Getting Started with PerfHUD 6.0

DU-01231-001_v09
Nov 2007

DEVELOPMENT

Table of Contents

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com

Chapter 1. Introduction

PerfHUD 6 is a visual debugging and profiling tool that can help you find and fix problems in your graphics rendering pipeline.

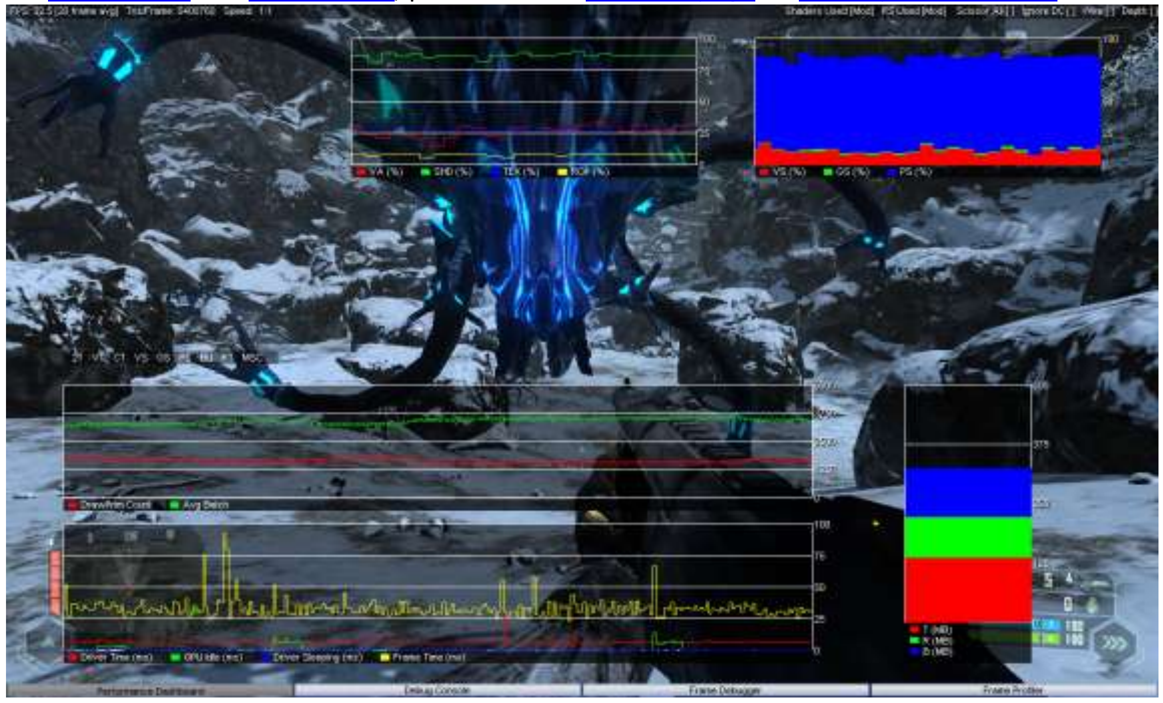
It does this by letting you explore and edit data in the graphics pipeline, all while your application is running. PerfHUD also offers automated performance analysis to quickly identify the most expensive draw calls.

A short overview video of PerfHUD is on the PerfHUD home page: http://developer.nvidia.com/PerfHUD.

PerfHUD is used by leading game developers around the worlds. Some examples are listed below:

Unreal Tournament 3 (Epic Games)	*Company of Heroes* (Relic Entertainment)	*Crysis* (Crytek)
EVE Online (CCP Games)	*World of Warcraft* (Blizzard Entertainment)	*Battlefield 2142* (DICE)
Hellgate: London (Flagship Studios)	*Gamebryo* (Emergent Technologies)	*Guild Wars* (ArenaNet)

For screenshots and testimonials, please see the PerfHUD page on developer.nvidia.com.



Crysis used courtesy of Crytek.

Chapter 2. Getting Started

This chapter will walk you through installing and configuring PerfHUD for the first time.

System Requirements

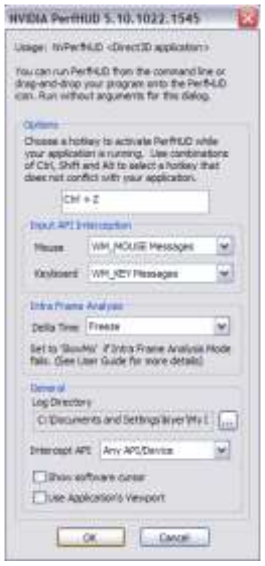
- An NVIDIA GPU: GeForce 6 Series and later GPUs are supported, as well as G80, G70, and NV4X-based Quadro FX GPUs. Older GPUs are supported with reduced functionality.
    - Microsoft DirectX 9.0c or Microsoft DirectX 10
    - Windows XP or Windows Vista

Installing Drivers

- **Windows Vista**
    - On Windows Vista, a stock driver of version 173 or later will allow you to use all the capabilities of PerfHUD.
- **Windows XP**
    - If you are using PerfHUD on Windows XP, you will need to replace your stock NVIDIA driver with the instrumented driver provided with **NVIDIA PerfKit**. Otherwise, certain portions of PerfHUD will not function properly, as they will not have the necessary access to the internals of the driver and GPU.

