NVIDIA

SIGGRAPH 2013
Shaping the Future
of Visual Computing

The Future of Visual Computing: OpenGL 4.4 on ARM

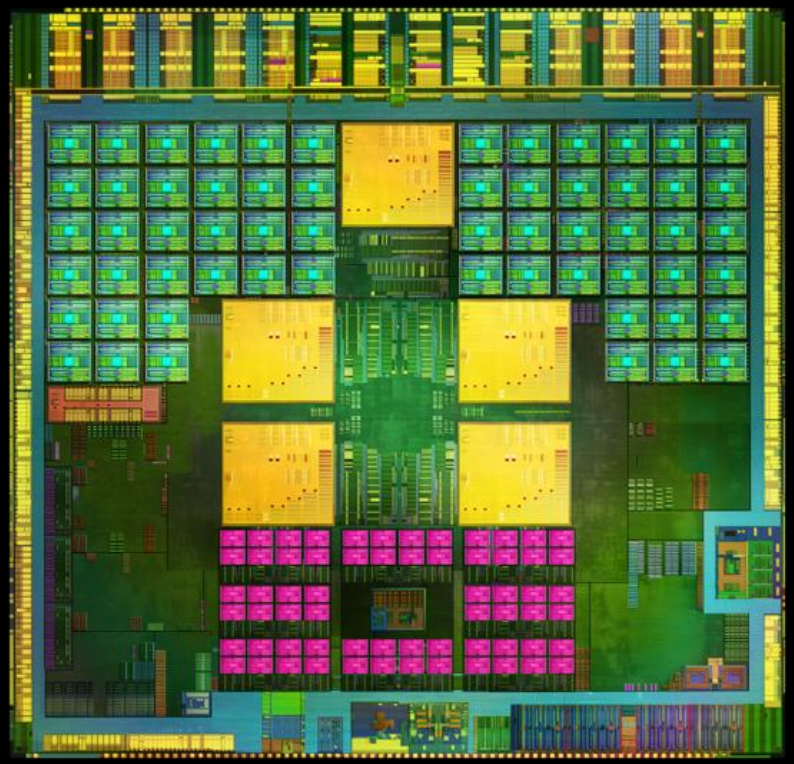
Cass Everitt, Principal Engineer

Major Influences

- Rise of the SoC
- Established path of desktop
- Re-emergence of OpenGL
  - My 1st SIGGRAPH was 1993 in Anaheim
  - Took the OpenGL course!

Rise of the SoC

- Dominant CPU architecture is ARM
- Mobility devices define the market today
- Embedded growing in relevance
  - Set-top-box, automotive, unusual form factors
  - Support for advanced 3D rendering ubiquitous
- Platforms tend to be Linux-like
  - The price is right
- Performance AND power draw matter
- Dynamic range of power draw high



Established Path of Desktop

- Desktop has continued forward march
  - Defining GL4-class (DX11-class)
  - Developer focus has lagged
    - Consoles remained GL2-class (DX9-class)
    - ES2 emerged as GL2-class
- New console generation announced
  - Developer focus shifting toward GL4
  - Mobile still lags, *but not for long*

Re-Emergence of OpenGL

- 5 years ago GL was (still) “almost dead”
- Now arguably the most important API for future development
  - Growth market
- But GL is quite fragmented
  - Many incompatible versions, portability suffers
  - Regal can help (more on this later)

New in OpenGL 4.4

NEW!

- Immutable buffer storage
  - Persistent mapping
- Texture clear
  - Without binding into FBO
- Enhanced layouts in GLSL
  - Simpler CPU / GPU sharing
- Multi-bind
  - All bindings in one fell swoop
- Query result -> buffer
  - skip CPU

- Texture mirror clamp
- Stencil-only textures
  - Render-buffers obsolete

New ARB Extensions

NEW!

- Dynamic compute group size
  - Argument to launch now
- Indirect parameters
  - Further CPU skipping
- Draw parameters in shader
- Seamless cubemap
  - Per-texture toggling
- Vote
  - Managing SIMD divergence

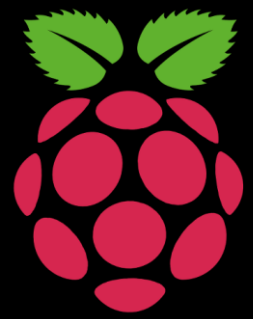
- 10/11/11 packed float vertex format
- Sparse texture
  - Partially resident textures
- Bindless texture
  - Almost unlimited now

SoC Porting Issues

- Power and heat are a constant concerns in mobile
  - Phones particularly sensitive
  - Both battery life and device temp are both central to the user experience
- ARM is less forgiving about unaligned reads
  - Result: unexpected segfaults
- Screen density and resolutions quickly outpaced desktop
  - Tricky for devices with a small fraction of desktop power draw

SoC PC

- Ubuntu 12.04 works great
- Tegra 4 best so far for desktop user experience
  - Slower devices can have noticable lag
- Distcc can really help building large source trees
  - Cross-compile with arm-linux-gnueabihf-g++-4.6
- Linux on ARM has all the tools that Linux on x86 does
  - So nice to apt-get 3rd party libraries already compiled
  - Cross compilers also just an apt-get away on x86 systems



