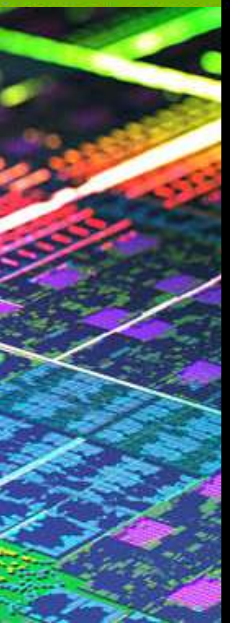
GPU TECHNOLOGY CONFERENCE



Making OpenGL Your Competitive Advantage

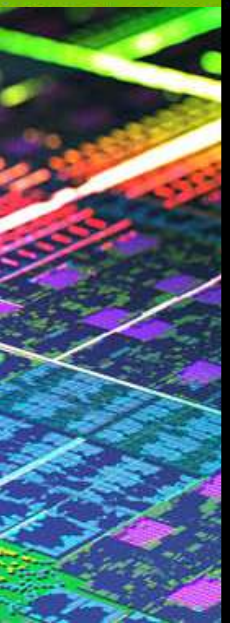
John McDonald

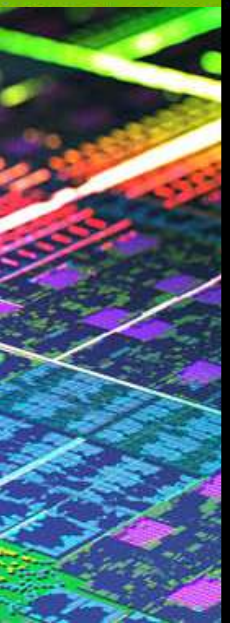
Senior Software Engineer, NVIDIA

gameworks.nvidia.com

Overview

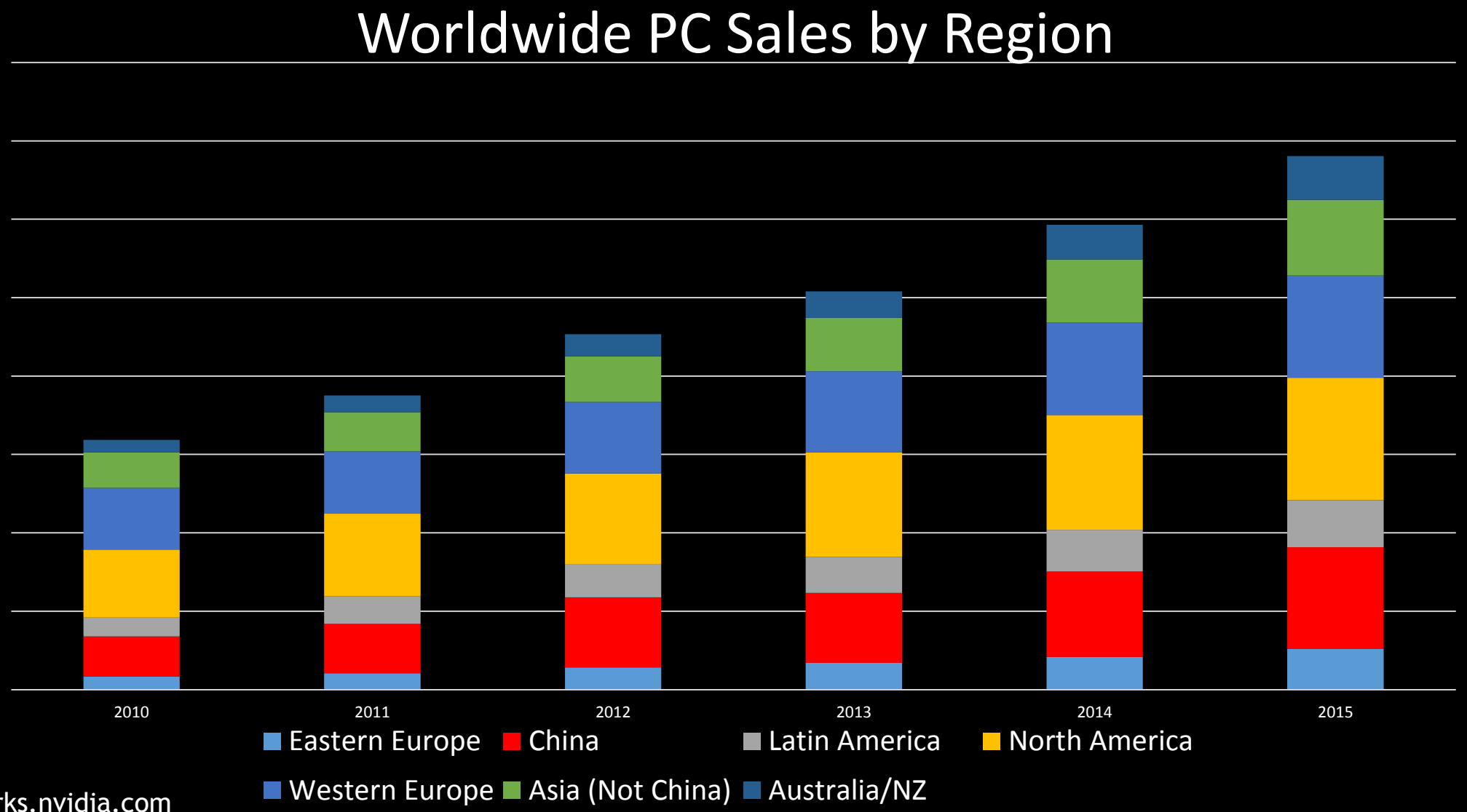
- Market Overview
- Why target OpenGL?
- Porting from Direct3D to OpenGL
- Porting to Mobile and Beyond





