NVIDIA®

パフォーマンス・ツール

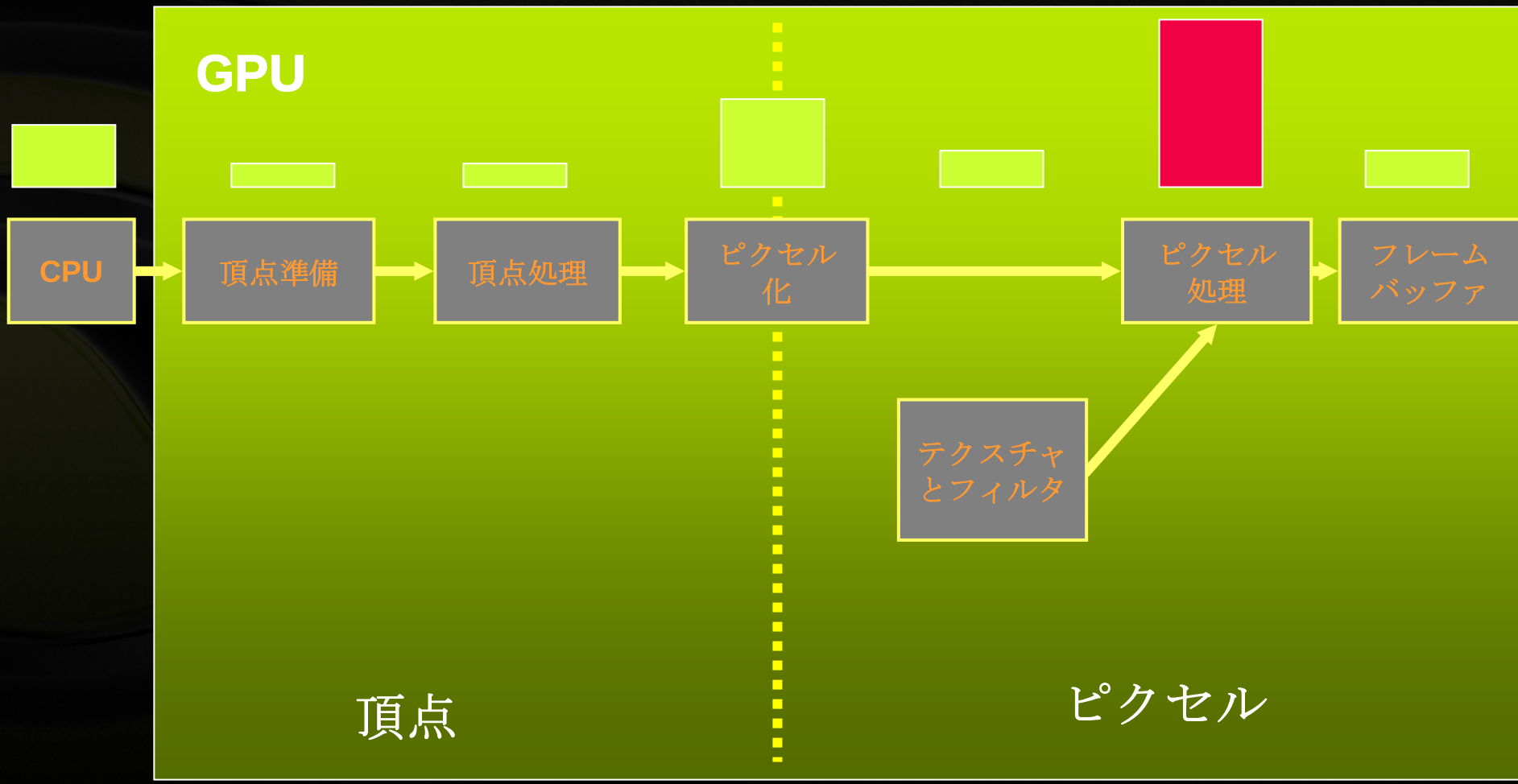
パフォーマンス・ツール概要

- **GPU**アーキテクチャについて
- **NVGLExpert: OpenGL API**の援助
- **NVShaderPerf:** シェーダのパフォーマンス
- **NVPerfKit:** ドライバと**GPU**のパフォーマンス
- **NVPerfHUD:** 対話式パフォーマンス解析

GPUアーキテクチャ

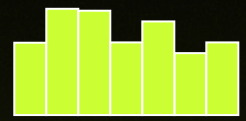
- パイプライン構成
  - 各ユニットは、仕事をするのに前のユニットからのデータが必要
- ボトルネックの発見と解消
- パイプラインを平均化

GPUパイプライン構成(簡略図)

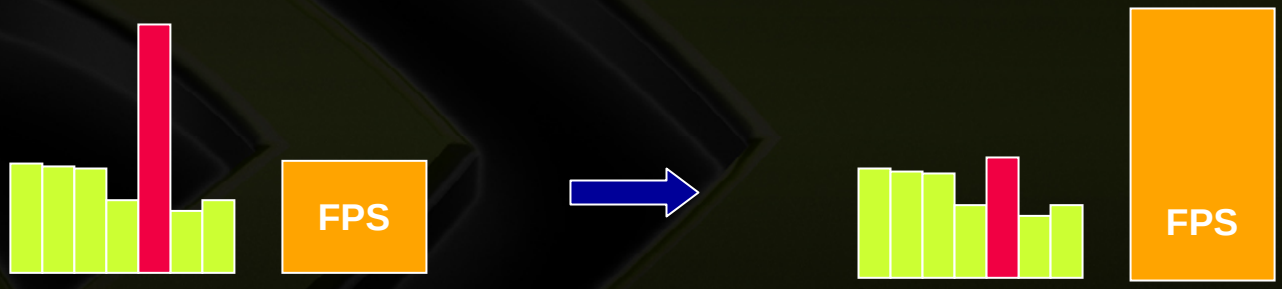


単一ユニットによりパイプライン速度が制限

ボトルネックの発見

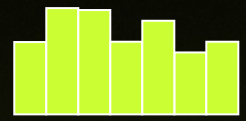


- ユニットの仕事量を変化させる
  - 負荷を低減

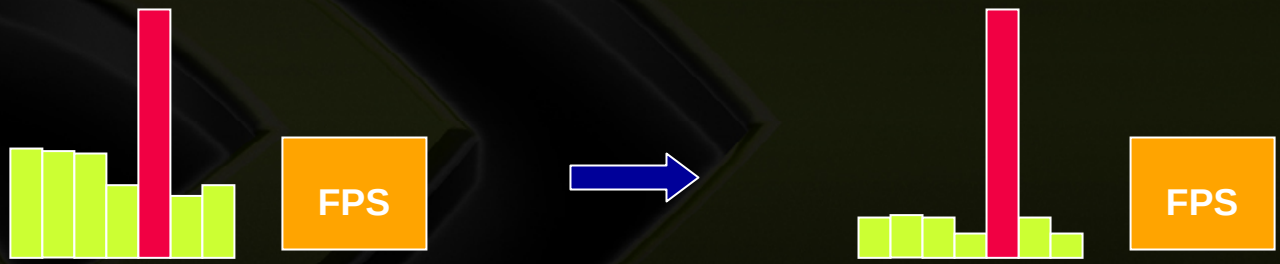


- パフォーマンス（**FPS**）の改善があれば、このユニットがボトルネック
- 他ユニットの仕事量を変えないように注意

ボトルネックの発見

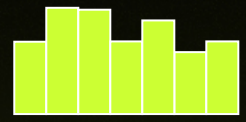


- 他ユニットの可能性を排除
  - 仕事量をほとんどなくす



  - パフォーマンスに顕著な変化が見られなければ、このユニットがボトルネック
  - 検証中ユニットの負荷を変えないように注意

ボトルネックの発見



- パイプライン中各点の計測
  - **NVPerfKit**と**NVPerfHUD**を使用
  - 最大処理速度と比較した実質処理量

NVGLExpert

- 概要
- プロジェクトの現状

概要

- CPU処理の問題を解消
- 計測用OpenGLドライバ
  - ファイル、コンソール、またはデバッガへの情報出力
  - 各グループ/レベルでの情報詳細
- 簡単なGUIツールで操作
  - Windows版ではレジストリを設定
  - Linux版では環境変数を設定
- 現時点での機能
  - OpenGLエラーの出力
  - ドライバがソフトウェア処理を使っているか
  - シェーダのコンパイルエラー
  - VBOの保存領域
  - GL_FRAMEBUFFER_UNSUPPORTED_EXTの理由
- 機能はこれからも拡張予定

プロジェクトの現状

- 次の大きなドライバ配布で公開
- **Windows**ならびに**Linux**
- 現在は**NV3x**と**NV4x**のみ
- 他に必要な情報は**?**

NVGLExpert@nvidia.com

NVShaderPerf

- 概要
- バージョン**1.8**での更新
- バージョン**2.0**で更新になる事項

NVShaderPerf

入力:
- HLSL
- GLSL(フラグメント)
- !!FP1.0
- !!ARBfp1.0
- PS1.x,PS2.x,PS3.x
- VS1.x,VS2.x, VS3.x
- Cg

NVShaderPerf

出力:
- アセンブリ
- サイクル数
- 一時レジスタ数
- ピクセル処理量
- 全てfp16と全てfp32のケース

GPUアーキテクチャ:
- GeForce 7800 GTX
- GeForce 6X00 FXシリーズ
- Quadro FXシリーズ

C:\WINDOWS\system32\cmd.exe

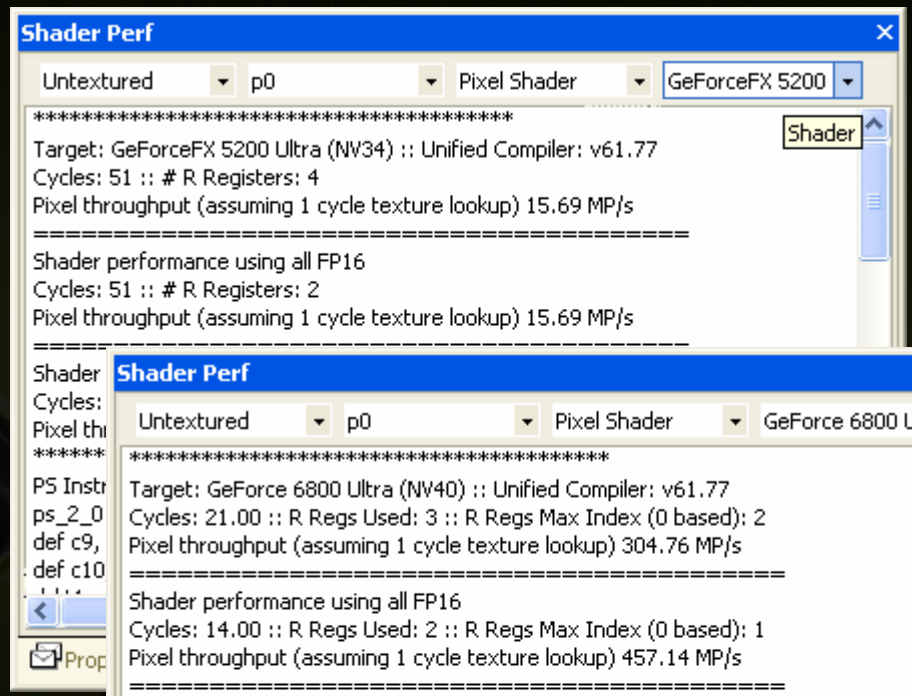
```
    dp3 r0.x, r1, r1
    rsq r0.w, r0.x
    nrm r0.xyz, t1
    mad r1.xyz, r1, r0.w, r0
    nrm r2.xyz, r1
    nrm r1.xyz, t2
    dp3 r2.x, r2, r1
    max r1.w, r2.x, c9.x
    pow r0.w, r1.w, c5.x
    add r1.w, r0.w, -c7.x
    mov r2.w, c6.x
    add r2.w, r2.w, -c7.x
    rcp r2.w, r2.w
    mul_sat r2.w, r1.w, r2.w
    mad r1.w, r2.w, c9.y, c9.z
    mul r2.w, r2.w, r2.w
    mul r1.w, r1.w, r2.w
    mov r2.x, c9.w
    add r2.w, r2.x, -c8.x
    mad r1.w, r1.w, r2.w, c8.x
    dp3 r0.x, r0, r1
    mul r0.w, r0.w, r1.w
    mul r1.xyz, r0.w, c4
    add r0.w, r0.x, c9.w
    mul r0.w, r0.w, c10.x
    mov r0.xyz, c0
    add r0.xyz, r0, -c1
    mad r0.xyz, r0.w, r0, c1
    mov r2.xyz, c2
    add r2.xyz, r2, -c3
    mad r2.xyz, r0.w, r2, c3
    mov r0.w, c9.w
    mul r1.x, r1.x, c9.w
    slots used
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v61.7
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index (0 based
Pixel throughput (assuming 1 cycle texture lookup) 304.76 M
=================================================
Shader performance using all FP32
Cycles: 14.00 :: R Regs Used: 2 :: R Regs Max Index (0 based
Pixel throughput (assuming 1 cycle texture lookup) 457.14 M
=================================================

Shader performance using all FP32
Cycles: 21.00 :: R Regs Used: 3 :: R Regs Max Index (0 based
Pixel throughput (assuming 1 cycle texture lookup) 304.76 M

C:\Temp\NVShaderPerf_61_77>
```

NVShaderPerf: 制作工程での使用

- 現在のパフォーマンス
  - 目標サイクル数に対して
  - 最適化の可能性を検証
- 自動的復帰試験
- **FX Composer 1.7**への統合

FX Composer 1.7 – Shader Perf

- アセンブリ
- 対象となる**GPU**
- ドライバのバージョンとの適合
- サイクル数
- レジスタ数
- ピクセル処理量
- 全て**fp16**と全て**fp32**のケース **(**パフォーマンス範囲の特定**)**



NVShaderPerf 1.8

- **GeForce 7800 GTX**と**Quadro FX 4500**のサポート
- **ForceWare 77.72**から統合されたコンパイラ
- 分岐パフォーマンスの向上
  - デフォルトではシェーダ中最長のパスを計算
  - **–minbranch**を使用すれば最短パスを計算

