Glide Reference Manual

Programming the 3Dfx Interactive Glide Rasterization Library 2.4

Document Release 017
25 July 1997

3Dfx Interactive, Inc.
4435 Fortran Drive
San Jose, CA 95134

Introduction

This document is the official programming reference for version 2.0 of 3Dfx Interactive Glide Rasterization Library. The Glide Library is a low-level rendering and state management sub-routine library that serves as a thin layer over the register level interface of the Voodoo Graphics™ family of graphics hardware. Glide permits easy and efficient implementation of 3D rendering libraries, games, and drivers on the Voodoo Graphics hardware. Glide only implements operations that are natively supported by the Voodoo Graphics family of graphics 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 Voodoo Graphics family of graphics hardware into a set of C-callable APIs, Glide allows developers targeting the Voodoo Graphics family of 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.

Voodoo Graphics consists of a Pixel*fx* chip which performs pixel rendering operations and manages the video frame buffer, and one or more Texel*fx* chips which perform texture mapping operations. Each Texel*fx* chip contains one Texture Mapping Unit, or TMU. The term TMU is used throughout the rest of this manual.

Voodoo Graphics was internally code-named SST-1, or SST for short. Some of the API names, e.g. **`grSstWinOpen`**, still reflect the internal code name. In general, Voodoo Graphics is used in the API description, but SST is used in the API names.

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:

Type Name	Format	ANSI C type
`FxI8`	8-bit signed	**`signed char`**
`FxU8`	8-bit unsigned	**`unsigned char`**
`FxU16`	16-bit unsigned	**`unsigned short int`**
`FxI16`	16-bit signed	**`signed short int`**
`FxU32`	32-bit unsigned	**`unsigned long int`**
`FxI32`	32-bit signed value	**`signed long int`**
`FxBool`	32-bit signed	**`long int`**

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.

grAADrawLine

NAME

grAADrawLine – draw an anti-aliased line segment

C SPECIFICATION

```
void grAADrawLine(GrVertex *va, GrVertex *vb)
```

PARAMETERS

va, vb	Vertices describing the line segment.

DESCRIPTION

Glide draws an anti-aliased line segment between the two vertices 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

Resultant lines will be somewhat ‘fatter’ than expected.

SEE ALSO

grAADrawPoint, **grAADrawTriangle**, **grAlphaBlendFunction**, **grAlphaCombine**

grAADrawPoint

NAME

grAADrawPoint – draw an anti-aliased point

C SPECIFICATION

```
void grAADrawPolygon(GrVertex *p)
```

PARAMETERS

p	The point to draw.

DESCRIPTION

Glide draws an anti-aliased point by rendering four pixels and 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

SEE ALSO

grAADrawLine, **grAADrawTriangle**, **grAlphaBlendFunction**, **grAlphaCombine**

grAADrawPolygon

NAME

grAADrawPolygon – draw an anti-aliased convex polygon

C SPECIFICATION

```
void grAADrawPolygon(int nVerts, const int ilist[], const GrVertex vlist[])
```

PARAMETERS

nVerts	Number of vertices in the polygon.
ilist	Array of indices into *vlist*.
vlist	Array of vertices indexed by *ilist*.

DESCRIPTION

Glide draws an anti-aliased polygon from an array of vertices and indices into this array.

NOTES

See notes on **grAADrawTriangle** for the limitations of this routine. Further limitations are:

- All exterior polygon edges are anti-aliased
- Only convex polygons are drawn properly.

SEE ALSO

grAADrawPolygonVertexList, **grAADrawTriangle**

grAADrawPolygonVertexList

NAME

grAADrawPolygonVertexList – draw an anti-aliased convex polygon

C SPECIFICATION

```
void grAADrawPolygonVertexList(int nVerts, const GrVertex vlist[])
```

PARAMETERS

nVerts	Number of vertices in the polygon.
vlist	Array of vertices in the polygon.

DESCRIPTION

Glide draws an anti-aliased polygon from an array of vertices.

NOTES

See notes on **grAADrawTriangle** for the limitations of this routine. Further limitations are:

- All exterior polygon edges are anti-aliased.
- Only convex polygons are drawn properly.
- Vertices are drawn in the order presented in **vlist[]**.

SEE ALSO

grAADrawPolygon, **grAADrawTriangle**, **grAADrawLine**, **grAADrawPoint**

grAADrawTriangle

NAME

grAADrawTriangle – draw an anti-aliased triangle

C SPECIFICATION

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

PARAMETERS

a	Vertex a.
b	Vertex b.
c	Vertex c.
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.

SEE ALSO

grAADrawLine, **grAADrawPoint**, **grAlphaBlendFunction**, **grAlphaCombine**

grAlphaBlendFunction

NAME

grAlphaBlendFunction – specify the alpha blending function

C SPECIFICATION

```
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
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	

alpha_df Specifies the alpha destination blending factor. 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	

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* specifies which of nine methods is used to scale the source color and alpha components. *rgb_df* and *alpha_df* specifies which of eight methods is used to scale the destination color and alpha components.

Alpha blending is defined by the equations:

$$R = \min (255, R_s\, s_R + R_d\, d_R)$$

$$G = \min (255, G_s\, s_G + G_d\, d_G)$$

$$B = \min (255, B_s\, s_B + B_d\, d_B)$$

$$A = \min (255, A_s\, s_A + A_d\, d_A)$$

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:

Blending Factor	Component Blend Factor
GR_BLEND_ZERO	0
GR_BLEND_ONE	1
GR_BLEND_SRC_COLOR	C_s / 255
GR_BLEND_ONE_MINUS_SRC_COLOR	1 – C_s / 255
GR_BLEND_DST_COLOR	C_d / 255
GR_BLEND_ONE_MINUS_DST_COLOR	1 – C_d / 255
GR_BLEND_SRC_ALPHA	A_s / 255
GR_BLEND_ONE_MINUS_SRC_ALPHA	1 – A_s / 255
GR_BLEND_DST_ALPHA	A_d / 255
GR_BLEND_ONE_MINUS_DST_ALPHA	1 – A_d / 255
GR_BLEND_ALPHA_SATURATE	min (A_s / 255, 1 – A_d / 255)
GR_BLEND_PREFOG_COLOR	*color before fog is applied*

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

grAlphaBlendFunction(GR_BLEND_ONE, GR_BLEND_ZERO, GR_BLEND_ONE, GR_BLEND_ZERO)

NOTES

The source of incoming alpha and color are determined by **grAlphaCombine** and **grColorCombine** respectively.

Alpha blending that requires a destination alpha is mutually exclusive of either depth buffering or triple buffering. Attempting to use **GR_BLEND_DST_ALPHA**, **GR_BLEND_ONE_MINUS_DST_ALPHA**, or **GR_BLEND_ALPHA_SATURATE** when depth buffering or triple buffering are enabled will have undefined results.

For alpha source and destination blend function factor parameters, Voodoo Graphics supports only **GR_BLEND_ZERO** and **GR_BLEND_ONE**.

GR_BLEND_PREFOG_COLOR is useful when applying fog to a scene generated in multiple passes. See the *Glide Programming Guide* for more information.

SEE ALSO

grAADrawLine, **grAADrawPoint**, **grAADrawTriangle**, **grAlphaCombine**, **grColorCombine**

grAlphaCombine

NAME

grAlphaCombine – configure the alpha combine unit

C SPECIFICATION

```
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**.

Combine Function	Computed Alpha
ZERO	0
LOCAL	A_{local}
LOCAL_ALPHA	A_{local}
SCALE_OTHER **BLEND_OTHER**	$f * A_{other}$
SCALE_OTHER_ADD_LOCAL	$f * A_{other} + A_{local}$
SCALE_OTHER_ADD_LOCAL_ALPHA	$f * A_{other} + A_{local}$
SCALE_OTHER_MINUS_LOCAL	$f * (A_{other} - A_{local})$
SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL **BLEND**	$f * (A_{other} - A_{local}) + A_{local}$ $\equiv f * A_{other} + (1-f) * A_{local}$
SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (A_{other} - A_{local}) + A_{local}$
SCALE_MINUS_LOCAL_ADD_LOCAL **BLEND_LOCAL**	$f * (- A_{local}) + A_{local}$ $\equiv (1-f) * A_{local}$
SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (- A_{local}) + A_{local}$

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

Combine Factor	Scale Factor (*f*)
GR_COMBINE_FACTOR_NONE	Unspecified
GR_COMBINE_FACTOR_ZERO	0
GR_COMBINE_FACTOR_LOCAL	A_{local} / 255
GR_COMBINE_FACTOR_OTHER_ALPHA	A_{other} / 255
GR_COMBINE_FACTOR_LOCAL_ALPHA	A_{local} / 255
GR_COMBINE_FACTOR_TEXTURE_ALPHA	$A_{texture}$ / 255
GR_COMBINE_FACTOR_ONE	1
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL	$1 - A_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA	$1 - A_{other}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA	$1 - A_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_TEXTURE_ALPHA	$1 - A_{texture}$ / 255

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

Local Combine Source	Local Alpha (A_{local})
GR_COMBINE_LOCAL_NONE	Unspecified alpha
GR_COMBINE_LOCAL_ITERATED	Iterated vertex alpha
GR_COMBINE_LOCAL_CONSTANT	Constant alpha
GR_COMBINE_LOCAL_DEPTH	High 8 bits from iterated vertex *z*

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

Other Combine Source	Other Alpha (A_{other})
GR_COMBINE_OTHER_NONE	Unspecified alpha
GR_COMBINE_OTHER_ITERATED	Iterated vertex alpha
GR_COMBINE_OTHER_TEXTURE	Alpha from texture map
GR_COMBINE_OTHER_CONSTANT	Constant alpha

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 **GR_COMBINE_FACTOR_ZERO**, **GR_COMBINE_LOCAL_CONSTANT**, and **GR_COMBINE_OTHER_CONSTANT** respectively.

NOTES

The local alpha value specified by the *local* parameter and the other alpha value specified by the *other* parameter are used by the color combine unit.

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

grColorCombine, **grConstantColorValue**, **grDrawTriangle**

grAlphaControlsITRGBLighting

NAME

`grAlphaControlsITRGBLighting` – enables/disables alpha controlled lighting

C SPECIFICATION

```
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.

Value of *enable*	MSB of Alpha Channel	Color Combine Local Color
`FXTRUE`	**`0`**	Iterated RGB
`FXTRUE`	**`1`**	**`grConstantColorValue`**
`FXFALSE`	**`0`**	Set by **`grColorCombine`**
`FXFALSE`	**`1`**	Set by **`grColorCombine`**

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

```
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:

function	**Comparison Function**
GR_CMP_NEVER	Never passes.
GR_CMP_LESS	Passes if the incoming alpha value is less than the constant alpha reference value.
GR_CMP_EQUAL	Passes if the incoming alpha value is equal to the constant alpha reference value.
GR_CMP_LEQUAL	Passes if the incoming alpha value is less than or equal to the constant alpha reference value.
GR_CMP_GREATER	Passes if the incoming alpha value is greater than the constant alpha reference value.
GR_CMP_NOTEQUAL	Passes if the incoming alpha value is not equal to the constant alpha reference value.
GR_CMP_GEQUAL	Passes if the incoming alpha value is greater than or equal to the constant alpha reference value.
GR_CMP_ALWAYS	Always passes

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 linear frame buffer bypass is enabled (see **grLfbLock**).

NOTES

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

SEE ALSO

grAlphaCombine, **grAlphaTestReferenceValue**, **grLfbLock**

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 linear frame buffer bypass is enabled (see **grLfbLock**).

NOTES

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

SEE ALSO

grAlphaCombine, **grAlphaTestFunction**, **grLfbLock**

grBufferClear

NAME

grBufferClear – clear the buffers to the specified values

C SPECIFICATION

void grBufferClear(GrColor_t color, GrAlpha_t alpha, FxU16 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 **grAlphaBlendFunction**).
depth	16-bit unsigned value used for clearing the depth buffer (ignored if depth buffering is not enabled).

