OpenGL Driver Monitor User Guide

Graphics & Imaging > OpenGL

2008-02-08

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures

Introduction

OpenGL Driver Monitor is a developer tool that has two purposes. It is:

- An application that lets developers see how OpenGL works on a specific system and look at the capabilities of a driver
- An advanced diagnostic tool that OpenGL driver developers and experts can use to track down thorny performance issues

Most OpenGL developers should not use the driver monitor application to analyze performance issues; they should instead use Shark and OpenGL Profiler.

You'll want to read this document if you:

- Develop applications that use OpenGL on Mac OS X and you are curious as to how the GPU and CPU interact
- Want to look at the capabilities of a particular OpenGL driver
- Are an OpenGL driver developer who needs to investigate a driver bug
- Are an advanced OpenGL developer or consultant trying to track down a performance issue that you've been unable to analyze using Shark and OpenGL Profiler

Organization of This Document

This document is organized into the following chapters:

- "Using OpenGL Driver Monitor" (page 9) describes how to set preferences, collect real-time parameter values locally or remotely, and view renderer information.
- "Identifying and Solving Performance Issues" (page 17) provides strategies for using OpenGL Driver Monitor to analyze performance issues.
- "OpenGL Driver Monitor Parameters" (page 19) describes, by symbolic and descriptive names, the parameters that you can monitor.

See Also

These documents contain information that can help you analyze and optimize your OpenGL code:

- *OpenGL Programming Guide for Mac OS X* discusses best practices for getting optimal performance.

- *OpenGL Profiler User Guide* explains how to collect and analyze data that can help you tune your OpenGL application.
- *Shark User Guide* describes how to optimize application performance using this tool.

Using OpenGL Driver Monitor

OpenGL Driver Monitor is a developer tool that lets you investigate how the graphics processing unit (GPU) works on a system-wide basis. Using it, you can:

- Inspect the renderers available on a Mac OS X system
- See what OpenGL extensions each renderer supports
- Monitor rendering parameters and resources in real-time
- Track the interaction between the GPU and the CPU

This chapter provides an overview of OpenGL Driver Monitor and shows how you can use it to look at and track OpenGL parameters.

Setting Preferences

For most cases, the default values that OpenGL Driver Monitor uses are optimal. Before you start using the driver monitor for the first time, you might want to familiarize yourself with its preferences to see whether the default values are acceptable for your needs.

To set preferences:

1. Choose OpenGL Driver Monitor > Preferences.
2. Adjust the sampling interval.

   OpenGL Driver Monitor measures data throughput in bytes per sampling interval. For example, the default sampling interval is 1 second, so the data throughput is in bytes per second.

   If you change the sampling interval, the OpenGL Driver Monitor measures values for that interval; the values are not normalized to a 1 second interval. For example, if you set the sampling interval to 2 seconds, and the throughput value is 500 bytes, the reported sampling rate is 500 bytes per interval which, in this case, is 250 bytes per second.
3. Adjust the maximum data samples.

   The value is the maximum number bytes per sampling interval. The default is 2048.
4. Adjust graph colors.

   If you don't like that default colors for the graph background or graph labels, click the color wells and choose other colors.
5. Select whether or not to use descriptive parameter names.

OpenGL Driver Monitor provides a list of parameters from which you choose the ones you want to track. You can either view the list as the symbolic names used by driver monitor (such as `command2DBytes`, `gartMapOutBytes`, `finishAll2DWaitTime`) or as descriptive names (such as 2D Command Data, AGP Data Unmapped, CPU Wait for Operations to Finish).

6. If you want to monitor this computer remotely, select "Enable remote monitor."

   See "Monitoring an OpenGL Driver Remotely" (page 15) for additional details.

7. Click Apply to commit to the changes you made. Then click OK to close the Preferences window.

Collecting and Viewing Data

OpenGL Driver Monitor lets you collect system-wide data for a specific OpenGL driver. You can collect data for the system that you are running the driver monitor on or a system that you already set up for remote monitoring. (See "Monitoring an OpenGL Driver Remotely" (page 15).) The next few sections show how to collect the data.

After OpenGL Driver Monitor launches, choose a driver to investigate from the Monitors > Driver Monitors menu. Most systems have only one driver, but if your system has more than one graphics card installed, you'll see more than one entry. After choosing a driver, the driver window opens with a view of an empty graph. You'll notice that the Driver Monitor Parameters list below the graph is empty.