Configuring PerfHUD

Launch PerfHUD by double-clicking the desktop icon, or using the Start Menu. PerfHUD will display a configuration dialog that will let you customize it for your particular application.



NVIDIA PerfHUD 5.10.1022.1545

Usage: NVPerfHUD <Direct3D application>

You can run PerfHUD from the command line or drag-and-drop your program onto the PerfHUD icon. Run without arguments for this dialog.

Options

Choose a hotkey to activate PerfHUD while your application is running. Use combinations of Ctrl, Shift and Alt to select a hotkey that does not conflict with your application.

Ctrl + Z

Input API Interception

Mouse: WM_MOUSE Messages

Keyboard: WM_KEY Messages

Intra Frame Analysis

Delta Time: Freeze

Set to 'SlowMo' if Intra Frame Analysis Mode fails. (See User Guide for more details)

General

Log Directory: C:\Documents and Settings\[illegible]

Intercept API: Any API/Device

☐ Show software cursor

☐ Use Application's Viewport

The most commonly changed setting is the **Activation Hotkey**. This hotkey is what allows you switch input between PerfHUD and your application. Make sure to set it to something that does not conflict with other important keys in your application.

PerfHUD does have several other settings which can help you tune PerfHUD's methods of intercepting input and otherwise interacting with your application. Please see the PerfHUD 6 Reference for more information on these settings, or if you need help with troubleshooting.

Modifying your Application

Note: PerfHUD ships with two sample applications for you to analyze right out of the box without making **any** source code modificaitons. If you'd like to get working with PerfHUD right away, please skip to Using PerfHUD for the First Time.

PerfHUD requires a small code change in how you create your Direct3D device in order to enable analysis. Since PerfHUD lets you explore an application's rendering pipeline and methodology in extreme detail, we have built PerfHUD respect the application owners.

Note: Be sure you disable PerfHUD analysis in your application before you ship. Otherwise, anyone will be able use PerfHUD on your application.

One of the first functions called when setting up your graphics pipeline is the Direct3D CreateDevice() function that creates your display device. In your application it probably looks something like this:

```
HRESULT Res;
Res = g_pD3D->CreateDevice(
      D3DADAPTER_DEFAULT,
      D3DDEVTYPE_HAL,
      hWnd,
      D3DCREATE_HARDWARE_VERTEXPROCESSING,
      &d3dpp,
      &g_pd3dDevice
);
```

When your application is launched by PerfHUD, a special `NVIDIA PerfHUD` adapter is created. Select this adapter in order to give PerfHUD permission to analyze it. In addition, you must select the reference rasterizer as the device type, since some applications might select the

`NVIDIA PerfHUD` adapter ID unintentionally and expose themselves to unauthorized analysis. Your application will **not** actually use the reference rasterizer as long as you have selected the PerfHUD adapter.

Below is some sample code that will enable PerfHUD analysis for your DirectX9 application:

```
// Set default settings
UINT AdapterToUse=D3DADAPTER_DEFAULT;
D3DDEVTYPE DeviceType=D3DDEVTYPE_HAL;

#if SHIPPING_VERSION
// When building a shipping version, disable PerfHUD (opt-out)
#else
// Look for 'NVIDIA PerfHUD' adapter
// If it is present, override default settings
for (UINT Adapter=0;Adapter<g_pD3D->GetAdapterCount();Adapter++)
{
      D3DADAPTER_IDENTIFIER9  Identifier;
      HRESULT                 Res;

Res = g_pD3D->GetAdapterIdentifier(Adapter,0,&Identifier);
      if (strstr(Identifier.Description,"PerfHUD") != 0)
      {
            AdapterToUse=Adapter;
            DeviceType=D3DDEVTYPE_REF;
            break;
      }
}
#endif

if (FAILED(g_pD3D->CreateDevice( AdapterToUse, DeviceType, hWnd,
            D3DCREATE_HARDWARE_VERTEXPROCESSING,
            &d3dpp, &g_pd3dDevice) ) )
{
      return E_FAIL;
}
```

