GPU TECHNOLOGY CONFERENCE

NVIDIA OpenGL in 2012

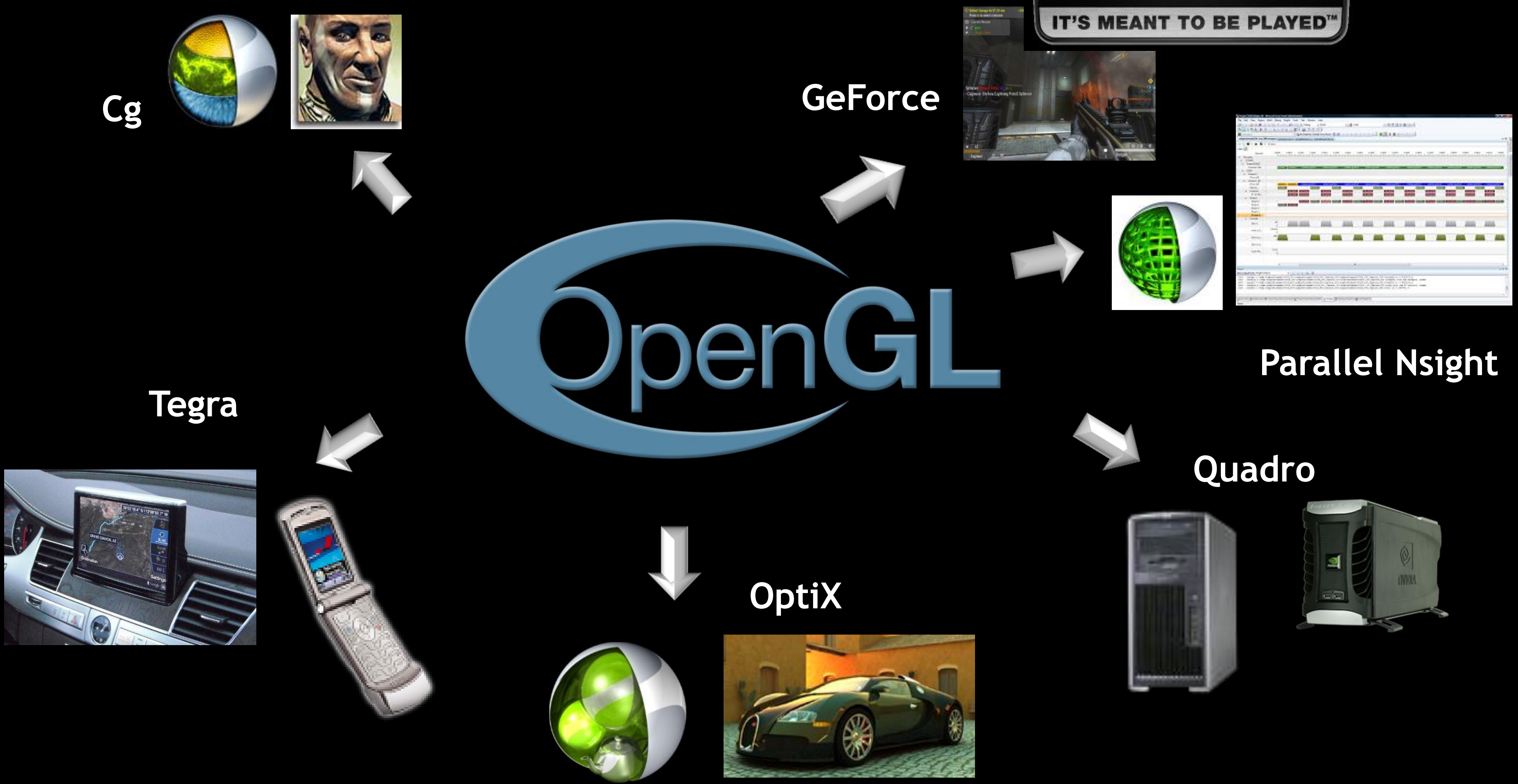
Mark Kilgard

- Principal System Software Engineer
  - OpenGL driver and API evolution
  - Cg (“C for graphics”) shading language
  - GPU-accelerated path rendering
- OpenGL Utility Toolkit (GLUT) implementer
- Author of *OpenGL for the X Window System*
- Co-author of *Cg Tutorial*

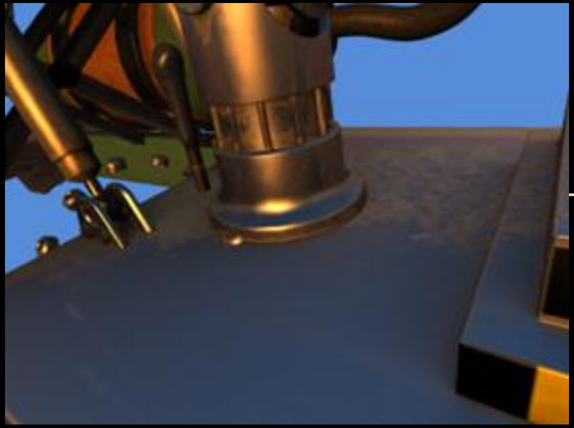
Outline

- OpenGL's importance to NVIDIA
- OpenGL API improvements & new features
  - OpenGL 4.2
  - Direct3D interoperability
  - GPU-accelerated path rendering
  - Kepler Improvements
    - Bindless Textures
- Linux improvements & new features
- Cg 3.1 update

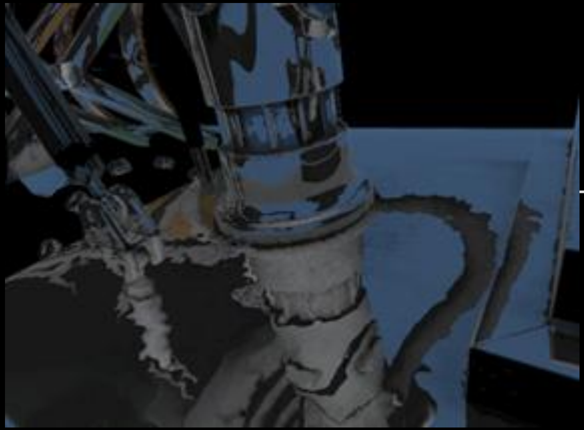
NVIDIA's OpenGL Leverage



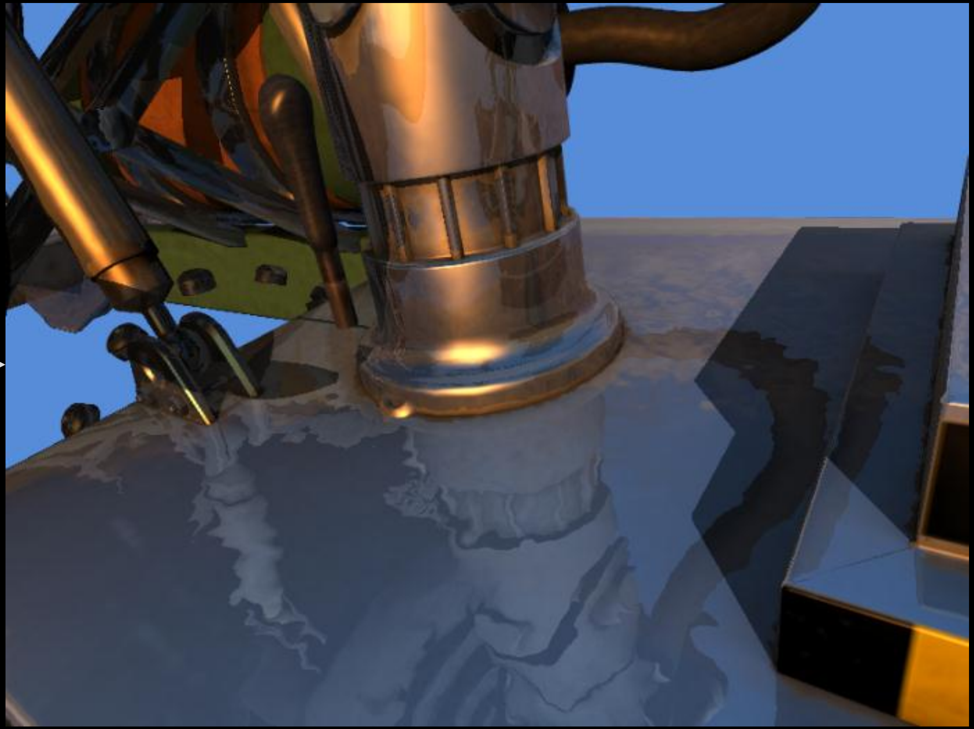
Example of Hybrid Rendering with OptiX



OpenGL (Rasterization)

