog table indices to eye-space w values.

NOTES

guFogTableIndexToW returns the following:

\[
pow(2.0, 3.0 + (\text{double})(i >> 2)) / (8 - (i \& 3));
\]

SEE ALSO

grFogMode, grFogTable, guFogGenerateExp, guFogGenerateExp2, guFogGenerateLinear
guGammaCorrectionRGB

NAME

guGammaCorrectionRGB – set up gamma correction tables

C SPECIFICATION

void guGammaCorrectionRGB (FxFloat gRed, FxFloat gGreen, FxFloat gBlue)

PARAMETERS

gRed The gamma correction value for the red component.
gGreen The gamma correction value for the green component.
gBlue The gamma correction value for the blue component.

DESCRIPTION

guGammaCorrectionRGB sets the gamma tables by computing a gamma correction curve for each color
component using the equation:

\[ c_\gamma = 255 \times (c/255)^{1/\gamma} \]

where \( c \) is the original red, green, or blue color component in the range \([0, 255]\), \( \gamma \) is the component-
specific gamma value, and \( c_\gamma \) is the gamma-corrected color component value.

The gamma-corrected values are used to generate a hardware-dependent gamma table, which is
automatically downloaded.

NOTES

Glide 3.0 is the first release to support guGammaCorrectionRGB. It replaces the now-obsolete
grGammaCorrectionValue.

SEE ALSO
<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Publication Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUTH74</td>
<td>Sutherland, I. E. and G. W. Hodgman</td>
<td>“Reentrant Polygon Clipping”, <em>CACM</em> 17(1),</td>
<td>32-42</td>
</tr>
<tr>
<td>WILL83</td>
<td>Williams, L.</td>
<td>“Pyramidal Parametrics”, <em>SIGGRAPH 83</em>, 1-</td>
<td></td>
</tr>
</tbody>
</table>