Squeezing Performance out of your Game with ATI Developer Performance Tools and Optimization Techniques

Jonathan Zarge, Team Lead Performance Tools
Richard Huddy, European Developer Relations Manager
ATI Technologies, Inc.

Outline

- DX9 Optimization Techniques: Richard Huddy – 25 minutes
  - Graphics pipeline overview
  - Optimizing at each pipeline stage
  - Tips for writing efficient code
- Performance Tools: Jonathan Zarge – 30 minutes
  - ATI developer performance tools overview
  - PerfDash
    - our new real-time performance analysis tool
  - PIX plug-in
    - track ATI hardware counters in PIX
  - gDEBugger
    - OpenGL debugging - now with support for ATI performance counters
- ATI Content Creation Tools: Jonathan Zarge – 5 minutes

DirectX 9 Optimization Techniques

Some early observations

- Graphics performance problems are both commoner and rarer than you'd think…
- The most common problem is that games are CPU-limited
- But you can certainly think of that as a graphics problem...
  - As it's most often caused by graphics abuse…

There's plenty of mileage in…

- Instancing
  - Available on all ATI's recent hardware
  - That's all SM3 hardware
  - On ATI's SM2b hardware thru a simple backdoor…
  - Use Instancing for objects up to ~100 polys
- Batching
  - You will be CPU limited if you don't send your triangles in large groups
    - You can think of this as pretty much a fixed overhead per Draw call in DX9, ***much less*** in DX10…

DirectX9 State Changes

- Top 5 by cost:
  - SetPixelShaderConstant()
  - SetPixeShader()
  - SetVertexShaderConstant()
  - SetVertexShader()
  - SetTexture()
- So try to avoid these when you can

Unified shaders?

- Think cross platform…
  - Xbox 360
- When it happens the dynamics of PC graphics will change radically

Shall we have a target?

- 1600x1200 and 1280x1024 (at least)
- 85Hz (then lower refresh rates will just work)
- 4xAA or 6xAA – so pixels can look good
- Because of the variability of the platform it makes no sense to ask blindly if we are pixel-limited or vertex-limited etc.
  - [And with U.S. that idea stops making sense…]

Down the bottleneck pipeline...

- Cache re-use
  - VFetch, Vertex, texture, Z
    - All caches are totally independent of each other...
- Vertex shaders
- Pixel shaders
- Z buffer
- Frame buffer

The pre-VS cache I

- Is purely a memory cache
- Has a common line size of 256 bits
  - (That's 32 bytes)
- Is accessible by all vertex fetches
- Is why vertex data is best aligned to 32 bytes or 64 bytes
  - 44 is very much worse than 64
  - Roughly sequential access should be your aim

The pre-VS cache II

- Because it's purely a memory cache...
  - Multiple streams can both help and hinder.
    - Multiple streams with random access is doubly bad...
  - Generally expect 0% to 10% hit for using additional streams

Vertex Engines I

- Consider compressing your vertex data if that helps you line things up with the 32 byte cache line...
  - Decompress in the Vertex Shader
  - Store compressed data in VB
- See previous slide for the point...
- This can be a significant win if it achieves some key alignment objectives

Vertex Engines II

- HLSL is your best approach...
  - We recommend that you compile with optimisations disabled
  - we'll get to know more that way and usually do better
- Expect one op per clock per pipe
  - Sometimes you'll get 2 ops instead...
  - Masking out unused channels helps
  - You *can* get up to 5 ops at once!
- I've never seen a game which is vertex-throughput limited at interesting resolutions on modern hardware

