PVRTrace

User Manual

Filename	:	PVRTrace.User Manual.1.6.2.2f.External.doc
Version	:	1.6.2.2f External Issue (Package: POWERVR SDK 2.06.26.0662)
Issue Date	:	13 Apr 2010
Author	:	Imagination Technologies

Contents

1. Introduction **3**
2. Package Contents **3**
2.1. Linux 3
2.1.1. OpenGL ES 2.0 3
2.1.2. OpenGL ES 1.1 3
2.2. Windows 3
2.2.1. OpenGL ES 2.0 3
2.2.2. OpenGL ES 1.1 3
2.3. Analysis Tool 4
2.4. Example trace files 4
2.5. PVRTrace – Recording Overview 5
2.6. Recording on Linux 5
2.6.1. Library structure – OpenGL ES 2.0 5
2.6.2. Library structure – OpenGL ES 1.1 5
2.6.3. Example – Recording on Linux OMAP3 6
2.7. Recording with PVRVFrame 7
2.7.1. Library structure 7
2.7.2. Example – Recording via PVRVFrame on Linux 7
2.7.3. Example – Recording via PVRVFrame on Windows 8
3. PVRTraceGUI – Analysis Tool **9**
3.1.1. Overview 9
3.2. Function Calls List 10
3.3. Function Counts List 10
3.4. Frame Summary 11
3.5. Textures List 11
3.6. Data Viewer 11
3.7. Find Function Dialog 12
3.8. Filtering Functions 12

List of Figures

Figure 1 - PVRTrace OpenGL ES2 linux library overview 5
Figure 2 - PVRTrace OpenGL ES2 Windows PC emulation library overview 7
Figure 3 - PVRTraceGUI main window 9
Figure 4 - Function call item 10
Figure 5 - Function Counts List 10
Figure 6 - Frame Summary 11
Figure 7 - Textures List 11
Figure 8 - Data Viewer 11
Figure 9 - Find Function Dialog 12

1. Introduction

PVRTrace is a utility that captures the graphics API calls made by an OpenGL ES application and displays this information so it can be analysed. It consists of several components. The recording is done by a library which sits between the application and the OpenGL ES libraries. The data analysis is done by a GUI application.

PVRTrace recording libraries are distributed as part of the SDK packages, the correct version must be used for recording on each platform. The PVRTrace analysis tool can be used with recordings taken from any platform.

2. Package Contents

PVRTrace consists of the following files:

2.1. Linux

2.1.1. OpenGL ES 2.0

libEGL.so:	PVRTrace record library for EGL1.4.
libGLESv2.so:	PVRTrace record library for OpenGL ES 2.0.
libPVRTrace.so:	PVRTrace library used by libEGL.so and libGLESv2.so.

2.1.2. OpenGL ES 1.1

libGLES_CM.so:	PVRTrace record library both EGL and OpenGL ES 1.1.
libEGL.so:	PVRTrace record library for EGL1.4.
libGLESv1_CM.so:	PVRTrace record library for OpenGL ES 1.1.
libPVRTrace.so:	PVRTrace library used by libEGL, libGLESv1_CM and libGLES_CM.

2.2. Windows

2.2.1. OpenGL ES 2.0

libEGL.dll:	PVRTrace record library for EGL1.4.
libGLESv2.dll:	PVRTrace record library for OpenGL ES 2.0.
PVRTrace.dll:	PVRTrace library used by libEGL.dll and libGLESv2.dll.
libEGL.lib:	Import ("stub") library to link against for functions in libEGL.dll.
libGLESv2.lib:	Import ("stub") library to link against for functions in libGLESv2.dll.

2.2.2. OpenGL ES 1.1

libGLES_CM.dll:	PVRTrace record library both EGL and OpenGL ES 1.1.
libEGL.dll:	PVRTrace record library for EGL1.4.
libGLESv1_CM.dll:	PVRTrace record library for OpenGL ES 1.1.
libPVRTrace.dll:	PVRTrace library used by libEGL, libGLESv1_CM and libGLES_CM.
libGLES_CM.lib:	Import ("stub") library to link against for functions in libGLES_CM.dll.
libEGL.lib:	Import ("stub") library to link against for functions in libEGL.dll.
libGLESv1_CM.lib:	Import ("stub") library to link against for functions in libGLESv1_CM.dll.

2.3. Analysis Tool

Win32/PVRTraceGUI.exe:	Windows version of the analysis application.
Linux/PVRTraceGUI:	Linux version of the analysis application.
MacOS/PVRTraceGUI:	Mac OS version of the analysis application.

2.4. Example trace files

OGLES2HelloTriangle.pvrt:	Example trace file.
OGLES2IntroducingPVRTools.pvrt:	Example trace file.

2.5. PVRTrace – Recording Overview

Recording is done by two libraries; one for OpenGL ES and one for EGL. The application to be recorded must link against the PVRTrace libraries instead of the platform's host libraries (see Figure 1). These PVRTrace libraries record the trace and call the appropriate functions in the host libraries, so the application will visually run just like it would when using the host libraries on their own.

PVRTrace looks for the host libraries in the location specified in the file `pvrtrace.cfg`, which is located in the same place as the executable. If no config file exists, PVRTrace will create a template one. The first time an application runs, you should open the configuration file in a text editor to ensure the paths point to the platform's host libraries. Once the paths have been updated, you should re-run the application.

