OGLES PVRVFrame

User Manual

Filename	:	OGLES PVRVFrame.User Manual.1.11f.External.doc
Version	:	1.11f External Issue (Package: POWERVR SDK 2.05.25.0782)
Issue Date	:	05 Aug 2009
Author	:	POWERVR

Contents

1. Introduction

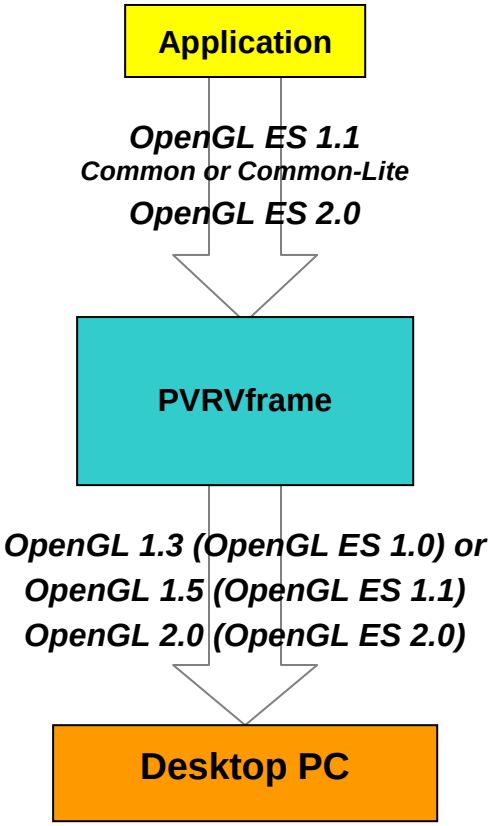
PVRVFrame is a desktop wrapper for OpenGL-ES tailored to POWERVR devices. It works by re-directing OpenGL ES API calls to the underlying OpenGL desktop implementation present on the system. PVRVFrame is aimed at developers wishing to write OpenGL ES1 and OpenGL ES2 applications for POWERVR-enabled embedded devices, without the need for any hardware or development 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 1.1, 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 to access functions in **libgles_cl.dll**.
libgles_cm.lib: Import ("stub") library to link against to access 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 to access functions in **libEGL.dll**.
libgles_cl.lib: Import ("stub") library to link against to access functions in **libgles_cl.dll**.
libgles_cm.lib: Import ("stub") library to link against to access 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 to access functions in **libEGL.dll**.
libGLESv2.lib: Import ("stub") library to link against to access 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).

The PC Emulation SDK demos contain ready-made Visual C++.NET solution and project files that are already setup to build with PVRVFrame.

To setup a brand new Visual C project to run with PVRVFrame follow 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 automatically included by Khronos' `egl.h`).

   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 automatically included by Khronos' `egl.h`).

4) Link your project with the libraries supplied.
5) Ensure PVRVFrame DLLs are accessible to the DLL search path. A simple solution is to copy them to the current directory that your application is running from.

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: "export LD_LIBRARY_PATH=<Folder containing the .so files>;$LD_LIBRARY_PATH"

The PC Emulation SDK demos contain ready-made makefiles that are already set up to build with PVRVFrame.

