Porting Source to Linux

Valve's Lessons Learned

NVIDIA

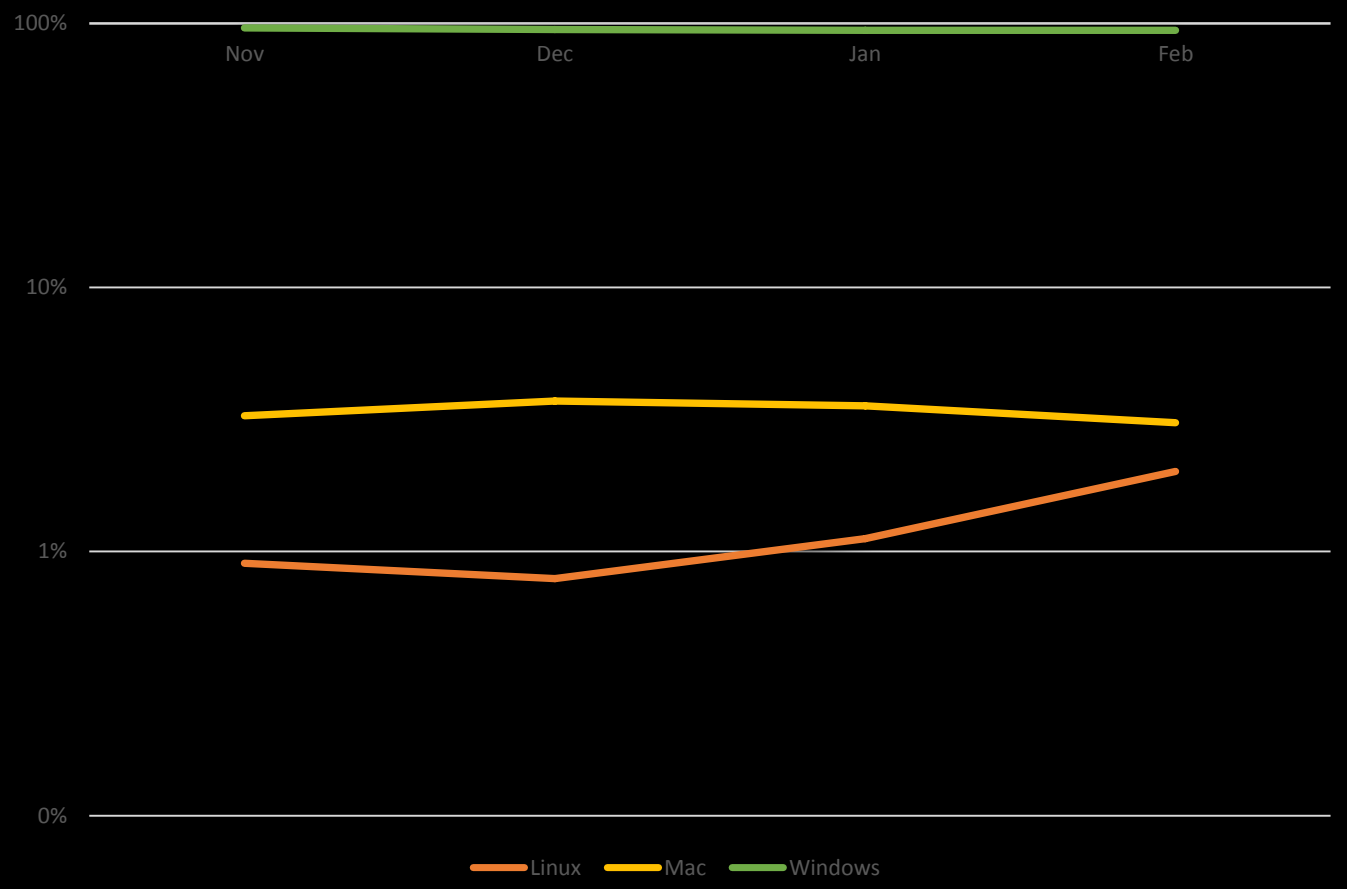
Overview

- Who is this talk for?
- Why port?
- Windows->Linux
- Linux Tools
- Direct3D->OpenGL

Why port?

Why port?

- Linux is open
- Linux (for gaming) is growing, and quickly
- Stepping stone to mobile
- Performance
- Steam for Linux



%	December	January	February
Windows	94.79	94.56	94.09
Mac	3.71	3.56	3.07
Linux	0.79	1.12	2.01

Why port? – cont'd

- GL exposes functionality by hardware capability—not OS.
- 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 port? – cont'd

- 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

Windows->Linux

Windowing issues

- Consider SDL!
- Handles all cross-platform windowing issues, including on mobile OSes.
- Tight C implementation—everything you need, nothing you don't.
- Used for all Valve ports, and Linux Steam

http://www.libsdl.org/

Filesystem issues

- Linux filesystems are case-sensitive
- Windows is not
- Not a big issue for deployment (because everyone ships packs of some sort)
- But an issue during development, with loose files
- Solution 1: Slam all assets to lower case, including directories, then tolower all file lookups (only adjust below root)
- Solution 2: Build file cache, look for similarly named files

Other issues

- Bad Defines
  - E.g. Assuming that LINUX meant DEDICATED_SERVER
- Locale issues
  - locale can break printf/scanf round-tripping
  - Solution: Set locale to en_US.utf8, handle internationalization internally
  - One problem: Not everyone has en_US.utf8—so pop up a warning in that case.

More Other Issues

- Font
  - Consider freetype and fontconfig
  - Still work determining how to translate font sizes to linux
- RDTSC (use clock_gettime(CLOCK_MONOTONIC) instead)
- Raw Mouse input
  - Great, but some window managers also grab the keyboard
  - This breaks alt-tab. Grr.
- Multi-monitor is less polished than Windows
  - SDL mostly handles this for you

Linux Tools

Steam Linux Runtime (and SDK)

- Runtime provides binary compatibility across many Linux distros for end users
- SDK has everything you'll need to target the runtime in one convenient set of packages
- Debug versions available, too
  - For both developers and end users
- http://media.steampowered.com/client/runtime/steam-runtime-sdk_latest.tar.xz
- https://github.com/ValveSoftware/steam-runtime