For DirectX10, use the following code example:

```
#if SHIPPING_VERSION
// When building a shipping version, disable PerfHUD (opt-out)
#else
// Look for 'NVIDIA PerfHUD' adapter
// If it is present, override default settings
IDXGIFactory *pDXGIFactory;
ID3D10Device *pDevice;
HRESULT hRes;

hRes = CreateDXGIFactory(__uuidof(IDXGIFactory), (void**)&pDXGIFactory);

// Search for a PerfHUD adapter.
UINT nAdapter = 0;
IDXGIAdapter* adapter = NULL;
IDXGIAdapter* selectedAdapter = NULL;
D3D10_DRIVER_TYPE driverType = D3D10_DRIVER_TYPE_HARDWARE;
```

```
while (pDXGIFactory->EnumAdapters(nAdapter, &adapter) !=
DXGI_ERROR_NOT_FOUND)
{
if (adapter)
{
DXGI_ADAPTER_DESC adaptDesc;
if (SUCCEEDED(adapter->GetDesc(&adaptDesc)))
{
const bool isPerfHUD = wcscmp(adaptDesc.Description, L"NVIDIA PerfHUD") == 0;

// Select the first adapter in normal circumstances or the PerfHUD one if it
exists.

if(nAdapter == 0 || isPerfHUD)
selectedAdapter = adapter;

if(isPerfHUD)
driverType = D3D10_DRIVER_TYPE_REFERENCE;

}
}
++nAdapter;

}

#endif

            if(FAILED(D3D10CreateDevice( selectedAdapter, driverType, NULL,
0, D3D10_SDK_VERSION, &pDevice)))
                  return E_FAIL;
```

This will enable PerfHUD analysis when you want to use it, and ensure that your application does not use the software reference rasterizer when run normally.

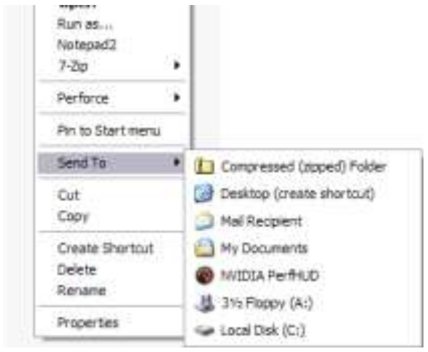
Note: Remember to use the **`NVIDIA PerfHUD`** device whenever your application enumerates devices, checks for capabilities, and so on. Otherwise you may see rendering errors.

Launching your Application

To launch PerfHUD, right-click on your application, and select the **NVIDIA PerfHUD** option in the *Send To* menu.

You can also drag the application's icon onto the PerfHUD launcher on the desktop.

Your application should now launch, with PerfHUD running on top of it.



Chapter 3. Using PerfHUD for the First Time

You've installed and configured PerfHUD! Congratulations, a new world of performance analysis is now open to you. This chapter teaches you the basics of debugging and profiling with PerfHUD and takes you on a step-by-step tour of PerfHUD 6.0's most popular features.

Overview

PerfHUD provides four different ways to look at your application's performance. By switching between them, you can identify large-scale performance problems, per-frame issues, and drill down all the way to a detailed analysis of the draw calls in a particular frame. The four modes are:

F5 Performance Dashboard
This mode lets you interact with your application normally, all while viewing real-time graphs of internal GPU and Direct3D performance metrics. You can also speed up and slow down your application to make it easier to identify problem frames, and do experiments like globally replacing all textures with 2x2 dummy textures.

F6 Debug Console
Review messages from the DirectX Debug runtime, PerfHUD warnings and custom messages from your application.

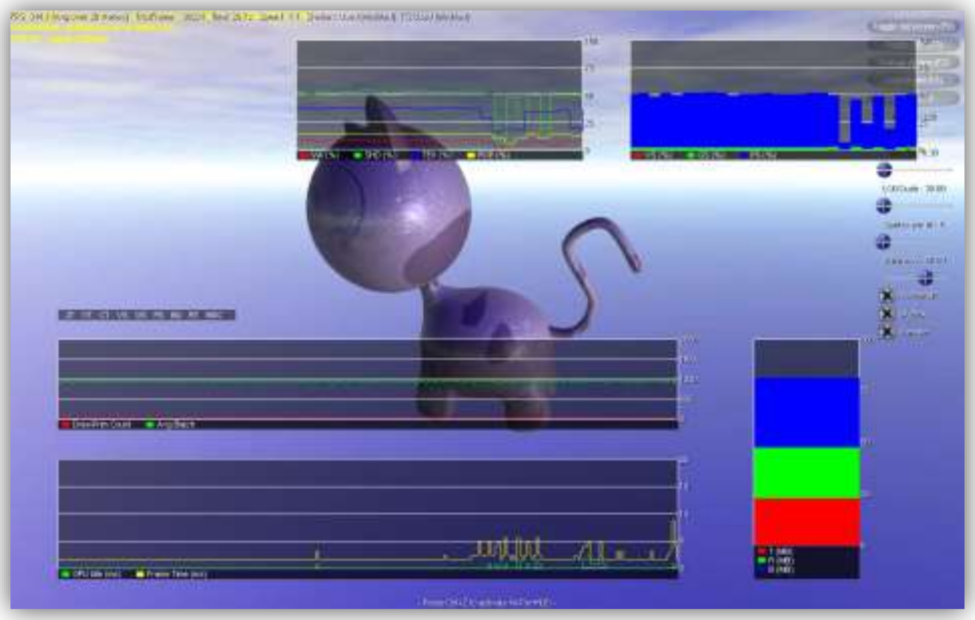
F7 Frame Debugger
Freeze the current frame and step through it one draw call at a time, drilling down to investigate the setup for each stage of the graphics pipeline using advanced State Inspectors that show details for each stage in the graphics pipeline.

