OGLES PVRVFrame

User Manual

Filename	:	OGLES PVRVFrame.User Manual.1.15f.External.doc
Version	:	1.15f External Issue (Package: POWERVR SDK 2.06.26.0709)
Issue Date	:	26 Feb 2010
Author	:	POWERVR

Contents

List of Figures

1. Introduction

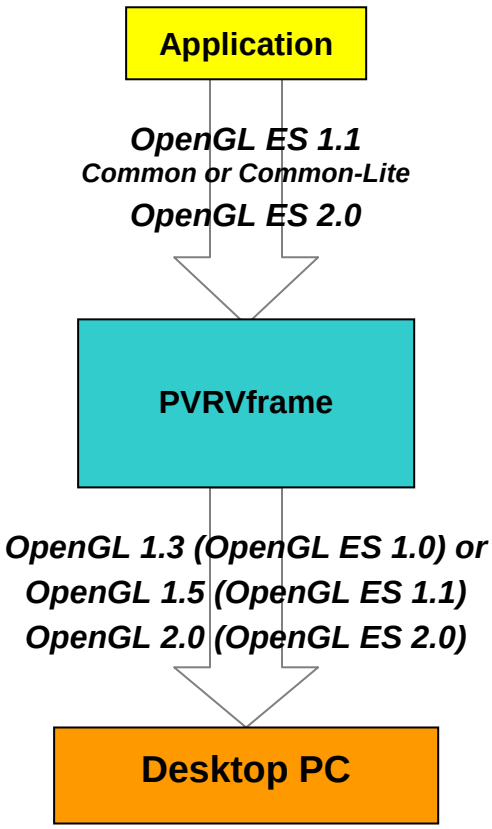
PVRVFrame is a desktop wrapper for OpenGL ES tailored to POWERVR devices. It works by re-directing OpenGL ES API calls to the underlying OpenGL desktop implementation present on the system. PVRVFrame is aimed at developers writing OpenGL ES 1.x and OpenGL ES 2.0 applications for POWERVR-enabled embedded devices, without the need for development hardware or a target platform.

PVRVFrame is ***not*** an emulator. The performance obtained when running OpenGL ES applications with PVRVFrame has no bearing on the performance obtained when running on real consumer hardware. PVRVFrame only gives a qualitative preview of an OpenGL ES application.

PVRVFrame is available for OpenGL ES 1.x, both Common and Common-Lite profiles, and OpenGL ES 2.0.

PVRVFrame must be used with the corresponding version of the Khronos OpenGL ES header files. The Khronos OpenGL ES header files can be retrieved from the Khronos website:

http://www.khronos.org/developers/specs/



1. Package contents

PVRVFrame consists of the following files:

1.1. Windows XP

libgles_cl.lib: Import ("stub") library to link against when accessing functions in **libgles_cl.dll**.
libgles_cm.lib: Import ("stub") library to link against when accessing functions in **libgles_cm.dll**

1.1.1. OGLES 1.1

libEGL.dll : PVRVFrame driver files for EGL1.2.

libgles_cl.dll : PVRVFrame driver files for OpenGL ES Common-Lite profile.
libgles_cm.dll : PVRVFrame driver files for OpenGL ES Common profile.

libEGL.lib: Import ("stub") library to link against when accessing functions in **libEGL.dll**.
libgles_cl.lib: Import ("stub") library to link against when accessing functions in **libgles_cl.dll**.
libgles_cm.lib: Import ("stub") library to link against when accessing functions in **libgles_cm.dll**.

1.1.2. OGLES 2.0

libEGL.dll : PVRVFrame driver files for EGL1.2.
libGLESv2.dll : PVRVFrame driver files for OpenGL ES 2.0.

libEGL.lib: Import ("stub") library to link against when accessing functions in **libEGL.dll**.
libGLESv2.lib: Import ("stub") library to link against when accessing functions in **libGLESv2.dll**.

1.2. Linux

1.2.1. OGLES 1.1

libEGL.so : PVRVFrame driver files for EGL1.2.
libgles_cl.so : PVRVFrame driver files for OpenGL ES Common-Lite profile.
libgles_cm.so : PVRVFrame driver files for OpenGL ES Common profile.

1.2.2. OGLES 2.0

libEGL.so : PVRVFrame driver files for EGL1.2.
libGLESv2.so : PVRVFrame driver files for OpenGL ES 2.0.