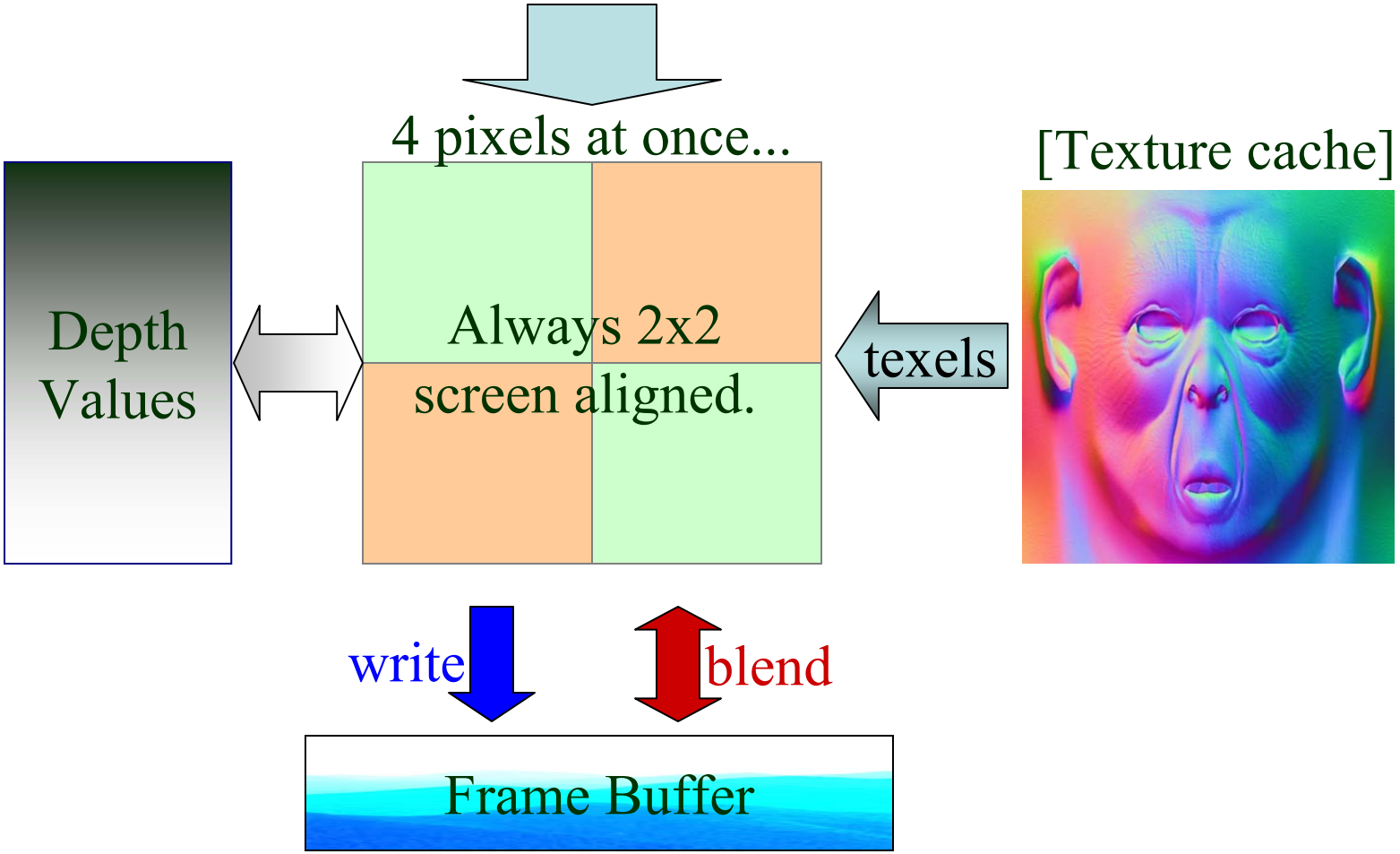
The post-VS cache

- Only accessible when using indexed primitives (can give you '*free*' triangles)
- Operates as a FIFO
- Use D3DXOptimizeMesh()
- Is 14 entries for triangles, 15 for lines and 16 for points
- Cache Size is independent of vertex format!
- Use highly local wending for best results
- Flushed between DrawPrim() calls

Triangle setup

- Never a bottleneck
- Just joins vertices into triangles
- Feeds the rasterizer which simply hands out quad pixel blocks to draw

A Quad-Pixel Processing Unit



Texture cache

- Probably smaller than you'd think...
  - Unless you thought "only a few KB"
- Partitioned over all active textures
  - So heavy multi-texturing can really hurt
  - Modern hardware has efficient fully associative caches
- Wrecked by random access!
  - Often from bump-map into env-map
  - Needs reuse to show benefits (i.e. don't minify!)
- Usually contains uncompressed data
  - At 8, 16, 32 or more bits per texel
  - Some hardware stores DXT1 in compressed format
- Texture fetches are per-pixel

Making Z work for you…

- We’re faster at rejecting than at accepting...
  - So draw roughly front to back
  - For complex scenes consider Z pre-pass (not for depth_complexity=1!)
  - Take care to Clear() Z (and stencil)
- Although Z is logically at the end of the shader that’s not the best way

Making Z work for you…

- Note that ATI hardware can do double speed Z/Stencil only work when:
  - Color-writes disabled
  - AA enabled
    - Good for general rendering
    - AA is your default, yes?
  - That’s up to 32 AA Z values per clock

Depth Values

- Can come from:-
  - Actual Z buffer (slow)
  - Compressed Z (fast & lossless)
- Your pixel can be Z-tested away before the shader has run at all!
- If you are performing any Z compare then please try hard not to write to oDepth
- Remember that depth values are per-sample...

Bashing the depth buffer

- You can reduce the huge(*) early Z benefits by...
  - Writing oDepth
    - Kills compressed Z and early Z
  - Using alpha-test etc on visible pixels
    - decompresses Z values
  - Changing the Z compare mode (sometimes)
    - Can disable Hi-Z
    - E.g. from LESS to GREATER

(*) Top class h/w can reject 256 pixels per clock!

The PS Unit I

- Shorter shaders generally faster
  - And we can cache them on chip too…
- At the high end there is roughly 4 times as much ALU power as texture power
- This ratio will only go up
  - Because available bandwidth doesn't rise as fast as chip density
- So generally push more maths into here

The PS Unit II

- Is a 4D vector processor
  - So try to match your math to your needs
    - i.e. Mask out unused channels
- Trust the compilers to schedule things well:-
  - You don’t worry about scheduling...
- PS runs once per pixel...

FB (Fog and) Blend

- Is *not* part of the PS unit
  - You can think of it as a special function of the memory controller
- Although there are lots of latency hiding tricks here...
  - This is still probably the easiest place to get B/W limited
- So disable blend whenever possible

Pure FB optimisations

- Fewer bits are written faster...
  - 16BPP > 32BPP > 64BPP > 128BPP
    - (here '>' means faster)
- Blending is slower than not
  - Often by more than a factor of 2
- ATI: Surfaces are 'faster' when allocated earlier!

PS Dynamic Flow Control

- DFC can be a significant benefit…
  - But only when the selection coherency is at least as big as the hardware batch size

Hardware	Batch Size
X1800	16 pixels
X1900	48 pixels
Xenos	48 pixels

Conclusion...

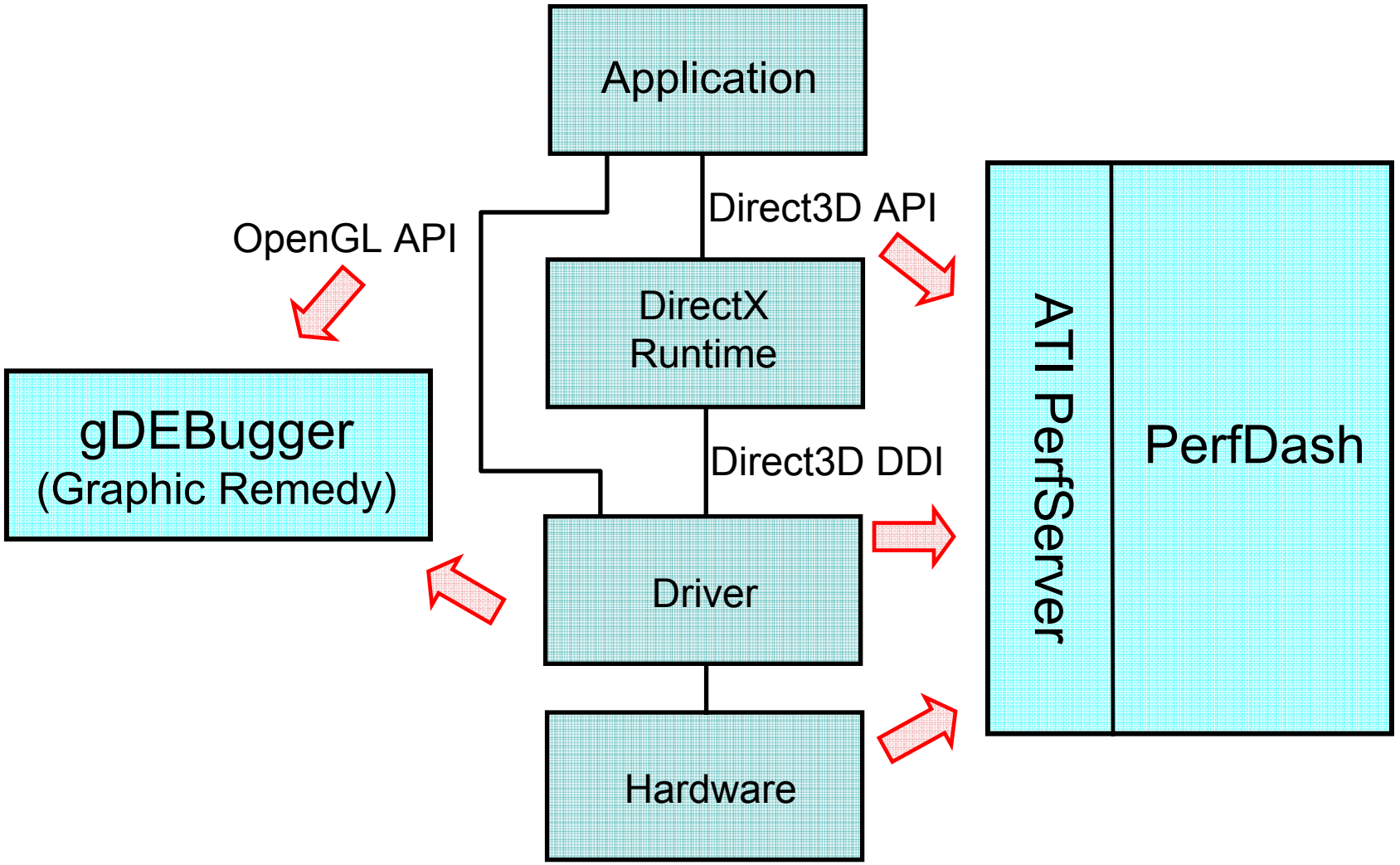
- Several classes of optimisation:
  - Pushing things back up the pipe:
    - E.G. Cull early, not late
  - Getting better parallelism:
    - E.g. Use write masks in your shader code to allow SIMD
  - Doing less is faster than doing more:
    - E.g. Short shaders are faster
  - Understand what is cached:
    - 32 byte vertices are fast! 16 bytes are faster...

