NVIDIA®

Mobile Performance Tools and GPU Performance Tuning

Lars M. Bishop, NVIDIA Handheld DevTech

Jason Allen, NVIDIA Handheld DevTools

NVIDIA GoForce5500 Overview

- World-class 3D
  - HW Geometry pipeline
  - 16/32bpp textures and color buffers
  - Programmable pixel shading
  - QCIF, QVGA, VGA, XGA screen sizes!
- Integrated multimedia features
  - HW video decode (video textures!)
  - HW video encode (videoconferencing)
  - HW camera support (live camera into a texture!)
  - HW audio support

NVIDIA GoForce5500 3D

- Geometry Pipeline
  - HW Transforms
  - Vertex Buffer Object Support
- High-performance Texturing
  - 1024x1024 textures
  - Mipmapping w/ trilerp
  - Compressed textures
- Powerful pixel shading programs
  - Up to 5 textures (and 12 texture samples!) per pass
  - Complex shader instructions
  - Access to additional per-pixel components

Performance Considerations

- Warning: Many of these items may look familiar
- 3D on Handhelds is not wildly different from desktop or mobile 3D
- But the specifics and balances are different
- We'll focus on these a bit

Holistic Performance Considerations

- The GPU doesn't exist in isolation
- Balance the three major system components:
  - CPU
  - System bus
  - GPU
- Any one of them can kill performance

- But on HW-accelerated handhelds, the GPU is the *least* likely candidate as the *initial* bottleneck today…

CPUs

- CPUs on today's handhelds (especially low-power devices) are limited compared to PCs even from years ago
- ARM9's with no FPUs are very common
- ARM11's are gaining in popularity
  - But are still a small subset, and the FPU is optional…
- Caches are smaller

Minimizing CPU Work

- Know your CPU
  - Avoid floating point on an ARM9!
  - Be careful with it even on ARM11+VFP
  - Be cache-friendly (avoid many passes over vertex arrays)
- Avoid redundant render state changes
  - Needless driver work can cost
  - Many drivers don’t “fast path” redundant calls
- Optimize triangles per call
  - Batch triangles
  - Use multitexture/shaders to avoid multipass rendering

System Bus

- Narrower and slower than the PC
- 16-bit still common, indirect buses still common
- Result is lower bus bandwidth
  - Especially when data is sent in small bursts

Minimizing System Bus Traffic

- Use VBOs wherever possible
  - Mark them as GL_STATIC_DRAW when possible
  - Use VBOs for index buffers, too! (almost always static)
- Avoid texture loads per frame
  - Use render-to-texture for dynamic textures
- Don’t read back the framebuffer
  - Unless you are taking screen shots…

GPUs

- Moving ahead very quickly (perf and features)
- So the key is to feed them well
- But it is still possible to choke a good GPU well below its peak rate
- In order to minimize power consumption, various rendering features are not “free”
  - I.e. you don’t pay for them when you don’t need them…

Maximizing GPU Performance

- Maximize texture throughput…
  - Format
  - Dimensions
  - Access coherence
- Maximize triangles-per-call
  - Single-pass effects
  - Batching

Texture Formats

- Use compressed textures
  - GoForce supports DXT1/3/5 natively at full performance!
- Save 16- and 32-bpp textures for when you need them
  - And prove that you do!
- Use single-channel (8bpp) textures when you can
  - Often useful in shaders
  - Good precision without large size

Optimizing Texture Sizes

- Don’t just blindly turn off mipmapping!
  - Dropping just the finest mip-level saves ~3x the pyramid
  - See if you are even *using* the finest mip-level!
- Create large “virtual textures”
  - Use single-pass multitexture and shaders
  - Compose smaller ones at different scales
  - E.g. (detail * base * darkmap * 2)
  - 3 256x256 textures can create the effect of 1024x1204 in 3/16 the space

Mipmapping

- Use mipmapping to increase performance
- But don’t waste mipmap pyramids
  - Skip them for 2D UI elements
- Remember that embedded memory on handheld is optimized for power, not just speed
- But use trilinear filtering only when needed

