PVRTune

User Manual

Filename : PVRTune.User Manual

Version : PowerVR SDK REL_3.3@2935291a External Issue

Issue Date : 28 May 2014

Author : Imagination Technologies Limited

Contents

List of Figures

1. Introduction

1.1. Software Overview

PVRTune is a performance analysis tool for the PowerVR series of graphics accelerators. It uses driver level counters and hardware registers to provide real-time data on the performance of an application running on a PowerVR GPU. PVRTune receives performance data from either the PVRPerfServer or the PVRHub application running on the target device.

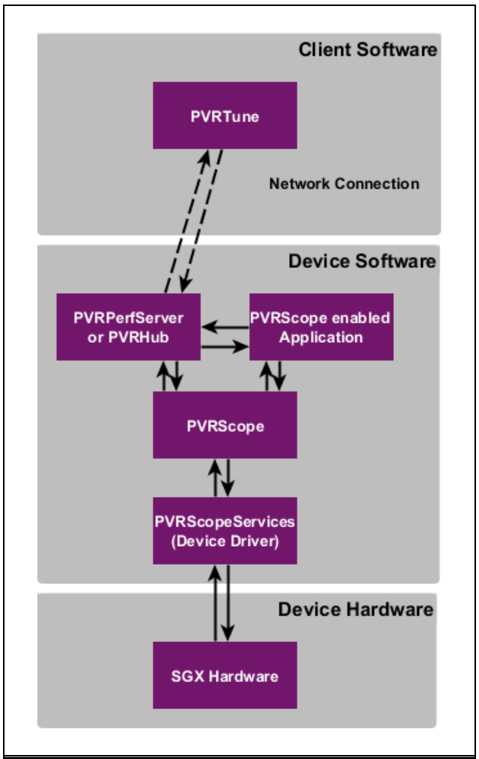


Figure 1-1 PVRTune Structure

1.1.1. PVRPerfServer

PVRPerfServer is an application which runs on the target device and reads counters and registers from the driver (and also optionally from any applications which use the PVRScope library) and either saves the data to a .pvrtune file, or transmits that data to PVRTune over a network.

1.1.2. PVRHub

PVRHub is an application which runs on Android and allows an instance of PVRPerfServer or PVRTrace to run on the target device. For the remainder of this document, all PVRPerfServer functionality can be accomplished using PVRHub, unless otherwise stated.

On Android, PVRHub is an application with a User Interface.

On Linux, PVRHub is a set of folders and scripts.

For more information, see the “PVRHub User Manual”.

1.1.3. PVRTune

PVRTune is the client-side element of the combined package. It provides a graphical representation of the real-time information being sent by PVRPerfServer, as well as providing a means to save and load .pvrtune files from previous sessions.

1.2. Document Overview

The purpose of this document is to serve as a complete user manual for PVRTune and PVRPerfServer. It includes installation instructions, a guide to the functionality of each application and a complete listing of all interface and command-line options and preferences, broken down by application.

2. PVRPerfServer

2.1. Overview

PVRPerfServer is a console application for Android, Linux, Neutrino, and Windows. Its purpose is to read counters and registers from the PowerVR hardware on a device and either save the data to a .pvrtune file, or transmit that data to PVRTune over a network.

2.2. Requirements

In order for PVRPerfServer to function correctly the driver must have performance profiling enabled. In many cases, as the overhead is so small, this functionality is available by default. This can be checked by confirming the existence of ‘PVRScopeServices.dll’, ‘libPVRScopeServices.so’ or an equivalent. If PVRPerfServer fails to initialise, it is likely that performance profiling support has been removed from the device drivers.

2.3. Installation

2.3.1. From Installer

Download the PVRTune package and follow the on screen instructions. Once the package has successfully installed PVRPerfServer can be found within the PVRTune folder in the install directory as follows:

```
<InstallDir>\PVRTuneDeveloper\PVRPerfServer\<PLATFORM>\
```

2.3.2. Android

PVRPerfServer functionality is obtained by installing the PVRHub application for Android. To install this application, run ‘`adb install PVRHub.apk`’ on your local machine.

2.3.3. Linux

With the package successfully installed, copy the PVRHub folder to the target device and follow the instructions found in the PVRHub User Manual.

2.3.4. Neutrino and Windows

With the package successfully installed, copy the PVRPerfServer binary to the target device.

2.4. Driver Compatibility

2.4.1. Compatible SGX DDK Ranges

On SGX devices, PVRPerfServer is limited to supporting the following driver versions:

Table 1. Compatible SGX DDK Ranges

1.3.13.1589 onwards; 1.4.14.593 onwards

2.5. Usage

2.5.1. Android

With the Android .apk installed, open the application menu and run ‘`PVRHub`’.

2.5.2. Linux

From a command-line interface run “`pvr_profile <binary>`” where `<binary>` is the application you wish to profile.

2.5.3. Neutrino and Windows

From a command-line interface run the PVRPerfServer executable.

IMPORTANT: Only one client at a time may be connected to an instance of PVRPerfServer.

2.5.4. Command-Line Options

PVRPerfServer supports several command-line options.

Option	Effect
-h	Show help text.
/?	Show help text.
--disable-user-input	Disable user input. Not available on all operating systems. Use to run PVRPerfServer in the background on Linux consoles.
--disable-hwperf	Disables the use of PVRScope’s hardware performance functionality.
-group=N	On start-up, switch the hardware to the specified counter group number.
-port=N	Network port to use; default is 6520.
-sendto=”myfile”	Instead of using the network, record data directly to file “myfile”
-t=N	Time, in milliseconds, between counter updates; the default value is 2. This value can be increased to reduce the CPU usage of PVRPerfServer.
-c=N	Time, in milliseconds, between CPU Load updates; the default value is 200.
-pid	Gather CPU usage data of relevant programs.
-pid=N,M	Gather CPU usage data of relevant programs, in addition track the specified PIDs.
-periodic=1/0	Enables/disables periodic timing tasks (for use when recording to a file).
-graphics=1/0	Enables/disables graphics timing tasks (for use when recording to a file).

2.5.5. Run-Time Options

PVRPerfServer supports several key-presses at run-time.

Table 2. Run-time hotkeys

Key	Effect
H	Show help text.
M	Send a Mark. Useful for placing simple markers into the data stream, annotated with a number that increments for each mark.
Q	Quit PVRPerfServer.

3. PVRTune

3.1. Overview

PVRTune is an application for Windows, Linux, and Mac OS. It provides a graphical representation of the real-time information sent by PVRPerfServer, as well as providing a means to save and load .pvrtune files from previous sessions. The information that displayed is broken down into groups of profiling counters which can be selected using the Counter Table. The currently active group is highlighted, and its contents updated in real-time in the legend on the right of the Timeline. Finally the Graph View allows for the graphing of these counters, as well as the display of frame-by-frame timing data.

3.2. Installation

3.2.1. From Installer

Download the PVRTune package and follow the on-screen instructions. Once the package has successfully installed, the PVRTune GUI application can be found within the PVRTune folder in the install directory as follows:

```
<InstallDir>\PVRTuneDeveloper\PVRTune\<PLATFORM>\
```

3.3. Connection Interface

3.3.1. Connection Window

The Connection Window is the first window that is seen when PVRTune is launched. It consists of several disabled areas, and an area providing several methods to load a .pvrtune file or connect to a target device running PVRPerfServer, as shown in Figure 3-1.

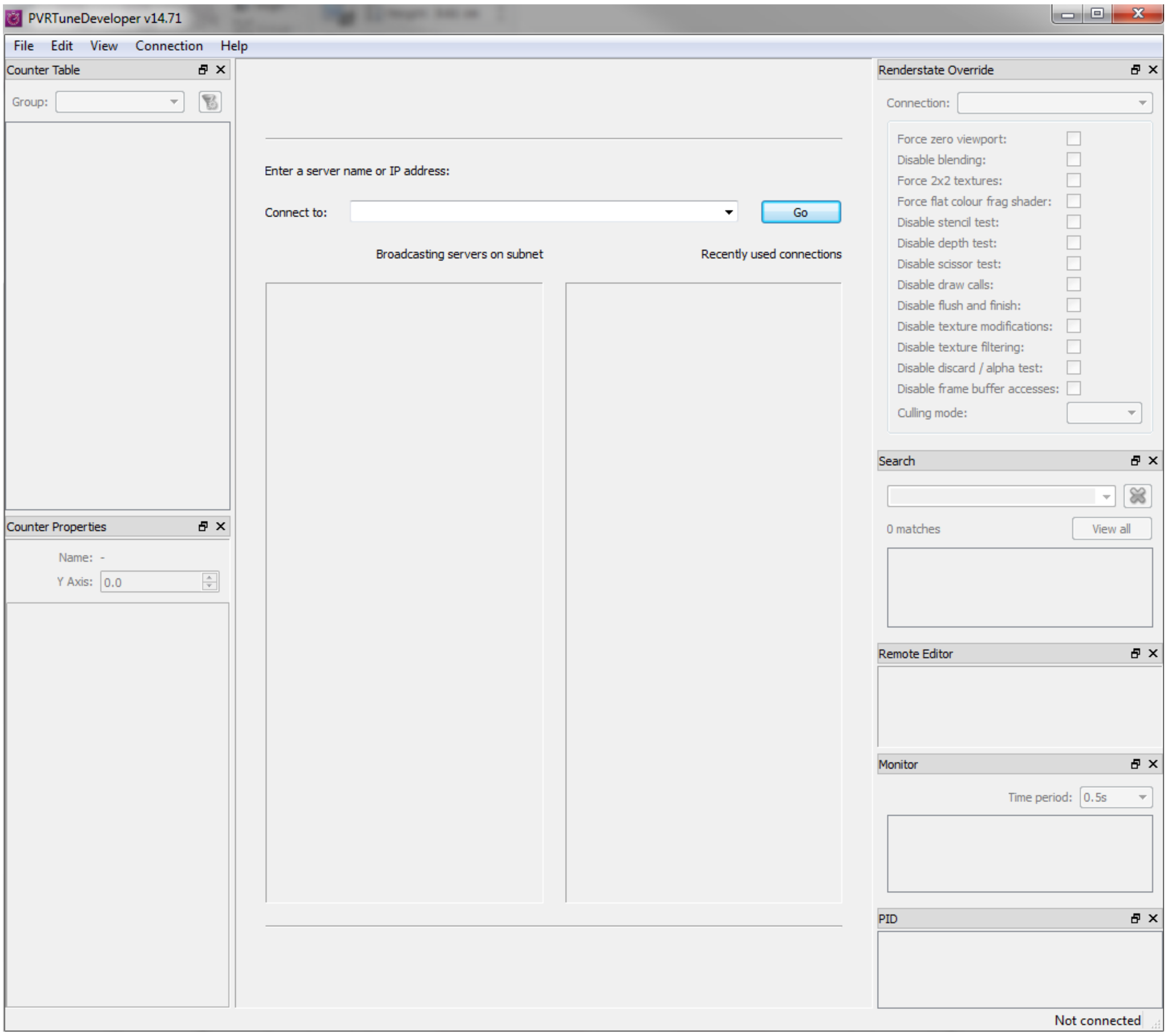


Figure 3-1 Connection Window

To connect to a target device:

Connect To

The `'Connect To'` box allows the user to enter an IP, or an IP resolvable name, of a target device. Upon hitting `'Go'`, if an instance of PVRPerfServer is found, PVRTune connects to the target device.

Broadcasting Servers

When launched, and periodically after launch, PVRPerfServer broadcasts its existence to the subnet to which it is connected – the 'Broadcasting Servers' box in PVRTune lists all the instances of PVRPerfServer currently broadcasting on the user's subnet. Connect to the target device by clicking the target IP address in the list.

Recently Used Connections

The 'Recently Used Connections' box contains a list of the IP addresses of devices which have recently been connected to. Connect to a device by clicking an IP address in the list.

Localhost

Setup a connection to a device connected directly to the localhost. This option is available through the Connection menu, for more information see Section 3.7.4.

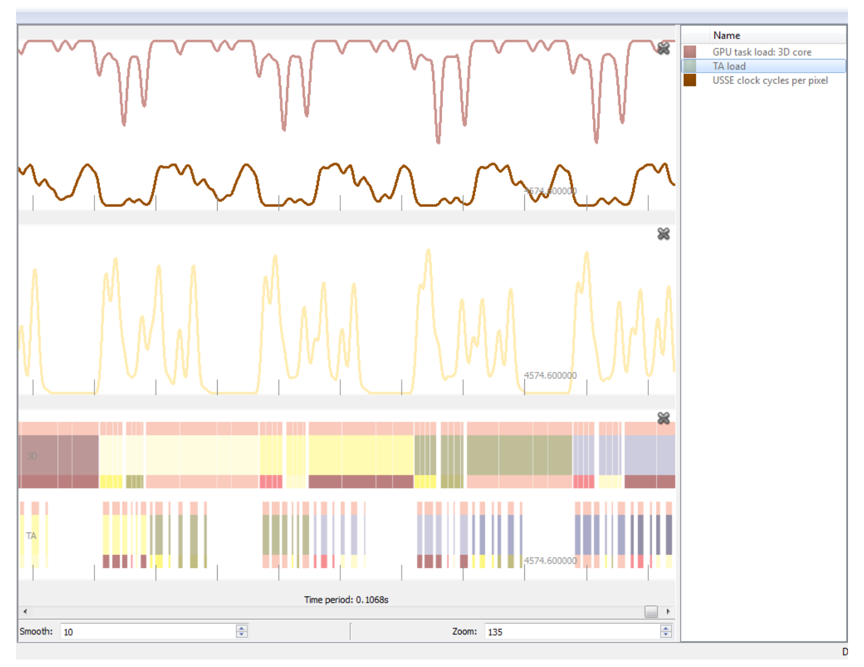
Note: PVRTune uses TCP port 6520 to send data to and from Android devices. In order to setup a localhost connection, `'adb forward'` *must be used to forward TCP data across the USB connection. To run* `'adb forward'`*, enter the following command in a command prompt:*

`'adb forward tcp:6520 tcp:6520'`

3.4. Main Interface

3.4.1. Timeline

The Timeline (as portrayed in Figure 3-2) allows for the various counters sent by PVRPerfServer to be displayed in one or more graphs. Timing data can be added or removed from any graph by right-clicking and selecting `'Render Timing Data'`. The Timeline includes a list of the counters currently being displayed.