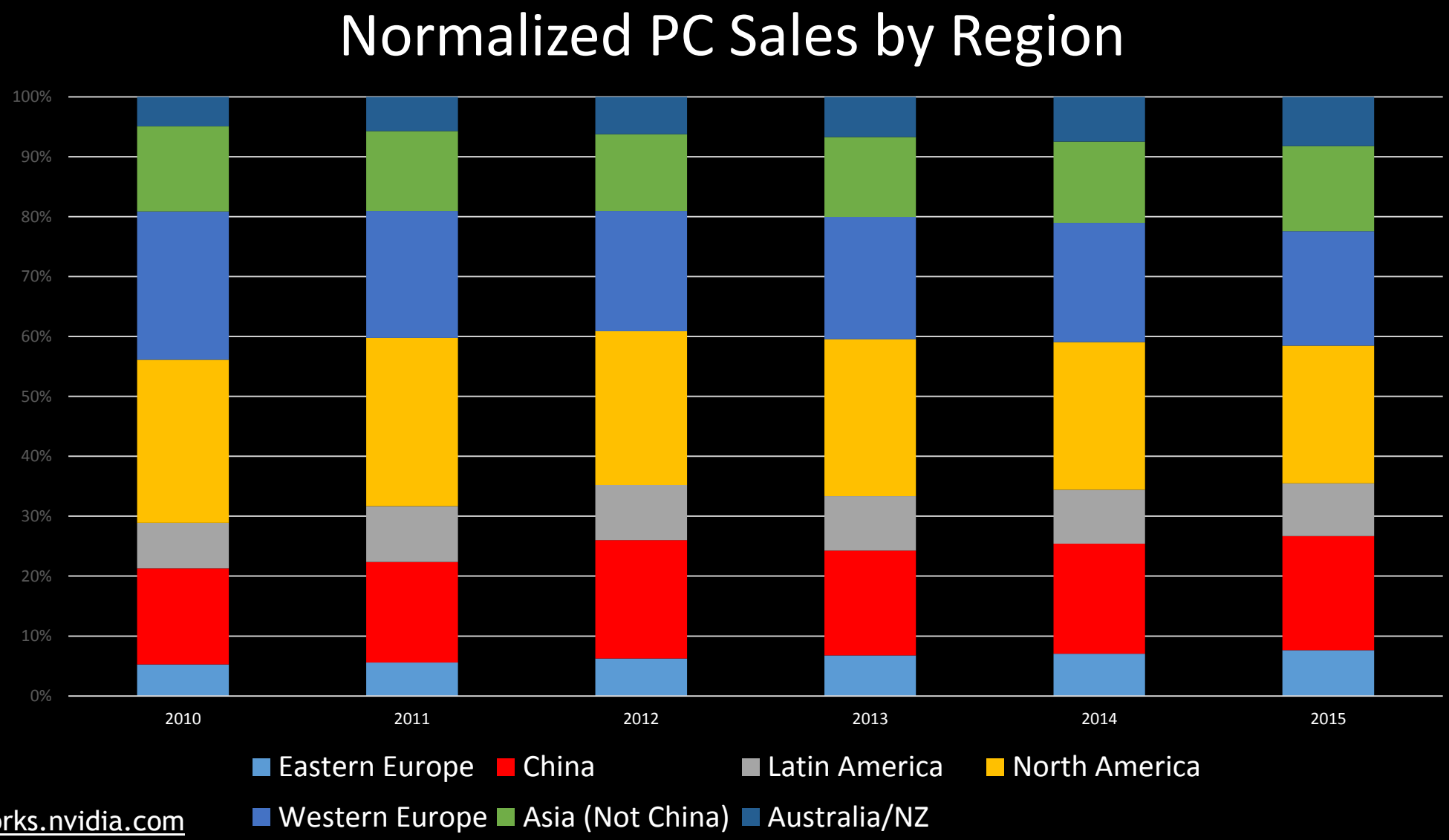
Market Overview

Market Overview

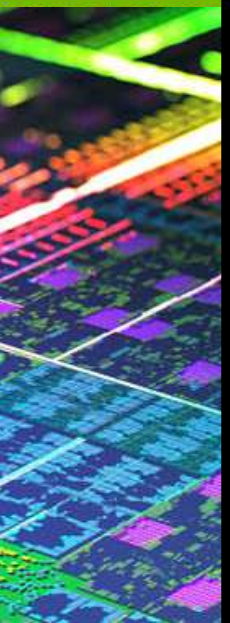


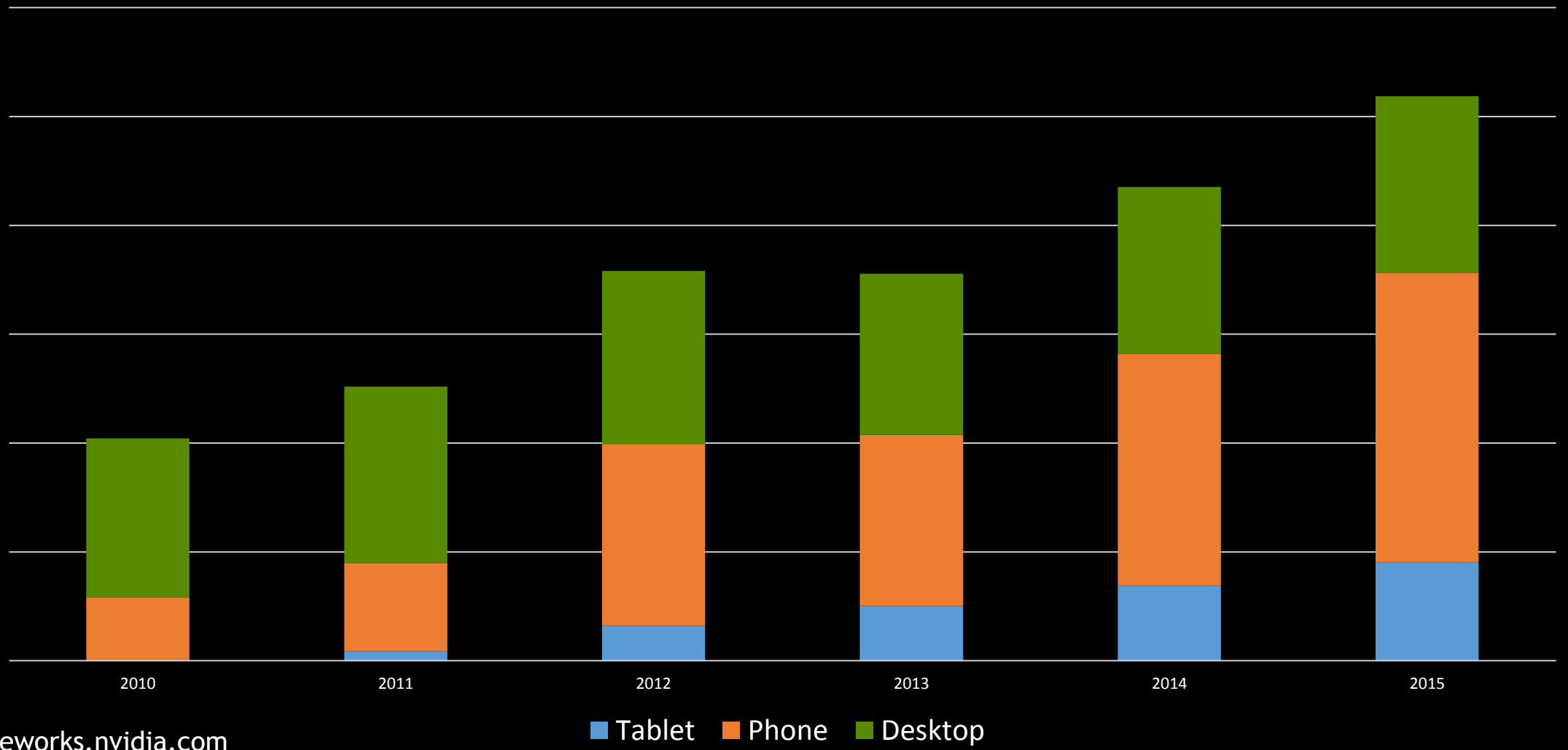
gameworks.nvidia.com

Market Overview (continued)

