PVRTune

User Manual

Filename	:	PVRTune.User Manual.1.0.12.DeveloperExternal.doc
Version	:	1.0.12 External Issue (Package: POWERVR SDK REL_3.0@2149525)
Issue Date	:	31 Aug 2012
Author	:	Imagination Technologies Ltd

Contents

List of Figures

1. Introduction

1.1. Software Overview

PVRTune, combined with PVRPerfServer, is a performance analysis tool for the PowerVR SGX series of graphics accelerators. It uses driver level counters and hardware registers to provide real-time data on the performance of the SGX, while an application is running, at negligible cost.

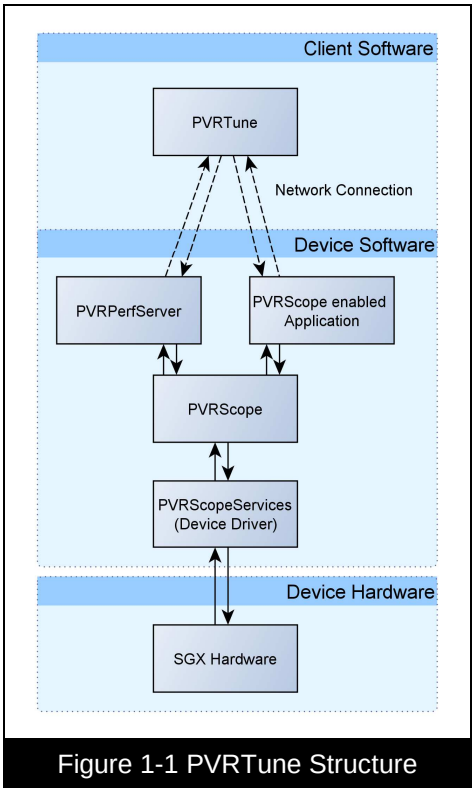

Figure 1-1 PVRTune Structure

1.1.1. PVRPerfServer

PVRPerfServer is the device-side element of the combined package. Its purpose is to read counters and registers from the SGX hardware and either save the data to a .pvrtune file, or transmit that data to PVRTune over a network.

1.1.2. 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 SGX 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

With the package successfully installed, copy `'PVRPerfServer.apk'` to the device where your graphics application is going to be run and select it with a file manager; Android will install the PVRPerfServer application. Once installation is complete, PVRPerfServer can be run by selecting it from the application menu.

2.3.3. Linux, and Windows

With the package successfully installed, copy the PVRPerfServer binary to the device where the application to be analysed is going to be run.

2.4. Driver Compatibility

2.4.1. Compatible DDK Ranges

Compatible driver 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 `'PVRPerfServer'`.

2.5.2. Linux, Neutrino, Windows

From a command-line interface run the PVRPerfServer executable.

2.5.3. General Notes

When PVRPerfServer has been started correctly, an interface like those in Figure 2-1 PVRPerfServer – Command-Line or Figure 2-2 PVRPerfServer – Graphical Interface will be displayed.

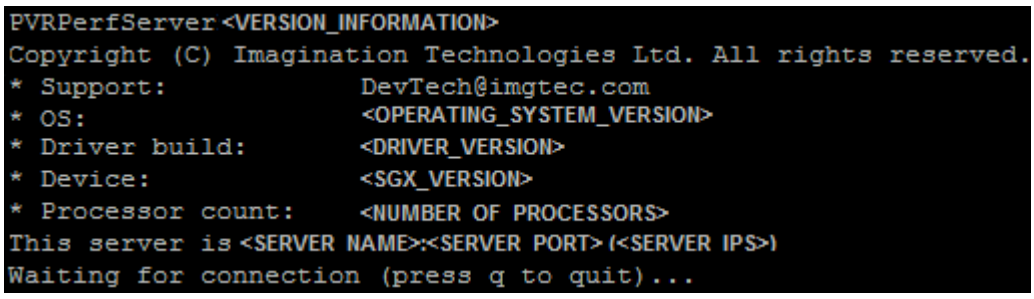

Figure 2-1 PVRPerfServer – Command-Line

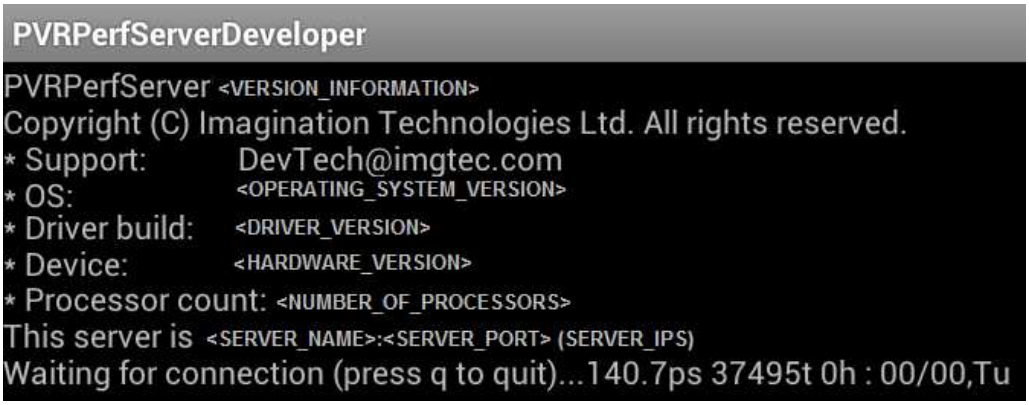

Figure 2-2 PVRPerfServer – Graphical Interface

Finally, it should be noted that 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.
-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.

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 it can display is broken down into groups of profiling counters which can be switched using the Remote Control Toolbar. The currently active group is highlighted, and its contents updated in real-time in the Counter List. Finally the Graph View allows for the graphing of these counters, as well as the displaying of frame-by-frame timing data with the 'TA/3D' view.