Tools - CPU Compilation/Debug

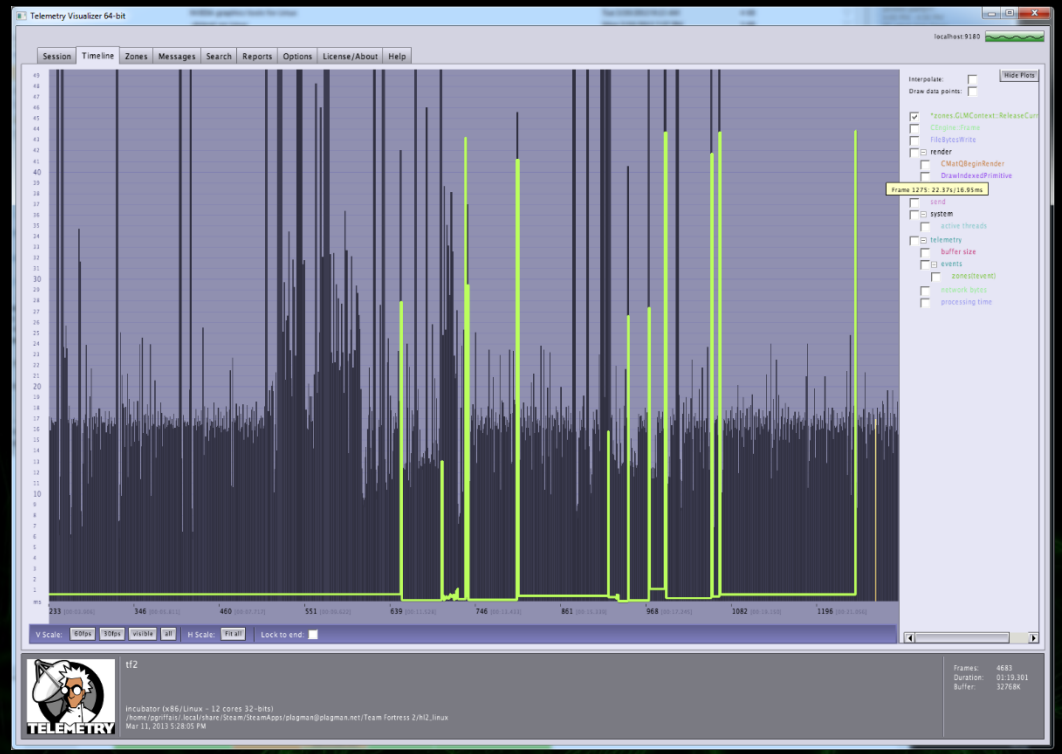
- Compilation / Debug
  - gcc - compilation
  - gdb - debugging from 1970
  - cgdb - debugging from 2000
  - ldd - dumpbin for linux
  - nm - for symbol information
  - objdump - disassembler / binary details
  - readelf - more details about binaries
  - make - no, really
- We'll talk about GPU Debug tools later

Tools – CPU Perf analysis

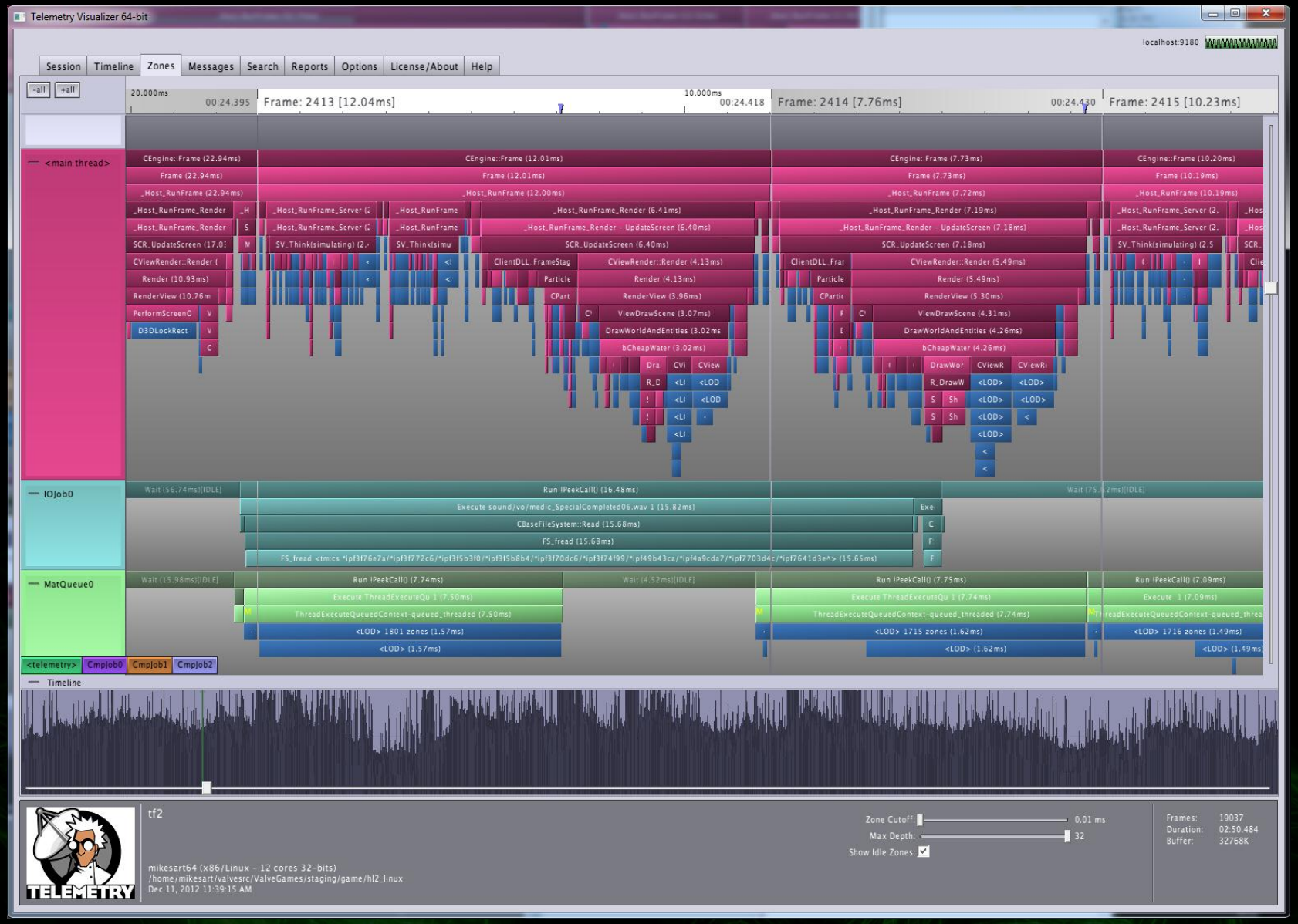
- perf – free sampling profiler
- vtune – Intel's tool works on Linux, too!
- Telemetry – You're using this already, right?
- Again, we'll talk about GPU perf tools later

Telemetry

- Telemetry is a performance visualization system on steroids, created by RAD Game Tools.
- Very low overhead (so you can leave it on all through development)
- Quickly identify long frames
- Then dig into guts of that frame