NVShaderPerf 1.8

```
////////////////////////////////////////////////////////////////////
 // determine where the iris is and update normals, and lighting parameters to simulate iris geometry
 ////////////////////////////////////////////////////////////////////

 float3 objCoord = objFlatCoord;
 float3 objBumpNormal = normalize( f3tex2D( g_eyeNermel, v2f.UVtex0 ) * 2.0 - float3( 1, 1, 1 ) );
 objBumpNormal = 0.350000 * objBumpNormal + ( 1 - 0.350000 ) * objFlatNormal;
 half3 diffuseCol = h3tex2D( g_irisWhiteMap, v2f.UVtex0 );
 float specExp = 20.0;
 half3 specularCol = h3tex2D( g_eyeSpecMap, v2f.UVtex0 ) * g_specAmount;

 float tea;

 float3 centerToSurfaceVec = objFlatNormal; // = normalize( v2f.objCoord )
 float firstDot = centerToSurfaceVec.y; // = dot( ce
 if( firstDot > 0.805000 )
 {
  // We hit the iris.  Do the math.

  // we start with a ray from the eye to the surface
  float3 ray_dir = normalize( v2f.objCoord - objEy
  float3 ray_origin = v2f.objCoord;

  // refract the ray before intersecting with the iris
  ray_dir = refract( ray_dir, objFlatNormal, g_refra

  // first, see if the refracted ray would leave the e
  float t_eyeballSurface = SphereIntersect( 16.0, r
  float3 objPosOnEyeBall = ray_origin + t_eyeball
  float3 centerToSurface2 = normalize( objPosOn

  if( centerToSurface2.y > 0.805000 )
  {
   // Display a blue color
   diffuseCol = float3( 0, 0, 0.7 );
   objBumpNormal = objFlatNormal;
   specularCol = float3( 0, 0, 0 );
   specExp = 10.0;
  }
  else
  {
   // transform into irisSphere space
   ray_origin.y -= 5.109000;

   // intersect with the Iris sphere
   float t = SphereIntersect( 9.650000, ray_origin, ray_dir );
   float3 SphereSpaceIntersectCoord = ray_origin + t * ray_dir;
   float3 irisNormal = normalize( -SphereSpaceIntersectCoord );
```

Lunaの目を描画するシェーダ
最長パスで674サイクル
最短パスで193サイクル

```
C:\WINDOWS\System32\cmd.exe

T:\tmp>t:\sw\devrel\sdk\tools\bin\release_pdb\nvshperf\nvshaderperf -a NV40 corn
ea2.txt
----------------------------------------------------------------------------

Running performance on file Cornea2.txt
-------------------- NV40 --------------------
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v77.72
Cycles: 674.25 :: R Regs Used: 12 :: R Regs Max Index (0 based): 11
Pixel throughput (assuming 1 cycle texture lookup) 9.50 MP/s

T:\tmp>t:\sw\devrel\sdk\tools\bin\release_pdb\nvshperf\nvshaderperf -minbranch -
a NV40 cornea2.txt
----------------------------------------------------------------------------

Running performance on file Cornea2.txt
-------------------- NV40 --------------------
Target: GeForce 6800 Ultra (NV40) :: Unified Compiler: v77.72
Cycles: 192.82 :: R Regs Used: 12 :: R Regs Max Index (0 based): 11
Pixel throughput (assuming 1 cycle texture lookup) 33.33 MP/s

T:\tmp>_
```

NVShaderPerf – バージョン 2.0

- 頂点処理量
- **GLSL**頂点シェーダ
- ひとつの**NVShaderPerf**で複数バージョンのドライバに対応
- 他に必要な機能は**?**

NVShaderPerf@nvidia.com

NVPerfKit

- 概要
- 関連ツール
- **NVPerfKit 2.0**

NVPerfKit: 解答を提供

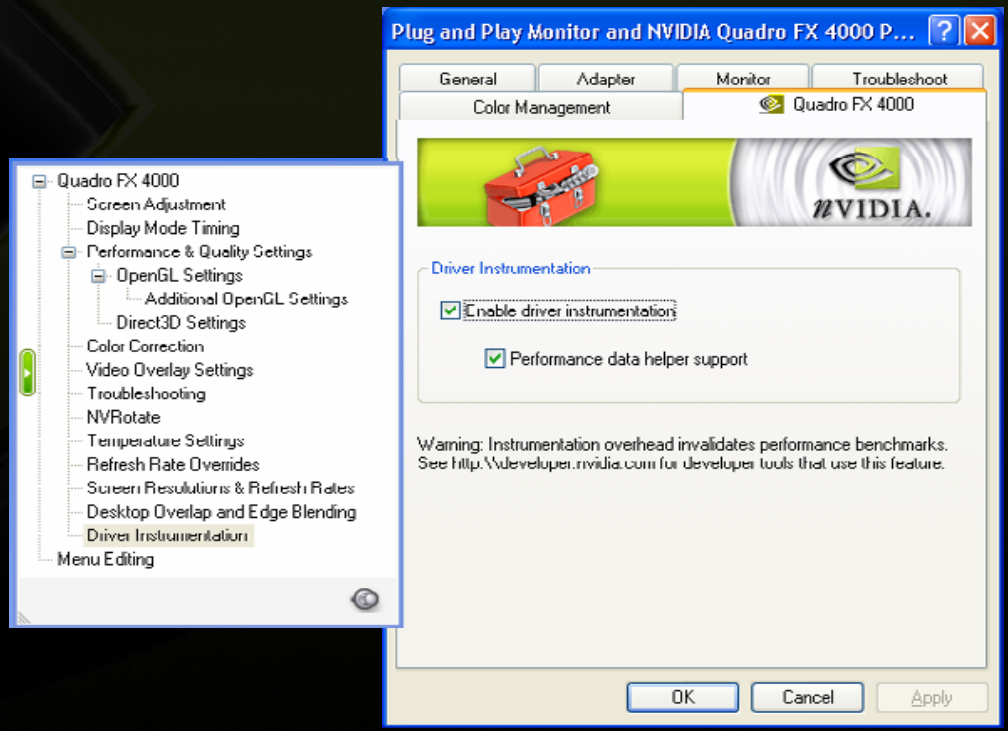
- CPU最適化をしても、13FPSで走るアプリ?
- GPU内で何が起こっているのか?
- なぜIHVエンジニアにはそれが分かるのか?

概要

- ドライバとGPUのパフォーマンス測定カウンタ
  - Performance Data Helper (PDH)
  - Microsoft PIX for Windows
- アプリケーションでNVPerfHUDの機能を実現
- アプリケーションでサンプリングを起動
- OpenGLとDirect3D

NVPerfKitで提供されるもの

- 測定用ドライバ
  - ドライバとGPUのカウンタ
  - OpenGLとDirect3Dのサポート
  - SLIサポート
- ツール
  - NVDevCPL
  - PIX Plugin
  - NVAppAuth
- SDK
  - サンプルコード
  - ヘルパークラス
  - ドキュメント



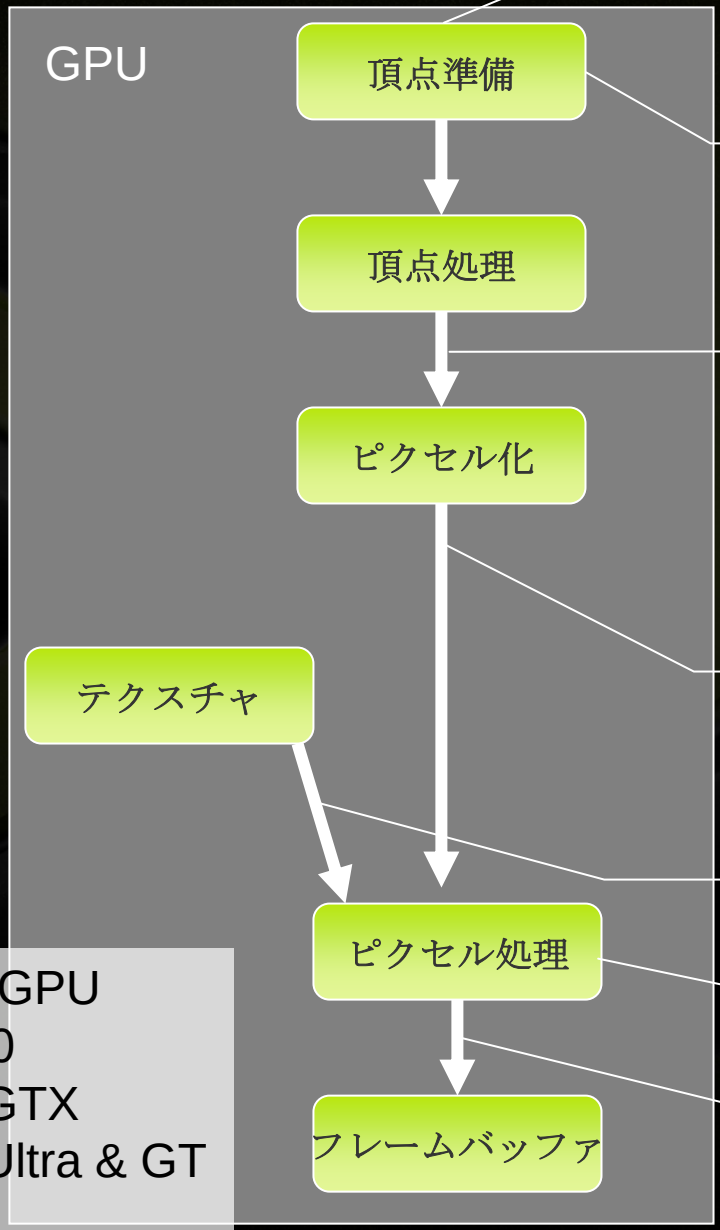
OpenGLカウンタ

説明	名前
FPS	OGL FPS
フレーム時間 (1/FPS)	OGL frame time mSec
ドライバのGPU待ち時間	OGL frame mSec Sleeping

Direct3Dカウンタ

説明	名前
FPS	D3D frame FPS
フレーム時間 (1/FPS)	D3D frame time mSec
AGPメモリ使用量	D3D frame agpmem MB
ビデオ・メモリ使用量	D3D frame vidmem MB
ドライバ時間	D3D frame mSec in driver
ドライバのGPU待ち時間	D3D frame mSec Sleeping
三角形数	D3D frame tris
バッチ数	D3D frame num batches
ロックされた描画ターゲット数	D3D Locked Render Targets