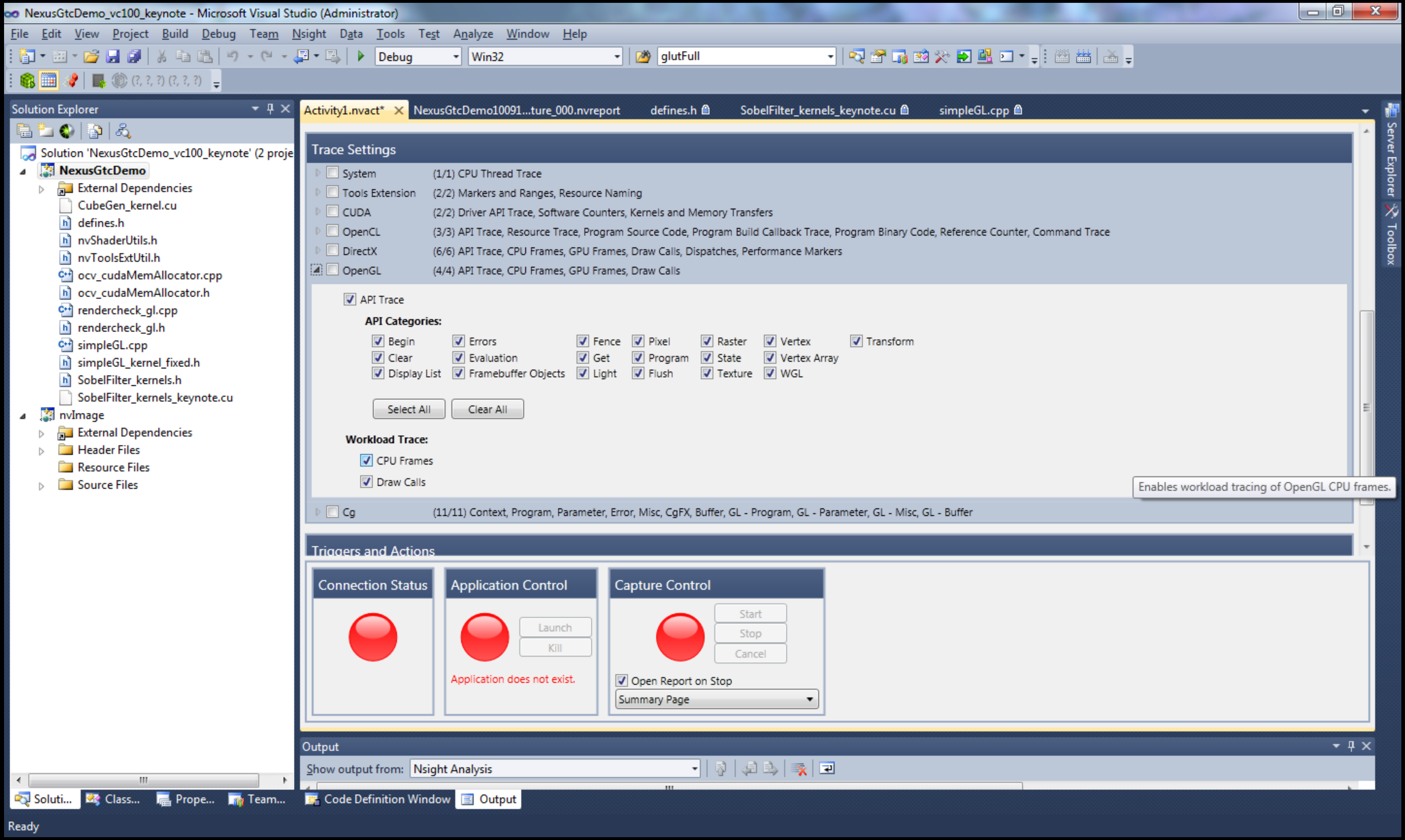


OptiX (Ray tracing)