Telemetry Details

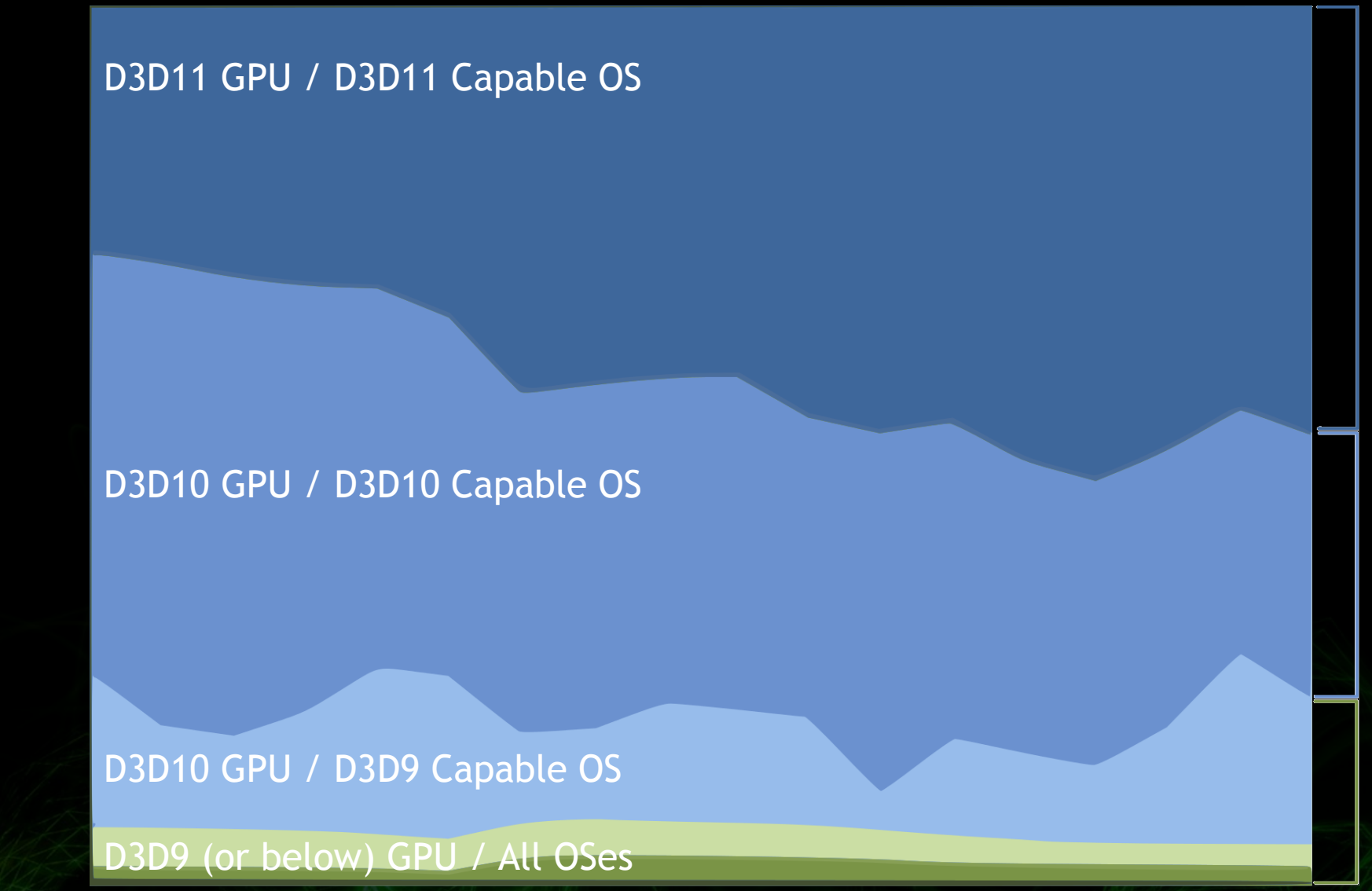


Direct3D -> OpenGL

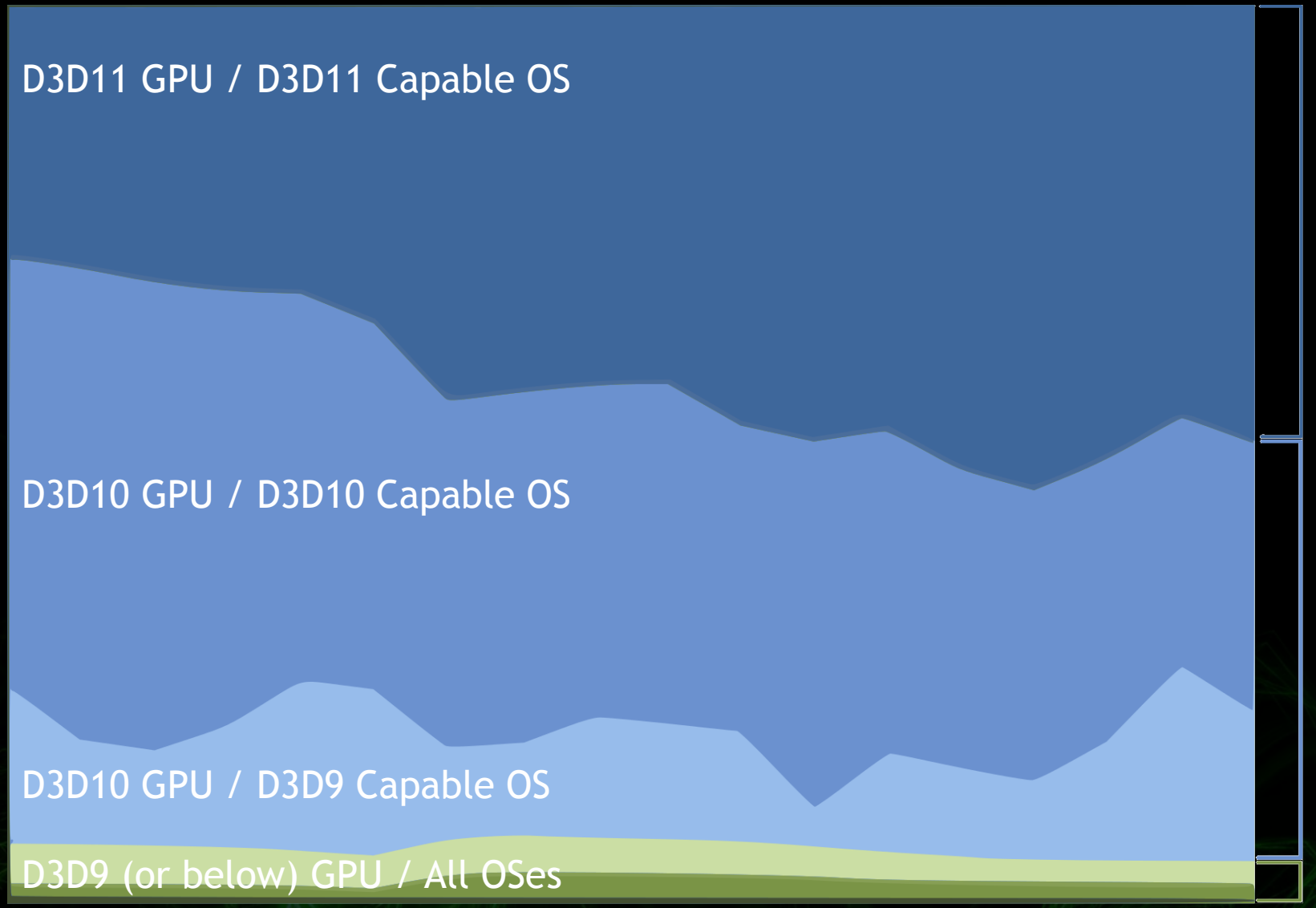
Which GL should you support?