Porting Down

- Develop high then port down

Break into more pieces -

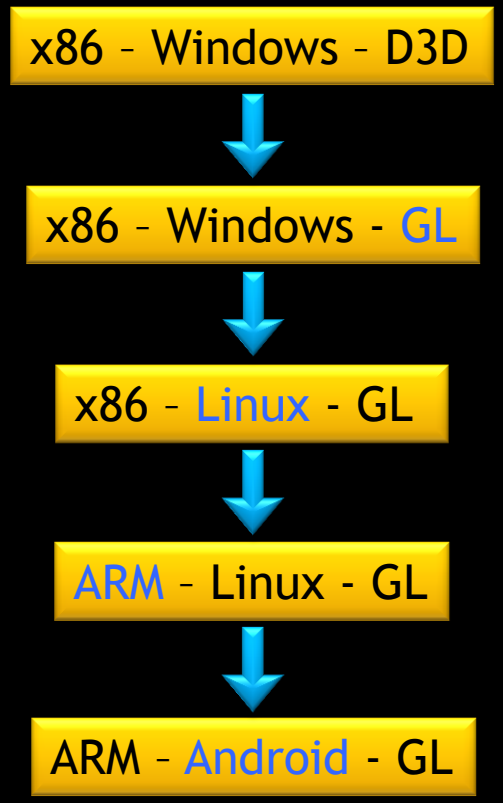
PC

SoC PC

Target

Small Steps

- Ports of big apps have lots of moving parts
- Take small steps when possible
  - Working system after each step
- Skip too many steps, isolating bugs gets hard
- Sometimes steps can be done in parallel
- After the port, still useful to have an intermediate system
  - e.g. ARM – Linux – GL convenient for regression systems that support Linux but not Android

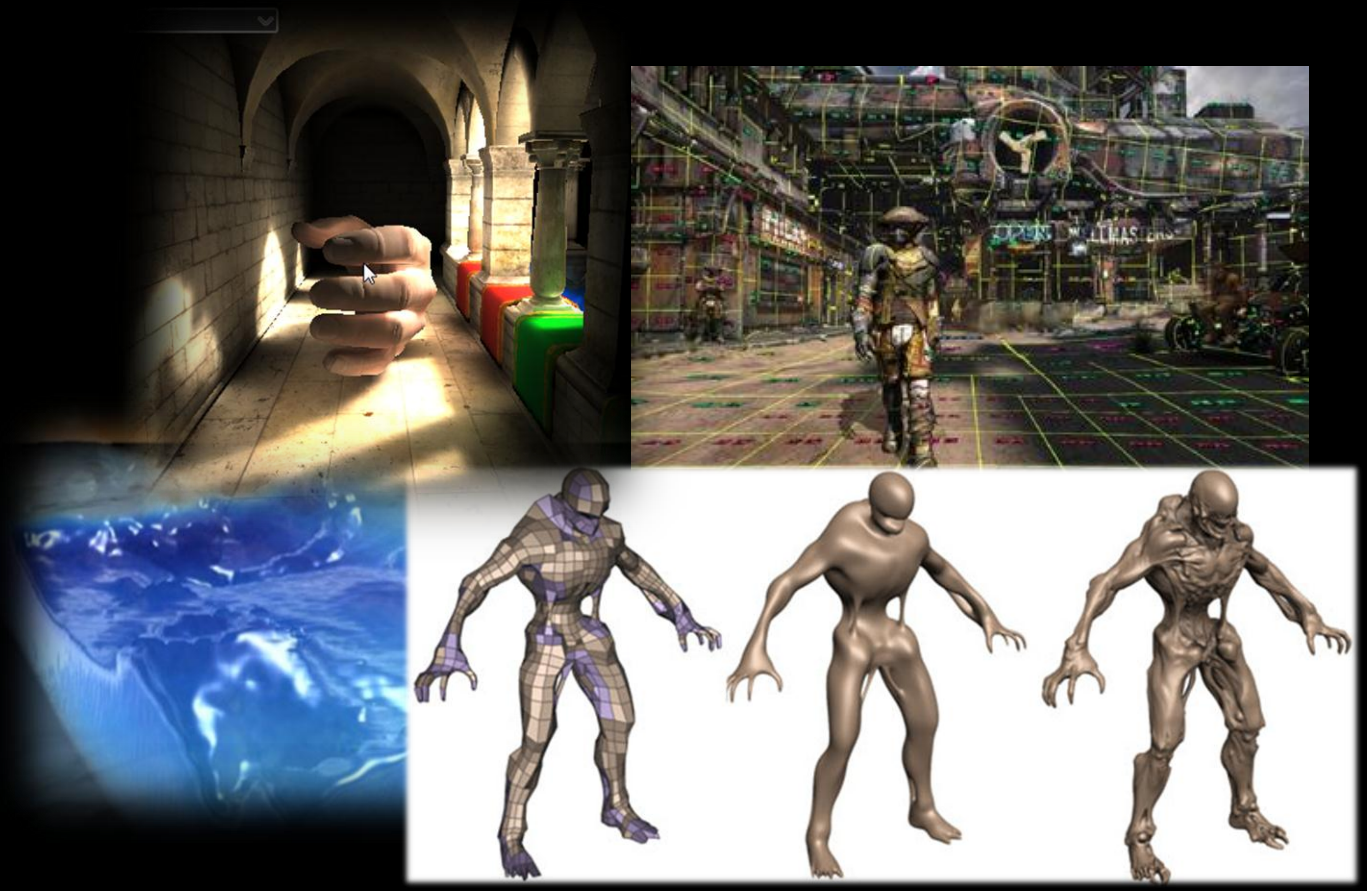


GL versions

- Mobile has historically been OpenGL ES only
  - Full OpenGL 4.4 support is coming to mobile though
- Key differences between OpenGL ES 2.0 and OpenGL 4.4
  - GL ES 2.0 is roughly at parity with GL 2.0 and Direct3D 9
  - GL 4.4 is at parity with Direct3D 11
  - Geometry, Tessellation, and Compute shaders in GL 4.4
  - Texture Arrays
  - Transform Feedback
  - Floating point textures and blending
  - Non-Power-of-Two textures
  - Multiple Draw Buffers

Techniques Enabled by GL 4.4 vs ES 2.0

- Global illumination
- Curved surfaces
- Shadow maps
- Deferred shading
- Forward Plus
- HDR rendering
- Virtual texturing
- Path rendering
- Compute integration