Smooth: 10

Zoom: 135

Figure 3-2 Timeline

Adding/Removing Counters

To add counters to a graph, drag the counter from the Counter Table onto the Timeline. The counter can be dragged to an existing graph, or to the narrow area between graphs – this creates a new graph and displays the counter by itself.

The y-axis scale for each counter can be changed in the `'Counter Properties'` box. Once a counter has been placed onto a graph, a legend appears displaying the counter colour and number, as can be seen in Figure 3-2. Remove a counter from the Timeline by clicking a counter and pressing `'Delete'` or by right clicking and choosing `'Delete'`.

Individual graph areas can be closed by clicking the `'X'` in the top right corner of the area.

Events and Marks

Several events (also known as 'Marks') are displayed on the Timeline in addition to all of the other information. These marks are signified by bars running from the top of the graph to the bottom. A full list of the supported events can be found in Appendix B.

Right-click Menus

Right click on the Timeline to display a menu with several options:

- Remove selected counter,
- Render 25%, 50%, 75% Quarters,
- Enable/disable the Rendering of Timing Data (TA/3D),
- Enable/disable the Rendering of Marks (events),
- Pause or un-pause the graph,
- View All, View Earliest, and View Latest navigation options,
- Save the current graph to an image file.
- Restore default view.

Right click on the counter legend to display a menu with options to select all the currently graphed counters, delete the selection, or delete all of the currently graphed counters.

Time-Range Selection

It is also possible to modify the selected time range using the graph view directly; this selected region displays a ruler at the bottom. Select a region of the graph by using ‘Ctrl’ and Left Click and Drag. Deselect with ‘Ctrl’ and Left Click on an empty area. Expand the selection using ‘Shift’ and ‘Ctrl’ and Left Click.

Time Period

This value represents the amount of time the graph covers from left to right.

3.4.2. Timing Data (TA/3D)

The Right-click Menu contains an option: `‘Render Timing Data’`; when this option is ticked the timing data is displayed, as shown in Figure 3-3.

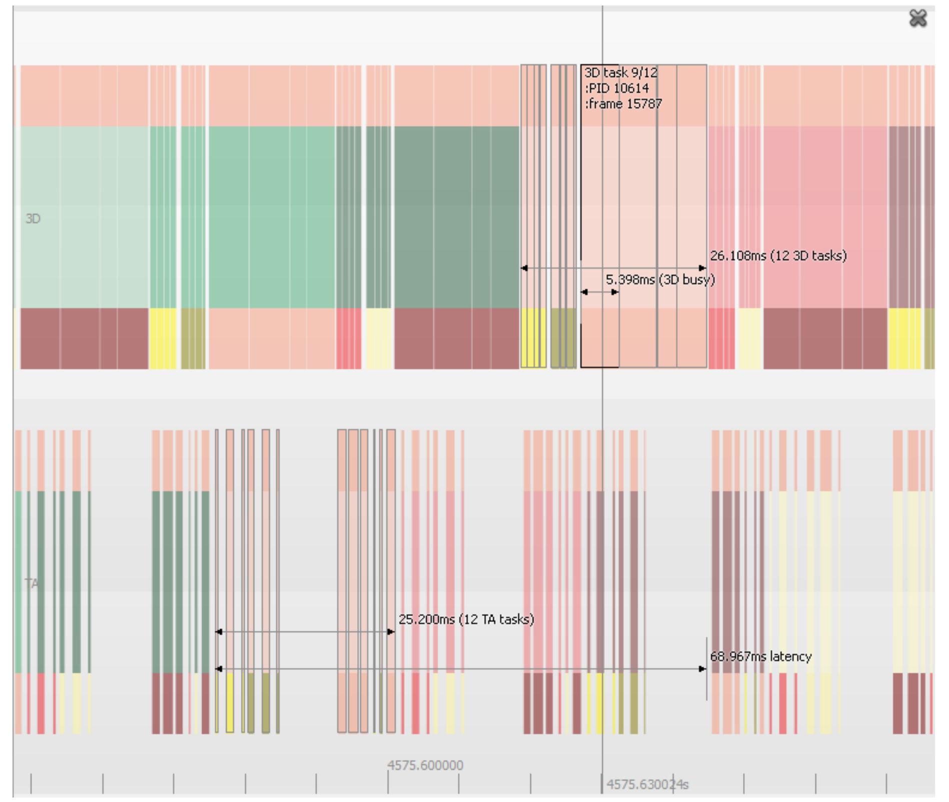


Figure 3-3 Timing Data

Tasks

Timing data appears in the bottom graph by default, and can also be displayed in other graphs by right-clicking and selecting `'Render Timing Data'`. The Timing Data is arranged on several timelines, each with their own label. The timeline labelled `'TA'` represents TA Core Time, this is a measure of time spent tiling/culling the frame and running vertex shaders. The timeline labelled `'3D'` represents 3D Core Time, and is a measure of how much time is spent fetching textures, processing fragment shaders and other fragment processing tasks. These timelines represent the two main stages in the TBDR (Tile Based Deferred Renderer) process.

Each block represents a given task/activity within a frame and is colour coded in order for the frame, process ID and render target to easily be spotted.

In addition to `'TA'` and `'3D'` the Timing Data can also include information on Transfer Tasks, 2D Core Time (for chips with a dedicated 2D Cores), Compute Time (for Series6 onwards), and custom timing data sent using PVRScope.

Mouse Hover Data

A wealth of information can be gained from the timing data by holding the mouse pointer over a task. This information details the process ID, the frame number, the total number of tasks on each timeline (such as TA and 3D) that can be attributed to the same frame, the amount of time spent on each task, the amount of time spent processing a set of tasks, and the total amount of time spent processing the frame.

Transfer Tasks

Transfer Tasks represent time spent processing tasks related to moving large amounts of memory, such as blitting or texture uploading. Figure 3-4 Transfer Tasks shows a series of these tasks as they appear in the Timing Data, as a series of grey blocks with pink tips.

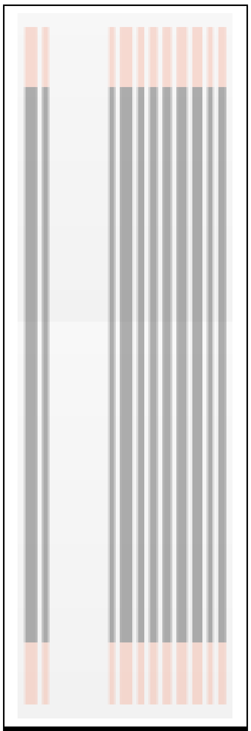


Figure 3-4
Transfer Tasks

Task Colour Coding

The timing data information is colour coded for convenience; blocks of the same colour represent a single frame as can be seen in Figure 3-3 (these colours are recycled every sixteen frames). In addition to the general block colour, each tip of a block is coloured. The top tips represent the process ID, different colours indicating different process IDs. The bottom tips represent the render target, a different colours indicating different render targets.

Driver Timing Data (when used in combination with PVRTrace)

Displayed on its own timeline, Driver Timing Data appears as a series of tasks labelled with the thread ID that called the driver.

Note: By default, only the most expensive calls are displayed (for example, glDrawElements, glReadPixels, shader compilation and texture uploads). For a more verbose output, open PVRHub, select Options, and set API Function Timing Events to `'Verbose'`*.*

WARNING: Setting API Function Timing Events to `'Verbose'` *may affect performance.*

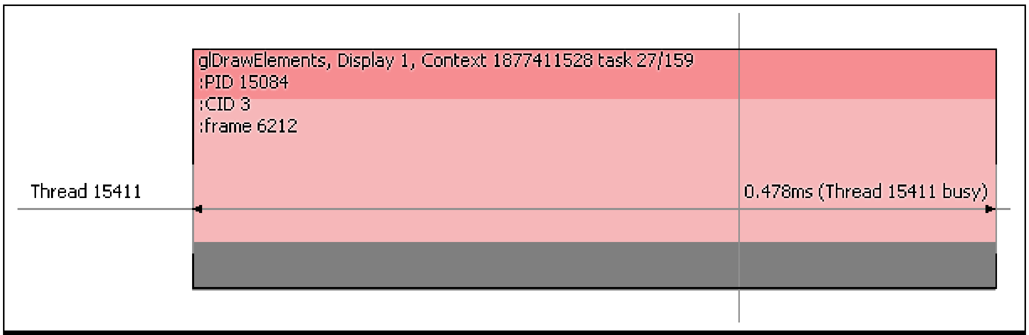

glDrawElements, Display 1, Context 1877411528 task 27/159
:PID 15084
:CID 3
:frame 6212
Thread 15411
0.478ms (Thread 15411 busy)

Figure 3-5 Driver Timing Data

Task Colour Coding Example

In Figure 3-6 Task Colour Coding Example, the core colour of a task represents a single frame; each frame has an associated process ID (top tip) and a render target (bottom tip).

Repeated top tip colours refer to the same process IDs. The tasks in Figure 3-6 were generated by one process, indicated by a single colour on the top tip (pale pink).

Repeated bottom tip colours refer to the same render targets. Two render targets are present in Figure 3-6; indicated by the two colours on the bottom tips of the tasks. In double or triple buffered situations the render targets are different for each frame, alternating between each of the back buffer targets.

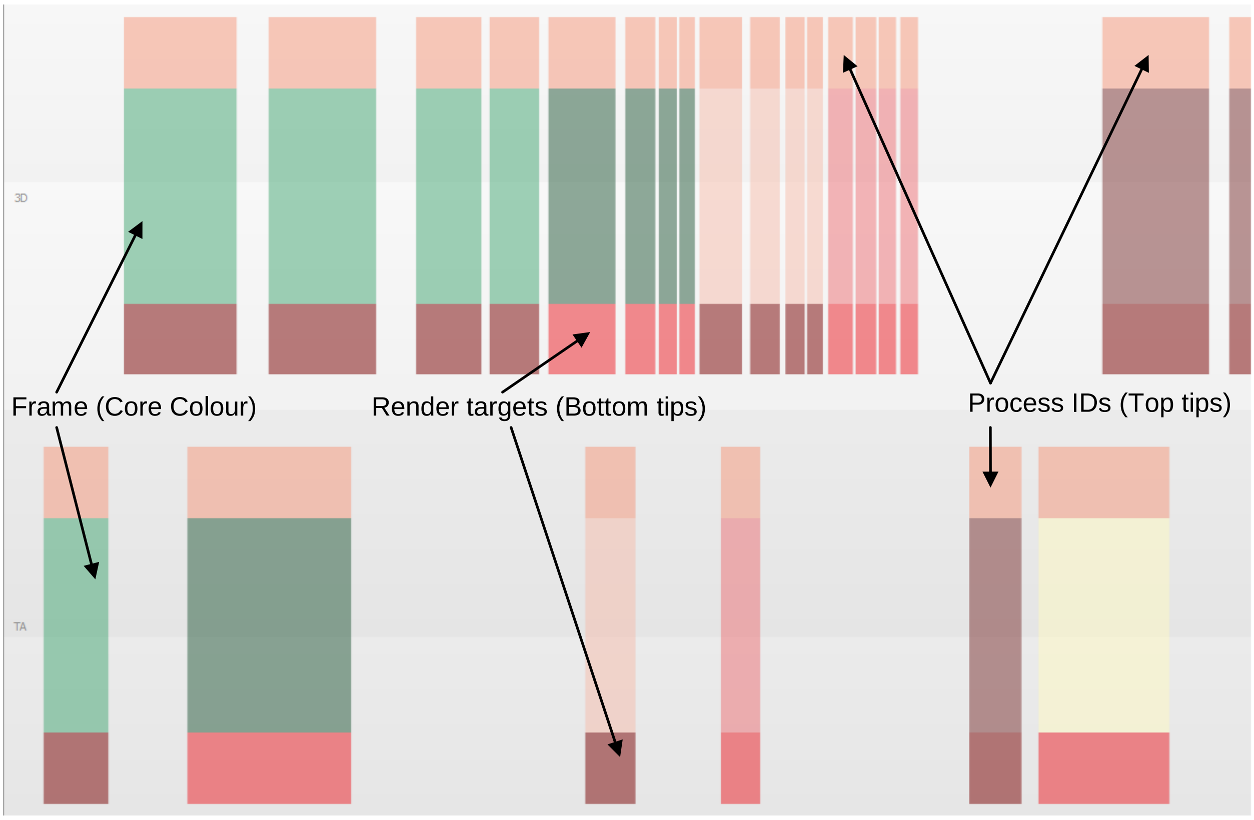
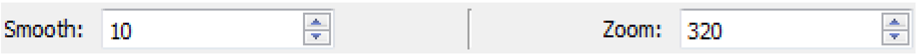


Figure 3-6 Task Colour Coding Example

Time-Range Selection

The selected time range can be modified using the timing-data activity blocks directly. ‘Ctrl’ and Left Click on an activity to exactly select it’s time range. ‘Shift’ and ‘Ctrl’ and Left Click on an activity to add the activity’s time range to the current selection.