Performance Tools

The Plan

- Overview of ATI developer performance tools
- PerfDash
- ATI PIX Plugin
- gDEBugger
- ATI content creation tools

ATI Developer Performance Tools

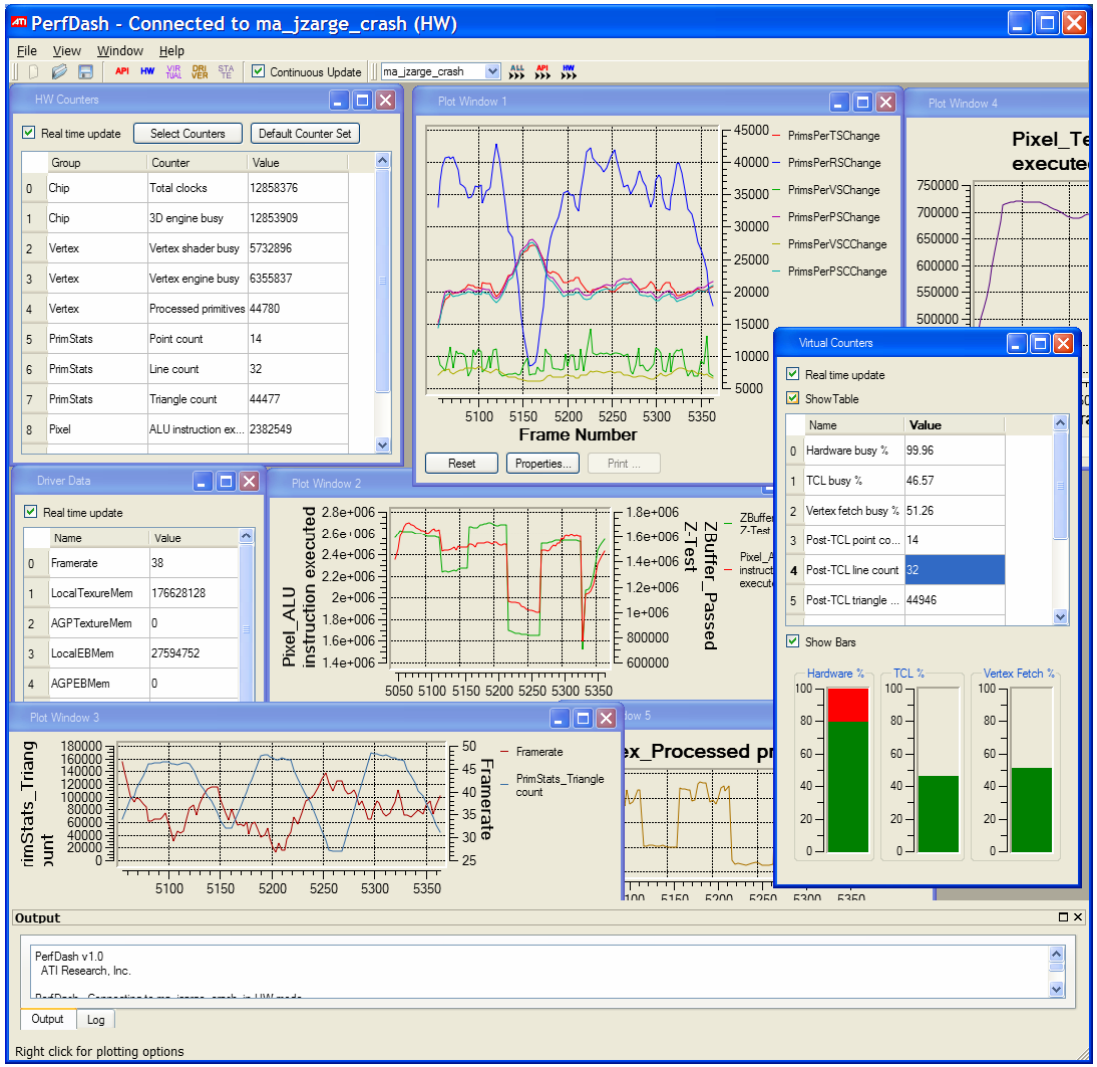


The Plan

- Overview of ATI developer performance tools
- PerfDash
- ATI PIX Plugin
- gDEBugger
- ATI content creation tools