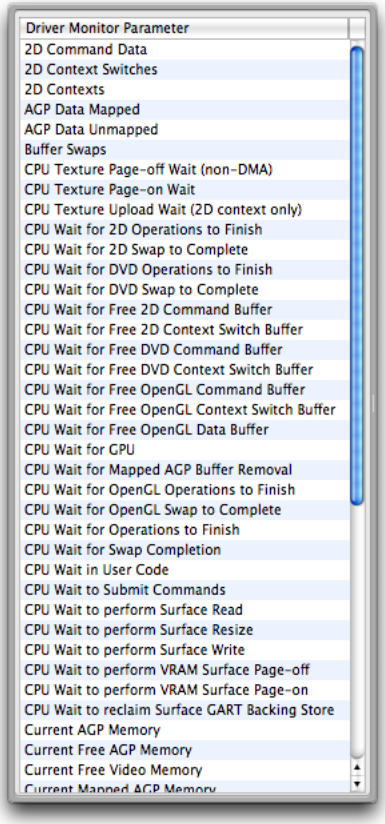
Click Parameters to open a drawer that lists all the driver parameters that you can monitor. Hover the pointer over a parameter name to see its description. See also "OpenGL Driver Monitor Parameters" (page 19) for definitions of the parameter names and for cross-references between the symbolic name and the descriptive name for a parameter.

Either double-click each of the parameters (shown in Figure 1-1 (page 11)) you want to monitor, or drag them to the Driver Monitor Parameters list below the graph. As you might expect, not all parameters are equally useful for every scenario. You'll need to choose accordingly.

Keep in mind that the parameters shown in Figure 1-1 (page 11) are for a particular driver. Not all drivers support the same parameters, so it's possible that the list you see doesn't match either what's shown in the figure or what's listed in the glossary.

You'll notice that when you see similarly named parameters, one of them is typically a "super parameter." The value of a super parameter includes the values of all its "child parameters." For example, the super parameter `commandBytes` (Total Command Data) includes all quantities represented by the similarly named parameters `command2DBytes` (2D Command Data), `commandGLBytes` (OpenGL Command Data), and `commandDVDBytes` (DVD Command Data).

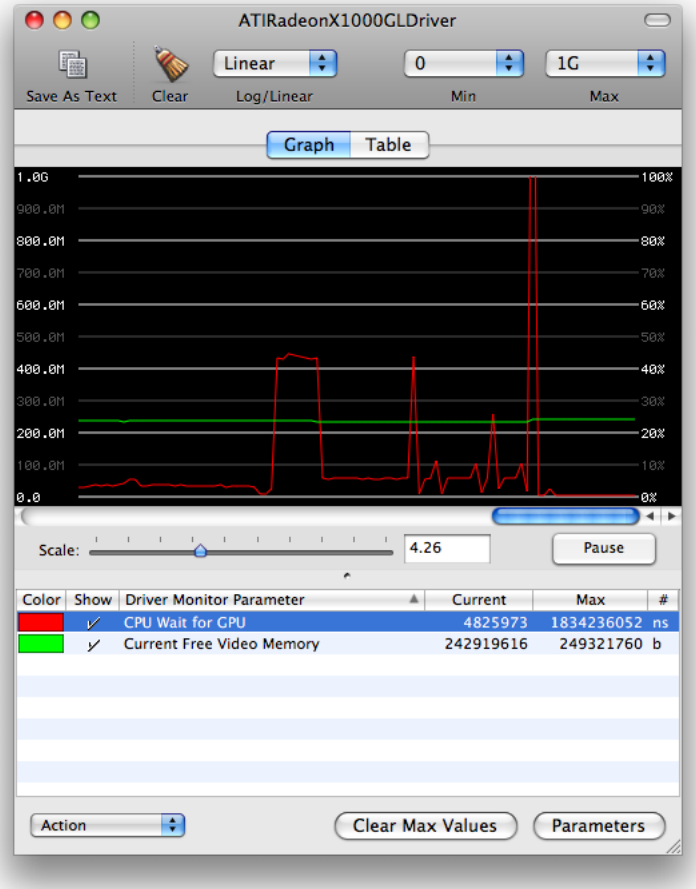
Figure 1-1 The parameters drawer



Viewing Graphed Data

After you choose a parameter, OpenGL Driver Monitor adds it to the Driver Monitor Parameter list and starts to display data on the graph and in the columns next to the parameter name. Figure 1-2 (page 12) shows the driver monitor graph window after displaying two parameters—CPU Wait for CPU and Current Free Video Memory. If you prefer to use colors other than the default ones for graphing the values, click the color well for a parameter and choose another from the color panel that appears.