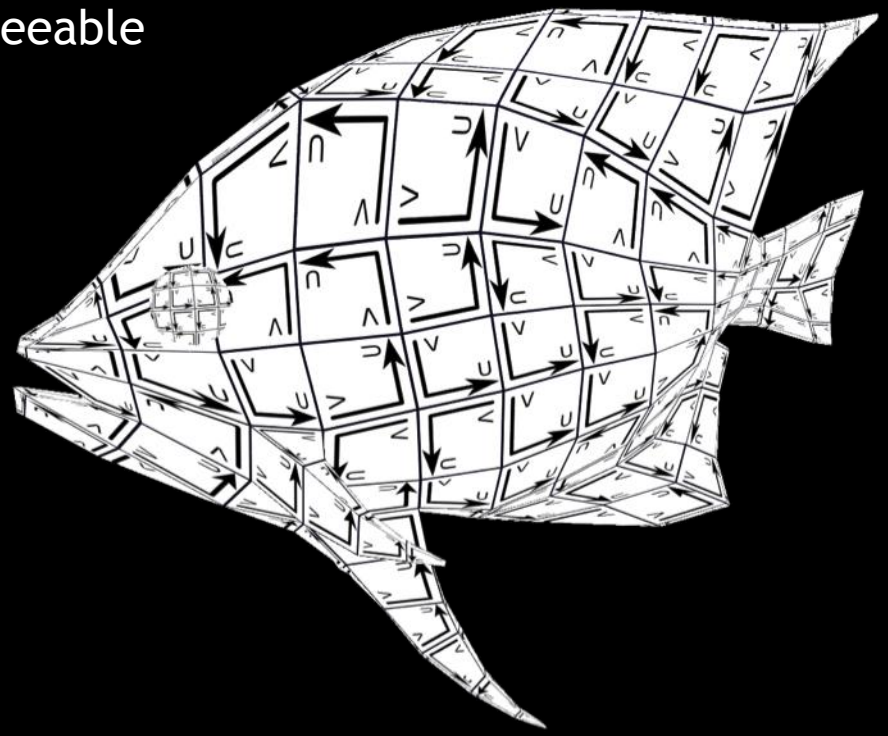


PTEX - What is it?

- The soul of Ptex:
  - Model with Quads instead of Triangles
    - You're doing this for your next-gen engine anyways, right?
  - Every Quad gets its own entire texture UV-space
  - UV orientation is implicit in surface definition
  - No explicit UV parameterization
  - Resolution of each face is independent of neighbors.

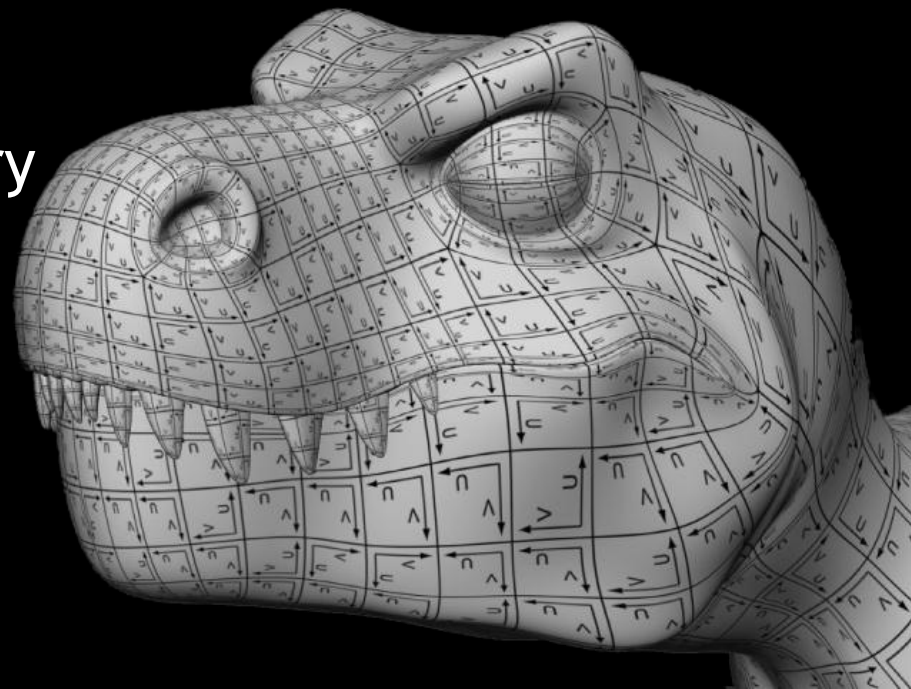
PTEX (2)

- Invented by Brent Burley at Walt Disney Animation Studios
  - Used in every animated film at Disney since 2007
    - 6 features and all shorts, plus everything in production now and for the foreseeable future
    - Used on ~100% of surfaces
  - Rapid adoption in DCC tools
  - Widespread usage throughout the film industry



PTEX - Benefits

- No UV unwraps
- Allow artists to work at any resolution they want
  - Perform an offline pass on assets to decide what to ship for each platform based on capabilities
- Ship a texture pack later for tail revenue
- Reduce your load times. And your memory footprint. Improve your visual fidelity.
- Reduce the cost of production's long pole—art.