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

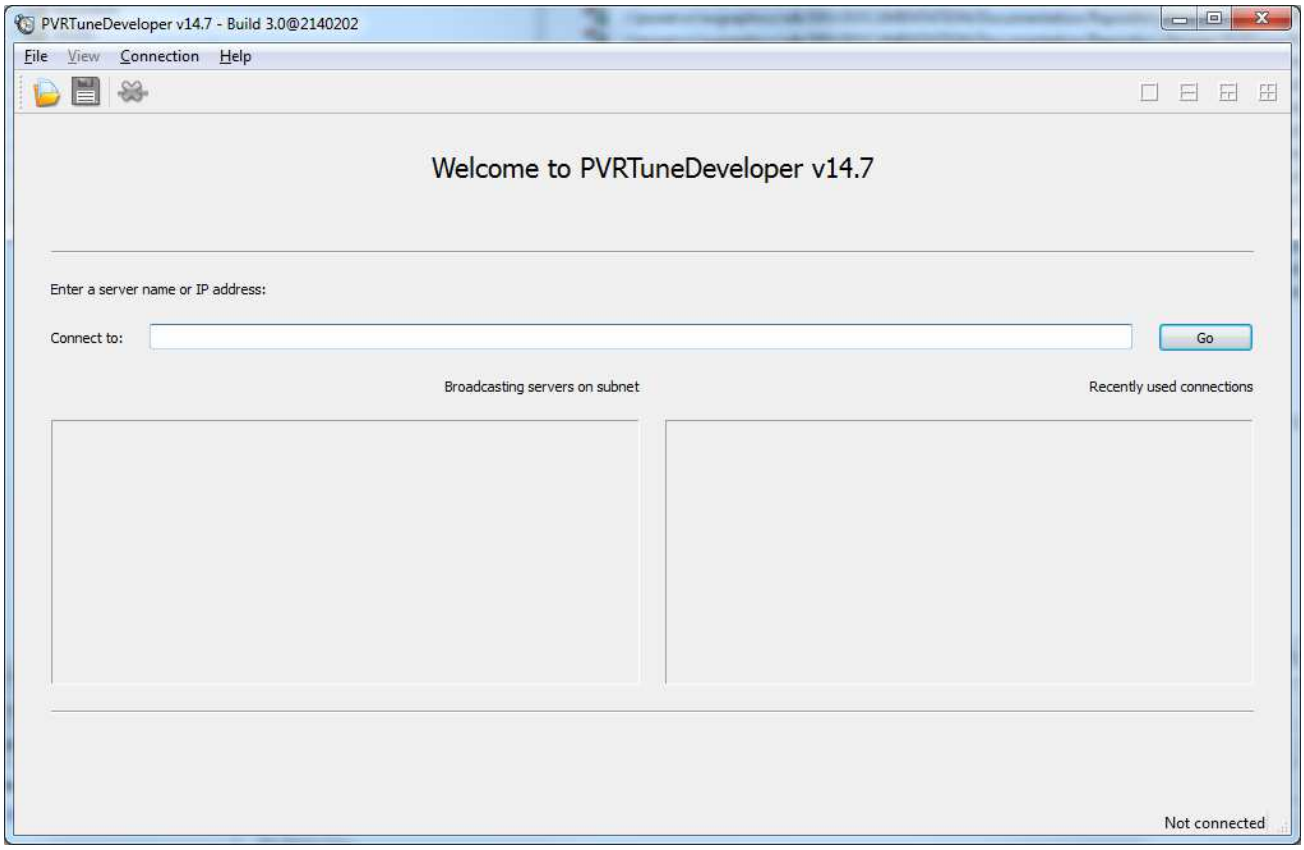

Figure 3-1 Connection Window

The Connection Window is the first window that is seen when PVRTune is launched. It consists of several areas and a single toolbar.

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 running on the targeted device, PVRTune will connect.

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 client machine's subnet. It is possible to connect to a device by clicking its IP address in this list.

Recently Used Connections

The 'Recently Used Connections' box contains a list of the IP addresses of the devices that have most recently been connected to. It is possible to connect to a device by clicking its IP address in this list.

3.4. Main Interface

3.4.1. Graph View

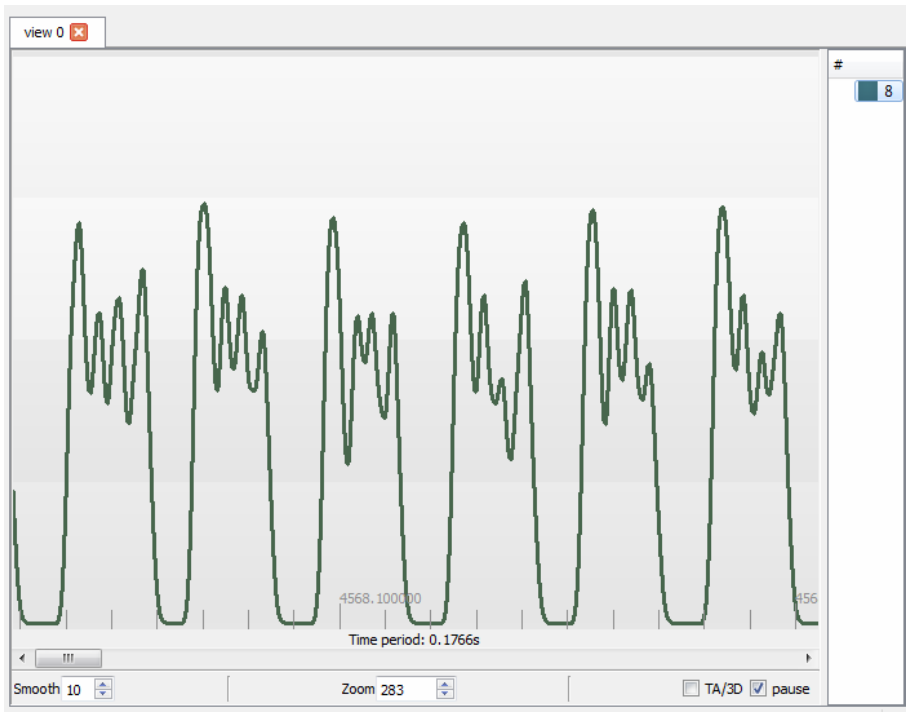

Figure 3-2 Graph View

The Graph View allows for the various counters sent by PVRPerfServer to be graphed over time. Multiple counters can be displayed on a single graph and multiple graphs can be displayed using the Graph View Control Toolbar. Finally it is also possible to have multiple Graph Views open simultaneously in multiple tabs; new tabs can be created by clicking `'File -> New Tab'`.