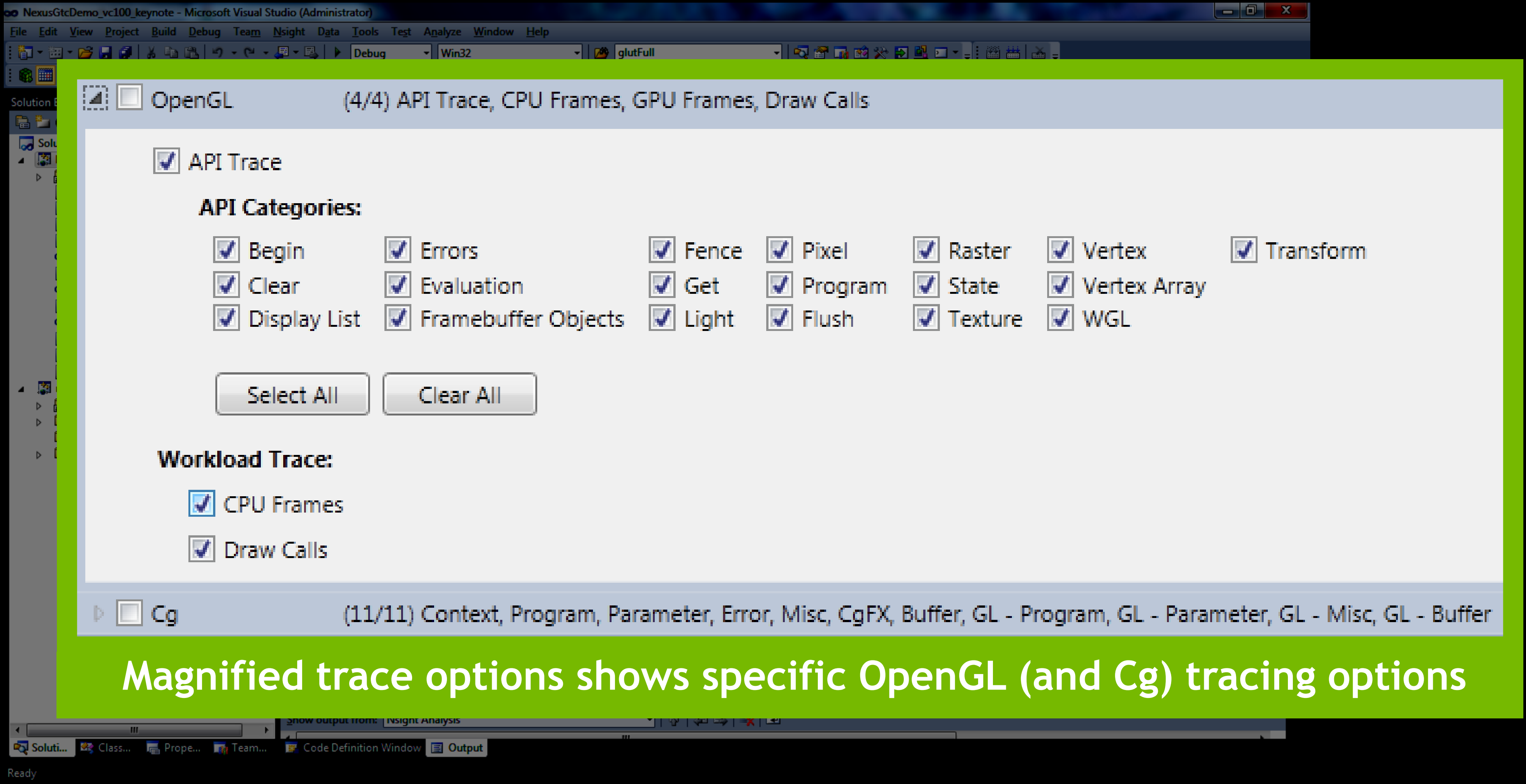
GPU TECHNOLOGY CONFERENCE

Parallel Nsight Provides OpenGL Profiling



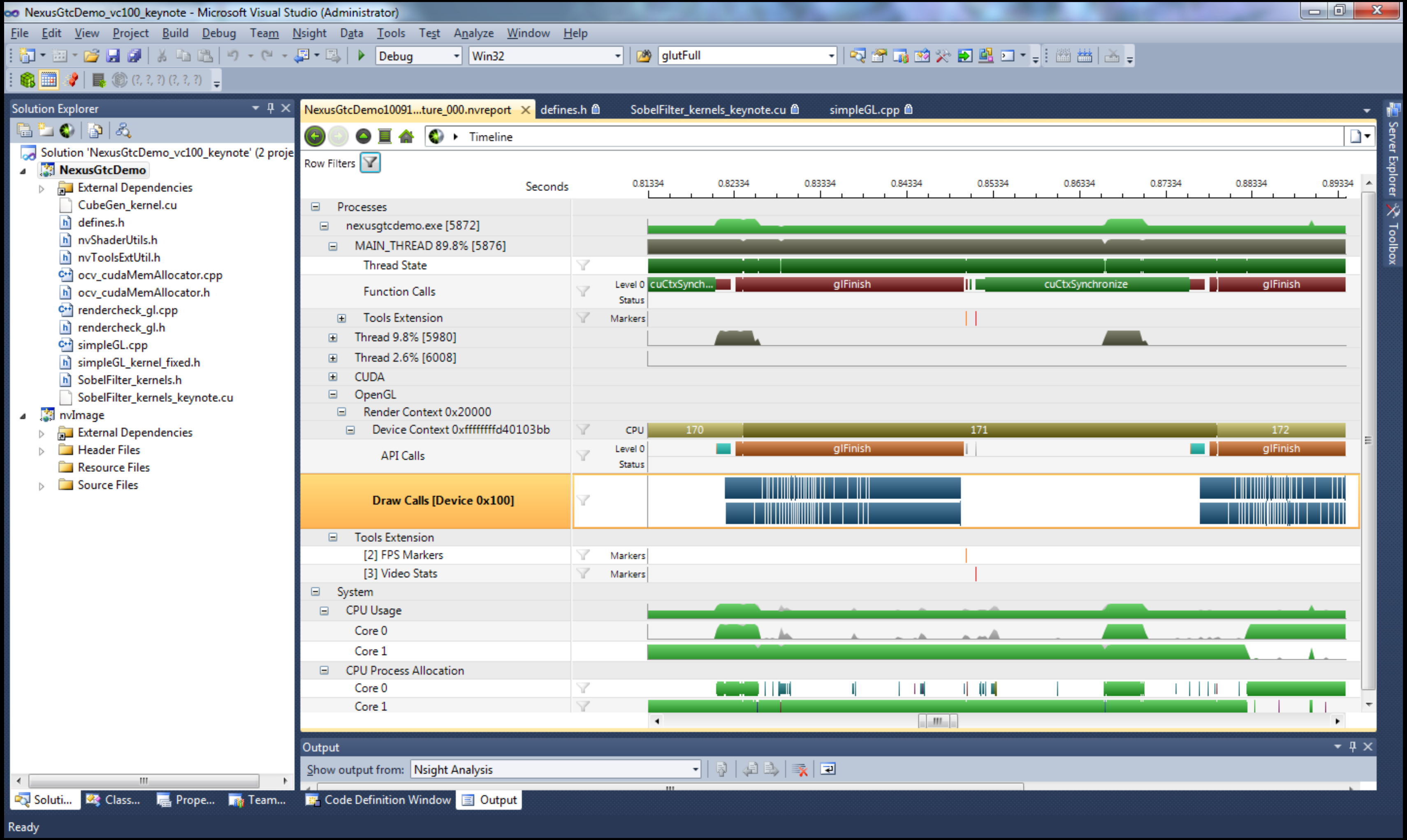
Configure Application Trace Settings

Parallel Nsight Provides OpenGL Profiling

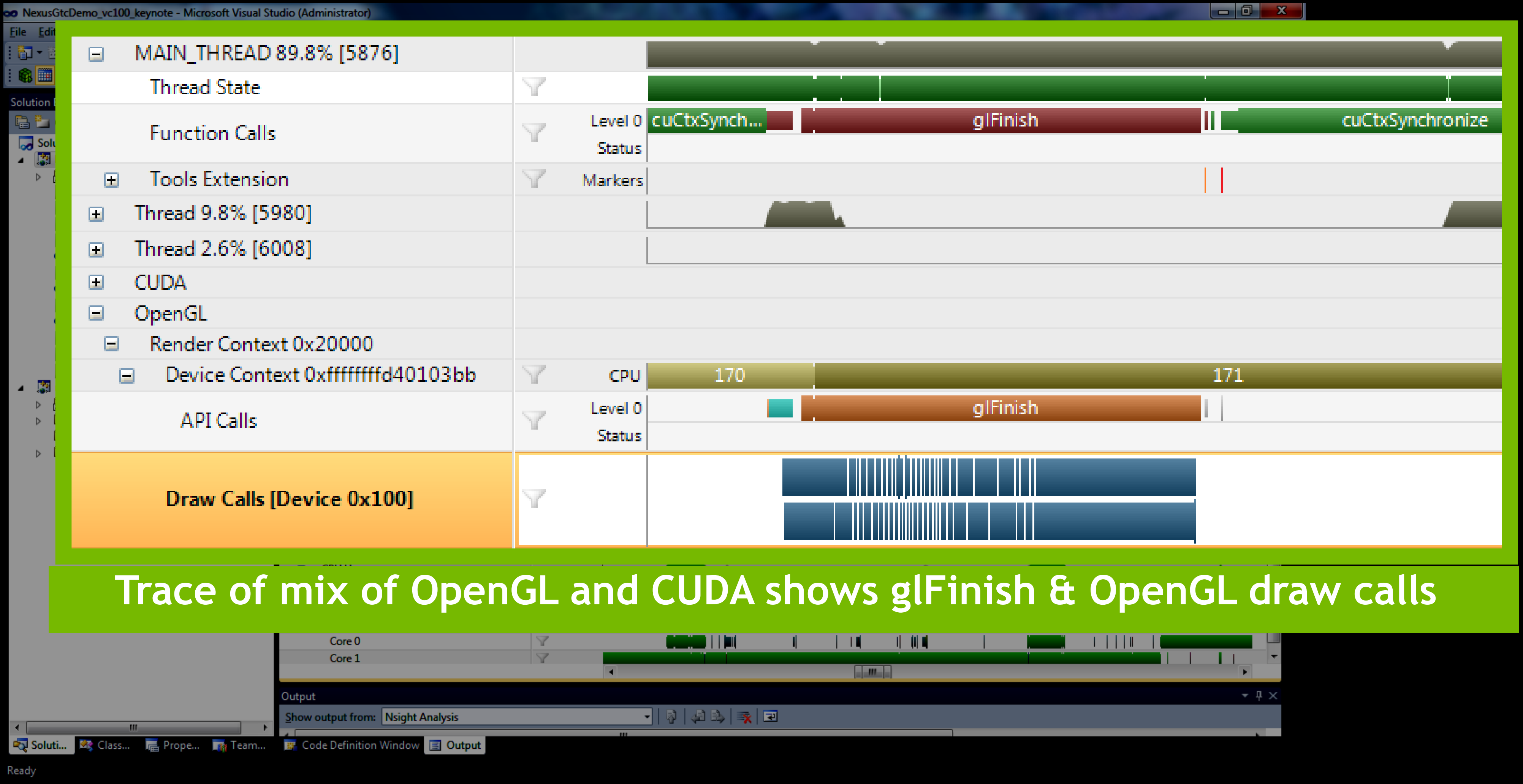


Magnified trace options shows specific OpenGL (and Cg) tracing options

Parallel Nsight Provides OpenGL Profiling



Parallel Nsight Provides OpenGL Profiling



Trace of mix of OpenGL and CUDA shows glFinish & OpenGL draw calls

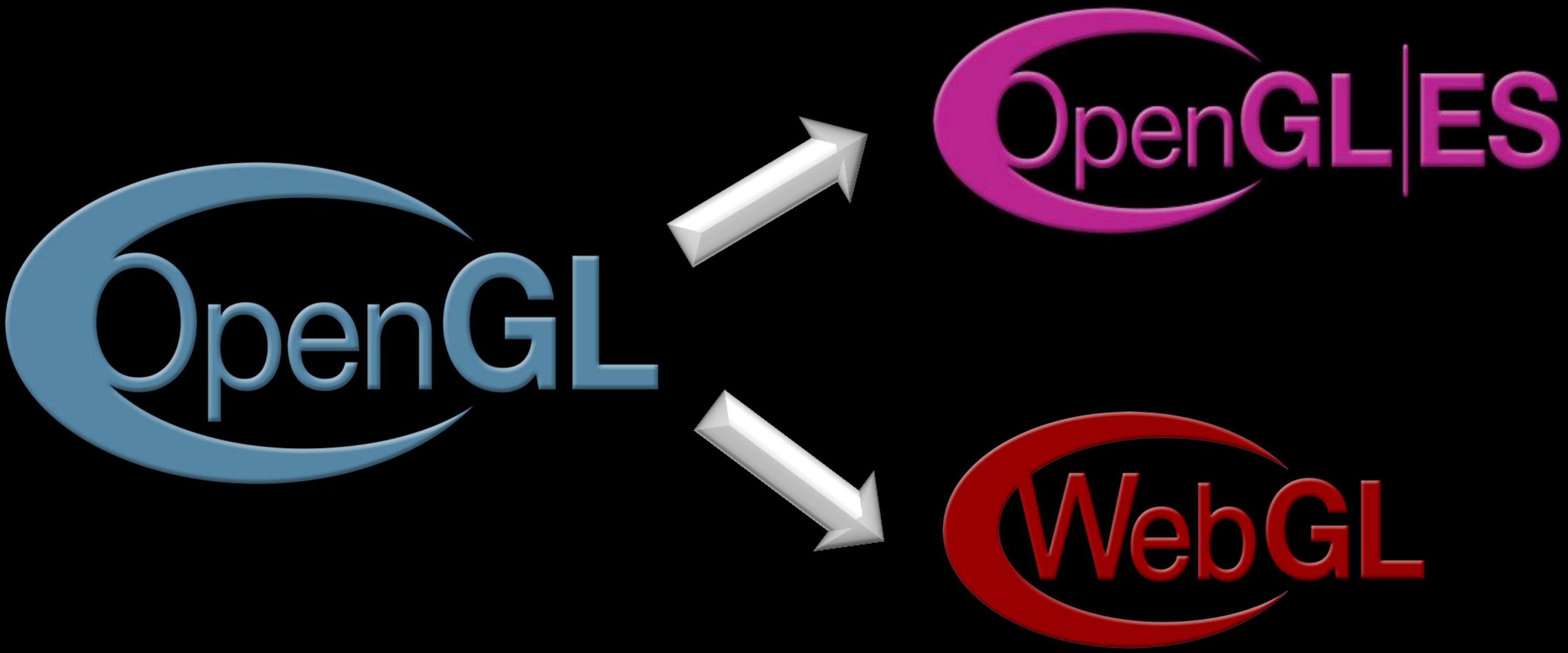
Only Cross Platform 3D API

GPU TECHNOLOGY CONFERENCE

OpenGL 3D Graphics API

- cross-platform
- most functional
- peak performance
- open standard
- inter-operable
- well specified & documented
- 20 years of compatibility

OpenGL Spawns Closely Related Standards



Congratulations: WebGL officially approved, February 2012

"The web is now 3D enabled"

