Introduction to OpenGL on OS/2

OpenGL is a highly precise, 3D rendering interface that is used in a wide variety of applications, including CAD, entertainment, industrial design and modeling, biochemistry, and scientific visualization. Functions of this powerful interface include real-time 3D rendering of points, lines, and polygons, support for lighting, texture mapping, anti-aliasing, fogging, motion blur, hidden surface removal, transparency, and double-buffering of images.

The OpenGL Architectural Review Board (ARB), of which IBM is a founding member, defines the rendering functions for the OpenGL specification. To implement support for OpenGL, a platform must provide a windowing and input interface for the rendering functions. The implementation must pass conformance tests defined by the ARB before the platform can legally use the OpenGL trademark. This requirement ensures the portability of OpenGL from one platform to another. The current OS/2 implementation of OpenGL conforms to the OpenGL 1.0 specification, and passes OpenGL 1.0 conformance tests.

OpenGL is intended to be an interface to graphics hardware. This means that application writers do not have to code for a particular piece of graphics hardware because they are ensured of consistent behavior across all graphics hardware that passes the OpenGL conformance tests.

OpenGL does not need special 3D hardware to run on OS/2. Every OS/2 platform contains a complete OpenGL pipeline and rasterizer implemented in software. This implementation enables developers to provide entry-level 3D graphics functionality to a broad base of users. OpenGL performance on OS/2 scales with processor power. Higher floating-point power yields better OpenGL performance.

This chapter includes the following sections:

OpenGL Rendering Interface

OpenGL provides functionality for the following.

- Drawing of a wide assortment of graphical primitives:
  - Points, lines, segments, loops, triangles, quadrangles, triangle strips, and more
  - NURBS and complex polygons, which are supported by OpenGL Utility (GLU) Library
- Flexible data input
  - Object data can be specified in 2, 3, or 4 dimensions and numerous data formats
  - Allows developer to use immediate or display list mode
- Complex light models
  - Up to 8 lights
  - Spot lights
  - Local or infinite lights
- Complex material models
  - One or two sided
  - Ambient, diffuse and specular components can be set
- Advanced texture mapping support
  - Automatic generation of texture coordinates
  - Two magnification filters
  - Six minification filters including mipmapping
  - Decal, Modulation, or Blending of texture
- Line and Polygon antialiasing
- Fogging effects
- Accumulation buffer can facilitate motion blur and full scene antialiasing effects
- Depth buffer and comparisons available for hidden surface removal
- Alpha buffer and blending operations facilitate transparency and compositing effects
- Stencil buffer can be used for Constructive Solid Geometry (CSG) modeling and shadows
- Dithering allows true color images to be drawn with a small palette
- Double-buffering for smooth animation

OS/2 Presentation Manager GL (PGL) Interface

A small set of functions integrates OpenGL with whatever windowing system is currently running. Because these functions are windowing-system specific, they cannot be the same across all platforms. In OS/2 these functions comprise the PGL (Presentation Manager GL) specification. Application programmers use the PGL interface to:

- Create and prepare an OpenGL context for rendering
- Swap a window's front and back buffers (front buffer is displayed)
- Select a color palette for a color index context
- Integrate GPI and OpenGL rendering and fonts

PGL provides much of the same functionality as glX in X Windows systems, and the WGL interface in Microsoft systems. This implementation of OpenGL works on OS/2 Warp.

OpenGL contexts are created using pglCreateContext. A parameter to this function is a VISUALCONFIG, a pointer to a visual configuration data structure that describes the type of ancillary buffers required for the rendering context. A list of available VISUALCONFIGs can be queried with pglQueryConfigs. When choosing a VISUALCONFIG, select the one that contains only the buffers that you need. For instance, choose a VISUALCONFIG with accumulation buffers only if you plan to use them. Accumulation buffers are 64 bits deep, and allocating memory for a 400 x 400 window would consume 1,280,000 bytes.

When an OpenGL context is created, it can be specified as either a direct context or an indirect context:

- A direct context bypasses the Presentation Manager when displaying OpenGL images and is generally faster than an indirect context.

- An indirect context uses a standard Presentation Manager bitmap to display images and allows the integration of OpenGL and GPI rendering function.

After a context is created, it must be bound to an OS/2 window by calling pglMakeCurrent When a context is current, that means it can finally be used for rendering.

OpenGL runs as a set of DLLs on top of OS/2:

- OPENGL.DLL contains all rendering entry points, glu entry points, PGL entry points, and the OpenGL pipeline code. The pipeline code is responsible for object transformation, lighting, culling and clipping. Applications only need to link with OPENGL.DLL to use the OpenGL API.

- RASTER.DLL contains a rasterizer, which takes vertices and creates fragments. These fragments then go through a series of possible steps including texture mapping, fogging, depth tests (Z-buffering), blending and many more. Some or all of these steps can be performed by 3D hardware, if available. (Using pglSwapBuffers will flush and display the rendered image.)

OpenGL Windowing and Input Toolkits

OpenGL on OS/2 is supplied with two toolkits (AUX and GLUT) that provide simple windowing and input handling. These toolkits are not intended for application development, but rather for uninhibited experimentation with 3D functionality. The GLUT toolkit has more features than AUX, including popup menus.

Toolkits are provided "as-is" on most OpenGL platforms, including OS/2. They are also used in sample code provided in the *OpenGL Programming Guide* published by Addison-Wesley (see Sources for OpenGL Information).

Demo Makefiles

Makefiles are included only for the IBM C Set and Visual Age compilers, not for all the compilers supported by OS/2. Compiler and linker flags are explained in the C Set makefile. Link statements are also provided for VACPP. If you are using VACPP, set VACPP=1 in your OS/2 environment before you compile.

Be sure to read the explanations in the Makefiles for the switches we use with the IBM C Set compiler. You will want to compile and link your demos with case exact turned on, meaning you want the compiler/linker to call glVertex3fv, not GLVERTEX3FV. Be sure to specify a large stack-if you look at our .def files (another C Set-ism), we are specifying a 1 MB stack for small demos. Currently to run Viewperf, the official OpenGL benchmark, we specify a STACKSIZE of 4 MB. You won't get a STACKSIZE this big unless you specify it. The OpenGL entry points pass the parameters on the stack, so your calling sample should do this (as opposed to passing in registers). Read through the C Set compiler/linker parameters as described in the makefiles.