DESCRIPTION

Clears the appropriate buffers with the given values. **grClipWindow** defines the area within the buffer to be cleared. Any buffers that are enabled are cleared by **grBufferClear**. 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 **grDepthMask** and/or **grColorMask** 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 *depth* parameter can be one of the constants **GR_ZDEPTHVALUE_NEAREST**, **GR_ZDEPTHVALUE_FARTHEST**, **GR_WDEPTHVALUE_NEAREST**, **GR_WDEPTHVALUE_FARTHEST**, 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 *w* 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 at twice the rate of triangle rendering, therefore the performance cost of clearing the buffer is half the cost of rendering a rectangle. Clearing buffers is not always necessary 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

grClipWindow, **grColorMask**, **grDepthMask**, **grRenderBuffer**

grBufferNumPending

NAME

grBufferNumPending – return the number of queued buffer swap requests

C SPECIFICATION

```
int grBufferNumPending( void )
```

PARAMETERS

none

DESCRIPTION

Voodoo Graphics has a large command queue located in off-screen memory. When time-consuming commands, such as large triangles or buffer swaps are executing, subsequent commands are placed in a command queue, including buffer swap requests. **grBufferNumPending** returns the number of queued buffer swap requests. An application typically wants to monitor this value and not get too far ahead of the rendering process.

NOTES

The maximum value returned is 7, even though there may be more buffer swap requests in the queue. To minimize rendering latency in response to interactive input, **grBufferNumPending** should be called in a loop once per frame until the returned value is less than some small number such as 1, 2, or 3.

SEE ALSO

grBufferSwap

grBufferSwap

NAME

grBufferSwap – exchange front and back buffers

C SPECIFICATION

```
void grBufferSwap( int swap_interval )
```

PARAMETERS

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

DESCRIPTION

grBufferSwap 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 **grBufferSwap** is called. If the application is double buffering, the Voodoo 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.

grBufferSwap waits until there are fewer than 7 pending buffer swap requests in the Voodoo Graphics command FIFO before returning.

SEE ALSO

grBufferNumPending

grChromakeyMode

NAME

grChromakeyMode – enable/disable hardware chroma-keying

C SPECIFICATION

```
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**

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**

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 640×480 mode and (0,0,800,600) for 800×600 mode. To disable clipping simply set 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**

grColorCombine

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:

Combine Function	computed color
GR_COMBINE_FUNCTION_ZERO	0
GR_COMBINE_FUNCTION_LOCAL	C_{local}
GR_COMBINE_FUNCTION_LOCAL_ALPHA	A_{local}
GR_COMBINE_FUNCTION_SCALE_OTHER GR_COMBINE_FUNCTION_BLEND_OTHER	$f * C_{other}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL	$f * C_{other} + C_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL_ALPHA	$f * C_{other} + A_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL	$f * (C_{other} - C_{local})$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND	$f * (C_{other} - C_{local}) + C_{local}$ $\equiv f * C_{other} + (1 - f) * C_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (C_{other} - C_{local}) + A_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND_LOCAL	$f * (- C_{local}) + C_{local}$ $\equiv (1 - f) * C_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (- C_{local}) + A_{local}$

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

Combine Factor	Scale Factor (*f*)
`GR_COMBINE_FACTOR_NONE`	Unspecified
`GR_COMBINE_FACTOR_ZERO`	0
`GR_COMBINE_FACTOR_LOCAL`	C_{local} / 255
`GR_COMBINE_FACTOR_OTHER_ALPHA`	A_{other} / 255
`GR_COMBINE_FACTOR_LOCAL_ALPHA`	A_{local} / 255
`GR_COMBINE_FACTOR_TEXTURE_ALPHA`	$A_{texture}$ / 255
`GR_COMBINE_FACTOR_ONE`	1
`GR_COMBINE_FACTOR_ONE_MINUS_LOCAL`	1 – C_{local} / 255
`GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA`	1 – A_{other} / 255
`GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA`	1 – A_{local} / 255
`GR_COMBINE_FACTOR_ONE_MINUS_TEXTURE_ALPHA`	1 – $A_{texture}$ / 255

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

Local Combine Source	Local Color (C_{local})
`GR_COMBINE_LOCAL_NONE`	Unspecified color
`GR_COMBINE_LOCAL_ITERATED`	Iterated vertex color (Gouraud shading)
`GR_COMBINE_LOCAL_CONSTANT`	Constant color

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

Other Combine Source	Other Color (C_{other})
`GR_COMBINE_OTHER_NONE`	Unspecified color
`GR_COMBINE_OTHER_ITERATED`	Iterated vertex color (Gouraud shading)
`GR_COMBINE_OTHER_TEXTURE`	Color from texture map
`GR_COMBINE_OTHER_CONSTANT`	Constant color

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

DESCRIPTION

`grColorCombine` configures the color combine unit of the Voodoo 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 **`grColorCombine(GR_COMBINE_FUNCTION_SCALE_OTHER, GR_COMBINE_FACTOR_ONE, GR_COMBINE_LOCAL_ITERATED, GR_COMBINE_OTHER_ITERATED)`**.

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_{local} is the *local* alpha value selected by **grAlphaCombine** and A_{other} 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**

grColorMask

NAME

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

C SPECIFICATION

```
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

```
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**

grCullMode

NAME

grCullMode – set the cull mode

C SPECIFICATION

```
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**.

Origin Location	Triangle Orientation	Signed Area
GR_ORIGIN_LOWERLEFT	clockwise	negative
GR_ORIGIN_LOWERLEFT	counter-clockwise	positive
GR_ORIGIN_UPPERLEFT	clockwise	positive
GR_ORIGIN_UPPERLEFT	counter-clockwise	negative

NOTES

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

SEE ALSO

grDrawTriangle

grDepthBiasLevel

NAME

grDepthBiasLevel – set the depth bias level

C SPECIFICATION

```
void grDepthBiasLevel( FxI16 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

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

```
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:

func	Comparison Function
GR_CMP_NEVER	Never passes.
GR_CMP_LESS	Passes if the pixel's depth value is less than the stored depth value.
GR_CMP_EQUAL	Passes if the pixel's depth value is equal to the stored depth value.
GR_CMP_LEQUAL	Passes if the pixel's depth value is less than or equal to the stored depth value.
GR_CMP_GREATER	Passes if the pixel's depth value is greater than the stored depth value.
GR_CMP_NOTEQUAL	Passes if the pixel's depth value is not equal to the stored depth value.
GR_CMP_GEQUAL	Passes if the pixel's depth value is greater than or equal to the stored depth value.
GR_CMP_ALWAYS	Always passes

The default comparison function is **GR_CMP_LESS**.

NOTES

SEE ALSO

grDepthBufferMode, **grDepthMask**, **grDepthBiasLevel**, **grLfbConstantDepth**

grDepthBufferMode

NAME

grDepthBufferMode – set the depth buffering mode

C SPECIFICATION

```
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_ZBUFFER_COMPARE_TO_BIAS**, or **GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS**. If **GR_DEPTHBUFFER_ZBUFFER** or **GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS** is selected, then the Voodoo 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 Voodoo Graphics subsystem will perform 16-bit floating point *w* buffering. By default the depth buffer node 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 Pixel*fx* chip (use **grSstQueryHardware** to obtain the revision number).

The Glide 2.1 release was the first release that supported **GR_DEPTHBUFFER_ZBUFFER_COMPARE_TO_BIAS** and **GR_DEPTHBUFFER_WBUFFER_COMPARE_TO_BIAS**.

SEE ALSO

grDepthBufferFunction, **grDepthMask**, **grDepthBiasLevel**, **grLfbConstantDepth**

grDepthMask

NAME

grDepthMask – enable/disable writing into the depth buffer

C SPECIFICATION

```
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, depth, and triple 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**

grDisableAllEffects

NAME

grDisableAllEffects – disable all special effects in the Voodoo Graphics subsystem

C SPECIFICATION

```
void grDisableAllEffects( void )
```

PARAMETERS

none

DESCRIPTION

grDisableAllEffects disables all special effects (alpha blending, alpha testing, chroma-keying, fog, depth buffering) in the Voodoo 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

```
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 GrVertex *a, const GrVertex *b )
```

PARAMETERS

a, b	Endpoints and attributes of the line.

DESCRIPTION

Renders a one-pixel-wide arbitrarily oriented line with the given endpoints. All current Glide attributes will affect the appearance of the line.

NOTES

SEE ALSO

grDrawPoint, **grDrawTriangle**

grDrawPlanarPolygon

NAME

grDrawPlanarPolygon – draw a convex planar polygon

C SPECIFICATION

```
void grDrawPlanarPolygon( int nVerts, int ilist[], const GrVertex vlist[] )
```

PARAMETERS

nVerts	Number of vertices in the polygon.
ilist	Array of indices into *vlist*.
vlist	Array of vertices indexed by *ilist*.

DESCRIPTION

grDrawPlanarPolygon renders a convex polygon of an arbitrary number of vertices. The polygon's coordinates and parameters are assumed to be planar, so parameter gradients will be computed only a single time for the entire polygon. It is assumed that the polygon does not need any form of clipping.

NOTES

The convex polygon is triangulated from the first vertex, *vlist*[*ilist*[0]].

Note that *all* parameters are assumed to be planar, including color and texture parameters. Any performance improvement realized by using **grDrawPlanarPolygon** or **grDrawPlanarPolygonVertexList** comes from computing gradients once for the whole polygon rather than once for each pair of vertices. **grDrawPlanarPolygon** and **grDrawPlanarPolygonVertexList** will not improve performance on systems with hardware triangle setup and may be come obsolete in future releases of Glide.

SEE ALSO

grDrawPlanarPolygonVertexList, **grDrawPolygon**, **grDrawTriangle**

grDrawPlanarPolygonVertexList

NAME

grDrawPlanarPolygonVertexList – draw a convex planar polygon

C SPECIFICATION

```
void grDrawPlanarPolygonVertexList( int nVerts, const GrVertex vlist[] )
```

PARAMETERS

nVerts	Number of vertices in the polygon.
vlist	Array of vertices in the polygon.

DESCRIPTION

grDrawPlanarPolygonVertexList renders a convex polygon of an arbitrary number of vertices. The polygon's coordinates and parameters are assumed to be planar, so parameter gradients will be computed only a single time for the entire polygon. It is assumed that the polygon does not need any form of clipping.

NOTES

The convex polygon is triangulated from the first vertex, *vlist*[0].

The Glide 2.1 release was the first release that supported **grDrawPlanarPolygonVertexList**.

SEE ALSO

grDrawPlanarPolygon, **grDrawPolygon**, **grDrawTriangle**

grDrawPoint

NAME

grDrawPoint – draw a point

C SPECIFICATION

```
void grDrawPoint( const GrVertex *a )
```

PARAMETERS

a	Location and attributes of the point.

DESCRIPTION

Renders a single 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.

NOTES

SEE ALSO

grDrawLine, **grDrawTriangle**, **grLfbLock**

grDrawPolygon

NAME

grDrawPolygon – draw a convex non-planar polygon

C SPECIFICATION

```
void grDrawPolygon( int nVerts, int ilist[], const GrVertex vlist[] )
```

PARAMETERS

nVerts	Number of vertices in the polygon.
ilist	Array of indices into *vlist*.
vlist	Array of vertices indexed by *ilist*.

DESCRIPTION

grDrawPolygon renders a convex polygon with an arbitrary number of vertices. The polygon's coordinates are assumed to be planar and to lie within the clipping window. Parameters need not be planar, and parameter gradients will be computed multiple times across the face of the polygon.

NOTES

The convex polygon is triangulated from the first vertex, *vlist*[*ilist*[0]].