GPUカウンタ



gpu_idle
vertex_attribute_count

vertex_shader_busy

culled_primitive_count
primitive_count
triangle_count
vertex_count

fast_z_count
shaded_pixel_count

shader_waits_for_texture

pixel_shader_busy

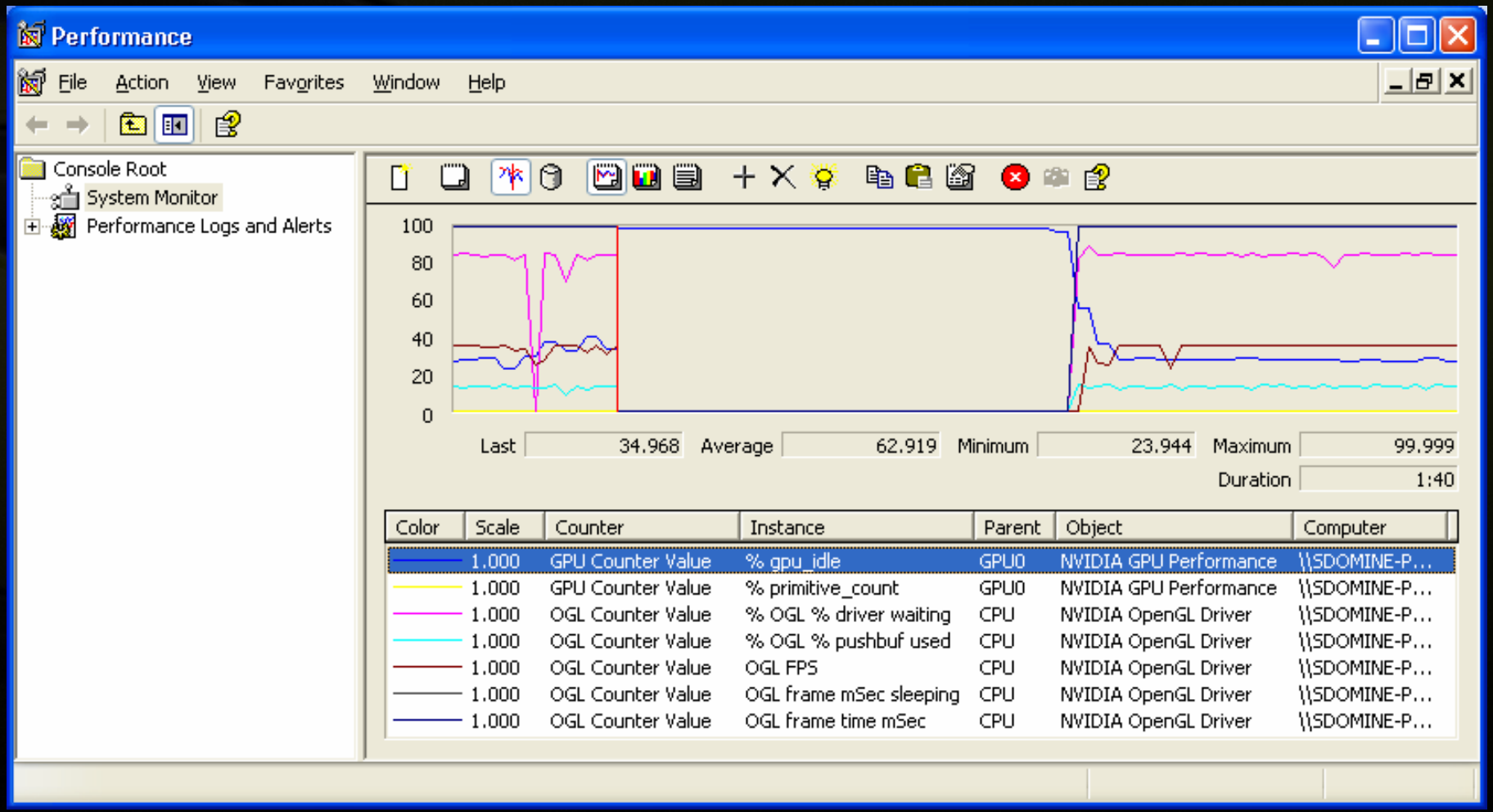
shader_waits_for_rop

サポートされるGPU
Quadro FX 4500
GeForce 7800 GTX
GeForce 6800 Ultra & GT
GeForce 6600

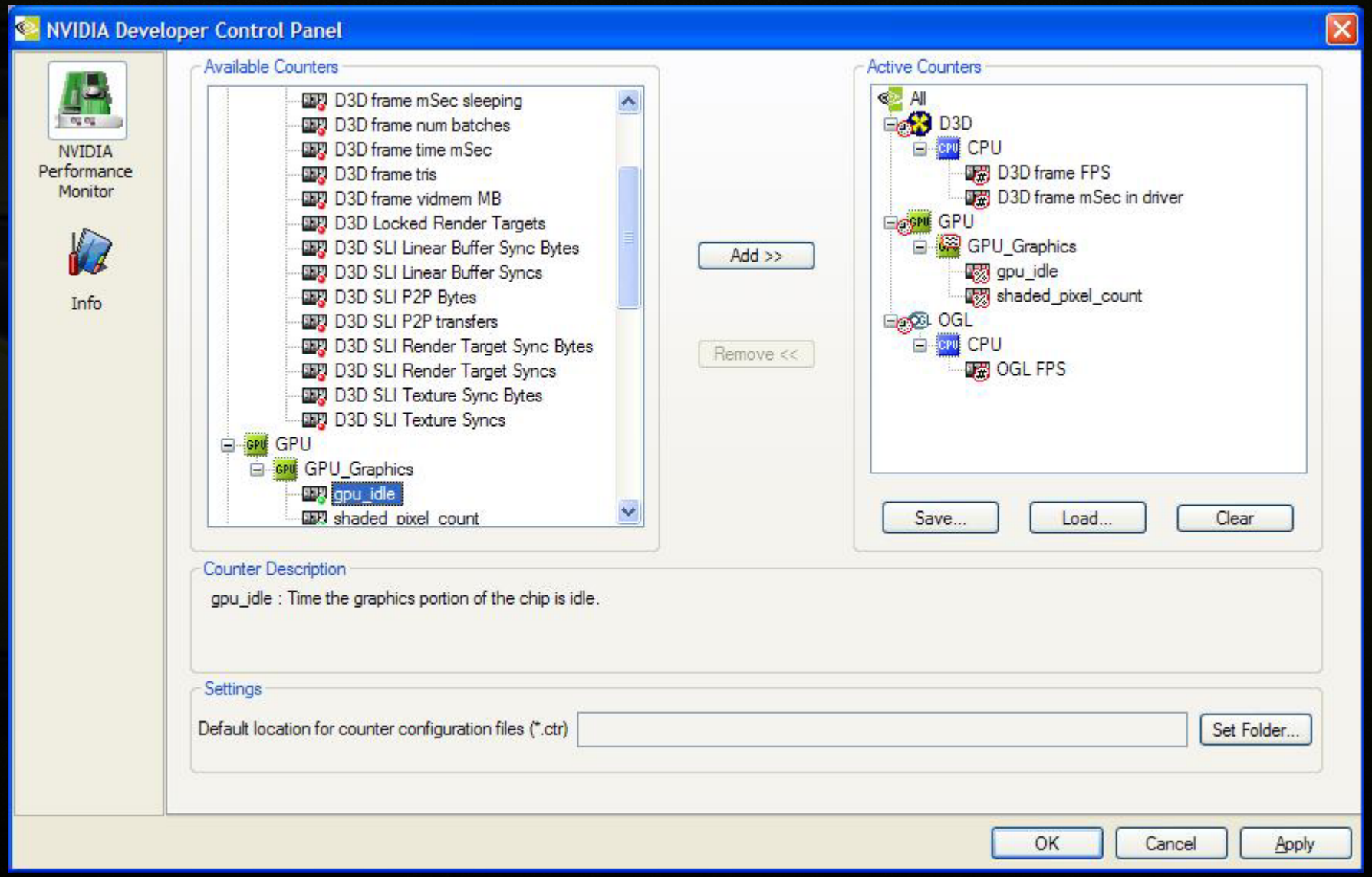
PDHとPerfmon

- PDH: Performance Data Helper for Windows
  - アプリケーションにパフォーマンス関連のデータを提供するためのWin32 API
  - 多くのアプリケーションで使用されている
- Perfmon: (またはMicrosoft Management Console)
  - PHDを使ったWin32アプリケーション
  - 低周波数のサンプリング (1X/s)
  - PDHのカウンタを表示:
    - OS: CPU使用量, memory使用量, スワップファイル使用量, ネットワーク情報など
    - NVIDIA: NVPerfKitの全てのカウンタ

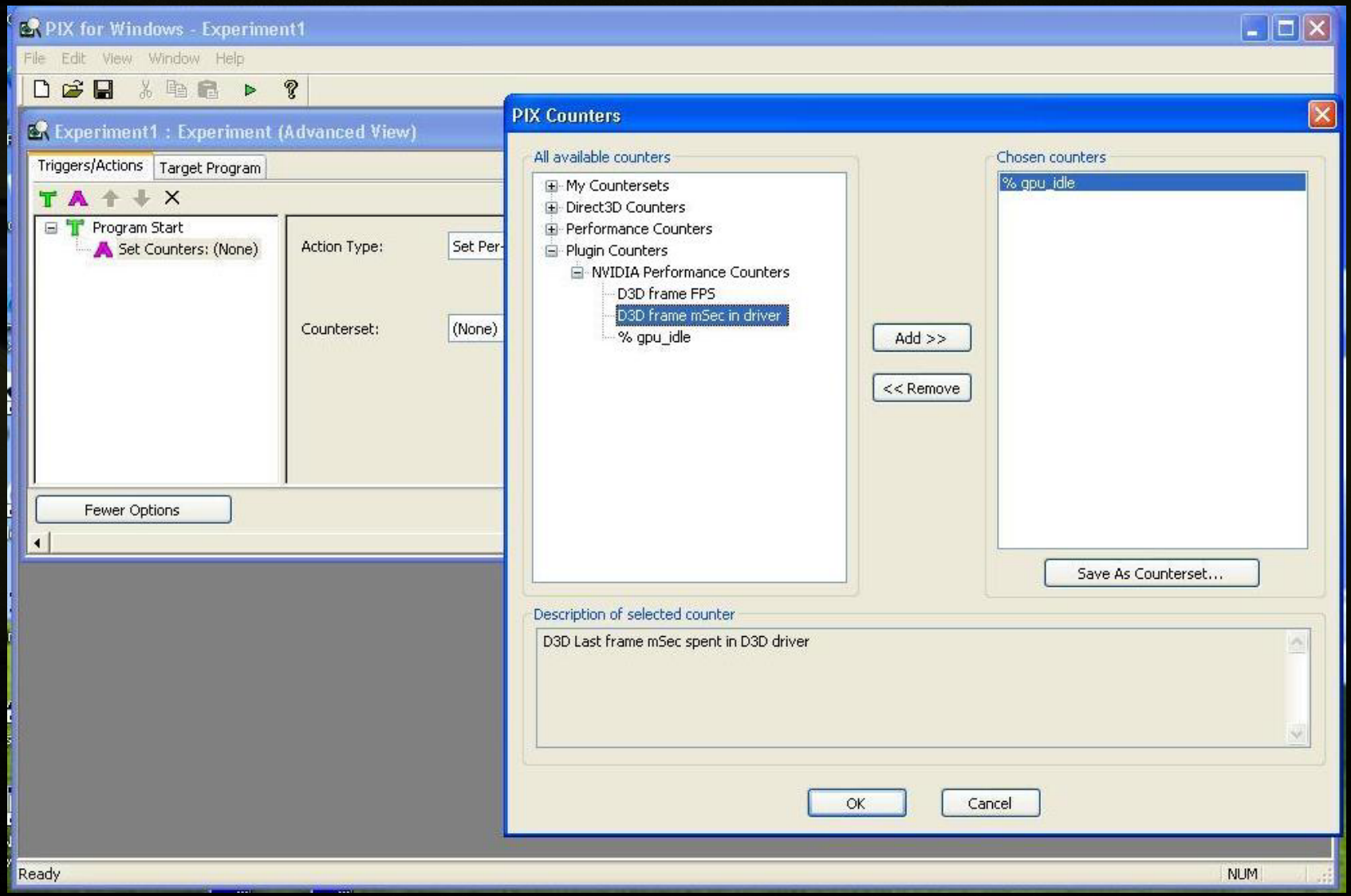
関連ツール: Perfmon



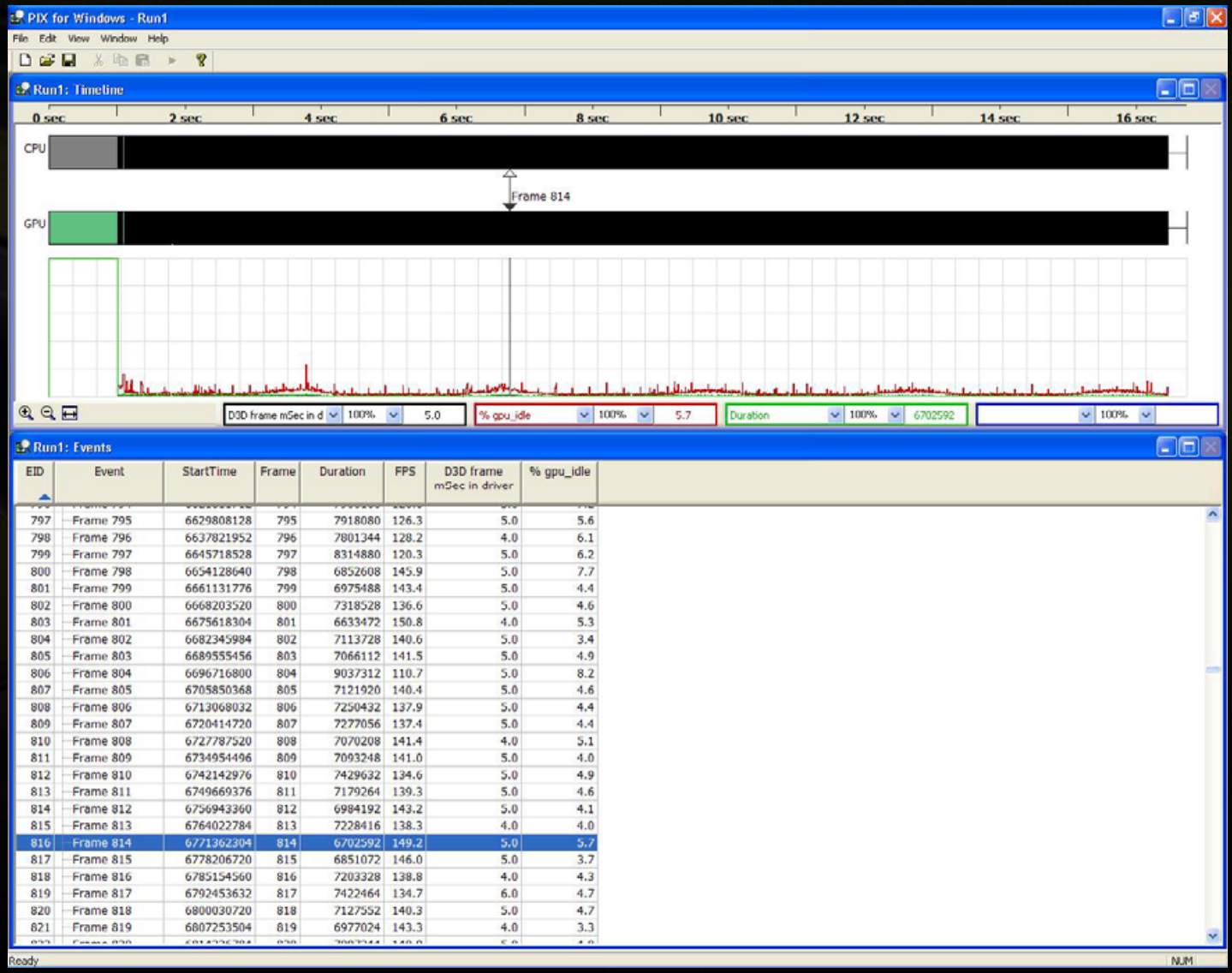
関連ツール: NVDevCPL



関連ツール: NVIDIA Plug-In for Microsoft PIX for Windows



関連ツール: NVIDIA Plug-In for Microsoft PIX for Windows



EID	Event	StartTime	Frame	Duration	FPS	D3D frame mSec in driver	% gpu_idle
797	Frame 795	6629808128	795	7918080	126.3	5.0	5.6
798	Frame 796	6637821952	796	7801344	128.2	4.0	6.1
799	Frame 797	6645718528	797	8314880	120.3	5.0	6.2
800	Frame 798	6654128640	798	6852608	145.9	5.0	7.7
801	Frame 799	6661131776	799	6975488	143.4	5.0	4.4
802	Frame 800	6668203520	800	7318528	136.6	5.0	4.6
803	Frame 801	6675618304	801	6633472	150.8	4.0	5.3
804	Frame 802	6682345984	802	7113728	140.6	5.0	3.4
805	Frame 803	6689555456	803	7066112	141.5	5.0	4.9
806	Frame 804	6696716800	804	9037312	110.7	5.0	8.2
807	Frame 805	6705850368	805	7121920	140.4	5.0	4.6
808	Frame 806	6713068032	806	7250432	137.9	5.0	4.4
809	Frame 807	6720414720	807	7277056	137.4	5.0	4.4
810	Frame 808	6727787520	808	7070208	141.4	4.0	5.1
811	Frame 809	6734954496	809	7093248	141.0	5.0	4.0
812	Frame 810	6742142976	810	7429632	134.6	5.0	4.9
813	Frame 811	6749669376	811	7179264	139.3	5.0	4.6
814	Frame 812	6756943360	812	6984192	143.2	5.0	4.1
815	Frame 813	6764022784	813	7228416	138.3	4.0	4.0
816	Frame 814	6771362304	814	6702592	149.2	5.0	5.7
817	Frame 815	6778206720	815	6851072	146.0	5.0	3.7
818	Frame 816	6785154560	816	7203328	138.8	4.0	4.3
819	Frame 817	6792453632	817	7422464	134.7	6.0	4.7
820	Frame 818	6800030720	818	7127552	140.3	5.0	4.7
821	Frame 819	6807253504	819	6977024	143.3	4.0	3.3

