Contents

1. Introduction ................................................................................................................. 5
   1.1. Document Overview ......................................................................................... 5
   1.2. Software Overview ......................................................................................... 5

2. Recording Libraries ...................................................................................................... 6
   2.1. Overview ........................................................................................................... 6
   2.2. Compatibility ..................................................................................................... 6
   2.3. Manual Installation ............................................................................................ 6
       2.3.1. Windows ..................................................................................................... 6
       2.3.2. Linux .......................................................................................................... 6
       2.3.3. Android ...................................................................................................... 7
   2.4. Configuration ...................................................................................................... 7
       2.4.1. Enabled Option ........................................................................................... 7
       2.4.2. Tracing Options .......................................................................................... 7
       2.4.3. Debug Options ............................................................................................ 9
       2.4.4. Network Options ....................................................................................... 10
       2.4.5. Host Options ............................................................................................. 10
       2.4.6. Profiling Options ....................................................................................... 11
   2.5. Recording with PVRTrace ................................................................................... 12
       2.5.1. On-Device Recording ............................................................................... 12
       2.5.2. Remote Network Recording ...................................................................... 13
       2.5.3. Recording on Android ............................................................................... 13
       2.5.4. Setting Global and Per-Process Flags ....................................................... 13

3. PVRTrace GUI ................................................................................................................ 15
   3.1. The Basics .......................................................................................................... 15
       3.1.1. Overview .................................................................................................... 15
       3.1.2. User Interface Layout ............................................................................... 15
   3.2. Menu Bar ............................................................................................................ 16
       3.2.1. File Menu ................................................................................................ 16
       3.2.2. Tools Menu ................................................................................................ 17
       3.2.3. View Menu ................................................................................................ 20
       3.2.4. Help Menu ................................................................................................ 21
   3.3. Working with the Function Call List ..................................................................... 22
       3.3.1. View Calls by Frame ............................................................................... 23
       3.3.2. Expand or Collapse Draw Calls ................................................................ 23
       3.3.3. Filter Options ........................................................................................... 23
       3.3.4. Highlight Options ..................................................................................... 24
       3.3.5. Search Options .......................................................................................... 25
       3.3.6. Disable or Enable Calls ............................................................................ 27
       3.3.7. Function Call Documentation .................................................................... 27
   3.4. Image Analysis ...................................................................................................... 28
       3.4.1. Select a Render Output Mode .................................................................... 29
       3.4.2. View Image Output by Draw Call Type .................................................... 30
       3.4.3. Other Actions ............................................................................................. 30
   3.5. Statistical Analysis ............................................................................................... 30
       3.5.1. Call Statistics .............................................................................................. 30
       3.5.2. Frame Statistics .......................................................................................... 31
       3.5.3. Graphical Representation of Statistics ....................................................... 31
       3.5.4. Other Forms of Statistical Analysis ............................................................ 33
   3.6. Static Analysis ...................................................................................................... 33
   3.7. Inspection of Draw Calls ...................................................................................... 34
   3.8. Current Call Inspection ........................................................................................ 35
       3.8.1. EGL State .................................................................................................. 35
       3.8.2. EGL Objects .............................................................................................. 36
       3.8.3. OpenGL ES State ....................................................................................... 36
       3.8.4. OpenGL ES Objects .................................................................................. 37
List of Figures

Figure 1. Overview of the Recording Libraries ......................................................... 6
Figure 2. General layout of the PVRTrace GUI .......................................................... 16
Figure 3. File menu ................................................................................................. 16
Figure 4. Tools menu .............................................................................................. 17
Figure 5. Remote Controller dialog box ................................................................. 18
Figure 6. Preferences dialog box .............................................................................. 20
Figure 7. View menu ............................................................................................. 20
Figure 8. Help menu ............................................................................................. 21
Figure 9. Function call list area .............................................................................. 22
Figure 10. Action menu displayed on right-clicking a function call ...................... 22
Figure 11. Selecting a frame ................................................................................. 23
Figure 12. Advanced filtering using the Filter/Highlight Functions dialog box ...... 24
Figure 13. Advanced highlighting using the Filter/Highlight Functions dialog box .. 25
Figure 14. Find dialog box .................................................................................... 26
Figure 15. Viewing search results ....................................................................... 26
Figure 16. Viewing disabled calls in the Modifications tab .................................. 27
Figure 17. Image Analysis window ..................................................................... 29
Figure 18. Viewing API call statistics .................................................................. 30
Figure 19. Viewing frame statistics ..................................................................... 31
Figure 20. Statistics Graph dialog box displaying per-frame statistics .............. 32
Figure 21. Statistics Graph dialog box displaying thread usage .......................... 32
Figure 22. Viewing the results of static analysis ............................................... 33
Figure 23. Viewing suggestions for resolving an issue ....................................... 34
Figure 24. Viewing the Draws Calls window ..................................................... 34
Figure 25. Current Call window.................................................................35
Figure 26. Viewing EGL state objects in the Object Data Viewer window........36
Figure 27. EGL Objects tab in the Current Call window.................................36
Figure 28. OpenGL ES State tab in the Current Call window..........................37
Figure 29. Viewing OpenGL ES state variables in the Object Data Viewer window 37
Figure 30. OpenGL ES Objects tab in the Current Call window.......................38
Figure 31. Inspecting textures......................................................................38
Figure 32. Viewing loaded shaders.................................................................39
Figure 33. Inspecting a shader........................................................................40
Figure 34. Viewing modified shaders in the Modifications window..................41
Figure 35. Selecting a pixel in the Image Analysis window..............................41
Figure 36. Viewing the Shader Analysis tab....................................................42
Figure 37. Viewing the Log window................................................................43

List of Tables
Table 1. Tracing options....................................................................................8
Table 2. Debug options......................................................................................9
Table 3. Network options.................................................................................10
Table 4. Host options......................................................................................10
Table 5. Profiling options................................................................................11
Table 6. Renderstate Overrides.........................................................................11
Table 7. Columns displayed in the Draw Calls window....................................35
Table 8. General command-line options..........................................................45
Table 9. Advanced command-line options......................................................46
Table 10. Regular expression syntax and description........................................56
1. Introduction

1.1. Document Overview
The purpose of this document is to serve as a complete user manual for the PVRTrace analysis tool and its associated libraries. It includes compatibility information, installation instructions, a guide to the functionality of the application and a complete listing of all interface options and preferences.

1.2. Software Overview
PVRTrace is a recording and analysis utility, which captures all the API calls made by an OpenGL ES application. It consists of two main components:

- **Recording Libraries**: These are shim libraries that capture API calls by sitting between an OpenGL ES application and the platform's native graphics libraries.
- **PVRTrace GUI**: This serves as the analysis interface of PVRTrace allowing user-friendly access to the contents of pre-recorded .pvrtrace files.
2. Recording Libraries

2.1. Overview

The recording of a trace is performed by multiple libraries, notably shim libraries for each API and one core recording library (Figure 1). Once on a platform, the shim libraries intercept graphics API calls and sends them to the recording library to write them to a file. The calls are then passed on to the host libraries, as defined in the PVRTrace configuration file (see Section 2.4). Calls are streamed to the .pvrtrace file during runtime so that even if an application crashes, data is recorded.

![Figure 1. Overview of the Recording Libraries](image)

Note: This diagram changes when using an Android target device without root permission. For more information on using a device without root permission, see the “PVRTrace Quick Start Guide for Android Unrooted”.

2.2. Compatibility

The following compatibility information is relevant to the PVRTrace Recording Libraries:

- **API compatibility**: PVRTrace Recording Libraries are compatible with EGL, OpenGL ES 1.1, OpenGL ES 2.0, OpenGL ES 3.0, and OpenGL ES 3.1.
- **Supported extensions**: PVRTrace only supports extensions supported by PowerVR devices.

2.3. Manual Installation

2.3.1. Windows

For manual installation on Windows, perform the following steps:

1. **Copy**: libEGL.dll, libGLESv2.dll (OpenGL ES 2.0 and 3.0), libGLES_CM.dll, libGLESv1_CM.dll (OpenGL ES 1.1) and PVRTrace.dll to the executable folder.
2. If a PVRTrace configuration file does not exist, make one manually (see Section 2.4), and then place pvrtraceconfig.json into the executable folder.
3. **Update** pvrtraceconfig.json so that EglLibraryPath, Es1LibraryPath and Es2LibraryPath are set to the location of the system’s graphics libraries (see Section 2.4).

2.3.2. Linux

For manual installation on Linux, perform the following steps:
1. **Copy:** `libEGL.so`, `libGLESv2.so` (OpenGL ES 2.0, 3.0 and 3.1), `libGLES_CM.so`, `libGLESv1_CM.so` (OpenGL ES 1.1) and `libPVRTrace.so` to any folder.

2. Set the `LD_LIBRARY_PATH` to the folder containing the new libraries. This can be set using `export LD_LIBRARY_PATH=...` and reverted once the trace is complete. Alternatively, this can be set by using `LD_LIBRARY_PATH=./<APPLICATION_NAME>` as part of running the trace.

3. If `pvrtraceconfig.json` does not exist, make one manually (see Section 2.4), and then place `pvrtraceconfig.json` into the executable folder.

4. **Update** `pvrtraceconfig.json` so that `EglLibraryPath`, `Es1LibraryPath` and `Es2LibraryPath` are set to the location of the system’s graphics libraries (see Section 2.4).

*Note: In addition to the manual approach, installation on Linux can also be made using PVRHub. For more information on using PVRHub on Linux, see the “PVRHub User Manual”.*

### 2.3.3. Android

Installation on Android can be done using PVRHub, which is the preferred method of controlling the Android Recording Libraries. The PVRHub APK can be found in the folder: `<InstallDir>/PVRHub/`. Once this APK has been installed on a rooted device it can be used to control and setup the Recording Libraries with no manual editing of the PVRTrace config file or file system. For more information on PVRHub, see the “PVRHub User Manual”.