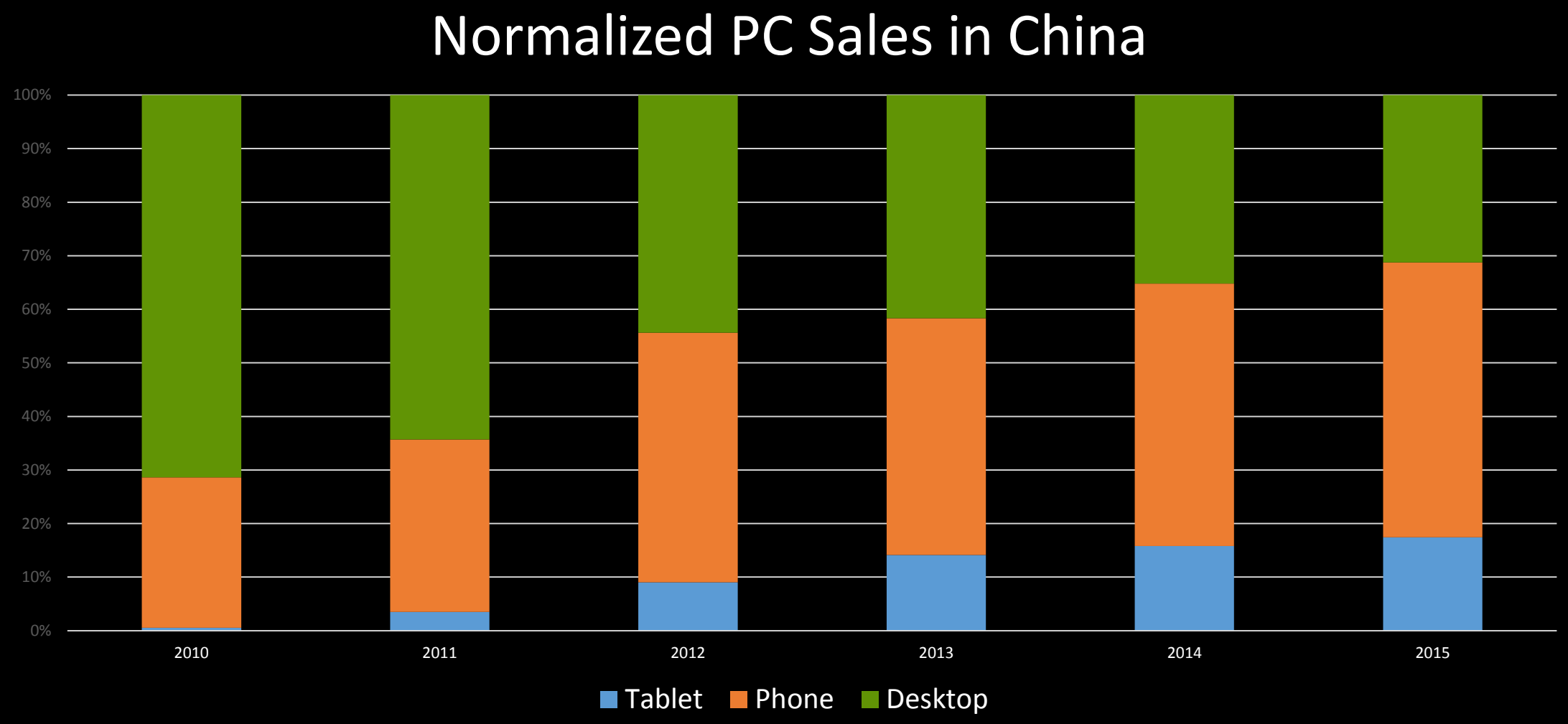


Market Breakdown in China

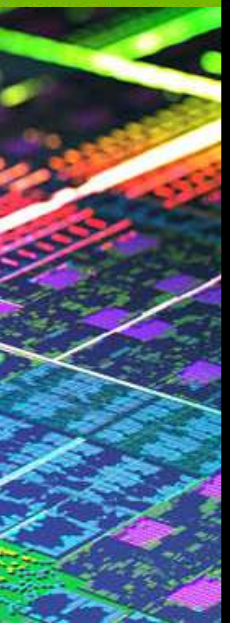




Market Breadown in China (continued)



gameworks.nvidia.com



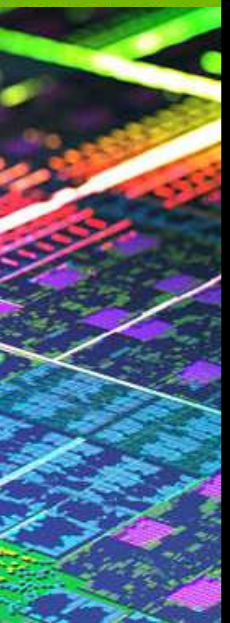
Why Target OpenGL?

Why target OpenGL?

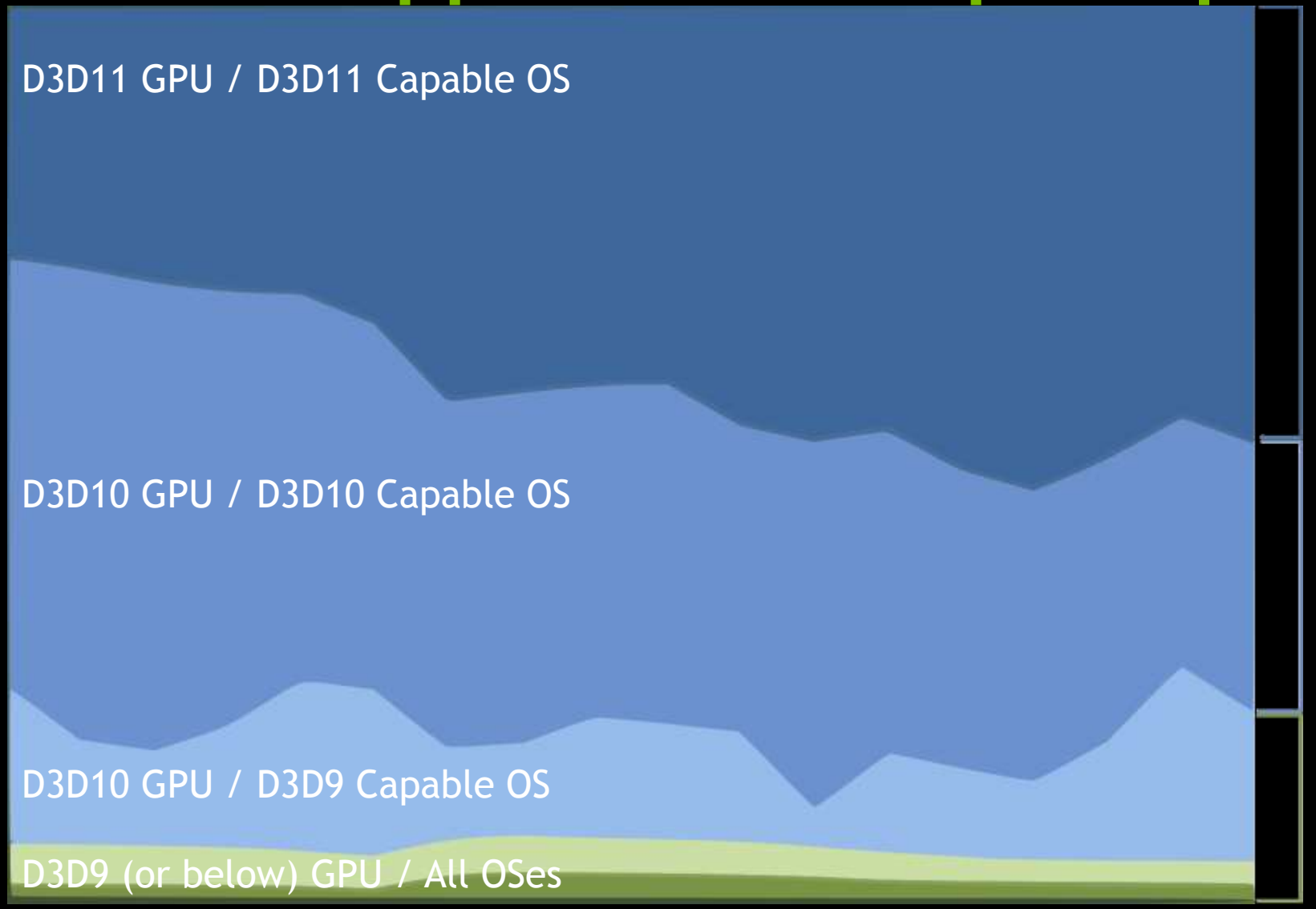
- GL exposes functionality by hardware capability—not OS.
- OpenGL is available on 100% of the devices from the previous charts
- Direct3D is available only on a (shrinking) fraction of them—Desktop/Laptop Devices
- China tends to have equivalent GPUs, but overwhelmingly still runs XP
  - OpenGL can allow DX10/DX11 (and beyond) features for all of those users