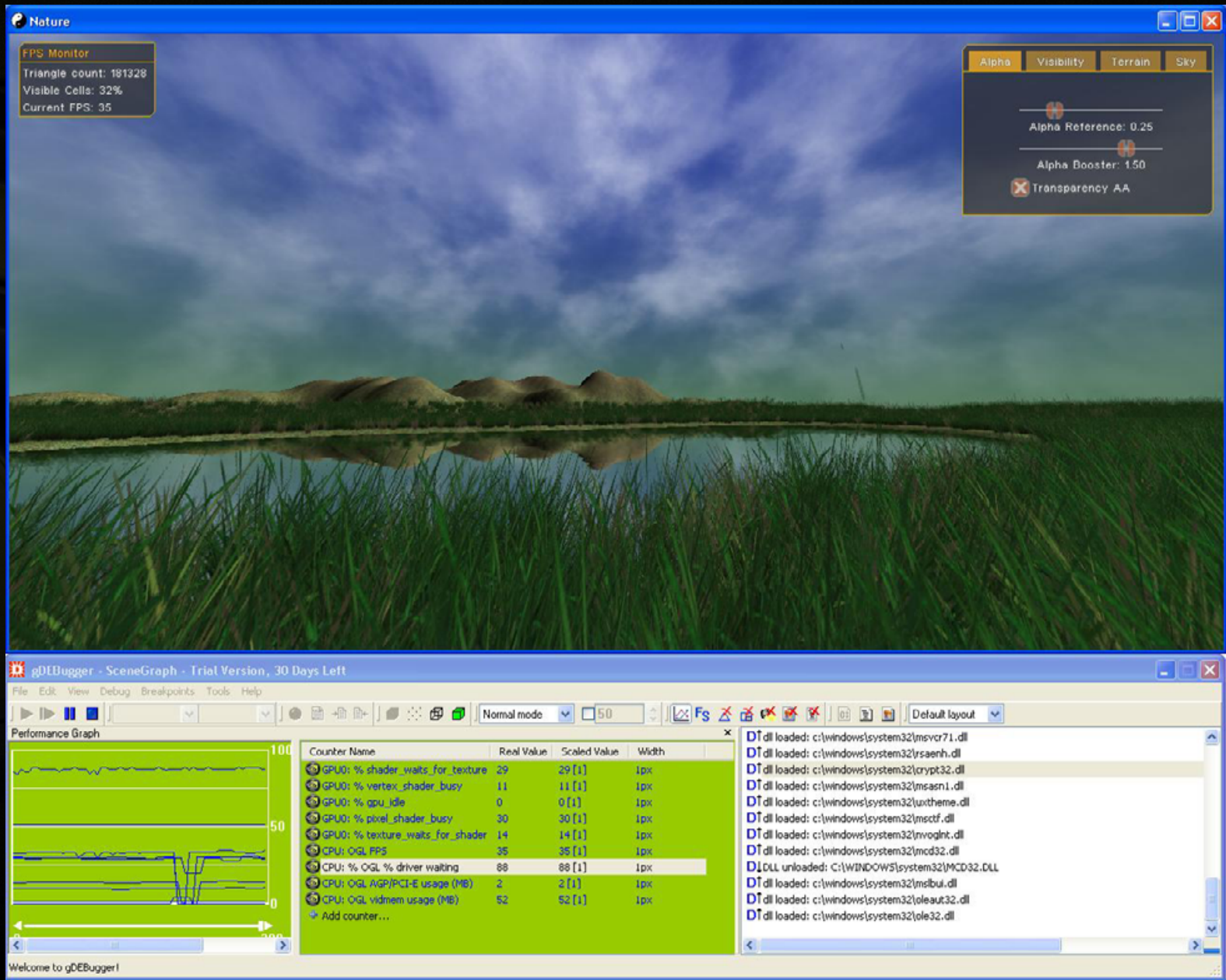
ヘルパークラスとコードサンプル

- **CPDHHelper: PDH**の使用を簡略化

```
int nIndex = pdh.add(“countername”);
pdh.sample();
float fValue = pdh.value(nIndex);
```

- **CTrace:** パフォーマンス情報を保持するためのバッファ
- **CTraceDisplay:** 簡単な**API**非依存の描画ライブラリ
- **OpenGL**と**Direct3D**でのサンプル
  - ヘルパークラスの使用方法
  - セキュリティ機構の使用方法

Graphic Remedyの gDEBugger 2.0



NVPerfKit 2.0

- 実験の簡略化
- ボトルネック発見のための解析
  - **PDH**による単一カウンタの補助
  - 複数の**GPU**カウンタが必要なマルチパスの実験
  - **NVPerfHUD 4.0**の全ての機能を提供
- より多くの**OpenGL**と**Direct3D**カウンタ
- **NVPerfHUD 4.0**
- **Linux**サポート

NVIDIA®

NVPerfHUD 4.0

Raul Aguaviva

NVPerfHUD

- 概要
- 仕組み
- デモ
- スケジュール

概要

- **PERFormance Heads Up Display**の略
  - グラフやダイアログをアプリケーションの上に表示
  - 対話式**HUD**

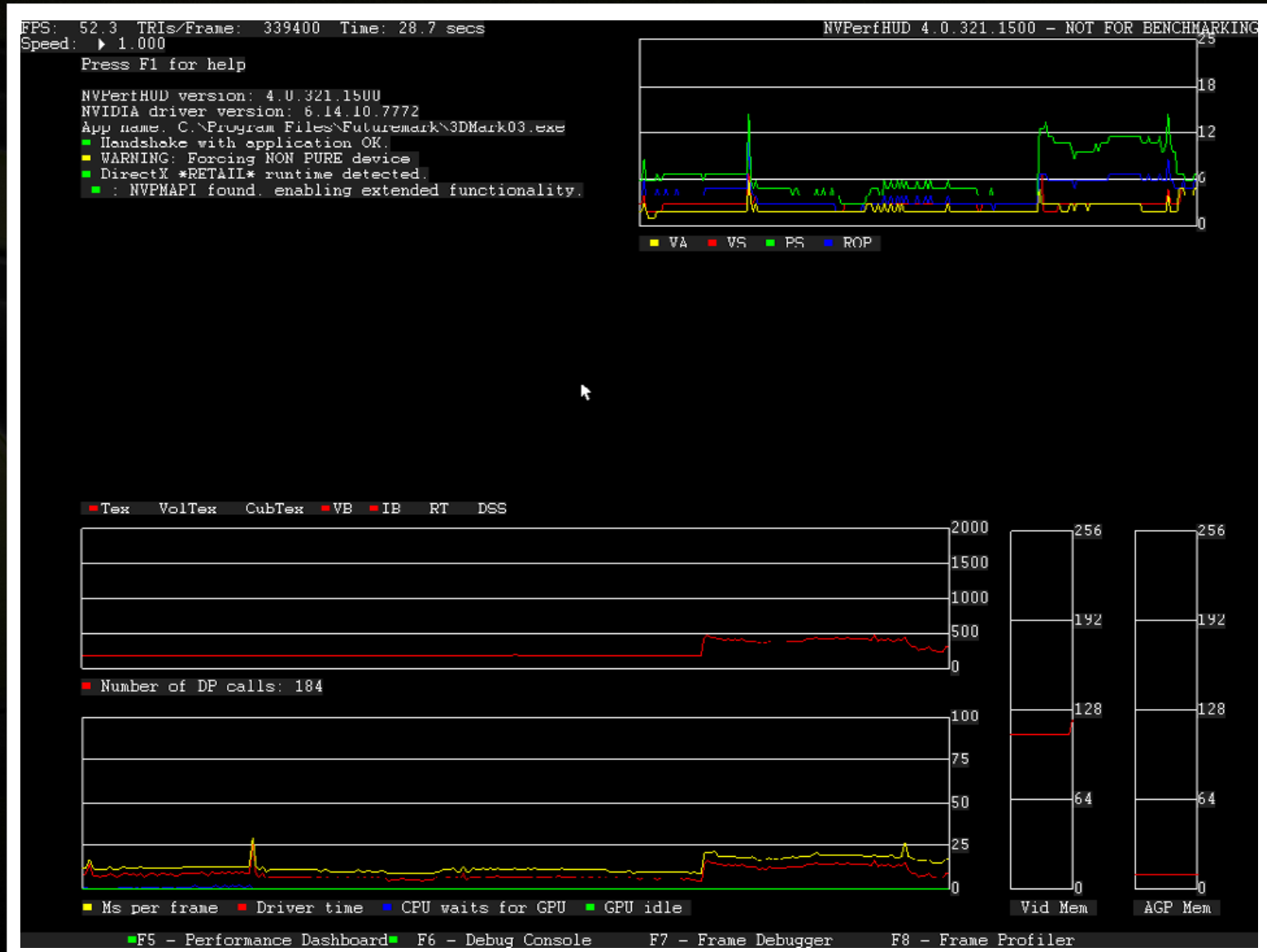
概要

- 4種類のHUD
  - パフォーマンス・ダッシュボード
  - デバッグ・コンソール
  - フレーム・デバッガ
  - フレーム・プロファイラ (4.0での新機能)

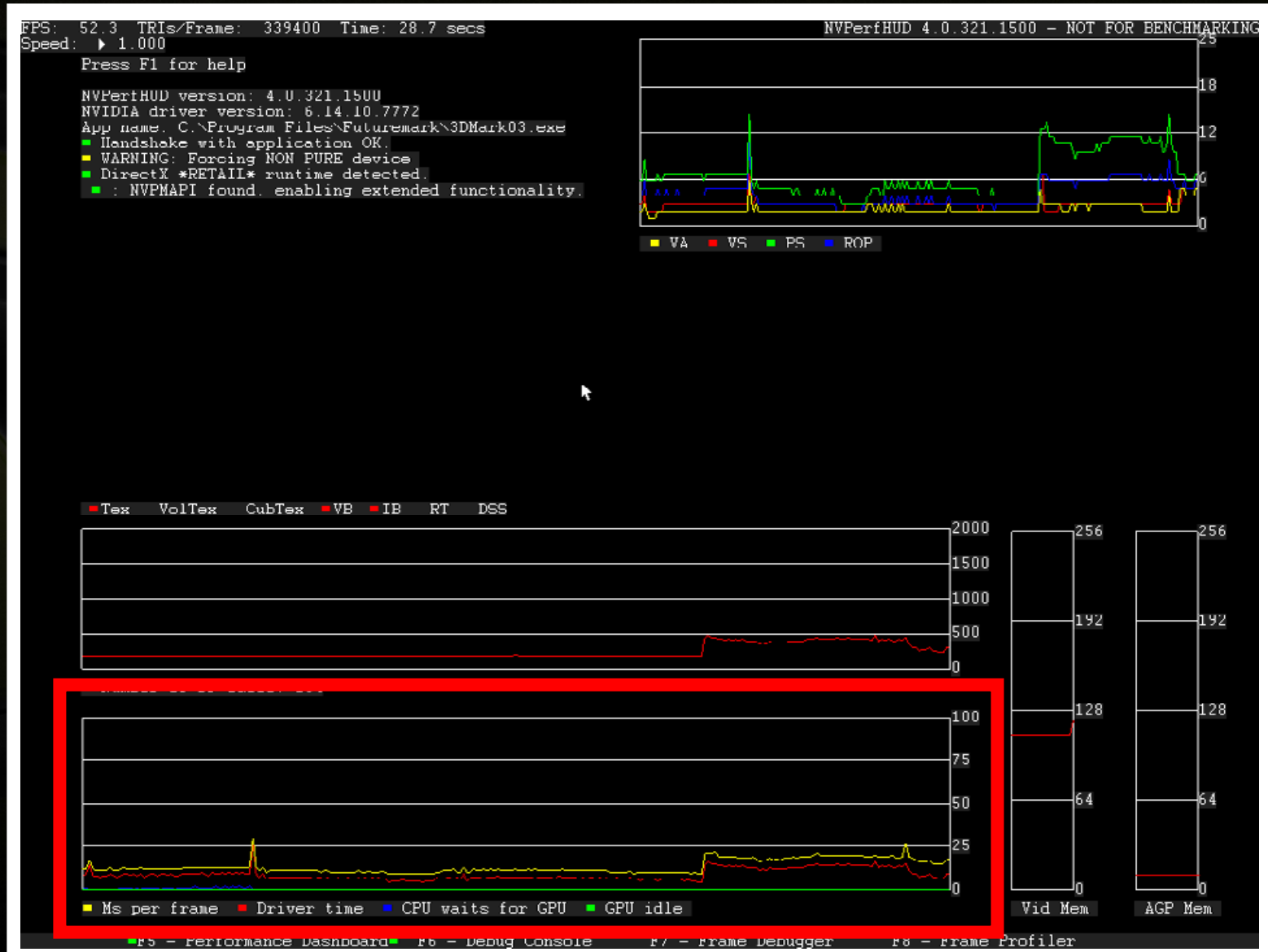
使い方

- **NVPerfHUD**と一緒にアプリケーションを起動
- 通常通り使用:
  - 機能の問題 - デバッガを使用
  - パフォーマンス問題 - プロファイラを使用