When the application quits, a `.pvrt` file is output containing all of the trace data. The default location of this file is the directory the application is called from, but it can be set to a different directory in `pvrtrace.cfg`.

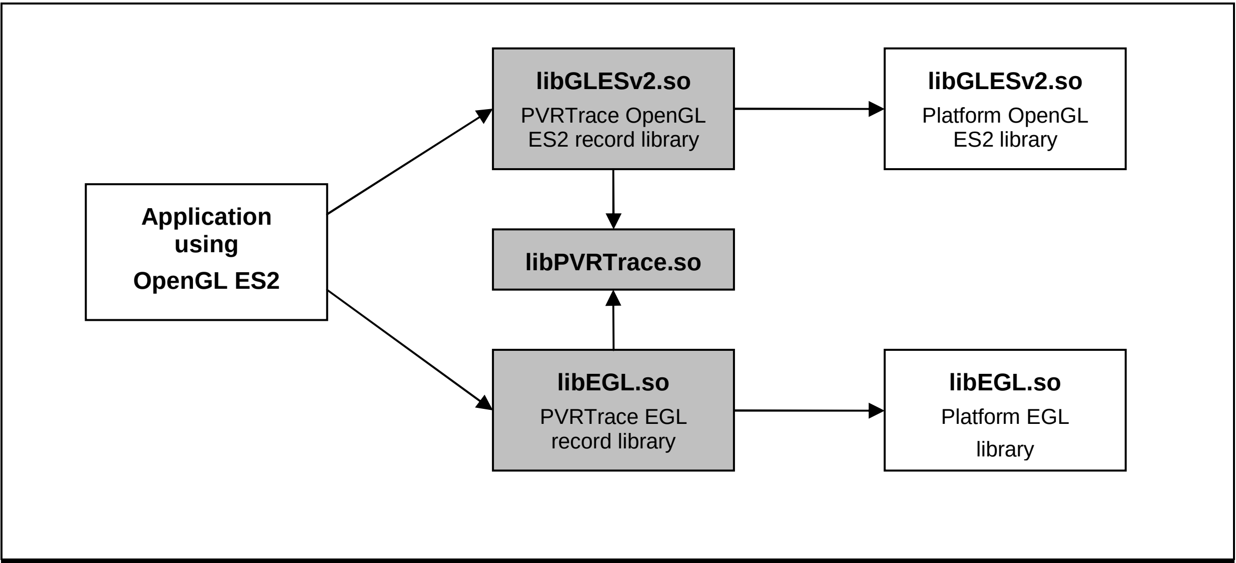
The names (and entry points) of the PVRTrace versions of libEGL and libGLES are the same as the host versions of those libraries. This means it is possible to run PVRTrace on a pre-built application as the PVRTrace libraries will be dynamically loaded at run-time instead of the host libraries.

The recording procedure is automatic, i.e. recording begins when you launch your application, and completes when the application shuts down.

2.6. Recording on Linux

2.6.1. Library structure – OpenGL ES 2.0

Figure 1 shows the structure of the libraries for an application using OpenGL ES2 on Linux. The application links against the PVRTrace versions of `libGLESv2.so` and `libEGL.so` and they call functions in the host platform's `libGLESv2.so` and `libEGL.so` dynamic libraries. They both use shared functionality in `libPVRTrace.so` as part of the recording process.



libGLESv2.so
PVRTrace OpenGL ES2 record library
libGLESv2.so
Platform OpenGL ES2 library
Application using OpenGL ES2
libPVRTrace.so
libEGL.so
PVRTrace EGL record library
libEGL.so
Platform EGL library

Figure 1 - PVRTrace OpenGL ES2 linux library overview

2.6.2. Library structure – OpenGL ES 1.1

Figure 1 shows the structure of the libraries for an application using OpenGL ES1.1 on Linux. The application links against the PVRTrace versions of `libGLES_CM.so` and it calls functions in the host platform's `libGLES_CM.so` dynamic library. Alternately, the application can link against `libGLESv1_CM.so` and `libEGL.so` similarly to the case for OpenGL ES2. They all use shared functionality in `libPVRTrace.so` as part of the recording process.

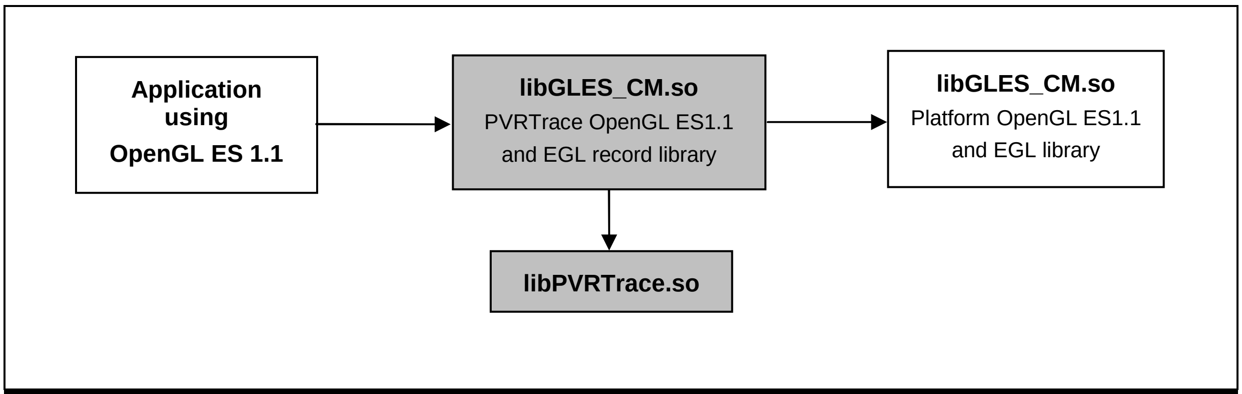

Application using OpenGL ES 1.1
libGLES_CM.so
PVRTrace OpenGL ES1.1 and EGL record library
libGLES_CM.so
Platform OpenGL ES1.1 and EGL library
libPVRTrace.so