2. Installation

2.1. Windows

The DLL files must be copied to a directory that is accessible to the DLL search path (e.g. the default Windows directory or the current directory your application is running from).

Each PC Emulation SDK demo contains a Visual C++.NET solution and corresponding project files that have been setup to build with PVRVFrame.

You can setup a new Visual C project to run with PVRVFrame by following the steps below:

1. Create a new Visual C workspace, adding your required source and include files as necessary.
2. Download the Khronos include files (`gl.h` and `egl.h for OpenGL ES 1.x or gl2.h and egl.h for OpenGL ES 2.0`) for the desired OpenGL ES version and ensure they can be accessed from your project's include path.

   OpenGL ES 1.x:

   It is good practice to store `gl.h` and `egl.h` in a `GLES\` subfolder so that your source files can access them by adding the following lines to your code:

   ```
   #include <GLES\egl.h>
   #include <GLES\gl.h>
   ```

   OpenGL ES 2.0:

   It is good practice to store `gl2.h` and `egl.h` in a `GLES2\ and EGL\` subfolders so that your source files can access them by adding the following lines to your code:

   ```
   #include <EGL\egl.h>
   #include <GLES2\gl2.h>
   ```

3. OpenGL ES 1.x:

   Copy the `egltypes.h` include file from <SDKPackage>\Builds\OGLES\WindowsPC\Include\GLES to a GLES\ folder that can be accessed from your project's include path (this file is included in Khronos' `egl.h`, so there is no need to manually include this in your project).

   OpenGL ES 2.0:

   Copy the `eglplatform.h` include file from <SDKPackage>\Builds\OGLES2\Include\EGL to a EGL\ folder that can be accessed from your project's include path (this file is included in Khronos' `egl.h`, so there is no need to manually include this in your project).

4. Link your project to the supplied OGLES/OGLES2 libraries.
5. Ensure the DLL search path can access the PVRVFrame DLLs. A simple solution is to copy them to your application's current directory, although it may be desireable to update the search path to a static DLL folder location to allow multiple projects to use the same DLLs.

2.2. Linux

The PVRVFrame SO libraries must be accessible by the OS. To make them accessible, put them into your library path by using the following terminal command: “export LD_LIBRARY_PATH=<Folder containing the .so files>;$LD_LIBRARY_PATH”

Each PC Emulation SDK demo contains a makefile that is already set up to build with PVRVFrame.

To set up a new project to run with PVRVFrame, you must follow the steps below:

1. Create a new makefile, adding your required source and include files as necessary.
2. Download the Khronos include files (`gl.h` and `egl.h for OpenGL ES 1.x or gl2.h and egl.h for OpenGL ES 2.0`) for the desired OpenGL ES version and ensure they can be accessed from your project’s include path.

   OpenGL ES 1.x:

   It is good practice to store `gl.h` and `egl.h` in a `GLES/` subfolder so that your source files can access them by adding the following lines to your code:

   ```
   #include <GLES/egl.h>
   #include <GLES/gl.h>
   ```

   OpenGL ES 2.0:

   It is good practice to store `gl2.h` and `egl.h` in a `GLES2/ and EGL/` subfolders so that your source files can access those by adding the following lines to your code:

   ```
   #include <EGL/egl.h>
   #include <GLES2/gl2.h>
   ```

3. OpenGL ES 1.x:

   Copy the `egltypes.h` include file from <SDKPackage>/Builds/OGLES/LinuxPC/Include/GLES to a GLES/ folder that can be accessed from your project’s include path (this file is included in Khronos’ `egl.h`, so there is no need to manually include this in your project).

   OpenGL ES 2.0:

   Copy the `eglplatform.h` include file from <SDKPackage>/Builds/OGLES2/Include/EGL to an EGL/ folder that can be accessed from your project’s include path (this file is included in Khronos’ `egl.h`, so there is no need to manually include this in your project).

4. Link your project to the supplied OGLES/OGLES2 so libraries.
5. Use the following terminal command to ensure the system can access the PVRVFrame *.so files:
   “export LD_LIBRARY_PATH=<folder containing the .so files>;$LD_LIBRARY_PATH”

After successfully building your application, you can check it works with the emulation library by typing the following in the terminal: “ldd <path to application>”.
For example, if the library was in the same directory as your application, you could use the following command: “ldd ./<the name of your application>“

3. Usage of PVRVFrame

3.1. Working with GUI

When an OpenGL ES application linked against PVRVFrame emulation libraries runs, the interface window (shown in Figure 1) will be displayed.