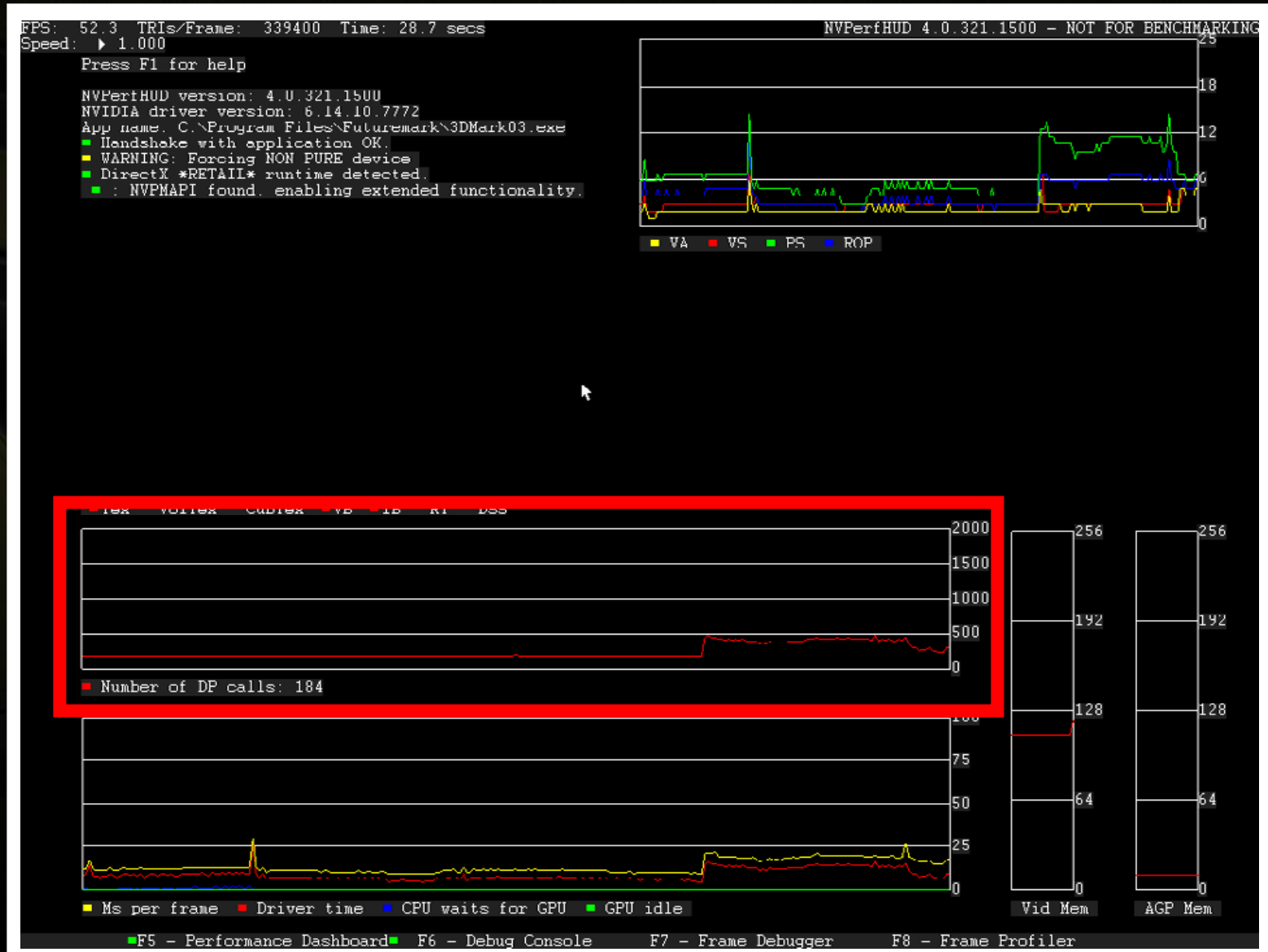
パフォーマンス・ダッシュボード



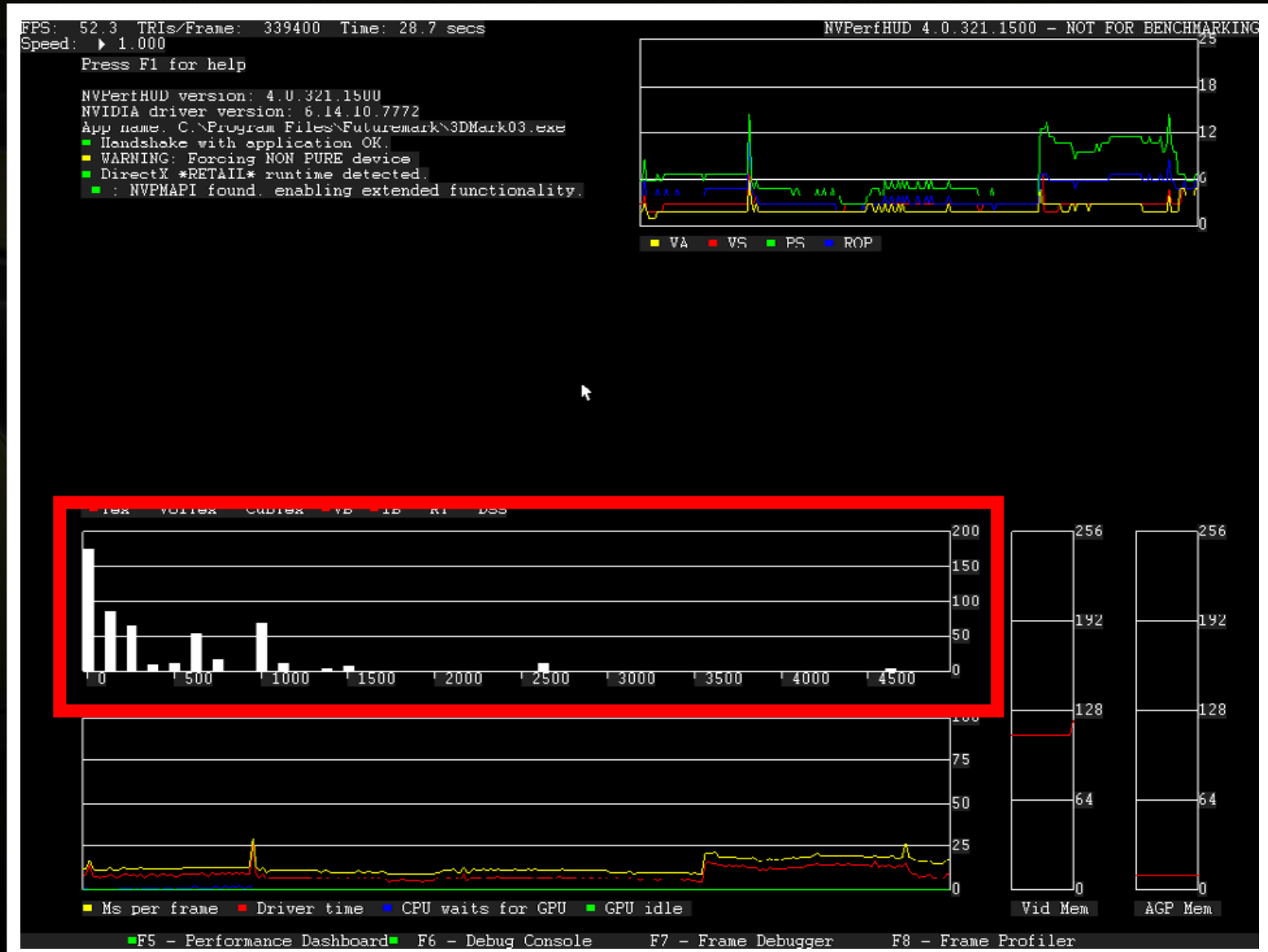
パフォーマンス・ダッシュボード



パフォーマンス・ダッシュボード



パフォーマンス・ダッシュボード

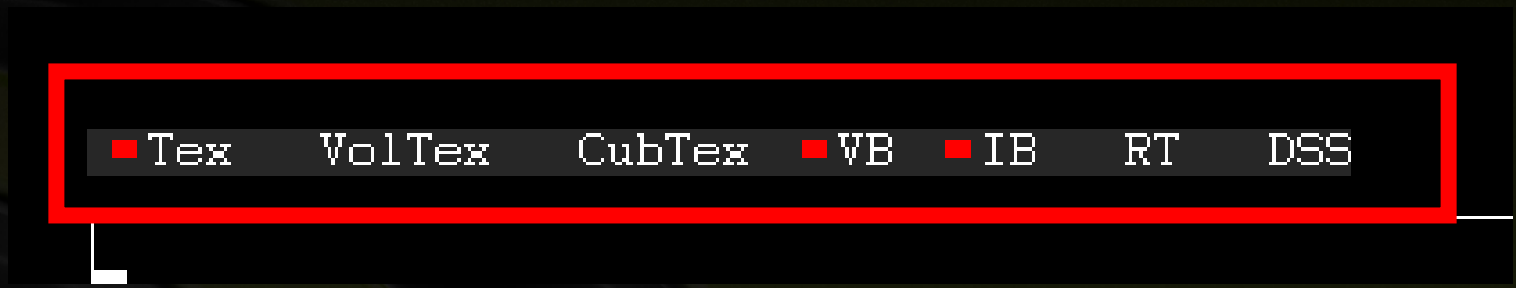


パフォーマンス・ダッシュボード

パフォーマンス・ダッシュボード

パフォーマンス・ダッシュボード

- リソース監視

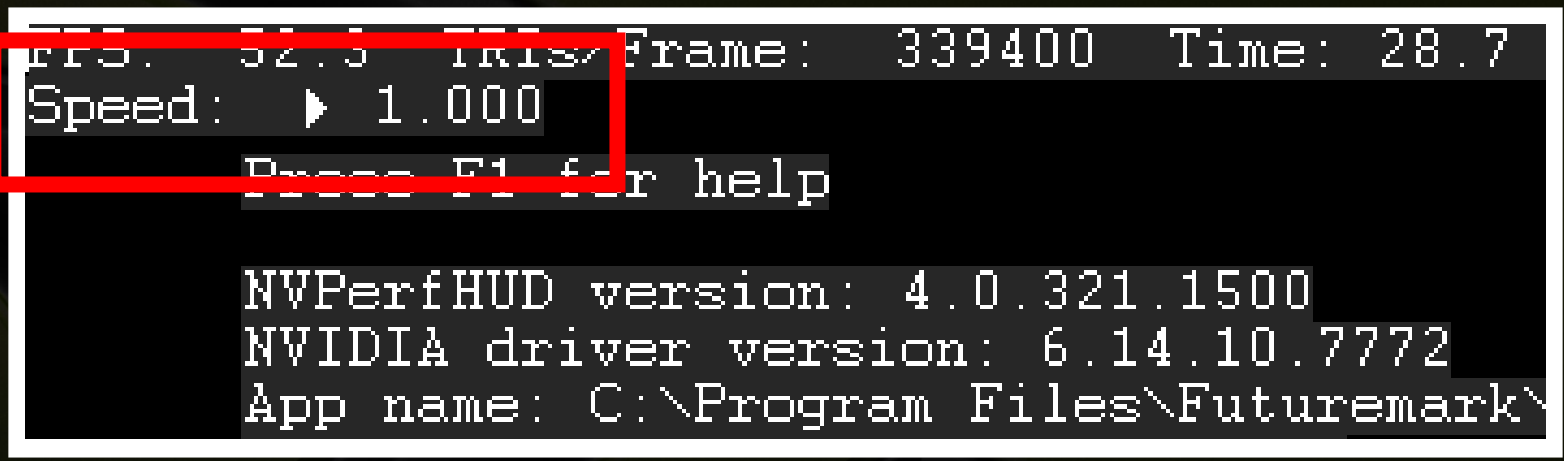


- 監視されるリソース
  - テクスチャ
  - ボリューム・テクスチャ
  - キューブ・テクスチャ
  - 頂点バッファ
  - インデックス・バッファ
  - ステンシルと深度バッファ