*Note: PVRTrace can also be manually installed on Android. For further details, see Appendix A.*

### 2.4. Configuration

The PVRTrace Recording Libraries require a configuration file named `pvrtraceconfig.json`. On non-Android systems, this file should be located in the working directory of the application you are tracing. However, for Android systems, the file should be located in `/ or /sdcard/`. If the file does not exist, create it and populate it with the required options.

If PVRHub is being used, it will generate its own copy of `pvrtraceconfig.json` in `/data/data/com.powervr.PVRHub/`. The modification of this file is done through PVRHub’s interface. Do not modify PVRHub’s `pvrtraceconfig.json` by hand unless modifying options that are not configurable through its interface.

The following sections cover the various options available in `pvrtraceconfig.json`.

*Note: If working on an Android device, please refer to the PVRTrace Quick Start Guide for Android Rooted/Unrooted depending on the specifics of your device.*

*Note: For illustration purposes, an example of a complete configuration file is provided in Appendix B.*

#### 2.4.1. Enabled Option

The configuration option `Enabled` exists at the root of the process sections. It is a Boolean value, set to `true` as default. When set as `true`, recording is enabled. When set as `false`, PVRTrace passes the calls onto the host and performs no actions.

#### 2.4.2. Tracing Options

Table 1 lists tracing options and their description.
### Table 1. Tracing options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutputFilename</strong></td>
<td>Type: String. Default: <code>%pname.pvrt</code>. This sets the name and location of the trace file that is output by the Recording Libraries. The process ID of an application can be used as part of its trace file name by using <code>%pid</code> in this field. Likewise, the application name can be employed by using <code>%pname</code>. It should be noted that the output file name must be set to a writeable location or no trace file will be output.</td>
<td>%pname.pvrt</td>
</tr>
<tr>
<td><strong>RecordData</strong></td>
<td>Type: Boolean. Default: true. This option states whether the trace libraries should record the data associated with each call or not. With this option set to true, texture information and buffer data, etc., are recorded along with the graphics API calls. With <code>RecordData</code> set to false only the graphics API calls are recorded. In general, this option should be set to true unless file size is a problem.</td>
<td>true</td>
</tr>
<tr>
<td><strong>RecordUnknownExtensionCalls</strong></td>
<td>Type: Boolean. Default: true. If enabled, attempts to record the function name for calls to extension functions that are currently unknown to PVRTrace.</td>
<td>true</td>
</tr>
<tr>
<td><strong>StartFrame</strong></td>
<td>Type: Integer. Default: 0. This states the frame at which recording should begin.</td>
<td>0</td>
</tr>
<tr>
<td><strong>EndFrame</strong></td>
<td>Type: Integer. Default: 0. This states the last frame to be recorded.</td>
<td>0</td>
</tr>
<tr>
<td><strong>ExitOnLastFrame</strong></td>
<td>Type: Boolean. Default: false. If set to true, the application will quit after the last frame has been recorded.</td>
<td>false</td>
</tr>
<tr>
<td><strong>ClientBufferRecordFrequency</strong></td>
<td>Type: Integer. Default: 0. This option is used to control when an EGLClientBuffer, which is the backing store of an EGLImage, is sampled for recording. 0 means that client buffers will not be recorded. 1 means that the client buffer will be sampled once each frame and 2 means that it will be sampled after each draw call. 3 is used to only sample the client buffer once for recording at the time the EGLImage is created. It is important to remember that the more samples PVRTrace performs the larger the output file will be and also may affect the application's performance.</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SaveFrameBuffer</td>
<td>Type: Boolean. Default: false. If this option is set to true, the application will save the framebuffer each frame, effectively taking a screenshot of the application running on the device. It is important to remember that this option will significantly increase the file size. <em>Note: When saving a trace file from the GUI the framebuffer will also be saved.</em></td>
<td></td>
</tr>
<tr>
<td>UseCompression</td>
<td>Type: Boolean. Default: false. Enables compression of the trace data to reduce output file size.</td>
<td></td>
</tr>
<tr>
<td>AppendTraceVersion</td>
<td>Type: Boolean. Default: true. Adds the PVRTrace version to the results of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- eglQueryString(display, EGL_VERSION)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- glGetString(GL_RENDERER)</td>
<td></td>
</tr>
<tr>
<td>DisableBinaries</td>
<td>Disables calls involving shader binaries, such as glShaderBinary, glProgramBinary and related glGet* calls.</td>
<td></td>
</tr>
</tbody>
</table>

### 2.4.3. Debug Options
Table 2 lists debug options and their description.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Type: Integer. Default: 0. This option will set the level of verbosity of the debug output. 0 means disabled, resulting in no output. 1 means only the most important data is displayed and 2 means all data is displayed.</td>
</tr>
</tbody>
</table>
2.4.4. Network Options
Table 3 lists network options and their description.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Type: Boolean. Default: false. With this flag set to true the PVRTrace GUI is able to connect to the application over a network allowing for remote recording. It should be noted that while this flag is set to true the StartFrame and EndFrame flags are ignored. With this flag set to false on-device recording is performed and the Network\Wait and Network\BufferSize flags are ignored.</td>
</tr>
<tr>
<td>Wait</td>
<td>Type: Boolean. Default: false. With this flag set to true the application being recorded pauses during the first API call until an instance of the PVRTrace GUI has connected and sent the command to continue playing. With this flag set to false the application being recorded continues to play regardless of whether a client has connected or not.</td>
</tr>
<tr>
<td>BufferSize</td>
<td>Type: Integer. Default: 256. Network\BufferSize sets the volume of data that is sent in each packet from the Recording Libraries to the PVRTrace GUI.</td>
</tr>
</tbody>
</table>

2.4.5. Host Options
Table 4 lists host options and their description.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EglLibraryPath</td>
<td>Type: String. Default: Null. EglLibraryPath refers to the location of the original EGL driver on the host system. This is usually found in /system/lib/egl or /system/vendor/lib/egl if running an Android device. On Linux, the path will usually be /usr/lib. On Android the Recording Libraries will automatically attempt to populate this with the system’s graphics libraries.</td>
</tr>
<tr>
<td>Es1LibraryPath</td>
<td>Type: String. Default: Null. Es1LibraryPath refers to the location of the original OpenGL ES 1.1 driver on the host system. This is usually found in /system/lib/egl or /system/vendor/lib/egl if running an Android device. On Linux, the path will usually be /usr/lib. On Android the Recording Libraries will automatically attempt to populate this with the system’s graphics libraries.</td>
</tr>
</tbody>
</table>
Option | Description
--- | ---
Es2LibraryPath | Type: String. Default: Null. `Es2LibraryPath` refers to the location of the original OpenGL ES 2.0/3.0/3.1 driver on the host system. This is usually found in `/system/lib/egl` or `/system/vendor/lib/egl` if running an Android device. On Linux, the path will usually be `/usr/lib`. On Android the Recording Libraries will automatically attempt to populate this with the system’s graphics libraries.

2.4.6. Profiling Options
Table 5 lists profiling options and their description.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Type: Boolean. Default: false. If set to <code>true</code>, this sets the Recording Libraries to profiling mode, disabling recording. When used in conjunction with PVRPerfServer and PVRTune this enables PVRTune’s Renderstate override capabilities, software statistics and driver timing data.</td>
</tr>
<tr>
<td>SoftwareCounters</td>
<td>Type: Boolean. Default: true. If this option is set to <code>true</code>, it enables API counters to be displayed in PVRTune.</td>
</tr>
<tr>
<td>FunctionTimelineLevel</td>
<td>Type: Integer. Default: 1. This option will specify the amount of calls that will be displayed in the PVRTune timeline. If set to 0 no calls will be displayed. If set to 1 then a selection of calls will be displayed and if set to 2, all calls will be displayed. It is important to remember that using the last option may slow down the application.</td>
</tr>
<tr>
<td>RenderstateOverride</td>
<td>Type: Boolean. Default: true. If set to <code>true</code>, this will enable PVRTune capabilities to toggle different selections of renderstates. Refer to the “PVRTune User Manual” for further details.</td>
</tr>
</tbody>
</table>

In addition to profiling options there is also an Overrides subsection which contains the options listed in Table 6.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForceViewport</td>
<td>Type: Boolean. Default: false. Modifies the viewport to have zero sized dimensions.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Force2x2Textures</td>
<td>Type: Boolean. Default: false. Forces textures to be 2 pixels by 2 pixels in size.</td>
</tr>
<tr>
<td>ForceFlatColourFrag</td>
<td>Type: Boolean. Default: false. Forces the fragment shader to output a single colour.</td>
</tr>
<tr>
<td>DisableStencilTest</td>
<td>Type: Boolean. Default: false. Disables the stencil test.</td>
</tr>
<tr>
<td>DisableDepthTest</td>
<td>Type: Boolean. Default: false. Disables the depth test.</td>
</tr>
<tr>
<td>DisableScissorTest</td>
<td>Type: Boolean. Default: false. Disables the scissor test.</td>
</tr>
<tr>
<td>DisableFlushNFinish</td>
<td>Type: Boolean. Default: false. Disables all glFlush() and glFinish() function calls.</td>
</tr>
<tr>
<td>DisableTextureModifications</td>
<td>Type: Boolean. Default: false. Disables glTexImage2D() and glTexImage2D(). This disables the updating of texture sub-regions.</td>
</tr>
<tr>
<td>DisableTextureFiltering</td>
<td>Type: Boolean. Default: false. Disables texture filters. All texture filters are set to the nearest point.</td>
</tr>
<tr>
<td>DisableAlphaTest</td>
<td>Type: Boolean. Default: false. Disables the alpha test.</td>
</tr>
<tr>
<td>CullingMode</td>
<td>Type: Integer. Default: 0. Forces the culling mode to perform culling as defined in the user application. The other values are: 1, which ensures no culling takes place. 2, which forces back culling. 3, which forces front culling.</td>
</tr>
</tbody>
</table>

### 2.5. Recording with PVRTrace

#### 2.5.1. On-Device Recording

The following steps highlight the procedure for on-device recording with PVRTrace:

1. Install the PVRTrace libraries on the target device, as per the installation instructions (see Section 2.3).
2. Ensure that pvrtraceconfig.json is set correctly (see Section 2.4) and that the application has permission to write to the output location set under Tracing\OutputFilename in pvrtraceconfig.json.
3. Ensure that pvrtraceconfig.json has Network\Enabled set to false.
4. Run the application to be traced.