Why target OpenGL? – continued

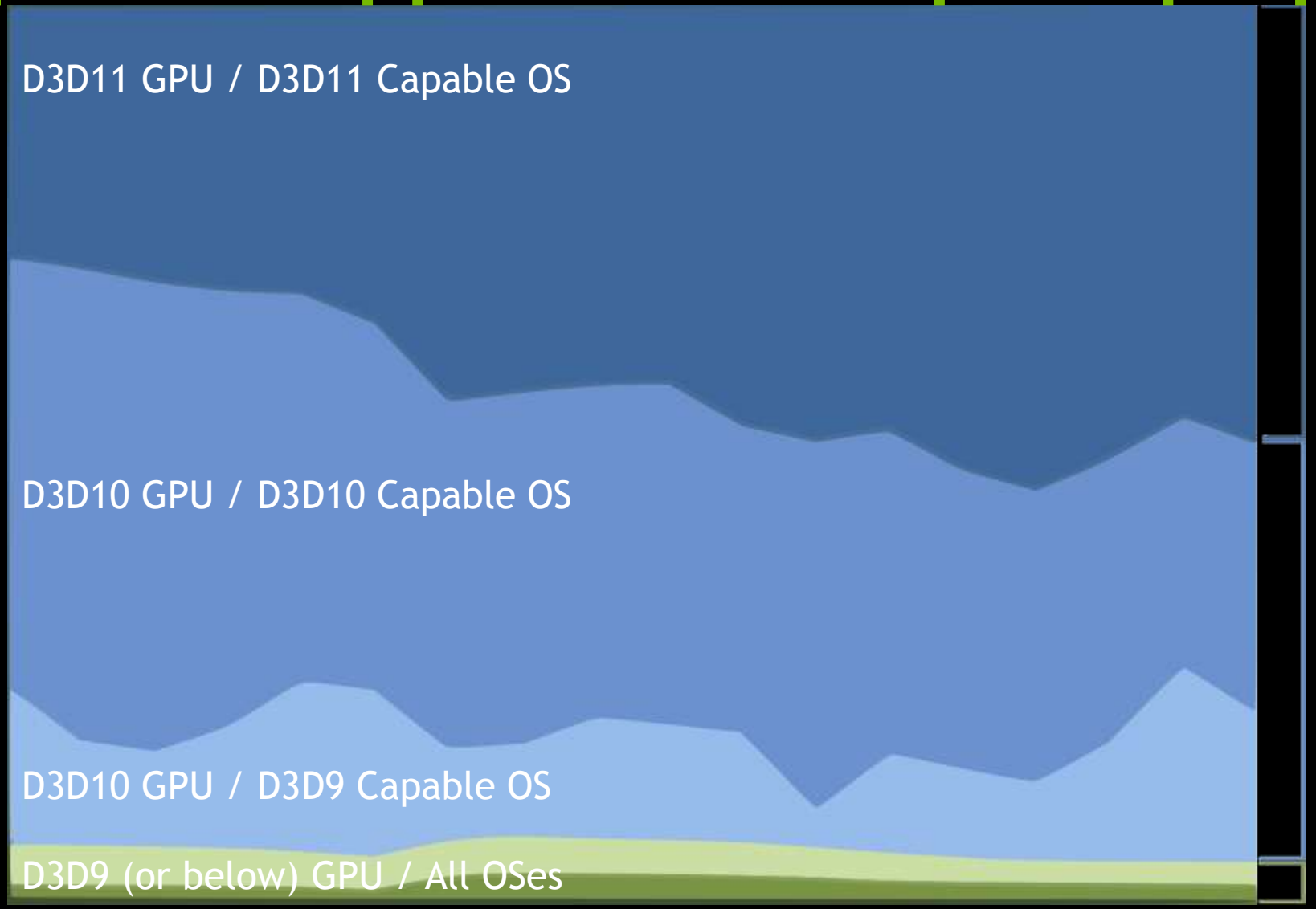
- Specifications are **public.**
- GL is owned by committee, membership is available to anyone with interest (and some, but not a lot, of $).
- GL can be extended quickly, starting with a single vendor.
- GL is extremely powerful



Direct3D Support – Desktop / Laptop

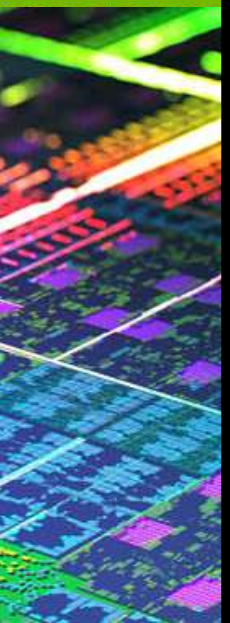


OpenGL Support – Desktop / Laptop



gameworks.nvidia.com

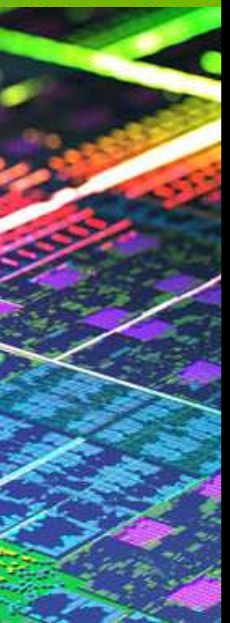
GPU TECHNOLOGY CONFERENCE



Porting from Direct3D to OpenGL

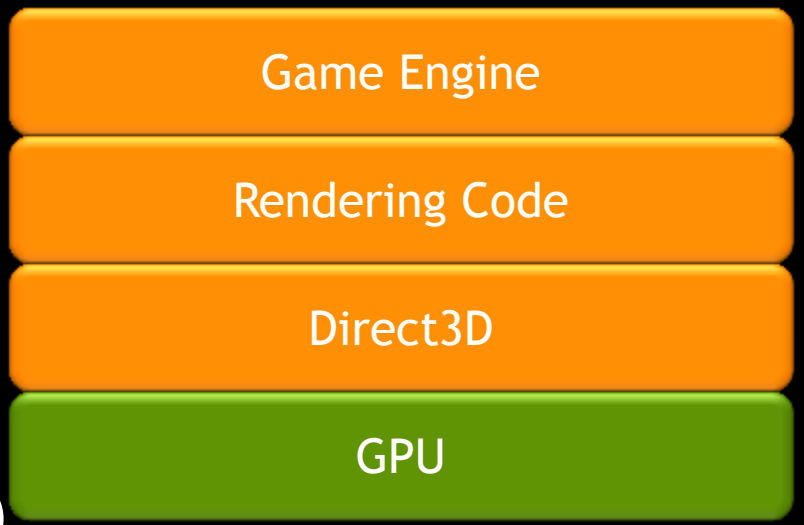
Which GL should you support?