Figure 1-2 The driver monitor graph window



The y-axis values on the left side of the graph are values that represent different units depending on the parameter. The y-axis values on the right side represent percentages. Keep the following in mind when reading the graph:

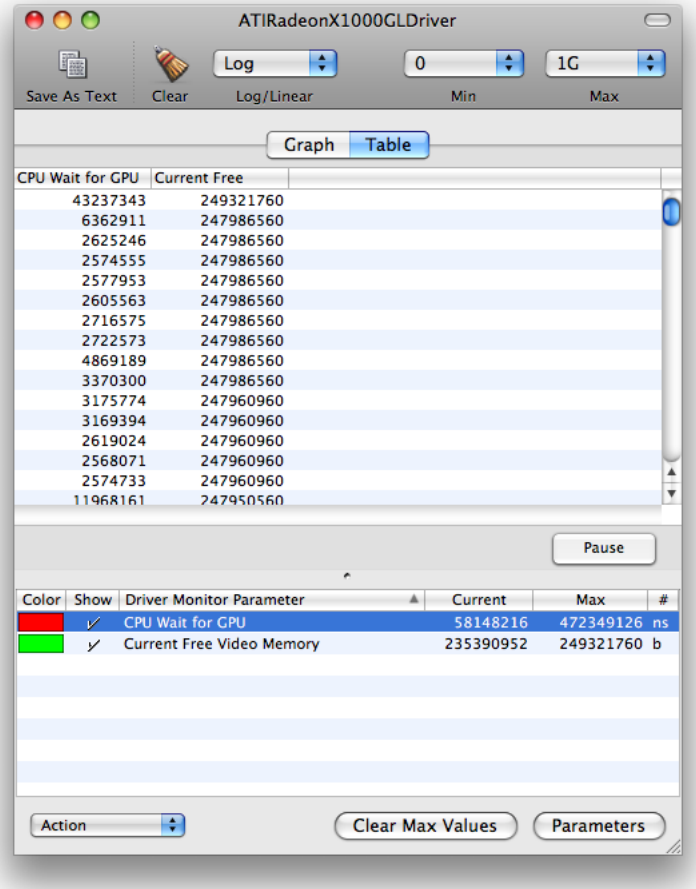
- Time-based parameters values represent nanoseconds. For example, 1 giga-nanosecond (or 1G, as shown on the graph) represents about 1 second spent on an operation. If the sampling interval is 1 second, then the percentage is 100%.
- Parameters that represent counts are absolute numbers, with quantities measured once per sampling interval. Counts are not affected by the length of the sampling interval, but may vary during the interval.

You can adjust the scale of the x-axis and the base (log or linear) of the y-axis to help you see changes in the values.

Viewing Tabular Data

You can view the data in tabular format if you prefer. This lets you compare running values among the parameters you are monitoring, as shown in Figure 1-3 (page 13).

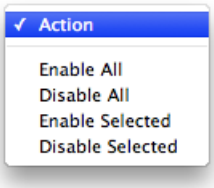
Figure 1-3 The driver monitor table window



Enabling and Disabling Parameters

You enable and disable the driver monitor parameters that you are monitoring by using the Action pop-up menu shown in Figure 1-4 (page 13). You can also click the Show column to show or hide data for a particular parameter.