SEE ALSO

grDrawPlanarPolygon, **grDrawPolygonVertexList**, **grDrawTriangle**

grDrawPolygonVertexList

NAME

grDrawPolygonVertexList – draw a convex non-planar polygon

C SPECIFICATION

```
void grDrawPolygonVertexList( int nVerts, const GrVertex vlist[] )
```

PARAMETERS

nVerts	Number of vertices in the polygon.
vlist	Array of vertices in the polygon.

DESCRIPTION

grDrawPolygonVertexList renders a convex polygon of an arbitrary number of vertices. The polygon's coordinates are assumed to be planar and to lie within the clipping window. Parameters need not be planar, and parameter gradients will be computed multiple times across the face of the polygon.

NOTES

The convex polygon is triangulated from the first vertex, *vlist*[0].

SEE ALSO

grDrawPlanarPolygon, **grDrawPlanarPolygonVertexList**, **grDrawPolygon**, **grDrawTriangle**

grDrawTriangle

NAME

grDrawTriangle – draw a triangle

C SPECIFICATION

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

PARAMETERS

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

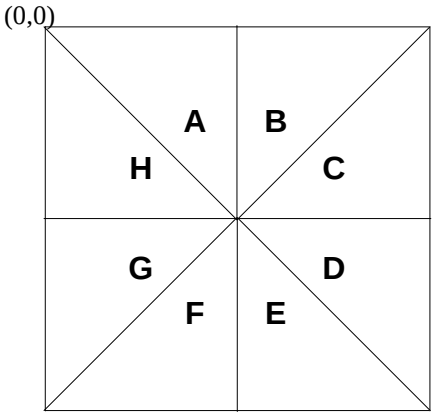
DESCRIPTION

Renders an arbitrarily oriented triangle. All current Glide attributes will affect the appearance of the triangle. 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 triangle D, the pixel lies exactly on the bottom and left edges and is therefore considered to be inside the triangle.



NOTES

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

SEE ALSO

grDrawLine, **grDrawPoint**, **grDrawPolygon**

grErrorSetCallback

NAME

grErrorSetCallback – install a user-defined error-handler

C SPECIFICATION

```
void grErrorSetCallback( void (*function)(const char *string, FxBool fatal) )
```

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

grFogColorValue

NAME

grFogColorValue – set the global fog color

C SPECIFICATION

```
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 either from iterated alpha or a user downloaded fog table based on the pixel's *w* component, depending on the current **grFogMode**.

The new color is computed as follows:

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

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

```
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_ITERATED_ALPHA**, and **GR_FOG_WITH_TABLE**. The last two modes can be used in combination with **GR_FOG_ADD2** or **GR_FOG_MULT2** to tailor the fog equation, as shown below.

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

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

Factor $\boldsymbol{f}$ is determined by *mode*. If *mode* is **GR_FOG_WITH_ITERATED_ALPHA**, then $\boldsymbol{f}$ is equal to the integer bits of iterated alpha. If *mode* is **GR_FOG_WITH_TABLE**, then $\boldsymbol{f}$ is computed by interpolating between fog table entries, where the fog table is indexed with a floating point representation of the pixel's *w* component.

if mode sets	*the fog equation is*	*where c_{in} is the color entering the fog unit, c_{out} is the result of fogging, c_{fog} is the fog color and*
GR_FOG_DISABLE	$c_{out} = c_{in}$	
GR_FOG_WITH_ITERATED_ALPHA	$c_{out} = a_i\, c_{fog} + (1-a_i)c_{in}$	*a_i is the high order byte of the iterated alpha value*
GR_FOG_WITH_TABLE	$c_{out} = \boldsymbol{f}_{fog[w]} \bullet c_{fog} + (1-\boldsymbol{f}_{fog[w]}) \bullet c_{in}$	*$\boldsymbol{f}_{fog[w]}$ is computed by interpolating between entries in a fog table indexed with $\boldsymbol{w}$*
GR_FOG_ADD2	$c_{out} = (1-\boldsymbol{f})c_{in}$	*$\boldsymbol{f}$ can be either the high order byte of iterated alpha or computed from a fog table*
GR_FOG_MULT2	$c_{out} = \boldsymbol{f}\, c_{fog}$	*$\boldsymbol{f}$ can be either the high order byte of iterated alpha or computed from a fog 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

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

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_{fog}) with each rasterized pixel's color (C_{in}) using a blending factor ***f***. Factor ***f*** depends upon the most recent call to **grFogMode**. If the **grFogMode** is set to **GR_FOG_WITH_TABLE**, 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 *w* 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 **guFogTableIndexToW(i)** or by computing:

```
pow(2.0,3.0+(double)(i>>2)) / (8-(i&3));
```

The new color is computed as follows:

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

An exponential fog table can be generated by computing $(1-e^{-kw})*255$ 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

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

grFogMode, **grFogColorValue**, **guFogTableIndexToW**

grGammaCorrectionValue

NAME

grGammaCorrectionValue – set the gamma correction value

C SPECIFICATION

```
void grGammaCorrectionValue( float value )
```

PARAMETERS

value The new gamma value.

DESCRIPTION

grGammaCorrectionValue sets the gamma correction value used during video refresh. Gamma is a positive floating point value from 0.0 to 20.0. Typical values are in the range [1.3..2.2]. The default value is 1.0 (i.e. a linear ramp is used).

The displayed RGB value (RGB_{gamma}) is computed from the RGB value read from the frame buffer (RGB_{fb}) according to the following equation:

$$RGB_{gamma} = [(RGB_{fb}/255)^{1/gamma}]*255$$

NOTES

Gamma correction is supported on all SST-1 systems, and **grGammaCorrectionValue** works as described. It is a no-op on SST-96 systems, which may lack gamma correction hardware.

SEE ALSO

For more information on gamma correction, refer to [FOLE90].

grGlideGetVersion

NAME

grGlideGetVersion – return the version of Glide

C SPECIFICATION

```
void grGlideGetVersion( char version[80] )
```

PARAMETERS

version	Character array to receive the text string describing the Glide version.

DESCRIPTION

grGlideGetVersion fills *version* with a null-terminated text string that describes the Glide version.

NOTES

A sample version string is “Glide Version 2.2”.

The Glide 2.1 release was the first release to include **grGlideGetVersion**.

SEE ALSO

grGlideInit

grGlideGetState

NAME

grGlideGetState – get the current state of the current Voodoo Graphics subsystem

C SPECIFICATION

```
void grGlideGetState( GrState *state )
```

PARAMETERS

state	Pointer to a **GrState** structure where the state is to be stored.

DESCRIPTION

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

NOTES

SEE ALSO

grGlideSetState

grGlideInit

NAME

grGlideInit – initialize the Glide library

C SPECIFICATION

```
void grGlideInit( void )
```

PARAMETERS

none

DESCRIPTION

grGlideInit initializes the Glide library, performing tasks such as finding any installed Voodoo Graphics subsystems, allocating memory, and initializing state variables. **grGlideInit** must be called before any other Glide routines are called.

NOTES

grSstQueryBoards can be called before **grGlideInit**.

SEE ALSO

grGlideGetVersion, **grGlideShutdown**, **grSstWinOpen**, **grSstQueryBoards**, **grSstQueryHardware**, **grSstSelect**

grGlideSetState

NAME

grGlideSetState – set the state of the currently active Voodoo Graphics subsystem

C SPECIFICATION

```
void grGlideSetState( const GrState *state )
```

PARAMETERS

state Pointer to a **GrState** structure containing the new state.

DESCRIPTION

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

SEE ALSO

grGlideGetState

grGlideShutdown

NAME

grGlideShutdown – shut down the Glide library

C SPECIFICATION

```
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

grHints

NAME

grHints – informs Glide of special conditions regarding optimizations

C SPECIFICATION

```
void grHints( GrHints_t type, FxU32 hintMask )
```

PARAMETERS

type	Specifies the type of hint.
hintMask	A bitmask of ORed hints.

DESCRIPTION

grHints informs Glide of special conditions regarding optimizations and operation. Each type of hint controls a different optimization or mode of operation. Hints of a given type are ORed together into a *hintMask*. The default *hintMask* is **0x00**.

The **GR_HINT_STWHINT** hint type controls *stw* parameter optimization. By default, Glide assumes that all *w* coordinates (*oow*) in the **GrVertex** structure are identical, and that all *s* and *t* coordinates (*sow* and *tow*) are also identical. This greatly reduces the amount of work Glide has to perform when computing gradients for *s*, *t*, and *w*, and transferring data to the graphics hardware. The *stw* hints alert Glide that specific values in the **GrVertex** structure are different and that gradients need to be computed for these values.

The *stw* hints also specify Glide's source for the parameter values. There is an implicit ordering of TMUs within Glide, starting with TMU0, followed by TMU1, and TMU2. By default, Glide reads *s* and *t* coordinates from the **GrVertex** structure for the first TMU that is active. Whenever *s* and *t* coordinates are read they are transmitted to all subsequent TMUs. For example, if texturing is active in TMU1 but not active in TMU0, then *s* and *t* coordinates are read from **GrVertex.tmuvtx[1]** and broadcast to TMU1 and TMU2. Once *s* and *t* coordinates are read, they will not be read again unless a hint is specified. If one of the subsequent units has a unique or different parameter value, then a hint must be used. If a hint is specified, the parameter value will be read again and sent to the specified unit and all other units following it.

The *w* hints inform Glide where to look for *w* coordinates. The rule for the *w* coordinate is very simple: the *w* coordinate is read from **GrVertex.oow** and broadcast to all TMUs unless a *w* hint is specified. If a *w* hint is specified, then if *w* buffering or table-based fog is enabled, **GrVertex.oow** is read and sent to all TMUs first. Then *w* is read from the **GrVertex.tmuvtx[]** structure corresponding to the hint and broadcast to all subsequent TMUs.

The tables below describe the values:

Hint	Description
GR_STWHINT_ST_DIFF_TMU0	*s* and *t* for TMU0 are different than previous values.
GR_STWHINT_ST_DIFF_TMU1	*s* and *t* for TMU1 are different than previous values.
GR_STWHINT_ST_DIFF_TMU2	*s* and *t* for TMU2 are different than previous values.
GR_STWHINT_W_DIFF_TMU0	*w* for TMU0 is different than previous *w* values.
GR_STWHINT_W_DIFF_TMU1	*w* for TMU1 is different than previous *w* values.
GR_STWHINT_W_DIFF_TMU2	*w* for TMU2 is different than previous *w* values.

The **GR_HINT_ALLOW_MIPMAP_DITHER** hint type controls whether or not **GR_MIPMAP_NEAREST_DITHER** mode can be used. If *hintMask* is zero, then **GR_MIPMAP_NEAREST_DITHER** mode cannot be enabled with **grTexMipMapMode()**. This is the default. To allow **GR_MIPMAP_NEAREST_DITHER** mode to be used, specify a non-zero *hintMask* with the hint.

Dithered mipmapping is disabled by default because it can cause a performance loss of 20% to 30% in some cases. And since the presence or absence of mipmap dithering is not very noticeable, it is very hard to determine the cause of the performance loss. Therefore, Glide disallows this mode by default, but it can be used by calling **grHints**.

If dithered mipmapping is used, measure performance with and without it. If there is a significant performance difference, don't use it. The trade-off is that there may be visible mipmap bands, which can be eliminated by using trilinear mipmapping. On multiple TMU boards this is a one-pass operation, otherwise it requires two passes. Alternatively, dithered mipmapping can be allowed but disabled for most polygons and enabled only for those polygons that require it.

If there is no performance difference with and without dithered mipmapping, but the image quality did not improve with dithered mipmapping, don't use it. As you enhance or extend your program, you run the risk of creating a situation in which performance loss due to dithered mipmapping could occur. It is best to selectively enabled dithered mipmapping just for the polygons that require it.

NOTES

Since TMU0 is the first functional unit with *s* and *t* coordinates, the **GR_STWHINT_ST_DIFF_TMU0** hint need never be given.

grSstWinOpen initializes various Glide state variables, including hints. Thus, **grHints** should be called after **grSstWinOpen** if you want something other than the default hint settings.

SEE ALSO

grDrawLine, **grDrawPoint**, **grDrawTriangle**

grLfbConstantAlpha

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_888**, **GR_LFBWRITEMODE_555_DEPTH**, and **GR_LFBWRITEMODE_DEPTH_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**

grLfbConstantDepth

NAME

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

C SPECIFICATION

```
void grLfbConstantDepth( FxU16 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**, **GR_LFBWRITEMODE_8888**, and **GR_LFBWRITEMODE_ALPHA_ALPHA**, do not possess 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**

grLfbLock

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:

GR_LFB_READ_ONLY	*info.lfbPtr* should only be used for read access; writing to this pointer will have undefined effects on the graphics subsystem.
GR_LFB_WRITE_ONLY	*info.lfbPtr* should only be used for write access; reading from this pointer will yield undefined data.