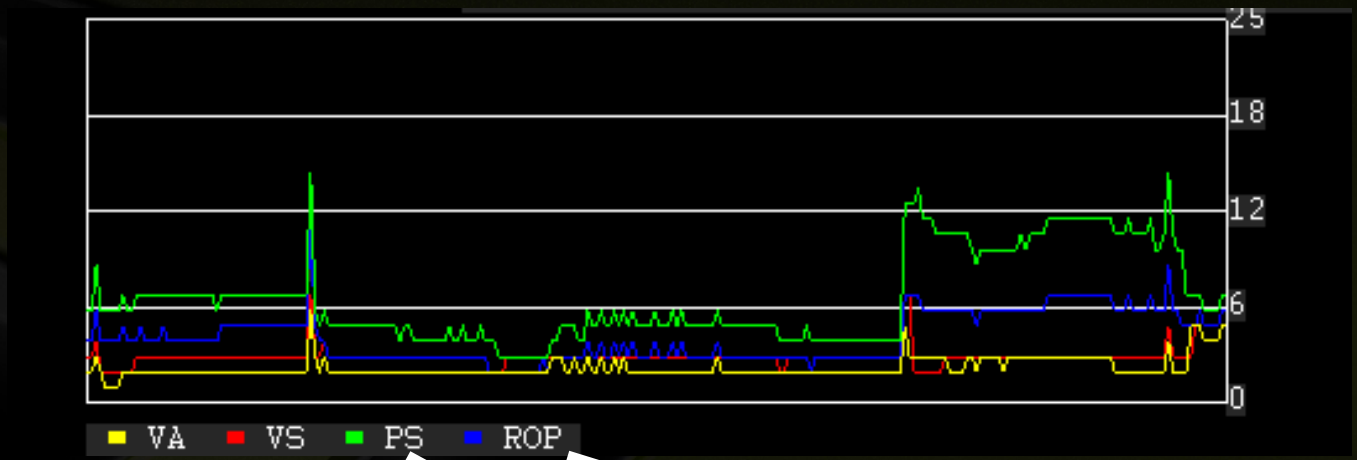
パフォーマンス・ダッシュボード

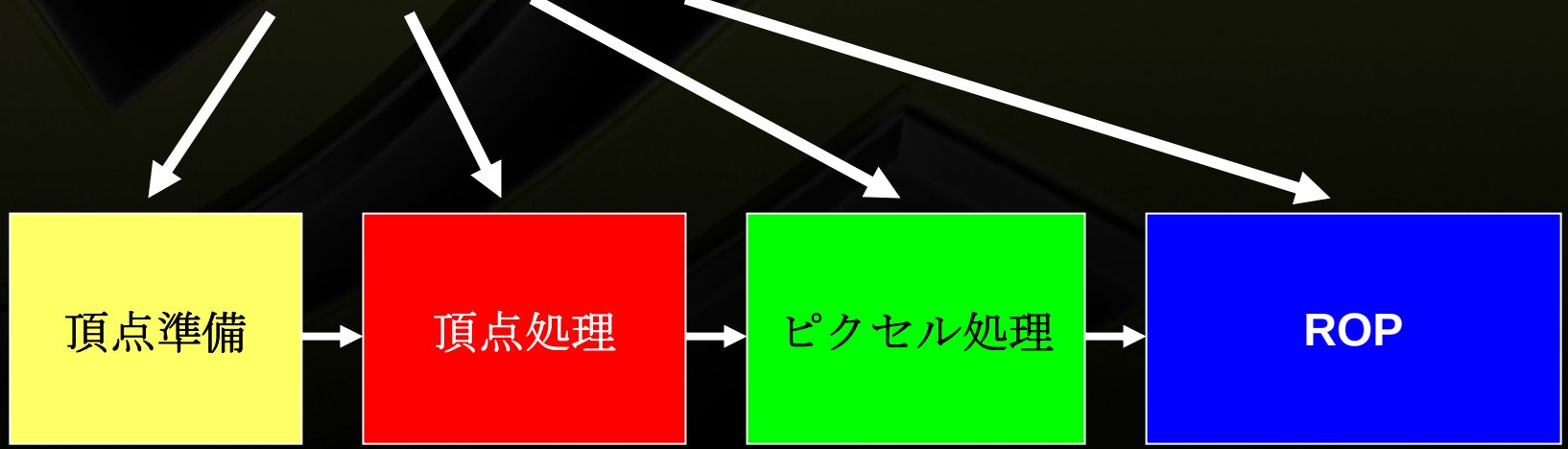
パフォーマンス・ダッシュボード

- 速度制御

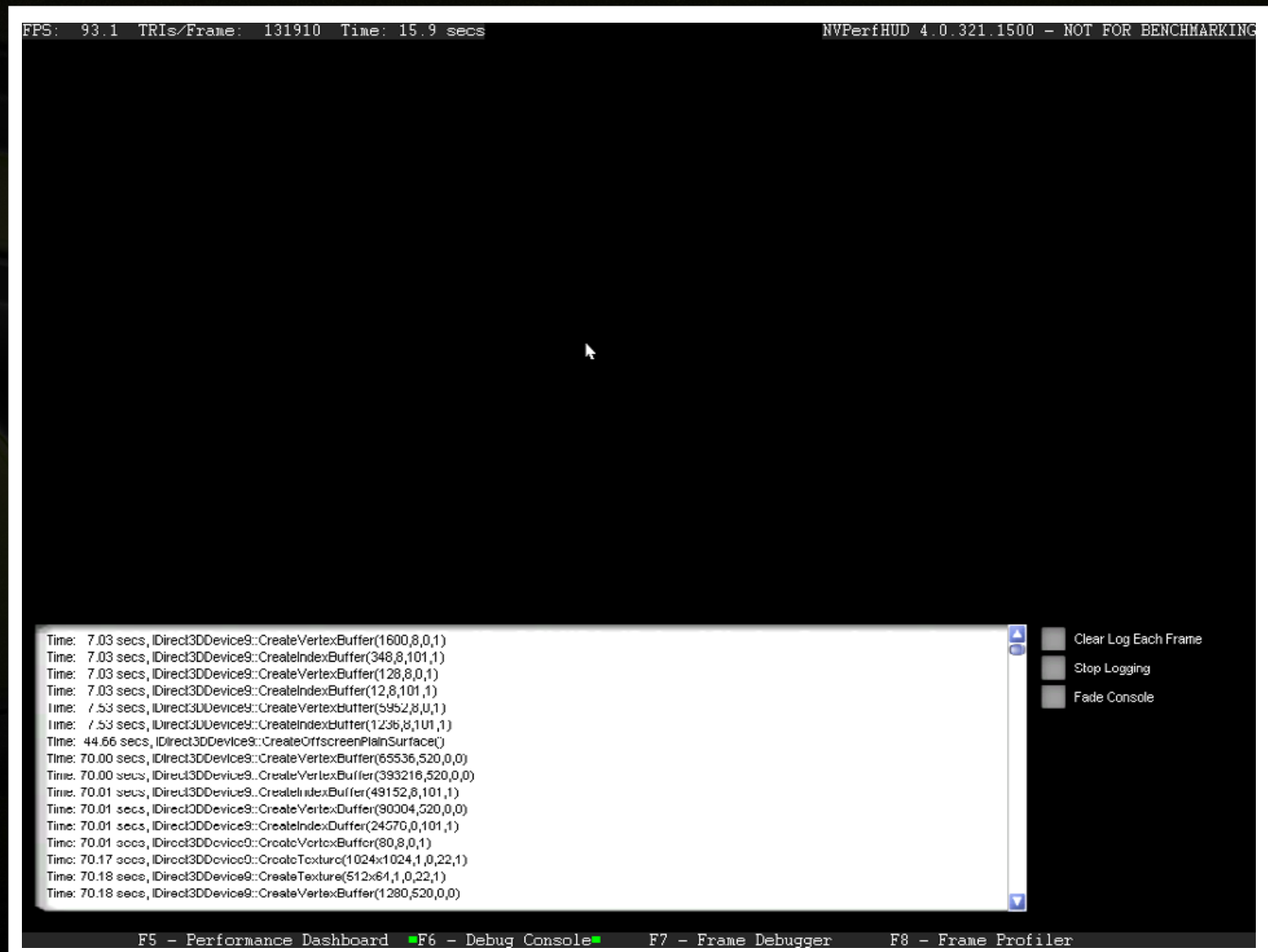


簡略化されたパイプライン





デバッグ・コンソール


FPS: 93.1 TRIs/Frame: 131910 Time: 15.9 secs
NVPerfHUD 4.0.321.1500 - NOT FOR BENCHMARKING
Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(1600,8,0,1)
Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(348,8,101,1)
Time: 7.03 secs, IDirect3DDevice9::CreateVertexBuffer(128,8,0,1)
Time: 7.03 secs, IDirect3DDevice9::CreateIndexBuffer(12,8,101,1)
Time: 7.53 secs, IDirect3DDevice9::CreateVertexBuffer(59528,8,0,1)
Time: 7.53 secs, IDirect3DDevice9::CreateIndexBuffer(1236,8,101,1)
Time: 44.66 secs, IDirect3DDevice9::CreateOffscreenPlainSurface()
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(65536,520,0,0)
Time: 70.00 secs, IDirect3DDevice9::CreateVertexBuffer(393216,520,0,0)
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(49152,8,101,1)
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(90004,520,0,0)
Time: 70.01 secs, IDirect3DDevice9::CreateIndexBuffer(24576,0,101,1)
Time: 70.01 secs, IDirect3DDevice9::CreateVertexBuffer(80,8,0,1)
Time: 70.17 secs, IDirect3DDevice9::CreateTexture(1024x1024,1,0,22,1)
Time: 70.18 secs, IDirect3DDevice9::CreateTexture(512x64,1,0,22,1)
Time: 70.18 secs, IDirect3DDevice9::CreateVertexBuffer(1280,520,0,0)
Clear Log Each Frame
Stop Logging
Fade Console
F5 - Performance Dashboard
F6 - Debug Console
F7 - Frame Debugger
F8 - Frame Profiler

フレーム・プロファイラ

- パフォーマンスのカウンタを測定
- 解析

フレーム・プロファイラでの測定

- **NVPerfHUDはNVPerfKitを使用する**
  - 約**40**のカウンタを使用
- 同時に全てを読むことはできない
- 全てのカウンタを読むために、同じフレームを何回も描画しなければならない

フレーム・プロファイラでの解析

- 最適化のための解析:
  - ステートごとにまとめる – ボトルネックごとのグループ
  - これらのグループは『**state buckets**』と呼ばれる
- 手順
  - ステートにより、描画コールを分類
  - 最も時間のかかっているグループのボトルネックを見つける
    - **NVPerfHUD**による解析
  - ボトルネックを解消

アプリケーションの一時停止

- アプリケーションが実時間ベースのアニメーションをしている場合のみに可能
- タイマーを停止
  - **QueryPerformanceCounter()、timeGetTime()**を使用
  - **RDTSC**は使用しない
- **Pos += V * DeltaTime**

予定

- ベータ: 8月
- リリース: 9月

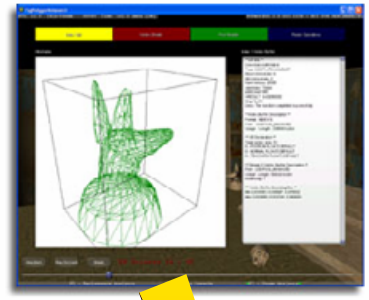
NVPerfHUD 3 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://developer.nvidia.com/object/nvperfhud_home.html
Firefox Help Firefox Support Plug-in FAQ Sign In - Yahoo! Uber Gizmo, the Ga... Microsoft Visual C++... DirectX Amazon.com: All Pr... sapere.it
developer.nvidia.com
THE SOURCE FOR GPU PROGRAMMING
Search
www developer.nvidia.com
REGISTERED DEVELOPERS
JOIN
HOME
NEWS
DOCUMENTATION
TOOLS & SDKs
PARTNERS
NEWSLETTER SIGN-UP
EVENTS CALENDAR
DRIVERS
CONTACT
Legal Info

NVPerfHUD 3

Quick Links

- Introductory Video
- Downloads
- GameDev.net Review

Overview



Modern GPUs generate images through a pipelined sequence of operations. A pipeline runs only as fast as its slowest stage, so tuning graphical applications for optimal performance requires a pipeline-based approach to performance analysis. NVPerfHUD analyzes your graphics pipeline performance in real time statistics you can use to diagnose performance bottlenecks in your D3D application.

The latest release includes several new analysis modes:

- **Frame Analysis Mode**
  Freeze your application and single-step through the current frame, allowing you to see what is happening inside the GPU at each draw call in your graphics pipeline units, including the Input Assembler, Vertex Unit, Vertex Shader, Pixel Shader & Raster Operations units.
- **Debug Console Mode**
  This mode shows you DirectX Debug Runtime warnings, errors, and custom messages from your application.
- **Performance**
  The powerful performance analysis experiments from the previous release are still available in Pe

Now Available!

The opt-in mechanism and code alterations are necessary if you have already enabled NVPerfHUD 2.0 in your

Be sure to check out the Getting Started instructions in the **NVPerfHUD User Guide**, and read through the methodology for effectively identifying and crushing performance bottlenecks in your application. We've also created a **Quick Reference Card** with tips and shortcuts that you can keep at your fingertips. Both these documents are available in English, Japanese, Chinese, and Korean.

See our "**NVIDIA Performance Analysis Tools**" talk from **GDC 2005** for more information on how to analyze your applications using NVPerfHUD. In addition, our "**Practical Performance Analysis and Tuning**" talk from **GDC 2004** explains the theory of pipeline analysis and bottleneck removal.

Downloads

Developer Reviews

NVPerfHUD has way more features than I expected! Blew me away!!
- Remy Saville
Graphics Programmer
Relic Entertainment, Inc.

[Frame Analysis Mode] is by far the most impressive mode of NVPerfHUD. ... A few moments in this mode does more to show how advanced graphic pipelines work than anything else I have seen. ... NVPerfHUD should be of interest to anybody who develops software that employs DirectX graphics. Even if you don't have an NVIDIA card it's worth the price of a GeForce 6600 or 6800 to get this tool.
- Bryan Mau
GameDev.net Review

NVPerfHUD 3 - it's simply amazing. I use it virtually every day.
- Chris King, President
IDV, Inc.

NVPerfHUD is a great tool for debugging and performance analysis.
- Richard Schubert
Graphics/Effects Programmer
Yager Development GmbH

I just wanted to drop a note to thank you for the hard work on the NVPerfHUD. This new version is incredibly useful. Not a day goes by that I don't appreciate the support NVIDIA gives developers with tools such as this.
- Matt Shaw
Director of Technology
Mythic Entertainment

Done
Start 2 Win... Oxford ... 4 Fire... 4 Micr... Total C... nvPerf... unar^M... Microso... 5:59 PM

質問

- サイト: http://developer.nvidia.com

NVGLExpert@nvidia.com
NVShaderPerf@nvidia.com
NVPerfKIT@nvidia.com
NVPerfHUD@nvidia.com
FXComposer@nvidia.com

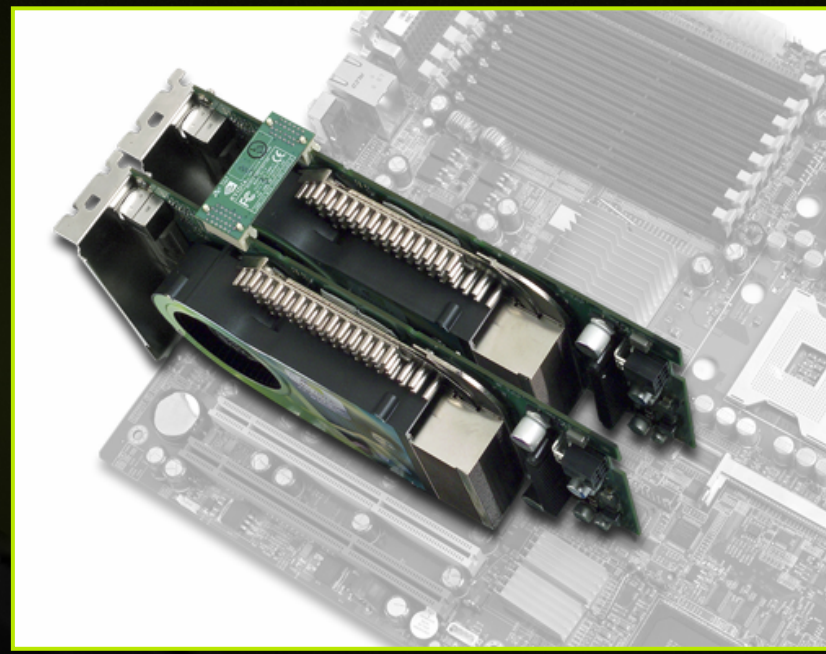
NVIDIA®

SLI

Matthias M Wloka
NVIDIA Corporation

SLI: Scalable Link Interface

- 同型のふたつのGPUをPCI-Eマザーに挿して使用
- ドライバーはひとつのGPUとして認識
  - 1.9x程度までのスピードアップ
- ビデオメモリは倍にはならない