Adding/Removing Counters

To add counters to a graph, drag the counter from the Counter onto the Graph View; the y-axis scale for the counter can be changed in the Counter Properties box. Once a counter has been placed onto a graph, a legend will appear displaying the counter colour and number, as can be seen in Figure 3-2 Graph View. Counters can be removed from the Graph View by selecting them and pressing 'Delete' or from right clicking and choosing 'Delete' from a contextual menu.

Events and Marks

Several events (also known as 'Marks') are displayed on the Graph View 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. Event List.

Context Menus

Right clicking on the Graph View will bring up a context menu with options to display the Timing Data (TA/3D), to pause or un-pause the graph, adjust whether PVRTune should render 'marks' or not, as well as the option to save the current graph to a .png file.

Right clicking on the legend will bring up a contextual menu with the options to select all the currently graphed counters, delete the selection, or delete all of the currently graphed counters.

It is also possible to select a region of the graph by using 'Ctrl' and Left Click; this selected region includes a ruler at the bottom.

Time Period

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

3.4.2. Timing Data (TA/3D)

The Graph View Control Toolbar contains an option: 'TA/3D'; when this option is ticked the timing data is displayed.

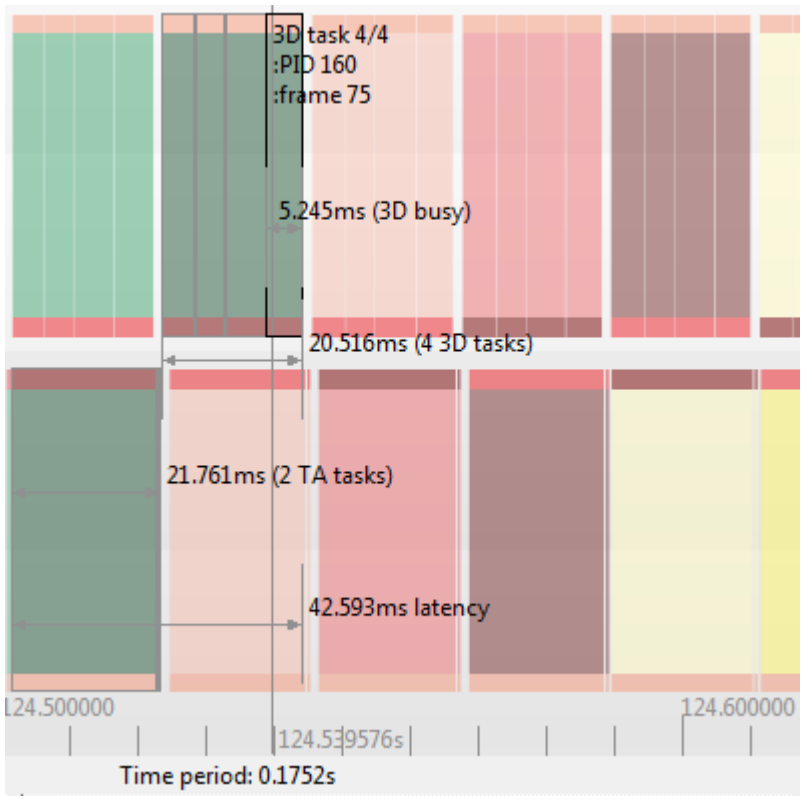

Figure 3-3 Timing Data

TA/3D Tasks

Timing data is split into two main parts. The bottom half represents 'TA time', this is a measure of how much time is spent tiling/culling the frame and running vertex shaders. The top half represents '3D time' and is a measure of how much time is spent fetching textures, processing fragments etc. These halves represent the two main stages in the TBDR process.

Each block represents a given task within a frame (e.g. tiling, vertex shading of a render target, fragment shading of a render target etc.) and is colour coded so frame, process ID and render target can be easily spotted.

In addition to 'TA Time' and '3D Time' the Timing Data can also include information on 'Transfer Tasks'.

Mouse Hover Data

A wealth of information can be gained from the timing data by hovering the mouse over a frame. This mouse hover information details the total number of TA and 3D tasks, the process ID, the frame number, the amount of time spent on each task, the amount of time spent processing a set of TA or 3D tasks, and the total amount of time spent processing a 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.

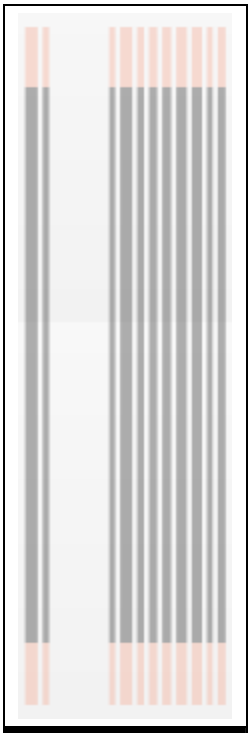

Figure 3-4 Transfer Tasks

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 Timing Data (these colours are recycled every sixteen frames). In addition to the general block colour, each tip of a block is coloured. The outside tips (the top of the 3D tasks and the bottom of the TA tasks) visualise process ID, a different colour representing a different process ID. The inside tips (the top of the TA tasks and the bottom of the 3D tasks) visualise the render target, a different colour representing a different render target.