Isolating Performance Bottlenecks

- All of these recommendations are great, but how do I figure out which ones will help *my* app?
- "Old-school" – Modify application to add:
  - Performance counters, timers, wrappers around your OpenGL calls, ability to turn off functionality
- The new way – NVIDIA PerfHUD ES
  - The PC developer's good friend is now available for GoForce mobile GPUs!
  - Makes it easy to do initial performance analysis of OpenGL ES applications

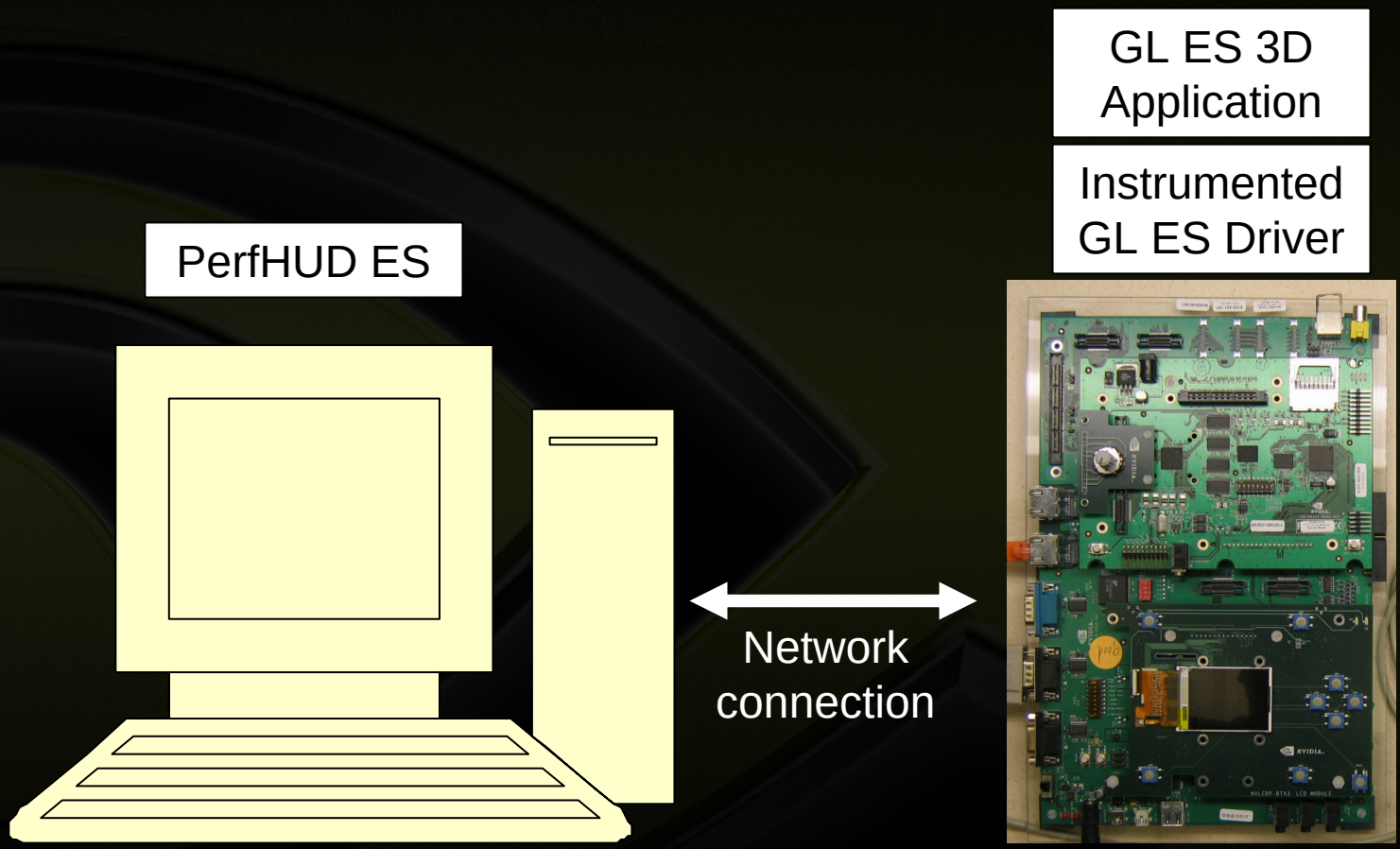
PerfHUD ES: What is it

- Perf HUD ES is the OpenGL ES analogue to the popular and powerful PC PerfHUD tool
- Provides an instrumented driver and a client UI