GPU TECHNOLOGY CONFERENCE

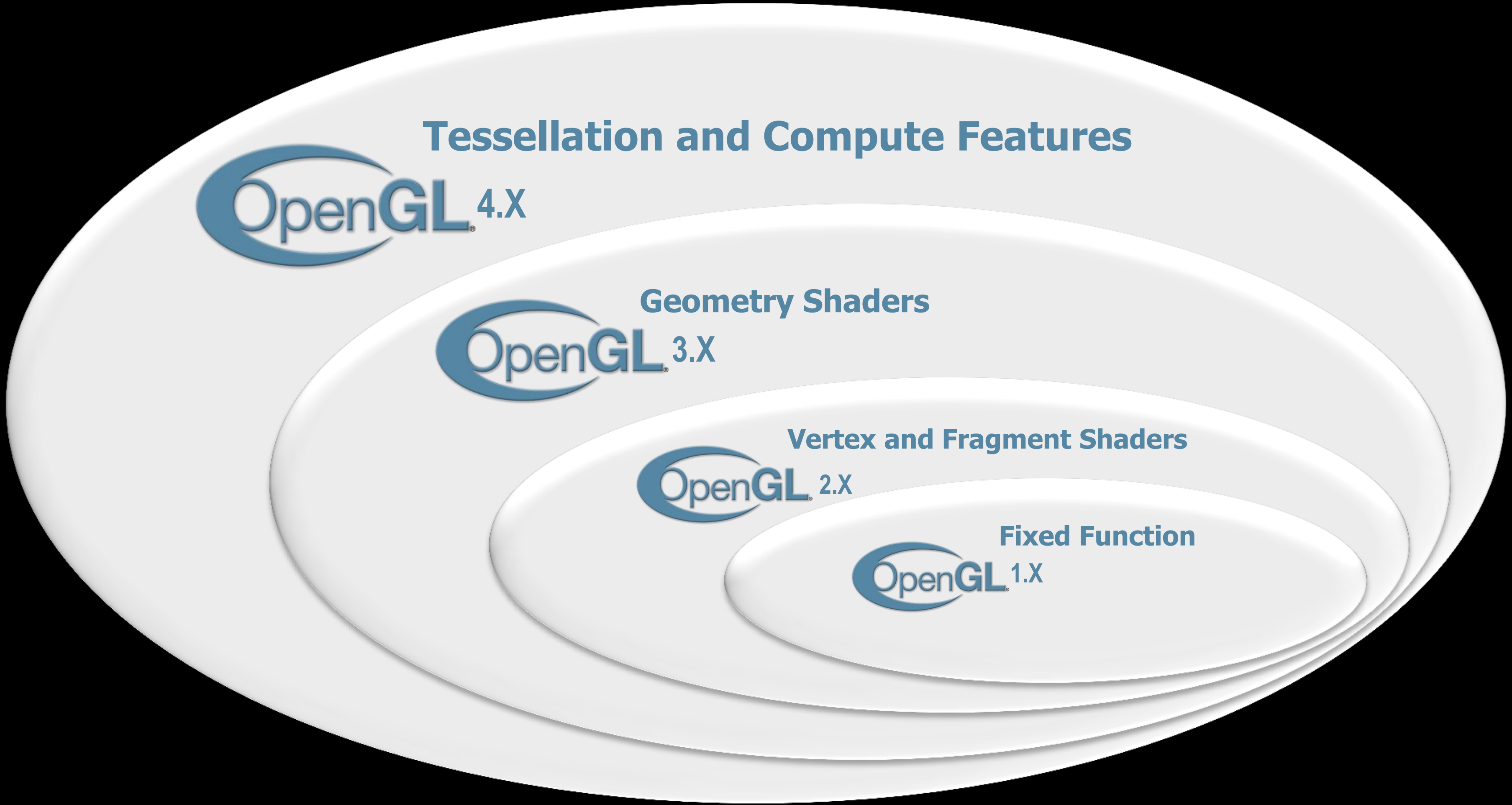
OpenGL 4 - DirectX 11 Superset

Buffer *and* Event Interop

- Interop with a complete compute solution
  - OpenGL is for graphics – CUDA / OpenCL is for compute
- Shaders can be saved to and loaded from binary blobs
  - Ability to query a binary shader, and save it for reuse later

- Flow of content between desktop and mobile
  - All of OpenGL ES 2.0 capabilities available on desktop
  - EGL on Desktop in the works
  - WebGL bridging desktop and mobile
- Cross platform
  - Mac, Windows, Linux, Android, Solaris, FreeBSD
  - Result of being an open standard