3.4.3. Graph View Control Bar



Smooth: 10

Zoom: 320

Figure 3-7 Graph View Control Toolbar

Smooth

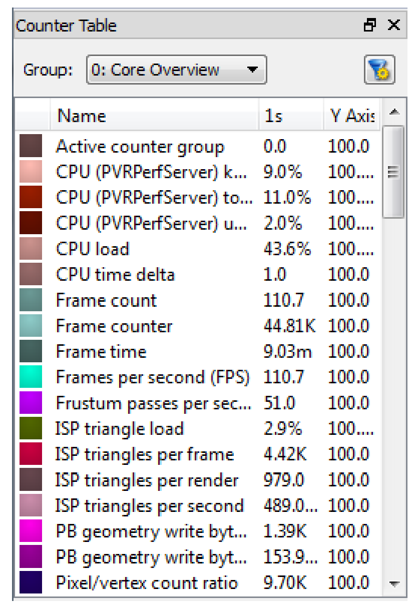
The `'Smooth'` option sets the value used to smooth out the counters displayed in the Timeline. This makes the counter graphs easier to interpret. The default value is `'10'`; the max value is `'80'`, with the graph smoothness increasing as the value is increased.

Zoom

`'Zoom'` represents the zoom being applied to the Timeline. As this value increases so does the time period displayed on the graph. This can also be adjusted by scrolling the mouse wheel while the cursor is in the Timeline area.

3.4.4. Counter Table

The counter table displays the counters available within the current group, as selected in the Group drop-down box. These counters are updated in real-time and can be dragged onto the Timeline to give a graphical representation of their change over time.



Group: 0: Core Overview

	Name	1s	Y Axis
■	Active counter group	0.0	100.0
■	CPU (PVRPerfServer) k...	9.0%	100....
■	CPU (PVRPerfServer) to...	11.0%	100....
■	CPU (PVRPerfServer) u...	2.0%	100....
■	CPU load	43.6%	100....
■	CPU time delta	1.0	100.0
■	Frame count	110.7	100.0
■	Frame counter	44.81K	100.0
■	Frame time	9.03m	100.0
■	Frames per second (FPS)	110.7	100.0
■	Frustum passes per sec...	51.0	100.0
■	ISP triangle load	2.9%	100....
■	ISP triangles per frame	4.42K	100.0
■	ISP triangles per render	979.0	100.0
■	ISP triangles per second	489.0...	100.0
■	PB geometry write byt...	1.39K	100.0
■	PB geometry write byt...	153.9...	100.0
■	Pixel/vertex count ratio	9.70K	100.0

Figure 3-8 Counter Table

A full list of the available counters can be found in Appendix A.

For instructions on how to add or remove counters from the Graph View see Section 3.4.1.

Software Counters

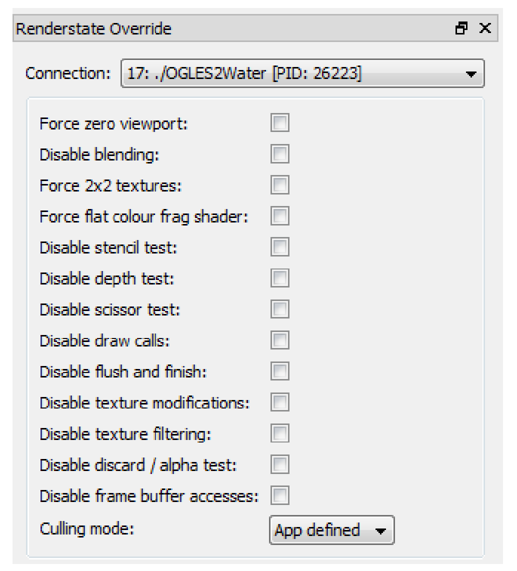
In addition to hardware counters from the driver, there can be software counters from PVRTrace (for information on receiving software counters from PVRTrace, see Section 4.3) and/or custom counters from the user's application (see PVRScopeComms).

3.4.5. Renderstate Override

The render state override panel allows remote control of the target device's render state. The following overrides are available:

- **Force zero viewport** – Modifies the viewport to have zero sized dimensions.
- **Disable blending** – Disables alpha blending.
- **Force 2x2 textures** – Forces textures to be 2 pixels by 2 pixels in size.
- **Force flat colour frag shader** – Forces the fragment shader to output a single colour.
- **Disable stencil test** – Disables stencil test.

- **Disable depth test** – Disables depth test.
- **Disable scissor test** – Disables scissor test.
- **Disable draw calls** – Disables all glDraw() function calls.
- **Disable flush and finish** – Disables all glFlush() and glFinish() function calls.
- **Disable texture modifications** – Disables glTexSubImage2D() and glTexSubImage3D(). This disables the updating of texture sub-regions.
- **Disable texture filtering** – Disables texture filters. All texture filters are set to the nearest point.
- **Disable discard / alpha test** – Disables alpha test.
- **Disable frame buffer accesses** – Disables glReadPixels() function calls.
- **Culling mode** – Forces the culling mode to one of the following settings:
    - **App defined:** Perform culling as defined in the user application.
    - **None:** Disable culling.
    - **Back:** Perform back-face culling.
    - **Front:** Perform front-face culling.



Connection: 17: ./OGLES2Water [PID: 26223]

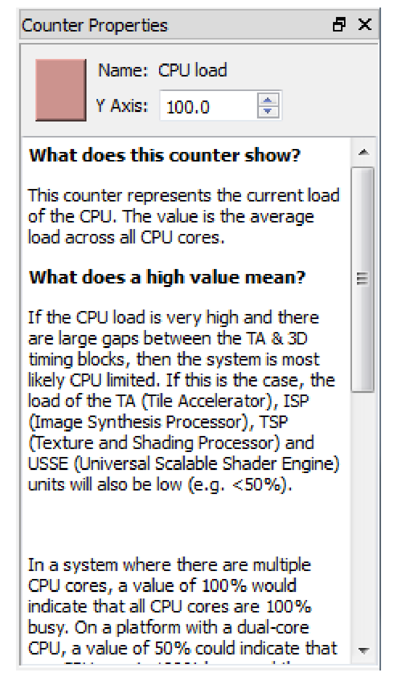
Force zero viewport: ☐
Disable blending: ☐
Force 2x2 textures: ☐
Force flat colour frag shader: ☐
Disable stencil test: ☐
Disable depth test: ☐
Disable scissor test: ☐
Disable draw calls: ☐
Disable flush and finish: ☐
Disable texture modifications: ☐
Disable texture filtering: ☐
Disable discard / alpha test: ☐
Disable frame buffer accesses: ☐
Culling mode: App defined

Figure 3-9 Renderstate Override

3.4.6. Counter Properties

This section of the interface displays the properties of the most recently clicked counter, and allows them to be edited. The following properties are displayed:

- Counter colour
- Counter name
- Maximum value of the Y-Axis on the Graph View
- Counter description.



Counter Properties

Name: CPU load

Y Axis: 100.0

What does this counter show?

This counter represents the current load of the CPU. The value is the average load across all CPU cores.

What does a high value mean?

If the CPU load is very high and there are large gaps between the TA & 3D timing blocks, then the system is most likely CPU limited. If this is the case, the load of the TA (Tile Accelerator), ISP (Image Synthesis Processor), TSP (Texture and Shading Processor) and USSE (Universal Scalable Shader Engine) units will also be low (e.g. <50%).

In a system where there are multiple CPU cores, a value of 100% would indicate that all CPU cores are 100% busy. On a platform with a dual-core CPU, a value of 50% could indicate that

Figure 3-10 Counter Properties

The Y-Axis scale can be edited by clicking in the box and entering a new number, for particularly large numbers (such as transformations per frame) it is necessary to enter a large scale. Likewise, for very small numbers (such as frame time) it is necessary to enter a fractional value.

The colour can be edited by clicking on the colour swatch; this opens a colour selection dialog where a new colour can be selected.

3.4.7. Search

Figure 3-11 shows the search bar as it appears in PVRTune.

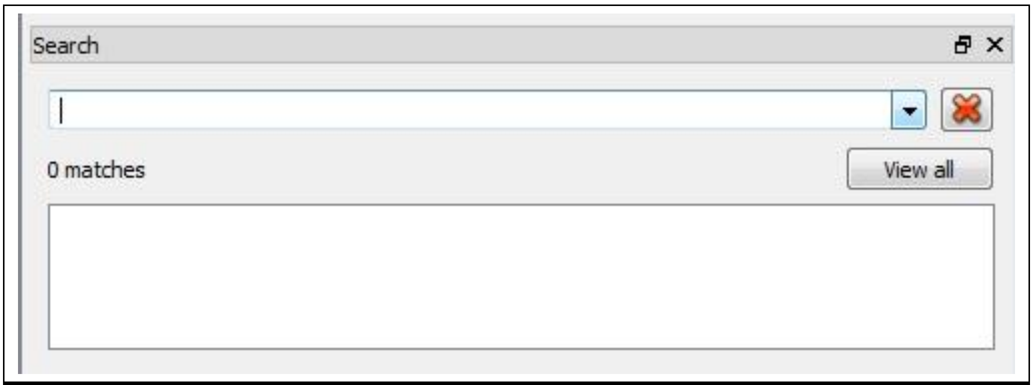


Figure 3-11 Search Bar

In PVRTune you can search for Activities (such as ‘3D’), Marks (such as ‘some string’), Counters (such as ‘triangles per’), Frames (for example ‘40000’), PIDs (such as ‘4242’), Time (as per the x-axis in graph view, so you would input ‘1200’ for example), and Time Range (by separating two numbers with a pair of dots – ‘1200.1..1200.2’ for example).

Time-Range Selection

You can also make a Time Range selection by holding ‘Ctrl’ whilst clicking on search results. Holding ‘Shift’ and ‘Ctrl’ will merge this selection with an existing selection, allowing you to extend the time range as necessary.

3.4.8. Remote Editor

PVRScope enabled applications can register data as being editable or readable by PVRTune; the value of this data can then be passed by PVRScope, via PVRPerfServer to PVRTune. Any values that have been registered appear in the `'Remote Editor'` panel; from here these values can be edited and the changes automatically sent back to the application. This functionality allows for real-time, remote editing of shaders, and various numerical value adjustments.

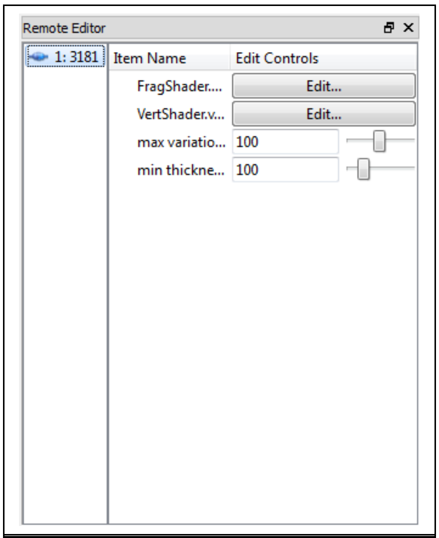


Figure 3-12 Remote Editor

More information on PVRScope can be found in the “PVRScope User Manual”.

3.4.9. Monitor

The Monitor section displays and categorizes the various loads being tracked by PVRTune, as shown in Figure 3-13. The time period dropdown list permits you to choose the over how much time the loads should be averaged. In the example below, the averages from every 0.5 seconds are displayed.

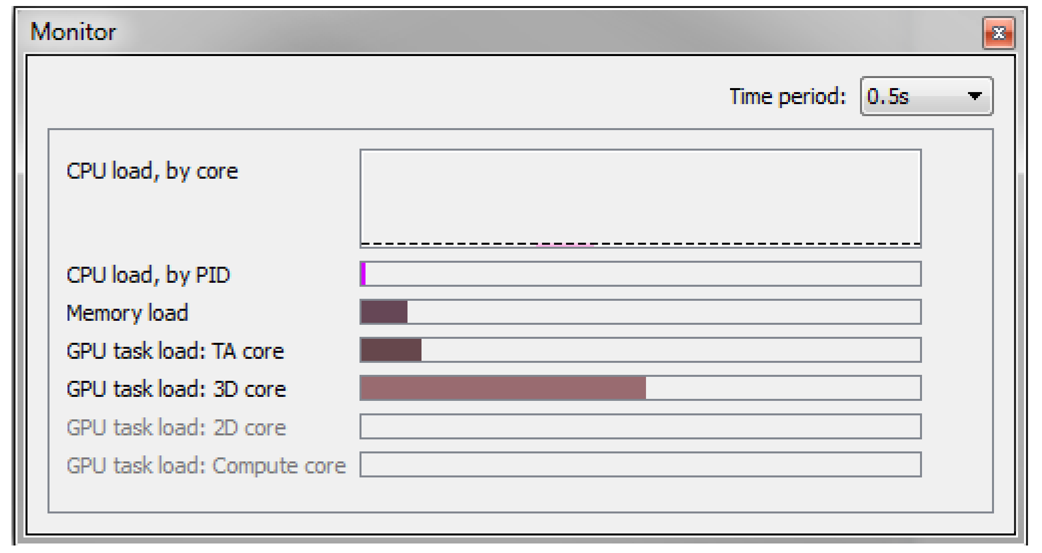


Figure 3-13 Monitor

3.4.10. PID

The PID section of the PVRTune GUI (as shown in Figure 3-14 PID) displays a list of the individual connections being used in the recording. It only lists the known PIDs relevant to the operation, and not all the PIDs running on the system.