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

Direct3D Support

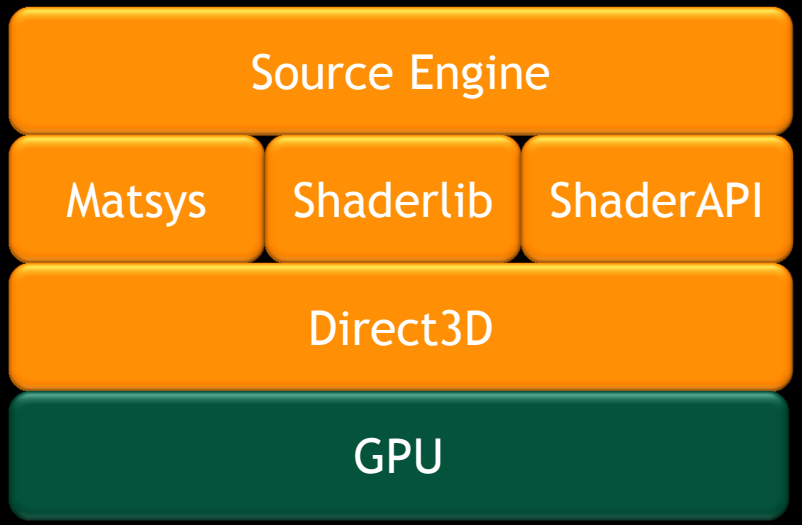


OpenGL Support



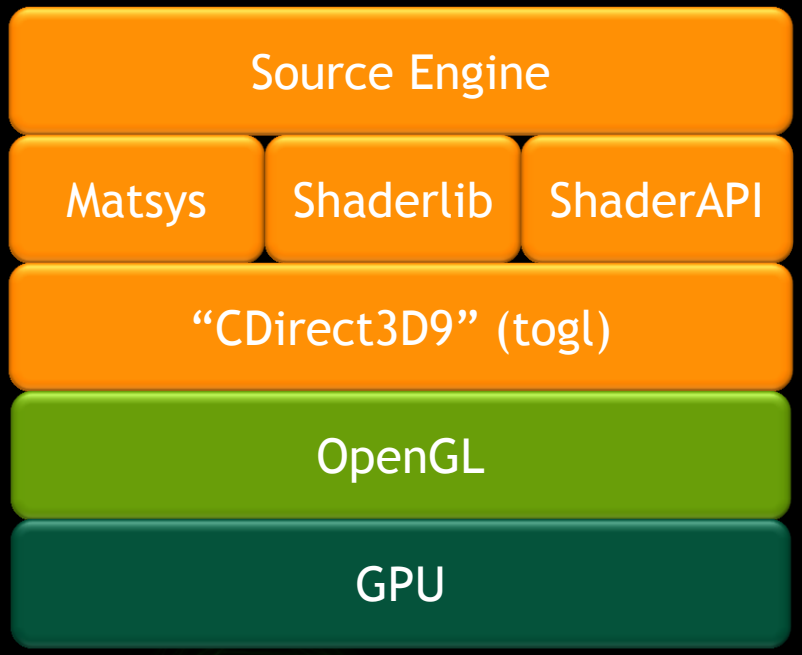
togl

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



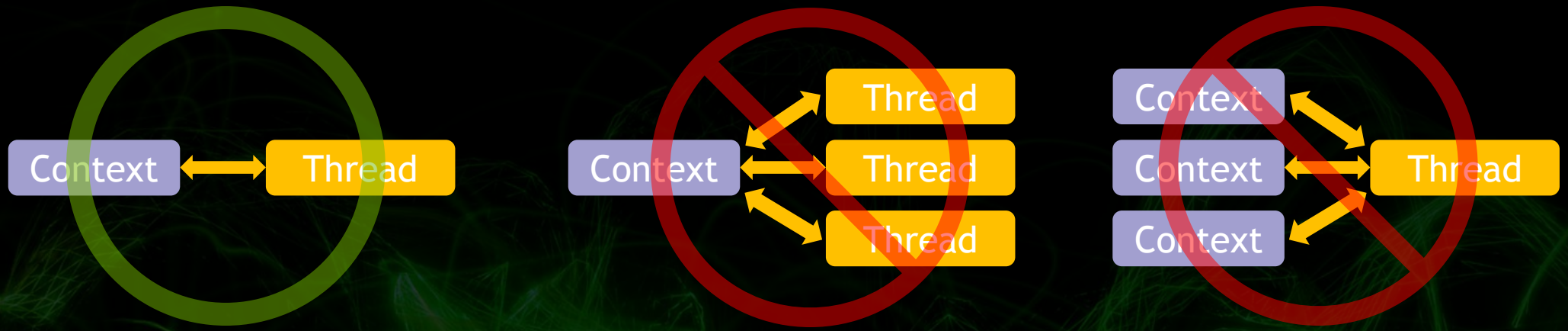
- Perf was a concern, but not a problem—this stack beats the shorter stack by ~20% in apples:apples testing.

togl: Major pieces

- Textures, VBs, IBs
- Device Creation
  - D3DCAPS9 (yuck!)
- Shaders
  - togl handles this, too!

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 / D3D differences

- GL is C based, objects referenced by handle
  - Many functions don’t take a handle at all, act on currently selected object
  - Handle is usually a GLuint.
- GL supports extensions
- GL is chatty, but shockingly efficient.
  - Do not judge a piece of code by the number of function calls.
  - Profile, profile, profile!
- GL doesn’t suffer lost devices

GL extensions

- NV|AMD|APPLE extensions are vendor specific (but may still be supported cross-vendor)
  - Ex: NV_bindless_texture
- EXT are multi-vendor specs
  - Ex: EXT_separate_shader_objects
- ARB are ARB-approved
  - Ex: ARB_multitexture
- Core extensions
  - A core feature from a later GL version exposed as an extension to an earlier GL version.
- Platform extensions (WGL|GLX|AGL|EGL)
- Consider GLEW or similar to wrangle extensions
- http://www.opengl.org/wiki/OpenGL_Extension

GL tricks

- When googling for GL functions, enums, etc, search with and without the leading gl or GL_
- Reading specs will make you more powerful than you can possibly imagine
- Don’t like where GL is heading? Join Khronos Group and shape your destiny.

GL objects

- GL has many objects: textures, buffers, FBOs, etc.
- Current object reference unit is selected using a selector, then the object is bound.
- Modifications then apply to the currently bound object.
- Most object types have a default object 0.

GL Object Model (cont'd)

```
// Select texture unit 3.
glActiveTexture( GL_TEXTURE0 + 3 );

// bind texture object 7, which is a 2D texture.
glBindTexture( GL_TEXTURE_2D, 7 );

// Texture object 7 will now use nearest filtering for
// minification.
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST );
```

Core vs Compatibility

- Some IHVs assert Core will be faster
- No actual driver implementations have demonstrated this
- Tools starting with Core, but will add Compat features as needed.
- Some extensions / behaviors are outlawed by Core.
- Recommendation: Use what you need.