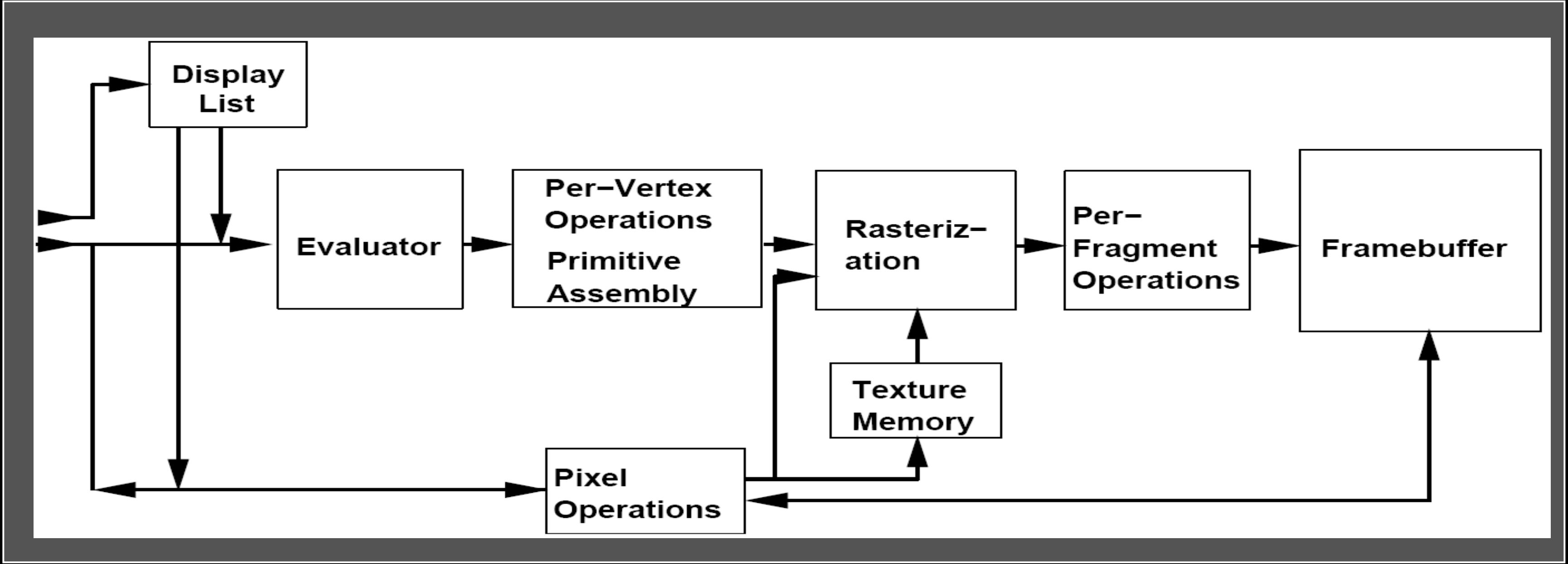
Increasing Functionality of OpenGL



Classic OpenGL State Machine

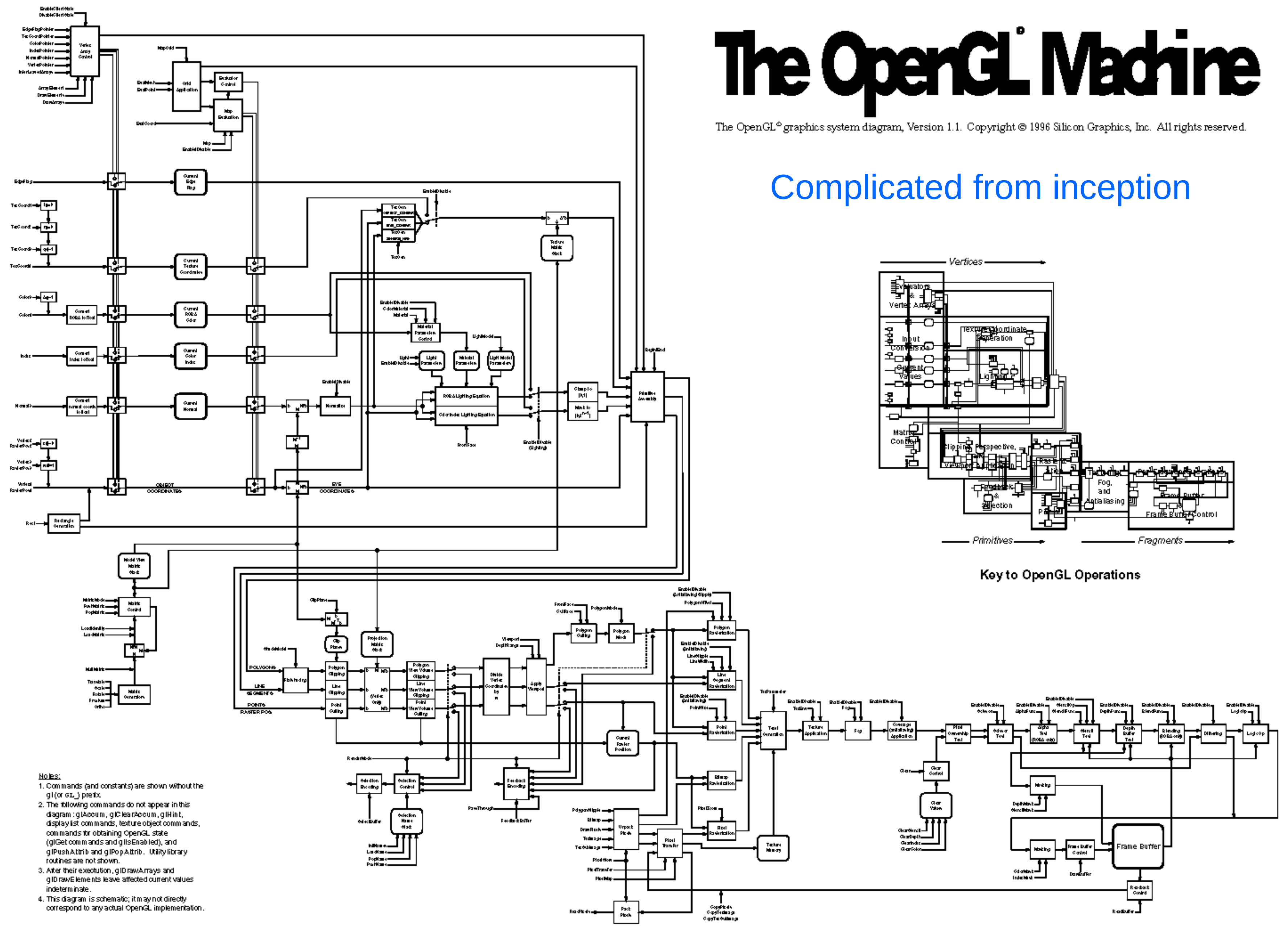
- From 1991-2007

  * vertex & fragment processing got programmable 2001 & 2003

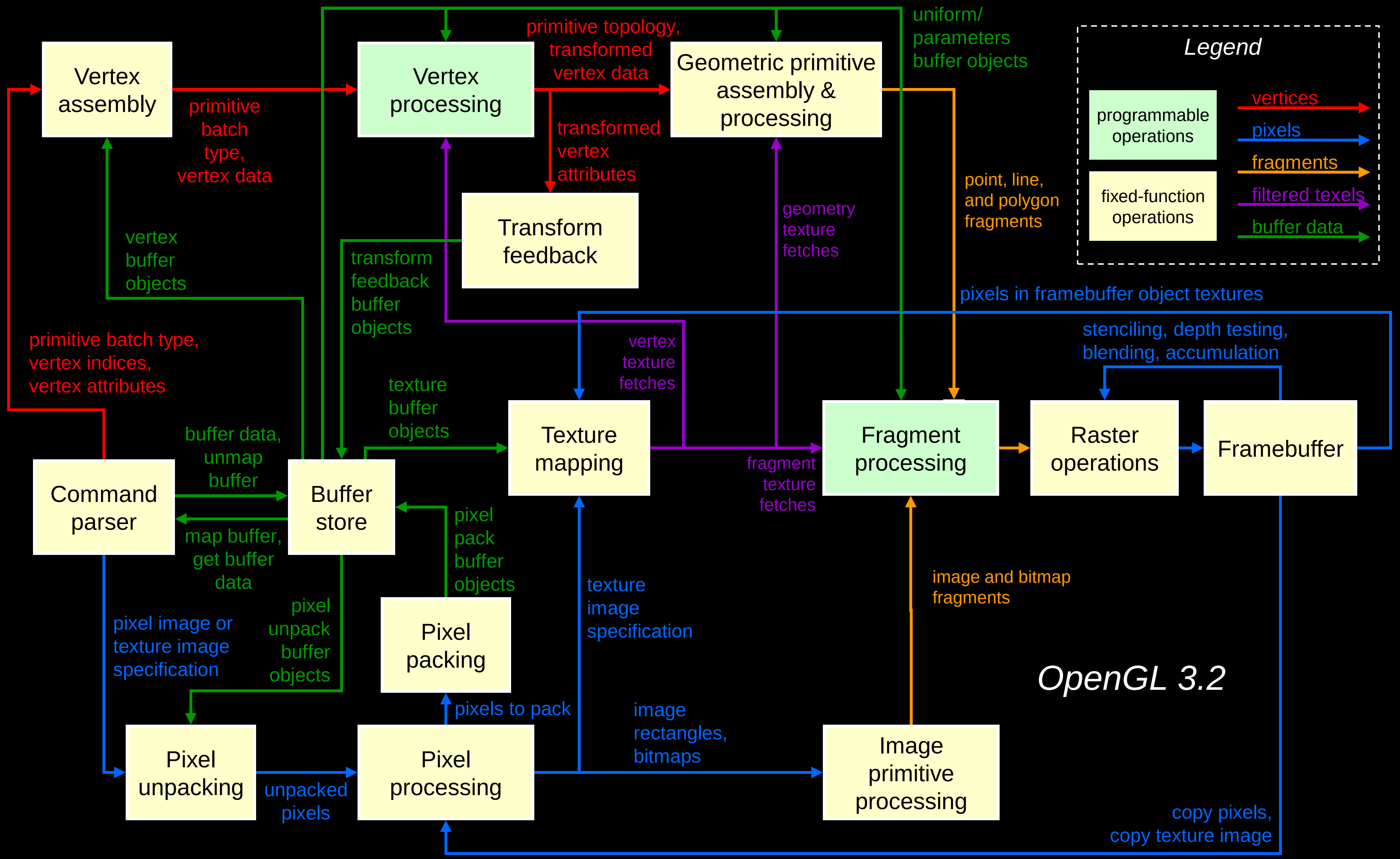


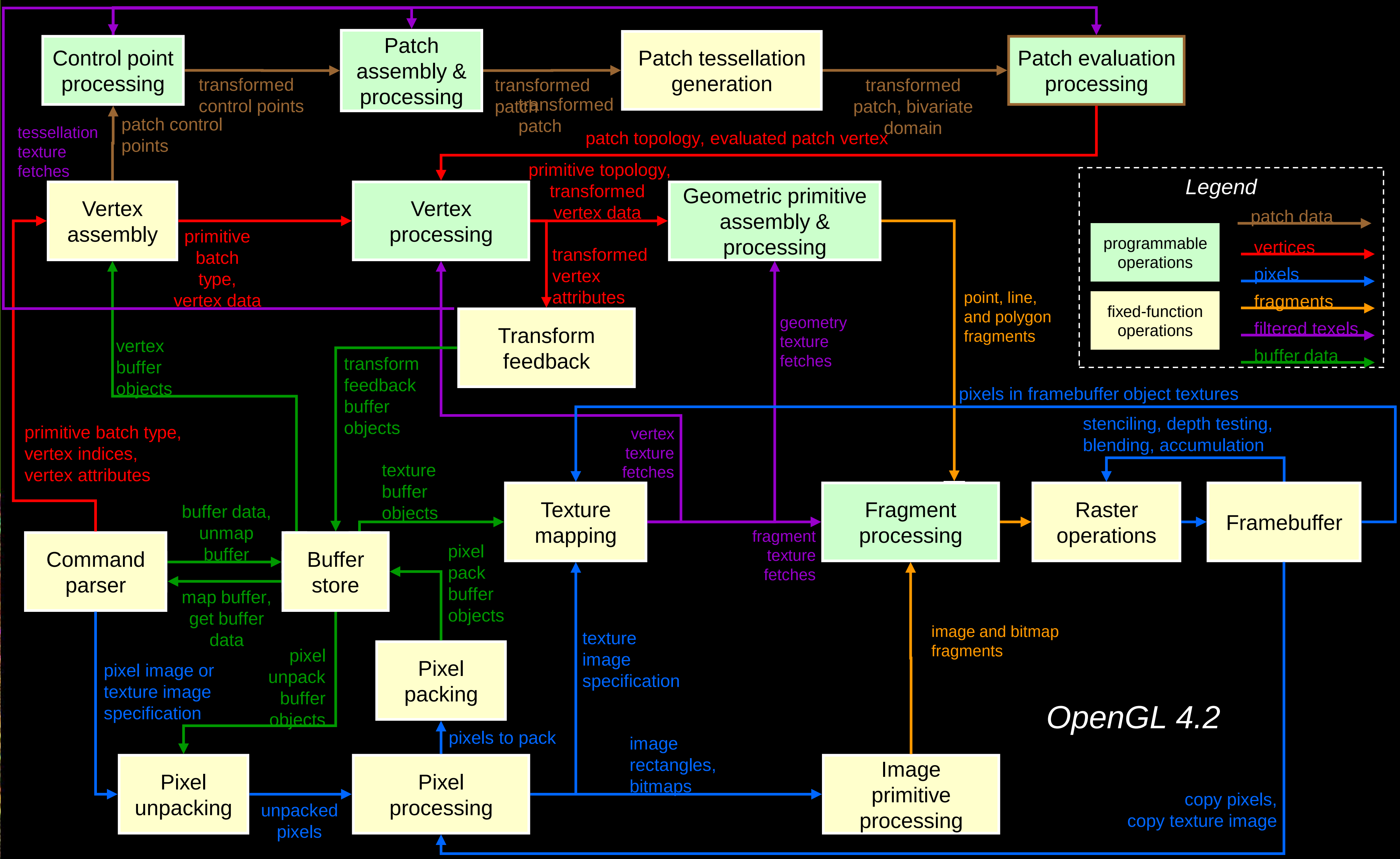
[source: GL 1.0 specification, 1992]