To set up a new project to run with PVRVFrame 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>
   ```

6) 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 automatically included by Khronos' `egl.h`).

   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 automatically included by Khronos' `egl.h`).

3. Link your project with the so libraries supplied.
4. Ensure PVRVFrame *.so are accessible to the system using:
   "export LD_LIBRARY_PATH=(folder containing the .so files);$LD_LIBRARY_PATH"
5. After a successful build of your application, you can check if your application is working with the emulation library by typing : "ldd [path to application]".
   Example where the library is in the same directory as your application: "ldd ./myapp"

3. Usage of PVRVFrame

3.1. Working with GUI

After starting any OpenGL ES application linked against PVRVFrame emulation libraries the interface Window will come up as (Figure 1).

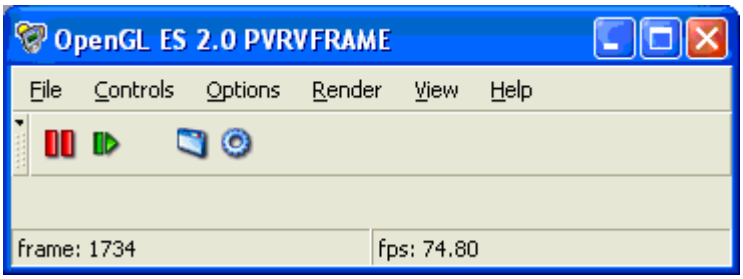


Figure 1

This interface is a way of controlling and inspecting your application behaviour and its configuration.

Controlling and inspecting OpenGL ES application behaviour:

- To Stop (pause) your application, click the Stop button (Figure 2) or choose Stop command from Menu *Controls*
  Note: It is possible to start an application paused ticking the appropriate checkbox in General options control window (Figure 9)

Figure 2

- To Resume playing your application, click the Play button (Figure 3) or choose Play command from Menu *Controls*

Figure 2

- To play the application step by step, click Step button (Figure 4) or choose Step command from Menu *Controls.*
  Note: The execution of this command depends on weather 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 among two subsequent steps.

Figure 3

- To change rendering mode of application (To *Wireframe* or to *Normal*), select appropriate menu command button (Figure 5) from *Render* menu

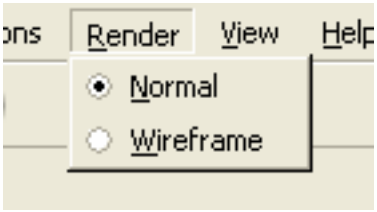


Figure 4

- To toggle the Log, click the button (Figure 6) or choose the option *ShowLog* from *View* menu.

Figure 5

The Log display contains information about OpenGL ES: VENDOR, RENDERER, VERSION, SHADING_LANGUAGE_VERSION, EXTENSIONS, etc. It also outputs error messages generated by the API and extra warnings and information produced internally by the emulator.

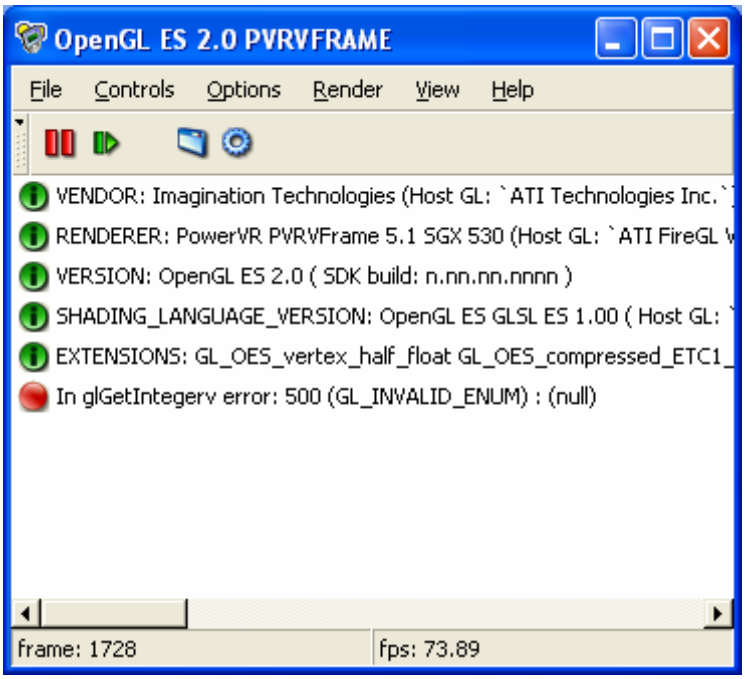


Figure 6

Configuring PVRVFrame:

- To change hardware profile of OGLES for application , click Options button (Figure 8) or choose *General* command from Menu *Options*

Figure 7

Depending on the API used in by the application you will see list of hardware profiles in control window *General Options (Figure 9)*. Select the profile that matches your target platform. Changing the profile will take effect after restarting application.

Note: Some profiles maybe not enough powerful to handle some of applications eg. MbxLite may not be sufficient to run *SDK ChameleonMan* demo. There might be a problem changing a profile back since you will not be able to run the application again to access the options. Solution to this problem is described in section 3.2

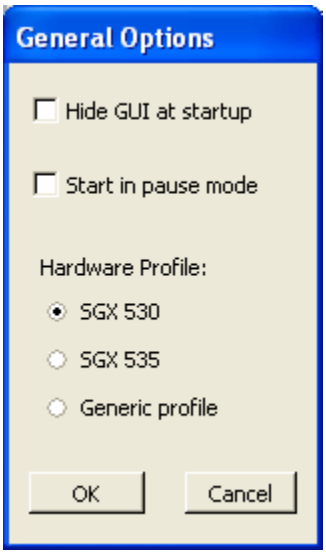


Figure 8

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

3.2.1. Configuring PVRVFrame via Registry (Windows):

In some cases it may be required to change manually the PVRVFrame configuration.

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

  To change the hardware profile simply set a value for *hardware_profile* 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 being stored in the registry under the key:

  HKEY_CURRENT_USER\Software\Imagination Technologies\PVRVFrame\OGLES2.

  To change a profile simply set a value for *hardware_profile* of that key to one of the following strings: `"SGX530”,"SGX535" or "SUPPORT_ALL`
- To enable or disable the control window (GUI) modify the value stored in registry file under in key:

  HKEY_CURRENT_USER\Software\Imagination Technologies\PVRVFrame\STARTUP

  To enable GUI set *hide_gui* field to value "NO" to make disable it set the value to "YES"

3.2.2. Configuring PVRVFrame via configuration file (Linux):

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

3.2.3. Enabling/disabling PVRVFrame from the application:

It is possible to enable or disable PVRVFrame GUI directly from the application. This method updates the settings stored in registry file and it will be persistent.

- To enable PVRVFrame GUI put following lines in before your application calls any EGL function. If you are using PVRShell then drop these lines in InitApplication method of PVRShell inherited class.

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

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

- To disabled PVRVFrame GUI just call the control function with *false*.

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 draw to frame buffer but read from PBuffer. On desktop OpenGL this functionality is provided by 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 generated, this 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 might 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 some different value 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 might 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 will ignore any type which is not high-precision and will run in high-precision always.

Consumer devices might require to run in medium or low accuracy mode with a possible degradation in quality that will not be reflected by PVRVFrame.

6. Compatibility

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

In general it is advised to update the desktop OpenGL drivers to their latest revision.

OGLES 2.0 PVRVframe will run with most modern graphics cards. Advanced OGLES2.0 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).

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