F8 Frame Profiler
Profile your application's usage of the GPU. This is the most powerful mode that PerfHUD offers, allowing you to sort all draw calls in the current frame by cost. In addition, several performance graphs and analysis tools are available.

Using the Performance Dashboard

1. **Browse to a sample program, and launch the application with PerfHUD.**

   PerfHUD ships with two sample applications, FogPolygonVolume for DX9 and Sparkles for DX10, which are already properly modified with the opt-in code. The program should start with PerfHUD's Performance Dashboard as an overlay on top, as shown below.



2. **Activate PerfHUD by typing the Activation Hotkey.**

   The hotkey is Ctrl-Z by default, but you can configure it in any way you like. You'll see the status line at the bottom of the screen change to four buttons, one for each mode of PerfHUD:

   Now, any keyboard or mouse input you make will affect PerfHUD. You can toggle between PerfHUD and your application at any time. For example, you may want to navigate to a different part of the scene to analyze it, and then re-activate PerfHUD again.

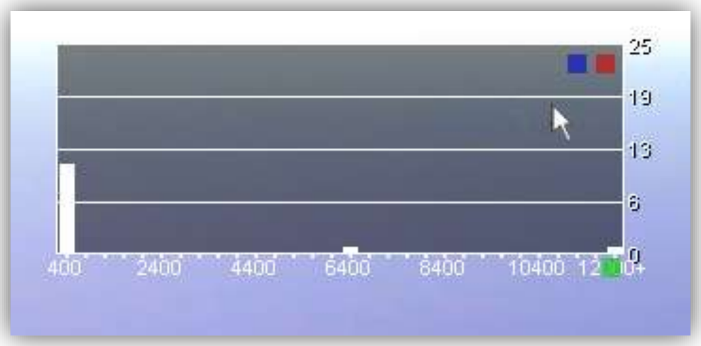
Note: Once PerfHUD is activated using the activation hotkey, all subsequent keyboard events are intercepted by PerfHUD. When you are done analyzing your application and want to close it, you'll have to press the activation hotkey again to disable PerfHUD so that you can use the keyboard and mouse to quit your application.

3. **Speed up and slow down time in your application.**

   By pressing the + and – keys, you can scale the passing of time in your application from 6x normal speed down to 1/8 normal speed. Controlling time is helpful when you want to find a particularly troublesome set of frames.

4. **Perform a global experiment.** Hit Ctrl-T to instantaneously switch all textures in your application to 2x2 dummy textures, or Ctrl-D to see a view of the depth complexity of your frame buffer. There are several other experiments you can run as well. Please see the PerfHUD 6 Reference Guide for full details.

5. **Customize a graph in the Performance Dashboard.** Each graph is completely customizable. You can move the graphs by dragging them, add new signals to existing graphs, or create a completely new graph.

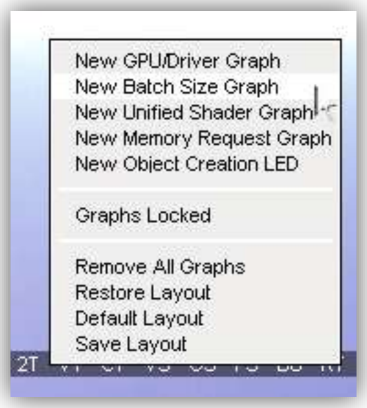


Left-clicking on the blue box brings up a configuration dialog, for adding and removing signals.