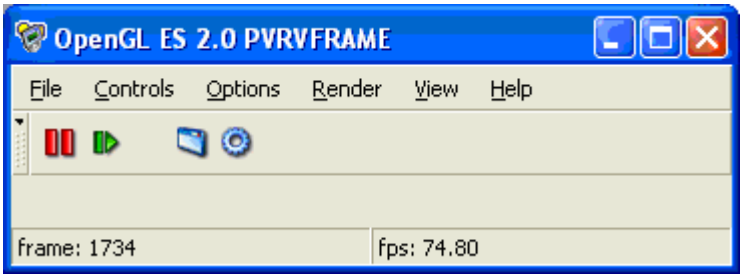


Figure 1

This interface allows you to control and inspect your application behaviour, and configuration.

Controlling and inspecting OpenGL ES application behaviour:

- To stop (pause) your application, click the *Pause* button (Figure 2) or choose *Stop* command from the toolbar's *Controls* drop-down menu
  Note: It is possible to start an application in a paused state by ticking the appropriate checkbox in the *General Options* window, which can be opened from the toolbar by clicking *Options->General* (Figure 9)

Figure 2

- To resume your application, click the *Play* button (Figure 3)(which replaces the pause icon when the application is in a paused state) or choose *Play* command from the toolbar's *Controls* drop-down menu.

Figure 3

- To run through your application step by step, you can click *Step* button (Figure 4) or choose *Step* command from the toolbar's *Controls* drop-down menu.

Note: The execution of this command depends on whether the application is time-based or frame-based. For frame-based application steps are frame length. For time-based application, steps differ among themselves and depend on the time passed during two subsequent steps.

Figure 4

- To change the rendering mode of the application (i.e. *Wireframe* or *Normal*), select the desired option from the toolbar's *Render* drop-down box (Figure 5).

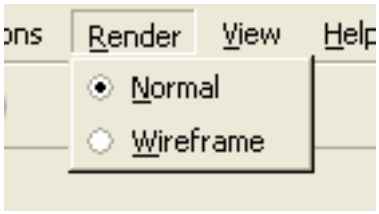


Figure 5

- To toggle the *Log*, click the *Log* button (Figure 6) or select the *ShowLog* option from the toolbar's *View* drop-down menu.

Figure 6

- The *Log Display* (Figure 7) contains information about OpenGL ES, such as: VENDOR, RENDERER, VERSION, SHADING_LANGUAGE_VERSION and EXTENSIONS. It also outputs error messages generated by the API, as well as information produced internally by the emulator, such as warnings.

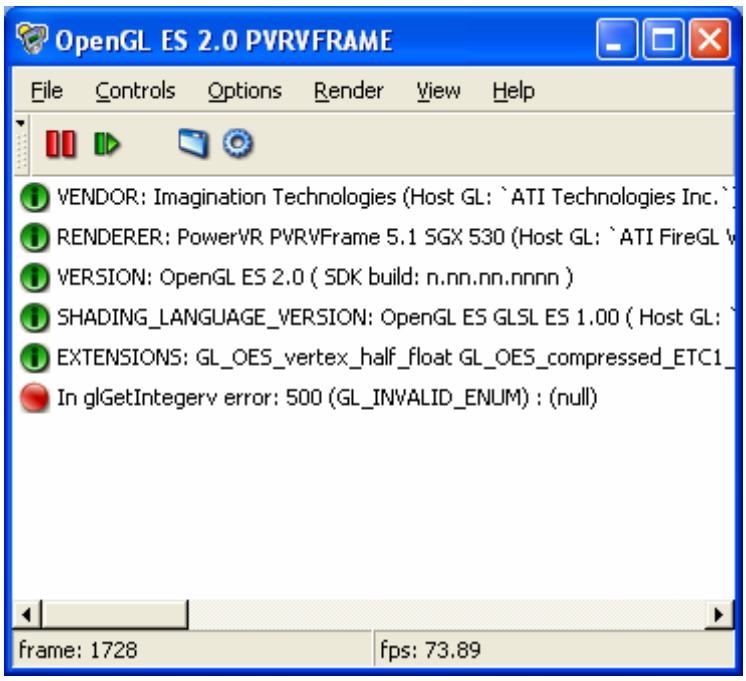


Figure 7

Configuring PVRVFrame:

Figure 8

- To change hardware profile of OGLES for application, click the *Options* button (Figure 8) or click *Options->General* in the toolbar.