2.5.2. Remote Network Recording
The following steps highlight the procedure for local recording with the PVRTrace GUI:

1. Install the recording libraries on the target device, as per the installation instructions (see Section 2.3).
2. Ensure that pvrtraceconfig.json is set correctly for network recording (see Section 2.4).
3. If required, set Network\Wait to True. This allows the initial setup calls of the application to be remotely recorded, and the app will start playing upon connection.
4. Run the PVRTrace GUI and launch the Remote Controller by clicking Tools -> Remote Controller... (see Section “Remote Controller”).
5. Run through the steps identified in Section “Remote Controller” for recording and opening a trace file.

2.5.3. Recording on Android
Recording on Android can be performed using PVRHub. For more information, refer to the “PVRHub User Manual”. The user manual can also be accessed from the PVRHub application.

2.5.4. Setting Global and Per-Process Flags
Below is an example of a configuration file for PVRTrace:

```json
{
  "*": {
    "Enabled": false,
    "Tracing": {
      "OutputFilename": "/sdcard/%pname.pvrtrace",
      "RecordData": true,
      "StartFrame": 0,
      "EndFrame": 100,
      "ExitOnLastFrame": true,
    },
    "Host": {
      "Es2LibraryPath": "/system/lib/libGLESv2.so",
      "Es1LibraryPath": "/system/lib/libGLESv1_CM.so",
      "EglLibraryPath": "/system/lib/libEGL.so"
    }
  },
  "com.powervr.*": {
    "Enabled": true,
    "Tracing": {
      "UseCompression": true
    }
  },
  "com.powervr.OGLES2Water": {
    "Tracing": {
      "StartFrame": 100,
      "EndFrame": 200,
      "SaveFrameBuffer": true
    }
  }
}
```

In the above example, the first section ("") specifies global options that apply to all processes. This is followed by the “com.powervr.*” section that lists options that override the global options for processes starting with “com(powervr.” Finally, the “com.powervr.OGLES2Water” section applies to that specific process, overriding all previous options.

The order of the sections in the config file does not matter. The precedence is based purely on the length of the section name, so that fully qualified process names such as “com.powervr.OGLES2Water” take priority over both “com.powervr.*” and ".".
Note: .pvrtraceconfig files for previous PowerVR Tools and SDK versions (e.g. SDK 3.4) are not compatible with PVRTrace from SDK 3.5 onwards. To update your previous configs, you will need to add a top-level process flag to be specified (e.g. the '*' all processes wildcard) around the rest of the config.
3. PVRTrace GUI

3.1. The Basics

3.1.1. Overview

The PVRTrace GUI is the graphical user interface of PVRTrace. It opens .pvrtrace files created with the PVRTrace Recording Libraries and displays them in a user-friendly format. A wealth of information is displayed in the user interface and that information is broken down frame by frame, including details such as the number of times a given call occurs in a trace to the exact values of a specific matrix. PVRTrace currently supports the following APIs:

- EGL.
- OpenGL ES 1.1.
- OpenGL ES 2.0, 3.0 and 3.1.

3.1.2. User Interface Layout

Figure 2 illustrates the default layout of the PVRTrace GUI. The interface consists of the following sections:

- **Menu bar** (Figure 2a): This section enables access to several options related to file, tools, view and help.
- **Function call list area** (Figure 2b): This section lists all function calls that have been recorded.
- **Image Analysis window** (Figure 2c): This section allows the user to interact with several visualisation modes for a render output.
- The section identified in Figure 2d provides multiple windows organised as tabs for performing static analysis, viewing statistics about function calls and frame information, viewing the standard log output from the PVRTrace GUI, and viewing search results.
- **Draw Calls window** (Figure 2e): This section lists all draw calls for the current frame.
- **Current Call window** (Figure 2f): This section allows viewing information regarding EGL state, EGL objects, OpenGL ES state and OpenGL ES objects on the currently selected call.
- The section identified in Figure 2g provides multiple windows organised as tabs for viewing object data, performing shader analysis, viewing modifications, and recorded framebuffer information.
3.2. Menu Bar

3.2.1. File Menu

Figure 3 illustrates the File menu.

![File menu](image)

**Open a File**

To open a `.pvrtrace` file, click **File -> Open** (Figure 3). This will open a dialog box for browsing to the required file.

**Open a Recent File**

To open a recently accessed file, click **File -> Open Recent File** (Figure 3). This will display a list of the recently accessed files from which the desired file can then be selected for opening.

**Clear Recent Files**

To clear the list of recently accessed files, click **File -> Open Recent File** (Figure 3). An option called **Clear recent files** will become available to use.
Export Data

Exporting data is essential for saving information in formats that can then be interpreted and used by external applications for subsequent analysis or for performing other tasks outside of the PVRTrace environment. By clicking File -> Export Data (Figure 3), it becomes possible to achieve the following:

- To export the contents in the Function call list area for the current frame to a TXT file, select the option called Current Frame (.txt).
- To export the contents in the Function call list area for all frames, broken down frame by frame, to a TXT file, select the option called All Frames (.txt).
- To export shaders as VSH and FSH files, select the option called Shaders.
- To export saved framebuffers as TGA file, select the option called Saved Framebuffers (.tga).
- To export the contents of the Frame Summary tab in the Statistics window (see Section 3.5.2) for all frames to a CSV file, select the option called Frame Statistics (.csv).
- To export the contents of the Calls Summary tab of the Statistics window (see Section 3.5.1) for all frames to a CSV file, select the option called Call Statistics (.csv).

Save .pvrtrace File
To save a trace to the .pvrtrace file format, click File -> Save As… (Figure 3). This will open a dialog box which allows all or a set number of user-specified frames to be saved.

View Trace Information
To display a listing of trace information about the loaded trace file, click File -> Trace Info (Figure 3). This will open a dialog box containing the information, which includes details such as the trace version, file version, and platform.

Exit PVRTrace
To close the PVRTrace GUI, click File -> Exit.

3.2.2. Tools Menu

Figure 4 illustrates the Tools menu. The menu provides options for using the Remote Controller, Statistics Graph and also enables preferences to be set.

Remote Controller
The Remote Controller dialog box (Figure 5) is used to connect to an application whose Recording Libraries are set up for remote recording. To open the dialog box, click Tools -> Remote Controller…
To use the **Remote Controller** dialog box for recording purposes:

1. In the **Remote Controller** dialog box (Figure 5), enter the IP address of the target device in the **Connect to** box. The box also has a dropdown menu which lists recently used IP connections.
2. Click the **Go** button to establish the connection.
3. Run the application to be traced. If the application is already running and is waiting for the network, connecting will start the application.
4. Click the **Start Recording** button to start recording the trace. Once the recording has started, the button will then display the **Stop Recording** option. Notice that the **Start recording on connection** checkbox should be ticked if it is desired to start recording from frame 0 when the application starts running after the connection is established, although a later frame to start from may be picked, if necessary.
5. To stop recording the trace, click the **Stop Recording** button.
6. Once the trace has been recorded, it becomes possible to open and save the recording by clicking the **Download** button. Notice that if the **Open trace file after download** checkbox is ticked, then this will open the trace in the PVRTrace GUI immediately after the download.

**Statistics Graph**

When a `.pvrtrace` file has been loaded, the **Statistics Graph** dialog box can be used to visualize software counters in the traced application on a per-frame basis as well as thread activity on a per-frame basis in multi-threaded applications, if applicable. To open the dialog box, click **Tools -> Statistics Graph**. For more information on the **Statistics Graph**, see Section 3.5.

**Preferences**

User preferences are set by clicking **Tools -> Preferences** in the **Menu** bar. This opens the **Preferences** dialog box (Figure 6) which displays various options to enable the customisation of the PVRTrace GUI. The **Preferences** dialog box can be used to achieve the following:

- **Max. number of returned search results**: This option is available under the **General** section of the **Preferences** dialog box (Figure 6). To specify the maximum number of hits returned during a search activity, value in the provided field should be altered.
- **Frame delimiter**: This specifies the call(s) to use for splitting the trace file into frames.
- **Image Analysis**: This option should be used when there are issues with image analysis or resolving the frames is taking too long and the user is only interested in the function calls. Deselecting the checkbox prompts for reloading the `.pvrtrace` file, after which the image analysis functionality is disabled.
- **Allow scrubber to update Current Call**: Below the render is a slider. This is often referred to as the scrubber. With the **Allow scrubber to update Current Call** option ticked
moving the slider will change the current call and the call list widgets, showing up to the selected call.

- **Normalize Depth and Stencil data:** On the image analysis there is a drop down that allows you to specify colour, depth, stencil, wireframe, depth complexity etc. If you select depth or stencil you’ll be able to see the depth and stencil information of the render. OpenGL automatically scales this data so it is between 0.0 and 1.0 but often this results in all the data being bunched up so you end up seeing a flat coloured render. The Normalize Depth and Stencil data option when enabled just scales the data so it uses the whole of the 0.0 to 1.0 range.

- **Use rendered frame caching:** Enables/Disables frame caching.

- **Analyse Fragments based on scrubber instead of whole frame:** By default Fragment Analysis only analyses the frame as a whole. With this option ticked, analysis will be performed up to the scrubber position. This allows for partial frame analysis but will have a performance cost when scrubbing.

- **Play calls that failed during recording when re-evaluating:** The option, when ticked, allows replaying failed calls that occurred during the recording of the trace file (see Section 3.4).

- **Single Primitive Clear Colour:** The clear colour used when image analysis is set to single primitive mode.

- **Crosshair Colour:** This option allows a colour to be specified for the crosshair displayed in the Image Analysis window, if Fragment Analysis is enabled.