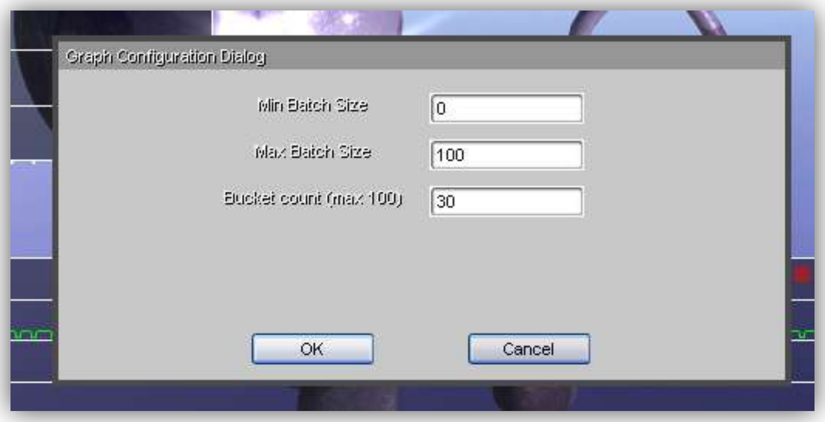
Left-clicking and dragging on the green box resizes the current graph.

Left-clicking on the red box closes the current graph.

6. **Create a new graph in the Performance Dashboard.** Right-click on the background and choose New Batch Size Graph from the context menu.



7. **Customize the new graph.** Click on the **blue box** to see the Graph Configuration Dialog. Set the **Maximum Batch Size** to **100**. Then click **OK**. The graph will now show more bars. Different graph types have different signals and options available.

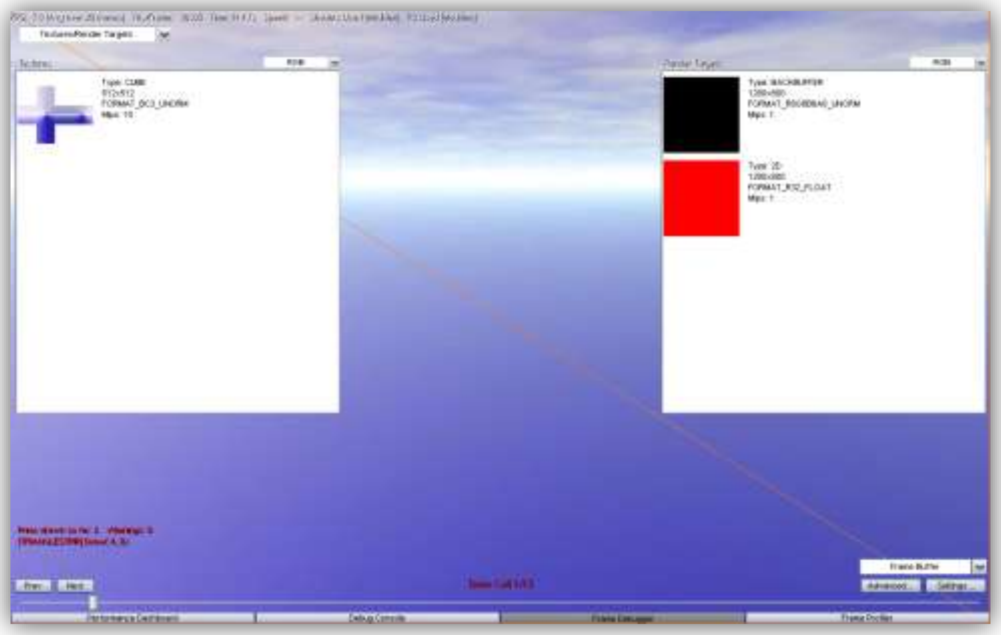


Min Batch Size	0
Max Batch Size	100
Bucket count (max 100)	30

Using the Frame Debugger

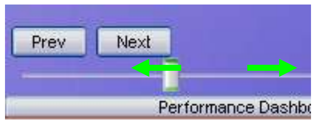
The Performance Dashboard is most useful for finding a troublesome spot in your scene. Once you've found that spot, you will often want to freeze the frame, debug its draw calls, and analyze its performance in detail.

1. **Press F7 to switch to the Frame Debugger**. The Frame Debugger will show you just the first draw call in the scene:

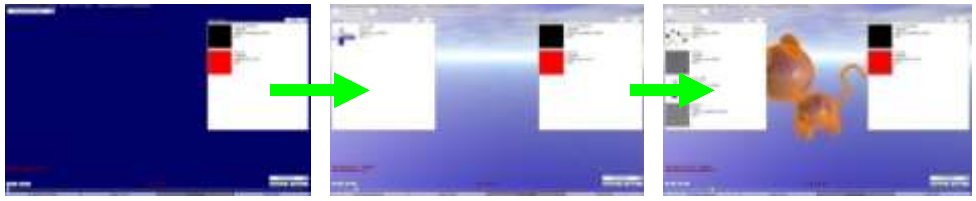


2. **Scrub through the frame and watch how the draw calls accumulate into the final frame.**

Click and drag the slider at the bottom of the screen from side to side.



You'll see how the frame builds up with various draw calls.



The current draw call is highlighted with an orange wireframe.

You can use the up and down arrow keys to decrement or increment the current draw call. Home jumps to the first draw call, and End jumps to the last draw call.

Page Up and Page Down decrement or increment the current draw call by larger amounts.