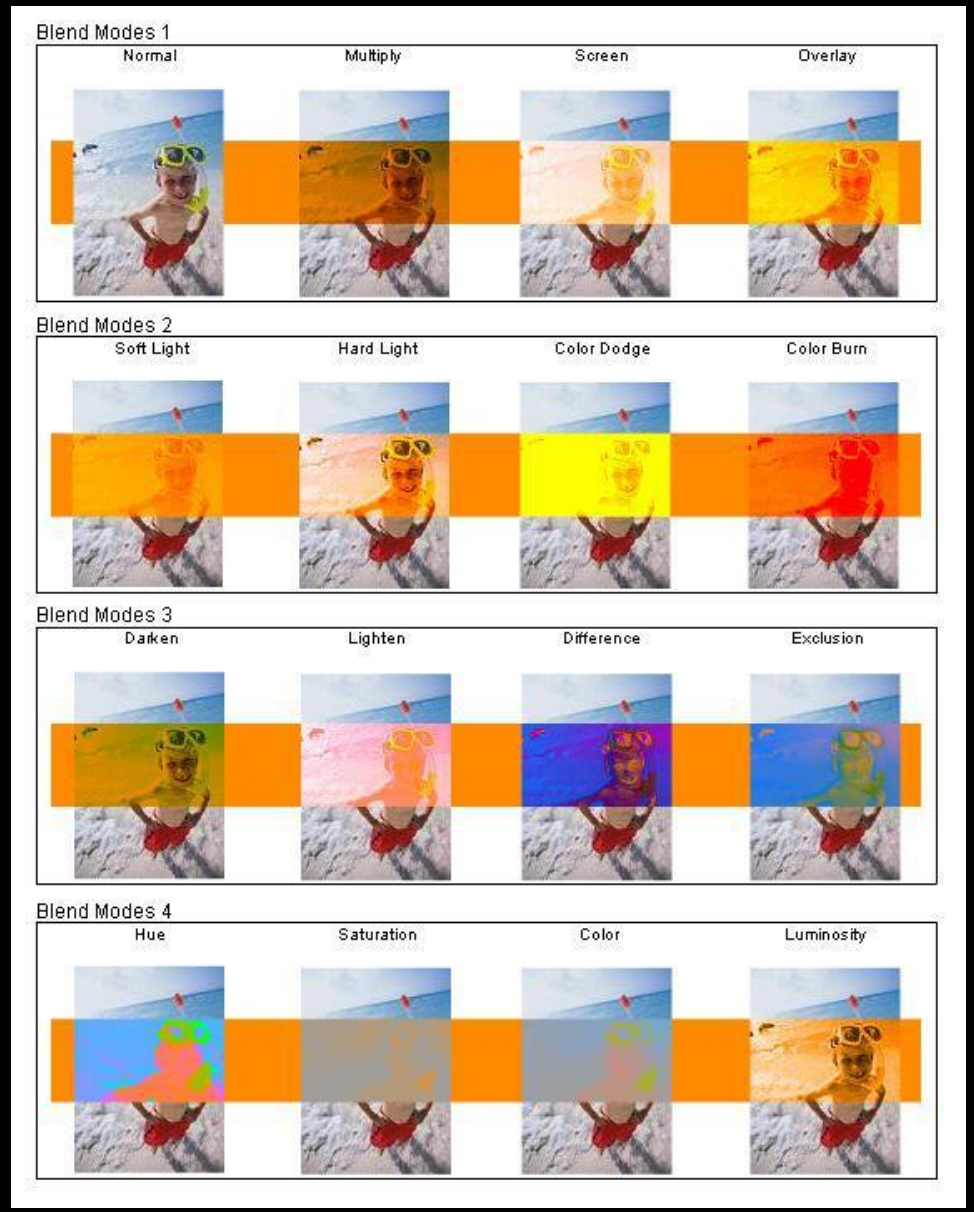


Advanced Blending

- Various standards support “blend modes”
  - Compositing standards
  - 2D standards
  - Path rendering standards
- Example standards
  - PostScript, PDF, SVG, OpenVG, XRender, Cairo, Skia, Mac’s Quartz 2D, Flash, Java 2D, Photoshop, Illustrator
- Blend modes have a theory distinct from 3D’s glBlendFunc, etc. functionality
  - glBlendFunc, etc. expose hardware operations
  - Blend modes based on sound compositing theory

Blend Mode Examples

- Normal
- Multiply
- Screen
- Overlay
- Soft Light
- Hard Light
- Color Dodge
- Color Burn
- Darken
- Lighten
- Difference
- Exclusion
- Hue
- Saturation
- Color
- Luminosity



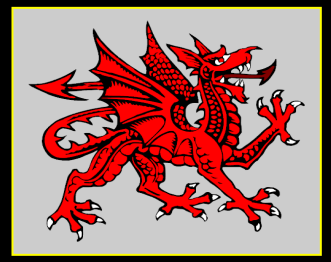
Conventional OpenGL blend modes support a small subset of those used in other environments.

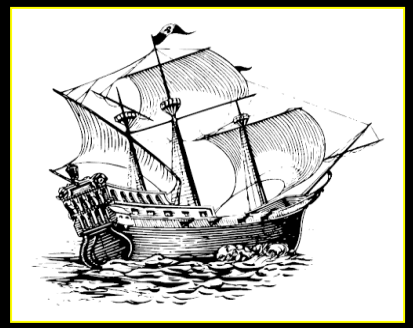
Target for Advanced Blending

- Market justification: 2D, compositing, and path rendering standards key to smart phones, tablets, and similar devices
  - Motivation is primarily for low-end, power-constrained devices
- Also part of content creation
  - Autodesk Mudbox, Adobe Illustrator, etc. all use blend modes as their vocabulary for compositing
- Power-efficient hardware support for “blend modes” exposed in NV_blend_equation_advanced

Path Rendering

- A rendering approach
  - Resolution-independent two-dimensional graphics
  - Occlusion & transparency depend on rendering order
    - So called “Painter’s Algorithm”
  - Basic primitive is a path to be filled or stroked
    - Path is a sequence of path commands
    - Commands are
      - moveto, lineto, curveto, arcto, closepath, etc.