- **Wireframe Effect Colour:** The option enables a colour to be specified for the wireframe effects displayed in the Image Analysis window (see Section 3.4). To specify the colour, click the corresponding button.

- **Primitive Highlight:** The option, when ticked, will highlight the currently selected primitive in the Image Analysis window (see Section 3.4), drawing it as an overlay wireframe with the selected colour.

- **Primary Colour:** The colour that the primitive highlight is displayed.

- **Secondary Colour:** This is the other colour that the primitive highlight is displayed in.

- **Compiler:** A dropdown menu enables a PowerVR compiler to be chosen to verify and profile shaders. The choice of compiler should reflect the platform that is being targeted.

- **Custom Compiler Path:** The option is enabled when the Custom Compiler value is chosen for the Compiler field. Once enabled, it becomes possible to select the path for pointing to a custom compiler supplied by Imagination Technologies.
3.2.3. View Menu

Figure 7 illustrates the View menu. The menu provides options for showing or hiding the various windows present in the PVRTrace GUI, allowing for workspace customisation.

Note: The workspace can also be customised by dragging and dropping the individual dockable windows to either the right or left hand sides of the interface or as standalone windows.

Show or Hide Statistics Window
To show or hide the Statistics window, toggle View -> Show Statistics (Figure 7).

Show or Hide Draw Calls Window
To show or hide the Draw Calls window, toggle View -> Show Draw Calls (Figure 7).

Show or Hide Current Call Window
To show or hide the Current Call window, toggle View -> Show Current Call (Figure 7).
Show or Hide Find Results Window
To show or hide the Find Results window, toggle View -> Show Find Results (Figure 7).

Show or Hide Shader Analysis Window
To show or hide the Shader Analysis window, toggle View -> Show Shader Analysis (Figure 7).

Show or Hide Image Analysis Window
To show or hide the Image Analysis window, toggle View -> Show Image Analysis (Figure 7).

Show or Hide Log Window
To show or hide the Log window, toggle View -> Log (Figure 7).

Show or Hide Modifications Window
To show or hide the Modifications window, toggle View -> Show Modifications (Figure 7).

Show or Hide Recorded Framebuffer Window
To show or hide the Recorded Framebuffer window, toggle View -> Show Recorded Framebuffer (Figure 7).

Show or Hide Object Data Viewer Window
To show or hide the Object Data Viewer window, toggle View -> Show Object Data Viewer (Figure 7).

Show or Hide Static Analysis Window
To show or hide the Static Analysis window, toggle View -> Show Static Analysis (Figure 7).

3.2.4. Help Menu
Figure 8 illustrates the Help menu. The menu provides options for accessing PVRTrace help assets, sending feedback, viewing general PVRTrace release information and checking for software updates.

View PVRTrace User Manual
To view the “PVRTrace User Manual”, click Help -> PVRTrace Help (Figure 8).

Submit Feedback
To provide feedback, click Help -> Feedback… This will open a dialog box where instructions are displayed on how to post feedback and request for support (Figure 8).

About PVRTrace
To view basic information about PVRTrace release information such as versioning and contact details, click Help -> About… (Figure 8).
Check for Updates
As of PowerVR Tools and SDK release 3.0, PVRTrace is able to auto-update. However, to force-check for software updates, click Help -> Check for Updates (Figure 8).

3.3. Working with the Function Call List
Upon loading a .pvrtrace file, the Function call list area of the PVRTrace GUI is populated with all the function calls that have been recorded for a particular frame. Part of the Function call list area is illustrated in Figure 9.

![Figure 9. Function call list area](image)

In the Functional call list area, for individual calls listed, it is possible to perform the following basic actions:

- Hover over a given call to display a tooltip which identifies both its contents and its data type, e.g., stride, size, pointer, etc.
- Each function call can be clicked on. This updates the renderstate to illustrate the actual state after the function call has completed.
- Individual calls can be right-clicked to reveal an action menu, as shown in Figure 10. This menu provides options for performing filtering and highlighting tasks, amongst others, and is explained in subsequent subsections of this user manual.

![Figure 10. Action menu displayed on right-clicking a function call](image)

Note: Any function call that has spawned an error is highlighted with red text after the function parameter in the Function call list area.
3.3.1. View Calls by Frame

For a given .pvrtrace file, the PVRTrace GUI can be used to analyse calls frame by frame. Every time a frame is selected, the contents of Function call list area are updated to show the calls that have been traced at the chosen frame. In order to view the calls for a specific frame, perform the following:

1. Use the frame scrubber to select a frame (Figure 11a). Alternatively, double-click the frame selector area (Figure 11b) to open a dialog box in which a frame value can be typed in. You can also enter a specific call UID to jump to that call.

2. Selecting a frame will then display the calls for that frame in the Function call list area. If image analysis is enabled, then the image output will also be updated in the Image Analysis window (see Section 3.4).

3.3.2. Expand or Collapse Draw Calls

Draw calls in the Function call list area can be expanded or collapsed in one mouse click to either show or hide the functions between the draw calls, respectively. To expand or collapse all draw calls, use the icon identified in Figure 9a. Alternatively, to expand or collapse a specific draw call, use the expand/collapse icon for that call (Figure 9b).

3.3.3. Filter Options

The ability to effectively filter calls in the PVRTrace GUI is a key part of working with the contents in the Function call list area. By filtering the displayed information, you are able to hide or reveal certain calls, as appropriate.

Quick Filtering

To quickly apply a filter to the desired listed calls, perform the following:

1. Mark the desired call for filtering by right-clicking the call and selecting the Add Filter option from the action menu (Figure 10). This will filter all calls of the same type, e.g., if a glBindBuffer call is marked for filtering, then all occurrences of the call in the Function call list area are marked.

2. Click the filter icon identified in Figure 9c to apply the filter. This will then hide all the calls that have been marked for filtering.

Advanced Filtering

Options for advanced filtering are accessed by selecting the icon identified in Figure 9d. This opens the Filter/Highlight Functions dialog box which provides further options to achieve a finer level of filtering (Figure 12).
Figure 12. Advanced filtering using the Filter/Highlight Functions dialog box

To use advanced filtering:

1. Open the Filter/Highlight Functions dialog box.
2. Under the Hide column tick the checkbox for the desired API. Finer filtering can be achieved by expanding an API name and selecting the required function groups for that API.
3. Click the Accept button to mark the calls present in the Function call list area for filtering.
4. Click the filter icon identified in Figure 9c to apply the filter. This will then hide all the calls that have been marked for filtering.

Filter by Thread ID
In the case of multi-threaded applications, it is possible to filter the contents in Function call list area to only show functions called in a certain thread. To use this filter, open the Thread ID dropdown menu (Figure 9e) and select the required listed thread value option.

Remove Filters
In order to remove the filters that have been applied to particular calls, perform the following:

1. Toggle the filter icon (Figure 9c) to reveal any previously filtered and hidden calls.
2. Right-click the call for which the filter is to be removed. This will reveal an action menu.
3. Select the Remove Filter option from the menu to unmark the call for filtering.

Note: All applied filters can be removed by right-clicking anywhere in the Function call list area and selecting the Remove All Filters option.

3.3.4. Highlight Options
The PVRTrace GUI enables you to apply highlighting to the calls displayed in the Function call list area. This is a useful feature that provides a visual way of quickly differentiating between API calls through the use of colour.

Quick Highlighting
To quickly add highlight to a call, perform the following:

1. Right-click a call in the Function call list area. This will reveal an action menu.
2. Select the Add Highlight option from the menu. This will then highlight the row corresponding to the chosen call, as well as all occurrences of the call in the Function call list area.
Note: It is also possible to add highlight to a given call by identifying the call in the Calls Summary tab of the Statistics window (see Section 3.5.1), right-clicking the call and selecting the Add Highlight option.

Advanced Highlighting
Options for advanced highlighting are accessed by selecting the icon identified in Figure 9d. This opens the Filter/Highlight Functions dialog box which provides further options to achieve a finer level of highlighting (Figure 13).

![Filter/Highlight Functions dialog box](image)

Figure 13. Advanced highlighting using the Filter/Highlight Functions dialog box

To use advanced highlighting:

1. Open the Filter/Highlight Functions dialog box.
2. Under the Highlight column tick the checkbox for the desired API. Finer highlighting can be applied by expanding an API name and selecting the required function groups for that API.
3. Click the Accept button to apply the highlighting settings.

Remove Highlights
In order to remove the highlights that have been applied to particular calls, perform the following:

1. Right-click a highlighted call in the Function call list area. This will reveal an action menu.
2. Select the Remove Highlight option from the menu. This will clear any highlight that has been applied to the call, as well as all occurrences of the highlighted call in the Function call list area.

Note: All highlights can be cleared by right-clicking anywhere in the Function call list area and selecting the Remove All Highlights option.

3.3.5. Search Options

Search by Criteria
In the PVRTrace GUI, you are able to search for call information. To use the search functionality:

1. In the Function call list area, select the icon identified in Figure 9f. This will open the Find dialog box with options to specify the search criteria (Figure 14).
2. Type into the provided text box the required string of characters for searching. Recent search terms are stored and can be retrieved by using the dropdown arrow (Figure 14a).

3. In order to refine the search criteria, use the various options provided in the dialog box (Figure 14b). For example, in the case of multi-threaded applications, specific threads can be selected. Also, search can be applied to the current selected frame, the entire trace or for a specific user-defined frame range. Additional options are present for searching by matching case and by using a search method which can be of two types, namely textual or by regular expression.

4. After the search criteria have been specified, click the Find button to retrieve the search results. The results are displayed in the Find Results window. An example of retrieved search results is illustrated in Figure 15.

Note: The syntax used in regular expression searching is documented in Appendix B.

Note: Clicking a call from the retrieved search results updates the currently selected call in the function call list and if required the current frame.