3. Switch between different views on the same draw call.

In the upper-left corner of the frame debugger is a combo-box with **4** different options for viewing stuff.

a. **Textures:** View and modify the textures and render targets associated with the current draw call.

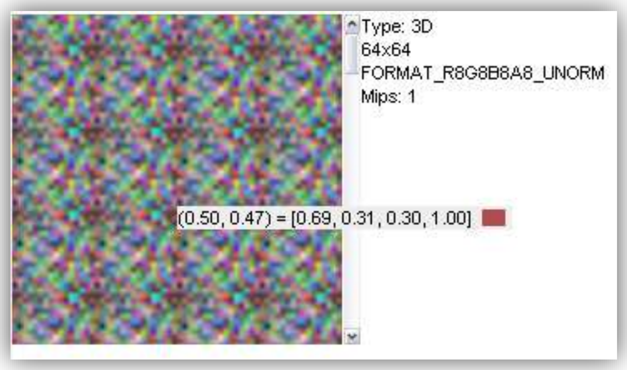
b. **Call List:** View the entire Direct3D call list.

c. **Dependencies:** View both producer and consumer dependencies between draw calls highlighted right on the scrub bar.

d. **Perf Events:** View application specific performance labels which describe portions of the frame. Click on any label to jump immediately to the corresponding draw call.

4. View Textures and Render Targets for each draw call.

All the textures used by the current draw call are shown in the Textures panel on the left of the screen. Click on the Textures panel (to get focus) and press + twice to enlarge the textures. (Pressing - will reduce the textures.) Note that if you hover over a texture, a tooltip will appear showing u-v coordinates and RGBA color information.



Type: 3D
64x64
FORMAT_R8G8B8A8_UNORM
Mips: 1

(0.50, 0.47) = [0.69, 0.31, 0.30, 1.00] 🟥

On the right is the list of Render Targets. You can perform the same operations in that panel as in the Textures panel.

5. Modify a texture by replacing it with a mipmap visualization texture.

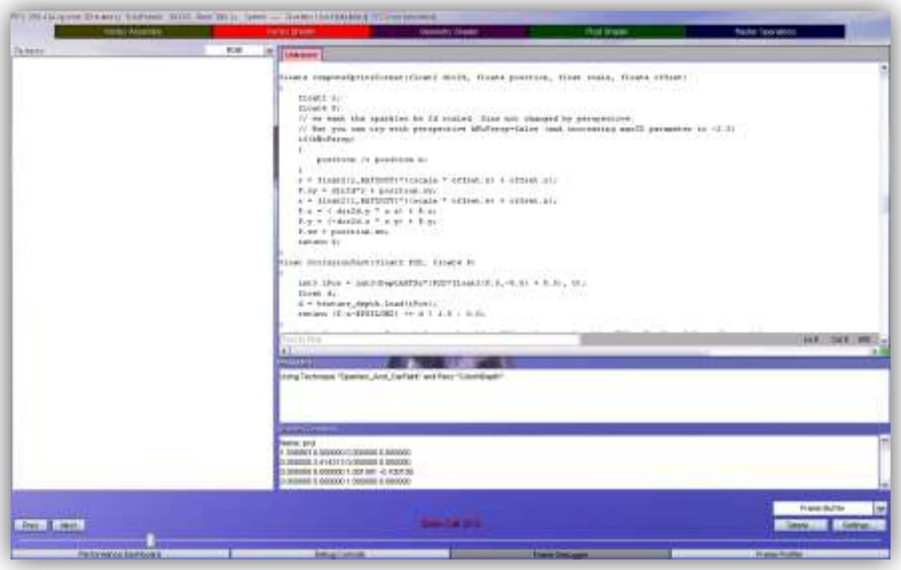
You can replace any texture directly in your live application by right-clicking on the texture and choosing from a set of debug textures. The mipmap visualization texture is especially useful for determining how well your textures are mipped.

6. **Switch back to the Performance Dashboard to see your edits now running in your live game.**

   All edits you make while in the Frame Debugger are carried back and override your application's settings. Switch back to the Frame Debugger (F7) when you are ready to continue.

7. **Click the *Advanced* button to see and edit even more information.**

   The Advanced button is located in the lower right corner of the screen. Once you click it, the advanced state inspector is shown. You can **view and edit** data for each stage of the graphics pipeline, including "edit and continue" functionality on any shader, or raster operations like the blend function.