Figure 1 - PVRTrace OpenGL ES 1.1 linux library overview

2.6.3. Example – Recording on Linux OMAP3

The example below explains how to run the IntroducingPVRTools tutorial application with PVRTrace using the POWERVR OpenGL ES 2.0 Linux OMAP3 SDK.

Build the desired application for OMAP3.

```
cd TrainingCourse/06_IntroducingPVRTools/OGLES2/Build/LinuxGeneric
make PLATFORM=LinuxOMAP3
```

Change directory to the location of the executable.

```
cd ../LinuxOMAP3/ReleaseRaw
```

Setup the `LD_LIBRARY_PATH` to include the location of the PVRTrace `libEGL.so` and `libGLESv2.so` libraries for OMAP3. You may prefer to use the absolute path to this location.

```
export LD_LIBRARY_PATH=../../../../../../Utilities/PVRTrace/Record/LinuxOMAP3
```

Run the executable.

```
./OGLES2IntroducingPVRTools
```

If the file `pvrtrace.cfg` does not exist, this file will be created in the current directory and the application will exit. Edit `pvrtrace.cfg` so that the paths to `libEGL.so` and `libGLESv2.so` point to the correct host libraries for your system.

```
EglLibraryPath = /usr/lib/libEGL.so
Es2LibraryPath = /usr/lib/libGLESv2.so
```

Run the executable.

```
./OGLES2IntroducingPVRTools
```

The executable should now run correctly, recording a trace as it goes. Quit the application and the file `trace.pvrt` will be created in the current directory. You can now open this file in PVRTraceGUI.

2.7. Recording with PVRVFrame

2.7.1. Library structure

Figure 2 shows the structure of the libraries for an application using OpenGL ES 2.0 on Windows via the PVRVFrame emulation libraries. The application links against the PVRTrace versions of `libGLESv2.dll` and `libEGL.dll`. The PVRTrace libraries call functions in the PVRVFrame libraries. Finally, the PVRVFrame libraries call functions in the host OpenGL and WGL libraries.

The same structure is followed with the Linux version of PVRVFrame, but using `.so` files instead of `dll` files and the host will be GLX instead of WGL.

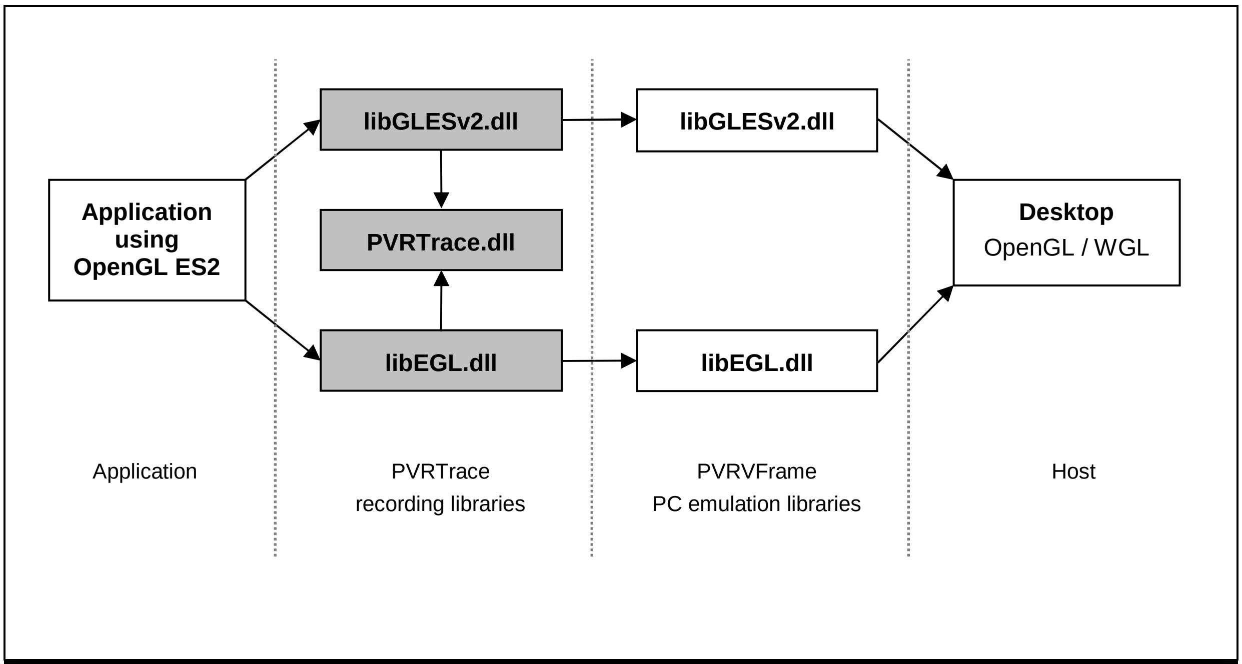


Figure 2 - PVRTrace OpenGL ES2 Windows PC emulation library overview

2.7.2. Example – Recording via PVRVFrame on Linux

The example below explains how to run the IntroducingPVRTools application with PVRTrace using the POWERVR OpenGL ES 2.0 Linux PC Emulation SDK.