---

**Find Calls**

Occurrences of a call can be quickly narrowed down for inspection by performing the following:

1. Identify the desired call in the Function call list area and right-click it to reveal an action menu.

2. To search for all occurrences of the call in the current frame, select the option Find all in frame. Otherwise, if it is required to search for all occurrences of the call in the entire trace, select the option Find all in Whole Trace. This will display the search results in the Find Results window (Figure 15).
**3.3.6. Disable or Enable Calls**

During the analysis of calls in the Function call list area it is sometimes required to disable certain calls to facilitate the identification of problems in a trace. To disable a call, perform the following:

1. In the Function call list area, identify the call that is of interest.
2. Right-click the call to reveal an action menu.
3. Select the Disable Call option from the menu. The call will be disabled and greyed out. If the trace was recorded with data and, therefore, viewable in the Image Analysis window, then disabling a call causes it not to be processed when determining the image output.

Furthermore, all disabled calls will be summarised in the Modifications window, where, for a given call, the call number is also shown (Figure 16).

**Note:** Only a subset of calls may be disabled.

![Figure 16. Viewing disabled calls in the Modifications tab](image)

**Note:** To re-enable all the previously disabled calls, uncheck the disabled call from the Modifications window and apply changes. Alternatively, right-click anywhere in the Function call list area and select the Clear Disabled Calls option.

**3.3.7. Function Call Documentation**

In most cases, the documentation for function calls can be accessed by performing the following:

1. Identify the desired function call and right-click it in the Function Call List area to reveal an action menu.
2. Select the option called Function Quick Help. This will open the documentation page for the function call on the Khronos website.
3.4. Image Analysis

By default, if a trace being analysed was recorded with data, the Image Analysis window would display the overall render output of the current frame, from the first draw call to last (Figure 17). The image analysis feature supported in the PVRTrace GUI provides access to various means of visualizing an image output. The following basic actions can be accomplished during image analysis:

- **Zoom in and out**: Use the zoom menu provided to magnify or decrease the size of the displayed image (Figure 17a).
- **Select a frame**: Use the frame scrubber, located at the bottom right hand side of the PVRTrace GUI, to choose the desired frame to display (see Section 3.3.1).
- **Select a draw call**: It is possible, using the Image Analysis window to draw only a subset of the overall draw calls, down to a single draw call if required (Figure 17b). You can move between different calls by moving the slider beneath the image, or by selecting the call from the second dropdown menu. The first dropdown menu allows you to pick which surfaces to display. When you click on a draw call in the call view list, it will highlight the call in the Image Analysis window. To remove this, right click on the window and select Remove Highlighting.
- **Select a pixel**: A pixel can be selected by clicking a point in the displayed image. This functionality only works during shader analysis (see Section 3.10.3). The position of the pixel is marked with horizontal and vertical lines through the pixel coordinates.

*Note: When jumping frames, content that uses framebuffer objects or compute shaders may not render correctly. If this occurs, right-click the image and select Re-Evaluate.*

*Note: To remove the highlight for a selected pixel, right-click the image and select Remove Crosshair.*

*Note: Fragment analysis takes into account colour mask, render target changes and opaque pixel writes that sandwich translucent writes when calculating.*
3.4.1. Select a Render Output Mode

There are several visualization modes that can be used to view different types of render outputs. To select a specific render output mode, perform the following:

1. Click the dropdown menu provided in the Image Analysis window (Figure 17c). This will display several options, from which the required one can then be chosen. The following options are available:

   - **Colour**: This option displays a standard image output which is rendered, coloured, shaded, etc.
   - **Depth**: This is a visualisation of the depth buffer of the render.
   - **Stencil**: This is a visualization of the stencil buffer.
   - **Wireframe**: This option displays the wireframe of each object within the scene being rendered. The wireframe of each object is shaded.
   - **Wireframe (No Shaders)**: This option displays unshaded wireframes for each object in the scene.
   - **Depth Complexity**: This option displays a standard depth complexity render, with each object being drawn with varying transparency, where brighter areas are used to signal greater depth complexity. The data is normalised to efficiently use the data range.
   - **PowerVR Depth Complexity**: This option displays a standard depth complexity render built from pixel analysis data, indicating the depth complexity of pixels rendered by PowerVR hardware.

*Note: Tiling and Hidden Surface Removal are not simulated in image analysis, so the output should not be considered indicative of how PowerVR hardware performs rendering. This information is given for debugging purposes only.*
3.4.2. View Image Output by Draw Call Type

The Draw Call View Type dropdown menu identified in Figure 17c can be used to select how the draw calls are displayed in the Image Analysis window. To view the image output by draw call type, use the dropdown menu and select the required mode. The available modes are explained next:

- **Range**: This option displays a range of draw calls which are activated upon choosing the option.
- **Single Primitive**: This option displays the result of a single draw call value specified in the upper value selector box.

3.4.3. Other Actions

Right-clicking an image reveals an action menu with several options allowing the user to adjust its display size and orientation, as well as providing the ability to save the image.

3.5. Statistical Analysis

Statistical information is very important when analysing a trace. PVRTrace GUI displays various forms of statistical information to provide a digest of the fundamental metrics for calls and frames, as well as presenting interactive graphical information for inspection.

3.5.1. Call Statistics

The Calls Summary tab in the Statistics window provides a list of all the function calls within the currently selected frame and the number of times each function has been called within that frame (Figure 18). The function calls are normally arranged in descending order of frequency.

![Figure 18. Viewing API call statistics](image)

Watch a Call

Any call listed in the Calls Summary tab can be marked for watch purposes. The ability to watch a call facilitates the task of analysing a trace. To watch a listed call:

1. Right-click the desired call in the Calls Summary tab to open an action menu.
2. Select the Add Watch option from the menu. This will not only highlight the call in the Calls Summary tab, but also places it at the top of the list for easy identification (Figure 18a).

Remove Watch

Calls that are being watched can be unmarked by performing the following:

1. Right-click the watched call in the Calls Summary tab to open an action menu.
2. Select the **Remove Watch** option from the menu. This will unmark the call. If multiple calls are being watched, then it is possible to unmark all the calls by selecting the **Remove All Watches** option from the action menu.

### 3.5.2. Frame Statistics

The **Frames Summary** tab in the **Statistics** window provides an overall summary of the currently selected frame by listing the frequency of each listed item (Figure 19).

*Note: The statistics for the total number of triangles and lines correspond to the number sent to the API before any Hidden Surface Removal or culling.*

![Figure 19. Viewing frame statistics](image)

### 3.5.3. Graphical Representation of Statistics

The PVRTrace GUI provides a graphical representation of per-frame statistics (Figure 20) as well as thread usage statistics in the case of multi-threaded applications (Figure 21). To view the graph of statistics, from the **Menu** bar click **Tools** -> **Statistics Graph**. Alternatively, select the **Statistics Graph** button located in the bar above the **Function Call List** area (Figure 9g).

The horizontal axis provides a timeline corresponding to the recorded frames and double-clicking at a particular frame will load the associated data in the PVRTrace GUI.

The **Per-Frame Statistics** tab displays software counters in the traced application on a per-frame basis (Figure 20). To add or remove graphs of specific statistics and/or function calls, use their corresponding checkboxes. Values can be made to display on the graph by ticking the required checkbox under the **Label** column.
A number of options are provided in the dialog:

- From the File menu in the Statistics Graph dialog, it is possible to save the graph as an image, export the data as CSV files or close the window.
- Use the View menu to access various options to show or hide data labels, hide all statistics, zoom in and out, view the entire graph, centre the view on the active frame, and view the start or end portion of the graph.

To navigate through the timeline, click an area of the graph and drag horizontally to the right hand side. Also use the scroll bar in the graph area, as appropriate. An alternative way to zoom in and out of the graph is by hovering the cursor over the graph area and scrolling up or down using the mouse wheel. It is also possible to specify the zoom amount in the zoom level counter shown in Figure 20.

To display thread activity on a per-frame basis in the case of multi-threaded applications, click the Thread Usage tab (Figure 21). The thin bar at the top of the graph indicates the primary thread which called eglSwapBuffers for each frame. Each row illustrates a thread which has submitted calls for every frame. Note that it is not possible to save data to CSV files when viewing the thread usage graph.
3.5.4. Other Forms of Statistical Analysis

PVRTrace GUI displays other forms of statistical information, notably:

- **Draw calls statistics**: The number of vertices for each draw call in a particular frame is available from the Draw Calls window (see Section 3.7).
- **Shader analysis statistics**: It is possible to visualize the result of automated shader analysis from the Shader Analysis window (see Section 3.10).

3.6. Static Analysis

Static analysis is a crucial part of automated inspection of a trace as it helps identify a list of issues, in the form of errors and warnings, which may affect performance. This subsequently allows you to focus on rectifying the issues to improve an application. In the PVRTrace GUI, the results of static analysis are summarised in the Static Analysis window (Figure 22).

The column labelled as Level depicts the severity of an issue in descending order starting from the top of the list and the Title column contains a brief description of an issue. The Count column indicates the number of times that an issue has occurred.

![Figure 22. Viewing the results of static analysis](image.png)

**View Suggestions to Resolve an Issue**

The static analysis feature of PVRTrace allows you to view suggestions on how to resolve certain issues. In order to access that information, perform the following:

1. Click the desired error or warning displayed in the Static Analysis window. This will open an information box, an example of which is shown in Figure 23.
2. It is then possible to read the suggestion message displayed at the bottom of the information box (Figure 23a). The information box also lists occurrences of the function which caused the issue, together with their associated frame and call numbers.

*Note: Suggestion messages are hardware-specific.*

### 3.7. Inspection of Draw Calls

A summary of draw calls is displayed in the Draw Calls window (Figure 24), which details several bits of information for each entry in the list.

![Figure 24. Viewing the Draws Calls window](image)

To view the effect of a draw call for a given frame, perform the following:

1. Go to the desired frame by using the frame scrubber or frame selector. This will populate the Draw Calls window with a list of draw calls for that frame.
2. Click the required draw call in the list. This highlights the draw call in the Function Call List area. Information is also updated in the various windows of the PVRTrace GUI, e.g., the Image Analysis window will be updated, as appropriate, with the draw highlighted in the image.
The various columns displayed in the window can be enabled or disabled by right-clicking on the draw call list’s heading. The various columns that can be displayed and their description are explained in Table 7.