Complicated from inception

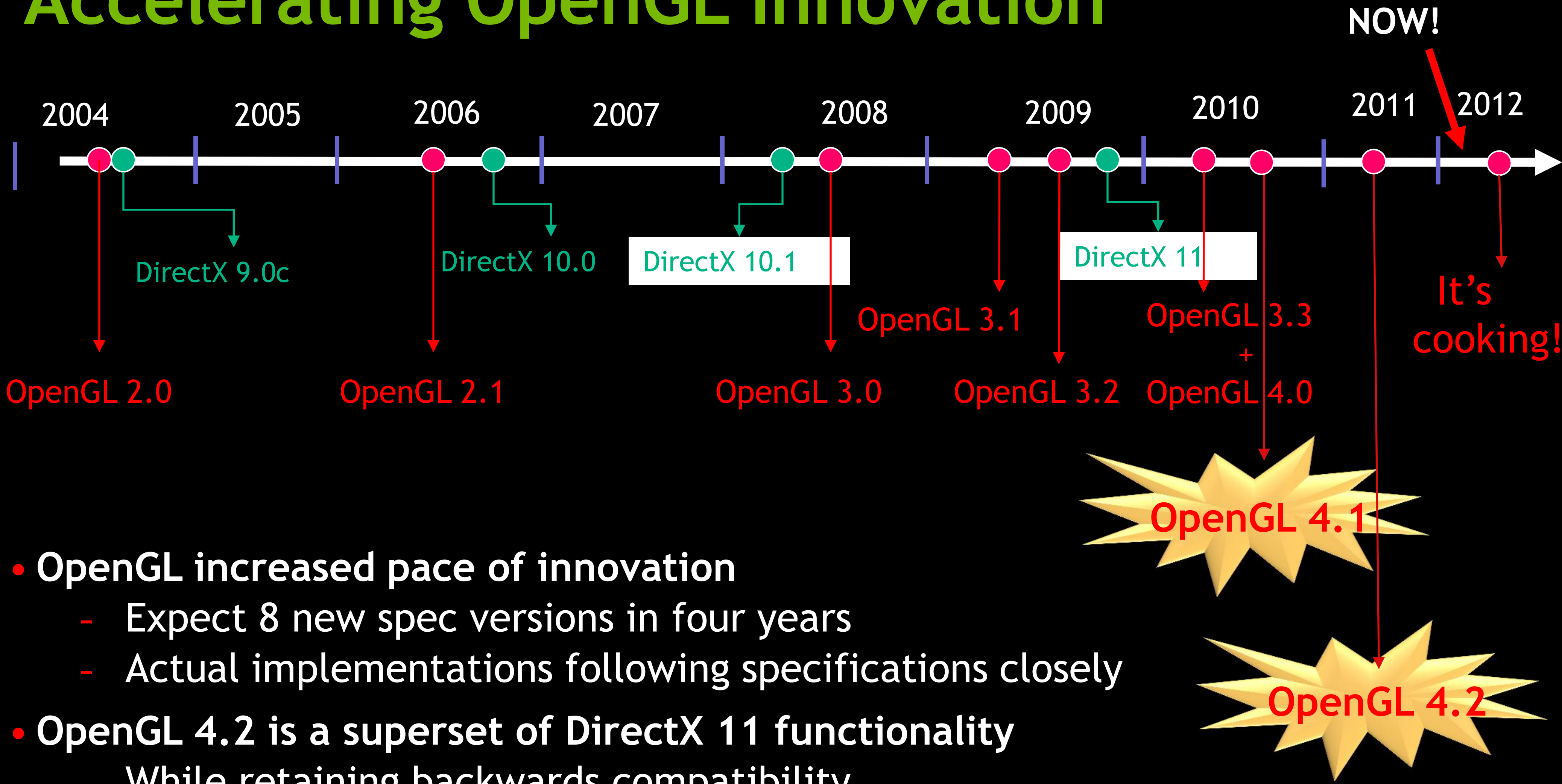


OpenGL 3.x Conceptual Processing Flow (pre-2010)




Control point processing
transformed control points
Patch assembly & processing
transformed patch
transformed patch
Patch tessellation generation
transformed patch, bivariate domain
Patch evaluation processing
tessellation texture fetches
patch control points
patch topology, evaluated patch vertex
Vertex assembly
primitive batch type, vertex data
Vertex processing
primitive topology, transformed vertex data
Geometric primitive assembly & processing
transformed vertex attributes
Transform feedback
geometry texture fetches
point, line, and polygon fragments
Legend
patch data
vertices
pixels
fragments
filtered texels
buffer data
programmable operations
fixed-function operations
vertex buffer objects
transform feedback buffer objects
pixels in framebuffer object textures
stenciling, depth testing, blending, accumulation
primitive batch type, vertex indices, vertex attributes
texture buffer objects
vertex texture fetches
Texture mapping
Fragment processing
Raster operations
Framebuffer
buffer data, unmap buffer
Command parser
Buffer store
map buffer, get buffer data
pixel pack buffer objects
fragment texture fetches
image and bitmap fragments
texture image specification
pixel unpack buffer objects
Pixel packing
pixel image or texture image specification
OpenGL 4.2
pixels to pack
image rectangles, bitmaps
Pixel unpacking
unpacked pixels
Pixel processing
Image primitive processing
copy pixels, copy texture image

Accelerating OpenGL Innovation



- **OpenGL increased pace of innovation**
  - Expect 8 new spec versions in four years
  - Actual implementations following specifications closely
- **OpenGL 4.2 is a superset of DirectX 11 functionality**
  - While retaining backwards compatibility

What's new in OpenGL 4.2?

- OpenGL 4.2 standardized August 2011
  - Immediately shipped by NVIDIA
  - All Kepler and Fermi GPUs support 4.2
- Standardizes:
  - Compressed RGBA texture
  - Shader atomic counters
  - Immutable texture images
  - Further Direct3Disms
  - OpenGL Shading Language (GLSL) 4.20

OpenGL 4.2

- Official compressed RGBA texture formats
  - GL_ARB_texture_compression_bptc
  - S3TC/DXTC was widely implemented as EXT, but was never an ARB standard
  - Now, there's standard RGBA compressed format
- Pixel store modes for compressed block texture specification
  - ARB_compressed_texture_pixel_storage
  - New pixel store modes for dealing with sub-rectangle loads of blocked compressed texture formats

Details on BPTC

- Four new compressed formats
  - GL_COMPRESSED_RGBA_BPTC_UNORM
    - RGBA format, lower signal-to-noise than S3TC
    - [0,1] component range for low-dynamic range (LDR) images
  - GL_COMPRESSED_SRGB_ALPHA_BPTC_UNORM
    - sRGB version of GL_COMPRESSED_RGBA_BPTC_UNORM
  - GL_COMPRESSED_RGB_BPTC_SIGNED_FLOAT
    - Compressed high dynamic range (HDR) format
  - GL_COMPRESSED_RGB_BPTC_UNSIGNED_FLOAT
    - HDR format for magnitude (unsigned) component values
- Each has fixed compression ratio with 4x4 blocks
  - Much more intricate encoding than S3TC format (see specification)

Easy Adoption of BPTC Compression

- If you are loading uncompressed texture data
  - Just use BPTC internal texture format with glTexImage2D
  - Driver will compress the image for you
- And you can read back the compressed image
  - Then subsequent loads can use glTexCompressedTexImage2D to be faster
- Same strategy works for DXTC/S3TC
  - But BTC encoder is more involved so makes more sense

GPU TECHNOLOGY CONFERENCE

OpenGL 4.2 Compressed Texture Format Table

Token name	Block Size	Fixed lossy Compression ratio	Range
GL_COMPRESSED_RGBA_BPTC_UNORM	4x4	6:1 RGB	LDR
GL_COMPRESSED_SRGB_ALPHA_BPTC_UNORM	4x4	6:1 sRGB	LDR
GL_COMPRESSED_RGB_BPTC_SIGNED_FLOAT	4x4	3:1 for RGB 4:1 for RGBA	Unsigned HDR
GL_COMPRESSED_RGB_BPTC_UNSIGNED_FLOAT	4x4	3:1 for RGB 4:1 for RGBA	Signed HDR

OpenGL 4.2 Compressed Texture Pixel Storage

- New glPixelStorei state settings
  - Unpack state
    - GL_UNPACK_COMPRESSED_BLOCK_WIDTH
    - GL_UNPACK_COMPRESSED_BLOCK_HEIGHT
    - GL_UNPACK_COMPRESSED_BLOCK_DEPTH
    - GL_UNPACK_COMPRESSED_BLOCK_SIZE
  - Pack state, similar just starting GL_PACK_ ...
- Allows sub rectangle update for glCompressedTexImage2D, glCompressedTexSubImage2D, etc.