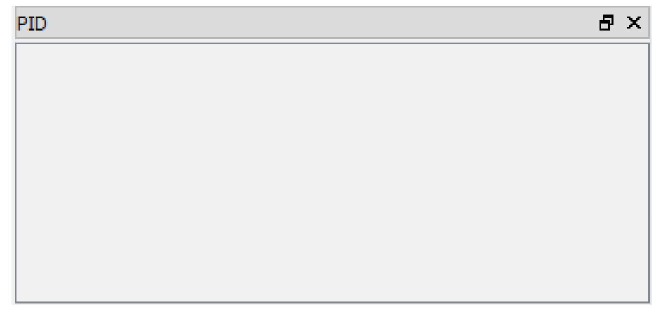
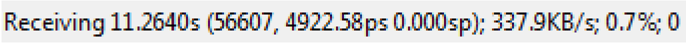


Figure 3-14 PID

3.5. Status Bar

The Status Bar is split into eight sections, from left to right these sections are:

- Status: The current status of PVRTune. Available options are; ‘Connecting’, ‘Receiving’, or ‘Disconnected’.
- Time: The amount of time PVRTune has been receiving data, or the amount of time PVRTune was connected (if current status is disconnected).
- Count: The number of times PVRTune has performed an action related to its current status (e.g. If PVRTune is receiving, the number of times it has received data from PVRPerfServer).
- Count/s: The rate at which ‘Count’ moves per second.
- Gap: The average time period between actions as measured by ‘Count’.
- RcvRate: The rate at which PVRTune is receiving data.
- DC%: The percentage progress towards automatic disconnect due to memory usage (after PVRTune has recorded 1GB of data, it disconnects from PVRPerfServer).
- Group: The currently active counter group.



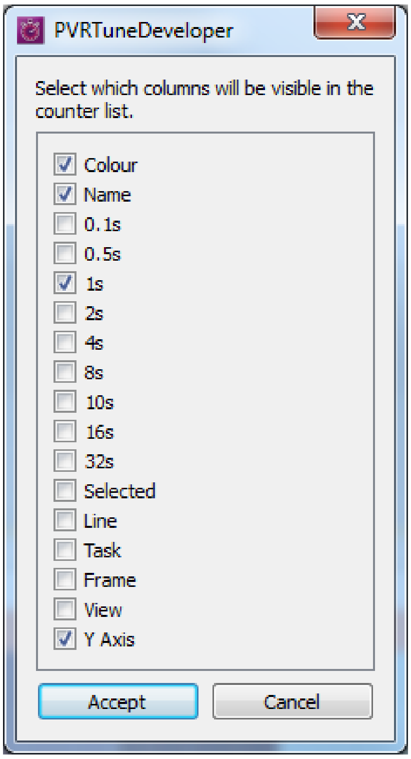
Receiving 11.2640s (56607, 4922.58ps 0.000sp); 337.9KB/s; 0.7%; 0

Figure 3-15 Status Bar

3.6. Dialogs

3.6.1. Select Columns

The `'Select Columns'` dialog is used to select the columns that appear in the Counter Table and is accessed by right clicking on said list; by default `'#'`, `'name'` and `'y axis'` are ticked.



PVRTuneDeveloper

Select which columns will be visible in the counter list.

☑ Colour
☑ Name
☐ 0.1s
☐ 0.5s
☑ 1s
☐ 2s
☐ 4s
☐ 8s
☐ 10s
☐ 16s
☐ 32s
☐ Selected
☐ Line
☐ Task
☐ Frame
☐ View
☑ Y Axis

Figure 3-16 Select Columns

The following options are also available:

Time Frames

The `'0.1s'` to `'32s'` columns represent time frames. These columns show the average value of counters over the most recent amount of time (i.e. the `'32s'` column gives the average of the last 32 seconds).

Selected

It is possible to select a region (time range) of the graph by holding 'Ctrl', then clicking and dragging (see Time-range Selection). This column shows the average over the selected region.

Line

This column responds to mouse hover; it shows the values of counters at the time associated with the position of the mouse in the Graph View.

Task

This column responds to mouse hover; it shows the average value of counters over the time frame of a single TA or 3D task.

Frame

This column responds to mouse hover; it shows the average value of given counters over the time frame of a single frame.

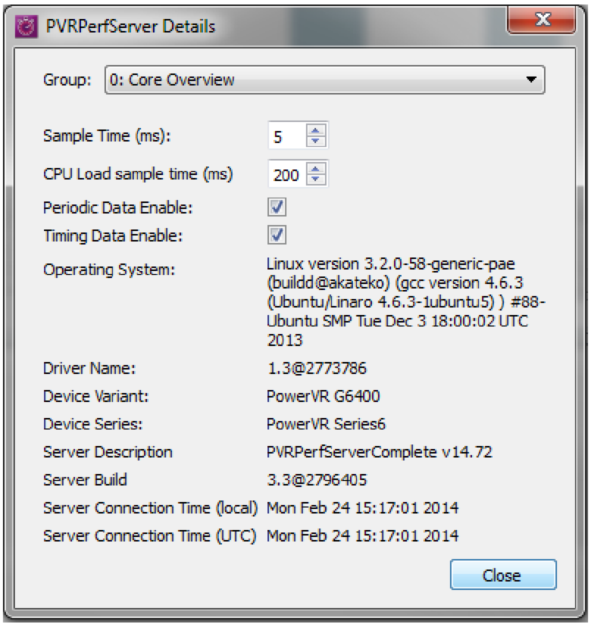
View

This column responds to mouse hover; it shows the average value of counters over the time frame of the graph the mouse is currently over.

3.6.2. PVRPerfServer Details

The PVRPerfServer details dialog is available in Connection > PVRPerfServer.

When a PVRTune is connected with PVRPerfServer, this dialog allows PVRTune to remotely control the counter groups that are currently active on PVRPerfServer; this in turn then activates or deactivates certain counters on the PowerVR hardware. A full list of the available counters and the groups they exist in can be found in Appendix A. Counter List.



Group: 0: Core Overview

Sample Time (ms): 5

CPU Load sample time (ms) 200

Periodic Data Enable: ☑

Timing Data Enable: ☑

Operating System: Linux version 3.2.0-58-generic-pae (buildd@akateko) (gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)) #88-Ubuntu SMP Tue Dec 3 18:00:02 UTC 2013

Driver Name: 1.3@2773786

Device Variant: PowerVR G6400

Device Series: PowerVR Series6

Server Description PVRPerfServerComplete v14.72

Server Build 3.3@2796405

Server Connection Time (local) Mon Feb 24 15:17:01 2014

Server Connection Time (UTC) Mon Feb 24 15:17:01 2014

Figure 3-17 PVRPerfServer Details dialog

Sample Time

The figure in this box represents the time between the hardware samples taken by PVRPerfServer.

CPU Load Sample Time

The figure in this box represents the time between the CPU Load samples taken by PVRPerfServer.

Periodic Data Enable

When this option is ticked PVRTune receives counter updates much quicker than otherwise. This option is on by default.

Timing Data Enable

When this option is ticked PVRTune receives TA/3D timing data. This option is enabled by default.

Server Information

The Details also include information about the target device; the operating system of the target device, the version of the PowerVR driver being used, the device name, and the device's revision number.

3.7. Menus

3.7.1. File

Clicking `'File'` opens the File Menu.

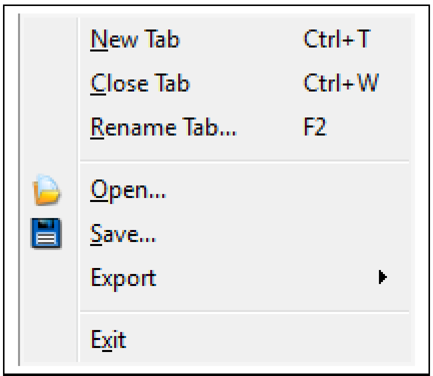


Figure 3-18 File Menu

New Tab

Opens up a new tab in the Timeline.

Close Tab

Closes the current tab in the Timeline.

Rename Tab…

Renames the current tab in the Timeline.

Open…

Opens a .pvrtune file. This option is only available from the Connection Window.

Save…

Saves the current tune data to a .pvrtune file.

Exit

Closes PVRTune.

3.7.2. Edit

Received Timing Data

Specify Clock Speed…

PVRTune calculates the device clock speed dynamically as data is received. This command overrides this calculation with a fixed clock speed.

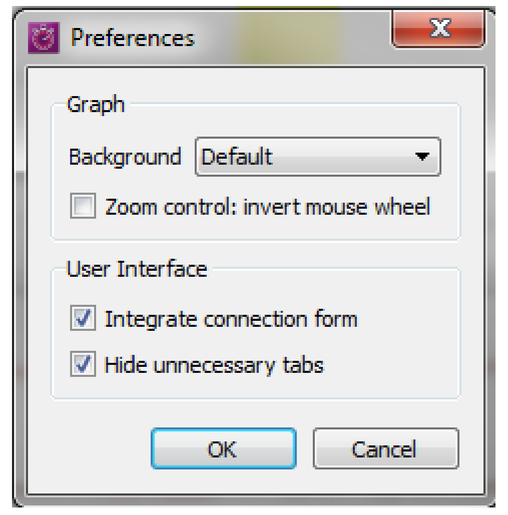
Note: If the clock speed varies during data collection, this option results in inaccurate data being collected.

Layout timing data evenly

The default behaviour; select this to reapply it.

Preferences…

Select to bring up the preferences dialog box, as shown in Figure 3-19. This dialog permits you to change the background colour of the timeline from default to light or dark to make it easier for you to see the lines. The mouse wheel can also be inverted in regards to the Zoom controls by ticking the box beneath. Unchecking the 'Integrate connection form' and 'Hide unnecessary tabs' box alters the aesthetics of the connectivity screen, making the connection form a pop up window, and keeping the tabs visible during the connection process.



Background: Default

☐ Zoom control: invert mouse wheel

☑ Integrate connection form

☑ Hide unnecessary tabs

Figure 3-19 Preferences Dialog

3.7.3. View

Clicking `'View'` opens the View Menu.

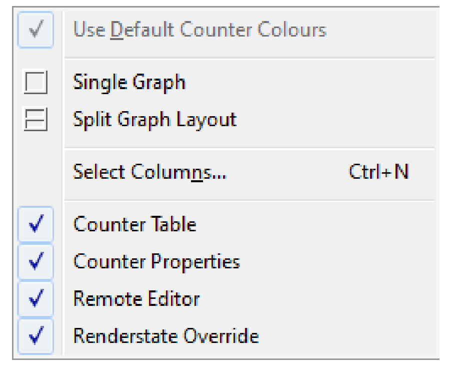


Figure 3-20 View Menu

Find…

Shows and brings to the foreground the Search box.

Use Default Counter Colours

This option informs PVRTune that you wish to use the default colour scheme; this is enabled by default.

Graphing Styles

These two options change the layout of the graph view.

Select Columns…

Opens the Select Columns dialog.

Show Counter Table

Toggles displaying of the Counter Table.

Show Counter Properties

Toggles displaying of the Counter Properties

Show Renderstate Overrride

Toggles displaying of the Renderstate Override.

Show Monitor

Toggles displaying the Monitor.

Show PID

Toggles displaying the PID.

Show Search

Toggles the displaying of the Search bar.

Show Remote Editor

Toggles displaying of the Remote Editor.

3.7.4. Connection Menu

Clicking `'Connection'` opens the Connection menu.

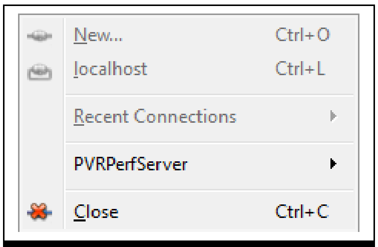


Figure 3-21 Connection Menu

New

`'New'` brings up a dialog box; enter an IP address or an IP resolvable name of a target device into the box and press `'Connect'` to connect to the targeted device. This option is only available on the Connection Window.

Localhost

This option connects PVRTune to an instance of PVRPerfServer running on the same device PVRTune is running on. This option is only available on the Connection Window.

Recent Connections

This function lists the IP addresses of the devices that have most recently been connected, note this is similar to the Recently Used Connections box in the Connection Window. It is possible to connect to a device by clicking the device's IP address in the list. This option is only available in the Connection Window.

PVRPerfServer

Details…

Displays the PVRPerfServer Details dialog (see Section 0).

Send Quit Message

This sends a message to PVRPerfServer which requests the process to exit.

Close

The `'Close'` option first closes the currently open connection to PVRPerfServer; choosing it a second time discards the open data.

If PVRTune is still receiving data, when first pressed it does not return PVRTune to the Connection Window but only closes the connection, this allows the data retained so that the user can save the tune to a file or continue analysis of it.

If this option is chosen when PVRTune is already disconnected this returns the user to the Connection Window.

3.7.5. Help

Clicking `'Help'` opens the Help Menu.

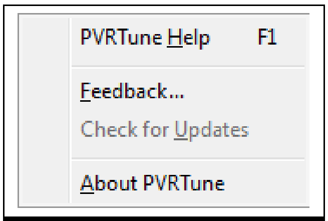


Figure 3-22 Help Menu

PVRTune Help

`'PVRTune Help'` opens this document.