Useful extensions

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

EXT_direct_state_access

- Common functions take an object name directly, no binding needed for manipulation.
- Code is easier to read, less switching needed.
- More similar to D3D usage patterns
- http://www.opengl.org/registry/specs/EXT/direct_state_access.txt

EXT_direct_state_access cont'd

```
GLint curTex;
glGetIntegeriv( GL_TEXTURE_BINDING_2D, &curTex);
glBindTexture( GL_TEXTURE_2D, 7 );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glBindTexture( GL_TEXTURE_2D, curTex );
```

- Becomes

```
glTextureParameteriEXT( 7, GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                        GL_NEAREST );
```

DSA when DSA is unavailable

- DSA is a driver-only extension—hardware is irrelevant.
- Write client code that assumes DSA
- Provide your own DSA function(s) when DSA is unavailable
- When resolving functions, use a pointer to your function if extension is unavailable.

```
void myTextureParameteriEXT( GLuint texture, GLenum target,
                             GLenum pname, GLint param)
{
  GLint curTex;
  glGetIntegeriv( GL_TEXTURE_BINDING_2D, &curTex );
  glBindTexture( target, texture );
  glTexParameteri( target, pname, param );
  glBindTexture( target, curTex );
}
```

EXT_swap_interval

- Vsync, but can be changed dynamically at any time.
- Actually a WGL/GLX extension.

```
wglSwapInterval(1); // Enable VSYNC
wglSwapInterval(0); // Disable VSYNC
```

- http://www.opengl.org/wiki/Swap_Interval
- http://www.opengl.org/registry/specs/EXT/wgl_swap_control.txt
- http://www.opengl.org/registry/specs/EXT/swap_control.txt

EXT_swap_control_tear

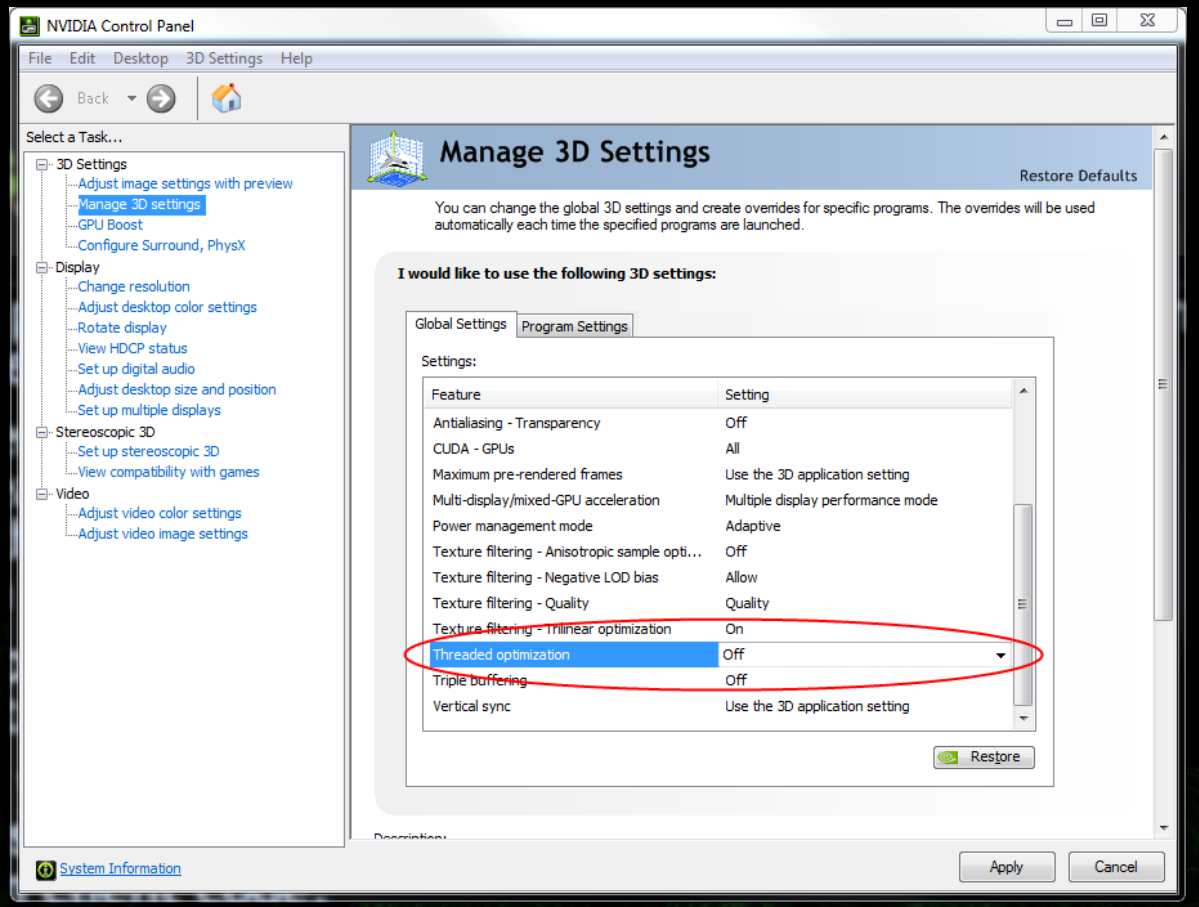
- XBox-style Swap-tear for the PC.
  - Requested by John Carmack.
    - First driver support a few weeks later
    - All vendors supported within a few months

```
wglSwapIntervalEXT(-1); // Try to vsync, but tear if late!
```

- http://www.opengl.org/registry/specs/EXT/wgl_swap_control_tear.txt
- http://www.opengl.org/registry/specs/EXT/glx_swap_control_tear.txt

ARB_debug_output

- You provide a callback when the driver detects an error—get fed a message.
- When the driver is in single-threaded mode, you can see all the way back into your own stack.
- Supports fine-grained message control.
- And you can insert your own messages in the error stream from client code.
- Quality varies by vendor, but getting better.



ARB_debug_output cont'd

```
// Our simple callback
void APIENTRY myErrorCallback( GLenum _source,
    GLenum _type, GLuint _id, GLenum _severity,
    GLsizei _length, const char* _message,
    void* _userParam)
{
  printf("%s\n", _message);
}

// First check for GL_ARB_debug_output, then...
glDebugMessageCallbackARB( myErrorCallback, NULL );
glEnable( GL_DEBUG_OUTPUT );
```

More Useful GL Extensions

- NVX_gpu_memory_info / GL_ATI_meminfo
  - Get memory info about the underlying GPU
- GL_GREMEDY_string_marker
  - D3DPERF-equivalent
- GL_ARB_vertex_array_bgra
  - better matches UINT-expectations of D3D
- GL_APPLE_client_storage / GL_APPLE_texture_range
  - Not for linux, but useful for Mac.

GL Pitfalls

- Several pitfalls along the way
  - Functional
    - Texture State
    - Handedness
    - Texture origin differences
    - Pixel Center Convention (D3D9->GL only)
  - Performance
    - MakeCurrent issues
    - 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.