Figure 1-4 The Action pop-up menu

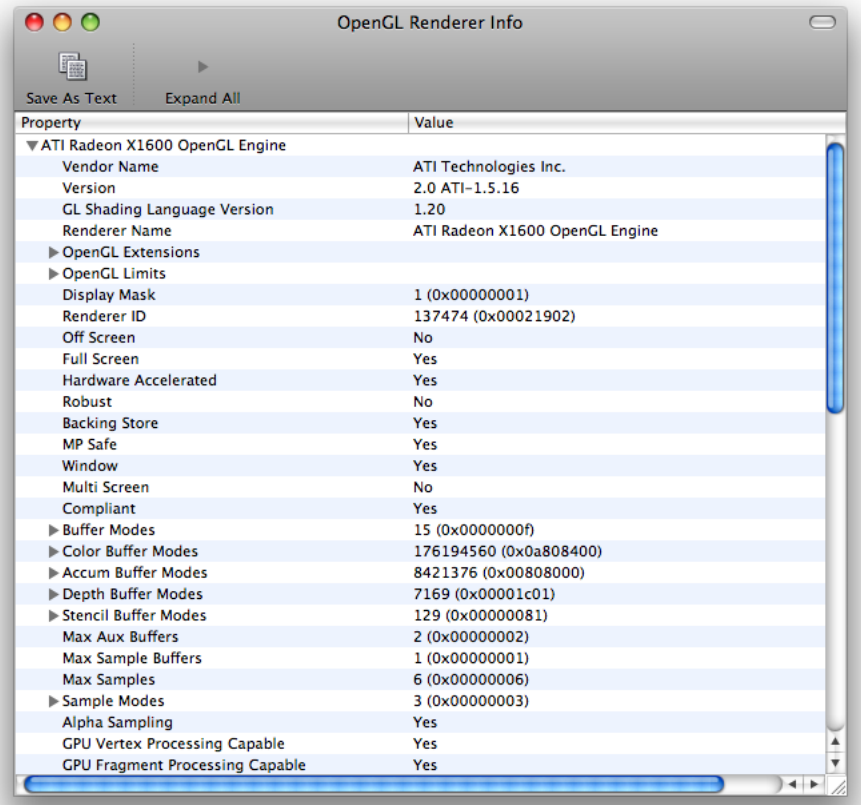


Looking at Renderer Information

An OpenGL Driver has at least two renderers associated with it—one hardware renderer and one software renderer. You can take a look at what each renderer supports by choosing Monitors > Renderer Information. Then, when the OpenGL Renderer Info window opens, you can click each of the disclosure triangles to view the settings for a particular renderer.

As Figure 1-5 (page 14) shows, you can view the vender name and version, the OpenGL extensions that the driver supports, the buffer modes, various processing capabilities, and so on.

Figure 1-5 The renderer information window



If you want to view renderer information for renderers other than those on your system, you can set up remote monitoring on the computers whose renderers you want to inspect. See "Monitoring an OpenGL Driver Remotely" (page 15).

Monitoring an OpenGL Driver Remotely

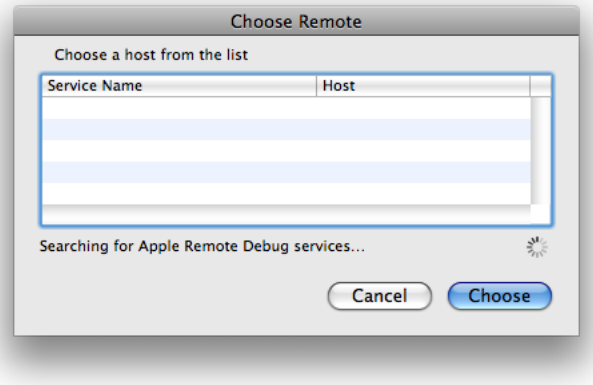
Before you can monitor an OpenGL driver on a remote computer, you need to enable remote monitoring on that computer by following these steps:

1. Launch OpenGL Driver Monitor on the computer that you want to monitor remotely.
2. Choose OpenGL Driver Monitor > Preferences.
3. Click "Enable remote monitor."
4. Make a note of the remote monitor name. You'll need this later.
5. Click Apply.

To monitor a computer after you enable remote monitoring:

1. Launch OpenGL Driver Monitor on the computer that you plan to monitor from.
2. Choose Monitors > Connect To.
3. When the remote host window opens (see Figure 1-6 (page 15)), select the name of the computer you want to monitor and click Choose.

Figure 1-6 The remote host window



4. Collect, view, and interpret data the same way you would for a local OpenGL driver.

Identifying and Solving Performance Issues

OpenGL Driver Monitor is not the primary tool for analyzing performance issues in an OpenGL application. It's the backup tool that experts use when Shark and OpenGL Profiler don't reveal the cause of a performance problem. This chapter assumes that you have already used Apple's other tools to analyze your OpenGL application.

The strategies described here can help you identify the most common problems that occur in OpenGL applications. Keep in mind that analyzing difficult performance problems is more of an art than a science. Although you'll want to start with these basic strategies, you'll need to devise additional ones tailored to the type of problem you see, and to whether you are trying to solve a driver issue or an application one.

Checking for Best Practices

Before you begin to use OpenGL Driver Monitor as an analysis tool, it's a good idea to check your code to see if you are following the most recent best practices for using OpenGL. See:

- The "Improving Performance" chapter in *OpenGL Programming Guide for Mac OS X* for a discussion of best practices
- *OpenGL Profiler User Guide*. You'll find advice on functions that you need to make sure you use correctly, if you use them at all.

Checking Data Transfer Rates

To check data transfer rates, monitor the following:

- VRAM usage. See whether VRAM usage is at capacity by looking at Current Video Memory in Use (`vramUsedBytes`) or Current Free Video Memory (`vramFreeBytes`). If it is at capacity, investigate whether the system is low on VRAM or whether VRAM usage is unusually high for an application running on the system.
- Swap rate. A variety of parameters, such as Buffer Swaps (`bufferSwapCount`), let you investigate the cause of unusually high swap rates. Check to see whether the swapped data is dynamic or static. If the data os static, make sure you are using caches, vertex buffer objects, or some other technique that's optimized for static data. Use swaps only for dynamic data, and only when the data changes.
- The time spent by the CPU waiting for the GPU. Look at CPU Wait for GPU (`hardwareWaitTime`). If the CPU spends a lot of time simply waiting, check to see whether you are calling `glFlush` or `glFinish` inappropriately. There are only a few cases where you actually need to use these calls, and these cases are rare. For more information, see the "Improving Performance" chapter in *OpenGL Programming Guide for Mac OS X*.