Table 7. Columns displayed in the Draw Calls window

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>Displays the unique IDs of the listed calls.</td>
</tr>
<tr>
<td>Call</td>
<td>Lists the names of the API calls.</td>
</tr>
<tr>
<td>Mode</td>
<td>This column lists the primitive types in the calls.</td>
</tr>
<tr>
<td>Target</td>
<td>Displays the surface the call is rendering into.</td>
</tr>
<tr>
<td>Program</td>
<td>From this column, we can see the programs used for the listed calls.</td>
</tr>
<tr>
<td>Vertices</td>
<td>Lists how many vertices are drawn by the calls.</td>
</tr>
<tr>
<td>Vertex Cost</td>
<td>This column displays the cost of vertex processing associated with the listed calls.</td>
</tr>
<tr>
<td>Fragments</td>
<td>Lists how many fragments are drawn by the calls.</td>
</tr>
<tr>
<td>Fragment Cost</td>
<td>Lists the costs of fragment processing.</td>
</tr>
<tr>
<td>Total Cost</td>
<td>Displays the total of both fragment and vertex costs.</td>
</tr>
</tbody>
</table>

Note: The draw call list and its data can be saved as a .csv file.

3.8. Current Call Inspection

The PVRTrace GUI facilitates the inspection of EGL and OpenGL ES states and objects at any point in a frame. This can be achieved by selecting a call in the Function Call List area. Upon selection the Current Call window (Figure 25) will be updated with the relevant information for the call.

3.8.1. EGL State

The state of EGL API for the selected call is displayed in the EGL State tab in the Current Call window (Figure 25). To view further details about the participating EGL state objects for that call, perform the following:

1. Select an item in the list. This loads additional details in the Object Data Viewer window (Figure 26).
2. Depending on the chosen item, different tabs are displayed in the Object Data Viewer for browsing the information.
3.8.2. **EGL Objects**

The **EGL Objects** tab in the **Current Call** window provides a summary of the EGL objects that are currently active up to the selected call (Figure 27). To view further details about the participating EGL objects, perform the following:

1. Select an item in the list. This loads additional details in the **Object Data Viewer** window, similar to the example shown in Figure 26.
2. Depending on the chosen item, tabs are displayed in the **Object Data Viewer** for browsing the information.

*Note: It is possible to toggle between showing and hiding the currently uncreated EGL objects by using the provided checkbox (Figure 27a).*

3.8.3. **OpenGL ES State**

The ability to view changes in OpenGL ES renderstate is an important aspect of current call analysis since, by minimising renderstate changes, it becomes possible to boost performance. The renderstate for the selected call is displayed in the **OpenGL ES State** tab in the **Current Call** window (Figure 28).
During renderstate inspection, any state change that occurred since the previous call is highlighted in red. The information is listed and grouped under collapsible sections such as program, textures, framebuffers, colour, polygon offset values, scissoring details, etc. These groupings are visible depending on the OpenGL ES version of the current EGL context for the current call.

To view further details about the participating OpenGL ES variables, perform the following:

1. Select an item in the list. This loads additional details in the Object Data Viewer window, an example of which is shown in Figure 29.
2. Depending on the chosen item, tabs are displayed in the Object Data Viewer for browsing the information.

Note: It is possible to toggle between showing and hiding unchanging OpenGL ES states by using the provided checkbox (Figure 28a).

3.8.4. OpenGL ES Objects

The OpenGL ES Objects tab in the Current Call window provides a summary of the OpenGL ES objects that are currently active up to the selected call (Figure 30). The information is listed and grouped under collapsible sections per object type.
To view further details about the participating OpenGL ES objects, perform the following:

1. Select an item in the list. This loads additional details in the Object Data Viewer window.
2. Depending on the chosen item, tabs are displayed in the Object Data Viewer for browsing the information.

Note: It is possible to toggle between showing and hiding the currently uncreated OpenGL ES objects by using the provided checkbox (Figure 30a).

### 3.9. Texture Inspection

The textures recorded in a trace can be inspected using the PVRTrace GUI. This allows you to visualize textures and cross-reference them to the calls that have participating textures. In order to view a list of recorded textures select the OpenGL ES Objects tab in the Current Call window (Figure 31). By default, all the textures active at the currently selected call are listed. Textures are usually shown with thumbnails and hold a GLES handle/name of the object ID. Details are also provided for the calls at which they were created and/or destroyed.

To inspect a texture, perform the following:

1. Select the OpenGL ES Objects tab in the Current Call window and go to the Textures section of the list (Figure 31).
2. Select the desired texture. This will then display the texture in the Object Data Viewer window (Figure 33). Alternatively, a texture can be picked from the relevant call in the Function call list area.

3. Further inspection of a texture can be done by viewing the data associated with it such as any mip-map levels, parameters, etc.

Note: The size of the displayed texture image can be adjusted by using the zoom feature provided (Figure 26a).

3.10. Shader Inspection, Modification and Analysis

3.10.1. Shader Inspection
The PVRTrace GUI supports the inspection of vertex, fragment and compute shaders. This is facilitated through the use of an editor which supports syntax highlighting and displays approximate per-line cycle counts and register information. To inspect a shader, perform the following:

1. Select the OpenGL ES Objects tab in the Current Call window. This will display the loaded OpenGL ES objects including any textures, shaders, programs, framebuffers, etc. (Figure 32).

2. In the list of objects, scroll to the Shaders section. The shaders are arranged as vertex and fragment shaders, where each shader is given a unique identifier.

Figure 32. Viewing loaded shaders

3. Select the required vertex or fragment shader by clicking it. This will then display the contents of the shader program in the Source tab of the Object Data Viewer (Figure 33).

Note: The editor present in the PVRTrace GUI is a built-in instance of PVRShaderEditor, which is also available as a separate utility in the PowerVR Graphics Tools & SDK. For more information, see the "PVRShaderEditor User Manual".

4. Once opened, inspect the shader program as appropriate.

There are alternative methods that can be used to open a shader program. These are:

- Pick a shader from the relevant call in the Function call list area.
• A shader can be opened by looking in the Programs section of the list of OpenGL ES Objects and selecting a program number. Then in the Shaders tab in the Object Data Viewer, click the desired vertex or fragment shader to open it.

![Figure 33. Inspecting a shader](image

3.10.2. Shader Modification

When analysing a trace, it may be required to modify a shader program in order to test new behaviour in image output and performance. To modify a shader, perform the following:

1. Repeat the procedure for shader inspection (see Section 3.10.1).
2. Make modifications to the shader program. If the program compiles successfully, the Apply button (Figure 33a) will become active to allow saving changes made to the shader.
3. Save changes by clicking the Apply button. For tracking purposes, the modified shader program is displayed in the Modifications window (Figure 34).

*Note: Shader modification can be removed by unchecking the modified shader and clicking on Apply Changes from the Modifications window (Figure 34). Alternatively, use the Reset button in the Source tab of the Object Data Viewer window (Figure 33b).*
3.10.3. Shader Analysis

PVRTrace GUI supports automated shader analysis through the provision of statistical information about a pixel’s contribution to the rendering of a given frame. This is a useful feature which allows you to quickly interpret the impact of fragment and vertex shaders, as well as fragment texture reads, in order to identify areas of concern that should be addressed. To use the shader analysis feature, perform the following:

1. In the Shader Analysis window, activate the Fragment Analysis checkbox (Figure 36). This will enable pixel selection in image analysis.
2. In the Image Analysis window, select the desired pixel by clicking the image (Figure 35). The position of the pixel will be marked with horizontal and vertical lines through the pixel coordinates.

3. Refer back to the Shader Analysis window which now displays a range of information, including:
   - Coordinates of the selected pixel (Figure 36a).
   - Calls that contributed to the shading of the selected pixel (Figure 36b). Double-clicking a listed fragment shader opens it in the Object Data Viewer window.
   - Summarized frame information about fragment counts, clock cycles in the shader compiler and texture reads for each fragment shader (Figure 36c). Several graphs are available for display.

   Note: The Selected Pixel view features thread colour for ease of analysis.

4. Select the desired graph from the Graph dropdown menu (Figure 36d). This will display a pie chart summarizing frame information. High count items accounting for larger segments of the
pie chart are generally a good indication of areas of concern that require addressing. There are five types of charts that can be viewed, namely:

- **Fragment Cost**: This option displays a pie chart representing the cost of each fragment shader as the number of fragments multiplied by the number of cycles per fragment.
- **Fragment Count**: This option displays the number of fragments per frame output by each fragment shader.
- **Fragment Tex. Reads**: This option displays the number of texture reads performed per fragment shader.
- **Vertex Cost**: This option displays the cost of vertices output by each primitive, which is the number of vertices multiplied by the number of cycles per vertex.
- **Vertex Count**: This option displays the number of vertices per primitive per frame.
- **Overdraw Distribution**: This option shows a bar graph illustrating the number of times each pixel is drawn.

![Figure 36. Viewing the Shader Analysis tab](image)

#### 3.11. Log Information

Any error from the application or from playing back a trace is reported in the PVRTrace GUI and to access this information select the Log window (Figure 37).

*Note: Any output not from the current frame is greyed out.*
3.12. Recorded Framebuffer

The Recorded Framebuffer window is active if the trace was recorded with the SaveFrameBuffer parameter enabled (see Section 2.4.1). The window displays the contents of the final framebuffer from the original device.
4. PVRTracePlayback

4.1. Overview

PVRTracePlayback is a tool for playing back a pre-recorded .pvrtrace file. It does this by reading the .pvrtrace file and repeating each function call from the file. As this requires the .pvrtrace file to contain information pertaining to the textures, vertex buffers, etc., from the original recording, it is required that the recording be performed with RecordData set to true in the configuration file (see Section 2.4).