Running the Demos

To successfully run the demos, use the following checklist:

- If you are running OS/2 Warp on your system, make sure multimedia is installed, and the following device statement is in CONFIG.SYS.

  ```
  DEVICE=c:\mmos2\ssmdd.sys
  ```

- Verify that you are running in 8-bit (256 colors), 16-bit or 24-bit mode.
- Check the demo source code to see if it requires flags to be passed to the compiler.
  wave -db -dr

  -db for double buffered

  -dr for direct rendering (-ir for slower indirect rendering)

Sources for OpenGL Information

There are many sources for OpenGL information:

- Web sites on the Internet
  - OpenGL benchmarking numbers for OS/2 can be found in the GPC Quarterly published by the Graphics Peformance Characterization group. For more information, see http://www.specbench.org/gpc/opc.static/index.html.
  - You can get information about an OS/2 mailing list, as well as a list of the FAQ for OpenGL on OS/2 from: http://www.utsi.com/˜kgl/os2-opengl/index.html.
  - And don't forget to visit the IBM web site for OpenGL at http://www.austin.ibm.com/software/OpenGL.
- USENET newgroup
  - comp.graphics.api.opengl
- OpenGL reference books
  - *OpenGL Reference Manual, The Official Reference Document for OpenGL, Release 1* , published by Addison Wesley, ISBN: 0-201-63276-4
  - *OpenGL Programming Guide* , published by Addison Wesley, ISBN: 0-201-63274-8
  - *OpenGL Specification* available using anonymous ftp from sgi.com in /sgi/opengl/doc

----

OS/2 Windowing and Input Functions

The following table describes the Presentation Manager GL (PGL) windowing and input functions that support OpenGL rendering function.

FUNCTION	DESCRIPTION
pglChooseConfig	Returns a pointer to a visual configuration structure, based on minimum requirements input
pglCopyContext	Copies a portion of state from the source OpenGL context to the destination OpenGL context.
pglCreateContext	Creates an OpenGL context using the visual configuration structure, and returns a handle to the context.
pglDestroyContext	Destroys an OpenGL context and the resources it owns.
pglGetCurrentContext	Returns the handle to the current OpenGL context.
pglGetCurrentWindow	Returns the current window handle that is bound to an OpenGL context.
pglGrabFrontBitmap	Grabs the bitmap in the front buffer of the current OpenGL window.
pglIsIndirect	Determines whether the context handle belongs to a direct or indirect context.
pglMakeCurrent	Binds an OpenGL context to an OS/2 PM window.
pglQueryCapability	Queries the OpenGL capability of the current machine.
pglQueryConfigs	Returns pointers to available visual configuration structures.
pglReleaseFrontBitmap	Releases the bitmap that was grabbed using pglGrabFrontBitmap.
pglSwapBuffers	Swaps the front and back buffers of the current window handle.
pglUseFont	Creates the specified number of OpenGL display lists containing bitmaps of the named logical character set identifier.
pglWaitGL	Ensures that all OpenGL rendering functions are completed before any GPI rendering functions are executed.
pglWaitPM	Ensures that all GPI rendering functions are completed before any OpenGL rendering functions are executed.

----

pglChooseConfig

----

pglChooseConfig - Syntax

This function returns a pointer to a visual configuration structure that meets the specified minimum requirements. A PVISUALCONFIG structure is required when creating an OpenGL rendering context.

```
#include <pgl.h>

HAB                 hab;
int                *attriblist;
PVISUALCONFIG      *pVisualConfig;

pVisualConfig = pglChooseConfig(hab, attriblist);
```

----

pglChooseConfig Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglChooseConfig Parameter - attriblist

attriblist (int *) - input
A list of attribute/value pairs. The last attribute in the list must be None.

The following is a list of valid attributes:

PGL_USE_GL	Ignored. Only visual configurations that can be rendered with OpenGL are considered.
PGL_BUFFER_SIZE	Must be followed by a nonnegative integer that indicates the desired color index buffer size. The smallest index buffer of at least the specified size is preferred. Ignored if PGL_RGBA is asserted.
PGL_LEVEL	Must be followed by an integer buffer-level specification. This specification is honored exactly. Buffer level zero corresponds to the default frame buffer of the display. Buffer level one is the first overlay frame buffer, level two the second overlay frame buffer, and so on. Negative buffer levels correspond to underlay frame buffers.
PGL_RGBA	If present, only RGBA visual configs are considered. Otherwise, Color Index visual configs are considered.
PGL_DOUBLEBUFFER	If present, only double bufferered visual configs are considered. Otherwise both single buffered and double buffered visual configs may be considered.
PGL_SINGLEBUFFER	If present, only single buffered visual configs are considered. Otherwise both single buffered and double buffered visual config may be considered.
PGL_STEREO	If present, only stereo visual configs are considered. Otherwise, both monoscopic and stereoscopic visual configs are considered.
PGL_AUX_BUFFERS	Must be followed by a nonnegative integer that indicates the desired number of auxiliary buffers. Visual configs with the smallest number of auxiliary buffers that meets or exceeds the specified number are preferred.
PGL_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest

	available red buffer is preferred. Otherwise, the largest available red buffer of at least the minimum size is preferred.
PGL_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available green buffer is preferred. Otherwise, the largest available green buffer of at least the minimum size is preferred.
PGL_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available blue buffer is preferred. Otherwise, the largest available blue buffer of at least the minimum size is preferred.
PGL_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available alpha buffer is preferred. Otherwise, the largest available alpha buffer of at least the minimum size is preferred.
PGL_DEPTH_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no depth buffer are preferred. Otherwise, the largest available depth buffer of at least the minimum size is preferred.
PGL_STENCIL_SIZE	Must be followed by a nonnegative integer that indicates the desired number of stencil bitplanes. The smallest stencil buffer of at least the specified size is preferred. If the desired value is zero, visual configs with no stencil buffer are preferred.
PGL_ACCUM_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no red accumulation buffer are preferred. Otherwise, the largest possible red accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no green accumulation buffer are preferred. Otherwise, the largest possible green accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no blue accumulation buffer are preferred. Otherwise, the largest possible blue accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no alpha accumulation buffer are preferred. Otherwise, the largest possible alpha accumulation buffer of at least the minimum size is preferred.

----

pglChooseConfig Return Value - pVisualConfig

pVisualConfig (PVISUALCONFIG *) - returns
Pointer to a null-terminated array of pointers to VISUALCONFIG structures containing buffer configs. A NULL is returned if no suitable VisualConfig was found.

----

pglChooseConfig - Parameters

hab (HAB) - input
Handle to anchor block.

attriblist (int *) - input

A list of attribute/value pairs. The last attribute in the list must be None.