- This code doesn't do what you want:

* Not to scale

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)

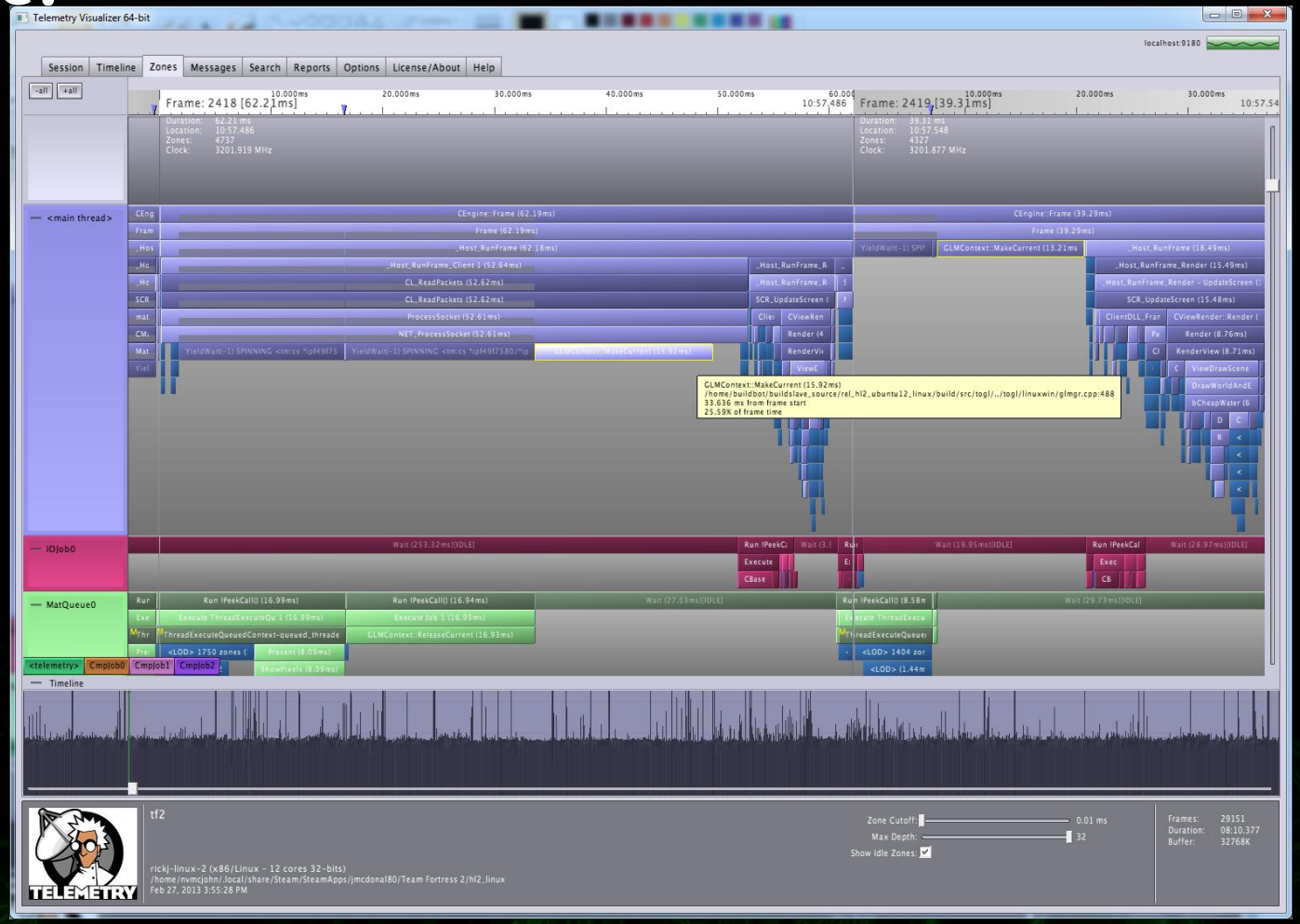
- Handedness
  - D3D is left-handed everywhere, GL is right-handed everywhere
  - Texture origin is lower-left in GL (flip coordinates about v)
  - Consider rendering upside-down, flipping at the end.
- GLSL uses column-major matrices by default
  - Including when specifying constants/uniforms
- Pixel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

MakeCurrent issues

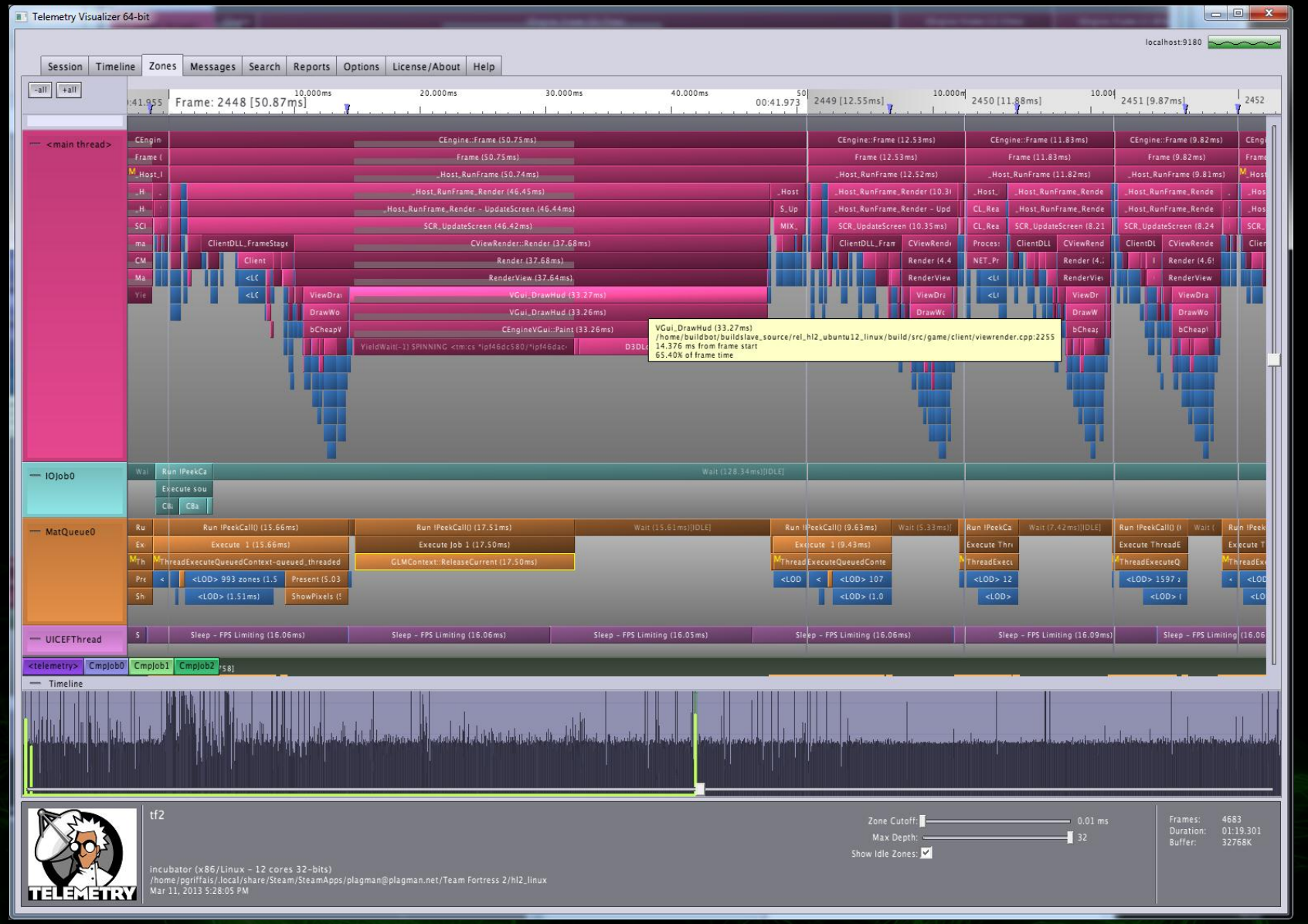
- Responsible for several bugs on TF2
- Font rendering glitches (the thread creating text tries to update the texture page, but didn't own the context

MakeCurrent Performance

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



MakeCurrent - Fixed



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.”