4.2. Installation

PVRTracePlayback takes the form of a native executable on most operating systems and thus requires no installation. On some operating systems, e.g., Android, PVRTracePlayback takes the form of a native package, e.g., Android application package (APK). These native packages should be installed following the operating systems standard installation procedure, e.g., Android Debug Bridge (ADB) install.

4.3. Using PVRTracePlayback on Android

On Android, the PVRTracePlayback only supports playback of one window surface. If a trace file is being played with multiple window surfaces it will fail unless --window=n is specified to define the required window to play. The possible windows are listed by the playback on start up.

The following code is an example of a command line instruction:

```
am start -a android.intent.action.MAIN -n com.powervr.PVRTracePlayback/android.app.NativeActivity -e args "--window=1" -e file "/sdcard/trace.pvrtrace"
```
4.3.1. General Command-Line Options

Table 8 lists the general command-line options and their description.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--optimised-memory</td>
<td>Forces PVRTracePlayback to use a fixed amount of memory (~20MB). This is done avoiding loading the whole trace in memory.</td>
</tr>
<tr>
<td>--vsync=&lt;NUM&gt;</td>
<td>Overrides v-sync to a specified value (NUM).</td>
</tr>
<tr>
<td>--dont-resize</td>
<td>Forces PVRTracePlayback to not resize the viewport/scissor to fit screen resolution.</td>
</tr>
<tr>
<td>--delay=&lt;NUM&gt;</td>
<td>Forces PVRTracePlayback to wait a specified number of milliseconds between rendering frames.</td>
</tr>
<tr>
<td>--raise-context-versions</td>
<td>Raises the context client versions that PVRTracePlayback believes are incorrectly defined.</td>
</tr>
<tr>
<td>--skip-calls=&lt;UIDS&gt;</td>
<td>Forces PVRTracePlayback to skip calls by their UIDs. Example: --skip-calls=100,102-105</td>
</tr>
<tr>
<td>--skip-failed-calls</td>
<td>Makes PVRTracePlayback skip calls that failed during recording.</td>
</tr>
<tr>
<td>--ignore-thread=&lt;THREADS&gt;</td>
<td>Makes PVRTracePlayback not play the specified threads. Example: --ignore-thread=1,5</td>
</tr>
<tr>
<td>--ignore-window=&lt;WINDOWS&gt;</td>
<td>Forces PVRTracePlayback to not play the specified windows. Example: --ignore-window=6,8</td>
</tr>
<tr>
<td>--window=&lt;WINDOW&gt;</td>
<td>Only render the specified window.</td>
</tr>
<tr>
<td>--save-frame-buffer</td>
<td>Takes an optional range of frames at the eglSwapBuffer call to output. If you wish to define an output location for the framebuffers you can define it with &quot;--output=&quot;.</td>
</tr>
<tr>
<td>--export-saved-frame-buffers</td>
<td>Exports the framebuffers saved in the trace file.</td>
</tr>
<tr>
<td>--render-offscreen</td>
<td>Render to an offscreen framebuffer object instead of the draw surface</td>
</tr>
<tr>
<td>--run=&lt;NUM&gt;</td>
<td>Perform the specified value number of runs of the trace.</td>
</tr>
<tr>
<td>--verbose</td>
<td>Makes PVRTracePlayback display more information when playing back traces.</td>
</tr>
<tr>
<td>--info</td>
<td>Prints out the file information, such as a list of windows and threads.</td>
</tr>
<tr>
<td>--help</td>
<td>Displays this list and exits the utility.</td>
</tr>
<tr>
<td>--version</td>
<td>Prints version information and exits the utility.</td>
</tr>
</tbody>
</table>
### Advanced Command-Line Options

Table 9 lists the advanced command-line options and their description.

**Note:** The advanced command-line options are incompatible with the optimised-memory option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--range=RANGE</td>
<td>Renders a specified value of frames. Example: --range=1-10,15</td>
</tr>
<tr>
<td>--random</td>
<td>Sets PVRTracePlayback to play frames from the trace in a random order.</td>
</tr>
</tbody>
</table>
5. Troubleshooting

Local recording: The target device or PVRTrace keeps crashing
The library path(s) that are configured in the configuration file may be incorrect. Check the file for errors. Also check there is enough disk space on the target device.

Network recording: Network connection is not working
Check the target device and the host machine are both on the same subnet. If running Android, also ensure that the AndroidManifest.xml of the application being recorded contains internet permissions, similar to the following:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Network recording: Unexpected behaviour while recording
Ensure that the target device has enough disk space available to store a trace file. Network recording requires a good network connection. Consider recording a trace locally and pushing the file to the host machine if the network connection is poor.

Network recording: USB connection is not working (Android)
In order to connect a target device to a host machine using USB, the adb forward method must be used to open a TCP port on the host machine. For example, using the default TCP port (54321):

```
adb forward tcp:54321 tcp:54321
```

General: No graphical output on target Android device after reboot
This requires system recovery. PVRTraceInstaller can also be used to restore the system in case the trace libraries were wrongly installed. By executing with -u it will restore the system and terminate. For example:

```
/data/data/com.powervr.PVRHub/bin/ PVRTraceInstaller -u.
```

General: How do I use PVRTrace over remote desktop?
This will require using TightVNC. Firstly, go to http://tightvnc.com and download the installer for your desktop machine. Run the installer, and then follow the options to set VNC as a service on the machine (assuming you wish to use it as a server). Finally, set a password. It is also recommended you download and install the DFMirage Driver add-on, as it significantly speeds up server-side screen capture.

With all this done, you can now connect to a VNC server from another machine. Using your server password, you can now remote desktop your machine. As VNC only captures the image rendered by the remote machine, it efficiently supports OpenGL and DirectX applications.
6. Contact Details
For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics Tools and SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp
Appendix A. Manual Android Installation

A.1. Installation with Root Permission

PVRTrace can be installed manually on an Android device. This section captures the necessary steps for achieving manual installation on Android devices with root permission available.

Warning: Incorrectly installing PVRTrace on a device may damage it to the point where it is necessary to re-flash the device’s firmware.

The following provides some necessary information to get started with the manual installation procedure:

- Commands: $ means ‘command from a user shell’. # means ‘command from a root shell’. This requires Android Superuser (su) to be ran from the shell.
- cp vs. cat: Some Android devices may not have the cp utility used in the instructions provided next. If that is the case, then use cat instead.

For example:

```
# cp /src/path/<LIBRARY>.so /dst/path/<LIBRARY>.so
```

Can be exchanged by:

```
# cat /src/path/<LIBRARY>.so > /dst/path/<LIBRARY>.so
# chmod 0644 /dst/path/<LIBRARY>.so
```

A.1.1. Installation on Android 4.3 or Older Version

To manually install PVRTrace on a device with Android 4.3 (Jelly Bean) or older version:

1. Make the Android system directory writable.

   - Option 1: Using ADB.
     
     ```
     $ adb root
     $ adb remount
     ```

   - Option 2: Using the command-line.
     
     ```
     $ adb shell
     # mount -o rw,remount /system
     ```

2. Copy libPVRTrace.so to /system/vendor/lib.

   - Option 1: Using adb push directly (requires root shell, i.e., option 1 in step 1).
### Step 3: Copy the Recording Libraries to /system/vendor/lib/egl (or /system/lib/egl)

The libraries to copy are `libEGL_PVRTRACE.so`, `libGLESv1_CM_PVRTRACE.so` and `libGLESv2_PVRTRACE.so`.

**Note:** The location of the OpenGL ES driver is preferred, which in the example here is assumed to be `/system/vendor/lib` (the default location in Android 4.0 and newer).

- **Option 1:** Using `adb push` directly (requires root shell, i.e., option 1 in step 1).

  ```bash
  $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /system/vendor/lib/libPVRTrace.so
  $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/PVRTrace.so /system/vendor/lib/libPVRTrace.so
  # cp /sdcard/libPVRTrace.so /system/vendor/lib/libPVRTrace.so
  ```

- **Option 2:** Using `cp` to copy the file from an SD card on the device.

  ```bash
  $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /sdcard/libPVRTrace.so
  $ adb shell
  # cp /sdcard/libPVRTrace.so /system/vendor/lib/libPVRTrace.so
  ```

### Step 4: Create the `pvrtraceconfig.json` (see Section 2.4) and place it in the SD card. Make sure that `TraceFile` points to a writable location.

The `TraceFile` variable must be set accurately with an absolute path. If it is not correctly set, no file will be output. It is suggested that write permissions for the app be checked before setting this. As all subsequent processes will be traced after installation, it is recommended to use an output folder that allows write access for all processes so that PVRTrace output does not fail and potentially threaten the stability of the device.

### Step 5: Edit `egl.cfg` in `/system/lib/egl/` to use PVRTrace instead of the original driver.

**Note:** This change can be done at any time and will only affect new processes that are launched after the change. It is recommended that two copies of `egl.cfg` be used, one referencing the PVRTrace libraries and the other referencing the default libraries. Each file should be named separately and copied over the original `egl.cfg` as required.

For example, if the original file stated:

```
0 0 android
0 1 POWERVR_SGX_540_120
```

**Edit to:**

```
0 1 POWERVR_SGX_540_120
```
6. Once a trace is complete the changes to egl.cfg must be reverted to avoid any undesired behaviour. This should be done before a reboot to avoid the need for flashing the device’s firmware.

A.1.2. Installation on Android 4.4 or Newer Version

To manually install PVRTrace on a device with Android 4.4 (KitKat) or newer version:

1. Make the Android system directory writable.
   - Option 1: Using ADB.
   
   ```
   $ adb root
   $ adb remount
   ```
   - Option 2: Using the command-line.
   
   ```
   $ adb shell
   # mount -o rw,remount /system
   ```

2. Copy libPVRTrace.so to /system/vendor/lib.
   - Option 1: Using adb push directly (requires root shell, i.e., option 1 in step 1).
   
   ```
   $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /system/vendor/lib/libPVRTrace.so
   ```
   - Option 2: Use cp to copy the file from an SD card on the device.
   