Build the desired application for Linux PC.

```
cd TrainingCourse/06_IntroducingPVRTools/OGLES2/Build/LinuxGeneric
make PLATFORM=LinuxPC
```

Change directory to the location of the executable.

```
cd ../ LinuxPC/ReleaseX11
```

Setup the `LD_LIBRARY_PATH` to include the location of the PVRTrace `libEGL.so` and `libGLESv2.so` libraries for Linux PC. You may prefer to use the absolute path to this location.

```
export LD_LIBRARY_PATH=../../../../../../Utilities/PVRTrace/Record/LinuxPC
```

Run the executable.

```
./OGLES2IntroducingPVRTools
```

If the file `pvrtrace.cfg` does not exist, this file will be created in the current directory and the application will exit. Edit `pvrtrace.cfg` so that the paths to `libEGL.so` and `libGLESv2.so` point to the location of the PVRVFrame libraries.

```
EglLibraryPath = ../../../../../../Utilities/PVRVFrame/OGLES-2.0/Linux/libEGL.so
Es2LibraryPath = ../../../../../../Utilities/PVRVFrame/OGLES-2.0/Linux/libGLESv2.so
```

Run the executable.

```
./OGLES2IntroducingPVRTools
```

The executable should now run correctly, recording a trace as it goes. When you quit the application, the `trace.pvrt` file will be created in the current directory. You can now open this file in PVRTraceGUI.

2.7.3. Example – Recording via PVRVFrame on Windows

This is an example of how to run the IntroducingPVRTools application with PVRTrace from the POWERVR OpenGL ES 2.0 Windows PC Emulation SDK.

Open the Visual Studio project OGLES2IntroducingPVRTools.sln, found under

```
TrainingCourse\06_IntroducingPVRTools\OGLES2\Build\WindowsPC
```

Copy PVRTrace libraries `libEGL.dll` and `libGLESv2x.dll` to a DLL-accessible directory, or the folder where the visual studio project is.

```
Utilities\PVRTrace\Record\WindowsPC\libEGL.dll
Utilities\PVRTrace\Record\WindowsPC\libGLESv2.dll
```

Run the application from visual studio.

If the file `pvrtrace.cfg` does not exist, this file will be created in same folder as the visual studio project, i.e.

```
TrainingCourse\06_IntroducingPVRTools\OGLES2\Build\WindowsPC\pvrtrace.cfg
```

Edit `pvrtrace.cfg` so that the paths to `libEGL.dll` and `libGLESv2.dll` point to the location of the PVRVFrame libraries. You may prefer to use the absolute path to these locations.

```
EglLibraryPath = ..\..\..\..\..\..\Utilities\PVRVFrame\OGLES-2.0\Windows\libEGL.dll
Es2LibraryPath = ..\..\..\..\..\..\Utilities\PVRVFrame\OGLES-2.0\Windows\libGLESv2.dll
```

Run the application from visual studio.

The executable should now run correctly, recording a trace as it goes. Quit the application and the file `trace.pvrt` will be created in the working directory. You can now open this file in PVRTraceGUI.