- DX9 ≈ OpenGL 2
  - Shaders
- DX10 ≈ OpenGL 3
  - Streamlined API
  - Geometry Shaders
- DX11 ≈ OpenGL 4
  - Tessellation and Compute



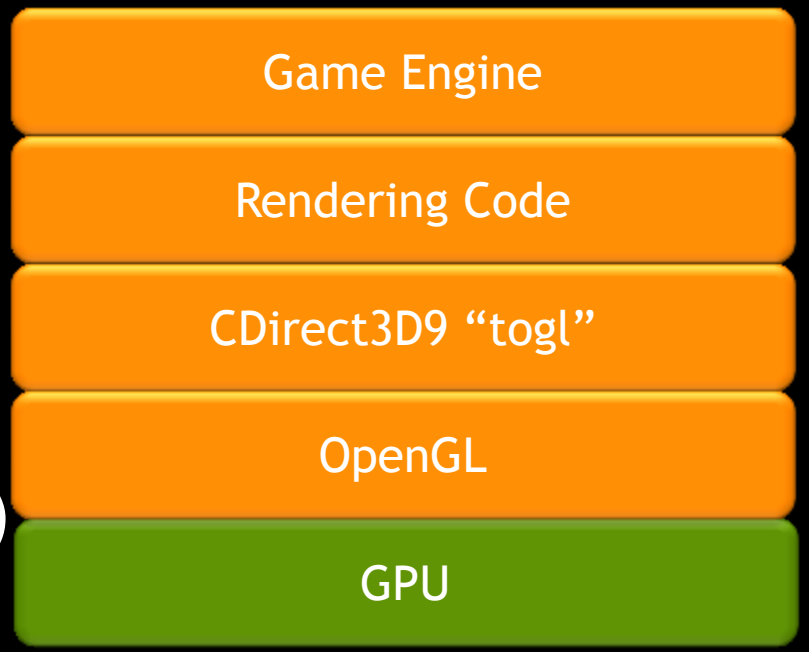
togl

- “to GL”
- A D3D9/10/11 implementation using OpenGL
- Originally implemented by Valve.
- In application, using a DLL.
- Engine code is overwhelmingly (99.9%) unaware of which API is being used—even rendering.



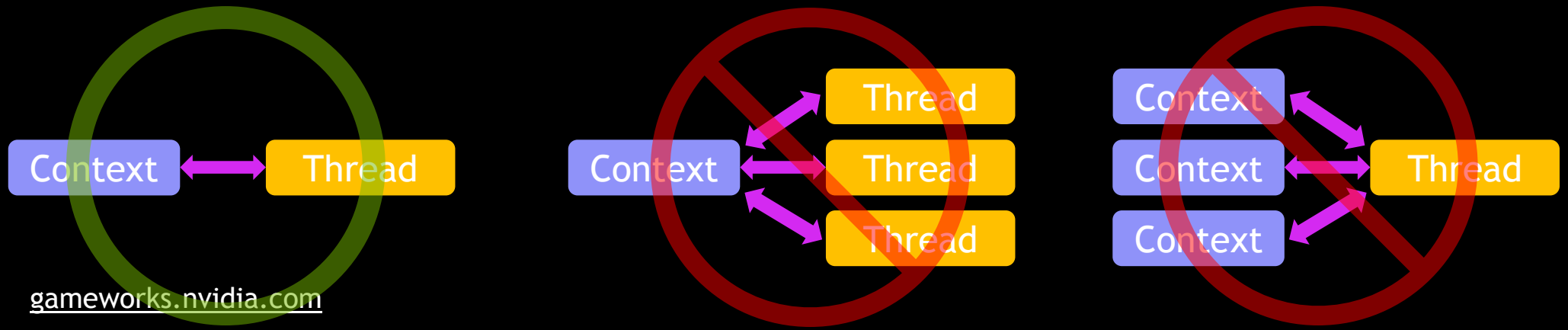
togl

- “to GL”
- A D3D9/10/11 implementation using OpenGL
- Originally implemented by Valve.
- In application, using a DLL.
- Engine code is overwhelmingly (99.9%) unaware of which API is being used—even rendering.
- Perf was a concern, but not a problem—this stack beats the shorter stack by ~20% in apples:apples testing.



GL / D3D differences

- GL has thread local data
  - A thread can have at most one Context current
  - A Context can be current on at most one thread
  - Calls into the GL from a thread that has no current Context are specified to “have no effect”
  - MakeCurrent affects relationship between current thread and a Context.



GL Pitfalls

- Several pitfalls along the way
  - Functional
    - Texture State
    - Handedness
    - Pixel Center Differences
  - Performance
    - MakeCurrent
    - Driver Serialization
- Vendor differences—be sure to test your code on multiple vendors

Texture State

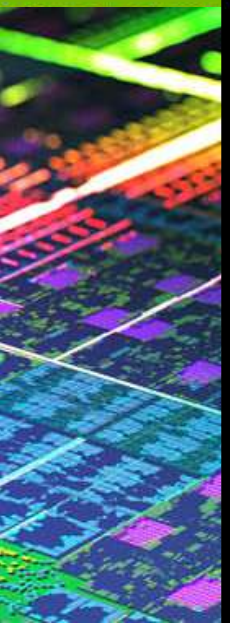
- By default, GL stores information about how to access a texture in a header that is directly tied to the texture.

* Not to scale

- This code doesn't do what you want:

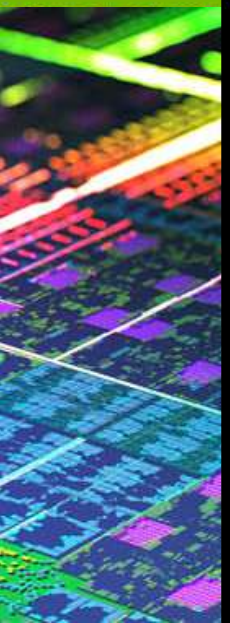
Texture State cont'd

```
glBindMultiTextureEXT( GL_TEXTURE0 + 0, 7 );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST );

glBindMultiTextureEXT( GL_TEXTURE0 + 1, 7 );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_LINEAR );

// Draw
```



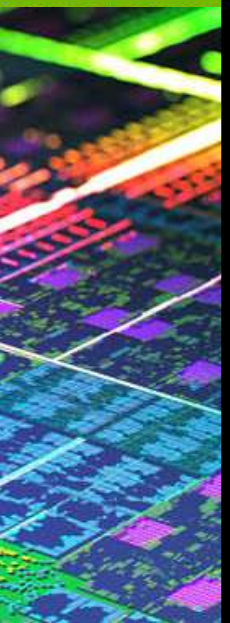
ARB_sampler_objects

- With ARB_sampler_objects, textures can now be accessed different ways through different units.
- Samplers take precedence over texture headers
- If sampler 0 is bound, the texture header will be read.
- No shader changes required
- http://www.opengl.org/registry/specs/ARB/sampler_objects.txt