Colour Coding Example

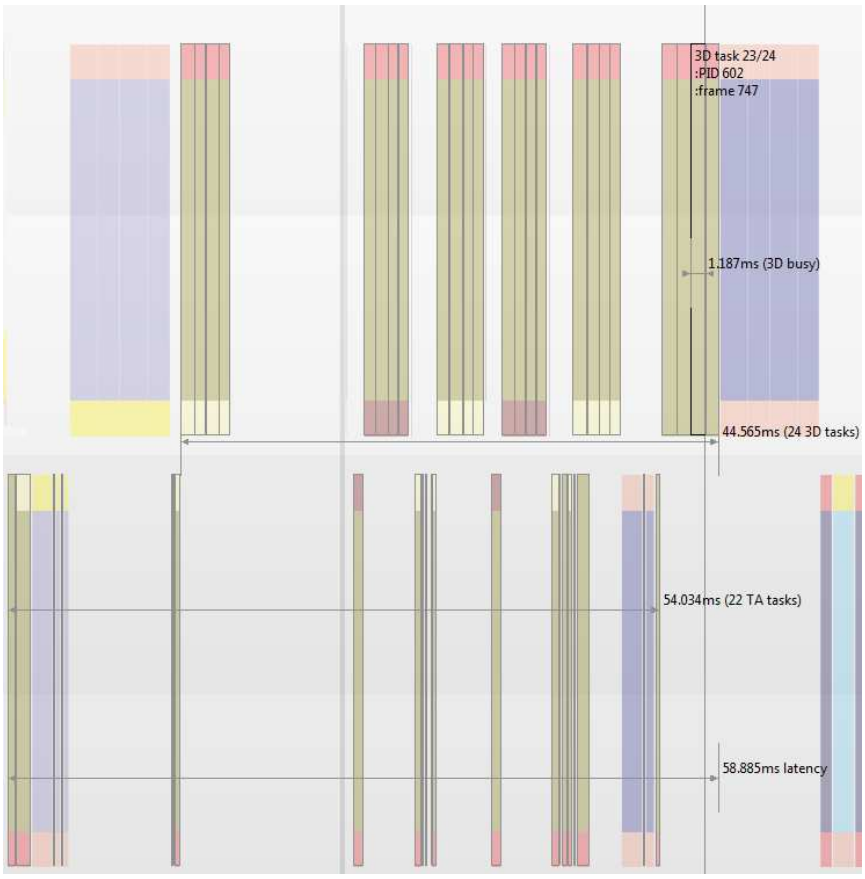

Figure 3-5 Colour Coding Example

In Figure 3-5 Colour Coding Example the green tasks all represent a single frame. They also represent a single process; this is shown by the matching pink tips on the outside of the tasks. The two blocks either side of the green frame in '3D Tasks' are a different process; this can be identified by the paler pink colouration on their outer tip. Both are generated by the same process since the tip colours match, but each represent a different frame of that process, as the 'core' colour does not. Within the green frame three render targets are used, one marked by the white inner tip, one by the purple inner tip, and one by the green inner tip. The white render target is used three times, the render targets share a colouration identifying them as the same target; the pink render target is used two times, again the same colouration identifying them as the same target; the green render target is used once at the end of the frame and thus is likely to represent the back buffer. In double or triple buffered situations this final render target would be different for each frame, alternating between each of the back buffer targets as can be seen in Figure 3-6 Double Buffering.

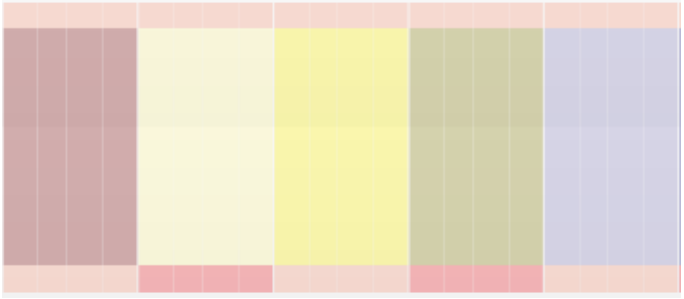

Figure 3-6 Double Buffering

3.4.3. Counter Table

The counter list represents the counters available within the current group as selected from the Remote Control Toolbar. These counters are updated in real-time and can be dragged onto the Graph View to give a graphical representation of their change over time. A number of counters are hidden by default, these 'advanced counters' can be accessed by right clicking and un-ticking `'Hide advanced counters'`. It is also possible to hide inactive counters by right clicking and selecting `'Hide inactive counters'`. A full list of the available counters can be found in Appendix A. Counter List. For instructions on how to add or remove counters from the Graph View see Section 3.4.1 Graph View.

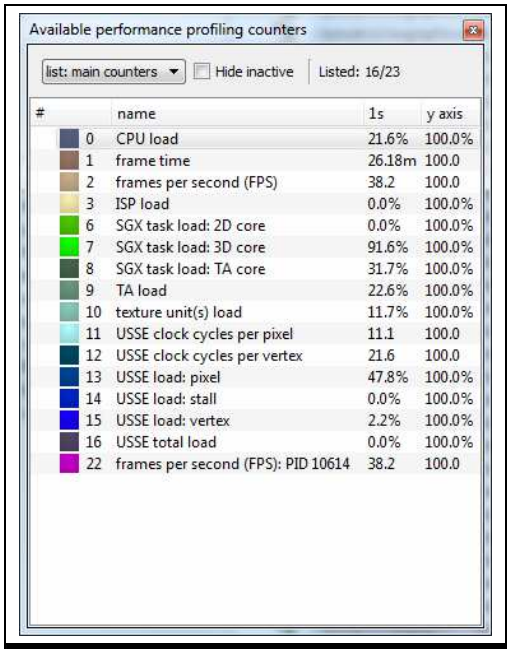

Figure 3-7 Counter Table

3.4.4. Counter Properties

This section of the interface displays the properties of the currently selected counter, and allows them to be edited. It displays the following:

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

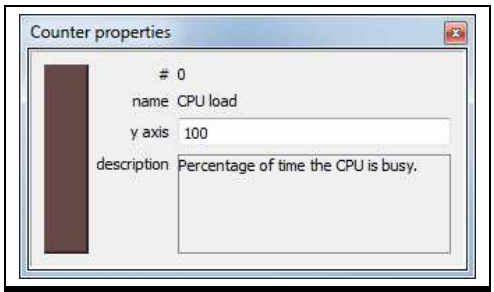

Figure 3-8 Counter Properties

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 large block of colour; this will open a colour selection dialog from which a new colour can be selected.