The idle request flag can be one of:

GR_LFB_IDLE	The 3D engine will be idled before **grLfbLock** returns. This is the default behavior if no idle request flag is specified.
GR_LFB_NOIDLE	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.

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_LFBWRITEMODE_ANY**. If the destination pixel format specified is not supported on the target hardware, then the lock will fail. Supported pixels formats are:

GR_LFBWRITEMODE_565	Frame buffer accepts 16-bit RGB 565 pixel data.
GR_LFBWRITEMODE_555	Frame buffer accepts 16-bit RGB-555 pixel data. The MSB of each pixel is ignored.
GR_LFBWRITEMODE_1555	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 **grColorMask**.
GR_LFBWRITEMODE_888	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.
GR_LFBWRITEMODE_8888	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 **grColorMask**. If dithering is enabled, then color will be dithered down to the real frame buffer storage format if necessary.
GR_LFBWRITEMODE_565_DEPTH	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.
GR_LFBWRITEMODE_555_DEPTH	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.
GR_LFBWRITEMODE_1555_DEPTH	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 **grColorMask**.
GR_LFBWRITEMODE_ZA16	Frame buffer accepts 16-bit auxiliary buffer values. This is the only *writeMode* that is valid when locking the auxiliary buffer. Alpha buffer values are taken from the 8 least significant bits of each sixteen bit word.
GR_LFBWRITEMODE_ANY	Lock will return the pixel format that most closely matches the true frame buffer storage format in the *info.writeMode*.

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:

GR_ORIGIN_UPPER_LEFT	Addressing originates in the upper left hand corner of the screen.
GR_ORIGIN_LOWER_LEFT	Addressing originates in the lower left hand corner of the screen.
GR_ORIGIN_ANY	Lock will always choose **GR_ORIGIN_UPPER_LEFT**

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. When enabled, color, alpha, and depth data from the linear frame buffer write will be processed as if it 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.

An application may not call any Glide routines other than **grLfbLock** and **grLfbUnlock** while any lock is active. Any such calls will result in undefined behavior.

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:

```
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.:

```
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

The Glide 2.2 release is the first release to include **grLfbReadRegion**. The following APIs are obsolete in Glide 2.2: **grLfbBegin**, **grLfbEnd**, **grLfbGetReadPtr**, **grLfbGetWritePtr**, **grLfbBypassMode**, **grLfbWriteMode**, **grLfbOrigin**, **guFbReadRegion**, and **guFbWriteRegion**.

SEE ALSO

grLfbUnlock, **grLfbConstantAlpha**, **grLfbConstantDepth**, **grLfbReadRegion**, **grLfbWriteRegion**

grLfbReadRegion

NAME

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

C SPECIFICATION

```
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

The Glide 2.2 release is the first release to include **grLfbReadRegion**. The following APIs are obsolete in Glide 2.2: **grLfbBegin**, **grLfbEnd**, **grLfbGetReadPtr**, **grLfbGetWritePtr**, **grLfbBypassMode**, **grLfbWriteMode**, **grLfbOrigin**, **guFbReadRegion**, and **guFbWriteRegion**.

SEE ALSO

grLfbLock, **grLfbUnlock**, **grLfbConstantAlpha**, **grLfbConstantDepth**, **grLfbWriteRegion**

grLfbUnlock

NAME

grLfbUnlock – unlock a frame buffer previously locked with **grLfbLock**.

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**. It is important to note that after a successful call to **grLfbUnlock**, accessing the *info.lfbPtr* used in the **grLfbLock** call will have undefined results.

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

NOTES

The Glide 2.2 release is the first release to include **grLfbUnlock**. The following APIs are obsolete in Glide 2.2: **grLfbBegin**, **grLfbEnd**, **grLfbGetReadPtr**, **grLfbGetWritePtr**, **grLfbBypassMode**, **grLfbWriteMode**, and **grLfbOrigin**.

SEE ALSO

grLfbLock, **grLfbConstantAlpha**, **grLfbConstantDepth**

grLfbWriteRegion

NAME

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

C SPECIFICATION

```
FxBool grLfbWriteRegion( GrBuffer_t dst_buffer,
                         FxU32 dst_x, FxU32 dst_y,
                         GrLfbSrcFmt_t src_format,
                         FxU32 src_width, FxU32 src_height,
                         FxU32 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.
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

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.

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

Not all 3Dfx graphics subsystems will support all source image formats. The function will fail if the source format supplied is not supported by the detected 3D hardware.

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 2.2 release is the first release to include **grLfbReadRegion**. The following APIs are obsolete in Glide 2.2: **grLfbBegin**, **grLfbEnd**, **grLfbGetReadPtr**, **grLfbGetWritePtr**, **grLfbBypassMode**, **grLfbWriteMode**, **grLfbOrigin**, **guFbReadRegion**, and **guFbWriteRegion**.

SEE ALSO

grLfbLock, **grLfbUnlock**, **grLfbConstantAlpha**, **grLfbConstantDepth**, **grLfbReadRegion**

grRenderBuffer

NAME

grRenderBuffer – selects the current color buffer for drawing and clearing

C SPECIFICATION

```
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**

grSstControlMode

NAME

grSstControlMode – perform SST-1 and SST-96 control functions

C SPECIFICATION

```
FxBool grSstControlMode( GrSstControlMode_t mode)
```

PARAMETERS

mode The control mode. Valid values are:

mode	Description
GR_CONTROL_ACTIVATE	activate 3D display
GR_CONTROL_DEACTIVATE	activate 2D display
GR_CONTROL_RESIZE	resize back buffers and auxiliary buffers *(SST-96 only)*
GR_CONTROL_MOVE	validate location after window move *(SST-96 only)*

DESCRIPTION

grSstControlMode determines whether the VGA display or Voodoo Graphics display is visible, depending on the value of *mode.*

The variable, *mode*, specifies one of four values. The first two values apply to all systems. When **GR_CONTROL_ACTIVATE** is specified, the Voodoo Graphics frame buffer will be displayed in full screen mode. On SST-96 systems, the video tile is enabled.

If *mode* is **GR_CONTROL_DEACTIVATE**, the 2D VGA frame buffer is displayed. On SST-96 systems, the video tile is disabled.

GR_CONTROL_RESIZE is ignored under DOS, SST-1, and SST-96 in full screen mode. For windowed Glide applications, this call resizes the back buffers and auxiliary buffers, and is typically made by Win32 applications in response to **WM_SIZE** messages. The **grSstControlMode** call may fail if there is not enough offscreen video memory to accommodate the resized buffers.

GR_CONTROL_MOVE is ignored under DOS, SST-1, and SST-96 in full screen mode. For windowed Glide applications, this call is used to validate the location and clip region associated with the front buffer when the user moves a window, and is typically made by Win32 applications in response to **WM_MOVE** messages. This call may fail if underlying DirectDraw implementation fails.

NOTES

On SST-1, since the 2D and 3D graphics exist on different devices (and frame buffers), activating or deactivating pass through does not require you repaint either the 2D or 3D graphics. On the SST-96, the application is responsible for repainting the 2D graphics or 3D graphics when you use **GR_CONTROL_ACTIVATE** or **GR_CONTROL_DEACTIVATE**.

This routine supersedes the now-obsolete **grSstPassthru** routine and provides the same functionality for SST-1 hardware; additional functionality is provided for windowed SST-96 applications.

SEE ALSO

grSstIdle

NAME

grSstIdle – returns when the Voodoo Graphics subsystem is idle

C SPECIFICATION

```
void grSstIdle( void )
```

PARAMETERS

none

DESCRIPTION

grSstIdle returns when the Voodoo Graphics subsystem is no longer busy. The system is busy when either the hardware FIFO is not empty or the graphics engine is busy.

NOTES

SEE ALSO

grSstStatus

grSstIsBusy

NAME

grSstIsBusy – indicates whether or not the Voodoo Graphics subsystem is busy

C SPECIFICATION

```
FxBool grSstIsBusy( void )
```

PARAMETERS

none

DESCRIPTION

grSstIsBusy returns **FXTRUE** if the Voodoo Graphics subsystem is busy; otherwise, it returns **FXFALSE**. The system is busy when either the hardware FIFO is not empty or the graphics engine is busy.

NOTES

SEE ALSO

grSstIdle, **grSstStatus**, **grSstVRetraceOn**

grSstOrigin

NAME

grSstOrigin – establishes a *y* origin

C SPECIFICATION

```
void grSstOrigin( 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

grSstOrigin sets the *y* origin for all triangle operations, fast fill, and clipping rectangles.

NOTES

grSstOrigin overrides the *y* origin specified in **grSstWinOpen**.

SEE ALSO

grSstWinOpen

grSstPerfStats

NAME

grSstPerfStats – get pixel rendering statistics

C SPECIFICATION

```
void grSstPerfStats( GrSstPerfStats_t *pStats )
```

PARAMETERS

pStats Pointer to a structure in which the performance statistics will be returned.

DESCRIPTION

The Voodoo Graphics hardware maintains a set of five counters that collect statistics about the fate of pixels as they move through the pixel pipeline. Glide provides access to these counters through the **GrSstPerfStats_s** structure and **grSstPerfStats**. The following information is returned in the structure pointed to by *pStats*:

***pStats* field**	**Description**
FxU32 *pixelsIn*	Number of pixels processed
FxU32 *chromaFail*	Number of pixels not drawn due to chroma-key failure
FxU32 *zFuncFail*	Number of pixels not drawn due to depth comparison failure
FxU32 *aFuncFail*	Number of pixels not drawn due to alpha comparison failure.
FxU32 *pixelsOut*	Number of pixels drawn. Note that this number includes all pixels drawn (i.e. pixels drawn by **grBufferClear** and LFB writes that have bypassed the pixel pipeline), not just those that have gone through the rendering pipeline.

All five counters are reset whenever **grSstResetPerfStats** is called. The hardware counters are only 24-bits wide, so regular calls to **grSstResetPerfStats** are required to avoid overflow. Alternatively, counter overflows can be detected and accounted for without calling **grSstResetPerfStats**.

NOTES

In order to account for every pixel counted and saved in *pixelsOut*, one must use the following equation:

pixelsOut = LFBwritePixels + bufferClearPixels + (*pixelsIn – zFuncFail – chromaFail – aFuncFail*)

bufferClearPixels represents the number of pixels written as a result of calls to **grBufferClear** and can be calculated as:

bufferClearPixels = (# of times the buffer was cleared)* (clip window width) * (clip window height)

grSstPerfStats does not wait for the system to be idle, and hence does not include statistics for commands that are still in the FIFO. Call **grSstIdle** to empty the FIFO.

SEE ALSO

grSstResetPerfStats

grSstQueryBoards

NAME

grSstQueryBoards – detect and determine the number of 3Dfx Voodoo Graphics subsystems installed in the host system

C SPECIFICATION

```
FxBool grSstQueryBoards( GrHwConfiguration *hwConfig )
```

PARAMETERS

hwConfig points to a **GrHwConfiguration** structure where the system's hardware configuration will be stored.