Checking for Suboptimal Surface and Texture Paging

You need to make sure that your application is not paging texture and surface data unnecessarily. When it does page, you should use the accelerated graphics port (AGP pathway, which is also known as DMA transfer). Non-AGP transfers slow performance. You can check for less optimal paging by looking at these parameters:

- Surface Page Off Data (Non-AGP) (`surfacePageOffBytes`)
- Surface Page On Data (Non-AGP) (`surfacePageInBytes`)
- Texture Page Off Data (Non-AGP) (`texturePageOutBytes`)
- Texture Page Off Data (Non-AGP) (`texturePageInBytes`)

Non-AGP transfer is acceptable only if you must reorder data or align it. If possible, use this type of data transfer at initialization time and not during a rendering loop.

If your application has a lot of paging activity, whether it's AGP or non-AGP, consider using framebuffer objects.

OpenGL Driver Monitor Parameters

OpenGL Driver Monitor parameters are represented as descriptive names and as symbolic names. The parameters you can set for a specific driver are often a subset of what's listed here.

Descriptive Names

Each descriptive name describes what the parameter represents and lists its symbolic name.

2D Command Data
: The number of bytes sent using 2D graphics contexts. `command2DBytes`

2D Context Switches
: The total number of context switches to a 2D context on the GPU. `context2DSwitchCount`

2D Contexts
: The total number of 2D contexts in use on the GPU. `context2DCount`

AGP Data Mapped
: The number of bytes that are mapped into the AGP Graphics Address Remapping Table (GART) or equivalent hardware. `gartMapInBytes`

AGP Data Unmapped
: The number of bytes that are unmapped from the AGP Graphics Address Remapping Table (GART) (or equivalent hardware). `gartMapOutBytes`

Buffer Swaps
: The total number of buffer swaps (or blits) that the GPU perform. `bufferSwapCount`

CPU Texture Page-off Wait (non-DMA)
: The amount of time, in nanoseconds, that the CPU waits for the GPU to finish activity. This is the that the CPU could use to modify a texture prior to paging the texture. This metric applies only if the texture must be paged using the CPU and not using direct memory access (DMA). `texturePageOffWaitTime`

CPU Texture Page-on Wait
: The amount of time, in nanoseconds, that the CPU waits for a texture upload command to be completed by the GPU. This is the that the CPU could use for updating and reloading a texture. Typically, there is very little, if any time, spent waiting here. `texturePageInWaitTime`

CPU Texture Upload Wait (2D context only)
: The amount of time, in nanoseconds, that the CPU waits for a texture upload to complete before the buffer can be modified. This particular metric tracks usage only by 2D contexts, and is somewhat obsolete. `textureWaitTime`

CPU Wait for 2D Operations to Finish
: The amount of time, in nanoseconds, that the CPU waits for all 2D commands issued on a single context to complete. `finish2DWaitTime`

CPU Wait for 2D Swap to Complete
: The amount of time, in nanoseconds, that the CPU waits for a previously issued 2D buffer swap to complete. `swapComplete2DWaitTime`

CPU Wait for DVD Operations to Finish
: The amount of time, in nanoseconds, that the CPU waits for all DVD commands issued on a single context to complete. `finishDVDWaitTime`

CPU Wait for DVD Swap to Complete
: The amount of time, in nanoseconds, that the CPU waits for a previously issued DVD buffer swap to complete. `swapCompleteDVDWaitTime`

CPU Wait for Free 2D Command Buffer
: The amount of time, in nanoseconds, that the CPU waits for a 2D command buffer to become available. `freeCommandBuffer2DWaitTime`

CPU Wait for Free OpenGL Command Buffer
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL command buffer to become available. `freeCommandBufferGLWaitTime`

CPU Wait for Free OpenGL Data Buffer
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL data buffer to become available. `freeDataBufferGLWaitTime`

CPU Wait for Free 2D Context Switch Buffer
: The amount of time, in nanoseconds, that the CPU waits for a 2D context-switching buffer to become available. `freeContextBuffer2DWaitTime`

CPU Wait for Free OpenGL Context Switch Buffer
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL context-switching buffer to become available. `freeContextBufferGLWaitTime`