3.5. Toolbars

3.5.1. Main Toolbar

Open

This button brings up an ‘Open File’ dialog for the opening of .pvrtune files. This option is only available on the Connection Window.

Figure 3-9 Open

Save

This button brings up a ‘Save File’ dialog for the saving of .pvrtune files. This option is only available while connected to PVRPerfServer or while viewing a .pvrtune file.

Figure 3-10 Save

Disconnect

While connected to PVRPerfServer, a single click on this button will close the connection to the server, but will not return the user to the Connection Window; this allows the user to view the currently gathered data without gathering more. This data can still be saved to a file after disconnecting. In order to return to the Connection Window a second click is required.

Figure 3-11 Disconnect

While reading a pre-recorded .pvrtune file only a single click is required to return to the Connection Window.

3.5.2. Remote Control Toolbar

Figure 3-12 Remote Control Toolbar

When connected to PVRPerfServer the Remote Control Toolbar allows PVRTune to remotely control the counter groups that are currently active on the server; this in turn then activates or deactivates certain counters on the SGX hardware. A full list of the available counters and the groups they exist in can be found in Appendix A. Counter List.

Advanced Options

The advanced options can be displayed clicking the button marked `'More'`.

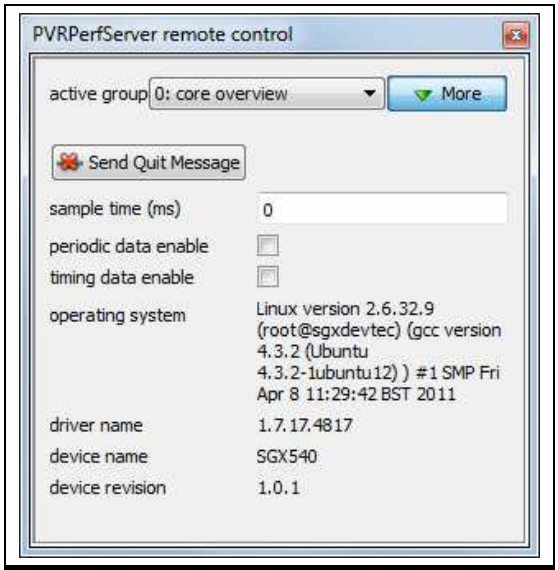

Figure 3-13 Remote Control Toolbar – Advanced Options

Send Quit Message

This option forces the PVRPerfServer to which PVRTune is currently connected, to close.

Sample Time

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

Periodic Data Enable

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

Timing Data Enable

When this option is ticked PVRTune will receive TA/3D timing data. This option is on by default.

Server Information

The Advanced Options also include information about the device on which PVRPerfServer is running; the operating system of the device, the version of the SGX driver being used, the device whose counters are being read, and the device's revision number.

3.5.3. Graph View Control Toolbar

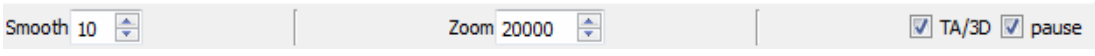

Figure 3-14 Graph View Control Toolbar

Smooth

The `'Smooth'` option sets the value used to smooth out the graphs in the Graph View to make them more readable. The default value is `'10'`, the max value is `'80'`, with the graph smoothness increasing as the value is increased.

Zoom

'Zoom' represents the amount a graph has been zoomed by. As this value changes so does the time period displayed on the graph.

TA/3D

With this option ticked, Timing Data is displayed in the graph. Further information on this can be found in Section 3.4.1 Graph View.

Pause

This option stops the graph at the current time. When un-paused the graph will jump to the current time and continue graphing.

3.5.4. Graph View Layout Toolbar

Figure 3-15 Graph View Layout Toolbar

The Graph View Layout Toolbar controls the layout of the Graph View. It consists of four options that allow for 1-4 separate graphs in a variety of layouts:

- One full size graph.
- Two graphs separated horizontally.
- Three graphs arranged as two graphs split vertically below another.
- Four graphs, each graph covering a quarter of the graph view area.

3.5.5. Geek Stats

Figure 3-16 Geek Stats

The ‘Geek Stats’ 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 in the current status.
- 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: A measure of the amount ‘Count’ moves by 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 that PVRTune is towards disconnecting due to memory usage (after PVRTune has recorded 1GB of data it will disconnect from PVRPerfServer).
- Group: The currently active counter group.

3.6. Dialogs

3.6.1. 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 will appear in the Remote Editor; from here these values can be edited and the changes sent back to the application. This powerful functionality allows for real-time, remote editing of anything from shaders, to any number of numerical values. More information on PVRScope can be found in the PVRScope User Manual.

3.6.2. Select Columns

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

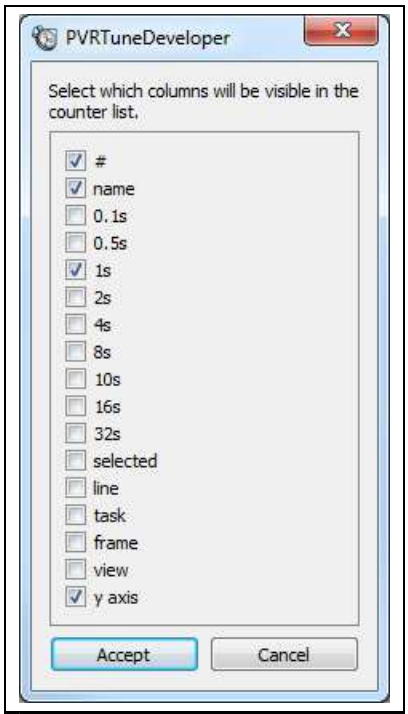

Figure 3-17 Select Columns

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).