Using the Frame Profiler

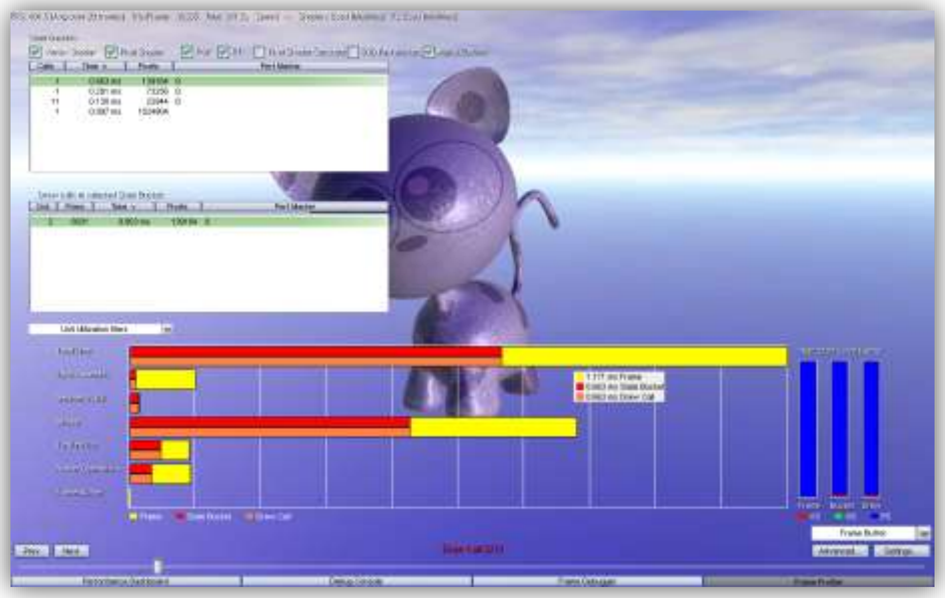
The Frame Profiler gives you extremely detailed performance information per draw call statistics about performance and GPU usage. This is one of the uniquely powerful features of PerfHUD – complete bottleneck analysis with just one key press.

1. **Press F8 to switch to the Frame Profiler.**

   PerfHUD will quickly run a series of performance tests on the current frame, and then the frame profiler will open.

2. **Examine the Unit Utilization bars, and scrub the frame using the Draw Call Slider.**

   The default graph is the Unit Utilization bars. This graph shows you how long each GPU unit was used for the selected draw call, state bucket, and the frame. You can change the selected draw call by using the scrubber, just as you would in the Frame Debugger, and you can define state bucket groupings using the checkboxes at the top of the screen.



3. **Change the graph shown to see different sets of statistics.**

   Click the Combo box on the middle left border of the screen. As you can see, the Unit Utilization Bars graph is just the tip of the iceberg! You can also graph draw call duration, shaded pixels, and % utilization among others.

Profiling Effectively with PerfHUD

The GPU is a pipelined processor, and thus it is critically important to identify and optimize the largest bottlenecks first, as optimizing a non-bottleneck will lead to little or no performance gain.

By using PerfHUD's various modes effectively, you can do just that.

1. **Always check whether your application is CPU bound or GPU bound.**

   If your application is CPU-bound, then no amount of GPU optimization will speed it up. You can use the PerfHUD's Performance Dashboard to quickly tell if your application is CPU or GPU bound in two ways.

   The first method is to examine the graph that is displaying the *Total Frame Time* and *Driver Time* counters. If your application is CPU-bound, you'll see a big gap between the yellow line ("Total Frame Time") and the red line ("Driver Time").

   Another easy way to check if your application is CPU-bound is to press "N" to ignore all draw calls. If your frame rate doesn't increase, then even an infinitely fast GPU wouldn't help your application run faster – therefore, it is definitely limited by your CPU speed.

   In this case, you should use a CPU performance analyzer such as Intel's VTune or AMD's CodeAnalyst to make your CPU code more efficient.

2. **Solve rendering errors using the Frame Debugger.**

   Use your application just like you normally do in the Performance Dashboard. If you see any graphical errors, you can immediately switch to the Frame Debugger by typing F7, and then examine what geometry, textures, shaders, raster operations are used for each draw call.

3. **Solve performance issues with the Frame Profiler.**

   The Frame Profiler (GeForce 6 Series or later required) provides advanced profiling features which can help you quickly identify your performance issue. It provides automated performance analysis, giving you very detailed information about your draw calls and time spent in the various GPU stages, as well as other useful GPU statistics. It also allows you to group draw calls into buckets to identify specific types of bottlenecks.

If you don't have a GeForce 6 Series or later GPU, you can use the global experiments in the Performance Dashboard to get a general notion of your bottleneck.