DESCRIPTION

grSstQueryBoards determines the number of installed Voodoo Graphics subsystems and stores this number in *hwConfig->num_sst*. No other information is stored in the structure at this time. **grSstQueryBoards** may be called before **grGlideInit**. **grSstQueryHardware** can be called after **grGlideInit** to fill in the rest of the structure.

NOTES

grSstQueryBoards does not change the state of any hardware, not does it render any graphics.

The Glide 2.1 release was the first release to include **grSstQueryBoards**.

SEE ALSO

grGlideInit, **grSstQueryHardware**

grSstQueryHardware

NAME

grSstQueryHardware – detect and determine the nature of any 3Dfx Voodoo Graphics subsystems installed in the host system

C SPECIFICATION

```
FxBool grSstQueryHardware( GrHwConfiguration *hwConfig )
```

PARAMETERS

hwConfig points to a **GrHwConfiguration** structure where the system's hardware configuration will be stored.

DESCRIPTION

grSstQueryHardware determines the system's Voodoo Graphics hardware configuration, specifically the number of installed Voodoo Graphics subsystems and each of their hardware configurations (memory, scan line interleaving, etc.). If no Voodoo Graphics hardware can be found, **grSstQueryHardware** returns **FXFALSE**; otherwise it returns **FXTRUE**.

grSstQueryHardware should be called after **grGlideInit** and before **grSstWinOpen**.

The **GrHwConfiguration** structure is defined as follows:

```
typedef struct
{
   int num_sst;
   struct {
      GrSstType type;          /* Which hardware is it? */
      union SstBoard_u {
         GrVoodooConfig_t VoodooConfig;
         GrSst96Config_t SST96Config;
         GrAT3DConfig_t AT3DConfig;
      } sstBoard;
   } SSTs[MAX_NUM_SST];  /* configuration for each board */
} GrHwConfiguration;
```

The structure contains mostly information on the configuration of a Voodoo Graphics subsystem. When two Voodoo Graphics subsystems are configured as a single scan-line interleaved system they are viewed by Glide and an application as a single Voodoo Graphics. Note that each Voodoo Graphics has its own private state and texture table.

NOTES

Refer to **glide.h** for possible values for hardware types. Current values are:

```
typedef int GrSstType;
#define GR_SSTTYPE_VOODOO   0
#define GR_SSTTYPE_SST96    1
#define GR_SSTTYPE_AT3D     2
```

SEE ALSO

grGlideInit, **grSstWinOpen**, **grSstQueryBoards**, **grSstSelect**

grSstResetPerfStats

NAME

grSstResetPerfStats – reset the pixel statistics counters

C SPECIFICATION

```
void grSstResetPerfStats( void )
```

PARAMETERS

none

DESCRIPTION

grSstResetPerfStats resets the pixel counters to **0x00**.

NOTES

SEE ALSO

grSstPerfStats

grSstScreenHeight

NAME

grSstScreenHeight – get the height (in pixels) of an SST screen

C SPECIFICATION

```
FxU32 grSstScreenHeight( void )
```

PARAMETERS

none

DESCRIPTION

grSstScreenHeight returns the height in pixels of the current SST board.

NOTES

SEE ALSO

grSstWinOpen, **grSstSelect**

grSstScreenWidth

NAME

grSstScreenWidth – get the width (in pixels) of an SST screen

C SPECIFICATION

```
FxU32 grSstScreenWidth( void )
```

PARAMETERS

none

DESCRIPTION

grSstScreenWidth returns the width in pixels of the current SST board.

NOTES

SEE ALSO

grSstWinOpen, **grSstSelect**

grSstSelect

NAME

grSstSelect – make a Voodoo Graphics subsystem current

C SPECIFICATION

```
void grSstSelect( int which_sst )
```

PARAMETERS

which_sst	The ordinal number of the Voodoo Graphics subsystem to make current. This value must be between 0 and the number of installed subsystems returned by **grSstQueryHardware**.

DESCRIPTION

grSstSelect selects a particular installed Voodoo Graphics subsystem as active. If the value passed is greater than the number of installed Voodoo Graphics subsystems and you are using the debug build of Glide, a run-time error will be generated. If you are using the release build of Glide, undefined behavior will result.

NOTES

SEE ALSO

grSstWinOpen, **grSstQueryHardware**

grSstStatus

NAME

grSstStatus – return the value of the graphics status register

C SPECIFICATION

```
FxU32 grSstStatus( void )
```

PARAMETERS

none

DESCRIPTION

grSstStatus returns the value of the Voodoo Graphics status register. The bits within this register are defined as follows:

Bit	Description
5:0	PCI FIFO free space (**0x3F**=FIFO empty)
6	Vertical retrace (0=vertical retrace active; 1=vertical retrace inactive).
7	Pixel*fx* graphics engine busy (0=engine idle; 1=engine busy)
8	TMU busy (0=engine idle; 1=engine busy)
9	Voodoo Graphics busy (0=idle; 1=busy)
11:10	Displayed buffer (0=buffer 0; 1=buffer 1; 2=auxiliary buffer; 3=reserved)
27:12	Memory FIFO free space (**0xFFFF**=FIFO empty)
30:28	Number of swap buffer commands pending
31	PCI interrupt generated (not implemented)

NOTES

SEE ALSO

grSstIdle, **grSstIsBusy**, **grSstSelect**, **grSstVRetraceOn**

grSstVideoLine

NAME

grSstVideoLine – returns the current line number of the display beam

C SPECIFICATION

```
FxU32 grSstVideoLine( void )
```

PARAMETERS

none

DESCRIPTION

grSstVideoLine returns the current line number of the display beam. This number is 0 during vertical retrace and increases as the display beam progresses down the screen.

NOTES

There are a small number of video lines that are not displayed at the top of the screen; the vertical backporch. Thus, **grSstVideoLine** returns a small positive number when the display beam is at the top of the screen; as the beam goes off the bottom of the screen, the line number may exceed the number returned by **grSstScreenHeight**.

The Glide 2.1 release was the first release to include **grSstVideoLine**.

SEE ALSO

grSstStatus, **grSstVRetraceOn**, **grSstScreenHeight**

grSstVRetraceOn

NAME

grSstVRetraceOn – return **FXTRUE** if vertical retrace is active

C SPECIFICATION

```
FxBool grSstVRetraceOn( void )
```

PARAMETERS

none

DESCRIPTION

grSstVRetraceOn returns **FXTRUE** if the monitor is in vertical retrace; otherwise **FXFALSE** is returned.

NOTES

SEE ALSO

grSstStatus, **grSstVideoLine**

grSstWinClose

NAME

grSstWinClose – close the graphics display device

C SPECIFICATION

```
void grSstWinClose( void )
```

PARAMETERS

none

DESCRIPTION

grSstWinClose returns the state of Glide to the one following **grGlideInit**, so that **grSstWinOpen** can be called with either a different resolution (SST-1 and SST-96), or a different *hwnd* parameter (SST-96 only).

NOTES

SEE ALSO

grSstWinOpen, **grSstControlMode**

grSstWinOpen

NAME

grSstWinOpen – opens the graphics display device

C SPECIFICATION

```
FxBool grSstWinOpen(  FxU32 hwnd,
                      GrScreenResolution_t res,
                      GrScreenRefresh_t ref,
                      GrColorFormat_t cformat,
                      GrOriginLocation_t org_loc,
                      int num_buffers,
                      int num_aux_buffers
                    )
```

PARAMETERS

hwnd Specifies a handle to the window. The interpretation of this value depends on the system environment. DOS applications must specify **NULL**. Applications run on SST-1 graphics hardware must specify **NULL** as well. Win32 full screen applications running on a SST-96 system must specify a window handle; a **NULL** value for *hwnd* 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 SST-1 and a **NULL** window handle is required. Finally, Glide Win32 applications that run in a window may either specify **NULL** (if there is only one window), or the correct *hwnd,* cast to **FxU32**.

System Environment	*hwnd* value
DOS	**NULL**
Win32 Full Screen	**NULL** or *hwnd*
Win32 Pure Console	**NULL** (SST-1 only)
Win32 Glide Applications	**NULL** or *hwnd* (SST-96 only)

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 for the SST-96. This signals Glide to use the user specified window (see the *hwnd* parameter). Specifying **GR_RESOLUTION_NONE** on an SST-1 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 (SST-96 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:

Color Format	Hex Variable Organization
GR_COLORFORMAT_RGBA	0xRRGGBBAA
GR_COLORFORMAT_ARGB	0xAARRGGBB
GR_COLORFORMAT_BGRA	0xBBGGRRAA
GR_COLORFORMAT_ABGR	0xAABBGGRR

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 both SST-1 and SST-96 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 SST-96, 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 both SST-1 and SST-96, 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, a value of **FXTRUE** is returned; otherwise a value of **FXFALSE** is returned. If **grSstWinOpen** is called on an already open window, **FXFALSE** will be returned. This routine replaces the obsolete **grSstOpen** call from previous versions of Glide.

NOTES

grSstWinOpen initializes various Glide state variables, including hints. Thus, **grHints** should be called after **grSstWinOpen** if you want something other than the default hint settings.

SEE ALSO

grSstControlMode, **grSstQueryHardware**, **grSstResetPerfStats**, **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	The smallest LOD in the mipmap.
largeLod	The largest LOD in the mipmap.
aspect	Aspect ratio of the mipmap.
format	Format of the mipmap.

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 for the mipmap.

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. If the levels are loaded individually, the starting address for each level must be 8-byte aligned.

SEE ALSO

grTexTextureMemRequired

grTexClampMode

NAME

grTexClampMode – set the texture map clamping/wrapping mode

C SPECIFICATION

```
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 will wrap around to 0. If *clamping* is enabled, then texture map indices will be clamped to 0 and 255.

NOTES

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

SEE ALSO

grTexSource

grTexCombine

NAME

grTexCombine – configure a texture combine unit

C SPECIFICATION

```
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:

Combine Function	Effect
GR_COMBINE_FUNCTION_ZERO	0
GR_COMBINE_FUNCTION_LOCAL	C_{local}
GR_COMBINE_FUNCTION_LOCAL_ALPHA	A_{local}
GR_COMBINE_FUNCTION_SCALE_OTHER GR_COMBINE_FUNCTION_BLEND_OTHER	$f * C_{other}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL	$f * C_{other} + C_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL_ALPHA	$f * C_{other} + A_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL	$f * (C_{other} - C_{local})$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND	$f * (C_{other} - C_{local}) + C_{local}$ $\equiv f * C_{other} + (1 - f) * C_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (C_{other} - C_{local}) + A_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND_LOCAL	$f * (- C_{local}) + C_{local}$ $\equiv (1 - f) * C_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (- C_{local}) + A_{local}$

rgb_factor Specifies the scaling factor *f* used in texture color generation. Valid parameters are described below:

Combine Factor	Scale Factor (*f*)
GR_COMBINE_FACTOR_NONE	Unspecified
GR_COMBINE_FACTOR_ZERO	0
GR_COMBINE_FACTOR_LOCAL	C_{local} / 255
GR_COMBINE_FACTOR_OTHER_ALPHA	A_{other} / 255
GR_COMBINE_FACTOR_LOCAL_ALPHA	A_{local} / 255
GR_COMBINE_FACTOR_DETAIL_FACTOR	β
GR_COMBINE_FACTOR_LOD_FRACTION	
GR_COMBINE_FACTOR_ONE	1
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL	$1 - C_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA	$1 - A_{other}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA	$1 - A_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_DETAIL_FACTOR	$1 - \beta$
GR_COMBINE_FACTOR_ONE_MINUS_LOD_FRACTION	