最上級のニッチ市場だけではない

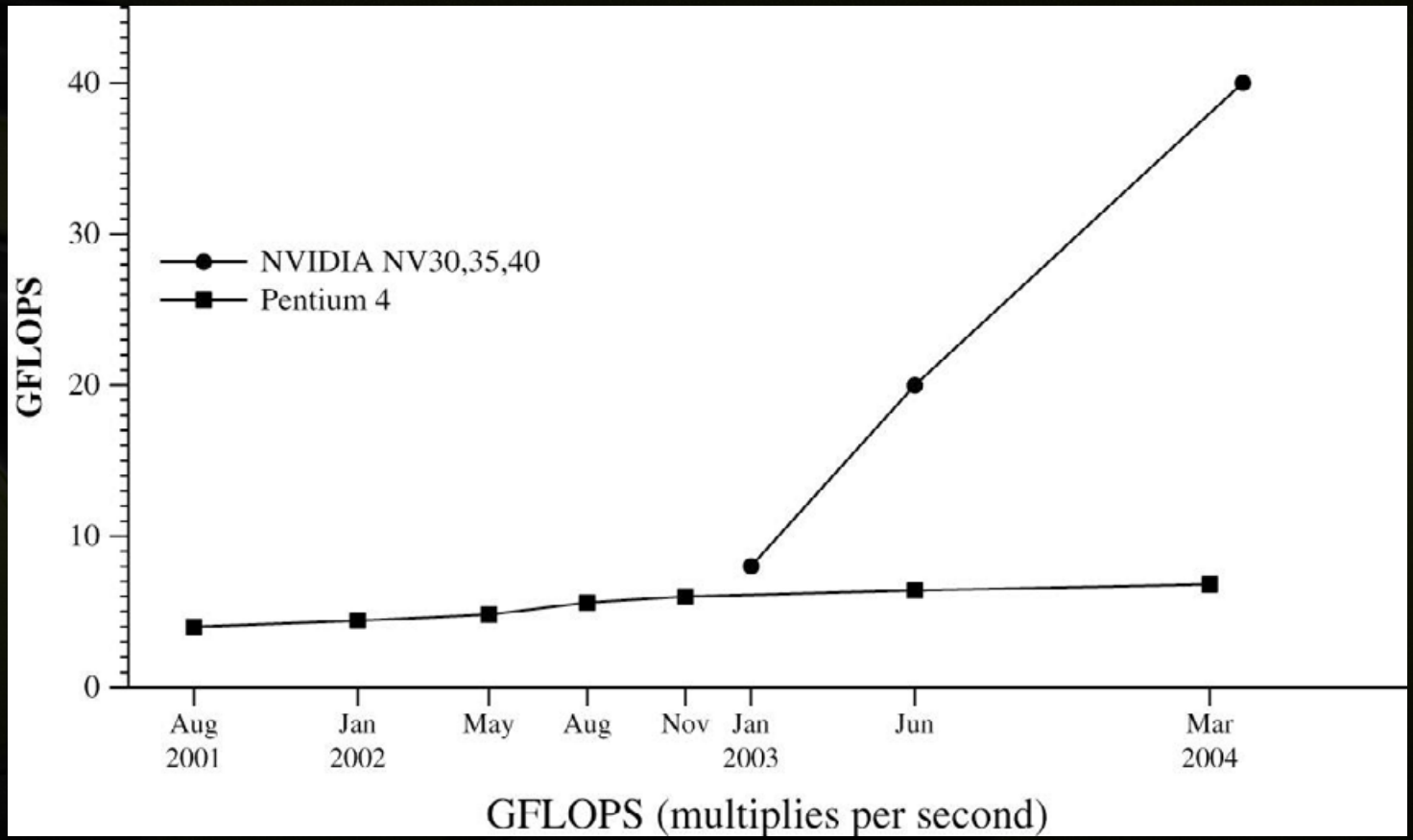
- 主流製品での**SLI:**
  - **GeForce 6600 GT SLI**
  - もちろん**6800 GT**と**6800 Ultra**
- デュアル・コアのボード
  - **Gigabyte 3D1:**
    Dual 6600 GT

- **SLI**マザー
  現在**(2005**年３月**)**までの販売数**: 350,000**以上
  - これは**nForce4**販売数の**25%**以上

ゲーム開発サイクル

- ２年(かそれ以上)で
  - CPUパフォーマンスが倍(もしくはそれ以下)
  - GPUパフォーマンスが４倍
- CPU/GPUのバランスが変わる!
  - さらに悪化: CPUが必要なモジュールが後で足される: AI、物理演算、ゲームプレイ
- SLIで将来のGPU対CPUバランスがうかがえる
  - 目標とする『主流』性能に対して

過去数年



Courtesy Ian Buck, Stanford University

SLIの仕組み

- 互換モード:
  - ひとつのGPUのみ使用
  - SLIの効果なし
- 交互フレーム描画 - Alternate frame rendering (AFR)
- 分割フレーム描画 - Split frame rendering (SFR)

AFR

- 各GPUはそれぞれのフレームを描画

- スキャン・アウトは読み出すフレームバッファを交互する

AFRでの一般的な描画

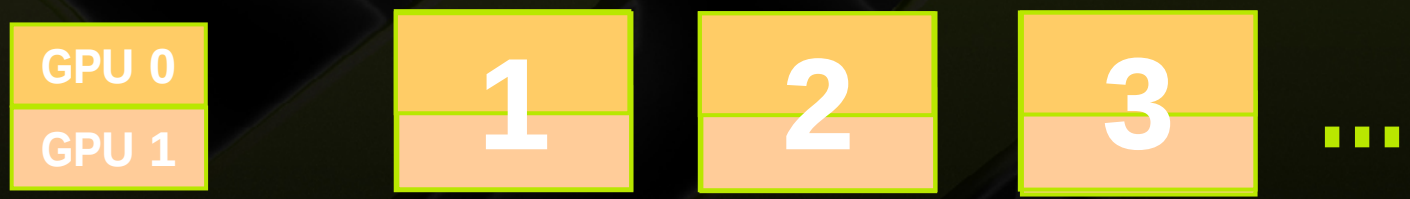
- フレームが独立していない:
  - 必要なデータをもうひとつのGPUへ送る
  - 例: テクスチャへの描画をフレームひとつおきに行う
- データのGPU同士の移動は余分な負荷
  - そのため2倍のスピードは出ない

AFRの利点

- すべて並列処理
  - ピクセルのフィル、ピクセル化、頂点変形
- できればこちらを使用したい
- フレームが独立していると良い
  - 前のフレームのデータを使用しない
  - GPU間の通信による負荷を低減

SFR

- 両方のGPUが同一のフレームを処理
  - GPU 0が画面上
  - GPU 1が画面下



- スキャン・アウトはフレームバッファを合成

SFRでの一般的な描画

- 『上』と『下』での負荷調整
  - ひとつのGPUが時間をかけ過ぎた場合
  - 負荷を調整
- 頂点も上下にクリップ
  - 両方のGPUで全ての頂点を処理することを防ぐ
  - 必ずうまくいくとは限らない
- これでもいくらかのデータ共有は必要:
  - 例: テクスチャ描画

AFRと比較したSFR

- SFRはためておくフレーム数を制限しても有効
  - 言い換えれば、AFRではうまくいかないこともある
- 一般的にSFRの方が通信負荷が大きい
- 頂点処理が多い場合、SFRの価値は少ない

SLIシステムを検出する

- NVCpl API:
  - 全てのNVIDIAドライバをサポートするNVIDIA特有API
- 機能:
  - NVCpl APIが使用可能かの検出
  - バスのモード(PCI/AGP/PCI-E)と速度(1x-8x)
  - ビデオRAMの大きさ
  - SLI

NVCpl APIのSLI検出

- SDKサンプルとドキュメント

```
HINSTANCE hLib = ::LoadLibrary("NVCPL.dll");

NvCplGetDataIntType NvCplGetDataInt;
NvCplGetDataInt =
   (NvCplGetDataIntType)::GetProcAddress(hLib,
                    "NvCplGetDataInt");

long    numSLIGPUs  = 0L;
NvCplGetDataInt(NVCPL_API_NUMBER_OF_SLI_GPUS,
                           &numSLIGPUs);
```

ゲームからSLIを強制的に有効にする

- NVCplを使用
  - NvCplSetDataInt()で
    AFR、SFR、互換モードに設定
  - SDKサンプル参照
- プロファイルを変更、もしくは新しく作成:
  - http://nzone.com/object/nzone_sli_appprofile.html
  - ユーザがプロファイルを作成することもできる

概要: SLIが有効ではないケース

- CPU制限
  - またはvsyncが有効になっている
- バッファするフレーム数
- 通信負荷

CPU制限

- SLIではどうしようもない
- CPU負荷を削減するか、もしくは:
- CPU負荷をGPUへ移す
  - http://GPGPU.org参照
- フレーム・レートを制御しない

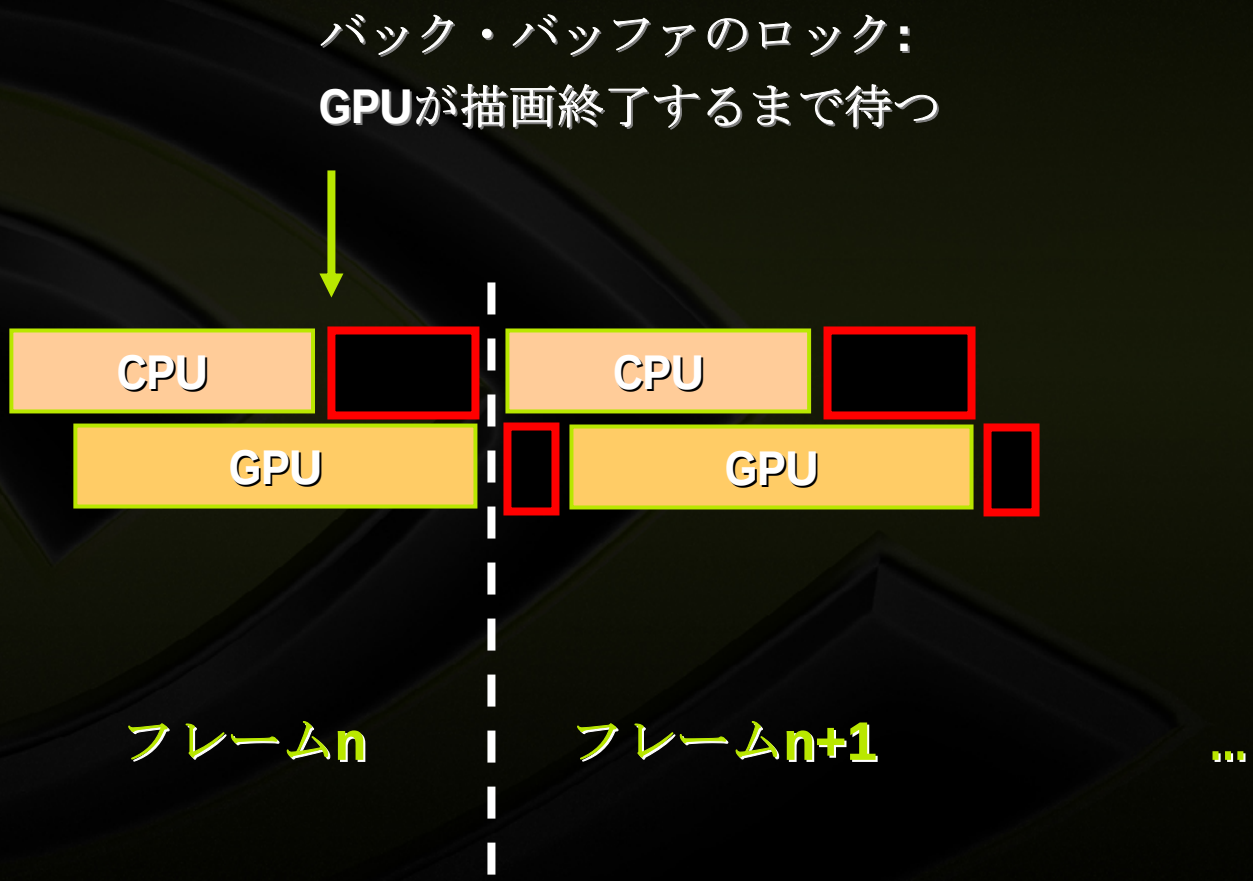
VSync有効

- モニタのリフレッシュにフレーム・レートを固定
- トリプル・バッファでは**vsync**の解決にはならない:
  - 描画速度がモニターのリフレッシュより速い場合、
  - **vsync**はやはり**GPU**を制限する
- さらに問題な、トリプル・バッファ
  - 時間差を広げる
  - 非常に多くのビデオ・メモリを使用

バッファされるフレームを制限

- 最大ひとつのフレームしかバッファしないゲームもある
  - 時間差を制限するため
  - イベント・クエリを使用
  - バック・バッファのロック/読み出しはCPUを制限する!
- AFR SLIの高速化を制限
- しかしSLIは最大約1.9倍の速度
  - 例: SLIシステムが約1.9倍時間差が少ない場合