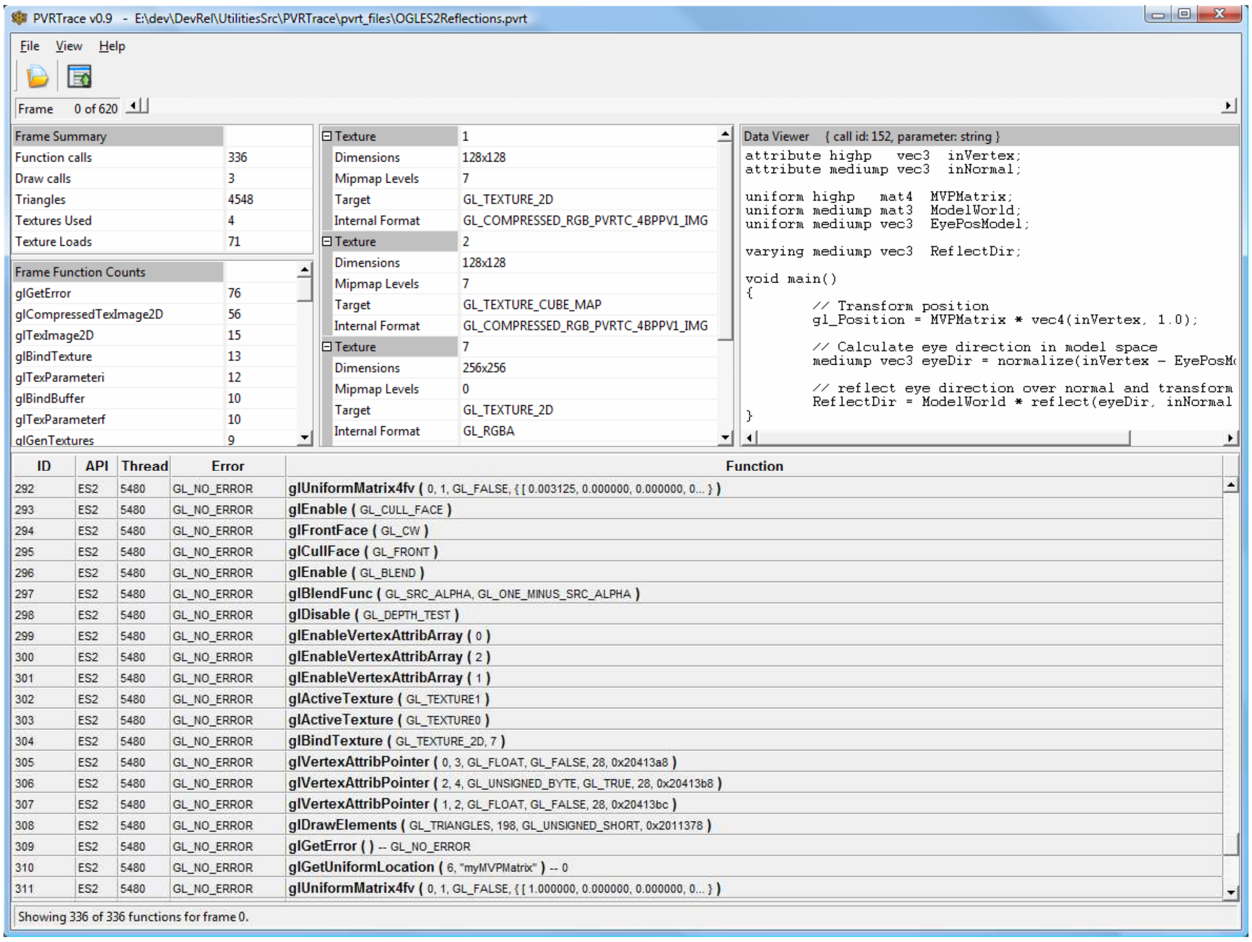
```
TrainingCourse\06_IntroducingPVRTools\OGLES2\Build\WindowsPC\trace.pvrt
```

3. PVRTraceGUI – Analysis Tool

3.1.1. Overview

PVRTraceGUI is an application that opens and displays the contents of a recorded trace file (`.pvrt`). It provides a complete list of function calls that the application made to OpenGL ES and EGL.

Figure 3 shows the main window view of PVRTraceGUI. The scroll bar across the top allows you to scroll through data for every frame that the application rendered, with the panels below displaying the list of function calls and summaries for the currently selected frame.


PVRTrace v0.9 - E:\dev\DevRel\UtilitiesSrc\PVRTrace\pvrt_files\OGLES2Reflections.pvrt
File
View
Help
Frame 0 of 620

Frame Summary	
Function calls	336
Draw calls	3
Triangles	4548
Textures Used	4
Texture Loads	71

Frame Function Counts	
glGetError	76
glCompressedTexImage2D	56
glTexImage2D	15
glBindTexture	13
glTexParameteri	12
glBindBuffer	10
glTexParameterf	10
glGenTextures	9

⊟ Texture	1
Dimensions	128x128
Mipmap Levels	7
Target	GL_TEXTURE_2D
Internal Format	GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG
⊟ Texture	2
Dimensions	128x128
Mipmap Levels	7
Target	GL_TEXTURE_CUBE_MAP
Internal Format	GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG
⊟ Texture	7
Dimensions	256x256
Mipmap Levels	0
Target	GL_TEXTURE_2D
Internal Format	GL_RGBA

Data Viewer { call id: 152, parameter: string }

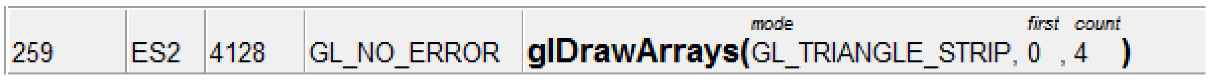
```
attribute highp   vec3  inVertex;
attribute mediump vec3  inNormal;

uniform highp   mat4  MVPMatrix;
uniform mediump mat3  ModelWorld;
uniform mediump vec3  EyePosModel;

varying mediump vec3  ReflectDir;

void main()
{
        // Transform position
        gl_Position = MVPMatrix * vec4(inVertex, 1.0);

        // Calculate eye direction in model space
        mediump vec3 eyeDir = normalize(inVertex - EyePosMo

        // reflect eye direction over normal and transform
        ReflectDir = ModelWorld * reflect(eyeDir, inNormal
}
```