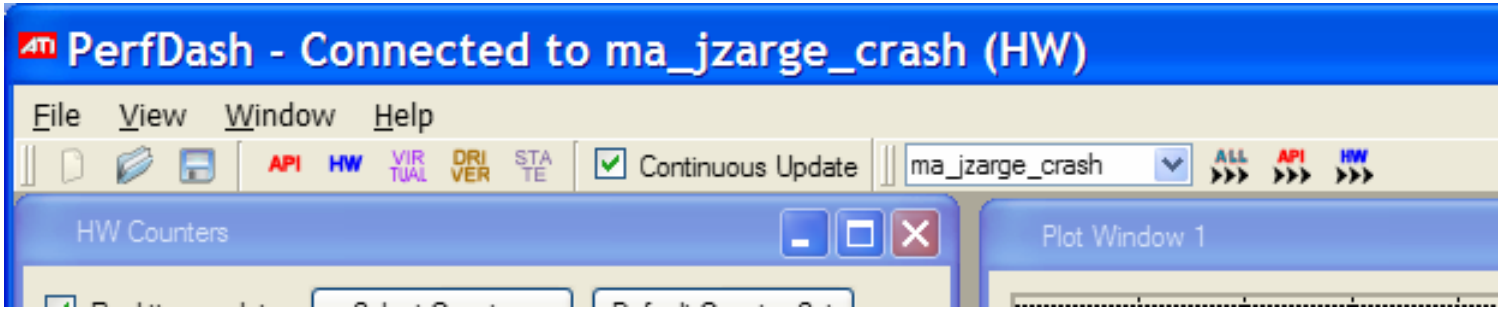
PerfDash Overview

- PerfDash: Performance Dashboard
- Real-time visualization:
  - API statistics
  - Hardware counters
  - Driver data
  - "Virtual" counters
- Local or remote performance profiling
- Overriding rendering states
- Loading/saving session data and preferences
- No special driver
- No code modifications
- Plugin architecture



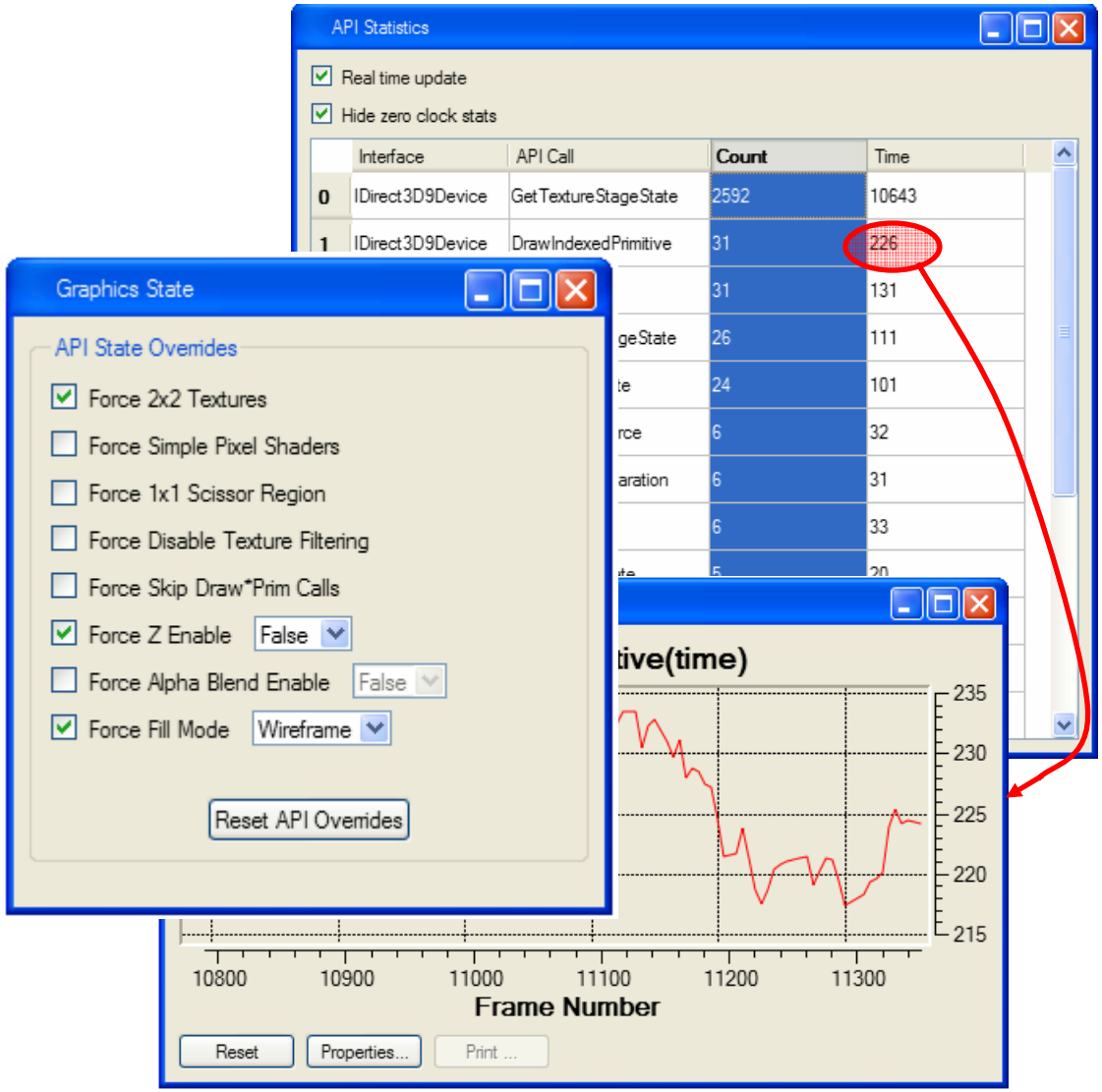
PerfDash Features: Toolbar

- Toggle global data collection
- Connect to local or remote machine
  - API, Hardware, All modes
  - Performance server must be running on target machine
    - Virtually no performance impact if not running PerfDash