No tricks

Every glyph is rendered from its outline; no render-to-texture

Not just zoomed & rotated, also perspective

Magnify & minify with no transitional pixelization or tile popping artifacts

Zoomed in

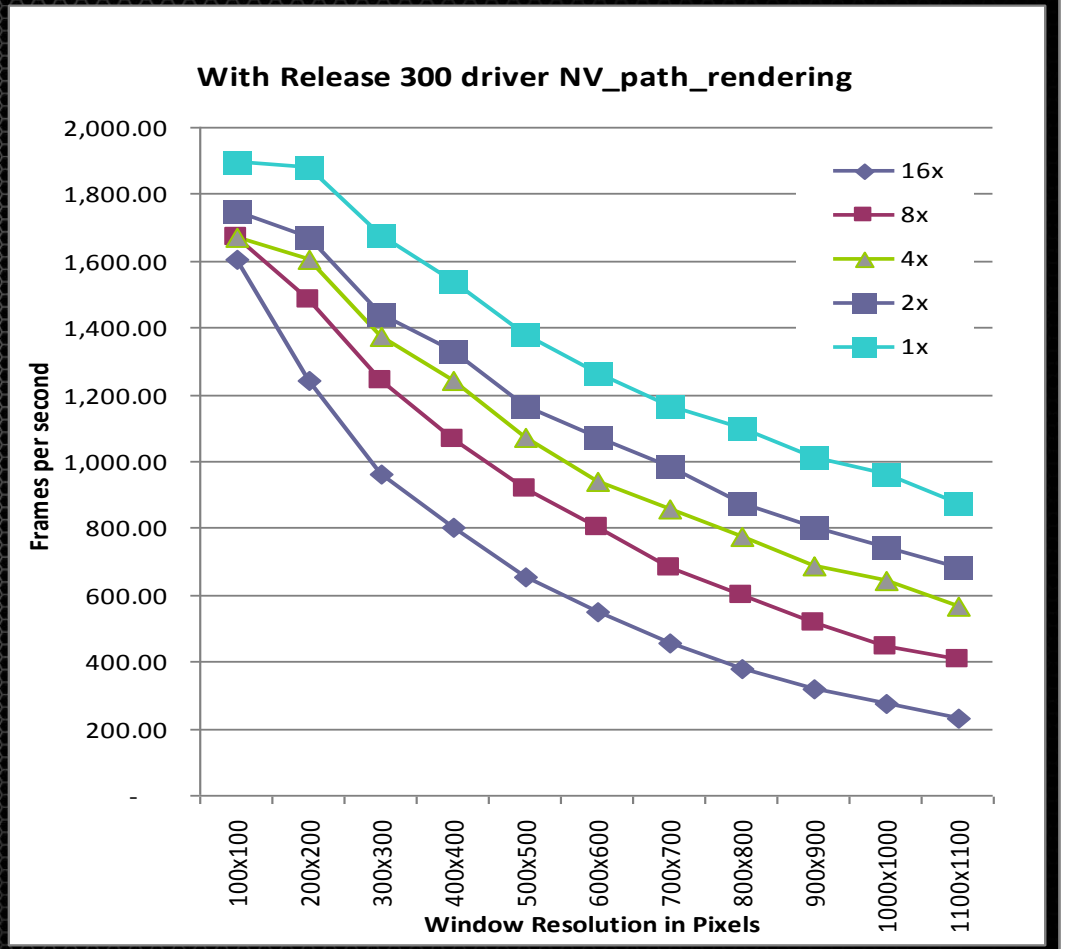
Web page

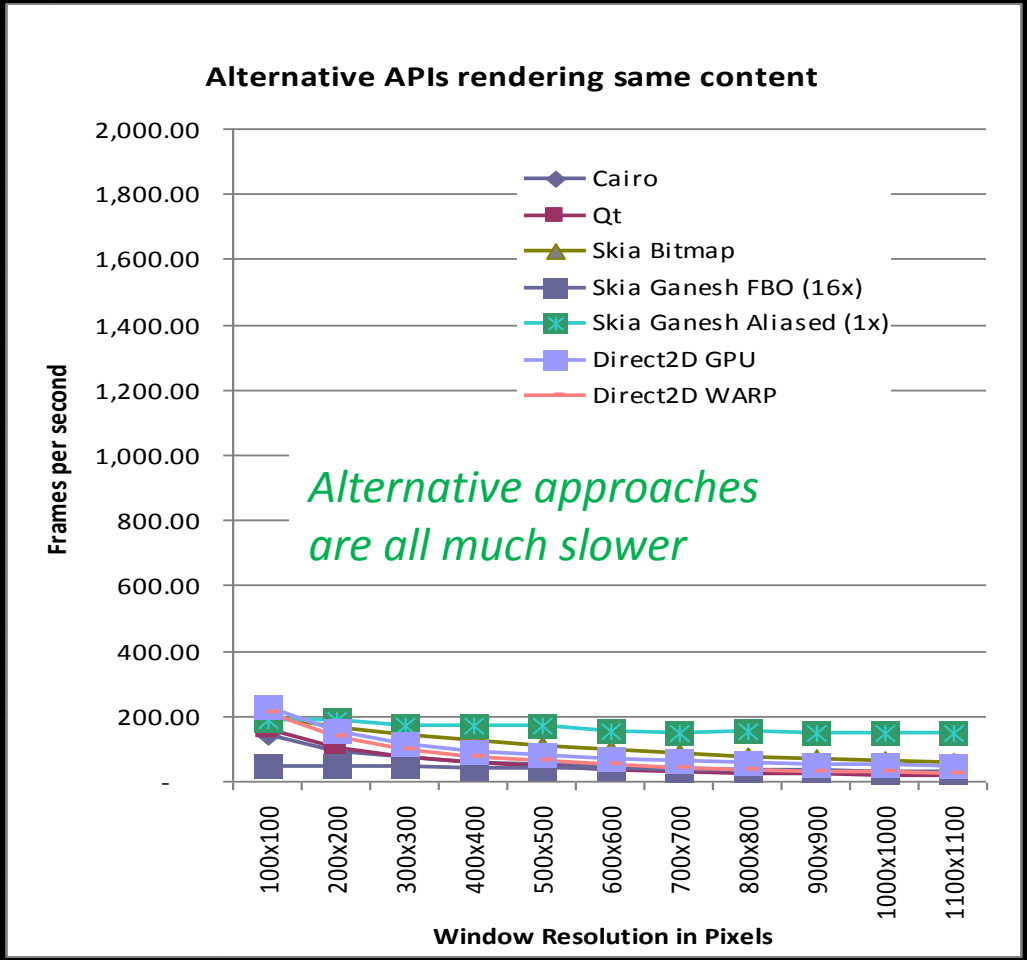
Control points of TrueType glyphs visualized

Projected

Configuration
GPU: GeForce 480 GTX (GF100)
CPU: Core i7 950 @ 3.07 GHz

NV_path_rendering Compared to Alternatives





Path Rendering Standards

Document Printing and Exchange

Open XML Paper (XPS)