ID	API	Thread	Error	Function
292	ES2	5480	GL_NO_ERROR	glUniformMatrix4fv (0, 1, GL_FALSE, { [0.003125, 0.000000, 0.000000, 0... })
293	ES2	5480	GL_NO_ERROR	glEnable (GL_CULL_FACE)
294	ES2	5480	GL_NO_ERROR	glFrontFace (GL_CW)
295	ES2	5480	GL_NO_ERROR	glCullFace (GL_FRONT)
296	ES2	5480	GL_NO_ERROR	glEnable (GL_BLEND)
297	ES2	5480	GL_NO_ERROR	glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
298	ES2	5480	GL_NO_ERROR	glDisable (GL_DEPTH_TEST)
299	ES2	5480	GL_NO_ERROR	glEnableVertexAttribArray (0)
300	ES2	5480	GL_NO_ERROR	glEnableVertexAttribArray (2)
301	ES2	5480	GL_NO_ERROR	glEnableVertexAttribArray (1)
302	ES2	5480	GL_NO_ERROR	glActiveTexture (GL_TEXTURE1)
303	ES2	5480	GL_NO_ERROR	glActiveTexture (GL_TEXTURE0)
304	ES2	5480	GL_NO_ERROR	glBindTexture (GL_TEXTURE_2D, 7)
305	ES2	5480	GL_NO_ERROR	glVertexAttribPointer (0, 3, GL_FLOAT, GL_FALSE, 28, 0x20413a8)
306	ES2	5480	GL_NO_ERROR	glVertexAttribPointer (2, 4, GL_UNSIGNED_BYTE, GL_TRUE, 28, 0x20413b8)
307	ES2	5480	GL_NO_ERROR	glVertexAttribPointer (1, 2, GL_FLOAT, GL_FALSE, 28, 0x20413bc)
308	ES2	5480	GL_NO_ERROR	glDrawElements (GL_TRIANGLES, 198, GL_UNSIGNED_SHORT, 0x2011378)
309	ES2	5480	GL_NO_ERROR	glGetError ( ) -- GL_NO_ERROR
310	ES2	5480	GL_NO_ERROR	glGetUniformLocation (6, "myMVPMatrix") -- 0
311	ES2	5480	GL_NO_ERROR	glUniformMatrix4fv (0, 1, GL_FALSE, { [1.000000, 0.000000, 0.000000, 0... })

Showing 336 of 336 functions for frame 0.

Figure 3 - PVRTraceGUI main window

3.2. Function Calls List



```
                                            mode                    first  count
259   ES2  4128  GL_NO_ERROR  glDrawArrays(GL_TRIANGLE_STRIP, 0  , 4   )
```

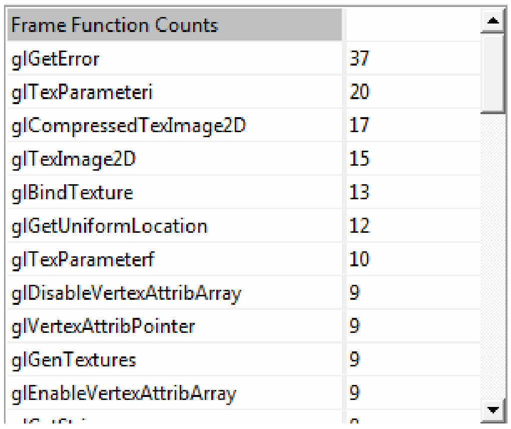
Figure 4 - Function call item

Error! Reference source not found. shows one item from the function calls list. The columns in order of left to right are explained below.

Column	Description
ID	A unique identifier for each function call throughout the whole trace.
API	The API from which the function belongs e.g. ES1, ES2, or EGL.
Thread	The thread ID that the function was called in.
Error	The error returned from calling glGetError, or eglGetError, after the function call.
Function	The function name plus the arguments it was called with. The parameter name can be optionally displayed above the corresponding arguments.

Enum values are converted into the corresponding name, hovering the mouse over the argument will display the actual value in the tool-tip caption. In this release the data behind the pointers is not yet available.

3.3. Function Counts List

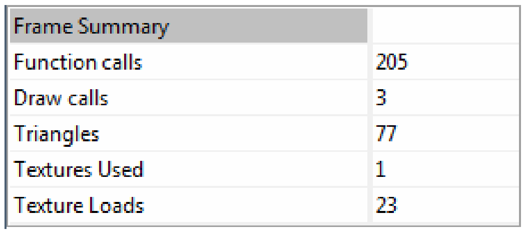


Frame Function Counts	
glGetError	37
glTexParameteri	20
glCompressedTexImage2D	17
glTexImage2D	15
glBindTexture	13
glGetUniformLocation	12
glTexParameterf	10
glDisableVertexAttribArray	9
glVertexAttribPointer	9
glGenTextures	9
glEnableVertexAttribArray	9

Figure 5 - Function Counts List

This displays a list of all the functions used in the current frame ordered by the number of times the function was used (Figure 5).

3.4. Frame Summary

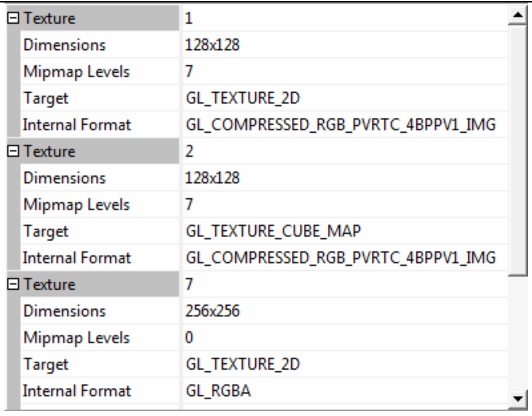


Frame Summary	
Function calls	205
Draw calls	3
Triangles	77
Textures Used	1
Texture Loads	23

Figure 6 - Frame Summary

This displays a summary of the current frame, including the number of function calls, number of draw calls, the number of triangles drawn, the number of textures bound during drawing and the number of textures uploaded.