The following is a list of valid attributes:

PGL_USE_GL	Ignored. Only visual configurations that can be rendered with OpenGL are considered.
PGL_BUFFER_SIZE	Must be followed by a nonnegative integer that indicates the desired color index buffer size. The smallest index buffer of at least the specified size is preferred. Ignored if PGL_RGBA is asserted.
PGL_LEVEL	Must be followed by an integer buffer-level specification. This specification is honored exactly. Buffer level zero corresponds to the default frame buffer of the display. Buffer level one is the first overlay frame buffer, level two the second overlay frame buffer, and so on. Negative buffer levels correspond to underlay frame buffers.
PGL_RGBA	If present, only RGBA visual configs are considered. Otherwise, Color Index visual configs are considered.
PGL_DOUBLEBUFFER	If present, only double bufferered visual configs are considered. Otherwise both single buffered and double buffered visual configs may be considered.
PGL_SINGLEBUFFER	If present, only single buffered visual configs are considered. Otherwise both single buffered and double buffered visual config may be considered.
PGL_STEREO	If present, only stereo visual configs are considered. Otherwise, both monoscopic and stereoscopic visual configs are considered.
PGL_AUX_BUFFERS	Must be followed by a nonnegative integer that indicates the desired number of auxiliary buffers. Visual configs with the smallest number of auxiliary buffers that meets or exceeds the specified number are preferred.
PGL_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available red buffer is preferred. Otherwise, the largest available red buffer of at least the minimum size is preferred.
PGL_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available green buffer is preferred. Otherwise, the largest available green buffer of at least the minimum size is preferred.
PGL_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available blue buffer is preferred. Otherwise, the largest available blue buffer of at least the minimum size is preferred.
PGL_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, the smallest available alpha buffer is preferred. Otherwise, the largest available alpha buffer of at least the minimum size is preferred.
PGL_DEPTH_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no depth buffer are preferred. Otherwise, the largest available depth buffer of at least the minimum size is preferred.
PGL_STENCIL_SIZE	Must be followed by a nonnegative integer that indicates the desired number of stencil bitplanes. The smallest stencil buffer of at least the specified size is preferred. If the desired value is zero, visual configs with no stencil buffer are preferred.
PGL_ACCUM_RED_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no red accumulation buffer are preferred. Otherwise, the largest possible red accumulation

	buffer of at least the minimum size is preferred.
PGL_ACCUM_GREEN_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no green accumulation buffer are preferred. Otherwise, the largest possible green accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_BLUE_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no blue accumulation buffer are preferred. Otherwise, the largest possible blue accumulation buffer of at least the minimum size is preferred.
PGL_ACCUM_ALPHA_SIZE	Must be followed by a nonnegative minimum size specification. If this value is zero, visual configs with no alpha accumulation buffer are preferred. Otherwise, the largest possible alpha accumulation buffer of at least the minimum size is preferred.

pVisualConfig (PVISUALCONFIG *) - returns
Pointer to a null-terminated array of pointers to VISUALCONFIG structures containing buffer configs. A NULL is returned if no suitable VisualConfig was found.

----

pglChooseConfig - Remarks

The pglChooseConfig function returns a suitable PVISUALCONFIG based on the minimum requirements passed with *attriblist*. Each Boolean attribute is matched exactly, and each integer attribute meets or exceeds the specified value. All Boolean attributes default to False, except PGL_USE_GL, which defaults to True. Default attributes are superseded by the attributes listed in *attriblist*. Boolean attributes included in *attriblist* are understood to be true, and integer attributes are followed immediately by the corresponding desired or minimum value. Attributes that are not specified have default values listed below. The *attriblist* is terminated by None.

Following is an example for *attriblist*:

```
attribList ={PGL_RGBA, PGL_RED_SIZE, 4, PGL_GREEN_SIZE, 4,PGL_BLUE_SIZE, 4,
             PGL_DOUBLEBUFFER, None};
```

The example specifies a double-buffered RGB visual configuration in the normal frame buffer, not an overlay or underlay buffer. The returned visual configuration supports at least four bits each of red, green, and blue, and possibly no bits of alpha. It does not support color index mode or stereo display. It may or may not have one or more auxiliary color buffers, a depth buffer, a stencil buffer, or an accumulation buffer.

Do not modify the fields of the returned VISUALCONFIG structure. When you are done with the PVISUALCONFIG, you can free the memory by calling the C library routine free().

----

pglChooseConfig - Related Functions

Related Functions

- pglQueryConfigs
- pglCreateContext

----

pglChooseConfig - Topics

Select an item:
Syntax
Parameters

Returns
Remarks
Related Functions

---

pglCopyContext

---

pglCopyContext - Syntax

This function copies a portion of state from the source OpenGL context to the destination OpenGL context. The *attrib_mask* parameter determines the group of state variables that are copied.

```
#include <pgl.h>

HAB        hab;
HGC        hgc_src;
HGC        hgc_dst;
GLuint     attrib_mask;
BOOL       rc;

rc = pglCopyContext(hab, hgc_src, hgc_dst,
       attrib_mask);
```

---

pglCopyContext Parameter - hab

hab (HAB) - input
    Handle to anchor block.

---

pglCopyContext Parameter - hgc_src

hgc_src (HGC) - input
    Handle to the source OpenGL context.

---

pglCopyContext Parameter - hgc_dst

hgc_dst (HGC) - input
    Handle to the destination OpenGL context.

----

pglCopyContext Parameter - attrib_mask

attrib_mask (GLuint) - input
Specifies the portions of the source context to be copied to the destination context.

----

pglCopyContext Return Value - rc

rc (BOOL) - returns
The following values can be returned:

TRUE	Context was successfully copied from hgc_src to hgc_dst.
FALSE	Error occurred.

----

pglCopyContext - Parameters

hab (HAB) - input
Handle to anchor block.

hgc_src (HGC) - input
Handle to the source OpenGL context.

hgc_dst (HGC) - input
Handle to the destination OpenGL context.

attrib_mask (GLuint) - input
Specifies the portions of the source context to be copied to the destination context.

rc (BOOL) - returns
The following values can be returned:

TRUE	Context was successfully copied from hgc_src to hgc_dst.
FALSE	Error occurred.

----

pglCopyContext - Remarks

attrib_mask contains the bitwise OR of the same symbolic names that can be passed to glPushAttrib. Not all values of OpenGL state can be copied. For example, pixel pack and pixel unpack state, render mode state, select and feedback state are not copied.

To copy the maximum amount of state, set *attrib_mask* to GL_ALL_ATTRIB_BITS. This copy can be done only if *hgc_src* and *hgc_dst* are created in the same process.

If the source context is not current to the thread issuing the request, the state of the destination context is undefined.

----

pglCopyContext - Related Functions

Related Functions

- pglCreateContext

----

pglCopyContext - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglCreateContext

----

pglCreateContext - Syntax

This function creates an OpenGL context, using the specified visual configuration structure, and returns a handle to the context.

```
#include <pgl.h>

HAB                 hab;
PVISUALCONFIG       pVisualConfig;
HGC                 ShareList;
BOOL                IsDirect;
HGC                 hgc;

hgc = pglCreateContext(hab, pVisualConfig,
        ShareList, IsDirect);
```

----

pglCreateContext Parameter - hab

hab (HAB) - input
    Handle to anchor block.

----

pglCreateContext Parameter - pVisualConfig

pVisualConfig (PVISUALCONFIG) - input
Pointer to VISUALCONFIG structure that contains the desired buffer configuration.

----

pglCreateContext Parameter - ShareList

ShareList (HGC) - input
Handle of OpenGL context with which to share display lists within a process.