バック・バッファのロックが問題な理由

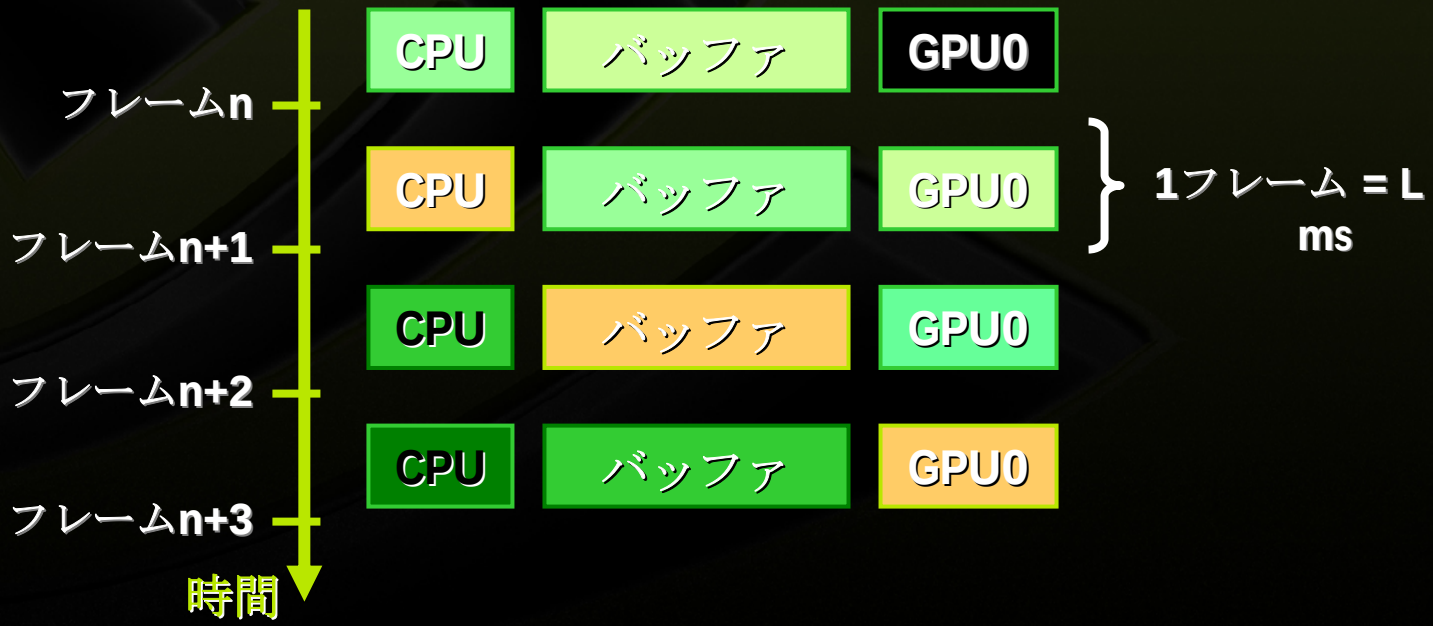


バッファするフレーム数を、GPU数に制限

- 単一GPUのシステム:
  最大１フレームのみバッファ
- SLIシステムの場合:
  最大２フレームのバッファ

パイプライン

フレームは時間に沿ってパイプラインを流れる:



単一GPUでの待ち時間

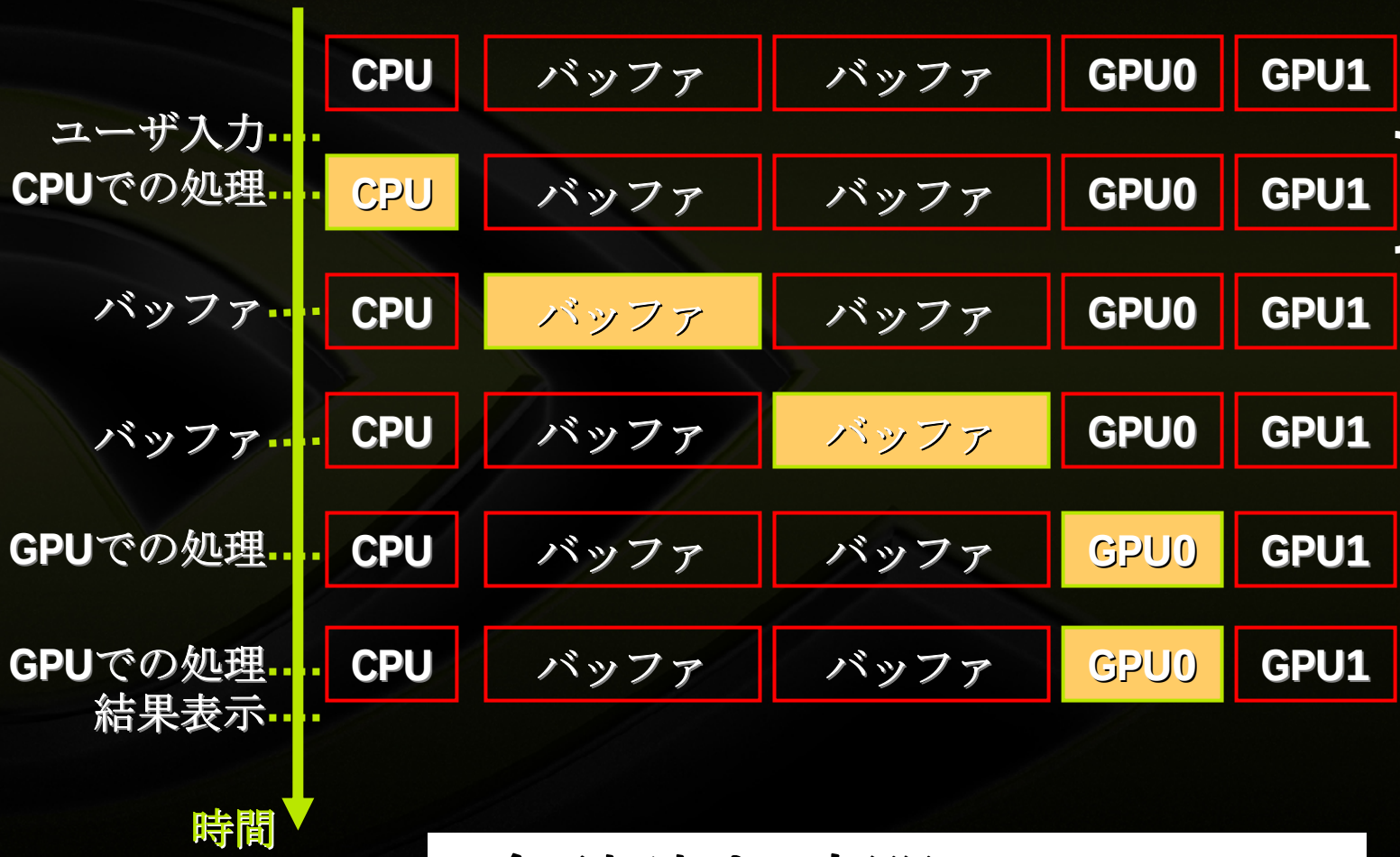
合計待ち時間: 3L ms

待ち時間の仮定

- GPU制限
  - そうでなければ、バッファには１フレーム以下たまっている
  - バッファを制限する意味はない
- SLIで２倍速
  - あとでこれをゆるめる!
- バッファするフレームをふたつに:

AFR SLIを流れるフレーム

AFR SLI待ち時間



合計待ち時間: 5·L/2 ms

待ち時間比較: 単一対AFR

- 単一GPU待ち時間: 3L ms
  - L時間が3フレーム分
- AFR SLI GPU待ち時間: 5 L/2 = 2.5L ms!
  - L/2時間が5フレーム分
    (フレーム・レートが倍)
  - 倍のフレームをバッファしているにもかかわらず
- SLIの高速化が1.66で良い!
  - 3L = 5L/x → x = 5L/3L = 1.66
  - ほとんどのゲームでは約1.8倍の高速化

SFRの待ち時間?

- SFRは１フレームのバッファでも問題ない
- SFRでの高速化はそのまま待ち時間も減らす
  - もしSFRで２倍速なら
  - 待ち時間は半分に

さらに: FPS依存で時間差を制御

- ゲームが**100fps**超える場合、
  - ３フレームのバッファは問題ないだろう
- ゲームが**15**以下の場合
  - １フレームだけバッファする
- 高速な**SLI**システムでは自動的に効果が上がる
- ドライバーはすでにこれを実装
  - **15fps**以上で３フレームをバッファ
  - **15fps**未満でバッファするフレーム数を減らす

概要: SLIが有効ではないケース

- CPU制限
  - またはvsyncが有効になっている
- バッファするフレーム数
- 通信負荷

通信負荷

- SLIメモリ転送
  - 転送そのものに帯域幅と時間がかかる
  - 転送を待つためにGPUが制限
- 作業を両方のGPUで行う
  - テクスチャ描画など
- 関連リソース:
  - 頂点/インデックス・バッファ
  - テクスチャ
  - 描画領域

実行時にリソースをアップロード

- ビデオRAMは複製されている
- 両方のビデオRAMへの転送が必要
- 開発者にできることはほぼない

描画領域

- Zクリア
  - 常にZはクリア!
- SLI検出の際には色もクリア
  - ドライバーに前フレームのデータは必要ないと明示
  - 古いデータをGPU間で転送する必要がなくなる
- フレーム間でデータの再利用をしない
  - フレームを独立させる

更新省略『最適化』

- 余分なSLI負荷:

- GPU 1は GPU 0のRTT終了と転送を待つ
- もしくはGPU 1はRTT作業をする
- いっそSLI上では正しい動作をさせたほうが良いだろう

速めに描画、遅めに使用!

- 同期待ちを回避
  - **AFR SLI**で上のように
  - 単一**GPU**モードでも
  - 更新負荷は存在

非常に問題: 速めに使用、遅めに描画

もしくは: SLIの時はテクスチャをリング・バッファに!

SLIパフォーマンスのデバッグ

- NVPerfKitのSLIサポート:
  - ハードウェアとドライバのシグナル
  - PIX
  - perfmon.exe
  - pdh (ゲーム、VTune、その他…)

SLIパフォーマンスのシグナル

- 合計SLI転送量
- 合計SLI転送回数

- これは
  - 頂点/インデックス・バッファ
  - テクスチャ
  - 描画対象

質問?

- GPU Programming Guide、第8章
  http://developer.nvidia.com/object/gpu_programming_guide.html
- http://developer.nvidia.com
  The Source for GPU Programming
- mwloka@nvidia.com

The Source for GPU Programming

developer.nvidia.com

- Latest News
- Developer Events Calendar
- Technical Documentation
- Conference Presentations
- GPU Programming Guide
- Powerful Tools, SDKs and more ...

Join our FREE registered developer program for early access to NVIDIA drivers, cutting edge tools, online support forums, and more.

NVIDIA

developer.nvidia.com

©2004 NVIDIA Corporation. NVIDIA, and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation. Nalu is ©2004 NVIDIA Corporation. All rights reserved.

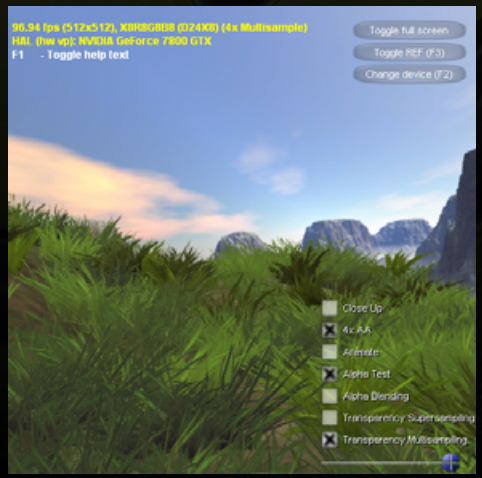
NVIDIA SDK

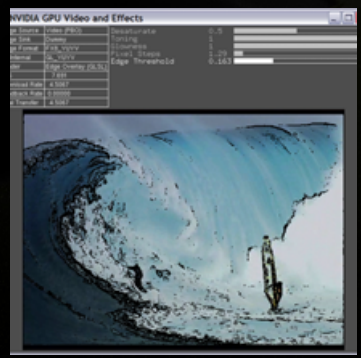
The Source for GPU Programming

Hundreds of code samples and effects that help you take advantage of the latest in graphics technology.

gamedeveloper FRONTLINE AWARD '04

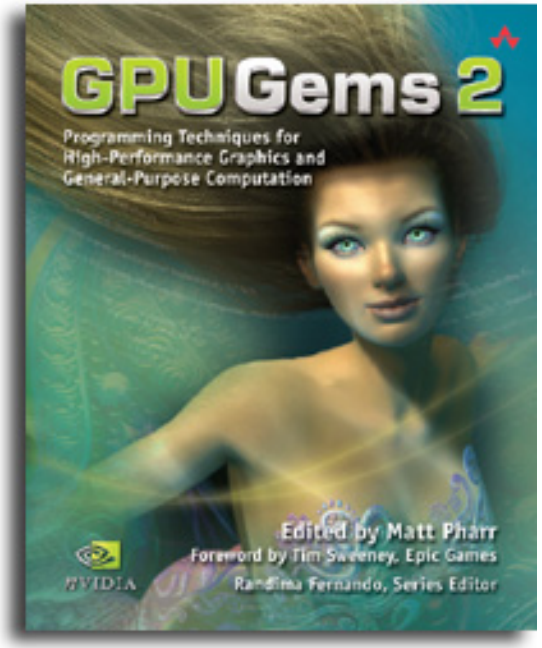
- Tons of updated and all-new DirectX and OpenGL code samples with full source code and helpful whitepapers:
  - **Transparency AA, GPU Cloth, Geometry Instancing, Rainbow Fogbow, 2xFP16 HRD, Perspective Shadow Maps, Texture Atlas Utility, ...**
- Hundreds of effects, complete with custom geometry, animation and more:
  - **Shadows, PCSS, Skin, Plastics, Flame/Fire, Glow, Image Filters, HLSL Debugging Techniques, Texture BRDFs, Texture Displacements, HDR Tonemapping, and even a simple Ray Tracer!**





GPU Gems 2

Programming Techniques for High-Performance Graphics and General-Purpose Computation

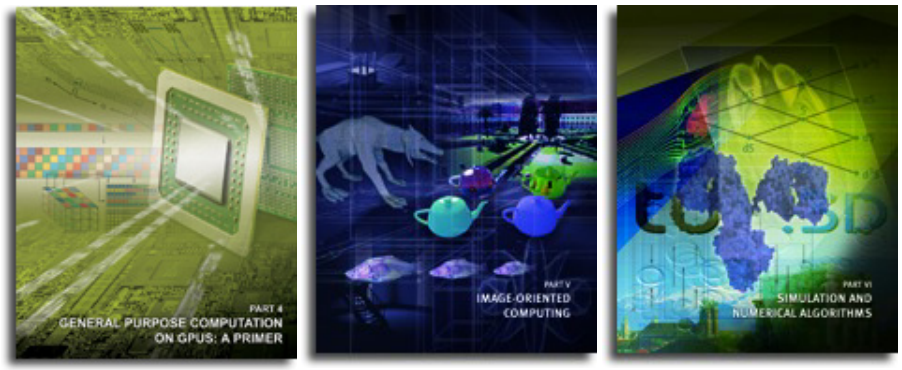


- **880 full-color pages**
- **330 figures**
- **Hard cover**
- **$59.99**
- **Experts from universities and industry**

Graphics Programming

- **Geometric Complexity**
- **Shading, Lighting, and Shadows**
- **High-Quality Rendering**

GPGPU Programming



- **General Purpose Computation on GPUs: A Primer**
- **Image-Oriented Computing**
- **Simulation and Numerical Algorithms**