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

Combine Function	Effect
GR_COMBINE_FUNCTION_ZERO	0
GR_COMBINE_FUNCTION_LOCAL	A_{local}
GR_COMBINE_FUNCTION_LOCAL_ALPHA	A_{local}
GR_COMBINE_FUNCTION_SCALE_OTHER GR_COMBINE_FUNCTION_BLEND_OTHER	$f * A_{other}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL	$f * A_{other} + A_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_ADD_LOCAL_ALPHA	$f * A_{other} + A_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL	$f * (A_{other} - A_{local})$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND	$f * (A_{other} - A_{local}) + A_{local}$ $\equiv f * A_{other} + (1 - f) * A_{local}$
GR_COMBINE_FUNCTION_SCALE_OTHER_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (A_{other} - A_{local}) + A_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL GR_COMBINE_FUNCTION_BLEND_LOCAL	$f * (- A_{local}) + A_{local}$ $\equiv (1 - f) * A_{local}$
GR_COMBINE_FUNCTION_SCALE_MINUS_LOCAL_ADD_LOCAL_ALPHA	$f * (- C_{local}) + A_{local}$

alpha_factor Specifies the scaling factor *f* used in texture alpha generation. Valid parameters are described below:

Combine Factor	Scale Factor (*f*)
GR_COMBINE_FACTOR_NONE	Unspecified
GR_COMBINE_FACTOR_ZERO	0
GR_COMBINE_FACTOR_LOCAL	A_{local} / 255
GR_COMBINE_FACTOR_OTHER_ALPHA	A_{other} / 255
GR_COMBINE_FACTOR_LOCAL_ALPHA	A_{local} / 255
GR_COMBINE_FACTOR_DETAIL_FACTOR	β
GR_COMBINE_FACTOR_LOD_FRACTION	
GR_COMBINE_FACTOR_ONE	1
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL	$1 - A_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_OTHER_ALPHA	$1 - A_{other}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_LOCAL_ALPHA	$1 - A_{local}$ / 255
GR_COMBINE_FACTOR_ONE_MINUS_DETAIL_FACTOR	$1 - \beta$
GR_COMBINE_FACTOR_ONE_MINUS_LOD_FRACTION	

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 Voodoo 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 the *rgb_function* and *alpha_function* combining functions and the *rgb_factor* and *alpha_factor* scale factors on the local filtered texel (C_{local} and A_{local}) and the filtered texel from the upstream TMU (C_{other} and A_{other}). 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 Pixel*fx* chip.

In the *rgb_factor* and *alpha_factor* tables, β is the detail blend factor which is computed as a function of 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 – color) for floating point color values in the range [0..1] or (255 – color) for 8-bit color values in the range [0..255].

The TMU closest to the Pixel*fx* 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`**

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, β, 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 β is calculated as

$$\beta = \min(detailMax, \max(0, (lodBias - LOD) << detailScale) / 255.0)$$

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:

$$C_{out} = \beta * detail\_texture + (1 - \beta) * main\_texture$$

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**

grTexDownloadMipMap

NAME

grTexDownloadMipMap – download a complete mipmap to texture memory

C SPECIFICATION

```
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** and must be 8-byte aligned.

NOTES

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

```
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	Offset into texture memory where the texture will be loaded.
thisLod	Constant describing LOD to be downloaded.
largeLod	Constant describing largest LOD in the complete mipmap of which *thisLod* is a part.
aspectRatio	Constant describing aspect ratio of texture image.
format	Constant describing format of color data in texture image.
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

grTexDownloadMipMapLevel downloads a single mipmap level to an area of texture memory specified by *startAddress*.

startAddress must lie between the values returned by **grTexMinAddress** and **grTexMaxAddress** and must be 8-byte aligned.

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**

grTexDownloadMipMapLevelPartial

NAME

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

C SPECIFICATION

```
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 offset into texture memory for download.
thisLod	Constant describing LOD to be downloaded.
largeLod	Constant describing largest LOD in the complete mipmap of which *thisLod* is a part.
aspectRatio	Constant describing aspect ratio of texture image.
format	Constant describing format of color data in texture image.
evenOdd	Which mipmap levels to download. Valid values are **GR_MIPMAPLEVELMASK_EVEN**, **GR_MIPMAPLEVELMASK_ODD**, and **GR_MIPMAPLEVELMASK_BOTH**.
data	Raw texture image data.
start, end	Starting and ending rows of the mipmap to download.

DESCRIPTION

grTexDownloadMipMapLevelPartial downloads part of a single mipmap level to an area of texture memory specified by *startAddress*. Valid values for *startAddress* must be between the values returned by **grTexMinAddress** and **grTexMaxAddress** and must be 8-byte aligned. *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.

The Glide 2.1 release was the first release to include **grTexDownloadMipMapLevelPartial**.

SEE ALSO

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

grTexDownloadTable

NAME

grTexDownloadTable – download an NCC table or color palette

C SPECIFICATION

void grTexDownloadTable(GrChipID_t tmu, GrTexTable_t type, void *data)

PARAMETERS

tmu	Texture Mapping Unit to modify. Valid values are **GR_TMU0**, **GR_TMU1**, and **GR_TMU2**.
type	Type of texture table. The valid values are: **GR_TEX_NCC0** - Narrow-channel compression table 0 **GR_TEX_NCC1** - Narrow-channel compression table 1 **GR_TEX_PALETTE** - 256 entry color palette
data	Table data, either of type **GuNccTable** or **GuTexPalette**.

DESCRIPTION

grTexDownloadTable downloads either an NCC table or a 256-entry color palette to a TMU. There are two NCC tables and one color palette on each TMU. 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**. Which NCC table is used for decompression is specified by **grTexNCCTable**.

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.

SEE ALSO

grTexDownloadTablePartial, **grTexNCCTable**, **grTexSource**

grTexDownloadTablePartial

NAME

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

C SPECIFICATION

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

PARAMETERS

tmu	Texture Mapping Unit to modify. Valid values are **GR_TMU0**, **GR_TMU1**, and **GR_TMU2**.
type	Type of texture table. Valid values are: **GR_TEX_NCC0** - Narrow-channel compression table 0 **GR_TEX_NCC1** - Narrow-channel compression table 1 **GR_TEX_PALETTE** - 256 entry color palette
data	Table data, either of type **GuNccTable** or **GuTexPalette**.
start, end	Starting and ending entries to download.

DESCRIPTION

grTexDownloadTablePartial downloads part of an NCC table or a 256-entry color palette to a TMU. Entries from *start* up to and including *end* are downloaded. There are two NCC tables and one color palette on each TMU. 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

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

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

The Glide 2.1 release was the first release to include **grTexDownloadTablePartial**.

SEE ALSO

grTexDownloadTablePartial, **grTexNCCTable**, **grTexSource**

grTexFilterMode

NAME

grTexFilterMode – specify the texture minification and magnification filters

C SPECIFICATION

```
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

```
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

grTexSource

grTexMinAddress

NAME

grTexMinAddress – return the lowest start address for texture downloads

C SPECIFICATION

```
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**

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_1**.

SEE ALSO

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

grTexMipMapMode

NAME

grTexMipMapMode – set the mipmapping mode

C SPECIFICATION

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 when doing trilinear mipmapping. **FXFALSE** disables LOD blending.

DESCRIPTION

grTexMipMapMode sets the mipmapping mode for the Voodoo Graphics hardware. The Voodoo 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%. This is not always the case, as it is very application dependent. If this mode is used, performance should be benchmarked to determine the cost of the increased quality.

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

grTexLodBiasValue, **grTexSource**

grTexMultibase

NAME

grTexMultibase – enables or disables multibase addressing

C SPECIFICATION

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.

The Glide 2.1 release was the first release to include **grTexMultibase**.

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

```
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_256**, one for **GR_LOD_128**, one for **GR_LOD_64**, and one for **GR_LOD_32** through **GR_LOD_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

grTexSource does not restore the multiple base addresses for a multibased texture, but does set the base address for mipmap level **GR_LOD_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.

The Glide 2.1 release was the first release to include **grTexMultibaseAddress**.

SEE ALSO

grTexMultibase, **grTexSource**

grTexNCCTable

NAME

grTexNCCTable – select an NCC table

C SPECIFICATION

```
void grTexNCCTable( GrChipID_t tmu, GrNCCTable_t table )
```

PARAMETERS

tmu	Texture Mapping Unit to modify. Valid values are **GR_TMU0**, **GR_TMU1**, and **GR_TMU2**.
table	NCC table to use for decompressing compressed textures. Valid values are **GR_TEXTABLE_NCC0** and **GR_TEXTABLE_NCC1**.

DESCRIPTION

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

NOTES

SEE ALSO

grTexDownloadTable, **grTexSource**

grTexSource

NAME

grTexSource – specify the current texture source for rendering

C SPECIFICATION

```
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**.

NOTES

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**, **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. The number returned may be added to the start address for a texture download to determine the next free location in texture memory.

NOTES

SEE ALSO

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

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*fi *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*fi *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

guAADrawTriangleWithClip

NAME

guAADrawTriangleWithClip – performs 2D clipping on triangle, and draws the resultant polygon with anti-aliasing

C SPECIFICATION

```
void guAADrawTriangleWithClip( const GrVertex *va, const GrVertex *vb, const GrVertex
                               *vc )
```

PARAMETERS

va, vb, vc	Vertices which specify the triangle.

DESCRIPTION

This routine performs 2D clipping on a triangle, and draws the resultant polygon with **grAADrawPolygonVertexList**.

NOTES

All edges of the triangle are anti-aliased.

SEE ALSO

grAADrawTriangle, **grAADrawPolygonVertexList**, **guDrawTriangleWithClip**

guAlphaSource

NAME

guAlphaSource – configure the alpha combine unit

C SPECIFICATION

```
void guAlphaSource( GrAlphaSourceMode_t mode )
```

PARAMETERS

mode The new alpha combine unit mode.

DESCRIPTION

guAlphaSource is a higher level interface to the Voodoo Graphics alpha combine unit than **grAlphaCombine**. The alpha combine unit has two configurable inputs and one output. The output of the alpha combine unit gets fed into the alpha testing and blending units. The selection of the A_{local} input is important because it is used in the color combine unit.

The following table describes how A_{local} and output alpha are computed based on the mode:

mode	Alpha Output	A_{local}
GR_ALPHASOURCE_CC_ALPHA	constant color alpha	constant color alpha
GR_ALPHASOURCE_ITERATED_ALPHA	iterated vertex alpha	iterated vertex alpha
GR_ALPHASOURCE_TEXTURE_ALPHA	texture alpha	none
GR_ALPHASOURCE_TEXTURE_ALPHA_TIMES_ITERATED_ALPHA	texture alpha * iterated alpha	iterated vertex alpha

NOTES

Constant color alpha is the value passed to **grConstantColorValue**.

If texture has no alpha component, texture alpha is 255.

guAlphaSource is a compatibility layer for **grAlphaCombine**.

SEE ALSO

grConstantColorValue, **grAlphaCombine**, **grColorCombine**

guColorCombineFunction

NAME

guColorCombineFunction – configure the color combine unit

C SPECIFICATION

```
void guColorCombineFunction( GrColorCombineFunction_t func )
```

PARAMETERS

func Specifies the source color generation function. Valid parameters are described below:

Color Combine Function	Effect
GR_COLORCOMBINE_ZERO	**0x00** per component
GR_COLORCOMBINE_ITRGB	Gouraud shading
GR_COLORCOMBINE_DECAL_TEXTURE	texture
GR_COLORCOMBINE_TEXTURE_TIMES_CCRGB	flat-shaded texture using constant color (**grConstantColorValue**) as the shading value
GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB	Gouraud-shaded texture
GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB_ADD_ALPHA	Gouraud-shaded texture + alpha
GR_COLORCOMBINE_TEXTURE_TIMES_ALPHA	texture * alpha
GR_COLORCOMBINE_TEXTURE_ADD_ITRGB	texture + iterated RGB
GR_COLORCOMBINE_TEXTURE_SUB_ITRGB	texture – iterated RGB
GR_COLORCOMBINE_CCRGB	flat shading using constant color (**grConstantColorValue**)
GR_COLORCOMBINE_CCRGB_BLEND_ITRGB_ON_TEXALPHA	blend between constant color and iterated RGB using an alpha texture, where alpha of 0 and 1 correspond to constant color and iterated RGB respectively
GR_COLORCOMBINE_DIFF_SPEC_A	texture * alpha + iterated RGB
GR_COLORCOMBINE_DIFF_SPEC_B	texture * iterated RGB + alpha
GR_COLORCOMBINE_ONE	**0xFF** per component

DESCRIPTION

guColorCombineFunction configures the Voodoo Graphics subsystem's hardware pipeline in a fashion dictated by the parameter *func*. This provides a high level mechanism for controlling common rendering modes without manipulating individual registers within the hardware. The default color combine function is undefined, so an application must set the color combine function before executing any rendering commands.