  Upon doing this, you will see a list of hardware profiles in control window *General Options* window (Figure 9*)*.The default profile should be adequate for most cases. If required, you can select a different profile from this list that is a closer match to your target platform. Changing the profile will take effect after restarting application.

  Note: Some profiles may be too restrictive to handle some applications, for example, MbxLite may not allow you to run the *SDK ChameleonMan* demo. For this reason, you should proceed with caution when changing your applications profile, as it could prevent your application from running, which in turn could prevent you from gaining access to the hardware profile list through the PVRVFrame GUI (although there is a solution to this problem discussed in section 3.2).

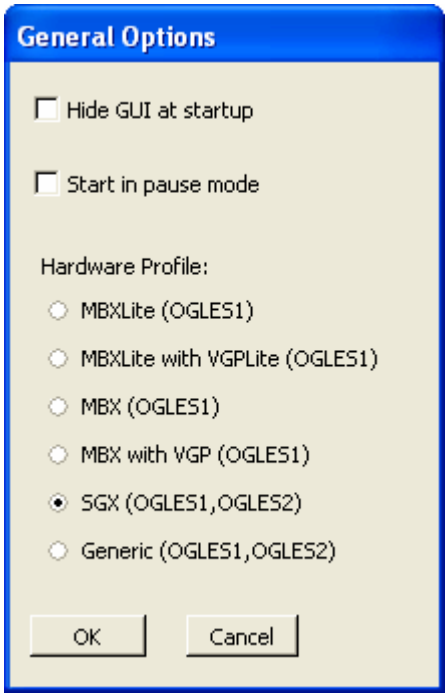


Figure 9

Configuring PVRVFrame interface:

- To hide GUI at start-up, tick the checkbox : *Hide GUI at start-up* in *General Options* control window (Figure 9)

  Note: To enable GUI again you need to manually configure PVRVFrame. Refer to section 3.2 to do this.

3.2. Manual configuration of PVRVFrame

In some cases, you may require a manual change to the PVRVFrame configuration without running your application. This can be easily done by using PVRVFrameSetup application (recommended method) or by manual editing the Windows Registry (Windows) or the configuration file (Linux).

3.2.1. Configuring PVRVFrame via PVRVFrameSetup

PVRVFrameSetup is an application that allows you to change the configuration of PVRVFrame, without having to run an application using PVRVFrame.

From here, you can change the hardware profile for OpenGL ES 1.x and OpenGL ES 2.0, toggle whether the control window should appear when running an application and choose if an application should start in pause mode (Figure 10).

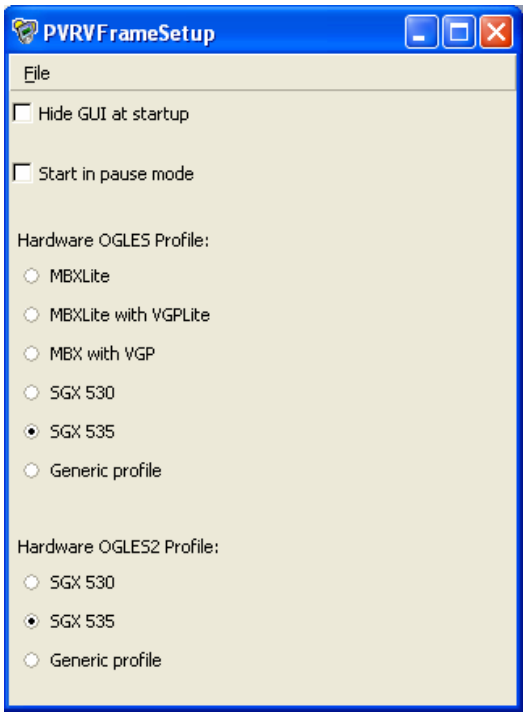


Figure 10

3.2.2. Configuring PVRVFrame via Registry (Windows)

- The current PVRVFrame 'profile' for OpenGL ES 1.x is stored in the registry file with the following key: HKEY_CURRENT_USER\Software\Imagination Technologies\PVRVFrame\OGLES

  To change the hardware profile, set the *hardware_profile* value to one of the following strings:

  `"MBXLITE","MBXLITE_VGPLITE”,"MBX_VGP","SGX530”,"SGX535" or "SUPPORT_ALL`
- The hardware 'profile' for OpenGL ES 2.0 is stored in the registry with the following key:

  HKEY_CURRENT_USER\Software\Imagination Technologies\PVRVFrame\OGLES2.

