OGLES PVRVFrame

User Manual

Filename	:	OGLES PVRVFrame.User Manual.mht
Version	:	1.3f (PowerVR SDK 2.03.23.1153)
Issue Date	:	18 Aug 2008
Author	:	PowerVR

Contents

1. Introduction 4

2. Package contents 5

2.1. Windows XP 5
2.1.1. OGLES 1.0 & OGLES 1.1 5
2.1.2. OGLES 2.0 5
2.2. Linux 5
2.2.1. OGLES 1.0 & OGLES 1.1 5
2.2.2. OGLES 2.0 5

3. Installation 6

3.1. Windows 6
3.2. Linux 6

4. Configuration 8

oglespcviewer.cfg 8

5. Supported Features and Extensions 9

5.1. Features 9
5.1.1. PBuffer 9
5.1.2. FSAA 9
5.2. OpenGLES 1.x Extensions 9
5.2.1. GL_IMG_texture_compression_pvrtc 9
5.2.2. GL_IMG_texture_env_enhanced_fixed_function 9
5.2.3. GL_ARB_texture_env_combine 9
5.2.4. GL_ARB_texture_env_dot3 9
5.2.5. GL_IMG_vertex_program 9

6. Future work and Current Limitations 10

6.1. PBuffer support 10
6.2. Pixmap support 10
6.3. Vertex arrays 10
6.4. Cofiguration and oglepcviewer.cfg 10
6.5. glVertexAttrib{1234}f[v](uint indx, T values) 10
6.6. GL_IMG_texture_env_enhanced_fixed_function extension 10
6.7. GL_IMG_vertex_program extension 10
6.8. OpenGLES 2.0 Shader Precision Qualifiers 11

7. Compatibility 12

1. Introduction

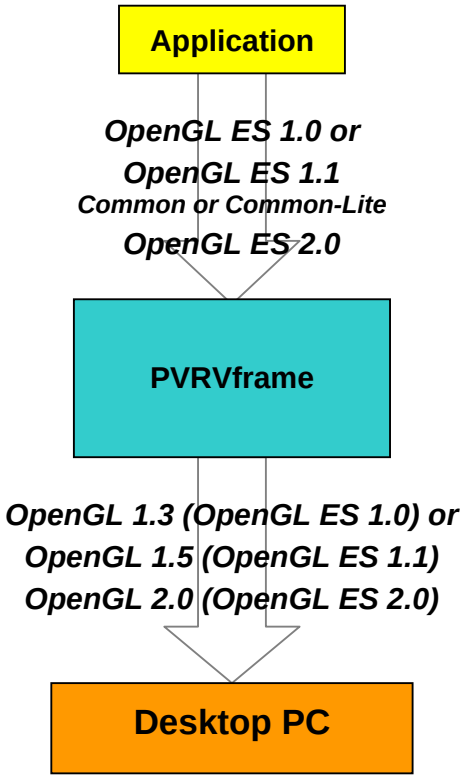
PVRVFrame is a desktop wrapper for OpenGL-ES tailored to PowerVR devices. It works by re-directing OpenGL ES API calls to the underlying OpenGL desktop implementation present on the system. PVRVFrame is aimed at developers wishing to write OpenGL ES applications for PowerVR-enabled embedded devices, 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 bearings 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 1.0, OpenGL ES 1.1, both Common and Common-Lite profiles, and OpenGL ES 2.0.

PVRVFrame must be used with its 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

1.1.1. OGLES 1.0

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

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 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.

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**.

1.1.3. 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 1.0

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 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.

1.2.3. 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 those 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\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 do so 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 setup to build with PVRVFrame.

To setup 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 those 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 a 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"

3. Configuration

PVRVFrame supports multiple configurations in order to represent the different variations of MBX 3D cores available. The choice of configuration is done through a text file called `oglespcviewer.cfg`. When a file built for PVRVFrame is executed it will attempt to open the configuration file in the current active directory. If found, the MBX configuration will be read from the file and applied to PVRVFrame settings. If this file cannot be found the default configuration will be used. Below is a list of supported keywords recognized by the PVRVFrame configuration file:

oglespcviewer.cfg

MBXConfig=[0/1/2/3/4]

The MBXConfig values indicate the MBX configuration to target. Different extensions and behaviour will be exposed depending on the selected configuration.

- “MBXConfig=0” indicate a “generic” MBX profile is used. In this mode, all MBX extensions are exposed. This is the default.
- “MBXConfig=1” indicates the MBX1 profile is used.
- “MBXConfig=2” indicates the MBX1 + VGP profile is used.
- “MBXConfig=3” indicates the MBX1Lite profile is used.
- “MBXConfig=4” indicates the MBX1Lite + VGPLite profile is used.

Note: These MBX configurations will be ignored in the OGLES 2.0 PVRVFrame

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

4.2.1. GL_IMG_texture_compression_pvrtc

This extension is emulated by converting PVRTC data to 16-bit RGBA formats.

4.2.2. GL_IMG_texture_env_enhanced_fixed_function

This extension is emulated with the GL_ARB_texture_env_combine extension. Support is currently incomplete though (see “Future Work and current limitations” section).

4.2.3. GL_ARB_texture_env_combine

This extension is emulated with the GL_ARB_texture_env_combine of the Host OpenGL implementation.

4.2.4. GL_ARB_texture_env_dot3

This extension is emulated with the GL_ARB_texture_env_dot3 of the Host OpenGL implementation.

4.2.5. GL_IMG_vertex_program

This extension is emulated by converting the binary vertex program to software instructions.

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 potential 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 cannot be passed to glVertexAttribPointer. This will be fixed in next releases, although the implementation will 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. Cofiguration and oglepcviewer.cfg

Different configurations are needed when new hardware arises, this makes PVRVFrame out of date every once in a while. On the other hand, oglespcviewer.cfg is barely being used. This is about to change in the future in a way that oglespcviewer.cfg will contain as much information about a HW as possible.

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

This function cannot be called with `indx` equal to zero. This is a limitation carried from the OpenGL and will be fixed in the future releases.

5.6. 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.

5.7. 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 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 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 debug output for warnings and contact PowerVR Developer Relations to resolve the issue.

5.8. 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 the PVRVFrame.

6. Compatibility

PVRVframe for OGLES1.0/1.1 has been tested and has found to be working with the following video cards:

- KYRO, 2.1 drivers (PVRVframe for OpenGL ES 1.0 only)
- Radeon 9500, 9700, 9800, catalyst drivers version 5.3.
- GeForce 6800, GeForce 2 GTS, drivers version 61.77.

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

OGLES 2.0 PVRVframe will run in most modern graphic cards. Advanced OGLES2.0 features will only be available on graphics cards that supports shader model 3.0 or better. Vertex, Frame and Render buffer objects are supported only if your graphic 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.