GR_COLORCOMBINE_ZERO Forces the output of the color combine unit to always be black (**0x00000000**).

GR_COLORCOMBINE_ITRGB Uses the iterated RGB values in the **GrVertex** structure to render a primitive. The effect of this is a smoothly iterated color across the entire face of a triangle.

GR_COLORCOMBINE_DECAL_TEXTURE
Uses the *sow* and *tow* values to texture map onto the primitive. The texture color used at each pixel is determined by the value of **guTexSource** and the current **guTexCombineFunction**.

GR_COLORCOMBINE_TEXTURE_TIMES_CCRGB
Like **GR_COLORCOMBINE_DECAL_TEXTURE**, except each texel is multiplied by the constant color value set with **grConstantColorValue**. This allows for lit flat-shaded texture maps.

GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB
Like **GR_COLORCOMBINE_TEXTURE_TIMES_CCRGB**, except each texel is multiplied by the current iterated RGB value. This allows for lit Gouraud-shaded texture maps.

GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB_ADD_ALPHA
Like **GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB**, except that the alpha value is added to the product of the texel and the iterated RGB color. This allows for specular highlights on Gouraud-shaded texture maps.

GR_COLORCOMBINE_TEXTURE_TIMES_ALPHA
Multiplies each texel by alpha (the *local* alpha within the alpha combine unit).

GR_COLORCOMBINE_TEXTURE_ADD_ITRGB
Like **GR_COLORCOMBINE_DECAL_TEXTURE**, except that the current iterated RGB value is added to each texel before it is passed down the pixel pipeline.

GR_COLORCOMBINE_TEXTURE_SUB_ITRGB
Like **GR_COLORCOMBINE_DECAL_TEXTURE**, except that the current iterated RGB value is subtracted from each texel before it is passed down the pixel pipeline.

GR_COLORCOMBINE_CCRGB Uses the color specified by **grConstantColorValue** to render a primitive. This is useful for flat-shaded primitives. This color combine function requires that a color value be packed into a 32-bit color value (in the format specified by the *cformat* argument to **grSstWinOpen**).

GR_COLORCOMBINE_CCRGB_BLEND_ITRGB_ON_TEXALPHA
Uses a blend between **grConstantColorValue** and the current iterated RGB value as the output color. The amount blended between the two colors is determined by the alpha component for the corresponding pixel in the current texture map. If the current texture map does not have an alpha component, then the alpha value used will be **0xFF**.

GR_COLORCOMBINE_DIFF_SPEC_A
Like **GR_COLORCOMBINE_TEXTURE_TIMES_ALPHA**, except the iterated RGB is also added to the final color. This allows for colored specular highlights to be added to a white lit texture mapped polygon.

GR_COLORCOMBINE_DIFF_SPEC_A
Like **GR_COLORCOMBINE_TEXTURE_TIMES_ITRGB**, except that alpha is also added to the final color. This allows for white specular highlights to be added to a colored lit texture mapped polygon.

GR_COLORCOMBINE_ONE Forces the color combine unit to always output white (**0xFFFFFFFF**).

NOTES

SEE ALSO

grConstantColorValue, **grColorCombine**, **grTexCombine**

guDrawTriangleWithClip

NAME

guDrawTriangleWithClip – draw a triangle with 2D clipping.

C SPECIFICATION

```
void guDrawTriangleWithClip(const GrVertex *va, const GrVertex *vb, const GrVertex *vc
                            )
```

PARAMETERS

va, vb, vc	The three vertices of the triangle.

DESCRIPTION

guDrawTriangleWithClip uses Sutherland-Hodgman clipping [SUTH74] to clip the triangle to the rectangle specified by **grClipWindow** and then draws the resultant polygon.

NOTES

SEE ALSO

grDrawTriangle

guFogGenerateExp

NAME

guFogGenerateExp – generate an exponential fog table

C SPECIFICATION

```
void guFogGenerateExp( GrFog_t fogTable[GR_FOG_TABLE_SIZE], float density )
```

PARAMETERS

fogTable	The array to 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:

$$e^{-density*w}$$

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

```
void guFogGenerateExp2( GrFog_t fogTable[GR_FOG_TABLE_SIZE], float density )
```

PARAMETERS

fogTable	The array to 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:

$$e^{-(density*w)2}$$

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[GR_FOG_TABLE_SIZE],
                          float nearW, float farW )
```

PARAMETERS

fogTable	The array to receive the generated fog table values.
nearW	The eye-space *w* coordinate where minimum fog exists.
farW	The eye-space *w* coordinate where maximum fog exists.

DESCRIPTION

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

$$(w - nearW)/(farW \quad nearW)$$

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 eye-space *w* value associated with entry *i* in a fog table. Because fog table entries are non-linear in *w,* 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+(double)(i>>2)) / (8-(i&3));
```

SEE ALSO

grFogMode, **grFogTable**, **guFogGenerateExp**, **guFogGenerateExp2**, **guFogGenerateLinear**

guTexAllocateMemory

NAME

guTexAllocateMemory – allocate texture memory for a mipmap

C SPECIFICATION

```
GrMipMapId_t guTexAllocateMemory(  GrChipID_t tmu,
                                   FxU8 evenOddMask,
                                   int width, int height,
                                   GrTextureFormat_t format,
                                   GrMipMapMode_t mmMode,
                                   GrLOD_t smallLod, GrLOD_t largeLod,
                                   GrAspectRatio_t aspectRatio,
                                   GrTextureClampMode_t sClampMode,
                                   GrTextureClampMode_t tClampMode,
                                   GrTextureFilterMode_t minFilterMode,
                                   GrTextureFilterMode_t magFilterMode,
                                   float lodBias,
                                   FxBool lodBlend )
```

PARAMETERS

tmu	Texture Mapping Unit to allocate memory on. Valid values are **GR_TMU0**, **GR_TMU1**, and **GR_TMU2**.
evenOddMask	Selects whether odd levels, even levels, or all levels of the mipmap are downloaded. Valid values are **GR_MIPMAPLEVELMASK_EVEN**, **GR_MIPMAPLEVELMASK_ODD**, and **GR_MIPMAPLEVELMASK_BOTH**.
width, height	Width and height of the largest mipmap level.
format	Format of the texture. Valid values are **GR_TEXFMT_RGB_332**, **GR_TEXFMT_YIQ_422**, **GR_TEXFMT_ALPHA_8**, **GR_TEXFMT_INTENSITY_8**, **GR_TEXFMT_ALPHA_INTENSITY_44**, **GR_TEXFMT_RGB_565**, **GR_TEXFMT_ARGB_8332**, **GR_TEXFMT_ARGB_1555**, **GR_TEXFMT_ARGB_4444**, **GR_TEXFMT_AYIQ_8422**, and **GR_TEXFMT_ALPHA_INTENSITY_88**.
mmMode	Type of mipmapping to be performed when this texture is current. Valid values are **GR_MIPMAP_DISABLE**, **GR_MIPMAP_NEAREST**, and **GR_MIPMAP_NEAREST_DITHER**. This value can be overridden by a call to **grTexMipMapMode**.
smallLod, largeLod	LOD values of the smallest and largest LOD levels in the texture. Valid parameters are **GR_LOD_256**, **GR_LOD_128**, **GR_LOD_64**, **GR_LOD_32**, **GR_LOD_16**, **GR_LOD_8**, **GR_LOD_4**, **GR_LOD_2**, and **GR_LOD_1**. The value in the LOD constant determines the size of the largest side of the texture. A combination of the *largeLod* parameter and the *aspectRatio* parameter determines the size of each of the mipmap levels. For example, a mipmap with *largeLod* of **GR_LOD_64** and *aspectRatio* of **GR_ASPECT_8x1** would have a 64×8 texture as its largest LOD. If *aspectRatio* were **GR_ASPECT_1x4**, an 16×64 texture would be its largest LOD.

aspectRatio Specifies the aspect ratio of the mipmaps as a factor of width to height. For example, a mipmap 256 texels wide by 128 texels tall has an aspect ratio of **GR_ASPECT_2x1**. Valid values are **GR_ASPECT_8x1**, **GR_ASPECT_4x1**, **GR_ASPECT_2x1**, **GR_ASPECT_1x1**, **GR_ASPECT_1x2**, **GR_ASPECT_1x4**, and **GR_ASPECT_1x8**. A combination of this parameter and the LOD parameter *smallLod* determines the size of each mipmap level.

sClampMode, tClampMode
Type of texture clamping to be performed when this texture is current. Clamping can be controlled in both the *s* and *t* directions. Valid parameters are **GR_TEXTURECLAMP_CLAMP** or **GR_TEXTURECLAMP_WRAP**. Note that these parameters should always be set to **GR_TEXTURECLAMP_CLAMP** for projected textures.

minFilterMode Specifies the type of minification filtering to perform. Valid parameters are **GR_TEXTUREFILTER_POINT_SAMPLED** and **GR_TEXTUREFILTER_BILINEAR**

magFilterMode Specifies the type of magnification filtering to perform. Valid parameters are **GR_TEXTUREFILTER_POINT_SAMPLED** and **GR_TEXTUREFILTER_BILINEAR**.

lodBias Specifies the LOD bias value in the range [-8..7.75] to be used during mipmapping. Smaller values make increasingly sharper images, larger values make increasingly blurrier images. The *lodBias* parameter is rounded to the nearest quarter increment. By default the LOD bias is 0.

lodBlend Specifies whether trilinear filtering is to be performed when this texture is current. Trilinear filtering blends between LOD levels based on LOD fraction, eliminating mipmap banding artifacts. Valid parameters are **FXTRUE** and **FXFALSE**.

DESCRIPTION

guTexAllocateMemory allocates memory on the specified TMUs and returns a handle to the allocated memory. The amount of memory allocated will be enough to hold a mipmap of the given format, LOD ranges, and aspect ratio. If the texture memory can not be allocated, a value of **GR_NULL_MIPMAP_HANDLE** is returned. After the memory has been allocated, individual mipmap levels can be downloaded one at a time using **guTexDownloadMipMapLevel**, or all mipmaps can be downloaded using **guTexDownloadMipMap**.

Whenever the texture is specified as a source texture, *sClampMode*, *tClampMode*, *minFilterMode*, *magFilterMode*, and *lodBias* will automatically take effect.

evenOddMask is used to selectively download LOD levels when LOD blending is to be used. Correct usage is to allocate and download the even levels onto one TMU, and the odd levels onto another, both with the *lodBlend* parameter set to **FXTRUE**. Then the texture combine mode for the lower numbered TMU is set to **GR_TEXTURECOMBINE_TRILINEAR_ODD** or **GR_TEXTURECOMBINE_TRILINEAR_EVEN** depending on whether the odd levels or the even levels were downloaded to it.