----

pglCreateContext Parameter - IsDirect

IsDirect (BOOL) - input
Context to be created is direct.

TRUE	Bypass Graphics Programming Interface and render directly to a PM window. (This method is generally faster.)
FALSE	Use the GpiBitBlt function to blit OpenGL rendering to a PM window.

----

pglCreateContext Return Value - hgc

hgc (HGC) - returns
Handle to successfully created context, or NULL if an error occurred.

----

pglCreateContext - Parameters

hab (HAB) - input
Handle to anchor block.

pVisualConfig (PVISUALCONFIG) - input
Pointer to VISUALCONFIG structure that contains the desired buffer configuration.

ShareList (HGC) - input
Handle of OpenGL context with which to share display lists within a process.

IsDirect (BOOL) - input
Context to be created is direct.

TRUE	Bypass Graphics Programming Interface and render directly to a PM window. (This method is generally faster.)
FALSE	Use the GpiBitBlt function to blit OpenGL rendering to a PM window.

hgc (HGC) - returns
Handle to successfully created context, or NULL if an error occurred.

----

pglCreateContext - Remarks

The PVISUALCONFIG structure specifies the frame buffer resources that are available to the rendering context. If an OpenGL context is successfully created, a handle to it is returned, else NULL is returned. The available configurations can be listed through pglQueryConfigs, and chosen using pglChooseConfig.

Direct contexts are not available on all OS/2 configurations. If the IsDirect parameter is True but a direct context is not available, an indirect context is created. To determine the type of OpenGL context you have created, use the pglIsIndirect function.

The indirect context uses GpiBitBlt to blit the image to the screen and has access to a PM bitmap containing the OpenGL rendered image. The bitmap is not guaranteed to contain any OpenGL rendering until the OpenGL graphics pipeline has been flushed by glFlush, pglSwapBuffers, or pglWaitGL.

An arbitrary number of contexts can share display lists, but each context must be owned by the same process. If ShareList contains an OpenGL context handle, all display list definitions are shared by ShareList and the newly created context.

----

pglCreateContext - Related Functions

Related Functions

- pglQueryConfigs
- pglChooseConfig
- pglMakeCurrent
- pglDestroyContext

----

pglCreateContext - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglDestroyContext

----

pglDestroyContext - Syntax

This function destroys an OpenGL context and the resources it owns. If the context is currently bound to a window handle, the call to pglDestroyContext is ignored.

```
#include <pgl.h>

HAB     hab;
HGC     hgc;
BOOL    rc;

rc = pglDestroyContext(hab, hgc);
```

----

pglDestroyContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglDestroyContext Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

----

pglDestroyContext Return Value - rc

rc (BOOL) - returns
The following values can be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

If the context handle is currently bound to a window, pglDestroyContext returns without destroying the handle. The context can be unbound from the window by calling pglMakeCurrent (hab,NULL,None).

----

pglDestroyContext - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - input
Handle to an OpenGL context.

rc (BOOL) - returns
The following values can be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

If the context handle is currently bound to a window, pglDestroyContext returns without destroying the handle. The context can be unbound from the window by calling pglMakeCurrent (hab,NULL,None).

----

pglDestroyContext - Related Functions

Related Functions

- pglCreateContext
- pglMakeCurrent

----

pglDestroyContext - Topics

Select an item:
Syntax
Parameters
Returns
Related Functions

----

pglGetCurrentContext

----

pglGetCurrentContext - Syntax

This function looks up the current OpenGL context for the process. A context is current if it is the last context to be bound to a window by calling pglMakeCurrent.

```
#include <pgl.h>

HAB    hab;
HGC    hgc;

hgc = pglGetCurrentContext(hab);
```

----

pglGetCurrentContext Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglGetCurrentContext Return Value - hgc

hgc (HGC) - returns
Handle to the current context. NULL is returned if no current context exists for this process.

----

pglGetCurrentContext - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - returns
Handle to the current context. NULL is returned if no current context exists for this process.

----

pglGetCurrentContext - Remarks

Only one current context is allowed per process.

----

pglGetCurrentContext - Related Functions

Related Functions

- pglMakeCurrent
- pglCreateContext
- pglGetCurrentWindow

----

pglGetCurrentContext - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglGetCurrentWindow

----

pglGetCurrentWindow - Syntax

This function returns the current window handle that is bound to an OpenGL context. A window is current if it was the last window to be bound to an OpenGL context by calling pglMakeCurrent.

```
#include <pgl.h>

HAB     hab;
HWND    hwnd;

hwnd = pglGetCurrentWindow(hab);
```

----

pglGetCurrentWindow Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglGetCurrentWindow Return Value - hwnd

hwnd (HWND) - returns
Handle to the current window that is bound to an OpenGL context. NULL is returned if no window is currently bound to any OpenGL context for the calling process.

----

pglGetCurrentWindow - Parameters

hab (HAB) - input
Handle to anchor block.

hwnd (HWND) - returns
Handle to the current window that is bound to an OpenGL context. NULL is returned if no window is currently bound to any OpenGL context for the calling process.

----

pglGetCurrentWindow - Remarks

Only one current window is allowed per process.

----

pglGetCurrentWindow - Related Functions

Related Functions

- pglMakeCurrent
- pglGetCurrentContext

----

pglGetCurrentWindow - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglGrabFrontBitmap

----

pglGrabFrontBitmap - Syntax

This function queries the HPS and HBITMAP containing the bitmap representation of the front buffer of the current OpenGL window. Valid values are returned only if you are rendering with an indirect context.

```
#include <pgl.h>

HAB          hab;
HPS         *phps;
HBITMAP     *phbitmap;
BOOL         rc;

rc = pglGrabFrontBitmap(hab, phps, phbitmap);
```

----

pglGrabFrontBitmap Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglGrabFrontBitmap Parameter - phps

phps (HPS *) - input
Pointer returning HPS with phbitmap set in it.

----

pglGrabFrontBitmap Parameter - phbitmap

phbitmap (HBITMAP *) - input
Pointer returning HBITMAP containing OpenGL rendering.

----

pglGrabFrontBitmap Return Value - rc

rc (BOOL) - returns

TRUE	Bitmap has been grabbed.
FALSE	No Current window exists for this process.

----

pglGrabFrontBitmap - Parameters

hab (HAB) - input
Handle to anchor block.

phps (HPS *) - input
Pointer returning HPS with phbitmap set in it.

phbitmap (HBITMAP *) - input
Pointer returning HBITMAP containing OpenGL rendering.

rc (BOOL) - returns

TRUE	Bitmap has been grabbed.
FALSE	No Current window exists for this process.

----

pglGrabFrontBitmap - Remarks

After locking the bitmap with pglGrabFrontBitmap, the application should call pglReleaseFrontBitmap as soon as possible to release the bitmap. While the bitmap is locked, the application cannot receive WM_SIZE or WM_ADJUSTPOSITION messages in the current OpenGL *hwnd*, and the current window cannot be sized. An implicit glFlush occurs before this call completes. Applications must call pglReleaseFrontBitmap when they are done with the HBITMAP.