Resolution-Independent Fonts

OpenType

TrueType

Immersive Web Experience

Flash

Scalable Vector Graphics

HTML 5

2D Graphics Programming Interfaces

Java 2D API

QtGui API

Mac OS X 2D API

Khronos API

Office Productivity Applications

Adobe Illustrator

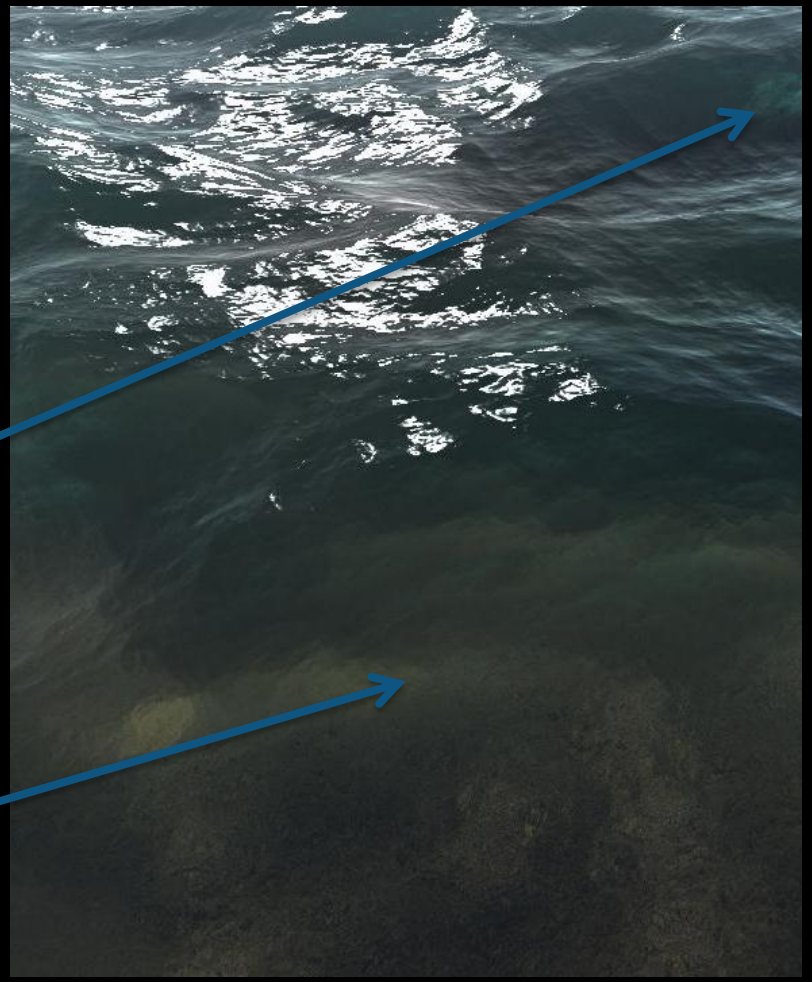
Inkscape Open Source

Ocean Simulation

- Simulation based on Tessendorf's algorithm (Phillips spectrum)
  - Used from Titanic ('97) to Life of Pi
  - Models a fully developed sea state
- Computed in the CPU using FFT/FFT^{-1} into a heightmap
  - FFT on heights
  - FFT on chop
  - 128x128 repeated

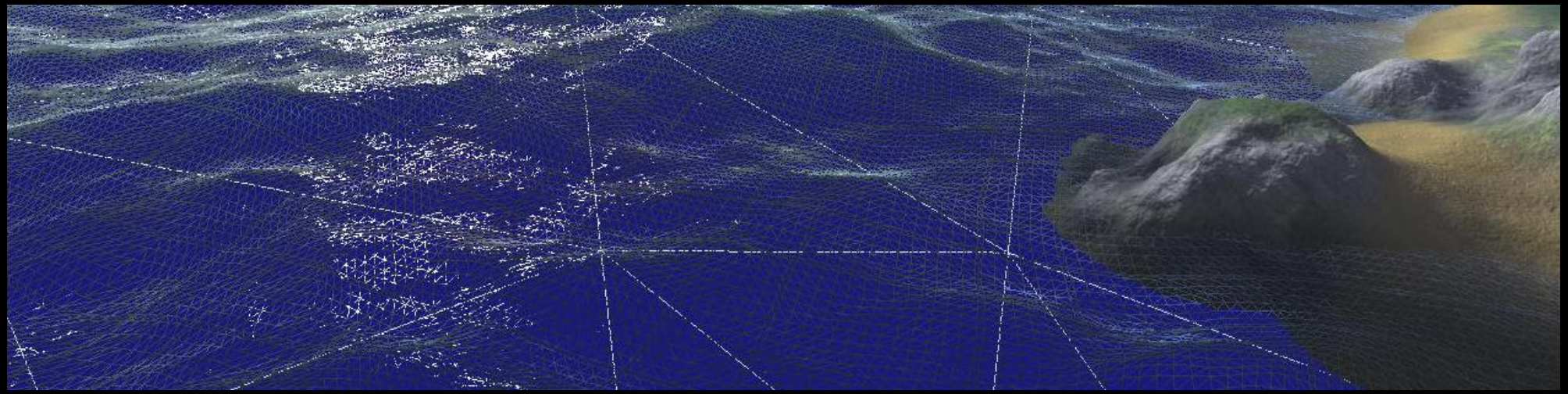
Ocean - Rendering

- Dynamic water surface
  - Per pixel reflection
  - Per pixel refraction
  - Fresnel
- Simple water light scattering
  - Shader math-gic
- Simple caustics simulation
  - 5x5 upward rays refracted and dotted against sun. Per Vertex after tess.