Feedback

When this item is clicked a box appears containing contact information for submitting feedback.

Check for Updates

As of SDK release 3.0 PVRTuneDeveloper can auto-update. `'Check for Updates'` is used to force an update.

About PVRTune

When this item is clicked a box appears containing the `'About'` information for the program, including its version and revision numbers.

4. Analysing an Application

Analysing an application with PVRTune can aid in the following:

- Improved frame rate to increase user enjoyment and application responsiveness.
- Reduced render time without increased frame rate, in order to gain more idle time and thereby save power.
- Increased visual quality without sacrificing frame rate.

The following instructions are written with the aim of improving frame rate.

4.1. Connected Analysis

In order to analyse a device which is connected using PVRPerfServer, you should:

1. Ensure the target device is using the PowerVR device drivers.
2. Boot the target device.
3. Install PVRPerfServer on the target device.
4. If necessary, initialise the PowerVR device drivers.
5. Run PVRPerfServer – If successful PVRPerfServer outputs the server's name, IP address and port.
6. Run PVRTune on the host machine – it is important that both PVRTune and PVRPerfServer have matching versions.
7. If PVRPerfServer is on the same subnet as PVRTune, the target device appears in the Broadcasting Servers panel of the Connection Window; if not, enter the IP address of the server and click the `'Go'` button. PVRTune connects.
8. Once successfully connected, identify the bottleneck in your application by analysing the output of PVRTune (for more information, see Section 4.4).
9. Attempt to fix this bottleneck – for more information on optimisation techniques see the "PowerVR Performance Recommendations" document.
10. Repeat these steps with the newly optimised application until performance is at the required level, or no further bottlenecks can be identified.

4.2. Offline Analysis

4.2.1. Overview

It is possible to analyse an application without being directly connected to PVRPerfServer. This requires a .pvrtune file of the target application to have been created prior to analysis. This is particularly useful when large amounts of data are being lost due to network load, or high CPU usage on the client machine.

4.2.2. Creating a .pvrtune File

.pvrtune files can be created either through the use of the `'-sendto='` command-line parameter for PVRPerfServer, or by saving a .pvrtune file of an existing trace from the `'File -> Save'` menu within the PVRTune GUI.

It should be noted that PVRPerfServer, only saves counter data for a single counter group when using the `'-sendto='` parameter. By default the group is set to zero, this can be changed with the `'-group='` command-line parameter.

4.2.3. Instructions

With a .pvrtune file created and copied to an accessible location, perform the following steps:

1. Identify the version of PVRTune/PVRPerfServer used to create the .pvrtune file.
2. Run the version of PVRTune that matches the identified version.
3. Click on `'File -> Open'`; select the file that you wish to view and click `'Open'`.
4. PVRTune displays the tuning data.
5. Identify the bottleneck (see Section 4.4 The Five Common Bottlenecks for more information).
6. Attempt to handle this bottleneck – see the document “PowerVR Performance Recommendations” for more information on optimisation techniques.
7. Repeat these steps with a recording of the newly optimised application until performance is at the required level, or no further bottlenecks can be identified.

4.3. Software Counters

Software counters provide data on certain statistics within an application being profiled by PVRTrace.

4.3.1. Enabling Software Counters

Android

Open PVRHub, on the Options screen, ensure ‘Enable software counters’ is enabled. The PVRTrace libraries must be installed on the device for this to work.

Linux

With PVRHub installed run `"pvr_profile <binary>"` where `<binary>` is the application you wish to profile with software counters active.

Neutrino and Windows

In the `pvrtrace.cfg` file on the device, ensure the `'SoftwareCounters'` flag is set to `'=1'` and `'Profile'` is set to `'=1'`.

4.3.2. List of Software Counters

The following software counters are available from the PVRTrace libraries:

- Indexed draw calls
- Non-indexed draw calls
- Points no.
- Line no. (list)
- Line no. (loop)
- Line no. (strip)
- Total line no.
- Triangle no. (list)
- Triangle no. (strip)
- Triangle no. (fan)
- Total triangle no.
- Texture uploads
- Texture modifications
- Scissor calls
- Viewport calls
- Frame buffer accesses

- Vertex shader compiles
- Fragment shader compiles
- Program links
- Framebuffer access (bytes)
- Texture uploads (bytes)
- Texture modifications (bytes)
- Buffer object uploads (bytes)
- Buffer object modifications (bytes)
- Uniform uploads
- Context binds

4.4. The Five Common Bottlenecks

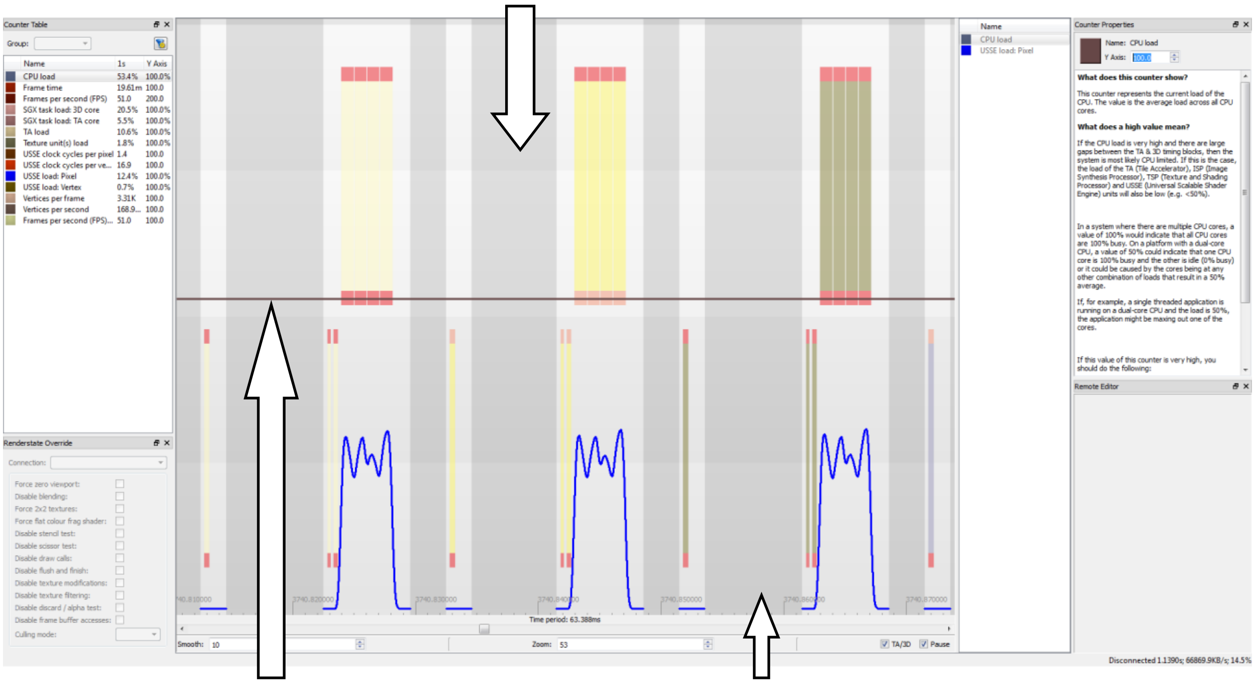
When analysing the performance of an application, bottlenecks usually fall into one of five categories:

- CPU limited,
- Vertex limited,
- V-Sync limited,
- Fragment limited, or
- Bandwidth limited.

4.4.1. CPU Limited

A CPU limited application is often identifiable as an application suffering from poor performance or frame rate even though the GPU usage is not high. In PVRTune this can be very easily identified; CPU limited applications have a CPU load that is either, at or near one hundred percent; or is wildly variable (as seen in Figure 4-1).

Gaps between TA & 3D Tasks



Counter Table

Group:

	Name	1s	Y Axis
■	CPU load	53.4%	100.0%
■	Frame time	19.61m	100.0
■	Frames per second (FPS)	51.0	200.0
■	SGX task load: 3D core	20.5%	100.0%
■	SGX task load: TA core	5.5%	100.0%
■	TA load	10.6%	100.0%
■	Texture unit(s) load	1.8%	100.0%
■	USSE clock cycles per pixel	1.4	100.0
■	USSE clock cycles per ve...	16.9	100.0
■	USSE load: Pixel	12.4%	100.0%
■	USSE load: Vertex	0.7%	100.0%
■	Vertices per frame	3.31K	100.0
■	Vertices per second	168.9...	100.0
■	Frames per second (FPS)...	51.0	100.0

Renderstate Override

Connection:

Force zero viewport: ☐
Disable blending: ☐
Force 2x2 textures: ☐
Force flat colour frag shader: ☐
Disable stencil test: ☐
Disable scissor test: ☐
Disable draw calls: ☐
Disable flush and finish: ☐
Disable texture modifications: ☐
Disable texture filtering: ☐
Disable discard / alpha test: ☐
Disable frame buffer accesses: ☐
Culling mode:

Time period: 63.388ms

Smooth: 10

Zoom: 53

☑ TA/3D ☑ Pause

Name
■ CPU load
■ USSE load: Pixel

Counter Properties

Name: CPU load
Y Axis: 100.0

What does this counter show?

This counter represents the current load of the CPU. The value is the average load across all CPU cores.

What does a high value mean?

If the CPU load is very high and there are large gaps between the TA & 3D timing blocks, then the system is most likely CPU limited. If this is the case, the load of the TA (Tile Accelerator), ISP (Image Synthesis Processor), TSP (Texture and Shading Processor) and USSE (Universal Scalable Shader Engine) units will also be low (e.g. <50%).

In a system where there are multiple CPU cores, a value of 100% would indicate that all CPU cores are 100% busy. On a platform with a dual-core CPU, a value of 50% could indicate that one CPU core is 100% busy and the other is idle (0% busy) or it could be caused by the cores being at any other combination of loads that result in a 50% average.

If, for example, a single threaded application is running on a dual-core CPU and the load is 50%, the application might be maxing out one of the cores.

If this value of this counter is very high, you should do the following:

Remote Editor

Disconnected 1.1390s; 66869.9KB/s; 14.5%

CPU Load hitting maximum on one or more processors.

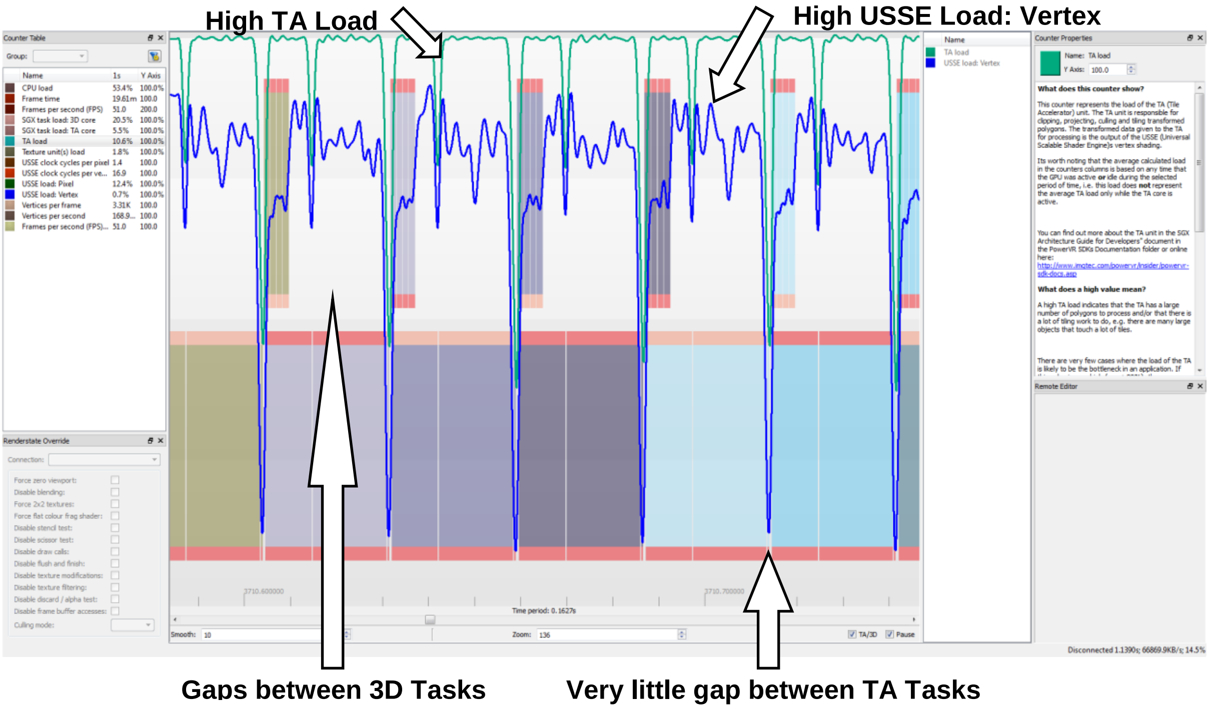
Gaps in the USSE Load

Figure 4-1 CPU Limited

Other identifying factors include gaps in the USSE Load, caused by the PowerVR hardware going to sleep while waiting for further instructions (see the shaded grey areas in Figure 4-1); and regular visible gaps between frames when displaying Timing Data.

4.4.2. Vertex Limited

Vertex limited applications are applications where the bottleneck comes from processing either large amounts of vertices per frame, or from the use of a complex vertex shader (or both). This can be identified by large gaps between 3D tasks while there is little or no gap between TA tasks.