OpenGL 4.2 Atomic Counters

- Shaders seek to write/read from buffers need a way to get a unique memory location across alls shader instances
  - Prior to atomic counters, no way to coordinate such reads or writes
- New: atomic counters
  - Allow GLSL shaders to write at unique offset within a buffer object
  - Also allow GLSL shader to read at unique offset within a buffer object
  - Counter value stored in a buffer
    - Updated via glBuferSubData, glMapBuffer, etc.
- GLSL declaration of opaque object
  - layout (binding = 2, offset = 0) uniform atomic_uint variable;
- GLSL usage
  - uint prior_value = atomicCounterIncrement(atomic_uint *counter*);

OpenGL 4.2 Atomic Counters

- More GLSL usage of atomic counters
  - uint updated_value = atomicCounterDecrement(atomic_uint *counter*);
  - uint current_value = atomicCounter(atomic_uint *counter*);
- Using
  - #extension GL_ARB_shader_atomic_counters : enable
- Expected usage
  - Atomic values are offsets for loads and store operations
  - Using GLSL imageLoad & imageStore built-in functions
  - Also part of OpenGL 4.2

In OpenGL 4.2, Shaders allowed Side-effects

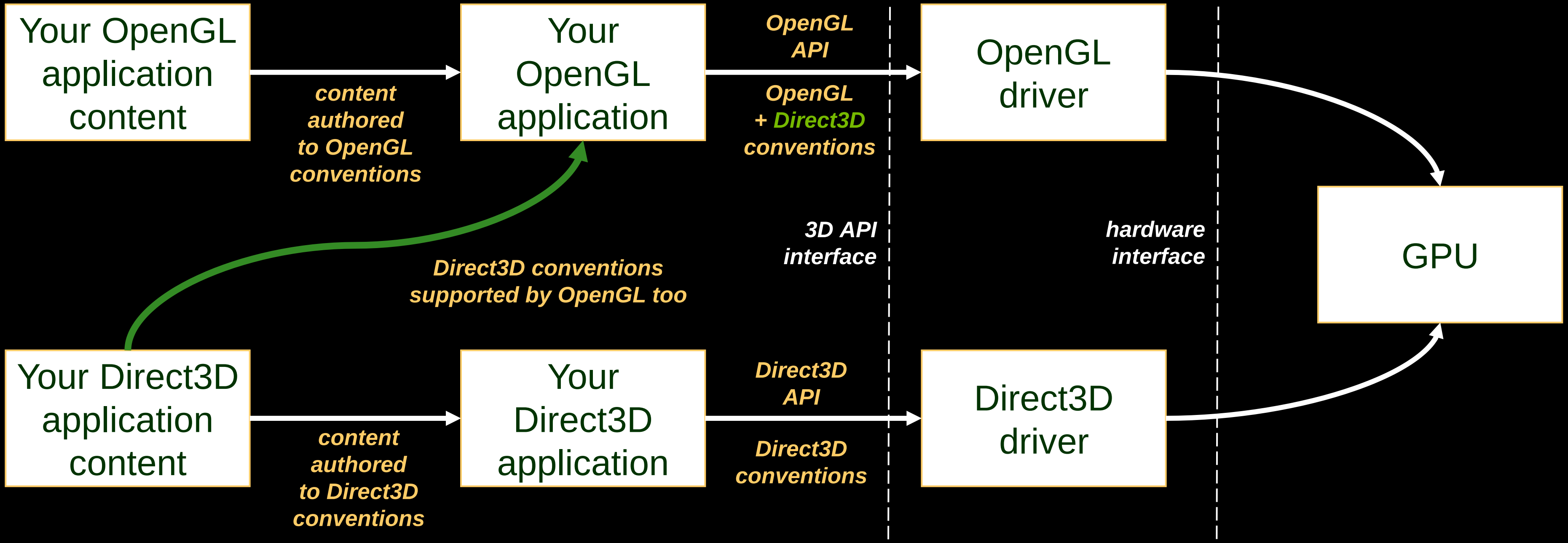
- Very exciting development
  - Prior to 4.2, standard GLSL shaders had their explicit outputs and that was it
- In OpenGL 4.2
  - Loads and stores to images and buffers allowed from shaders
  - Use atomic operations or atomic counters to synchronize those updates
  - You need to manage any required ordering though

OpenGL 4.2 Immutable Texture Storage

- OpenGL allows flexible layout of mipmap levels with a texture object
- New commands gives a frozen mipmap layout
  - glTexStorage1D, glTexStorage2D, glTexStorage3D
    - Free of selector
  - glTextureStorage1DEXT, glTextureStorage2DEXT, glTextureStorage3DEXT
    - Explicit texture parameter
- Allow driver to assume layout of texture immutable
  - Still can update texels, just not layout

Direct3Dism Concept

- Allows your 3D content to be API agnostic
  - OpenGL supports both OpenGL & Direct3D conventions, so support either style



Further Direct3Disms in OpenGL 4.2

- Direct3Disms = semantic compatibility with Direct3D behaviors
  - Not just same functionality as Direct3D—but same functionality with Direct3D-style semantics
- Alignment constraints for mapped buffers
  - ARB_buffer_alignment
  - Makes sure mapped buffers are aligned for SIMD CPU instruction access
  - Really just a query for minimum alignment
    - Expect alignment to be no smaller than 64 bytes

OpenGL 4.2 Base Instance

- Extension on instanced rendering
  - For drawing multiple instances of geometry
    - Designed to be compatible with Direct3D
  - See ARB_instanced_arrays
- Allows the specification of
  - *base vertex*
  - *base instance*
  - for each instanced draw
- Generalizes instanced rendering

OpenGL 4.2 GLSL Features

- Conservative Depth
  - Permits depth/stencil tests before shading
  - Re-declares gl_FragDepth to restrict depth changes
- Allows restrictions
  - New depth can only increase depth
    - layout (depth_greater) out float gl_FragDepth;
  - New depth can only decrease depth
    - layout (depth_less) out float gl_FragDepth;
- To use
  - #extension GL_ARB_conservative_depth : enable

OpenGL 4.2 GLSL Features

- Miscellaneous GLSL changes
  - Allow line-continuation character using ‘\’
  - Use UTF8 for character set, for comments
  - Implicit conversions of return values
  - const keyword allowed on variables with functions
  - No strict order of qualifiers
  - Adds layout qualifier binding for uniform blocks
  - C-style aggregate initializers
  - Allow .length method to return number of components or columns in vectors and matrices
  - Allow swizzles on scalars
  - New built-in constants for gl_MinProgramTexelOffset and Max
- All minor improvements for consistency with C/C++ and minimize aggravation

OpenGL-related Linux Improvements

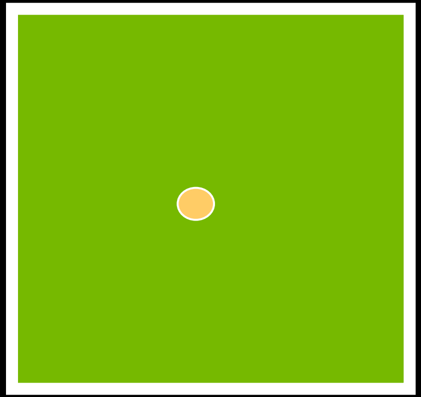
- Support for X Resize, Rotate, and Reflect Extension
  - Also known as RandR
  - Version 1.2 and 1.3
- OpenGL enables, by default, “Sync to Vertical Blank”
  - Locks your **glXSwapBuffers** to the monitor refresh rates
  - Matches Windows default now
  - Previously disabled by default

OpenGL-related Linux Improvements

- Expose additional full-scene antialiasing (FSAA) modes
  - 16x multisample FSAA on all GeForce GPUs
    - 2x2 supersampling of 4x multisampling
  - Ultra high-quality FSAA modes for Quadro GPUs
    - 32x multisample FSAA
      - 2x2 supersampling of 8x multisampling
    - 64x multisample FSAA
      - 4x4 supersampling of 4x multisampling
- Coverage sample FSAA on GeForce 8 series and better
  - 4 color/depth samples + 12 depth samples