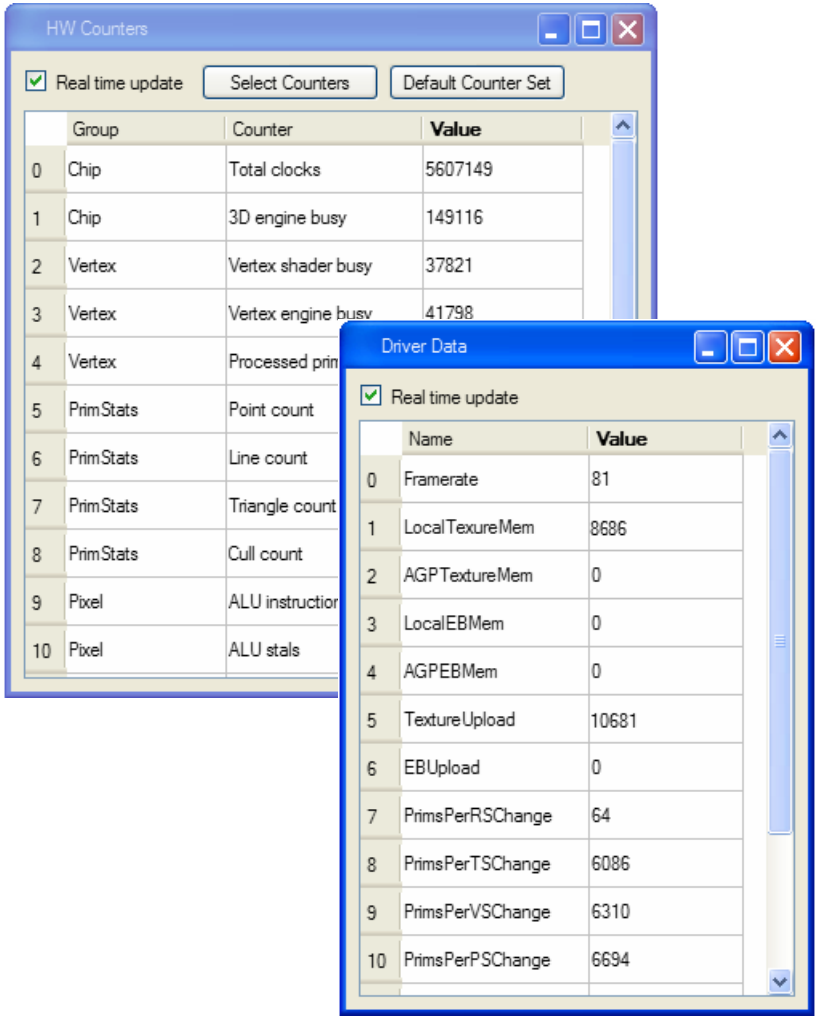
PerfDash Features: API Statistics

- Per-frame API call data
- Sorting of API call counts and times
- Flexible plotting of all numeric data
- Plot window properties control appearance
- Real-time state overrides



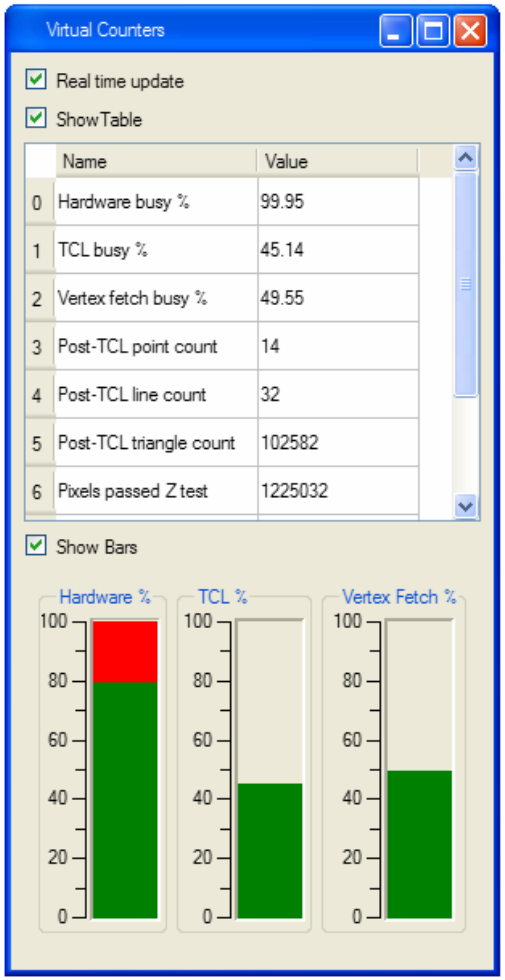
PerfDash Features: Hardware

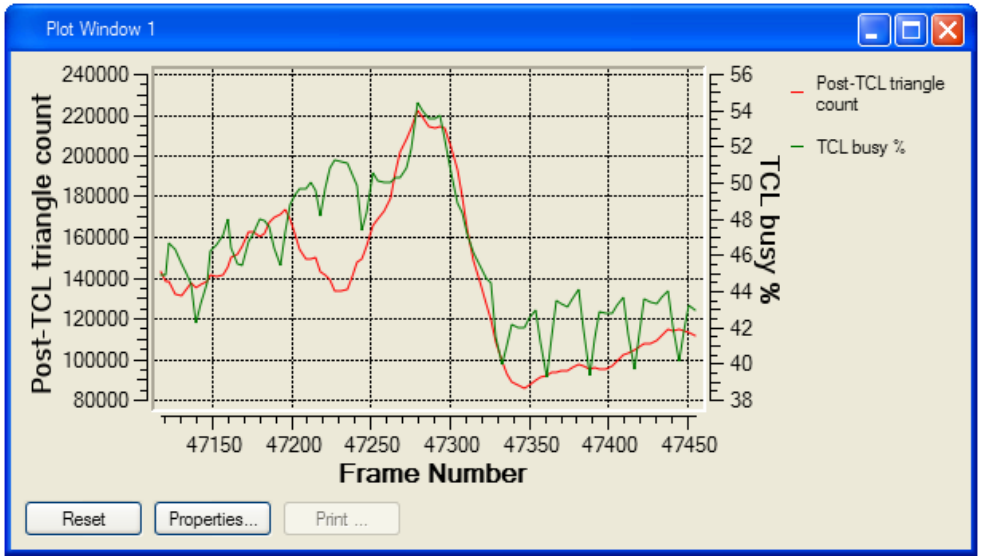
- Hardware counter values
  - 3D/TCL clocks
  - Primitive counts
  - ALU instructions executed
- Select custom set of counters
- Driver data
  - Framerate
  - Memory in use
  - Prims per state change
- Plotting of all numeric data



PerfDash Features: Virtual Counters

- Virtual (derived) counters
  - Hardware busy %
  - TCL busy %
  - Pixels passed z-test
- Temperature bars
- Plotting of all numeric data