Recommended Links

- [Link] PerfHUD Introductory Video
- NVIDIA Developer Web Site
  http://developer.nvidia.com
- [Link] Optimize your GPU with the Latest NVIDIA Performance Tools
- [Link] *NVIDIA GPU Programming Guide* – all the latest tips and tricks
- [Link] *Balancing the Graphics Pipeline for Optimal Performance*
- [Link] NVShaderPerf – shader performance analysis utility
- [Link] NVIDIA SDK – hundreds of code samples & effects
- [Link] PerfKit User Guide
- [Link] *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*
  Several of the performance-related chapters are particularly helpful.
- [Link] GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation
- [Link] *GPU Gems 3*
- [Link] Microsoft DirectX web site
- [Link] Microsoft Developer Network (MSDN) web site
  Search for "performance" and "optimization"
- Microsoft DirectX SDK documentation [in the Start menu after installation]

Frequently Asked Questions

PerfHUD says my application is not enabled for PerfHUD analysis.

To ensure that unauthorized third parties do not analyze your application without your permission, you must make a minor modification to enable PerfHUD analysis. Refer to the Getting Started section of this User Guide for instructions.

No data is reported in the Unit Utilization Graph in Performance Dashboard mode and/or Frame Profiler mode doesn't seem to work.

Both the Unit Utilization Graph and Frame Profiler require performance signals from PerfKit. Make sure PerfKit is installed and you are using a GPU that is supported by PerfKit.

My application does not respond while PerfHUD is active.

When PerfHUD is enabled using the hotkey feature, it consumes all keyboard input and does not pass any key stroke events to the application. You can toggle this mode on/off using the activation hotkey you selected.

My application does not respond while PerfHUD is active.

When PerfHUD is enabled using the hotkey feature, it consumes all keyboard input and does not pass any key stroke events to the application. You can toggle this mode on/off using the activation hotkey you selected.

I can see the PerfHUD header across the top of my screen, but it doesn't respond to my activation hotkey.

PerfHUD uses several methods of intercepting key stroke events. If you are using a method that is not yet supported, please let us know so we can update PerfHUD.

Note that in Win2K PerfHUD uses DirectInput to listen for your activation hotkey and to intercept keyboard commands while activated. DirectInput supplies two types of data: buffered and immediate. Buffered data is a record of events that are stored until an application retrieves them. Immediate data is a snapshot of the current state of a device.

What this means is that your application needs to use the IDirectInputDevice8::GetDeviceData interface instead of the IDirectInputDevice8::GetDeviceState interface if you want to access advanced features of PerfHUD such as bottleneck identification experiments, shader visualization, etc.

PerfHUD messes up alpha and some rendering states.

PerfHUD renders the HUD at the end of the frame. It also changes the rendering states to draw itself, but does not restore them to their original state. In other words, it does not push and pop rendering states for performance reasons—therefore, it is assumed that your application resets the rendering states at the beginning of each frame.

The GPU_IDLE (GREEN) line is not reporting any data.

This information is not available for GeForce4 (NV25) and older GPUs. You may also see this on newer GPUs if PerfHUD is unable to communicate properly with the display driver. Verify that you are using the latest version of PerfHUD and running the latest NVIDIA display drivers.

In the Frame Profiler's Advanced mode, why can't I use the RGB dropdown to visualize the individual channels of render targets in Raster Operations?

Currently the RGB dropdown only works for textures. It will be grayed out unless at least one texture is present.

Can I use PerfHUD without an instrumented driver?

Yes, PerfHUD will work with a normal driver, but you will not get access to performance counters. Therefore, the graphs in the Performance Dashboard will not work, and automated performance analysis in the Frame Profiler will not work either.

However, you can still use the pipeline experiments in the Performance Dashboard, as well as the Debug Console and Frame Debugger.

Please note that you must still have an NVIDIA GPU in order to use PerfHUD, whether you are using an instrumented driver or not.

I see some extra lines in the middle of my PerfHUD graphs.

If you are using very old drivers you may see portions of the old PerfHUD 1.0 graphs super-imposed on top of your PerfHUD graphs. Upgrading your drivers to 71.8x or later should fix the problem.

Some objects in my scene continue to animate when my application is frozen.

The PerfHUD time control feature stops the clock for your application, allowing you to perform in-depth analysis of the current frame while it is frozen. Your application must use and rely on the QueryPerformanceCounter() or timeGetTime() win32 functions. If your application uses the **rdtsc** instruction it will not function properly with Frame Debugger Mode, Frame Profiler Mode, or the time control features in Performance Dashboard.

If your application has implemented a frame rate limiter, you may need to disable this functionality to use the time control, debugging and profiling features of PerfHUD.

If your application uses frame-based animation, freezing time will have no effect on animated objects.

What else do I need to know about PerfHUD?

- Multi sampled render targets at not displayed in Frame Debugger Mode.
- PerfHUD may crash if you turn on "Break on D3D errors" in the DirectX Control Panel.
- Pixel shader visualization does not work for primitives that use VS 3.0.

I have discovered a problem that is not listed above

We want to make sure PerfHUD continues to be a useful tool for developers analyzing their applications. Please let us know if you encounter any problems or think of additional features that would be helpful while using PerfHUD.

PerfHUD@nvidia.com