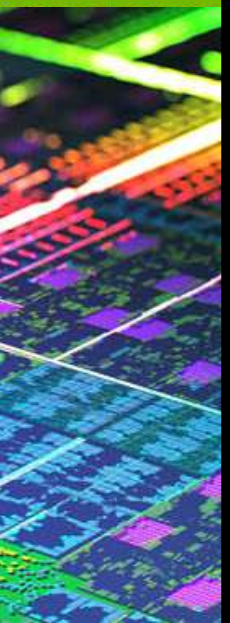
Using sampler objects

```
Gluint samplers[2];
glGenSamplers( 2, samplers );
glSamplerParameteri( samplers[0], GL_TEXTURE_MIN_FILTER,
                     GL_NEAREST );
glSamplerParameteri( samplers[1], GL_TEXTURE_MIN_FILTER,
                     GL_LINEAR );

glBindSampler( 0, samplers[0] );
glBindSampler( 1, samplers[1] );
glBindMultiTextureEXT( GL_TEXTURE0 + 0, 7 );
glBindMultiTextureEXT( GL_TEXTURE0 + 1, 7 );
// Draw
```



Other GL/D3D differences (cont'd)

- Handedness
  - D3D is left-handed everywhere, GL is right-handed everywhere
  - Texture origin is lower-left in GL (flip coordinates about v)
  - Consider rendering upside-down, flipping at the end.
- GLSL uses column-major matrices by default
  - Including when specifying constants/uniforms
- Pixel Centers
  - OpenGL matches D3D10+

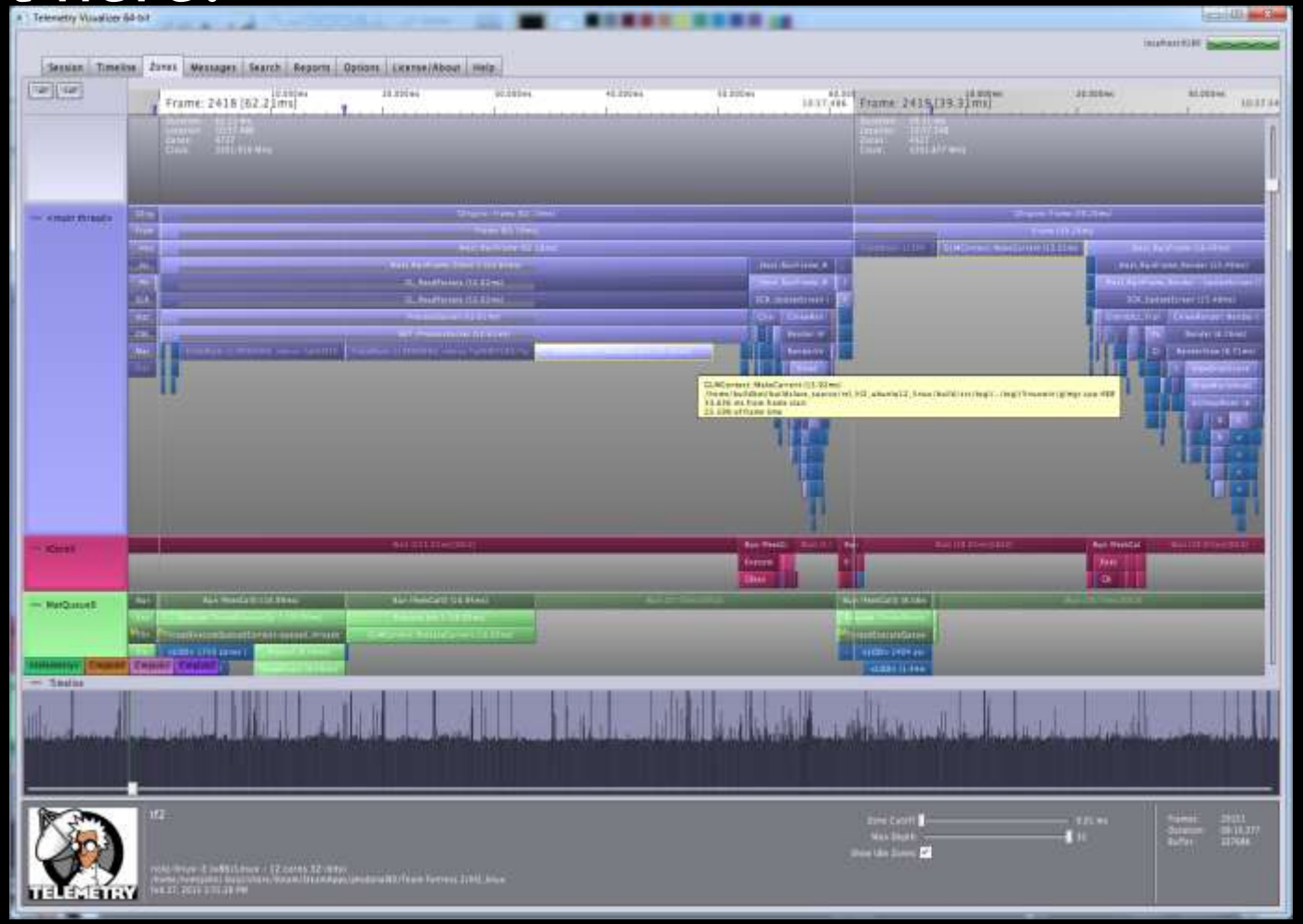


MakeCurrent issues

- Responsible for several bugs on Team Fortress 2
- Font rendering glitches (the thread creating text tries to update the texture page, but didn't own the context

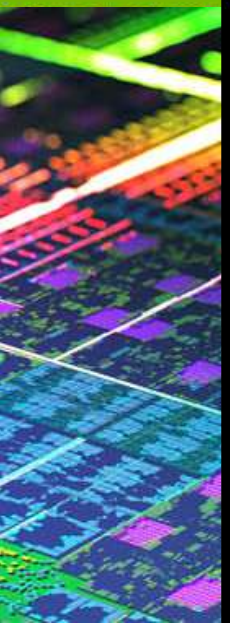
MakeCurrent Performance

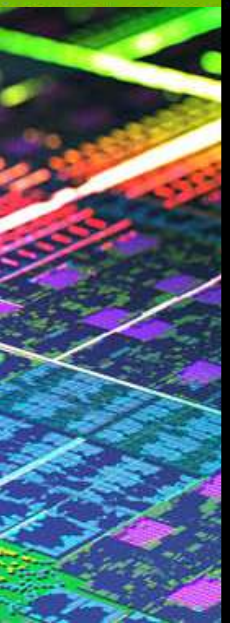
- Single-threaded is best here.
- MakeCurrent is *very* expensive—try not to call even once/twice per frame.