The HBITMAP is valid only for the time between calls to pglGrabFrontBitmap and pglReleaseFrontBitmap. The HPS is valid until the current window is unbound by calling pglMakeCurrent.

While the bitmap is locked, applications should not:

- Destroy the HPS or HBITMAP.
- Use GpiSetPalette to modify the HPS's color palette.
- Disassociate the HPS from its DC or unset the bitmap from HPS.
- Set any PS size, units, and format using GpiSetPS.

In other words, "Look, but don't touch!"

See pglWaitPM and pglWaitGL for integrating OpenGL and Gpi drawing.

----

pglGrabFrontBitmap - Related Functions

Related Functions

- pglIsIndirect
- pglReleaseFrontBitmap
- pglWaitGL
- pglWaitPM

----

pglGrabFrontBitmap - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglIsIndirect

----

pglIsIndirect - Syntax

This function determines whether a context handle belongs to a direct context or an indirect one. The specified context handle must have been returned from pglCreateContext.

```
#include <pgl.h>

HAB     hab;
HGC     hgc;
LONG    rc;

rc = pglIsIndirect(hab, hgc);
```

----

pglIsIndirect Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglIsIndirect Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

----

pglIsIndirect Return Value - rc

rc (LONG) - returns
The following values can be returned:

0	This Context bypasses GPI and is faster.
< 0	Error occurred.
1	This Context uses conventional PM blitting methods and is slower. However it allows an application to integrate OpenGL and GPI rendering commands. See Remarks section.

----

pglIsIndirect - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - input
Handle to an OpenGL context.

rc (LONG) - returns
The following values can be returned:

0	This Context bypasses GPI and is faster.
< 0	Error occurred.
1	This Context uses conventional PM blitting methods and is slower. However it allows an application to integrate OpenGL and GPI rendering commands. See Remarks section.

----

pglIsIndirect - Remarks

If pglIsIndirect returns 1, it uses GPI to blit OpenGL rendering. Integration of OpenGL and GPI rendering functions is allowed only on indirect contexts and is controlled through the use of pglWaitPM and pglWaitGL Applications must call pglGrabFrontBitmap when they need access to the actual bitmap, and call pglReleaseFrontBitmap when they are done with it. The PM bitmap is not guaranteed to contain any OpenGL rendering until the OpenGL graphics pipeline has been flushed by a call to glFlush, pglSwapBuffers, or pglWaitGL.

Most things that can be done with GPI can also be done with OpenGL, so in most cases it is simpler to use OpenGL with direct contexts, which are usually faster than indirect contexts.

----

pglIsIndirect - Related Functions

Related Functions

- pglCreateContext

----

pglIsIndirect - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglMakeCurrent

----

pglMakeCurrent - Syntax

This function binds an OpenGL context to an OS/2 PM window. Only one context can be bound to a window at a time.

```
#include <pgl.h>

HAB     hab;
HGC     hgc;
HWND    hwnd;
BOOL    rc;

rc = pglMakeCurrent(hab, hgc, hwnd);
```

----

pglMakeCurrent Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglMakeCurrent Parameter - hgc

hgc (HGC) - input
Handle to an OpenGL context.

----

pglMakeCurrent Parameter - hwnd

hwnd (HWND) - input
Handle to a PM window.

----

pglMakeCurrent Return Value - rc

rc (BOOL) - returns
The following values may be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

----

pglMakeCurrent - Parameters

hab (HAB) - input
Handle to anchor block.

hgc (HGC) - input
Handle to an OpenGL context.

hwnd (HWND) - input
Handle to a PM window.

rc (BOOL) - returns
The following values may be returned:

TRUE	Context was successfully destroyed, or context was bound to a window.
FALSE	Error occurred.

----

pglMakeCurrent - Remarks

This function replaces the old current context (if there was one) with the new context specified by *hgc*. Before unbinding the old context, an implicit flush of the old context takes place. Subsequent OpenGL rendering calls use *hgc* to modify *hwnd*.

The first time *hgc* is made current to a window, its viewport is initialized to the full size of *hwnd*. Subsequent calls to pglMakeCurrent with *hgc* do not affect its viewport.

To unbind the current context without binding a new context, issue the call pglMakeCurrent(hab,NULL,None).

For an OpenGL context to be bound to a PM window, the window should be created with window client class CS_SIZEREDRAW and CS_MOVENOTIFY. During the time the context is bound to the window, the application cannot subclass the window with WinSubclassWindow.

----

pglMakeCurrent - Related Functions

Related Functions

- pglCreateContext
- pglGetCurrentContext
- pglGetCurrentWindow

----

pglMakeCurrent - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglQueryCapability

----

pglQueryCapability - Syntax

This function queries the OpenGL capability of the current machine.

```
#include <pgl.h>

HAB     hab;
LONG    rc;

rc = pglQueryCapability(hab);
```

----

pglQueryCapability Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglQueryCapability Return Value - rc

rc (LONG) - returns

NULL	No OpenGL Support on this machine.
1	OpenGL Support available through PM and DIVE.
2	OpenGL support available through PM blitting only.
3	Advanced OpenGL support available.

----

pglQueryCapability - Parameters

hab (HAB) - input
Handle to anchor block.

rc (LONG) - returns

NULL	No OpenGL Support on this machine.
1	OpenGL Support available through PM and DIVE.
2	OpenGL support available through PM blitting only.
3	Advanced OpenGL support available.

----

pglQueryCapability - Topics

Select an item:
Syntax
Parameters
Returns

----

pglQueryConfigs

----

pglQueryConfigs - Syntax

This function returns a NULL terminated array of pointers to available visual configuration structures.

```
#include <pgl.h>

HAB               hab;
PVISUALCONFIG     pVisualConfig;

pVisualConfig = pglQueryConfigs(hab);
```

----

pglQueryConfigs Parameter - hab

hab (HAB) - input
    Handle to anchor block.

----

pglQueryConfigs Return Value - pVisualConfig

pVisualConfig (PVISUALCONFIG) - returns
    A NULL-terminated array of pointers to VISUALCONFIG structures. The list of structures is not guaranteed to be in any certain order. A NULL is returned if no suitable VISUALCONFIG was found.

----

pglQueryConfigs - Parameters

hab (HAB) - input
    Handle to anchor block.

pVisualConfig (PVISUALCONFIG) - returns
    A NULL-terminated array of pointers to VISUALCONFIG structures. The list of structures is not guaranteed to be in any certain order. A NULL is returned if no suitable VISUALCONFIG was found.

----

pglQueryConfigs - Remarks

A visual configuration structure defines the buffers that are available to the OpenGL rendering context. A VISUALCONFIG is required to create an OpenGL context and is used for rendering operations such as depth, accumulation, alpha, and stenciling.