Line

This column shows the values of counters at the time associated with the position of the mouse in the Graph View.

Task

This column shows the average value of counters over the time frame of a single TA or 3D task.

Frame

This column shows the average value of given counters over the time frame of a single frame.

View

This column shows the average value of counters over the time frame of the graph the mouse is currently over.

Selected

It is now possible to select a region of the graph using 'Ctrl' and Left Click. This column shows the average over the region selected.

3.7. Menus

3.7.1. File

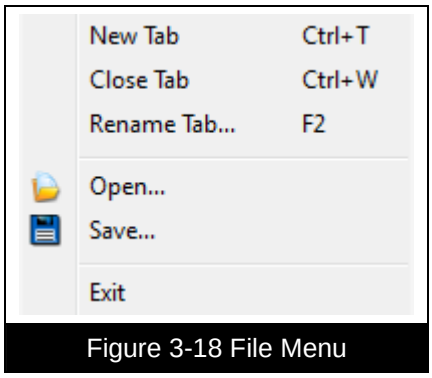

Figure 3-18 File Menu

New Tab
Opens up a new tab in the Graph View.

Close Tab
Closes the currently open tab in the Graph View.

Rename Tab
Renames the currently open tab in the Graph View.

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. View

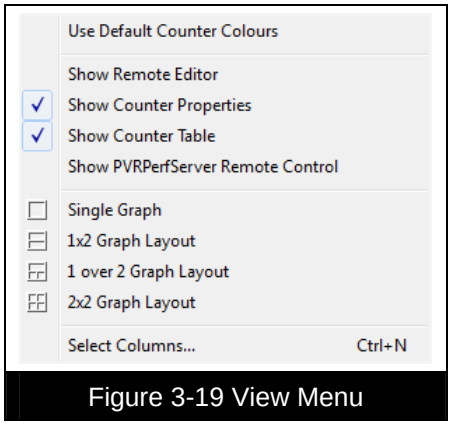

Figure 3-19 View Menu

Use Default Counter Colours
This option informs PVRTune that you wish to use its default colour scheme; it is turned on by default.

Show PVRPerfServer Remote Control
Toggles displaying of the Remote Control Toolbar.

Show Counter Table
Toggles displaying of the Counter Table.

Show Counter Properties
Toggles displaying of the Counter Properties

Show Remote Editor
Opens the Remote Editor dialog.

Graphing Styles
These four options change the layout of the Graph View in the same way as the Graph View Layout Toolbar.

Select Columns
Opens the Select Columns dialog.

3.7.3. Connection Menu

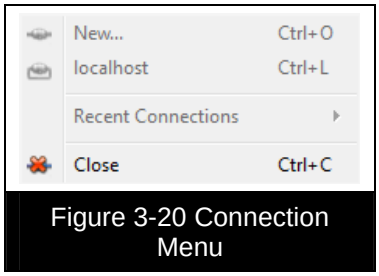

Figure 3-20 Connection Menu

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 functions similarly to the Recently Used Connections box on the Connection Window; it lists the IP addresses of the devices that have most recently been connected to. It is possible to connect to a device by clicking its IP address in this list. This option is only available on the Connection Window.

Close

The 'Close' option closes the currently open connection to PVRPerfServer. It does not return PVRTune to the Connection Window when first pressed only closing the connection, this allows the data retained so that the user can save the tune to a file or continue analysis of it; on a second press this will return the user to the Connection Window.

3.7.4. Help

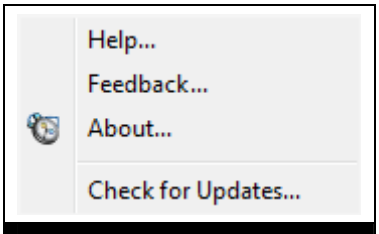

Figure 3-21 Help Menu

PVRTune Help

'PVRTune Help' opens this document.

Feedback

When this item is clicked a box will appear 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.

3.8. Analysing an Application

Analysing an application with PVRTune can help to achieve many possible goals; three are listed here:

- Improved frame rate to increase user enjoyment and improve responsiveness.
- Reduced render time without increasing 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 satisfying the first of these goals.

3.8.1. Connected Analysis

1. Ensure the device is using the PowerVR SGX device drivers.
2. Boot the device.
3. Install PVRPerfServer.
4. If necessary initialise the PowerVR SGX device drivers.
5. Run PVRPerfServer – If successful PVRPerfServer will output the server's name, IP address and port.
6. Run PVRTune on the client machine – It is important that both PVRTune and PVRPerfServer have matching versions.
7. If PVRPerfServer is on the same subnet as PVRTune it will appear in the Broadcasting Servers panel of the Connection Window; if it is not, enter the IP address of the server and click the 'Go' button.
8. PVRTune will now connect.
9. Once successfully connected identify the bottleneck in your application by analysing PVRTunes output (see Section 3.8.3 The Five Common Bottlenecks for more information).
10. Attempt to handle this bottleneck – see the document "PowerVR Performance Recommendations" for more information on optimisation techniques.
11. Repeat these steps with the newly optimised application until performance is at the required level, or no further bottlenecks can be identified.

3.8.2. Offline Analysis

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.

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, when using the `'-sendto='` parameter will only save counter data for a single counter group. By default this is group zero. This can be changed with the `'-group='` command-line parameter.

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 will display the tuning data.
5. Identify the bottleneck (see Section 3.8.3 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.