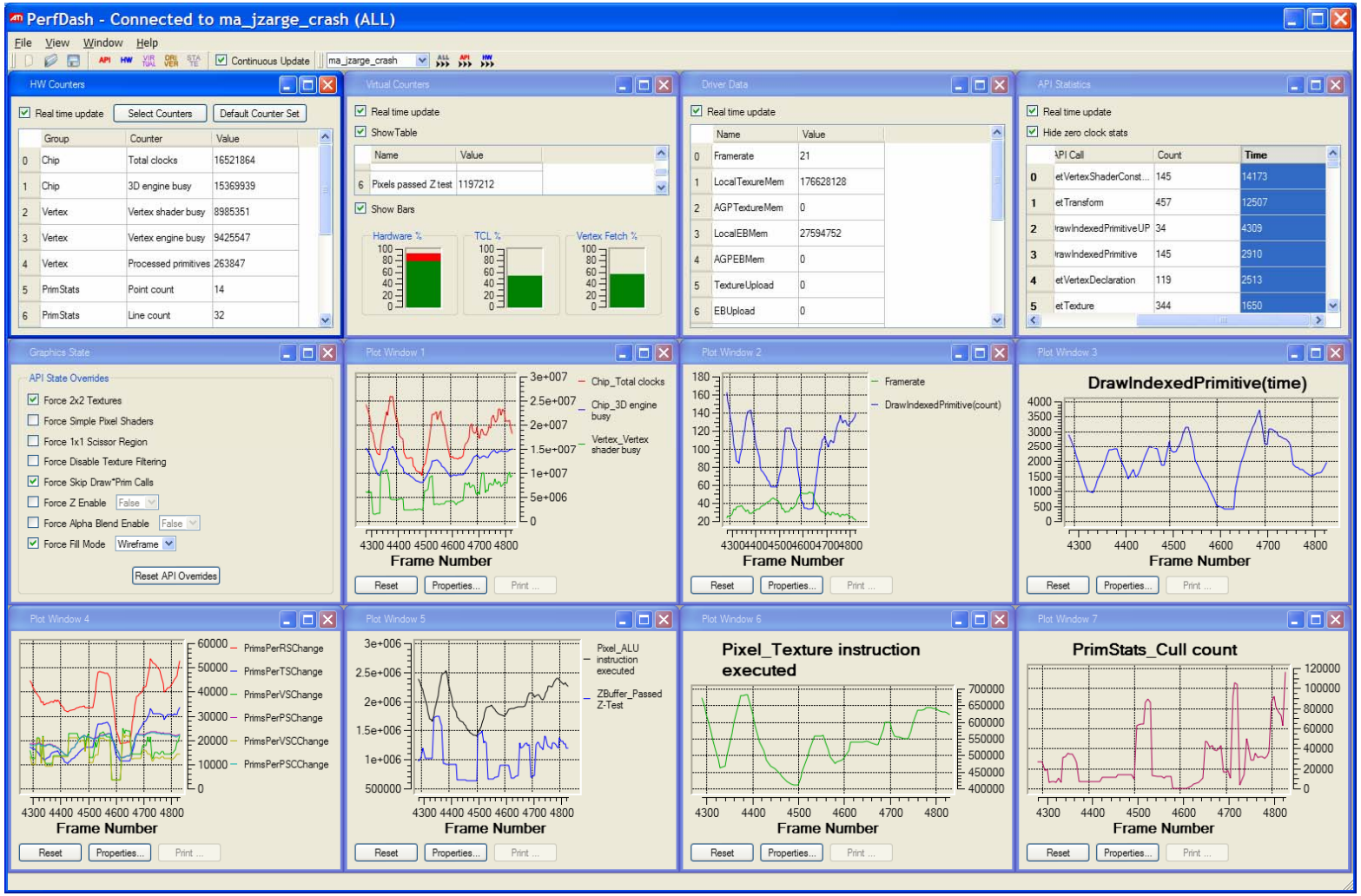




Performance Tuning w/PerfDash

- CPU vs GPU balance
  - Is the GPU saturated?
- TCL percentage
  - Is vertex processing the bottleneck?
- Pixel shader bottleneck
- Draw*Primitive per state change
  - Good indication of batch size
- Inefficient use of API
- High memory usage
- Texture bandwidth and filtering
- Efficient use of z-buffer and sorting
- Wireframe for geometry visualization

PerfDash Demo



PerfDash Future

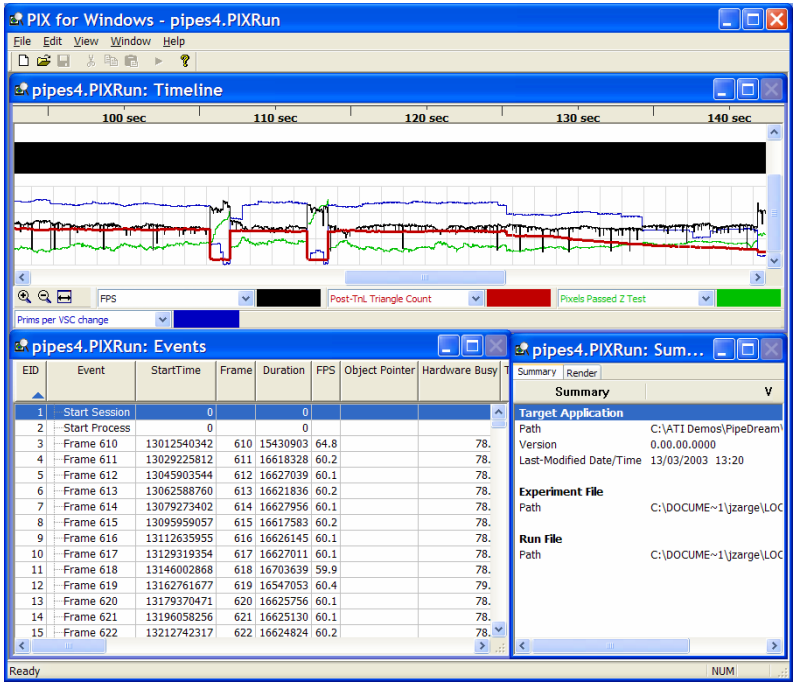
- Future enhancements
  - Record/playback/step through API calls
  - Support external plugins
  - Support for different platforms
  - More data: pixel stats, render states, buffers, surfaces
  - Bottleneck detection
  - Custom virtual counters
  - Support for OpenGL applications
- PerfDash release schedule
  - Beta in Q1 06
  - First release Q2 06

The Plan

- Overview of ATI developer performance tools
- PerfDash
- ATI PIX Plugin
- gDEBugger
- ATI content creation tools