   ```
   $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /sdcard/libPVRTrace.so
   $ adb shell
   # cp /sdcard/libPVRTrace.so /system/vendor/lib/libPVRTrace.so
   ```

3. Move the original EGL and OpenGL ES driver libraries from /system/vendor/lib/egl to /system/lib/egl. The libraries to copy are libEGL_<DRIVER_NAME>.so, libGLESv1_CM_<DRIVER_NAME>.so and libGLESv2_<DRIVER_NAME>.so.

   **Note:** Backing up the original libraries is extremely recommended. A good backup location is /data/local/tmp.
   - Option 1: Use mv to move the files.
   
   ```
   $ adb shell
   # mv /system/vendor/lib/egl/<LIBRARY>.so /system/lib/egl/<LIBRARY>.so
   ```

   **Note:** In case mv is not available in the system do a cp and then rm the source file.
4. Copy the recording libraries to /system/vendor/lib/egl. The libraries to copy are
libEGL_PVRTRACE.so, libGLESv1_CM_PVRTRACE.so and libGLESv2_PVRTRACE.so.

- Option 1: Using adb push directly (requires root shell, i.e., option 1 in step 1).

```
$ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/<LIBRARY>.so /system/vendor/lib/egl/
```

- Option 2: Use cp to copy the file from an SD card on the device.

```
$ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/<LIBRARY>.so /sdcard/
$ adb shell
# cp /sdcard/<LIBRARY>.so /system/vendor/lib/<LIBRARY>.so
```

5. Create the pvrtraceconfig.json (see Section 2.4) and place it in the SD card. Make sure that TraceFile points to a writable location.

The TraceFile variable must be set accurately with an absolute path. If it is not correctly set, no file will be output. It is suggested that write permissions for the app be checked before setting this. As all subsequent processes will be traced after installation, it is recommended to use an output folder that allows write access for all processes so that PVRTrace output does not fail and potentially threaten the stability of the device.

6. Reboot the device.

*Note: A hot reboot will work as well.*

- Option 1: Normal device reboot.

```
$ adb shell reboot
```

- Option 2: Hot reboot (only the Android runtime is restarted).

```
$ adb shell
# stop
# start
```

### A.1.3. Installation on 64-bit Android

To manually install PVRTrace on a 64-bit Android device:

1. Make the Android system directory writable.

- Option 1: Using ADB.

```
$ adb root
$ adb remount
```

- Option 2: Using the command-line.
$ adb shell
   # mount -o rw,remount /system

2. Copy `libPVRTrace.so` to `/system/vendor/lib64`.
   - Option 1: Using `adb push` directly (requires root shell, i.e., option 1 in step 1).

   ```
   $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /system/vendor/lib64/libPVRTrace.so
   $$ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/libPVRTrace.so /sdcard/libPVRTrace.so
   $ adb shell
      # cp /sdcard/libPVRTrace.so /system/vendor/lib64/libPVRTrace.so
   ```

   - Option 2: Use `cp` to copy the file from an SD card on the device.

3. Copy the recording libraries to `/system/vendor/lib64/egl`. The libraries to copy are `libEGL_PVRTRACE.so`, `libGLESv1_CM_PVRTRACE.so` and `libGLESv2_PVRTRACE.so`.
   - Option 1: Using `adb push` directly (requires root shell, i.e., option 1 in step 1).

   ```
   $ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/<LIBRARY>.so /system/vendor/lib/egl/
   $$ adb push /path/to/PVRTrace/Recorder/Android_<ABI>/<LIBRARY>.so /sdcard/
   $ adb shell
      # cp /sdcard/<LIBRARY>.so /system/vendor/lib64/<LIBRARY>.so
   ```

4. Create the `pvrtraceconfig.json` (see Section 2.4) and place it in the SD card. Make sure that `TraceFile` points to a writable location.

   The `TraceFile` variable must be set accurately with an absolute path. If it is not correctly set, no file will be output. It is suggested that write permissions for the app be checked before setting this. As all subsequent processes will be traced after installation, it is recommended to use an output folder that allows write access for all processes so that PVRTrace output does not fail and potentially threaten the stability of the device.

5. Rename the copied files using `mv` to remove the `_PVRTRACE` suffix.

   ```
   mv libEGL_PVRTRACE.so libEGL.so
   mv libGLESv1_CM_PVRTRACE.so libGLESv1_CM.so
   mv libGLESv2_PVRTRACE.so libGLESv2.so
   $$
   ```

6. Reboot the device.

   *Note: A hot reboot will work as well.*
A.2. Installation without Root Permission

For full instructions on how to perform manual installation without root permission, please refer to the “PVRTrace Quick Start Guide for Unrooted Android Devices”.

- Option 1: Normal device reboot.

```
$ adb shell reboot
```

- Option 2: Hot reboot (only the Android runtime is restarted).

```
$ adb shell
# stop
# start
```
Appendix B. Complete Configuration File

The following example, written for an Android OS, shows a standard trace config, which will apply to all applications ("*"). The last lines of this example show how to override the options per process, in this case our OpenGL ES 2.0 Water demo.

```json
{
  "*": {
    "Enabled": false,
    "Tracing": {
      "OutputFilename": "/sdcard/%pname.pvrtrace",
      "RecordData": true,
      "StartFrame": 0,
      "AppendTraceVersion": true,
      "EndFrame": 100,
      "OptimizeRS": false,
      "ExitOnLastFrame": true,
      "ClientBufferRecordFrequency": 1,
      "SaveFrameBuffer": false,
      "UseCompression": false
    },
    "Debug": {
      "Level": 1
    },
    "Network": {
      "Wait": true,
      "Enabled": false,
      "BufferSize": 256
    },
    "Host": {
      "Es2LibraryPath": "/system/lib/libGLESv2.so",
      "Es1LibraryPath": "/system/lib/libGLESv1_CM.so",
      "EglLibraryPath": "/system/lib/libEGL.so"
    },
    "Profiling": {
      "FunctionTimelineLevel": 1,
      "Enabled": false,
      "RendertateOverride": true,
      "SoftwareCounters": true,
      "Overrides": {
        "CullingMode": 0,
        "DisableStencilTest": false,
        "DisableAlphaTest": false,
        "ForceFlatColourFrag": false,
        "DisableDrawCalls": false,
        "DisableDepthTest": false,
        "DisableBlending": false,
        "Force2x2Textures": false,
        "DisableTextureModifications": false,
        "DisableScissorTest": false,
        "DisableFBAccess": false,
        "ForceViewport": false,
        "DisableFlushNFinish": false,
        "DisableTextureFiltering": false
      },
      "SwapInterval": 0
    },
    "com.powervr.OGLES2Water": {
      "Enabled": true,
      "Tracing": {
        "StartFrame": 100,
        "EndFrame": 200,
        "SaveFrameBuffer": true,
        "UseCompression": true
      }
    }
  }
}
```
Appendix C. Regular Expression Syntax

Table 10 provides a list of regular expression syntax and description which includes details for special constructs, logical operators, quantifiers, boundary matching, character classes, predefined character classes, characters and back references.

<table>
<thead>
<tr>
<th>Table 10. Regular expression syntax and description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Special constructs</strong></td>
</tr>
<tr>
<td>(?[i] X )</td>
</tr>
<tr>
<td>(?[!] X )</td>
</tr>
<tr>
<td>(?[n] X )</td>
</tr>
<tr>
<td>(?[N] X )</td>
</tr>
<tr>
<td>(?( X )</td>
</tr>
<tr>
<td>(??: X )</td>
</tr>
<tr>
<td>(?= X )</td>
</tr>
<tr>
<td>(?! X )</td>
</tr>
<tr>
<td>(?&lt;= X )</td>
</tr>
<tr>
<td>(?&lt;! X )</td>
</tr>
<tr>
<td>(?&gt; X )</td>
</tr>
<tr>
<td><strong>Logical operators</strong></td>
</tr>
<tr>
<td>X Y</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td><strong>Quantifiers</strong></td>
</tr>
<tr>
<td>X *</td>
</tr>
<tr>
<td>X +</td>
</tr>
<tr>
<td>X ?</td>
</tr>
<tr>
<td>X ()</td>
</tr>
<tr>
<td>X {n}</td>
</tr>
<tr>
<td>X {,m}</td>
</tr>
<tr>
<td>X {n,}</td>
</tr>
<tr>
<td>X {n,m}</td>
</tr>
<tr>
<td><strong>Note</strong>: These quantifiers are greedy. By following them with ? it becomes possible to turn them into lazy quantifiers, or follow them with + for possessive (non-backtracking) quantifiers.</td>
</tr>
<tr>
<td><strong>Boundary matching</strong></td>
</tr>
<tr>
<td>^</td>
</tr>
<tr>
<td>$</td>
</tr>
<tr>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
</tr>
<tr>
<td>\b</td>
</tr>
<tr>
<td>\B</td>
</tr>
<tr>
<td>\A</td>
</tr>
<tr>
<td>\Z</td>
</tr>
</tbody>
</table>

### Character classes

| []abc       | Match a, b, or c             |
| [^abc]      | Match any but a, b, or c     |
| [a-zA-Z]    | Match upper or lower-case a through z |
| [ ]         | Matches ]                   |
| [-]         | Matches -                   |

### Predefined character classes

| .           | Match any character          |
| \d          | Digit [0-9]                  |
| \D          | Non-digit                    |
| \s          | Space                        |
| \S          | Non-space                    |
| \w          | Word character [a-zA-Z_0-9]  |
| \W          | Non-word character           |
| \l          | Letter [a-zA-Z]              |
| \L          | Non-letter                   |
| \h          | Hex digit [0-9a-fA-F]        |
| \H          | Non-hex digit                |
| \u          | Single uppercase character   |
| \U          | Single lowercase character   |
| \p          | Punctuation (not including '"') |
| \P          | Non punctuation              |

### Characters

| \ \        | Back slash character        |
| \033        | Octal                       |
| \x1b        | Hex                         |
| \t          | Tab                         |
| \n          | Newline                     |

### Back references

| \1 to \9   | Reference to 1st to 9th capturing group |