After pglQueryConfigs returns the available visual configurations, the application should choose the visual configuration that has only the buffers that will be used by the rendering context. Your application can call pglChooseConfig, which chooses a suitable configuration based on the minimum requirements you specify.

Note: Do not modify the fields of the returned PVISUALCONFIG structures. When your application is done with the returned PVISUALCONFIG list , that memory can be freed by calling the C library routine free().

----

pglQueryConfigs - Related Functions

Related Functions

- pglChooseConfig
- pglCreateContext

----

pglQueryConfigs - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglReleaseFrontBitmap

----

pglReleaseFrontBitmap - Syntax

This function releases the HBITMAP that was grabbed using pglGrabFrontBitmap. The calling process can now begin receiving WM_SIZE and WM_ADJUSTPOSITION messages in its current OpenGL window again.

```
#include <pgl.h>

HAB     hab;
BOOL    rc;

rc = pglReleaseFrontBitmap(hab);
```

----

pglReleaseFrontBitmap Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglReleaseFrontBitmap Return Value - rc

rc (BOOL) - returns

TRUE	Bitmap was released.
FALSE	No bitmap needs to be released.

----

pglReleaseFrontBitmap - Parameters

hab (HAB) - input
Handle to anchor block.

rc (BOOL) - returns

TRUE	Bitmap was released.
FALSE	No bitmap needs to be released.

----

pglReleaseFrontBitmap - Remarks

See pglWaitPM and pglWaitGL for integrating OpenGL and PM drawing. Applications should call pglReleaseFrontBitmap as soon as possible after locking the bitmap with pglGrabFrontBitmap.

----

pglReleaseFrontBitmap - Related Functions

Related Functions

- pglIsIndirect
- pglGrabFrontBitmap
- pglWaitGL
- pglWaitPM

----

pglReleaseFrontBitmap - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglSelectColorIndexPalette

----

pglSelectColorIndexPalette - Syntax

This function passes a color index palette for OpenGL to use and is needed only when using a context created with a color index VISUALCONFIG. When using an RGB VISUALCONFIG, OpenGL sets up the palette itself and selects it when needed. The call to pglSelectColorIndexPalette must be made before a context can be bound to a window for the first time.

```
#include <pgl.h>

HAB      hab;    /*  Handle to the anchor block. */
HPAL     hpal;   /*  Handle to the palette. */
HGC      hgc;    /*  Handle to the color index OpenGL context. */
BOOL     rc;     /*  Success indicator. */

rc = pglSelectColorIndexPalette(hab, hpal,
       hgc);
```

----

pglSelectColorIndexPalette Parameter - hab

hab (HAB) - input
    Handle to the anchor block.

----

pglSelectColorIndexPalette Parameter - hpal

hpal (HPAL) - input
    Handle to the palette.

----

pglSelectColorIndexPalette Parameter - hgc

hgc (HGC) - input
    Handle to the color index OpenGL context.

----

pglSelectColorIndexPalette Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Palette selection was successful.
FALSE	An error occurred.

----

pglSelectColorIndexPalette - Parameters

hab (HAB) - input
Handle to the anchor block.

hpal (HPAL) - input
Handle to the palette.

hgc (HGC) - input
Handle to the color index OpenGL context.

rc (BOOL) - returns
Success indicator.

TRUE	Palette selection was successful.
FALSE	An error occurred.

----

pglSelectColorIndexPalette - Remarks

Do *not* call GpiSelectPalette; OpenGL does this for you. The pglSelectColorIndexPalette function is used only for Color Index contexts.

----

pglSelectColorIndexPalette - Topics

Select an item:
Syntax
Parameters
Returns
Remarks

----

pglSwapBuffers

----

pglSwapBuffers - Syntax

This function swaps the front and back buffers of *hwnd*. Double-buffered rendering is done when smooth animation between frames is desired.

```
#include <pgl.h>

HAB     hab;
HWND    hwnd;

pglSwapBuffers(hab, hwnd);
```

----

pglSwapBuffers Parameter - hab

hab (HAB) - input
Handle to anchor block.

----

pglSwapBuffers Parameter - hwnd

hwnd (HWND) - input
Handle to a PM window.

----

pglSwapBuffers - Return Value

----

pglSwapBuffers - Parameters

hab (HAB) - input
Handle to anchor block.

hwnd (HWND) - input
Handle to a PM window.

----

pglSwapBuffers - Remarks

When a window's buffers are swapped, the back buffer becomes the front, and the front buffer becomes the back. An implicit glFlush is done by pglSwapBuffers before it returns. OpenGL commands issued after calling pglSwapBuffers are not issued until the buffer swap is complete. This function has no effect on windows attached to single-buffered contexts.

The programmer can control which buffer is affected by OpenGL rendering calls through the use of glDrawBuffer. Front and back buffers are not created for a window until it has been bound to an OpenGL context. The window specified by *hwnd* does not have to be *currently* bound to an OpenGL context; it needs only to have been bound at some point. This function has no effect on a PM window that has never been bound to an OpenGL context.

----

pglSwapBuffers - Topics

Select an item:
Syntax
Parameters
Returns
Remarks

----

pglUseFont

----

pglUseFont - Syntax

This function creates the specified number (*count*) of OpenGL display lists containing bitmaps of the named logical character set identifier *(llcid*).

```
#include <pgl.h>

HAB       hab;
HPS       hps;
FATTRS    fattrs;
LONG      llcid;
int       first;
int       count;
int       listbase;
BOOL      rc;

rc = pglUseFont(hab, hps, fattrs, llcid, first,
       count, listbase);
```

----

pglUseFont Parameter - hab

hab (HAB) - input
    Handle to anchor block.

----

pglUseFont Parameter - hps

hps (HPS) - input
Handle to a PS that the logical font was created with.

----

pglUseFont Parameter - fattrs

fattrs (FATTRS) - input
Font Attributes (was used in GpiCreateLogFont).

----

pglUseFont Parameter - llcid

llcid (LONG) - input
Logical Set ID for created Logical Font.

----

pglUseFont Parameter - first

first (int) - input
Index of first glyph to be taken.

----

pglUseFont Parameter - count

count (int) - input
Number of glyphs to be taken.

----

pglUseFont Parameter - listbase

listbase (int) - input
Index of first display list to be created.

----

pglUseFont Return Value - rc

rc (BOOL) - returns

FALSE	Error occurred.
TRUE	Bitmap display lists were successfully created.

----

pglUseFont - Parameters

hab (HAB) - input
Handle to anchor block.

hps (HPS) - input
Handle to a PS that the logical font was created with.

fattrs (FATTRS) - input
Font Attributes (was used in GpiCreateLogFont).

llcid (LONG) - input
Logical Set ID for created Logical Font.

first (int) - input
Index of first glyph to be taken.

count (int) - input
Number of glyphs to be taken.

listbase (int) - input
Index of first display list to be created.

rc (BOOL) - returns

FALSE	Error occurred.
TRUE	Bitmap display lists were successfully created.