Device (Context) Creation in GL

- Creating a simple context in GL is easy:
  - Create a Window
  - Create a Context
- Whether this gets you a Core or Compatibility context is unspecified ☹, but most vendors give you Compatibility.
- Creating a “robust” context with a specific GL-support version requires using a WGL/GLX extension, and is trickier:

Context Creation – Cont'd

1. Create a window (don't show)
2. Create a context
3. Query for window-specific extensions
4. Create another window (this will be the application window)
5. Create a context using extension function from step 3.
6. Destroy Context from step 2.
7. Destroy window from step 1.

- Yuck.
- With SDL, SDL_GL_SetAttribute + SDL_CreateWindow.

Common D3D Idioms in GL

- Vertex Attributes
- Vertex Buffers
- Textures
- Render to texture
- Shaders

Vertex Attributes

```
glBindBuffer( GL_ARRAY_BUFFER, mPositions );
// glVertexAttribPointer remembers mPositions
glVertexAttribPointer( mProgram_v4Pos, 4, GL_FLOAT,
                       GL_FALSE, 0, 0 );
glEnableVertexAttribArray( mProgram_v4Pos );

glBindBuffer( GL_ARRAY_BUFFER, mNormals );
// glVertexAttribPointer remembers mNormals
glVertexAttribPointer( mProgram_v3Normal, 3, GL_FLOAT,
                       GL_FALSE, 0, 0 );
glEnableVertexAttribArray( mProgram_v3Normal );
```

Vertex Attribs – Alternative #1

- Vertex Attribute Objects (VAOs)
- Good mapping for D3D *(seductive!)*
- Slower than glVertexAttribPointer on all implementations
- Recommendation: Skip it

ARB_vertex_attrib_binding

- Separates Format from Binding
- Code is easy to read

```
glVertexAttribFormat( 0, 4, GL_FLOAT, FALSE, 0 );
glVertexAttribBinding( 0, 0 );
glBindVertexBuffer( 0, buffer0, 0, 24 );
```

- http://www.opengl.org/registry/specs/ARB/vertex_attrib_binding.txt

Vertex (and Index) Buffer Creation

```
GLuint vb = 0, ib = 0;
glGenBuffers( 1, &vb );
glNamedBufferDataEXT( vb, vbLengthBytes, vbPtr, vbUsage );

glGenBuffers( 1, &ib );
glNamedBufferDataEXT( ib, ibLengthBytes, ibPtr, ibUsage );
```

Vertex (and Index) Buffer Updates

```
// NO_OVERWRITE is implied if you specify non-overlapping
// regions.
glNamedBufferSubDataEXT( vb, vbOffset, vbLength, vbPtr );
glNamedBufferSubDataEXT( ib, ibOffset, ibLength, ibPtr );

// DISCARD.
glNamedBufferDataEXT( vb, vbLength, vbPtr, vbUsage );
glNamedBufferDataEXT( ib, ibLength, ibPtr, ibUsage );
```

Vertex (and Index) Buffer Using

```
// Binding VBs also involves setting up VB attributes.
glBindBuffer( GL_ARRAY_BUFFER, vb );
glVertexAttribPointer( mProgram_pos, 3, GL_FLOAT, GL_FALSE, 24, 0 );
glVertexAttribPointer( mProgram_n, 3, GL_FLOAT, GL_FALSE, 24, 12 );
glEnableVertexAttribArray( mProgram_pos );
glEnableVertexAttribArray( mProgram_n );

// We finally know what the type is!
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, ib );
```

Dynamic Buffer Updates

- Don’t use MapBuffer—because it returns a pointer, it causes driver serialization.
- Even worse, it probably causes a CPU-GPU sync point. ☹
- Instead, use BufferSubData on subsequent regions, then BufferData when it’s time to discard.

Render to Texture

- Render-to-texture in GL utilizes Frame Buffer Objects (FBOs)
- FBOs are created like other objects, and have attachment points. Many color points, one depth, one stencil, one depth-stencil
- FBOs must be “framebuffer complete” to be rendered to.
- FBOs, like other “container objects,” are not shared between contexts. ☹
- http://www.opengl.org/registry/specs/ARB/framebuffer_object.txt

Frame Buffers

- Spec has fantastic examples for creation, updating, etc, so not replicating here
- Watch BindRenderTarget (and BindDepthStencil) etc calls
- At draw time, check whether render targets are in an existing FBO configuration (exactly) via hash lookup
- If so, use it.
- If not, create a new FBO, bind attachments, check for completeness and store in cache.

Frame Buffers – Don’ts

- Do not create a single FBO and then swap out attachments on it.
- This causes *lots* of validation in the driver, which in turn leads to poor performance.

Shaders/Programs

- In GL, Shaders are attached to a Program.
  - Each Shader covers a single shader stage (VS, PS, etc)
- Shaders are Compiled
- Programs are Linked
- The Program is “used”
- This clearly doesn’t map particularly well to D3D, which supports mix-and-match.

Shaders/Programs cont'd

- GL Uniforms == D3D Constants
- Uniforms are part of program state
  - Swapping out programs also swaps uniforms
  - This also maps poorly to D3D. ☹

Uniform problem

- To solve the uniform problem, consider uniform buffer objects
  - Create a single buffer, bind to all programs
  - Modify parameters in the buffer
- Or, keep track of “global” uniform state and set values just prior to draw time
- If you’re coming from D3D11, Uniform Buffers ARE Constant Buffers—no problems there.
- http://www.opengl.org/wiki/Uniform_Buffer_Object
- http://www.opengl.org/registry/specs/ARB/uniform_buffer_object.txt

Shader Approach #1: Program Hash

- Pay attention to shaders that get set.
- At draw time, hash the names of the shaders to see if an existing program object has been linked
- Otherwise, link and store in the hash