3.5. Textures List

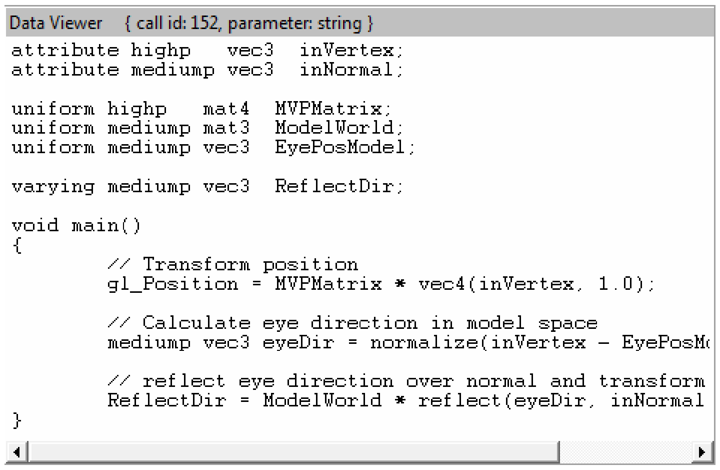


⊟ Texture	1
Dimensions	128x128
Mipmap Levels	7
Target	GL_TEXTURE_2D
Internal Format	GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG
⊟ Texture	2
Dimensions	128x128
Mipmap Levels	7
Target	GL_TEXTURE_CUBE_MAP
Internal Format	GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG
⊟ Texture	7
Dimensions	256x256
Mipmap Levels	0
Target	GL_TEXTURE_2D
Internal Format	GL_RGBA

Figure 7 - Textures List

This displays a list of textures bound during draw calls for the current frame and their properties.

3.6. Data Viewer



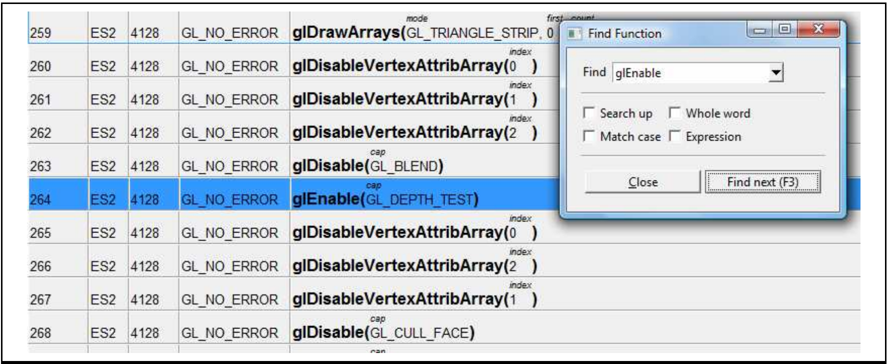
Data Viewer { call id: 152, parameter: string }

```
attribute highp   vec3  inVertex;
attribute mediump vec3  inNormal;

uniform highp   mat4  MVPMatrix;
uniform mediump mat3  ModelWorld;
uniform mediump vec3  EyePosModel;

varying mediump vec3  ReflectDir;

void main()
{
        // Transform position
        gl_Position = MVPMatrix * vec4(inVertex, 1.0);

        // Calculate eye direction in model space
        mediump vec3 eyeDir = normalize(inVertex - EyePosM

        // reflect eye direction over normal and transform
        ReflectDir = ModelWorld * reflect(eyeDir, inNormal
}
```

Figure 8 - Data Viewer

This displays the full data for the most recently clicked item in the calls list. This is particularly useful for viewing strings (such as shader source code), textures and arrays. Clicking the link in the title finds the corresponding function call in the calls list.

3.7. Find Function Dialog



259	ES2	4128	GL_NO_ERROR	glDrawArrays(GL_TRIANGLE_STRIP, 0
260	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(0)
261	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(1)
262	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(2)
263	ES2	4128	GL_NO_ERROR	glDisable(GL_BLEND)
264	ES2	4128	GL_NO_ERROR	glEnable(GL_DEPTH_TEST)
265	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(0)
266	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(2)
267	ES2	4128	GL_NO_ERROR	glDisableVertexAttribArray(1)
268	ES2	4128	GL_NO_ERROR	glDisable(GL_CULL_FACE)

Find Function

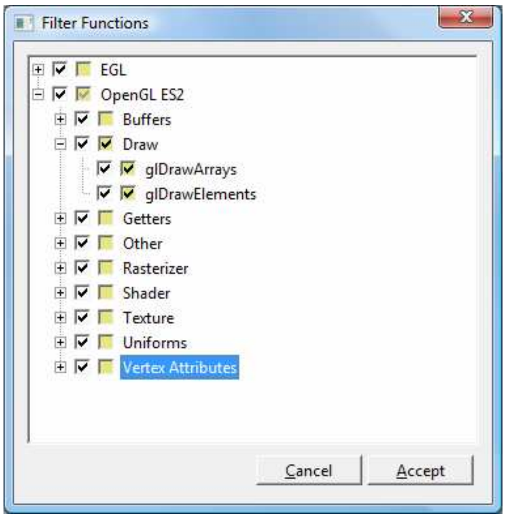
Find glEnable

Search up ☐ Whole word ☐ Match case ☐ Expression ☐

Figure 9 - Find Function Dialog

This dialog is used to search through the function calls list to find the next occurrence of a particular function name in the current frame.