3.8.3. The Five Common Bottlenecks

When analysing the performance of an application, bottlenecks will almost always fall into one of five categories: CPU limited, vertex limited, fragment limited, bandwidth limited, or V-Sync limited.

CPU Limited

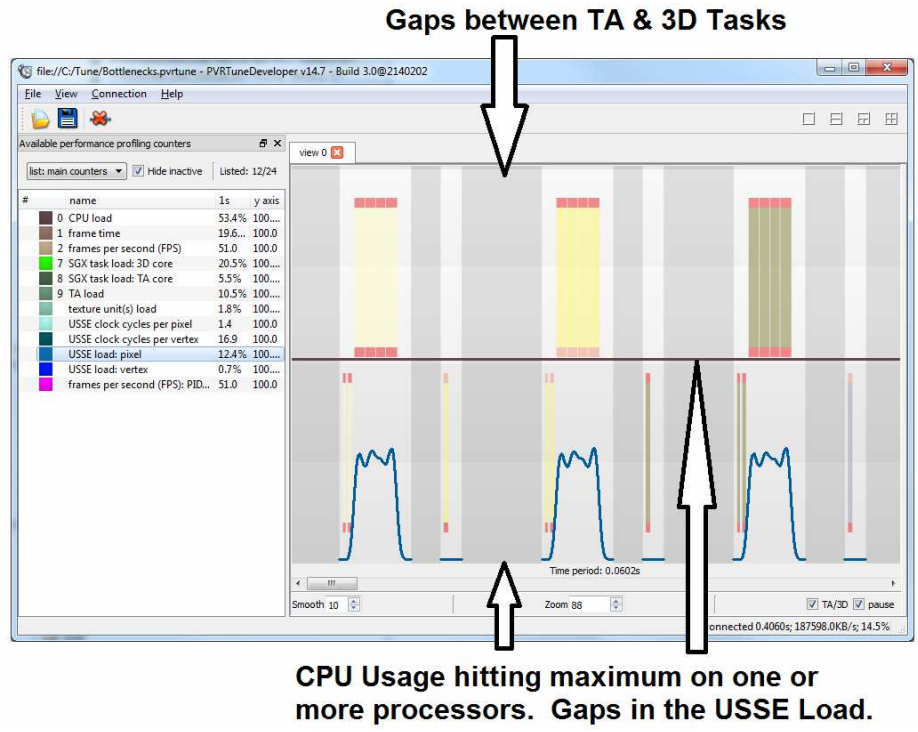

Figure 3-22 CPU Limited

A CPU limited application is often identifiable as an application suffering from poor performance or frame rate even though the GPU is not being taxed. In PVRTune this can be very easily identified; CPU limited applications will have a CPU load that is either, at or near one hundred percent; or is wildly variable (as can be seen in Figure 3-22 CPU Limited). Other identifying factors include gaps in the USSE load, caused by the SGX hardware going to sleep while waiting for further instructions; and regular visible gaps between frames when displaying Timing Data.

Vertex Limited

Figure 3-23 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 coupled with little or no gap between long vertex tasks. 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.

V-Sync Limited

V-Sync (short for Vertical Synchronisation) is a display option that forces an application to synchronise its 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 easy to spot; intermittent gaps will appear in the Graph View in which the SGX will enter sleep mode, and the frame rate will appear to cap 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 would be beneficial and if work has actually been successful.

Fragment Limited

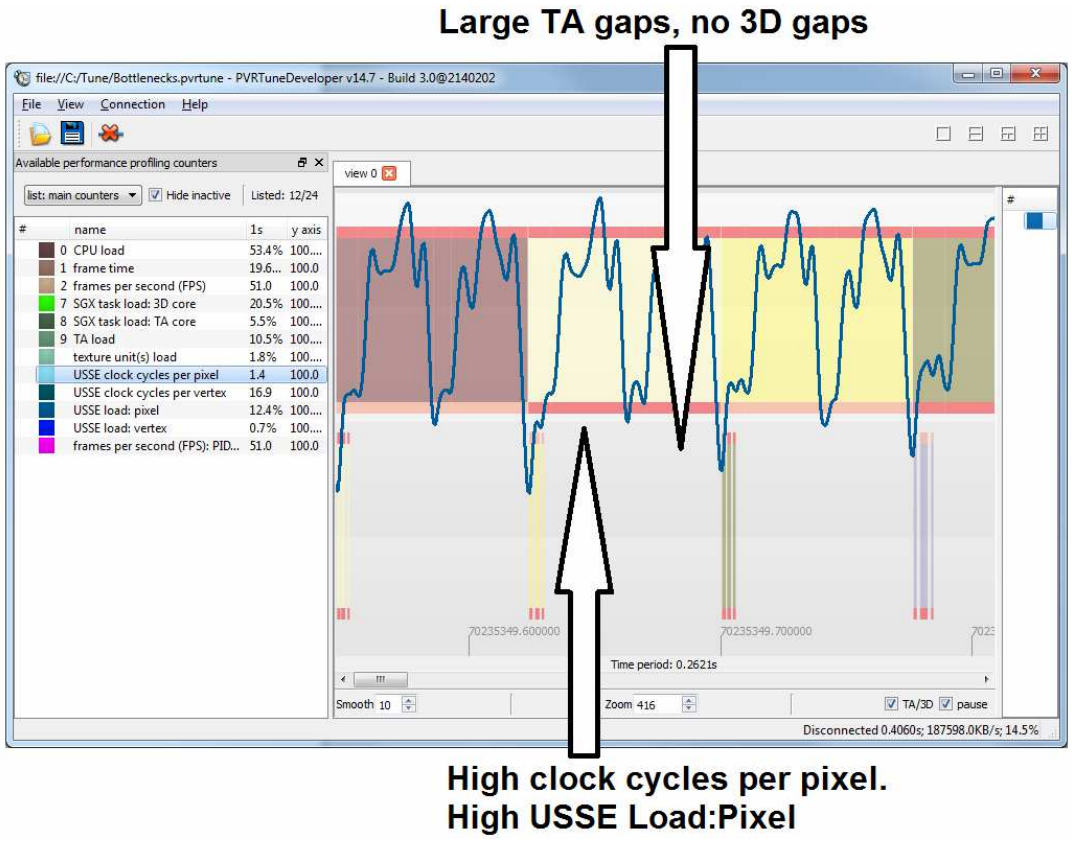

Figure 3-24 Fragment Limited

An application being fragment limited is very common; most scenes contain fewer vertices than the frame buffer does pixels. The primary means of identifying a fragment limited application are a lack of gaps between long 3D tasks, coupled with a large gap between TA tasks, and a high value of `'USSE load: Pixel'`.