- Supports
  - Live performance monitoring of running app
  - “Directed tests” to help isolate bottlenecks

- Unlike the PC PerfHUD, the ES version displays the results on a host PC, not the handheld screen
  - Would you *want* to see stats on a 3” VGA/QVGA?

PerfHUD ES: How it works



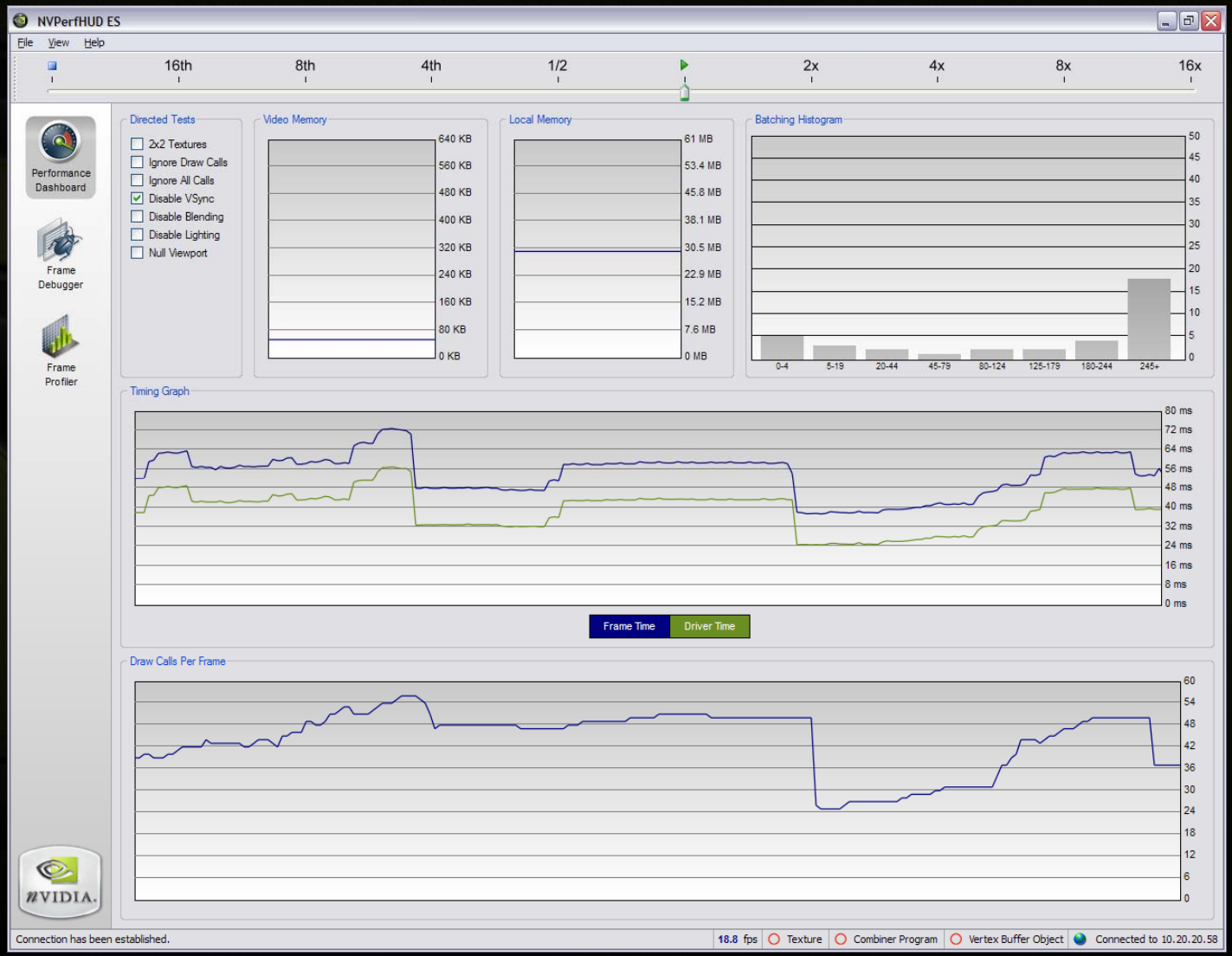
Host PC:

Runs PerfHUD ES UI Client and serves devkit file system

GoForce HW devkit:

Runs target app and the instrumented driver

PerfHUD ES Client

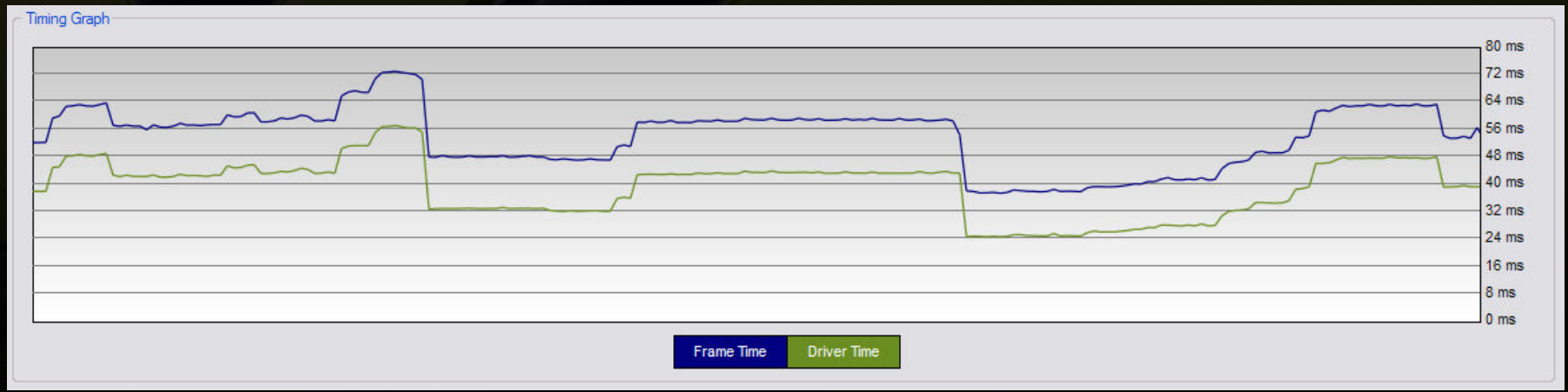


Performance Statistics

- Current (running mean) frame rate
- Timelines with per frame reporting of lots of data
- Histogram of batch sizes (tris per call)
- Indicator lights that flash on expensive operations

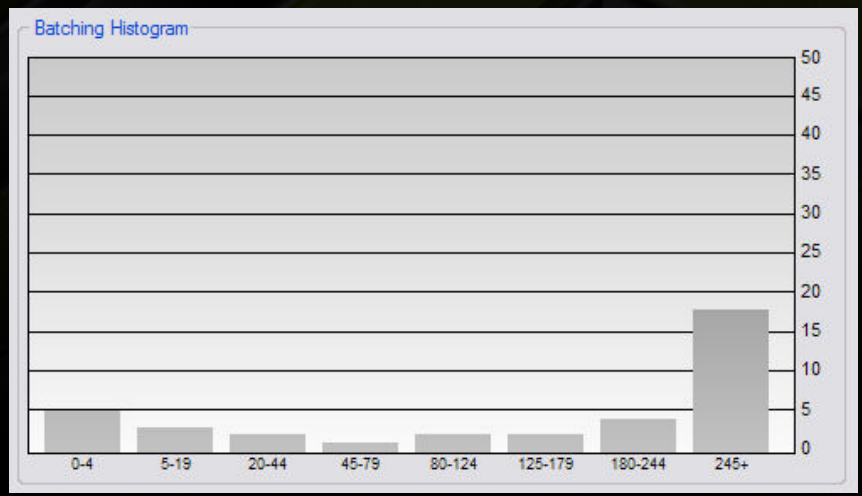
Timelines

- Total frame time
- Frame time spent in the driver/GL ES
- Number of draw calls in frame
- Video memory used
- System memory used