Driver Serialization

- Modern OpenGL drivers are dual-core / multithreaded
  - Your application speaks to a thin shim
  - The shim moves data over to another thread to prepare for submission
  - Similar to D3D
- Issuing certain calls causes the shim to need to flush all work, then synchronize with the server thread.
- This is very expensive



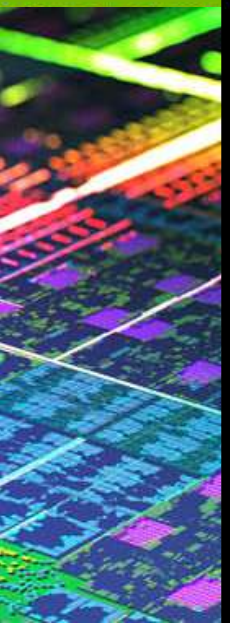


Known naughty functions

- glGet(…) – Most of these cause serialization; shadow state (just like D3D)
- glGetError - use ARB_debug_output!
- Functions that return a value
- Functions that copy a non-determinable amount of client memory, or determining the memory would be very hard

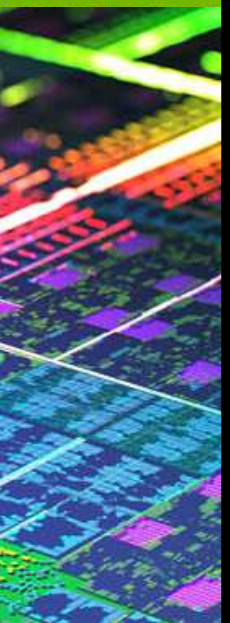
Detecting Driver Serialization

- ARB_debug_output to the rescue!
- Place a breakpoint in your callback, look up the callstack to see which call is causing the problem
- Message in ARB_debug_output to look for: “Synchronous call: stalling threaded optimizations.”



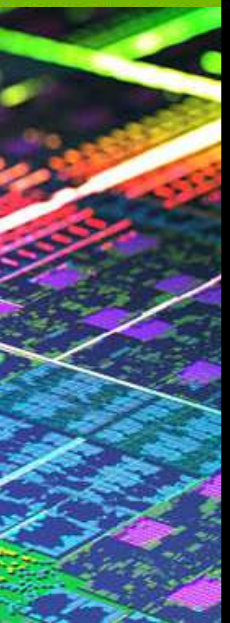
Shader Translation

- You have a pile of HLSL. You need to give GL GLSL.
  - ARB_vertex_program / ARB_fragment_program is a possible alternative, but only for DX9.
    - No ARB_tessellation_program



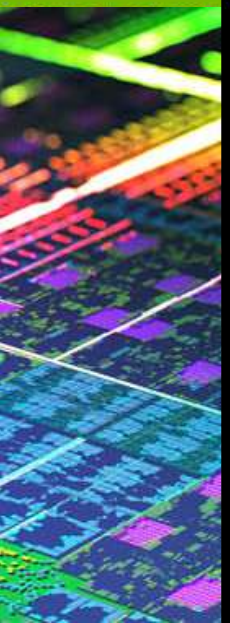
Shader Translation cont'd

- One approach: compile HLSL, translate the byte code to simple GLSL asm-like.
- Pro: One set of shaders goes public
- Pro: Can be *fast*
- Con: Can be *hard* to debug problems
- Con: Potentially slow fxc idioms end up in generated GLSL
- Con: Debugging requires heavy cognitive load



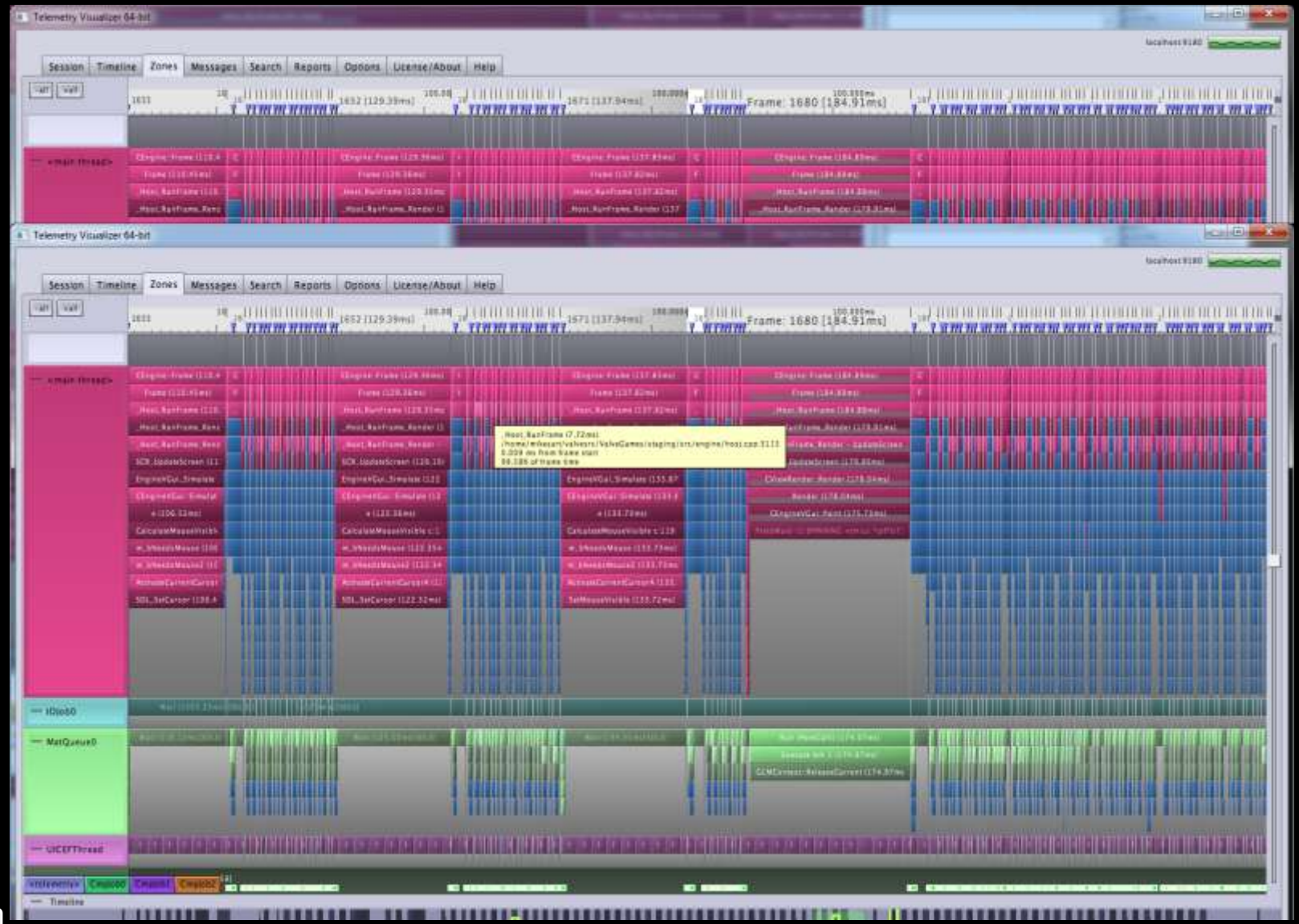
Other Translation Approaches

- Open Source Alternatives
  - HLSLCrossCompiler – D3D11 only (SM4/5)
  - MojoShader – SM1/2/3
    - Shipped in several games and engines, including Unreal Tournament 3, Unity.
- https://github.com/James-Jones/HLSLCrossCompiler
- http://icculus.org/mojoshader/