  To change a profile simply set the *hardware_profile* value to one of the following strings:

  `"SGX530”,"SGX535" or "SUPPORT_ALL`
- To enable or disable the GUI control window, modify the value stored in the registry file with the following key:

  HKEY_CURRENT_USER\Software\Imagination Technologies\PVRVFrame\STARTUP

  To enable GUI, set the *hide_gui* value to "NO" , or to disable it set the value to "YES".

3.2.3. Configuring PVRVFrame via a configuration file (Linux)

An equivalent configuration method is available under Linux. PVRVFrame configuration values are stored in a file in the home directory of the user running the application, for example: "*.foxrc/Imagination Technologies/PVRVFrame*". This file contains configuration sections that correspond to the keys (and values) described in the configuration of the Windows version (Configuring PVRVFrame via Registry (Windows)).

3.2.4. Enabling/disabling PVRVFrame from within your application

It is possible to enable or disable PVRVFrame GUI directly from your application. This method updates the settings stored in registry file/configuration file.

- To enable PVRVFrame GUI, put the following lines of code into your application before any EGL function calls are made. If you are using PVRShell, you should place this code into your implementation of the InitApplication() function.

```
void (*PVRVFrameEnableControlWindow)(bool)  =
(void(*)(bool))eglGetProcAddress("PVRVFrameEnableControlWindow");

if(PVRVFrameEnableControlWindow!=0)
{
        PVRVFrameEnableControlWindow (true);
}
```

- To disable the PVRVFrame GUI, usethe same code, but call PVRVFrameEnableControlWindow function with a *false* argument.

4. Supported Features and Extensions

4.1. Features

4.1.1. PBuffer

PBuffer is supported, but with limitations on some graphic cards (see “Future work and Current Limitations”).

4.1.2. FSAA

Fullscreen Anti-Aliasing is supported through the `wglChoosePixelFormatARB` extension that enumerates additional PFD configurations. Note that the FSAA quality obtained on the Host OpenGL implementation will not be a representation of the FSAA quality running on MBX/SGX devices.

4.2. OpenGLES 1.x Extensions

Extension	Comments
GL_IMG_texture_compression_pvrtc	Emulated by converting PVRTC data to 16-bit RGBA formats
GL_IMG_vertex_program	Software implementation, not available on MBXLite
GL_IMG_user_clip_plane	Implemented based on host functionality
GL_IMG_texture_format_BGRA8888	
GL_IMG_texture_env_enhanced_fixed_function	
GL_IMG_read_format	
GL_OES_framebuffer_object	SGX530,SGX535 and generic profiles only
GL_OES_compressed_ETC1_RGB8_texture	
GL_OES_compressed_paletted_texture	
GL_OES_depth24	
GL_OES_texture_float	
GL_OES_texture_half_float	
GL_OES_rgb8_rgba8	
GL_OES_stencil8	
GL_OES_matrix_palette	Software implementation
GL_OES_mapbuffer	Software implementation
GL_OES_point_size_array	
GL_OES_point_sprite	
GL_OES_vertex_half_float	
GL_EXT_multi_draw_arrays	SGX530,SGX535 and generic profiles only
GL_OES_byte_coordinates	
GL_OES_fixed_point	

Extension	Comments
GL_OES_single_precision	
GL_OES_matrix_get	
GL_OES_read_format	
GL_OES_query_matrix	
GL_OES_textureenv_crossbar	SGX530,SGX535 and generic profiles only
GL_OES_texture_mirrored_repeat	SGX530,SGX535 and generic profiles only
GL_OES_blend_func_separate	SGX530,SGX535 and generic profiles only
GL_OES_blend_equation_separate	SGX530,SGX535 and generic profiles only
GL_OES_draw_texture	
GL_EXT_multi_draw_arrays	SGX530,SGX535 and generic profiles only