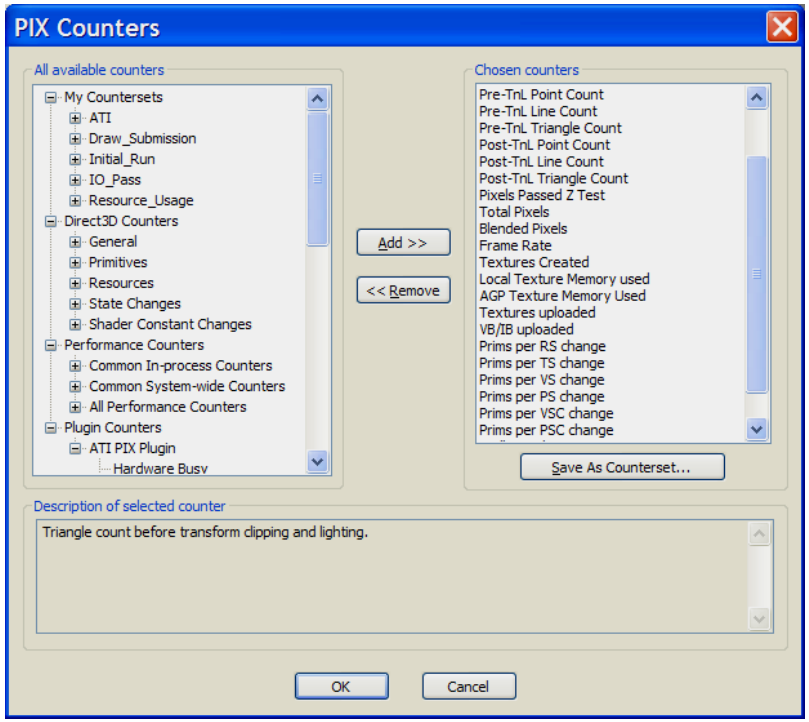
PIX for Windows

- Based on Xbox PIX
- Record/playback/visualize model
- Direct3D API level
- Numerous included counters
- Plugin API for additional counters
- Image comparison
- Display render states, textures, objects, shaders, vertex decl.
- 3 recording modes
  - Gather statistics (counters)
  - Record D3D calls
  - Record playable stream information



ATI PIX Plugin

- Communicates with ATI Driver (through D3D API)
- Virtual counters
  - Computed from actual hardware counters/driver data
  - Can be graphed with other performance counters
- Counter examples:
  - Hardware/TnL Busy
  - Vertex Fetch Busy
  - Triangle/Line/Point Count
  - Total/Blended/Pass Z Pixels
  - Local/AGP Texture Memory Used
  - Primitives per RS/TS/PSC/VSC
  - Stalls on Flip/VB

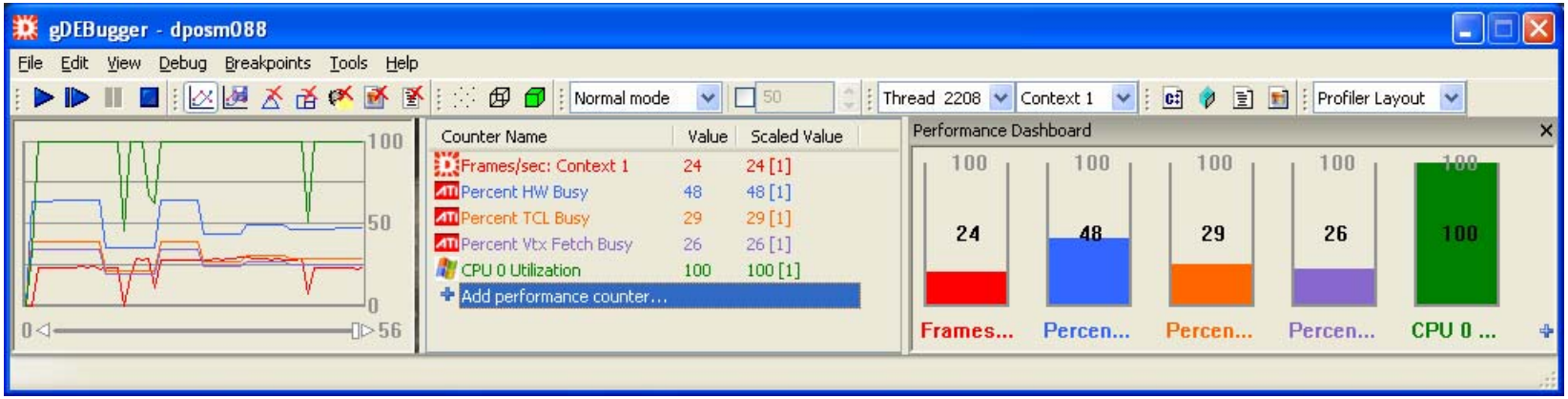


The Plan

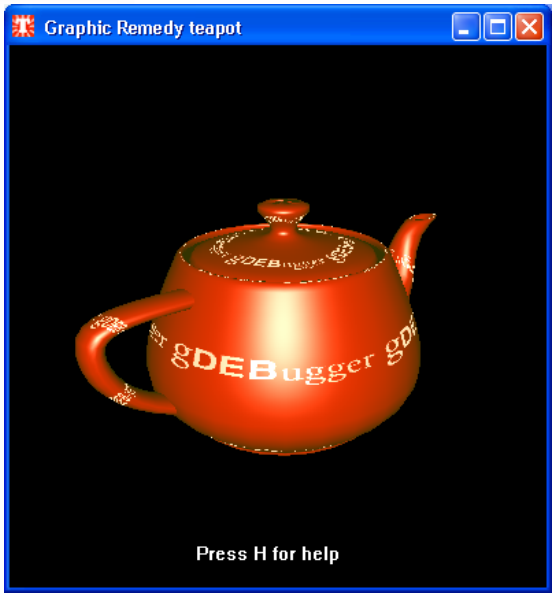
- Overview of ATI developer performance tools
- PerfDash
- ATI PIX Plugin
- gDEBugger
- ATI content creation tools