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 *_tessellation_program

Shader Translation cont'd

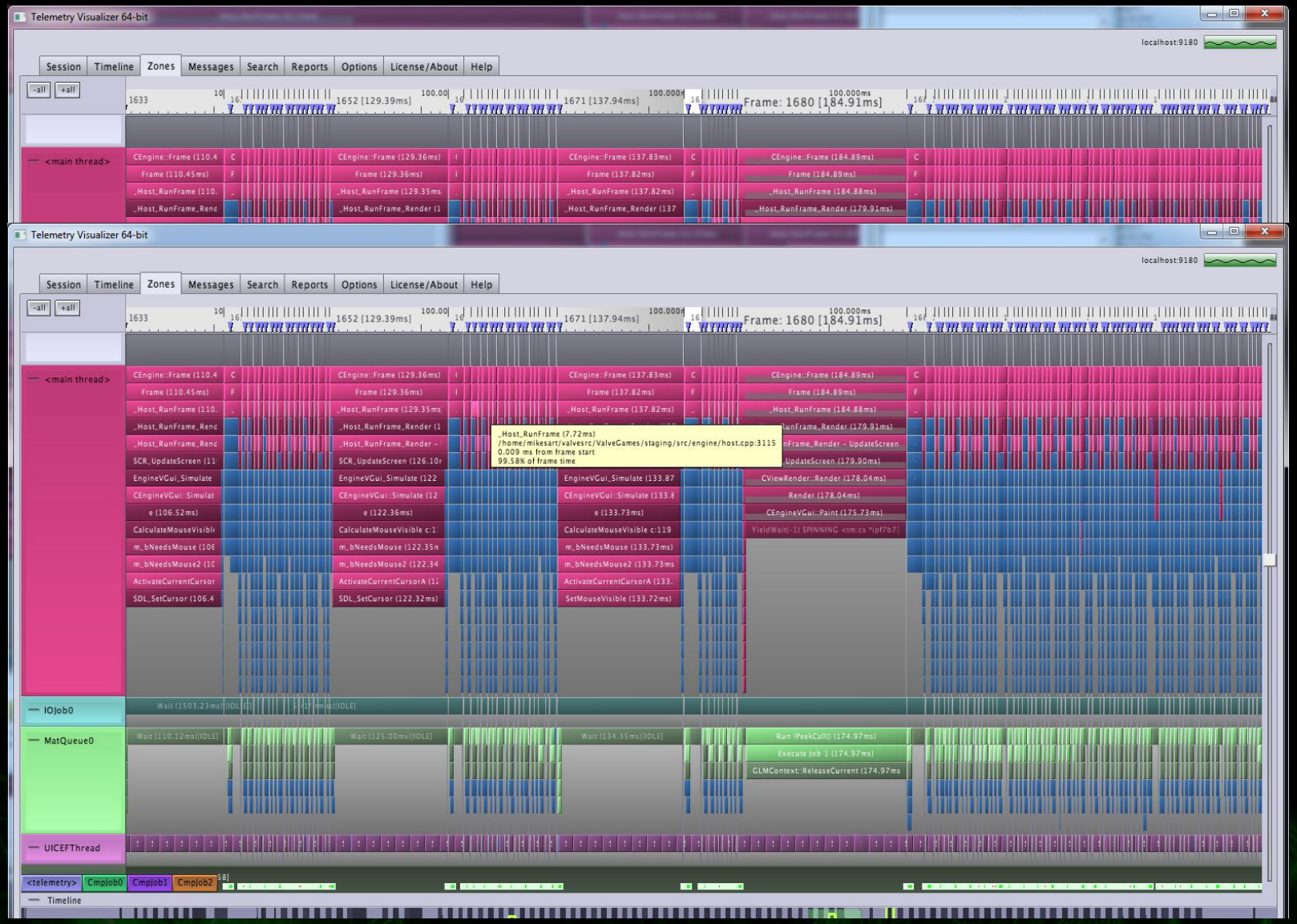
- One approach: compile HLSL, translate the byte code to simple GLSL asm-like.
- Pro: One set of shaders goes public
- Pro: Can be *fast*
- Con: Can be *hard* to debug problems
- Con: Potentially slow fxc idioms end up in generated GLSL
- Con: Debugging requires heavy cognitive load

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.

Questions?

- jmcdonald at nvidia dot com
- richg at valvesoftware dot com

Appendix

- Some other GL gotchas/helpers

Magic Symbol Resolution

- Linux equivalent of _NT_SYMBOL_PATH
- In ~/.gdbinit:
  - set debug-file-directory /usr/lib/debug:/mnt/symstore/debug
- /mnt/symstore/debug is a shared, remotely mounted share with your symbols
- Populate that server with symbols
- Currently only applied to gdb, should also apply to Google's perf tool "soon"

http://randomascii.wordpress.com/2013/02/20/symbols-on-linux-part-three-linux-versus-windows/
http://fedoraproject.org/wiki/Releases/FeatureBuildId
http://randomascii.wordpress.com/category/symbols-2/

Performance tips

- Force-inline is your friend—many of the functions you'll be implementing are among the most-called functions in the application.
- With few exceptions, you can maintain a GL:D3D call ratio of 1:1 or less.
  - For example, use glBindMultiTextureEXT instead of glActiveTexture/glBindTexture.
  - glBindMultiTextureEXT(texUnit, target, texture)

Other useful GL references

- http://www.opengl.org/wiki/Common_Mistakes
- OpenGL SuperBible: Comprehensive Tutorial and Reference (5th Edition)
  - http://www.amazon.com/OpenGL-SuperBible-Comprehensive-Tutorial-Reference/dp/0321712617/
- OpenGL 4.2 Quick Reference Card
  - http://www.khronos.org/files/opengl42-quick-reference-card.pdf

Sampler gotchas…

- On certain drivers, GL_TEXTURE_COMPARE_MODE (for shadow map lookups) is buggy when set via sampler.
- For robustness, use texture setting on those particular drivers.

Latched State

- Recall that GL is very stateful.
- State set by an earlier call is often captured (latched) by a later call.
- Vertex Attributes are the prime example of this, but there are numerous other examples.

Textures (Creation)

```
GLuint texId = 0;
// Says “This handle is a texture”
glGenTextures(1, &texId);

// Allocates memory
glTextureStorage2DEXT( texId, GL_TEXTURE_2D, mipCount,
                       texFmt, mip0Width, mip0Height );

// Pushes data—note that conversion is performed if necessary
foreach (mipLevel) {
    glTextureSubImage2DEXT( texId, GL_TEXTURE_2D, mipLevel,
                            0, 0, mipWidth, mipHeight,
                            srcFmt, srcType, mipData );
}
```

Textures (Updating)

- With TexStorage, updates are just like initial data specification (glTextureSubImage or glCompressedTextureSubImage).
- Texture->Texture updates are covered later
- On-GPU compression is straightforward, implemented in https://code.google.com/p/nvidia-texture-tools/
  - MIT License, use freely!
- Or copy Simon Green's technique:
  - http://developer.download.nvidia.com/SDK/10/opengl/samples.html#compress_YCoCgDXT

Textures (Setting State)

```
// Sets minification filtering on texture 7
// This parameter will be ignored if a sampler is bound.
glTextureParameteri( 7,  GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                     GL_NEAREST );
```

Textures (Using)

```
// Binds texture 7 to texture unit 3.
glBindMultiTextureEXT(3, GL_TEXTURE_2D, 7);
```

StretchRect

- Implementing StretchRect in GL involves using Read/Write FBOs.
- Bind source as a read target
- Bind destination as a write target
- Draw!
- Alternatives:
  - No stretching/format conversion? EXT_copy_texture
  - Stretching / format conversion? NV_draw_texture

StretchRect – MSAA case

- When MSAA is involved, use EXT_framebuffer_multisample_blit_scaled
- Allows resolving and resizing in a single blit
- Otherwise two blits needed (one for resolve, one for resize)

Other GL/D3D differences

- Clip Space
  - D3D:
    - -w <= x <= w
    - -w <= y <= w
    - 0 <= z <= w
  - GL
    - -w <= x <= w
    - -w <= y <= w
    - -w <= z <= w
  - But anything with w < 0 still clipped by W=0 clipping
- Latched State – let's get back to this.