Name	1s	Y Axis
CPU load	53.4%	100.0%
Frame time	19.61m	100.0
Frames per second (FPS)	51.0	200.0
SGX task load: 3D core	20.5%	100.0%
SGX task load: TA core	5.5%	100.0%
TA load	10.6%	100.0%
Texture unit(s) load	1.8%	100.0%
USSE clock cycles per pixel	1.4	100.0
USSE clock cycles per ve...	16.9	100.0
USSE load: Pixel	12.4%	100.0%
USSE load: Vertex	0.7%	100.0%
Vertices per frame	3.31K	100.0
Vertices per second	168.9...	100.0
Frames per second (FPS)...	51.0	100.0

Name: TA load
Y Axis: 100.0

What does this counter show?

This counter represents the load of the TA (Tile Accelerator) unit. The TA unit is responsible for clipping, projecting, culling and tiling transformed polygons. The transformed data given to the TA for processing is the output of the USSE (Universal Scalable Shader Engine)s vertex shading.

Its worth noting that the average calculated load in the counters columns is based on any time that the GPU was active **or** idle during the selected period of time, i.e. this load does **not** represent the average TA load only while the TA core is active.

You can find out more about the TA unit in the SGX Architecture Guide for Developers" document in the PowerVR SDKs Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high TA load indicates that the TA has a large number of polygons to process and/or that there is a lot of tiling work to do, e.g. there are many large objects that touch a lot of tiles.

There are very few cases where the load of the TA is likely to be the bottleneck in an application. If

Figure 4-2 Vertex Limited

Further information can be gained from the ‘USSE Load: Vertex’ and the ‘TA Load’ counters:

- If ‘TA Load’ is high but ‘USSE Load: Vertex’ is not then the scene has too many vertices in it and the cost is coming from the tiling process.
- If the ‘USSE Load: Vertex’ is high but ‘TA Load’ is not, then the bottleneck is likely to be in the vertex shader.

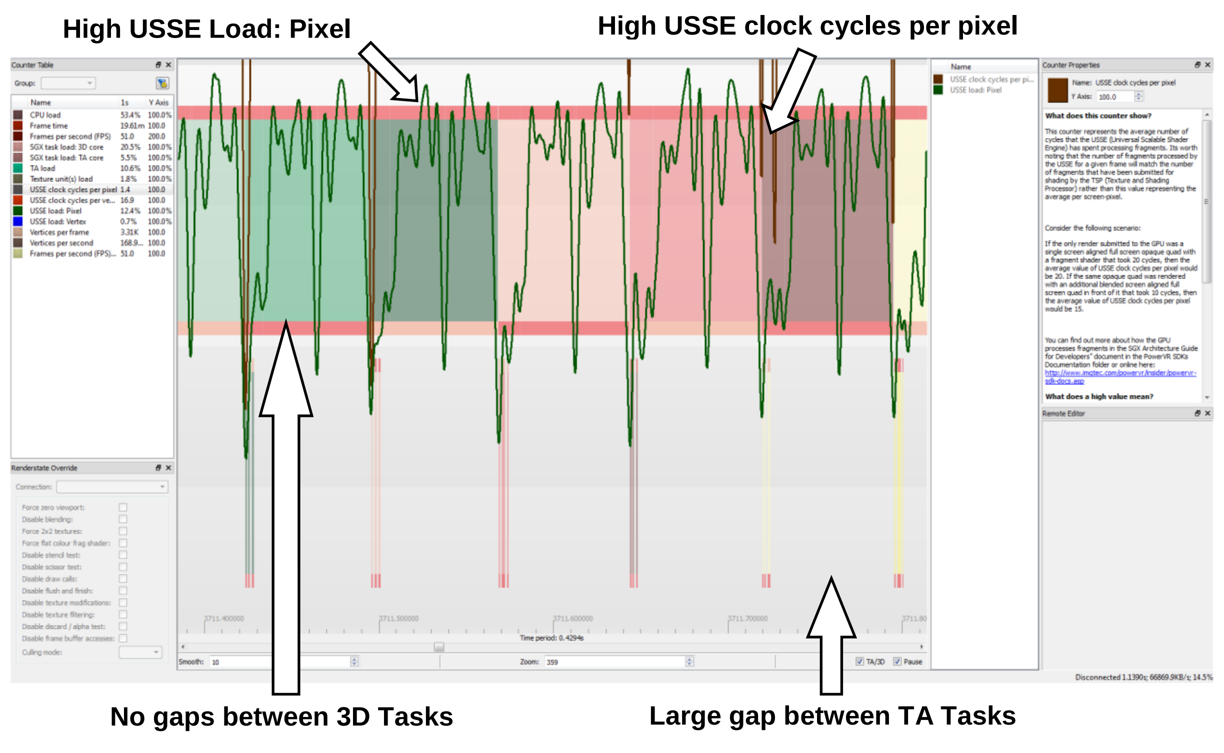
4.4.3. V-Sync Limited

V-Sync (short for Vertical Synchronisation) is a display option that forces an application to synchronise graphical updates with the update rate of the screen. This causes some frames to be slightly delayed and enforces a maximum refresh rate, but reduces screen tearing, and may save power. V-Sync limited applications are often characterised by intermittent gaps between frames in the Graph View, and the Frame Rate appears to be limited at a set maximum value.

If possible, V-sync should be disabled when profiling an application as it effectively adds noise to the PVRTune output and this makes it more difficult to diagnose where optimisation work could be beneficial and if completed optimisation has been successful.

4.4.4. Fragment Limited

Fragment limited applications are very common; most scenes have fewer vertices than the number of pixels in the frame buffer.



	Name	1s	Y Axis
	CPU load	53.4%	100.0%
	Frame time	19.61m	100.0
	Frames per second (FPS)	51.0	200.0
	SGX task load: 3D core	20.5%	100.0%
	SGX task load: TA core	5.5%	100.0%
	TA load	10.6%	100.0%
	Texture unit(s) load	1.8%	100.0%
	USSE clock cycles per pixel	1.4	100.0
	USSE clock cycles per ve...	16.9	100.0
	USSE load: Pixel	12.4%	100.0%
	USSE load: Vertex	0.7%	100.0%
	Vertices per frame	3.31K	100.0
	Vertices per second	168.9...	100.0
	Frames per second (FPS)...	51.0	100.0

Force zero viewport: ☐
Disable blending: ☐
Force 2x2 textures: ☐
Force flat colour frag shader: ☐
Disable stencil test: ☐
Disable scissor test: ☐
Disable draw calls: ☐
Disable flush and finish: ☐
Disable texture modifications: ☐
Disable texture filtering: ☐
Disable discard / alpha test: ☐
Disable frame buffer accesses: ☐
Culling mode:

Name: USSE clock cycles per pixel
Y Axis: 100.0

What does this counter show?

This counter represents the average number of cycles that the USSE (Universal Scalable Shader Engine) has spent processing fragments. Its worth noting that the number of fragments processed by the USSE for a given frame will match the number of fragments that have been submitted for shading by the TSP (Texture and Shading Processor) rather than this value representing the average per screen-pixel.

Consider the following scenario:

If the only render submitted to the GPU was a single screen aligned full screen opaque quad with a fragment shader that took 20 cycles, then the average value of USSE clock cycles per pixel would be 20. If the same opaque quad was rendered with an additional blended screen aligned full screen quad in front of it that took 10 cycles, then the average value of USSE clock cycles per pixel would be 15.

You can find out more about how the GPU processes fragments in the SGX Architecture Guide for Developers" document in the PowerVR SDKs Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

Disconnected 1.1390s; 66869.9KB/s; 14.5%

Figure 4-3 Fragment Limited

The primary methods of identifying a fragment limited application are:

- No gaps between 3D tasks.
- Large gaps between TA tasks.
- A high value of ‘USSE Load: Pixel’.
- High USSE clock cycles per pixel.

4.4.5. Bandwidth Limited

Cases of bandwidth limitation are both hard to visualise and hard to identify as they may appear as other bottlenecks. Programs may be bandwidth limited if:

- ‘TA/3D’ shows the application to be fragment limited but the ‘USSE Load: Pixel’ is low.
- ‘TA/3D’ shows the application to be vertex limited but the ‘USSE Load: Vertex’ and ‘TA load’ are low.

Other instances of bandwidth limitation may occur: if many units are accessing memory simultaneously then available system memory bandwidth limits can slow all operations on the hardware. This is platform specific and, as such, there is no counter to record it. As a rule of thumb, action should always be taken to reduce bandwidth use whenever possible through the correct use of texture compression, mesh optimisation, avoiding unnecessary texture reads etc. as bandwidth is always limited.

Note: Bandwidth in system-on-chip (SoC) devices is shared amongst all components of the chip. Non-GPU areas of the chip using large amounts of bandwidth may still cause application graphics to be bandwidth limited.

5. Related Materials

Software

- PVRScope
- PVRTrace

Documentation

- PVRScope User Manual
- PVRTrace User Manual
- PowerVR Performance Recommendations

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 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. Counter List

A.1. CPU Load

Tooltip

The percentage of time that the CPU is busy. The value is the average load across all CPU cores.

What does this counter show?

This counter represents the current load of the CPU. The value is the average load across all CPU cores.

What does a high value mean?

If the CPU load is very high and there are large gaps between the TA & 3D timing blocks, then the system is most likely CPU limited. If this is the case, the load of the TA (Tile Accelerator), ISP (Image Synthesis Processor), TSP (Texture and Shading Processor) and USSE (Universal Scalable Shader Engine) units will also be low (e.g. <50%).

In a system where there are multiple CPU cores, a value of 100% would indicate that all CPU cores are 100% busy. On a platform with a dual-core CPU, a value of 50% could indicate that one CPU core is 100% busy and the other is idle (0% busy) or it could be caused by the cores being at any other combination of loads that result in a 50% average.

If, for example, a single threaded application is running on a dual-core CPU and the load is 50%, the application might be maxing out one of the cores.

If this value of this counter is very high, you should do the following:

1. **Run a CPU profiler:** Run a CPU profiler to investigate the issue further.

A value beyond 100% could be caused by PVRPerfServer being unable to query the correct number of cores from the target platform or it may be caused by sampling at insufficient frequency.

A.2. Frame Time

Tooltip

The average time it has taken the GPU to process a frame (in milliseconds).

What does this counter show?

This counter represents the average time it has taken the GPU to process a frame (in milliseconds) over the selected period. The frame time is calculated based on time the GPU is idle and active (TA & 3D time).

It’s worth noting that this value is not calculated per CPU process, i.e. if there are two processes using the GPU to render, the average frame-time over the course of a second is the average frame-time of both processes during that period of time.

What does a high value mean?

The larger this value is, the longer it takes on average for the GPU to render a frame. It’s worth noting that this average includes all processes utilising the GPU during the selected time period so if a second process is running alongside the application you want to profile, e.g. OS composition, then this affects the average frame time value.

A.3. Frames per Second (FPS)

Tooltip

The average number of frames-per-second processed by the GPU.

What does this counter show?

This counter represents the average number of frames-per-second processed by the GPU over the selected period of time. This value is the reciprocal of the frame time (in seconds).

Similarly to the frame time counter, this value is **not** calculated per CPU process. This counter effectively represents the number of frames the GPU has been able to render within the selected period of time, regardless of which process submitted the rendering task.

What does a high value mean?

A high value indicates that the GPU has been able to process a high number of frames during the selected period of time

A.4. ISP Load

Tooltip

The load of the GPU's Image Synthesis Processor (ISP) unit.

What does this counter show?

This counter represents the average load of the GPU's Image Synthesis Processor (ISP). This unit is responsible for executing the per-tile Hidden Surface Removal (HSR). It also performs the depth and stencil operations for the tile using the GPU's on-chip memory.

As this unit is responsible for HSR, it's worth noting that the number of fragments processed by the ISP is the amount of fragments that have been submitted by the application before the GPU performs overdraw reduction. It's also worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average "ISP load" only while the 3D core is active.

You can find out more about the ISP unit in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high load value indicates that the ISP has been given a very large number of fragments to process. There are very few cases where the load of the ISP is likely to be the bottleneck in an application. If this value is very high (e.g. >80%), then you should try the following to reduce the ISP load:

1. **Improve CPU object culling within the view frustum:** As previously mentioned, the number of fragments processed by the ISP is affected by the number of fragments submitted by applications. Improving CPU culling of objects so there are fewer draw calls within the view frustum reduces the ISP load as fewer fragments is submitted to the GPU.
2. **Utilise back-face culling:** Enabling back-face culling allows the TA (Tile Accelerator) to cull more polygons, which reduces the amount of data that writes to the Parameter Buffer. This will, in turn, reduce the number of primitives that the ISP has to fetch and process from the Parameter Buffer, therefore decreasing the ISP unit's workload.

A.5. On-Screen Primitives per Frame

Tooltip

Number of primitives that pass the TA (Tile Accelerator)'s clipping and culling, per-frame.

What does this counter show?

"On-screen primitives" are primitives that have passed the TA (Tile Accelerator)'s clipping and culling and are subsequently written to the Parameter Buffer. This counter shows the average number of on-screen primitives that have been processed per-frame by the GPU over the selected time period. It should be noted that this average value is calculated as the number of on-screen primitives per-frame that have been processed by the GPU during the selected time period and it is not PID specific.

You can find out more about the TA and Parameter Buffer in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

As this counter does not represent a load of a GPU unit, it's difficult to determine anything from this value alone.