CPU Wait for Free DVD Context Switch Buffer
: The amount of time, in nanoseconds, that the CPU waits for a DVD context-switching buffer to become available. `freeContextBufferDVDWaitTime`

CPU Wait for GPU
: The amount of time, in nanoseconds, that the CPU stalled while waiting on the GPU for any reason. `hardwareWaitTime`

CPU Wait for Mapped AGP Buffer Removal
: The amount of time, in nanoseconds, the CPU waits for the GPU to finish an operation on a buffer that needs to be removed from the Graphics Address Remapping Table (GART). `removeFromGARTWaitTime`

CPU Wait for OpenGL Swap to Complete
: The amount of time, in nanoseconds, that the CPU waits for a previously issued OpenGL buffer swap to complete. `swapCompleteGLWaitTime`

CPU Wait for Operations to Finish
: The amount of time, in nanoseconds, that the CPU waits for all GPU operations to complete and then to be idle. Generally, only the window server waits for this state. `finishAll2DWaitTime`

CPU Wait for OpenGL Operations to Finish
: The amount of time, in nanoseconds, that the CPU waits for all OpenGL commands issued on a single context to complete. This is essentially the time spent in `glFinish`. `finishGLWaitTime`

CPU Wait in User Code
: The amount of time, in nanoseconds, that the CPU waits while the client (user-level) OpenGL driver waits for a hardware time stamp to arrive (usually for making texture modifications or waiting for a fence to complete). `clientGLWaitTime`

CPU Wait to perform Surface Read
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may read from a surface. `surfaceReadLockIdleWaitTime`

CPU Wait to perform Surface Resize
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may change the dimensions of a surface. `surfaceSetShapeIdleWaitTime`

CPU Wait to perform Surface Write
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may write to a surface. `surfaceWriteLockIdleWaitTime`

CPU Wait to perform VRAM Surface Page-off
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU can page a surface out of VRAM. `surfaceCopyOutWaitTime`

CPU Wait to perform VRAM Surface Page-on
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU can page a surface in to VRAM. `surfaceCopyInWaitTime`

CPU Wait to Submit Commands
: The amount of time, in nanoseconds, that the CPU waits before being able to submit a new batch of commands to the GPU. `hardwareSubmitWaitTime`

Current AGP Memory
: The total size, in bytes, of the AGP Graphics Address Remapping Table (GART). `gartSizeBytes`

Current Free AGP Memory
: The total number of free bytes in the AGP Graphics Address Remapping Table (GART). `gartFreeBytes`

Current Mapped AGP Memory
: The total number of bytes mapped into AGP Graphics Address Remapping Table (GART). `gartUsedBytes`

Current Free Video Memory
: The total number of bytes of free VRAM. This parameter is vendor specific and not available for all drivers. `vramFreeBytes`

Current Largest Free Video Memory Block
: The largest free contiguous chunk of VRAM, in bytes. This parameter is vendor specific and not available for all drivers. `vramLargestFree`

Current Video Memory in Use
: The total number of bytes of VRAM in use. This parameter is vendor specific and not available for all drivers. `vramUsedBytes`

DVD Command Data
: The number of bytes sent using DVD contexts. `commandDVDBytes`

DVD Context Switches
: The total number of context switches to a DVD context on the GPU. `contextDVDSwitchCount`

DVD Contexts
: The total number of DVD contexts in use on the GPU. `contextDVDCount`

Extra OpenGL Data
: The number of bytes used for extra OpenGL command traffic (usually vertex data). Not used by all drivers in all modes. `dataGLBytes`

Last GPU Submission Time
: The last submitted time stamp to the GPU, as an absolute time value. `submitStamp`

Last Read GPU time
: The last time stamp read back from the GPU, as an absolute time value. `lastReadStamp`

OpenGL Command Data
: The number of bytes sent using OpenGL contexts. `commandGLBytes`

OpenGL Contexts
: The total number of OpenGL contexts in use on the GPU. `contextGLCountcontextGLCount`

OpenGL Data Buffers
: The total number of extra OpenGL data buffers allocated. `dataBufferCount`

OpenGL Context Switches
: The total number of context switches to an OpenGL context on the GPU. `contextGLSwitchCount`

Surface Page Off Data (Non-AGP)
: The number of bytes transferred due to surface page-off operations. `surfacePageOffBytes`

Surface Page On Data (Non-AGP)
: The number of bytes transferred due to surface page-on operations. `surfacePageInBytes`