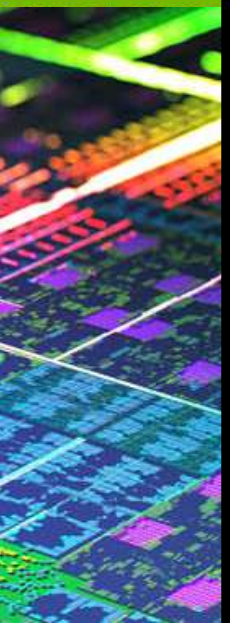
Performance tips

- Profile
- Profile
- Profile



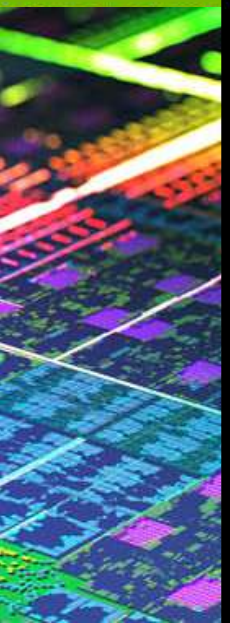
Performance tips – cont'd

- For best performance, you *will* have to write vendor-specific code in some cases.
- But you were probably doing this anyways
- And now behavior is specified in a public specification.



GL Debugging and Perf Tools

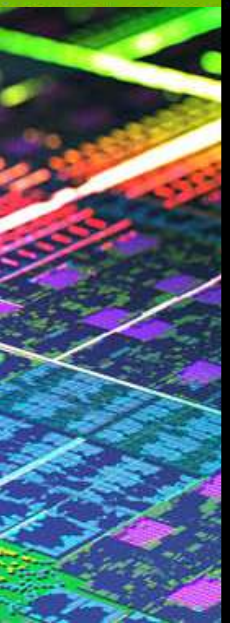
- NVIDIA Nsight supports GL 4.2 Core.
  - With some specific extensions
  - More extensions / features coming!
- PerfStudio and gDEBugger
- CodeXL
- Apitrace
  - Open Source api tracing tool—has scaling issues which Valve is working to fix.



GL Debugging Tricks

- Compare D3D to GL images
- Keep them both working on the same platform
- Bonus points: Have the game running on two machines, broadcast inputs to both, compare images in realtime.

GPU TECHNOLOGY CONFERENCE

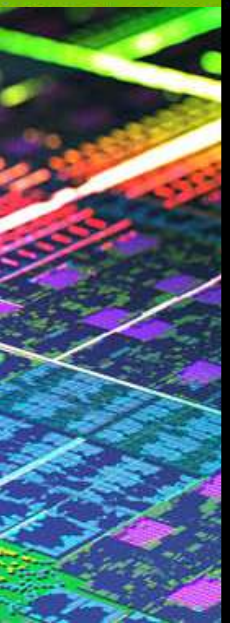


Porting to Mobile and Beyond

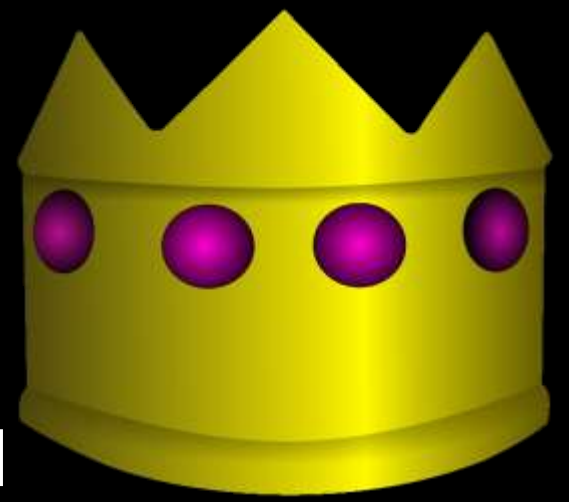
gameworks.nvidia.com

GL Fragmentation

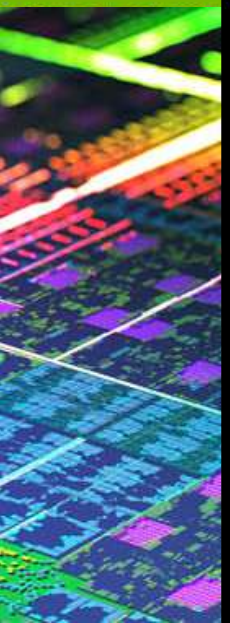
- OpenGL 4.x (Desktop)
- OpenGL 2.1 (OSX Compatability)
- OpenGL 3.2 (OSX Core)
- OpenGL ES (Mobile)
- WebGL



Regal

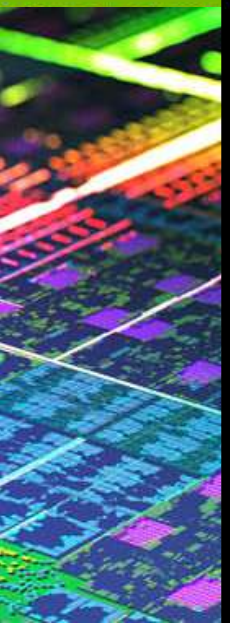


- Community maintained project on GitHub
  - http://github.com/p3/regal
- Defragmented GL – write one portable back end
- Support compatibility in software
  - If driver does not support
  - Immediate mode & fixed function work again!
  - Large class of graphics apps for which this model is preferable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



Regal - continued

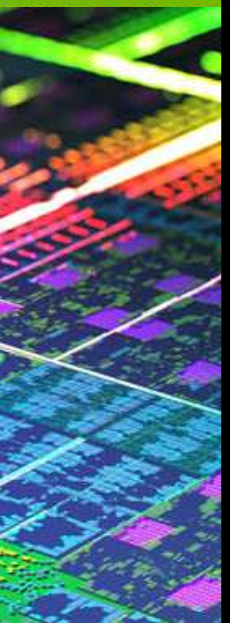
- Ecosystem foundation
  - Integrated http server for debug
    - Inspect or alter context state or objects, pause rendering
  - API log dumps
  - Apitrace integration
  - Shader, texture dump and replacement
- Open source – BSD license
  - On github (http://github.com/p3/regal)
  - Numerous contributors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



Questions?

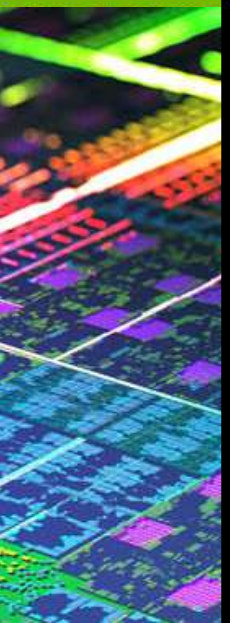
- jmcdonald at nvidia dot com

- For *very* detailed info on D3D->GL, check the talk “Porting Source to Linux: Valve’s Lessons Learned”



Appendix

- Some other GL gotchas/helpers



Useful extensions

- EXT_direct_state_access
- EXT_swap_interval (and EXT_swap_control_tear)
- ARB_debug_output
- ARB_texture_storage
- ARB_sampler_objects