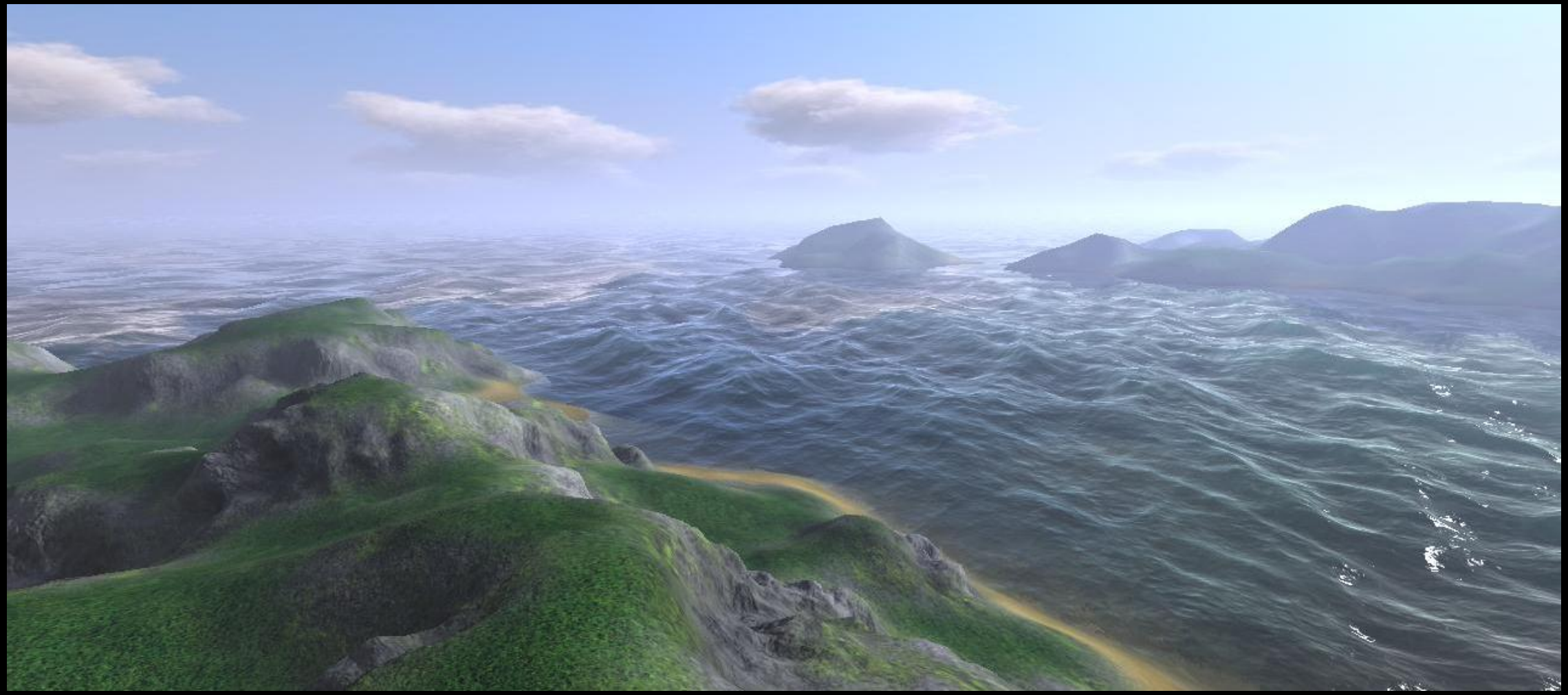


Ocean - Tessellation

- Tessellation
  - Used in terrain with displacement maps
  - On water surface using FFT
  - LOD control: dynamic tessellation factors base on k*1/distance to camera



Ocean – All together



Demo

API Futures Commentary

- API needs to develop better support for
  - High efficiency
  - Multiple threads
  - Clarity
- Modernize API with an eye toward preserving compatibilty
- Prematurely deprecated
  - Fixed function – great for efficiency, simplicity
  - Display lists – efficiency and multi-thread
    - Immediate mode – actually better with display lists…

Display Lists

- Display lists with client side vertex arrays were pointless
  - Had to source all the vertex data at compile time
  - Advent of VBO *should* have changed this to keep vertex data indirect
- Without display lists, API cannot express coherence well
  - Particularly frame-to-frame coherence
  - Need a way to express a static rendering sequence
- Display lists also natural for multi-threaded support
  - Compile lists on worker threads
  - Execute lists main render thread
  - Direct3D 11 attempted something similar, but it did not scale well

glCallList(list);

Multi-Draw Indirect

glMultiDrawIndirect();

- Another overhead reduction strategy
- Valuable, valid growth direction
  - But don’t allow state changes between the Draw calls like display lists do
- Makes things less obvious
  - Not simply back references in the log
  - But the draw calls can be constructed by the GPU shaders

Stable Abstractions

- Why do stable software abstractions exist?
  - Division of labor, allow competing implementations
  - Better for the ecosystem
- Graphics APIs have trended toward lower-level abstraction
  - Good to expose lower level abstractions
    - Can get better efficiency for some uses
    - However they are less portable
  - Higher level not bad
    - Allowed Reyes and Ray Tracing as back end for RenderMan
- Both high and low level important going forward

Random Thoughts

- Steal some good ideas from the web
  - State dump string in canonical form
    - Skip default state, alphabetize rest… (forward compatible)
    - Careful with “variable” names
      - Object names, slots
  - State setting via structured string – e.g. JSON
- Differential rendering
  - API that minimizes chatter without sacrificing clarity and efficiency
- Ecosystem needs better compiler tools
  - That run without a GL context, especially in mobile

Ecosystem

- Community of things that depend on each other
  - Success of individual components only part of the story
  - Prosperity depends on mix of components
- OpenGL ecosystem more bazaar than cathedral
  - Often difficult to monetize some vital components
- Some success stories
  - NSight
  - Apitrace
  - Regal