Surfaces
: The total number of surfaces allocated by the GPU. `surfaceCount`

Swap Data
: The number of bytes sent by swap commands. `swapBytes`

Target Minimum Mapped AGP Memory
: The minimum amount of data, in bytes, that a driver tries to keep mapped into AGP Graphics Address Remapping Table (GART). `gartCacheBytes`

Texture Page Off Data (Non-AGP)
: The number of bytes transferred for texture page-off operations. Under most conditions, textures are not paged off but are simply thrown away since a backup exists in system memory. Texture page-off traffic usually happens when VRAM pressure forces a page-off of a texture that only has valid data in VRAM, such as a texture created using the function `glCopyTexImage`, or modified using the functiona `glCopyTexSubImage` or *glTexSubImage*. `texturePageOutBytes`

Texture Page On Data (Non-AGP)
: The number of bytes transferred for texture page-ins. Textures mapped using AGP will not show up here. `texturePageInBytes`

Textures
: The total number of kernel textures allocated by the GPU. `textureCount`

Total Command Data
: The number of bytes sent using all graphics contexts (2D, OpenGL, DVD). `commandBytes`

Symbolic Names

Each symbolic name describes what the parameter represents and lists its descriptive name.

bufferSwapCount
: The total number of buffer swaps (or blits) that the GPU perform. (Buffer Swaps)

clientGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits while the client (user-level) OpenGL driver waits for a hardware time stamp to arrive (usually for making texture modifications or waiting for a fence to complete). (CPU Wait in User Code)

command2DBytes
: The number of bytes sent using 2D graphics contexts. (2D Command Data)

commandBytes
: The number of bytes sent using all graphics contexts (2D, OpenGL, DVD). (Total Command Data)

commandDVDBytes
: The number of bytes sent using DVD contexts. (DVD Command Data)

commandGLBytes
: The number of bytes sent using OpenGL contexts. (OpenGL Command Data)

context2DCount
: The total number of 2D contexts in use on the GPU. (2D Contexts)

context2DSwitchCount
: The total number of context switches to a 2D context on the GPU. (2D Context Switches)

contextDVDCount
: The total number of DVD contexts in use on the GPU. (DVD Contexts)

contextDVDSwitchCount
: The total number of context switches to a DVD context on the GPU. (DVD Context Switches)

contextGLCount
: The total number of OpenGL contexts in use on the GPU. (OpenGL Contexts)

contextGLSwitchCount
: The total number of context switches to an OpenGL context on the GPU. (OpenGL Context Switches)

dataBufferCount
: The total number of extra OpenGL data buffers allocated. (OpenGL Data Buffers)

dataGLBytes
: The number of bytes used for extra OpenGL command traffic (usually vertex data). Not used by all drivers in all modes. (Extra OpenGL Data)

finish2DWaitTime
: The amount of time, in nanoseconds, that the CPU waits for all 2D commands issued on a single context to complete. (CPU Wait for 2D Operations to Finish)

finishAll2DWaitTime
: The amount of time, in nanoseconds, that the CPU waits for all GPU operations to complete and then to be idle. Generally, only the window server waits for this state. (CPU Wait for Operations to Finish)

finishDVDWaitTime
: The amount of time, in nanoseconds, that the CPU waits for all DVD commands issued on a single context to complete. (CPU Wait for DVD Operations to Finish)

finishGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits for all OpenGL commands issued on a single context to complete. This is essentially the time spent in `glFinish`. (CPU Wait for OpenGL Operations to Finish)

freeCommandBuffer2DWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a 2D command buffer to become available. (CPU Wait for Free 2D Command Buffer)

freeCommandBufferGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL command buffer to become available. (CPU Wait for Free OpenGL Command Buffer)

freeContextBuffer2DWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a 2D context-switching buffer to become available. (CPU Wait for Free 2D Context Switch Buffer)

freeContextBufferDVDWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a DVD context-switching buffer to become available. (CPU Wait for Free DVD Context Switch Buffer)

freeContextBufferGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL context-switching buffer to become available. (CPU Wait for Free OpenGL Context Switch Buffer)

freeDataBufferGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits for an OpenGL data buffer to become available. (CPU Wait for Free OpenGL Data Buffer)

gartCacheBytes
: The minimum amount of data, in bytes, that a driver tries to keep mapped into AGP Graphics Address Remapping Table (GART). (Target Minimum Mapped AGP Memory)

gartFreeBytes
: The total number of free bytes in the AGP Graphics Address Remapping Table (GART). (Current Free AGP Memory)

gartMapInBytes
: The number of bytes that are mapped into the AGP Graphics Address Remapping Table (GART) or equivalent hardware. (AGP Data Mapped)

gartMapOutBytes
: The number of bytes that are unmapped from the AGP Graphics Address Remapping Table (GART) (or equivalent hardware). (AGP Data Unmapped)