----

pglUseFont - Remarks

Before calling pglUseFont, the application should create the OS/2 logical font using the specified *hps* , *llcid* , and *fattrs*

Each bitmap consists of a single glBitmap command. Display lists are numbered *listbase* , through *listbase* + count -1. The parameters to glBitmap for display list *listbase* +i are derived from bitmap *first* +i in the logical font. OpenGL might delay glBitmap creation until a font glyph is accessed.

Empty display lists are created for all glyphs requested but not defined in the logical font. No display lists are created if there is no current OpenGL context.

----

pglUseFont - Related Functions

Related Functions

- pglIsIndirect
- pglWaitGL
- pglWaitPM

---

pglUseFont - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

---

pglWaitGL

---

pglWaitGL - Syntax

This function ensures that all OpenGL rendering commands made before the call to pglWaitGL are executed, and that the GPI rendering calls made after pglWaitGL is issued are executed after the return of pglWaitGL.

```
#include <pgl.h>

HAB    hab;
HPS    hps;

hps = pglWaitGL(hab);
```

---

pglWaitGL Parameter - hab

hab (HAB) - input
Handle to anchor block.

---

pglWaitGL Return Value - hps

hps (HPS) - returns
Handle to the PS that is used to blit the OpenGL image to the screen. This HPS can be used for GPI drawing commands. NULL is returned if no current context exists, or if the current context is not using GPI to display OpenGL images.

----

pglWaitGL - Parameters

hab (HAB) - input
Handle to anchor block.

hps (HPS) - returns
Handle to the PS that is used to blit the OpenGL image to the screen. This HPS can be used for GPI drawing commands. NULL is returned if no current context exists, or if the current context is not using GPI to display OpenGL images.

----

pglWaitGL - Remarks

The pglWaitGL and pglWaitPM functions enable an application to alternate GPI rendering functions with OpenGL rendering functions. Intermingling of GPI drawing with OpenGL drawing is allowed only on OpenGL indirect contexts, which use GPI to blit OpenGL rendering functions to the screen.

The pglWaitGL function flushes the OpenGL rendering stream in preparation for the issuing of GPI rendering functions to the current indirect context. The call to pglWaitGL is ignored if there is no current context or if the current context is a direct context.

The return value for this function is HPS, which has the bitmap set in it to be the bitmap containing OpenGL rendering.

----

pglWaitGL - Related Functions

Related Functions

- pglIsIndirect
- pglWaitPM

----

pglWaitGL - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

----

pglWaitPM

----

pglWaitPM - Syntax

This function ensures that all GPI rendering commands made before the call to pglWaitPM are executed before the execution of OpenGL rendering calls made after pglWaitPM.

```
#include <pgl.h>

HAB      hab;

pglWaitPM(hab);
```

----

pglWaitPM Parameter - hab

hab (HAB) - input
    Handle to anchor block.

----

pglWaitPM - Return Value

----

pglWaitPM - Parameters

hab (HAB) - input
    Handle to anchor block.

----

pglWaitPM - Remarks

Intermingling of GPI drawing with OpenGL drawing is allowed only on OpenGL indirect contexts. Indirect contexts use GPI to blit OpenGL rendering functions to the screen. GPI drawing is not guaranteed to be visible until pglWaitPM has been called.

GPI has no concept of front and back buffers; therefore all GPI drawing commands are assumed to affect the OpenGL front buffer.

The call to pglWaitPM is ignored if there is no current context.

----

pglWaitPM - Related Functions

Related Functions

- pglIsIndirect
- pglWaitGL

----

pglWaitPM - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Related Functions

OpenGL Data Types

This section describes the following data types that are used with the pgl functions:

- VISUALCONFIG

VISUALCONFIG

The Visual Configuration structure is required when creating an OpenGL context.

```
typedef struct _VISUALCONFIG {
  ULONG                     vid;             /*  Visual ID. */
  BOOL                      rgba;
  int                       redSize;
  int                       greenSize;
  int                       blueSize;
  int                       alphaSize;
  ULONG                     redMask;
  ULONG                     greenMask;
  ULONG                     blueMask;
  ULONG                     accumRedSize;
  ULONG                     accumGreenSize;
  ULONG                     accumBlueSize;
  ULONG                     accumAlphaSize;
  BOOL                      doubleBuffer;
  BOOL                      stereo;
  int                       bufferSize;
  int                       depthSize;
  int                       stencilSize;
  int                       auxBuffers;
  int                       level;
  PVOID                     reserved;
  struct visualconfig      *next;
} VISUALCONFIG;

typedef VISUALCONFIG *PVISUALCONFIG;
```

VISUALCONFIG Field - vid

vid (ULONG)
Visual ID.

VISUALCONFIG Field - rgba

rgba (BOOL)

VISUALCONFIG Field - redSize

redSize (int)

VISUALCONFIG Field - greenSize

greenSize (int)

VISUALCONFIG Field - blueSize

blueSize (int)

VISUALCONFIG Field - alphaSize

alphaSize (int)

VISUALCONFIG Field - redMask

redMask (ULONG)

VISUALCONFIG Field - greenMask

greenMask (ULONG)

VISUALCONFIG Field - blueMask

blueMask (ULONG)

VISUALCONFIG Field - accumRedSize

accumRedSize (ULONG)

VISUALCONFIG Field - accumGreenSize

accumGreenSize (ULONG)

VISUALCONFIG Field - accumBlueSize

accumBlueSize (ULONG)

VISUALCONFIG Field - accumAlphaSize

accumAlphaSize (ULONG)

VISUALCONFIG Field - doubleBuffer

doubleBuffer (BOOL)

VISUALCONFIG Field - stereo

stereo (BOOL)

VISUALCONFIG Field - bufferSize

bufferSize (int)

VISUALCONFIG Field - depthSize

depthSize (int)

VISUALCONFIG Field - stencilSize

stencilSize (int)

VISUALCONFIG Field - auxBuffers

auxBuffers (int)

VISUALCONFIG Field - level

level (int)

VISUALCONFIG Field - reserved

reserved (PVOID)

VISUALCONFIG Field - next

next (struct visualconfig *)