4.3. OpenGLES 2.0 Extensions

Extension	Comments
GL_OES_vertex_half_float	
GL_OES_depth_texture	
GL_OES_compressed_ETC1_RGB8_texture	
GL_IMG_texture_compression_pvrtc	Emulated by converting PVRTC data to 16-bit RGBA formats
GL_OES_mapbuffer	
GL_OES_texture_half_float	
GL_OES_fragment_precision_high	
GL_OES_element_index_uint	
GL_EXT_multi_draw_arrays	

5. Future work and Current Limitations

5.1. PBuffer support

PBuffer support has two limitations:

- The eglMakeCurrent function call allows different surfaces to be set for drawing and reading. For example, drawing to the frame buffer but reading from the PBuffer. On desktop OpenGL, this functionality is provided via an extension: `WGL_ARB_make_current_read`. When this extension is not supported, the draw surface will be set for both reading and writing, resulting in potentially unexpected results. This extension is not supported on KYRO-based graphic cards.
- The eglCreatePbufferSurface function will create a separate context for each PBuffer surface, and the context will always be shared with the main window context. This behaviour is not correct, but was chosen due to compatibility issues with the supported pixel formats of the Desktop OpenGL implementations.

5.2. Pixmap support

PVRVFrame has currently no support for pixmap.

5.3. Vertex arrays

Data in GL_FIXED format is supported, although the implementation may be slow due to the fact that PVRVFrame will have to create a new array of the same size with data converted from GL_FIXED to a different format supported by the latest OpenGL.

5.4. glVertexAttrib{1234}f[v](uint indx, T values)

This function cannot be called with `index` equal to zero.

5.5. GL_IMG_texture_env_enhanced_fixed_function extension

The GL_ADD_BLEND_IMG mode of this extension is currently not emulated because combiners alone cannot offer an equivalent multitexture mode. Supporting this mode will involve using extra extensions.

5.6. GL_IMG_vertex_program extension

The IMG_position_invariant option is not guaranteed to deliver the same result for fixed function and vertex shading positions. This is because fixed function uses the desktop OpenGL fixed function implementation while the vertex shader functionality is emulated using the CPU.

The current Vertex Program implementation has an overhead per draw call, hence applications with a lot of draw calls may run slowly. While the PVRVFrame performance and MBX/SGX performance are not related to this, it is advisable to minimise the number of draw calls for optimal performance on both.

Bindings have not been extensively tested as such incorrect or unsupported bindings are a possibility. When an incorrect binding is suspected, check the debug output for warnings and contact POWERVR Developer Technology (devtech@imgtec.com) to resolve the issue.

5.7. OpenGLES 2.0 Shader Precision Qualifiers

PVRVFrame treats all precision qualifiers as high-precision.

Consumer devices may require medium or low accuracy mode to achieve better performance in shaders, but the degradation in quality caused by these changes will not be reflected in PVRVFrame's output. For this reason, you should always test shaders using medium and low precision modifiers on hardware to ensure the desired output is achieved. Compatibility and Requirements.

5.8. Hardware requirements

PVRVframe for OGLES 1.1 has been tested and has found to be working with the following video cards:

- Radeon 9500, 9700, 9800, catalyst drivers version 5.3.
- ATI FireGL V3350, catalyst drivers version 8.563
- GeForce6600, GeForce 6800, GeForce 2 GTS, drivers version 61.77.
- Intel GMA 3100

When using the graphics cards listed above (or better), you should update you desktop OpenGL drivers to their latest revision to ensure compatibility.

OGLES 2.0 PVRVframe will run with most modern graphics cards. Advanced OGLES 2.0 features will only be available on graphics cards that support shader model 3.0 or better. Vertex, Frame and Render buffer objects are supported only if your graphics card supports them (checking for these features is done during runtime and you will be warned if they are not present in your configuration).

5.9. Software requirements

PVRVFrame dosen't require any specific software, although graphics card drivers should be installed properly and a compiler and linker should be available for building your application.

Under Linu,x X server is required, but this is common in most of recent Linux distributions.

Note: If in some of distributions of Linux you encounter a problem when running applications using PVRVFrame then you may need to install glew libraries. On Ubuntu it may be done with commands :

sudo apt-get install libglut3-dev -y

sudo apt-get install libglew-dev -y

sudo apt-get install glew-utils -y

Bug reports should be addressed to devtech@imgtec.com.