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 SGX. 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 at a premium. It should also be noted that bandwidth in system-on-chip (SoC) devices is shared amongst all components of the chip; other, non-SGX, areas of the chip utilising large amounts of bandwidth may still cause an applications graphics to be bandwidth limited.

4. Related Materials

Software

- PVRScope
- PVRTrace

Documentation

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

5. Contact Details

For further support contact:

devtech@imgtec.com

PowerVR Developer Technology
Imagination Technologies Ltd.
Home Park Estate
Kings Langley
Herts, WD4 8LZ
United Kingdom

Tel: +44 (0) 1923 260511
Fax: +44 (0) 1923 277463

Alternatively, you can use the PowerVR Insider forums:

www.imgtec.com/forum

For more information about PowerVR or Imagination Technologies Ltd. visit our web pages at:

www.imgtec.com

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 will be 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 will affect 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 "SGX 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 will reduce the ISP load as fewer fragments will be submitted to the GPU
2. **Utilise back-face culling:** Enabling back-face culling will allow the TA (Tile Accelerator) to cull more polygons, which will reduce the amount of data that will be written 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 “SGX 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 “SGX 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 "SGX 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 “SGX 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 “SGX 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:

4. **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
5. **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
6. **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. SGX 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 will show the load of this unit. If a PowerVR 2D core is not present in the target device the counter will be greyed out.

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. SGX 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 "SGX 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. SGX 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 “SGX 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 “SGX 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 Series 5 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 "SGX 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 SGX 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 "SGX 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 (e.g. glslcompiler_sgx540) 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 "SGX 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 (e.g. glslcompiler_sgx540) 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 "SGX 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 (e.g. glslcompiler_sgx540) 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 “SGX 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
2. **<Any more?>**

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 “SGX 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 (e.g. glslcompiler_sgx540) 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 "SGX 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 Remote Control Toolbar and appears as a grey line.

Figure 5-1 Active Counters Changed

B.2. Event Ordinal Reset

In general this event will be hidden by an ‘Active Counters Changed’; it represents a change in the ordering of the counters or counter sources read by PVRPerfServer; it appears as a blue line.

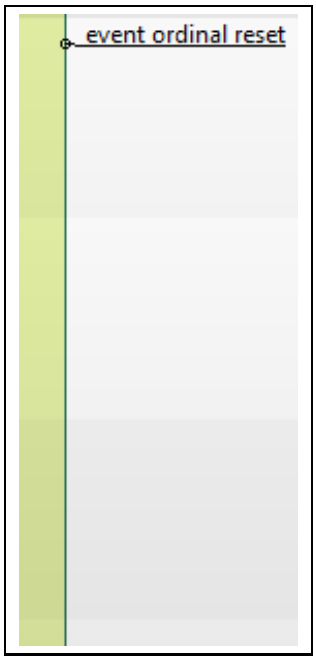

Figure 5-2 Event Ordinal Reset

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.

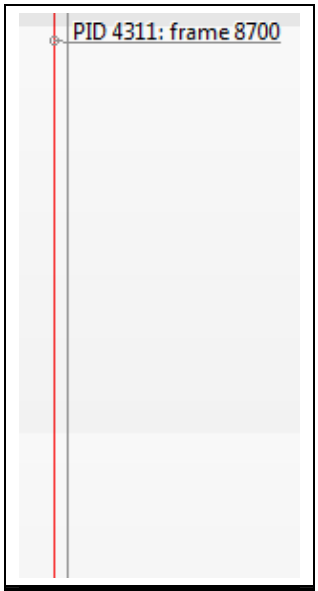

Figure 5-3 Custom Mark

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 SGX hardware has gone to sleep, powering down to save power due to a lack of work.

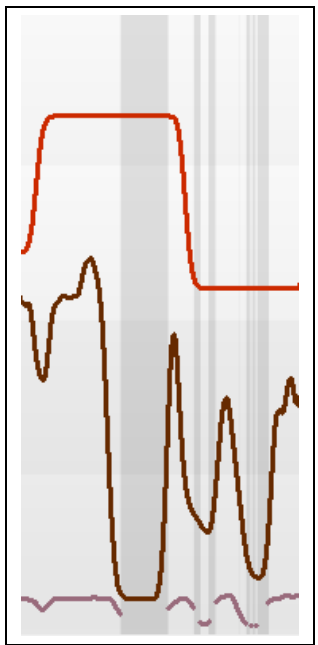

Figure 5-4 Power-off Period

B.5. Data Lost

During this period PVRTune has lost data that was sent by PVRPerfServer. In general this is because the network the data is being transmitted on is congested. This can be alleviated by using a less congested network or by attempting ad-hoc networking.

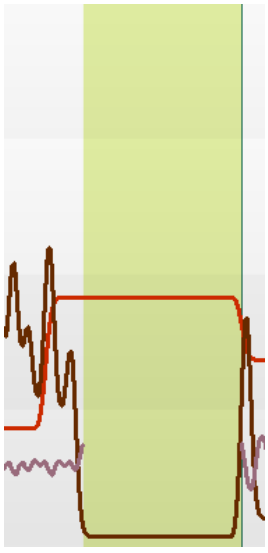

Figure 5-5 Data Lost