If the TA time is longer than expected, the "TA load" and "USSE load: vertex" are high (e.g. >60%) and the "USSE clock cycles per vertex" is relatively low, then the application may be bottlenecked by the number of on-screen primitives that are being processed. If this is a problem, then the following should be done to avoid this bottleneck:

1. **Improve CPU culling within the view frustum:** Improving CPU culling of objects within the view frustum so there are fewer draw calls will reduce the number of on-screen primitives that the GPU will have to process.
2. **Reduce polygon count/tessellation factor:** Reducing the number of polygon's that are submitted within the view frustum may reduce the number of primitives that are processed by the GPU.
3. **Improve triangle sorting in geometry:** The rendering order of triangles that make up a given object (as specified by an application when the data is uploaded to the driver) can affect the number of primitives that the GPU uses internally to render the object. Improving the sorting of triangles within an object so that triangles near each other are rendered sequentially can decrease the number of on-screen primitives.

A.6. On-screen primitives per second

Tooltip

Number of primitives that pass the TA (Tile Accelerator)'s clipping and culling, per-second.

What does this counter show?

"On-screen primitives" are primitives that have passed the TA (Tile Accelerator)'s clipping and culling and are subsequently written to the Parameter Buffer. This counter shows the average number of on-screen primitives that have been processed per-second by the GPU over the selected time period. It should be noted that this average value is calculated as the number of on-screen primitives per-second that have been processed by the GPU during the selected time period and it is not PID specific.

You can find out more about the TA and Parameter Buffer in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

As this counter does not represent a load of a GPU unit, it's difficult to determine anything from this value alone.

If the TA time is longer than expected, the "TA load" and "USSE load: vertex" are high (e.g. >60%) and the "USSE clock cycles per vertex" is relatively low, then the application may be bottlenecked by the number of on-screen primitives that are being processed. If this is a problem, then the following should be done to avoid this bottleneck:

1. **Improve CPU culling within the view frustum:** Improving CPU culling of objects within the view frustum so there are fewer draw calls will reduce the number of on-screen primitives that the GPU will have to process.
2. **Reduce polygon count/tessellation factor:** Reducing the number of polygon's that are submitted within the view frustum may reduce the number of primitives that are processed by the GPU.
3. **Improve triangle sorting in geometry:** The rendering order of triangles that make up a given object (as specified by an application when the data is uploaded to the driver) can affect the number of primitives that the GPU uses internally to render the object. Improving the sorting of triangles within an object so that triangles near each other are rendered sequentially can decrease the number of on-screen primitives.

A.7. On-Screen Vertices per Frame

Tooltip

Number of vertices that pass the TA (Tile Accelerator)'s clipping and culling, per-frame.

What does this counter show?

"On-screen vertices" are vertices that have passed the TA (Tile Accelerator)'s clipping and culling and are subsequently written to the Parameter Buffer. This counter shows the average number of on-screen vertices that have been processed per-frame by the GPU over the selected time period. It should be noted that this average value is calculated as the number of on-screen vertices per-frame that have been processed by the GPU during the selected time period and it is not PID specific.

You can find out more about the TA and Parameter Buffer in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

As this counter does not represent a load of a GPU unit, it's difficult to determine anything from this value alone.

If the TA time is longer than expected, the "TA load" and "USSE load: vertex" are high (e.g. >60%) and the "USSE clock cycles per vertex" is relatively low, then the application may be bottlenecked by the number of on-screen vertices that are being processed. If this is a problem, then the following should be done to avoid this bottleneck:

1. **Improve CPU culling within the view frustum:** Improving CPU culling of objects within the view frustum so there are fewer draw calls will reduce the number of on-screen vertices that the GPU will have to process.
2. **Reduce polygon count/tessellation factor:** By reducing the number of polygon's that are submitted within the view frustum, the GPU will have fewer on-screen vertices to process.
3. **Improve triangle sorting in geometry:** The rendering order of triangles that make up a given object (as specified by an application when the data is uploaded to the driver) can affect the number of vertices that the GPU uses internally to render the object. Improving the sorting of

triangles within an object so that triangles near each other are rendered sequentially can decrease the number of on-screen vertices.

A.8. On-Screen Vertices per Primitive

Tooltip

Measures the average efficiency of vertex sharing.

What does this counter show?

“on-screen vertices” are vertices that have passed the TA (Tile Accelerator)’s clipping and culling and are subsequently written to the Parameter Buffer.

This counter measures the efficiency of vertex sorting on average during the selected time period. Polygon sorting can affect the performance of vertex transformations. It should be noted that this average value is calculated as the number of on-screen vertices per-primitive that have been processed by the GPU during the selected time period and it is not PID specific.

You can find out more about the TA and Parameter Buffer in the “Architecture Guide for Developers" document in the PowerVR SDK’s Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that, on average, the GPU isn’t sharing vertices between polygons as efficiently as it could do.

If you suspect this is causing a bottleneck in your application, you should do the following:

1. **Improve triangle sorting in geometry:** The rendering order of triangles that make up a given object (as specified by an application when the data is uploaded to the driver) can affect the number of vertices that the GPU uses internally to render the object. Improving the sorting of triangles within an object so that triangles near each other are rendered sequentially can decrease the number of on-screen vertices per-primitive.

A.9. On-screen vertices per second

Tooltip

Number of vertices that pass the TA (Tile Accelerator)’s clipping and culling, per-second.

What does this counter show?

“On-screen vertices” are vertices that have passed the TA (Tile Accelerator)’s clipping and culling and are subsequently written to the Parameter Buffer. This counter shows the average number of on-screen vertices that have been processed per-second by the GPU over the selected time period. It should be noted that this average value is calculated as the number of on-screen vertices per-second that have been processed by the GPU during the selected time period and it is not PID specific.

You can find out more about the TA and Parameter Buffer in the “Architecture Guide for Developers" document in the PowerVR SDK’s Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

As this counter does not represent a load of a GPU unit, it’s difficult to determine anything from this value alone.

If the TA time is longer than expected, the “TA load” and “USSE load: vertex” are high (e.g. >60%) and the “USSE clock cycles per vertex” is relatively low, then the application may be bottlenecked by the number of on-screen vertices that are being processed. If this is a problem, then the following should be done to avoid this bottleneck:

1. **Improve CPU culling within the view frustum:** Improving CPU culling of objects within the view frustum so there are fewer draw calls will reduce the number of on-screen vertices that the GPU will have to process.
2. **Reduce polygon count/tessellation factor:** By reducing the number of polygon’s that are submitted within the view frustum, the GPU will have fewer on-screen vertices to process.
3. **Improve triangle sorting in geometry:** The rendering order of triangles that make up a given object (as specified by an application when the data is uploaded to the driver) can affect the number of vertices that the GPU uses internally to render the object. Improving the sorting of triangles within an object so that triangles near each other are rendered sequentially can decrease the number of on-screen vertices.

A.10. Task Load: 2D Core

Tooltip

The load of the 2D core (if the target GPU contains one).

What does this counter show?

If the GPU in the target device contains a PowerVR 2D graphics core, this counter shows the load of this unit. If a PowerVR 2D core is not present in the target device the counter is disabled.

The purpose of the 2D core is to perform efficient blitting operations. As an example, OS composition may utilise the 2D core so that the 3D core can be dedicated to application rendering.

What does a high value mean?

A high value indicates that the 2D core has a large load during the selected time period. It’s worth noting that the counter load will always be 100% in the “task” column when a 2D task is highlighted in the TA/3D view and will be either 100% or 0% when the “line” column is selected. This is because at very small time periods the counter is effectively a Boolean value that shows the core as either active or idle.

If this average value is very high (e.g. >80%), then you should try the following to reduce the 2D core load:

1. **Only blend when necessary:** Removing unnecessary blending will reduce the 2D core’s load

A.11. Task Load: 3D Core

Tooltip

The load of the “3D core”. This includes ISP (Image Synthesis Processor), TSP (Texture and Shading Processor), USSE (Universal Scalable Shader Engine) fragment processing and related stall time.

What does this counter show?

This counter represents the load of all 3D core processing. The term “3D core” in PVRTune refers to any time that is spent processing fragments and shading them. This includes time that the ISP (Image Synthesis Processor), TSP (Texture and Shading Processor) and USSE (Universal Scalable Shader Engine) units have spent processing fragments. It also includes time that these units have been stalled.

You can find out more about these GPU units in the “Architecture Guide for Developers" document in the PowerVR SDK’s Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the 3D core has a large load during the selected time period. It’s worth noting that the counter load will always be 100% in the “task” column when a 3D task is highlighted in the TA/3D view and will be either 100% or 0% when the “line” column is selected. This is because at very small time periods the counter is effectively a Boolean value that shows the core as either active or idle.

A.12. Task Load: TA Core

Tooltip

The load of the “TA core”. This includes USSE (Universal Scalable Shader Engine) vertex processing, TA (Tile Accelerator) processing and related stall time.

What does this counter show?

This counter represents the load of all TA core processing. The term “TA core” in PVRTune refers to any time that is spent executing vertex shaders and binning the vertex data into tiles. This includes time that the USSE (Universal Scalable Shader Engine) and TA (Tile Accelerator) units have spent processing vertices and writing transformed vertex data and tile data into the Parameter Buffer. It also includes time that these units have been stalled.

You can find out more about these GPU units in the “Architecture Guide for Developers" document in the PowerVR SDK’s Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the TA core has a large load during the selected time period. It’s worth noting that the counter load will always be 100% in the “task” column when a 3D task is highlighted in the TA/3D view and will be either 100% or 0% when the “line” column is selected. This is because at very small time periods the counter is effectively a Boolean value that shows the core as either active or idle.

A.13. TA Load

Tooltip

The percentage of time that the TA (Tile Accelerator) unit is busy.

What does this counter show?

This counter represents the load of the TA (Tile Accelerator) unit. The TA unit is responsible for clipping, projecting, culling and tiling transformed polygons. The transformed data given to the TA for processing is the output of the USSE (Universal Scalable Shader Engine)’s vertex shading.

It’s worth noting that the average calculated load in the counter’s columns is based on any time that the GPU was active **or** idle during the selected period of time, i.e., this load does not represent the average “TA load” only while the TA core is active.

You can find out more about the TA unit in the “Architecture Guide for Developers" document in the PowerVR SDK’s Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high TA load indicates that the TA has a large number of polygons to process and/or that there is a lot of tiling work to do, e.g. there are many large objects that touch a lot of tiles.

There are very few cases where the load of the TA is likely to be the bottleneck in an application. If this value is very high (e.g. >80%), then you should try the following to reduce the TA load:

1. **Improve CPU object culling:** Improving CPU culling of objects so there are fewer draw calls will reduce the TA load as fewer polygons will be submitted to the GPU and tiling work may also reduce. This culling should, ideally, be applied to geometry inside the view frustum as well as outside of it.
2. **Reduce polygon count/tessellation factor:** Reducing the number of polygon's that are submitted to the GPU can reduce the TA load (less clipping and culling operations will be required). Polygon/tessellation reduction should, ideally, be applied to geometry inside the view frustum as well as outside of it.
3. **Avoid submitting unnecessary objects that touch a lot of tiles:** In very rare cases, an application may be bottlenecked by the TA updating the lists of all tiles that transformed geometry touches. If you suspect that this is the bottleneck, you should try to cull any unnecessary objects/primitives that have been submitted. If there are large objects in your scene that are mostly occluded (e.g. >80% is hidden), then it may be worth breaking the object down into smaller pieces so the occluded parts can be culled on the CPU before being submitted to the GPU.

A.14. Texture Unit Load

Tooltip

The percentage of time that the texture unit is busy.

What does this counter show?

This counter represents the load of the texture unit. In addition to fetching texture data from memory, the texture unit is responsible for calculating sample points based on texture filtering modes and perspective. It is also responsible for combining retrieved samples into a single colour value that can be given to the fragment that requested the texture fetch.

This counter only represents time that the texture unit is active and does not include any stall time that may have occurred. Additionally, this counter does not give any indication of the time it has taken to fetch texture data from cached or uncached memory. It's also worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time.

What does a high value mean?

A high value indicates that the texture unit has a lot of work to perform during the selected period of time.

If this value is very high (e.g. >80%), then you should try the following to reduce the texture unit load:

1. **Simplify texture filtering modes:** As the complexity of texture filtering increases for a given texture fetch, the workload of the texture unit increases. You can reduce the texture unit load by choosing simpler texture filtering modes. It's worth noting that the cost of nearest and bilinear sampling is very similar in the GPU texture units, so you may not see a workload reduction when changing between these two filtering modes.

A.15. TSP Load

Tooltip

The percentage of time that the TSP (Texture and Shading Processor) unit is busy.

What does this counter show?

This counter represents the load of the TSP (Texture and Shading Processor). The TSP has three main responsibilities; the first is to perform the interpolation of shader varyings to determine per-fragment values, the second is to request texture pre-fetches (when texture coordinates are known before fragment shader execution) and the third is to submit fragments to be shaded to the USSE (Universal Scalable Shader Engine).

It's worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average TSP load only while the 3D core is active.

You can find out more about the TSP unit in the “Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the TSP has either a large number of fragments to process or it is spending a large amount of time interpolating shader varyings.