gartSizeBytes
: The total size, in bytes, of the AGP Graphics Address Remapping Table (GART). (Current AGP Memory)

gartUsedBytes
: The total number of bytes mapped into AGP Graphics Address Remapping Table (GART). (Current Mapped AGP Memory)

hardwareSubmitWaitTime
: The amount of time, in nanoseconds, that the CPU waits before being able to submit a new batch of commands to the GPU. (CPU Wait to Submit Commands)

hardwareWaitTime
: The amount of time, in nanoseconds, that the CPU stalled while waiting on the GPU for any reason. (CPU Wait for GPU)

lastReadStamp
: The last time stamp read back from the GPU, as an absolute time value. (Last Read GPU time)

removeFromGARTWaitTime
: The amount of time, in nanoseconds, the CPU waits for the GPU to finish an operation on a buffer that needs to be removed from the Graphics Address Remapping Table (GART). (CPU Wait for Mapped AGP Buffer Removal)

submitStamp
: The last submitted time stamp to the GPU, as an absolute time value. (Last GPU Submission Time)

surfaceCount
: The total number of surfaces allocated by the GPU.(Surfaces)

surfaceCopyInWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU can page a surface in to VRAM. (CPU Wait to perform VRAM Surface Page-on)

surfaceCopyOutWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU can page a surface out of VRAM. (CPU Wait to perform VRAM Surface Page-off)

surfacePageInBytes
: The number of bytes transferred due to surface page-on operations. (Surface Page On Data (Non-AGP))

surfacePageOffBytes
: The number of bytes transferred due to surface page-off operations. (Surface Page Off Data (Non-AGP))

surfaceReadLockIdleWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may read from a surface. (CPU Wait to perform Surface Read)

surfaceSetShapeIdleWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may change the dimensions of a surface. (CPU Wait to perform Surface Resize)

surfaceWriteLockIdleWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to become idle so that the CPU may write to a surface. (CPU Wait to perform Surface Write)

swapBytes
: The number of bytes sent by swap commands. (Swap Data)

swapComplete2DWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a previously issued 2D buffer swap to complete. (CPU Wait for 2D Swap to Complete)

swapCompleteDVDWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a previously issued DVD buffer swap to complete. (CPU Wait for DVD Swap to Complete)

swapCompleteGLWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a previously issued OpenGL buffer swap to complete. (CPU Wait for OpenGL Swap to Complete)

textureCount
: The total number of kernel textures allocated by the GPU. (Textures)

texturePageInBytes
: The number of bytes transferred for texture page-ins. Textures mapped using AGP will not show up here. (Texture Page On Data (Non-AGP))

texturePageInWaitTime
: The amount of time, in nanoseconds, that the CPU waits for a texture upload command to be completed by the GPU. This is the that the CPU could use for updating and reloading a texture. Typically, there is very little, if any time, spent waiting here. (CPU Texture Page-on Wait)

texturePageOffWaitTime
: The amount of time, in nanoseconds, that the CPU waits for the GPU to finish activity. This is the that the CPU could use to modify a texture prior to paging the texture. This metric applies only if the texture must be paged using the CPU and not using direct memory access (DMA). (CPU Texture Page-off Wait (non-DMA))

texturePageOutBytes

The number of bytes transferred for texture page-off operations. Under most conditions, textures are not paged off but are simply thrown away since a backup exists in system memory. Texture page-off traffic usually happens when VRAM pressure forces a page-off of a texture that only has valid data in VRAM, such as a texture created using the function `glCopyTexImage`, or modified using the functions `glCopyTexSubImage` or `glTexSubImage`. (Texture Page Off Data (Non-AGP))

textureWaitTime

The amount of time, in nanoseconds, that the CPU waits for a texture upload to complete before the buffer can be modified. This particular metric tracks usage only by 2D contexts, and is somewhat obsolete. (CPU Texture Upload Wait (2D context only))

vramFreeBytes

The total number of bytes of free VRAM. This parameter is vendor specific and not available for all drivers. (Current Free Video Memory)

vramLargestFree

The largest free contiguous chunk of VRAM, in bytes. This parameter is vendor specific and not available for all drivers. (Current Largest Free Video Memory Block)

vramUsedBytes

The total number of bytes of VRAM in use. This parameter is vendor specific and not available for all drivers. (Current Video Memory in Use)

Document Revision History

This table describes the changes to *OpenGL Driver Monitor User Guide*.

Date	Notes
2008-02-08	Fixed a link.
2007-12-11	New document that explains how to view the properties supported by the OpenGL drivers available on the system.

Document Revision History