NOTES

SEE ALSO

guTexGetCurrentMipMap, **guTexGetMipMapInfo**, **guTexSource**

guTexChangeAttributes

NAME

guTexChangeAttributes – change attributes of a mipmap

C SPECIFICATION

```
FxBool guTexChangeAttributes( GrMipMapID_t mmid,
                              int width, int height,
                              GrTextureFormat_t format,
                              GrMipMapMode_t mmMode,
                              GrLOD_t smallLod, GrLOD_t largeLod,
                              GrAspectRatio_t aspectRatio,
                              GrTextureClampMode_t sClampMode,
                              GrTextureClampMode_t tClampMode,
                              GrTextureFilterMode_t minFilterMode,
                              GrTextureFilterMode_t magFilterMode )
```

PARAMETERS

mmid	Handle of mipmap whose attributes are being changed.
width, height	New width and height of the largest mipmap level or –1 if not changed
format	New format of the texture or –1 if not changed.
mmMode	New type of mipmapping to be performed when this texture is in effect. Valid values are **GR_MIPMAP_DISABLE**, **GR_MIPMAP_NEAREST**, and **GR_MIPMAP_NEAREST_DITHER**. This value can be overridden by a call to **grTexMipMapMode**.
smallLod, largeLod	New LOD values of the smallest and largest LOD levels or –1 if not changed. Valid parameters are **GR_LOD_256**, **GR_LOD_128**, **GR_LOD_64**, **GR_LOD_32**, **GR_LOD_16**, **GR_LOD_8**, **GR_LOD_4**, **GR_LOD_2**, and **GR_LOD_1**. The value of *largeLod* determines the size of the largest side of the texture. A combination of the *largeLod* parameter and the *aspectRatio* parameter determines the size of each of the mipmap levels. For example, a mipmap with *largeLod* of **GR_LOD_64** and *aspectRatio* of **GR_ASPECT_8x1** would have a 64×8 texture as its largest LOD. If *aspectRatio* were **GR_ASPECT_1x4**, an 16×64 texture would be its largest LOD.
aspectRatio	New aspect ratio of the mipmap as a factor of width to height or –1 if not changed. For example, a mipmap 256 texels wide by 128 texels tall has an aspect ratio of **GR_ASPECT_2x1**. Valid values are **GR_ASPECT_8x1**, **GR_ASPECT_4x1**, **GR_ASPECT_2x1**, **GR_ASPECT_1x1**, **GR_ASPECT_1x2**, **GR_ASPECT_1x4**, and **GR_ASPECT_1x8**. A combination of this parameter and the LOD parameter *smallLod* determines the size of each mipmap level.
sClampMode	New type of texture clamping to be performed in the *s* direction or –1 if not changed. Valid parameters are **GR_TEXTURECLAMP_CLAMP** or **GR_TEXTURECLAMP_WRAP**.
tClampMode	New type of texture clamping to be performed in the *t* direction or –1 if not changed. Valid parameters are **GR_TEXTURECLAMP_CLAMP** or **GR_TEXTURECLAMP_WRAP**.

minFilterMode	New type of minification filtering to perform or –1 if not changed. Valid parameters are **GR_FILTER_POINT_SAMPLED** and **GR_FILTER_BILINEAR**.
magFilterMode	New type of magnification filtering to perform or –1 if not changed. Valid parameters are **GR_FILTER_POINT_SAMPLED** and **GR_FILTER_BILINEAR**.

DESCRIPTION

guTexChangeAttributes changes some of the attributes of a mipmap. This allows a section of texture memory to be reused without resetting all of texture memory. Upon success, **FXTRUE** is returned, else **FXFALSE** is returned.

NOTES

For projected textures the clamp modes, *sClampMode* and *tClampMode*, should always be set to **GR_TEXTURECLAMP_CLAMP**.

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, or **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory

guTexCombineFunction

NAME

guTexCombineFunction – configure the texture combine unit on a Texture Mapping Unit

C SPECIFICATION

```
void guTexCombineFunction( GrChipID_t tmu, GrTextureCombineFnc_t func )
```

PARAMETERS

tmu	Texture Mapping Unit to configure. Valid values are **GR_TMU0**, **GR_TMU1**, and **GR_TMU2**.
func	The new texture combine mode.

DESCRIPTION

guTexCombineFunction specifies the function used when combining textures on a TMU with incoming textures from the neighboring TMU. Texture combining operations allow for interesting effects such as detail and projected texturing as well as the trilinear filtering of LOD blending.

The following table describes the available texture combine functions and their effects. C_{local} represents the color components generated by indexing and filtering from the mipmap stored on the selected TMU and C_{other} represents the incoming color components from the neighboring TMU.

Texture Combine Function	Result	Effect
GR_TEXTURECOMBINE_ZERO	0	**0x00** per component
GR_TEXTURECOMBINE_DECAL	C_{local}	decal texture
GR_TEXTURECOMBINE_OTHER	C_{other}	pass through
GR_TEXTURECOMBINE_ADD	$C_{other} + C_{local}$	additive texture
GR_TEXTURECOMBINE_MULTIPLY	$C_{other} * C_{local}$	modulated texture
GR_TEXTURECOMBINE_SUBTRACT	$C_{other} - C_{local}$	subtractive texture
GR_TEXTURECOMBINE_DETAIL	blend (C_{other}, C_{local})	detail textures with detail on selected TMU
GR_TEXTURECOMBINE_DETAIL_OTHER	blend (C_{other}, C_{local})	detail textures with detail on neighboring TMU
GR_TEXTURECOMBINE_TRILINEAR_ODD	blend (C_{other}, C_{local})	LOD blended textures with odd levels on selected TMU
GR_TEXTURECOMBINE_TRILINEAR_EVEN	blend (C_{other}, C_{local})	LOD blended textures with even levels on selected TMU
GR_TEXTURECOMBINE_ONE	255	**0xFF** per component

guTexCombineFunction also keeps track of which TMUs require texture coordinates for the rendering routines.

NOTES

Many combine functions that simultaneously use both C_{local} and C_{other} can be computed with two passes on a single TMU system by using the frame buffer to store intermediate results and the alpha blender to combine the two partial results.

SEE ALSO

`grAlphaCombine`, **`grColorCombine`**, **`grDrawTriangle`**, **`grTexCombine`**, **`guTexSource`**

guTexDownloadMipMap

NAME

guTexDownloadMipMap – download a mipmap to texture memory

C SPECIFICATION

```
void guTexDownloadMipMap( GrMipMapId_t mmid, const void *src,
                          const GuNccTable *nccTable )
```

PARAMETERS

mmid	Handle of the mipmap memory accepting the download.
src	Pointer to row-major array of pixels of the format, size, and aspect ratio associated with *mmid*. Mipmap levels are arranged from largest to smallest size.
nccTable	Pointer to a Narrow Channel Compression table. This is only valid for 8-bit compressed textures loaded with **gu3dfLoad**.

DESCRIPTION

guTexDownloadMipMap downloads an entire mipmap to an area of texture memory previously allocated with **guTexAllocateMemory**. The data to be downloaded must have the pixel format and aspect ratio associated with *mmid.*

NOTES

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, and **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory, **guTexDownloadMipMapLevel**, **guTexMemReset**, **guTexSource**

guTexDownloadMipMapLevel

NAME

guTexDownloadMipMapLevel – download one level of a mipmap

C SPECIFICATION

void guTexDownloadMipMapLevel(GrMipMapId_t mmid, GrLOD_t lod, const void **src)

PARAMETERS

mmid	Handle of the mipmap memory accepting the download.
lod	LOD level of the mipmap level to download. Valid parameters are **GR_LOD_256**, **GR_LOD_128**, **GR_LOD_64**, **GR_LOD_32**, **GR_LOD_16**, **GR_LOD_8**, **GR_LOD_4**, **GR_LOD_2**, and **GR_LOD_1**. *lod* must lie between *mmid*'s *smallLod* and *largeLod* LOD levels specified during **guTexAllocateMemory**.
src	Pointer to a pointer to row-major array of pixels of the format, size, and aspect ratio associated with *mmid* and *lod*.

DESCRIPTION

guTexDownloadMipMapLevel downloads a single mipmap level within a mipmap to an area of texture memory previously allocated with **guTexAllocateMemory** and updates **src* to point to the next mipmap level. The data to be downloaded must have the pixel format and aspect ratio associated with *mmid* and must be of the correct size for *lod*.

NOTES

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, and **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory, **guTexDownloadMipMapLevel**, **guTexMemReset**, **guTexSource**

guTexGetCurrentMipMap

NAME

guTexGetCurrentMipMap – return the handle of the current mipmap

C SPECIFICATION

```
GrMipMapId_t *guTexGetCurrentMipMap ( GrChipID_t tmu )
```

PARAMETERS

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

DESCRIPTION

guTexGetCurrentMipMap returns the handle of the currently active mipmap on a selected TMU. Each TMU has one currently active mipmap. Mipmaps are made current with **guTexSource**.

NOTES

SEE ALSO

guTexAllocateMemory, **guTexSource**, **guTexGetMipMapInfo**

guTexGetMipMapInfo

NAME

guTexGetMipMapInfo – return a pointer to a structure containing information about a specific mipmap.

C SPECIFICATION

```
GrMipMapInfo *guTexGetMipMapInfo( GrMipMapId_t mmid )
```

PARAMETERS

mmid	Handle of the mipmap inquired about.

DESCRIPTION

guTexGetMipMapInfo allows an application to retrieve information about a mipmap.

NOTES

SEE ALSO

guTexAllocateMemory, **guTexSource**, **guTexGetCurrentMipMap**

guTexMemQueryAvail

NAME

guTexMemQueryAvail – return the amount of unallocated texture memory on a Texture Mapping Unit

C SPECIFICATION

```
FxU32 guTexMemQueryAvail( GrChipID_t tmu )
```

PARAMETERS

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

DESCRIPTION

guTexMemQueryAvail returns the amount of unallocated texture memory on a TMU. Only memory allocated with **guTexAllocateMemory** is taken into account.

NOTES

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, and **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory, **guTexMemReset**

guTexMemReset

NAME

guTexMemReset – free all allocated texture memory for all Texture Mapping Units

C SPECIFICATION

```
void guTexMemReset( void )
```

PARAMETERS

none

DESCRIPTION

guTexMemReset frees up all allocated texture memory. This allows for a simple form of texture memory management; all texture memory is allocated at once and then freed en masse. While simple, this form of memory management prevents some of the complexity associated with standard memory management techniques, such as garbage collection and memory fragmentation and compaction.

NOTES

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, and **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory, **guTexMemQueryAvail**

guTexSource

NAME

guTexSource – make a mipmap current on its Texture Mapping Unit

C SPECIFICATION

```
void guTexSource( GrMipMapId_t mmid )
```

PARAMETERS

mmid	Handle of the mipmap to make current.

DESCRIPTION

guTexSource makes current a mipmap for the TMU it resides on. Each TMU has one current mipmap. In systems with multiple TMUs, multiple mipmap sources are combined by the texture combine function and the output of the final combine is passed on to the pixel shading pipeline. By default all the TMUs have **NULL** texture handles associated with them.

When a mipmap is made current, all of its attributes take effect. See **guTexAllocateMemory** and **guTexChangeAttributes** for a complete listing of texture attributes. Some of these attributes can be temporarily overridden with **grTexClampMode**, **grTexFilterMode**, **grTexLodBiasValue**, and **grTexMipMapMode**. Note, however, that these routines do not change the mipmap's attribute, only the current mode of the rendering hardware.

NOTES

WARNING: do not use in conjunction with **grTexMinAddress**, **grTexMaxAddress**, **grTexNCCTable**, **grTexSource**, **grTexDownloadTable**, **grTexDownloadMipMapLevel**, **grTexDownloadMipMap**, **grTexMultiBase**, and **grTexMultibaseAddress**.

SEE ALSO

guTexAllocateMemory, **guTexChangeAttributes**, **grTexClampMode**, **grTexFilterMode**, **guTexGetCurrentMipMap**, **grTexLodBiasValue**, **grTexMipMapMode**

References

FOLE90 Foley, J., A. van Dam, S. Feiner, and J. Hughes, “Computer Graphics”, Addison-Wesley, Reading, 1990

SUTH74 Sutherland, I. E. and G. W. Hodgman, “Reentrant Polygon Clipping”, **CACM** 17(1), 32-42

WILL83 Williams, L., “Pyramidal Parametrics”, *SIGGRAPH 83*, 1-11