OpenGL Sample

The following sample code draws a gouraud shaded octahedron.

```
/* sample code for using OpenGL and PGL  */
/* Draws a gouraud shaded octahedron     */

#include <stdio.h>
#include "pgl.h"     /* PGL calls    */
```

```
#include "gl.h"      /* OpenGL calls */
#define PM_ESCAPE 0x0f
#define MSGBOXID 22
#define SQRT2  1.414

/* attributes passed into pglChooseConfig */
int attriblist[] = {
  PGL_DOUBLEBUFFER,  /* request doublebuffered visual config   */
  PGL_RGBA,          /* request rgb (true color) visual config */
  None               /* always end list with this              */
};

HAB hab;

void DispError(PSZ errstr)
{
  char buffer[256];
  sprintf(buffer, "Error (0x%x) in SAMPOGL.EXE:", WinGetLastError(hab));
  WinMessageBox(HWND_DESKTOP,HWND_DESKTOP,errstr,buffer,
                MSGBOXID,MB_MOVEABLE!MB_CUACRITICAL!MB_CANCEL);
  exit(0);
}

void Setup()
{
  glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_FALSE);
  glDepthMask(GL_FALSE);
  glEnable(GL_CULL_FACE);
}

float verts[][3] = {
  { 0.0, 0.0, (1.0/SQRT2)},
  { 0.5, 0.5, 0.0},
  {-0.5, 0.5, 0.0},
  {-0.5,-0.5, 0.0},
  { 0.5,-0.5, 0.0},
  { 0.0, 0.0, -(1.0/SQRT2)}
};

float colors[][3] = {
  {1.0, 1.0, 1.0},
  {1.0, 0.0, 0.0},
  {0.0, 1.0, 0.0},
  {0.0, 0.0, 1.0},
  {1.0, 0.0, 1.0},
  {0.0, 1.0, 1.0},
};

MRESULT EXPENTRY WindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
  static float t = 0.0;
  static SWP clientsize;
  static USHORT mycode;
  static UCHAR key;

  switch(msg) {
  case WM_SIZE:
    /* Upon a resize, query new window size and set OpenGL viewport */
    WinQueryWindowPos(hwnd,&clientsize);
    glViewport(0, 0, clientsize.cx, clientsize.cy);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
  case WM_TIMER:
    /* Upon getting a timer message, the invalidate rectangle call  */
    /* will cause a WM_PAINT message to be sent, enabling animation */
    WinInvalidateRect(hwnd, NULLHANDLE, NULL);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
  case WM_PAINT:
    /* This is what is done for every frame of the animation        */
    t += 1.0;
    glClear(GL_COLOR_BUFFER_BIT ! GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotatef(t, 1.0, 1.0, 1.0);
    glBegin(GL_TRIANGLE_FAN);
      glColor3fv(colors[0]);
      glVertex3fv(verts[0]);
      glColor3fv(colors[1]);
      glVertex3fv(verts[1]);
      glColor3fv(colors[2]);
      glVertex3fv(verts[2]);
      glColor3fv(colors[3]);
      glVertex3fv(verts[3]);
      glColor3fv(colors[4]);
      glVertex3fv(verts[4]);
      glColor3fv(colors[1]);
      glVertex3fv(verts[1]);
    glEnd();
    glBegin(GL_TRIANGLE_FAN);
      glColor3fv(colors[5]);
      glVertex3fv(verts[5]);
      glColor3fv(colors[1]);
      glVertex3fv(verts[1]);
      glColor3fv(colors[4]);
      glVertex3fv(verts[4]);
      glColor3fv(colors[3]);
      glVertex3fv(verts[3]);
      glColor3fv(colors[2]);
      glVertex3fv(verts[2]);
      glColor3fv(colors[1]);
      glVertex3fv(verts[1]);
    glEnd();
    glPopMatrix();
    pglSwapBuffers(hab, hwnd);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
  case WM_CHAR:
    mycode = (USHORT)SHORT1FROMMP(mp1);
    if ((mycode & KC_CHAR) && !(mycode & KC_KEYUP))
      key = CHAR1FROMMP(mp2);
    else if ((mycode & KC_VIRTUALKEY) && !(mycode & KC_KEYUP))
      key = CHAR3FROMMP(mp2);
    if (key == PM_ESCAPE)
      WinPostMsg(hwnd, WM_CLOSE, (MPARAM)0, (MPARAM)0);
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
  default:
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
  }
}
main(int argc, char **argv)
{
  PVISUALCONFIG vishead; /* visual configuration             */
  HMQ hmq;               /* message queue                    */
  HWND hwnd;
  HWND hwndFrame;
  ULONG createflags = FCF_TITLEBAR !
                      FCF_SYSMENU  !
                      FCF_MINMAX   !
                      FCF_SIZEBORDER;
  QMSG qmsg;             /* message                          */
  HGC hgc;               /* OpenGL context                   */
  int  major, minor;     /* OpenGL version                   */
  int err;

  hab = WinInitialize(0);
  hmq = WinCreateMsgQueue(hab, 0);
  if (!hmq)
    DispError("Couldn't create a message queue!\n");

  /* Check to see if OpenGL exists */
  if (pglQueryCapability(hab)) {
    pglQueryVersion(hab, &major, &minor);
    /* Version 1.0                */
    if ((major == 1) && (minor == 0)) {
      /* Choose a visual configuration that matches desired  */
      /* attributes in attriblist                            */
      vishead = pglChooseConfig(hab, attriblist);
      if (!vishead)
        DispError("Couldn't find a visual!\n");
      if (WinRegisterClass(
            hab,
            (PSZ)"PGLtest",
            WindowProc,
            CS_SIZEREDRAW ! CS_MOVENOTIFY, /* Need at least this! */
            0))
      {
        hwndFrame = WinCreateStdWindow (
                      HWND_DESKTOP,            /* Child of the desktop    */
                      WS_VISIBLE,              /* Frame style             */
                      &createflags,        /* min FCF_MENU!FCF_MINMAX */
                      (PSZ)"PGLtest",  /* class name              */
                      "OpenGL Sample", /* window title            */
                      WS_VISIBLE,             /* client style            */
                      0,                      /* resource handle         */
                      1,                      /* Resource ID             */
                      &hwnd);             /* Window handle           */
        if (!hwndFrame)
          DispError("Couldn't create a window!\n");
        /* you must set window size before you call pglMakeCurrent */
        if (!WinSetWindowPos(
               hwndFrame,
               HWND_TOP,
               0,
               0,
               300,
               300,
               SWP_ACTIVATE ! SWP_SIZE ! SWP_MOVE ! SWP_SHOW))
          DispError("Couldn't position window!\n");
        hgc = pglCreateContext(hab,  /* anchor block handle    */
                 vishead,            /* visual configuration   */
                 (HGC)NULL,          /* (no) shared contexts   */
                 (BOOL)TRUE);        /* direct (fast) context  */
        if (!hgc)
          DispError("Couldn't create an OpenGL context!\n");
        if(!pglMakeCurrent(hab, hgc, hwnd))
          DispError("Could not bind OpenGL context to window!\n");
        /* Don't subclass your window past here! */
        Setup();
        /* Start timer to cause WM_TIMER messages to be sent   */
        /* periodically.  This is used to animate.             */
        WinStartTimer(hab, hwnd, 0L, 0L);
        while (WinGetMsg(hab, &qmsg, NULLHANDLE, 0, 0))
          WinDispatchMsg(hab, &qmsg);
      }
    }
  }
}
```

Notices

Copyright Notices

Disclaimers

Trademarks

OpenGL	Silicon Graphics, Inc.