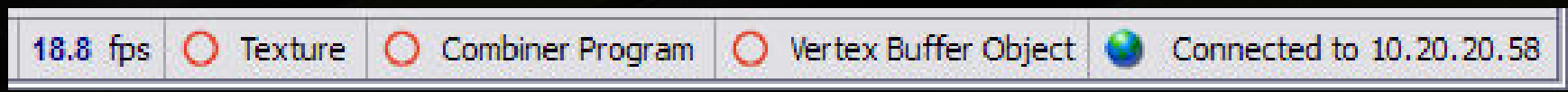
Batch Histogram

- A graph of the number of triangles per draw call
- Most experienced developers are pretty good these days at getting their batch sizes up
- But… Look out for particles and text systems, which are sometimes the causes of poor batching
  - Especially when they are used more heavily than they were designed



Event Indicator Lights

- Loading new texel data to a texture
- Loading new data to a VBO
- Creation of a new pixel shader
- Each of these should probably be investigated if they are blinking every frame (or frequently)



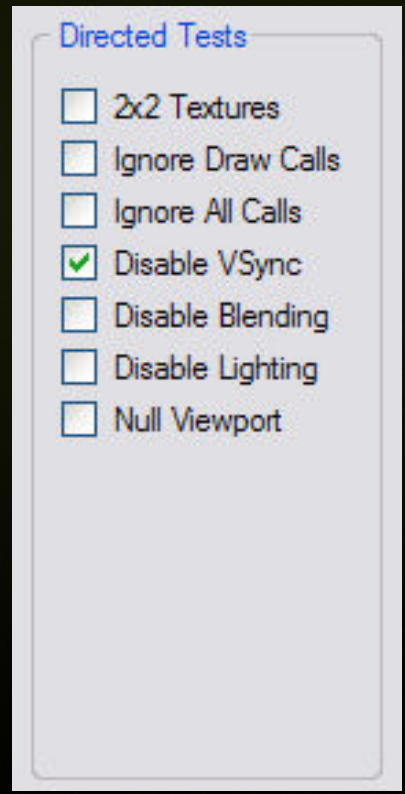
Using the Statistics as Triage

DevTech can use the passive stats to quickly analyze a newly-received app. We frequently look at:

- *The lights*
  - Are textures or VBOs getting created/updated frequently?
  - Textures are of particular interest
- *The batch-size histogram*
  - Is the mean number of tris per batch low?
- *The total-time and driver-time timelines*
  - What is the overall ratio of app time (total minus driver) and driver time? Focus where the time is being spent

The “Directed Tests”

- Optional modes that intercept rendering and state calls to change the rendering without having to modify the app
  - Replace all textures with a 2x2-texel texture
  - Ignore glDraw* calls
  - Ignore gl* calls
  - Disable VSYNC
  - Disable pixel (AKA Alpha) blending
  - Disable GL_LIGHTING
  - NULL viewport



Directed Test Use-case Examples

- Think you might be fill-rate bound?
  - Enable “NULL viewport”
  - If the frame rate shoots up, you may be fill-bound
- Think you might be texture-bandwidth bound?
  - Enable “2x2 textures”
  - If the f.r. shoots up, you may be texture-bound
- Think you might be app-bound?
  - Enable “Ignore all GL calls”
  - If the f.r. does *not* shoot up, you are likely app-bound
- And so on… All without touching your source code

Speed Control

- Advanced feature – requires a simple modification to the app (timer extension)
- Allows the user to slow or even stop time in the app
- Makes it possible to replay the same frame over and over while changing directed tests, etc
- Useful for isolating bottlenecks in a particular scene

Coming Attractions

- This is just the first version of PerfHUD ES
- Much more to come!
- Some features will be similar to those in PC PerfHUD
- Others will continue to be very handheld/embedded specific

Questions??

handset-dev@nvidia.com