3.8. Filtering Functions



Filter Functions

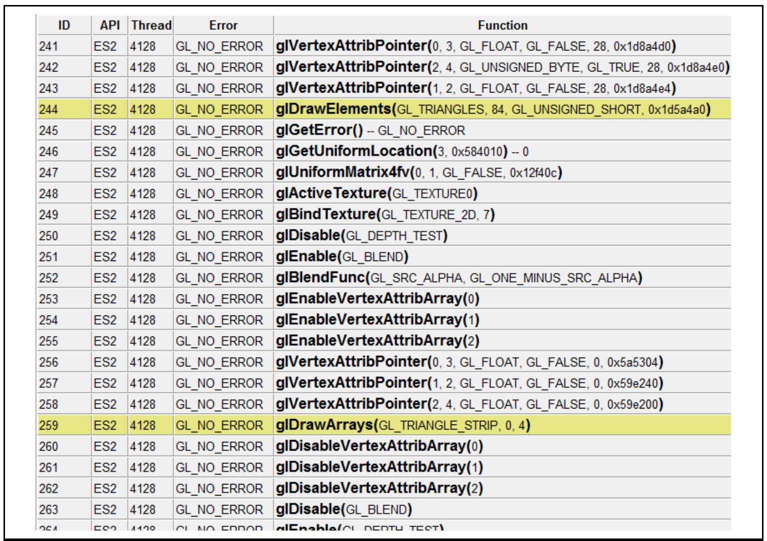
- ⊞ ✓ ☐ EGL
- ⊟ ✓ ✓ OpenGL ES2
  - ⊞ ✓ ☐ Buffers
  - ⊟ ✓ ✓ Draw
    - ✓ ✓ glDrawArrays
    - ✓ ✓ glDrawElements
  - ⊞ ✓ ☐ Getters
  - ⊞ ✓ ☐ Other
  - ⊞ ✓ ☐ Rasterizer
  - ⊞ ✓ ☐ Shader
  - ⊞ ✓ ☐ Texture
  - ⊞ ✓ ☐ Uniforms
  - ⊞ ✓ ☐ Vertex Attributes

Figure 10 - Filter Functions Dialog

Functions in the function calls list (shown in figure 4) can be hidden or highlighted as required. In the filter functions dialog window (shown in figure 10) the white box is ticked to display all occurrences of the function. The yellow box is ticked to highlight all occurrences of the function.

The hide and highlight options can also be accessed via right-clicking on the item in the function calls list.

An example of how this might be useful is to hide all calls to glGetError, or to highlight all the draw calls, allowing you to more clearly see the setup before each object is drawn.



ID	API	Thread	Error	Function
241	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**0, 3, GL_FLOAT, GL_FALSE, 28, 0x1d8a4d0**)**
242	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**2, 4, GL_UNSIGNED_BYTE, GL_TRUE, 28, 0x1d8a4e0**)**
243	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**1, 2, GL_FLOAT, GL_FALSE, 28, 0x1d8a4e4**)**
244	ES2	4128	GL_NO_ERROR	**glDrawElements(**GL_TRIANGLES, 84, GL_UNSIGNED_SHORT, 0x1d5a4a0**)**
245	ES2	4128	GL_NO_ERROR	**glGetError()** -- GL_NO_ERROR
246	ES2	4128	GL_NO_ERROR	**glGetUniformLocation(**3, 0x584010**)** -- 0
247	ES2	4128	GL_NO_ERROR	**glUniformMatrix4fv(**0, 1, GL_FALSE, 0x12f40c**)**
248	ES2	4128	GL_NO_ERROR	**glActiveTexture(**GL_TEXTURE0**)**
249	ES2	4128	GL_NO_ERROR	**glBindTexture(**GL_TEXTURE_2D, 7**)**
250	ES2	4128	GL_NO_ERROR	**glDisable(**GL_DEPTH_TEST**)**
251	ES2	4128	GL_NO_ERROR	**glEnable(**GL_BLEND**)**
252	ES2	4128	GL_NO_ERROR	**glBlendFunc(**GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA**)**
253	ES2	4128	GL_NO_ERROR	**glEnableVertexAttribArray(**0**)**
254	ES2	4128	GL_NO_ERROR	**glEnableVertexAttribArray(**1**)**
255	ES2	4128	GL_NO_ERROR	**glEnableVertexAttribArray(**2**)**
256	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**0, 3, GL_FLOAT, GL_FALSE, 0, 0x5a5304**)**
257	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**1, 2, GL_FLOAT, GL_FALSE, 0, 0x59e240**)**
258	ES2	4128	GL_NO_ERROR	**glVertexAttribPointer(**2, 4, GL_FLOAT, GL_FALSE, 0, 0x59e200**)**
259	ES2	4128	GL_NO_ERROR	**glDrawArrays(**GL_TRIANGLE_STRIP, 0, 4**)**
260	ES2	4128	GL_NO_ERROR	**glDisableVertexAttribArray(**0**)**
261	ES2	4128	GL_NO_ERROR	**glDisableVertexAttribArray(**1**)**
262	ES2	4128	GL_NO_ERROR	**glDisableVertexAttribArray(**2**)**
263	ES2	4128	GL_NO_ERROR	**glDisable(**GL_BLEND**)**
264	ES2	4128	GL_NO_ERROR	**glEnable(**GL_DEPTH_TEST**)**

Figure 11 - Highlight Draw Calls

Figure 11 shows a list of draws calls with the glDraw calls highlighted.