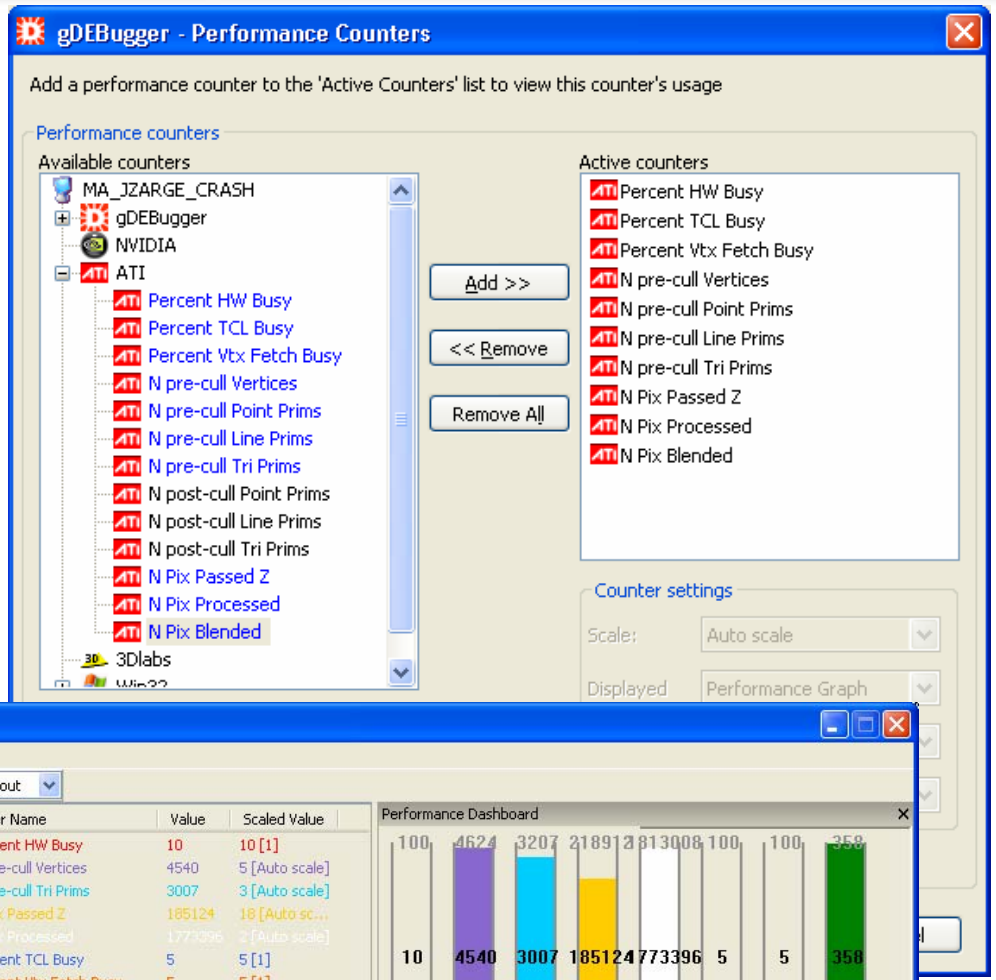
gDEBugger adds ATI performance counters

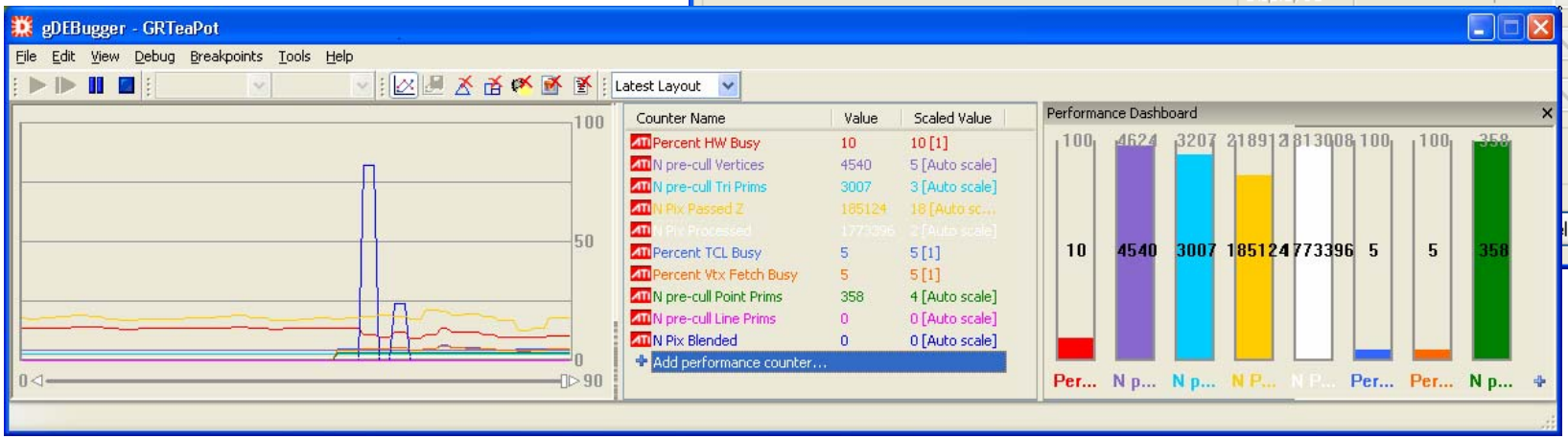
- gDEBugger & gDEBugger ES:
  - A professional OpenGL and OpenGL ES Debugger and Profiler
  - Shortens developer time required for debugging and profiling OpenGL and OpenGL ES based applications
  - Integrated with ATI performance counters to find graphic pipeline performance bottlenecks and optimize application performance
- More details:
  - www.gremedy.com
  - ATI Workstation SDK



gDEBugger Demo



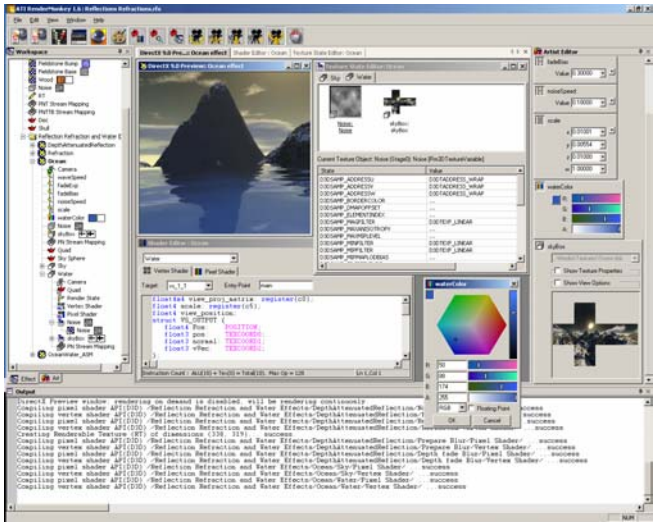




The Plan

- Overview of ATI developer performance tools
- PerfDash
- ATI PIX Plugin
- gDEBugger
- ATI content creation tools

ATI Content Creation Tools



- RenderMonkey
  - Shader development environment
  - Supports HLSL, D3D assembler and GLSL
  - New version available soon!
- Compressonator
  - Tool for compressing textures and creating mip-map levels
- CubeMapGen
  - Tool for creating filtered cube maps without seams
  - Uses angular extent filtering
- NormalMapper
  - Automatic normal map generation tool
  - Traces rays from low resolution geometry to high resolution geometry

Wrap up

- Graphics features are a key product differentiator
- Increasing graphics efficiency can lead to
  - Playable performance on more hardware
  - More time for non-graphics functions
  - Richer stunning effects for your game
- How do you ascend to these graphic heights?
  - Incorporate the optimization techniques in to your code
  - Utilize the performance tools in your development process

Thank You!

Questions?

JZarge@ati.com
RHuddy@ati.com
devrel@ati.com
http://www.ati.com/developer/