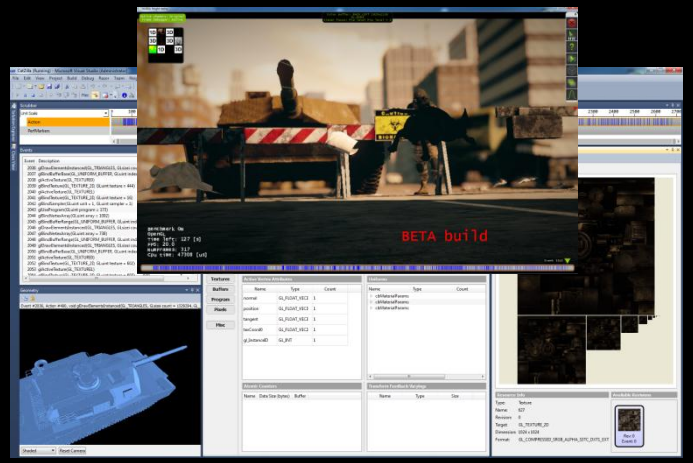
NVIDIA® Nsight™ Visual Studio Edition

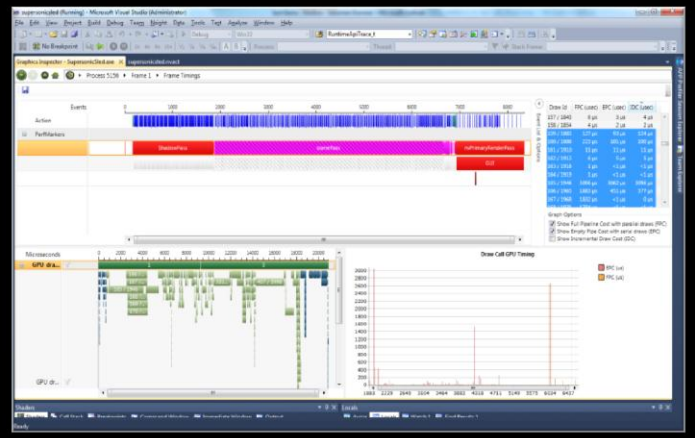
Visual Studio integrated development for GPU and CPU

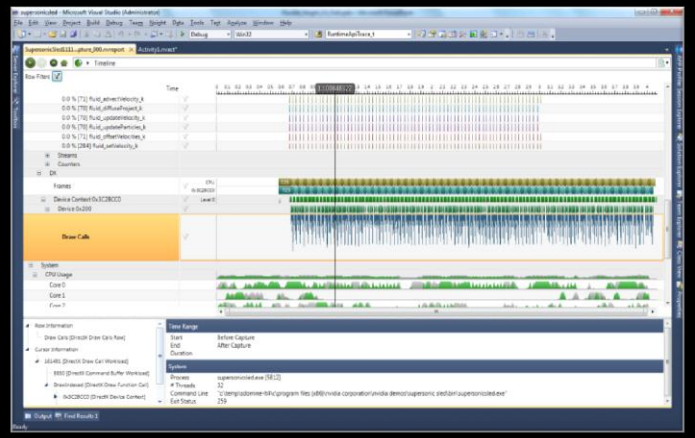
Build **Debug** **Profile**

Microsoft Visual Studio PARTNER

NVIDIA Nsight for Graphics Developers
OpenGL 4.2, Direct3D 9/11







Frame debugger - OpenGL 4.2, Direct3D 9/11

- Draw call and state inspection
- Frame capture and playback (source code gen D3D9/11)
- Nsight HUD for draw call scrubbing and inspection

HLSL and GLSL Shader debugger

- Native GPU shader debugging and GPU memory views
- Complex condition breakpoints and Pixel History
- Local Single GPU shader debugging

Frame profiler - OpenGL 4.2, Direct3D 9/11

- Automatic GPU bottleneck determination
- Draw call and Frame timings
- Direct3D Perf Markers and render state grouping/sorting

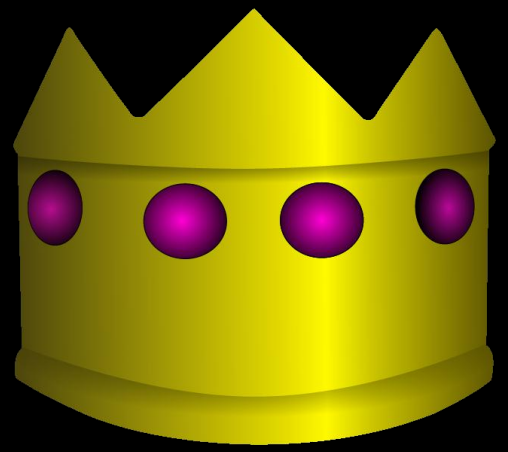
Application and system trace

- Inspect OpenGL and Direct3D activities / CPU and GPU
- Correlate threads, call stack, API calls, WDDM kernel queues and resulting GPU workloads
- Concurrent draw call execution and memory transfer trace

apitrace

- Community maintained project on GitHub
  - http://github.com/apitrace/apitrace
- Means of sharing repro command sequence
  - Good for functional bugs (repro app is often a real pain)
  - Analyzing best-case perf
  - Enables easy editing and variation experiments
  - Playback via glretrace not currently optimized for speed
  - Trace -> code probably most reliable for perf study
    - Though glretrace can be made *much* faster

Regal



- Community maintained project on GitHub
  - http://github.com/p3/regal
- Defragmented GL – write one portable back end
- Support compatibility in software
  - If driver does not support
  - Immediate mode & fixed function work again!
  - Large class of graphics apps for which this model is preferable for most rendering
- Broad support for emulated features – even on ES
  - DSA, VAO
  - Planned: SSO, path rendering, enhanced display lists

Regal (2)

- Ecosystem anchor point
  - Integrated http server for debug
    - Inspect or alter context state or objects, pause rendering
  - API log dumps
  - Apitrace integration
  - Shader, texture dump and replacement
- Open source – BSD license
  - On github (http://github.com/p3/regal)
  - Numerous contributors from all over
- Platform support
  - Windows, OS X, Linux, Android, iOS, NaCl

Questions?

- cass@nvidia.com
- Thanks to
  - Seth Williams, Bastiaan Aarts, John McDonald, Mark Kilgard, Miguel Sainz, Simon Green, Jeff Kiel, Jan Paul van Waveren

FIN