If this value is very high (e.g. >80%), then you should try the following to reduce the TSP load:

1. **Reduce alpha blending/discard:** In an entirely opaque scene with a fixed resolution, the TSP will always be given the same number of fragments to process (this will match the tile resolution - a fixed value for a given PowerVR core). In tiles that contain fragments that are blended or use the discard keyword in their shaders, there may be more fragments to process. By ensuring rendering is done as opaque first, discard second (if required) and blending last, the number of fragments processed by the TSP will be kept to a minimum. Additionally, if the number of blended and discarded objects submitted to the GPU can be reduced by an application, then the number of fragments processed by the TSP will be lower.
2. **Reduce the number of shader varyings:** If you suspect that the TSP may be limited by the number of shader varying interpolations it is performing, you can reduce the unit's workload by using fewer varyings in your shaders.

A.16. USSE Clock Cycles per Pixel

Tooltip

The average number of clock cycles that the USSE (Universal Scalable Shader Engine) has spent processing pixels.

What does this counter show?

This counter represents the average number of cycles that the USSE (Universal Scalable Shader Engine) has spent processing fragments. It's worth noting that the number of fragments processed by the USSE for a given frame will match the number of fragments that have been submitted for shading by the TSP (Texture and Shading Processor) rather than this value representing the average per screen-pixel.

Consider the following scenario:

If the only render submitted to the GPU was a single screen aligned full screen opaque quad with a fragment shader that took 20 cycles, then the average value of “USSE clock cycles per pixel” would be 20. If the same opaque quad was rendered with an additional blended screen aligned full screen quad in front of it that took 10 cycles, then the average value of “USSE clock cycles per pixel” would be 15.

You can find out more about how the GPU processes fragments in the “Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the average number of cycles it takes to shade a fragment during the selected time period is high. As previously mentioned, the value of this counter can be a little misleading. It's best to rely on this counter in benchmarks where a single fragment shader is applied to a single full screen opaque quad to test the performance of the shader on the target device.

If this value is high, then you should do the following:

Profile with the offline GLSL compiler: Batch process your application's fragment shaders with the appropriate profiling compiler in the PowerVR SDK using the "-perfsim" flag. You can then use the profile output to isolate the most expensive shaders in your scene, which will help you identify where the best place is to focus your optimization.

A.17. USSE Clock Cycles per Vertex

Tooltip

The average number of clock cycles that the USSE (Universal Scalable Shader Engine) has spent processing vertex.

What does this counter show?

This counter represents the average number of cycles that the USSE (Universal Scalable Shader Engine) has spent processing vertices. Vertex shaders are executed before any clipping and culling in a scene is performed, so the number of vertices processed will match the number of vertices submitted by the driver.

You can find out more about how the GPU processes vertices in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the average number of cycles it takes to shade a vertex during the selected time period is high.

If this value is high, then you should do the following:

1. **Profile with the offline GLSL compiler:** Batch process your application's vertex shaders with the appropriate profiling compiler in the PowerVR SDK using the "-perfsim" flag. You can then use the profile output to isolate the most expensive shaders in your scene, which will help you identify where the best place is to focus your optimization.

A.18. USSE Load: Pixel

Tooltip

Percentage of time that the USSE (Universal Scalable Shader Engine) has spent processing pixels.

What does this counter show?

This counter represents the pixel workload of the USSE (Universal Scalable Shader Engine). It's worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average "USSE load: pixel" only while the 3D core is active.

You can find out more about how the USSE shades fragments in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that a large percentage of the USSE's workload during the selected time period has been spent shading fragments.

If this value is high, then you should do the following:

1. **Profile with the offline GLSL compiler:** Batch process your application's fragment shaders with the appropriate profiling compiler in the PowerVR SDK using the "-perfsim" flag. You can then use the profile output to isolate the most expensive shaders in your scene, which will help you identify where the best place is to focus your optimization.
2. **Reduce alpha blending and discard/alpha test:** The number of fragments given to the USSE by the TSP (Texture and Shading Processor) is affected by the number of blended and alpha tested primitives that are being rendered. Because of this, reducing the amount of blending and discard in an applications render can reduce the "USSE load: pixel".

A.19. USSE Load: Stall

Tooltip

Percentage of time that the USSE (Universal Scalable Shader Engine) is stalled.

What does this counter show?

In Series 5 hardware, a thread that has to resolve a data dependency will be suspended while the resolve occurs so that other USSE (Universal Scalable Shader Engine) threads in the queue can be immediately scheduled in for execution. This per-USSE thread scheduling allows the majority of stalls to be hidden but if all threads are in a stalled state, then latency will be introduced into the render. This counter represents the average percentage of time that the USSE is in a stalled state, i.e. all USSE threads are stalled.

It's worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average "USSE load: stall" only while the TA or 3D cores are active.

You can find out more about why USSE stalls may occur in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here:
http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the USSE has spent a large percentage of time during the selected period in a stalled state.

If this value is high, then you should do the following:

1. **Reduce the number of dependant texture reads:** When a USSE thread performs a dependant texture read it will be scheduled out so another thread in the queue can be scheduled in. The purpose of this is to prevent the USSE stalling. Reducing the number of dependant texture reads in shaders will reduce the probability that all USSE threads will be in a stalled state, which will allow the USSE scheduler to mask latency better.

A.20. USSE Load: Vertex

Tooltip

Percentage of time that the USSE (Universal Scalable Shader Engine) has spent processing vertices.

What does this counter show?

This counter represents the vertex workload of the USSE (Universal Scalable Shader Engine). It's worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average "USSE load: vertex" only while the TA core is active.

You can find out more about how the USSE shades vertices in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that a large percentage of the USSE's workload during the selected time period has been spent shading vertices.

If this value is high, then you should do the following:

1. **Profile with the offline GLSL compiler:** Batch process your application's vertex shaders with the appropriate profiling compiler in the PowerVR SDK using the "-perfsim" flag. You can then use the profile output to isolate the most expensive shaders in your scene, which will help you identify where the best place is to focus your optimization.
2. **Improve CPU object culling:** Improving CPU culling of objects so there are fewer draw calls will reduce the USSE's vertex processing load as fewer vertices will be submitted to the GPU. This culling should, ideally, be applied to geometry inside the view frustum as well as outside of it.
3. **Reduce polygon count/tessellation factor:** Reducing the number of polygon's that are submitted to the GPU will reduce the USSE's vertex processing load. Polygon/tessellation reduction should, ideally, be applied to geometry inside the view frustum as well as outside of it.

A.21. USSE Total Load

Tooltip

Percentage of time that the USSE (Universal Scalable Shader Engine) is processing instructions (this does not include stall time).

What does this counter show?

This counter represents the total workload of the USSE (Universal Scalable Shader Engine), i.e. when it is processing vertex or fragments. It does not included time that the USSE is stalled.

It's worth noting that the average calculated load in the counter's columns is based on any time that the GPU was active or idle during the selected period of time, i.e. this load does not represent the average "USSE total load" only while the TA or 3D cores are active.

You can find out more about the in the "Architecture Guide for Developers" document in the PowerVR SDK's Documentation folder or online here: http://www.imgtec.com/powervr/insider/powervr-sdk-docs.asp

What does a high value mean?

A high value indicates that the USSE has spent a large percentage of time during the selected period shading vertices and fragments.

If this value is high, then you should do the following:

1. **Check the value of “USSE load: pixel”:** If the majority if the USSE total load has come from shading fragments, you should focus on reducing the “USSE load: pixel”. Please see the documentation for this counter for more information on how to reduce this workload.
2. **Check the value of “USSE load: vertex”:** If the majority if the USSE total load has come from shading vertices, you should focus on reducing the “USSE load: vertex”. Please see the documentation for this counter for more information on how to reduce this workload.

A.22. Vertex Transforms per Frame

Tooltip

Number of vertices transformed by the USSE (Universal Scalable Shader Engine) per-frame.

What does this counter show?

This counter represents the average number of vertex transformations that the USSE (Universal Scalable Shader Engine) has performed per-frame. It should be noted that this average value in the counter’s columns is calculated as the number of vertex transformations per-frame that have been processed during the selected time period and it is not PID specific.

What does a high value mean?

As this counter does not represent a load of a GPU unit, it’s difficult to determine anything from this value alone.

If this value is high and you suspect this may be a bottleneck, you should do the following:

1. **Check the value of “USSE load: vertex”:** If the USSE’s vertex processing load is very high, you may need to reduce the number of vertices that you are submitting to the GPU. Please see the documentation for this counter for more information on how to reduce this workload.

A.23. Vertex Transforms per Second

Tooltip

Number of vertices transformed by the USSE (Universal Scalable Shader Engine) per-second.

What does this counter show?

This counter represents the average number of vertex transformations that the USSE (Universal Scalable Shader Engine) has performed per-second. It should be noted that this average value in the counter’s columns is calculated as the number of vertex transformations per-second that have been processed during the selected time period and it is not PID specific.

What does a high value mean?

As this counter does not represent a load of a GPU unit, it’s difficult to determine anything from this value alone.

If this value is high and you suspect this may be a bottleneck, you should do the following:

1. **Check the value of “USSE load: vertex”:** If the USSE’s vertex processing load is very high, you may need to reduce the number of vertices that you are submitting to the GPU. Please see the documentation for this counter for more information on how to reduce this workload.

A.24. Frames per second (FPS): PID

Tooltip

The average number of frames-per-second processed by the GPU for a specific CPU process.

What does this counter show?

This counter represents the average number of frames per second processed by the GPU for a specific CPU process. For each process rendering using the GPU since PVRTune started capturing data, a new “frames per second (FPS): PID <x>” will be created (where <x> is the Process Identifier (PID) of a given process).

What does a high value mean?

A high value indicates that the GPU has been able to process a high number of frames during the selected period of time for a given process.

Appendix B. Event List

B.1. Active Counters Changed

This event represents the point at which the active counter group has been changed using the **Error! Reference source not found.**. This event appears as a grey line.

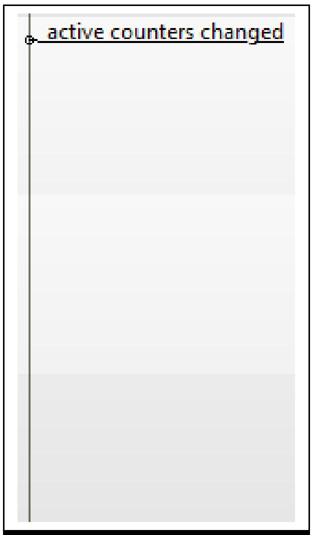


Figure 6-1 Active Counters Changed

B.2. Event Ordinal Reset

In general this event is hidden by an 'Active Counters Changed'; this event represents a change in the ordering of the counters or counter sources read by PVRPerfServer. This event appears as a green line.

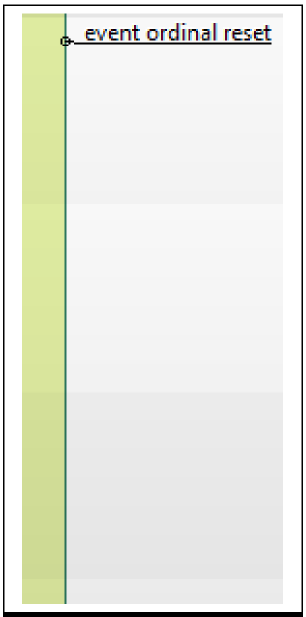


Figure 6-2 Event Ordinal Reset

B.3. Custom Mark

Custom marks are marks that have been sent to PVRTune either by a PVRScope enabled application or by pressing the 'M' key from within PVRPerfServer; they appear as a red line.

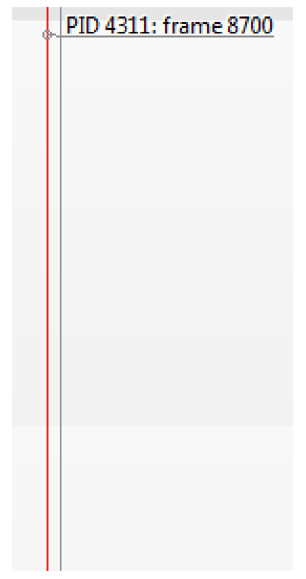


Figure 6-3 Custom Mark

B.4. Power-off Period

Power-off periods are represented by a long wide grey block in the Graph View; these events occur when the hardware has gone to sleep, powering down to save power due to a lack of work.

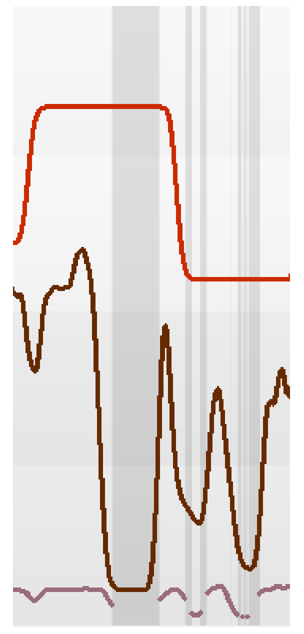


Figure 6-4 Power-off Period

B.5. Data Loss

During this period PVRTune has lost data that was sent by PVRPerfServer. There are several possibilities to improve this, some of which are listed here.

- Decrease device CPU load.
- Alter the PVRPerfServer data read rate, using the “-c” command-line option.
- There could be congestion on the network the data is being transmitted on. This can be alleviated by using a less congested network or by attempting ad-hoc networking.
- Save data to a file, rather than send it over the network; see the ‘-sendto=’ command-line option for PVRPerfServer.
- Reduce the data quantity; disable the ‘periodic’ or ‘graphics’ option via the PVRPerfServer command-line.

Note that some of the above PVRPerfServer command-line options can also be set remotely from PVRTune (see Section **Error! Reference source not found.**).