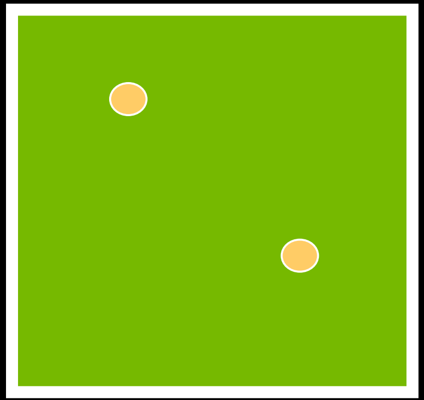
Multisampling FSAA Patterns

aliased
1 sample/pixel



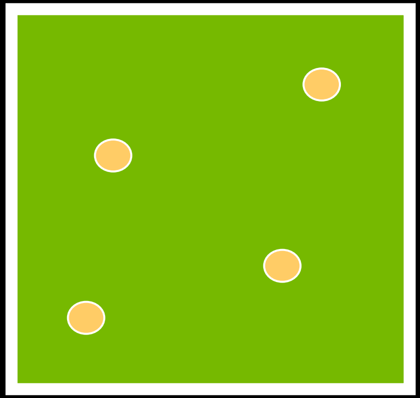
64 bits/pixel

2x multisampling
2 samples/pixel



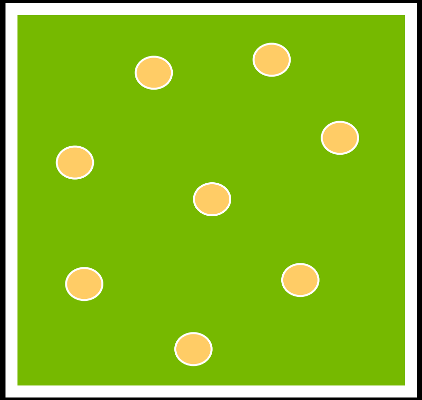
128 bits/pixel

4x multisampling
4 samples/pixel



256 bits/pixel

8x multisampling
8 samples/pixel

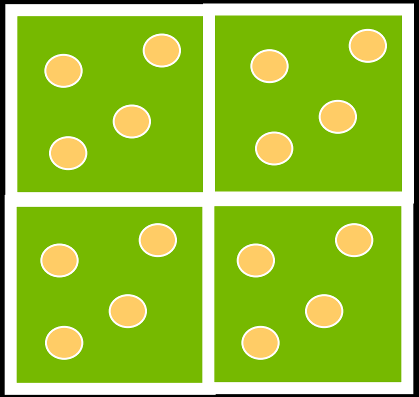


512 bits/pixel

Assume: 32-bit RGBA + 24-bit Z + 8-bit Stencil = 64 bits/sample

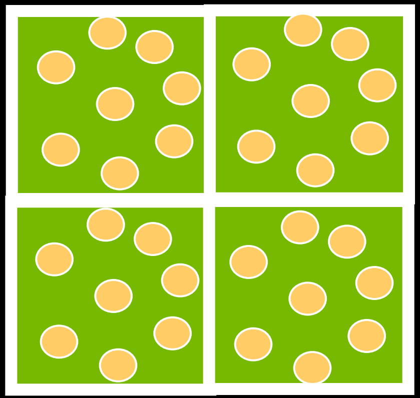
Supersampling FSAA Patterns

2x2 supersampling
of 4x multisampling
16 samples/pixel



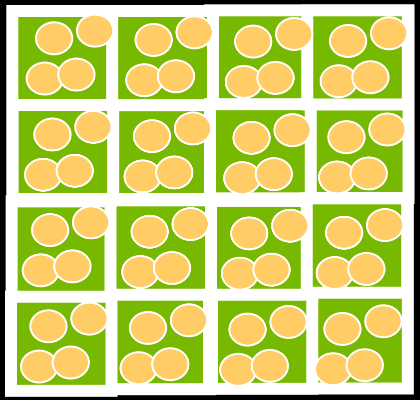
1024 bits/pixel

2x2 supersampling
of 8x multisampling
32 samples/pixel



2048 bits/pixel

4x4 supersampling
of 16x multisampling
64 samples/pixel

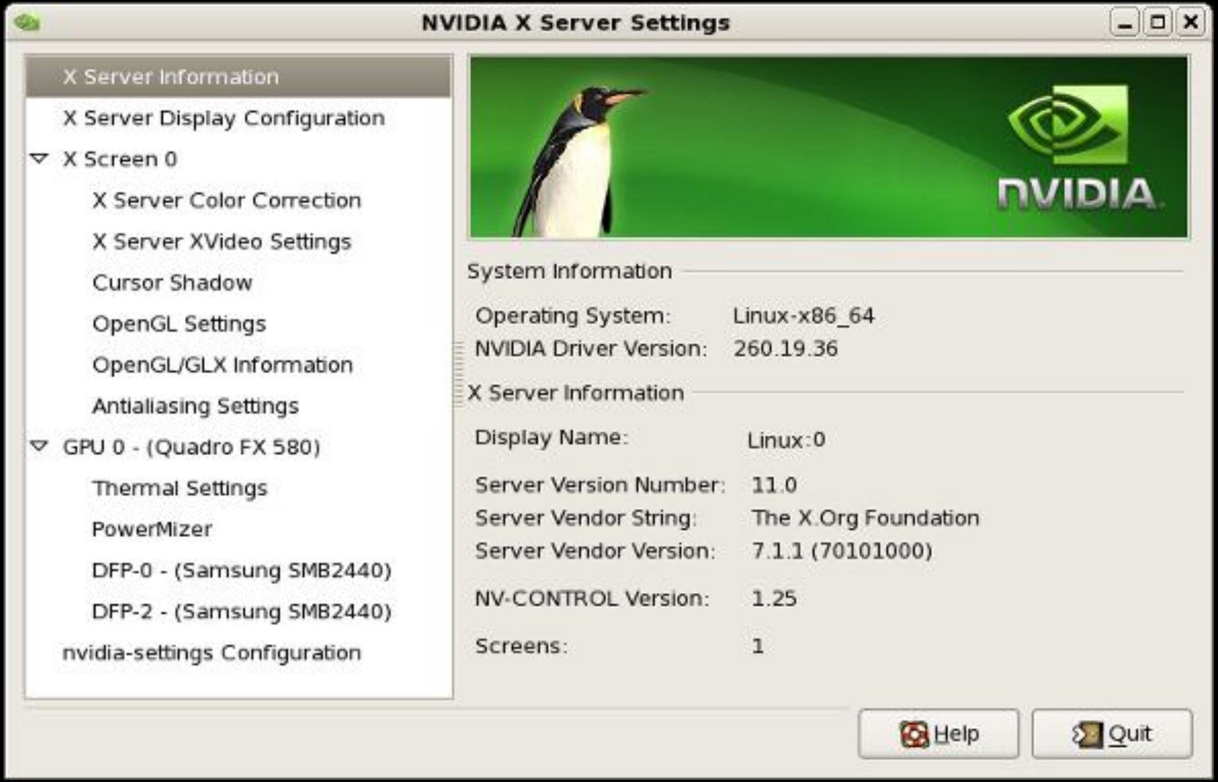


4096 bits/pixel

$\underbrace{\hspace{10em}}_{\text{Quadro GPUs}}$

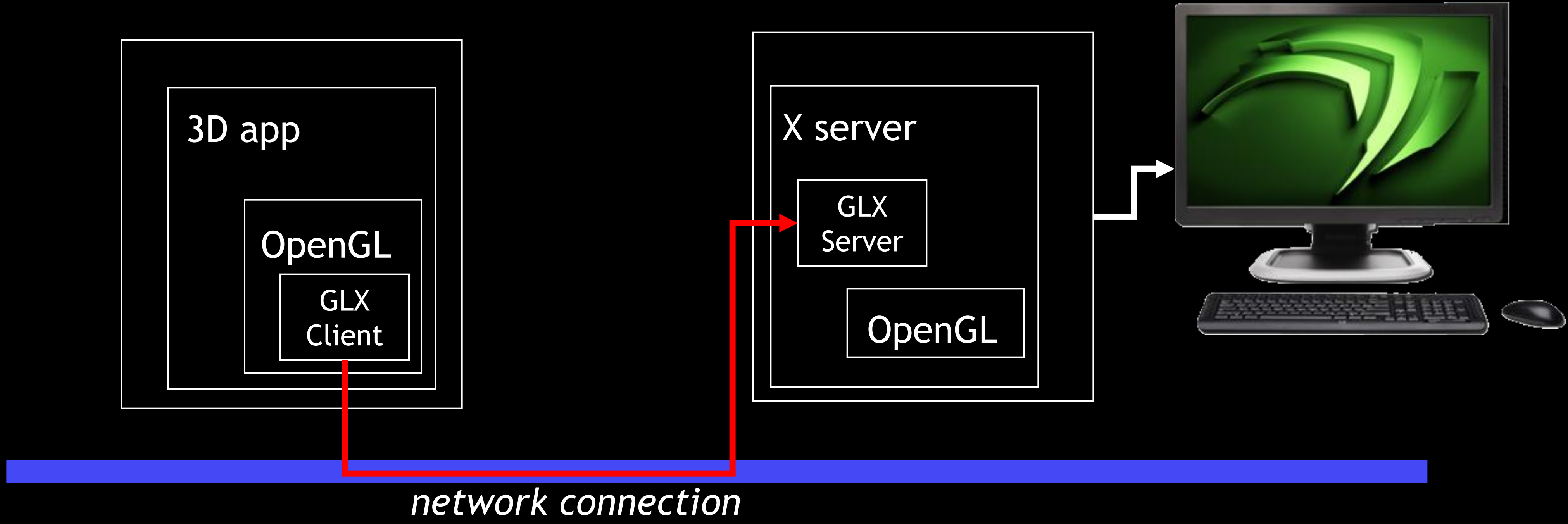
Assume: 32-bit RGBA + 24-bit Z + 8-bit Stencil = 64 bits/sample

NVIDIA X Server Settings for Linux Control Panel



GLX Protocol

- Network transparent OpenGL
  - Run OpenGL app on one machine, display the X and 3D on a different machine



OpenGL-related Linux Improvements

Official GLX Protocol support for OpenGL extensions

- GL_ARB_half_float_pixel
- GL_ARB_transpose_matrix
- GL_EXT_blend_equation_separate
- GL_EXT_depth_bounds_test
- GL_EXT_framebuffer_blit
- GL_EXT_framebuffer_multisample
- GL_EXT_packed_depth_stencil
- GL_EXT_point_parameters
- GL_EXT_stencil_two_side
- GL_NV_copy_image
- GL_NV_half_float
- GL_NV_occlusion_query
- GL_NV_point_sprite
- GL_NV_register_combiners2

OpenGL-related Linux Improvements

Tentative GLX Protocol support for OpenGL extensions

- GL_ARB_map_buffer_range
- GL_ARB_shader_subroutine
- GL_ARB_stencil_two_side
- GL_EXT_texture_integer
- GL_EXT_transform_feedback2
- GL_EXT_vertex_attrib_64bit
- GL_NV_conditional_render
- GL_NV_framebuffer_multisample_coverage
- GL_NV_texture_barrier
- GL_NV_transform_feedback2

Synchronizing X11-based OpenGL Streams

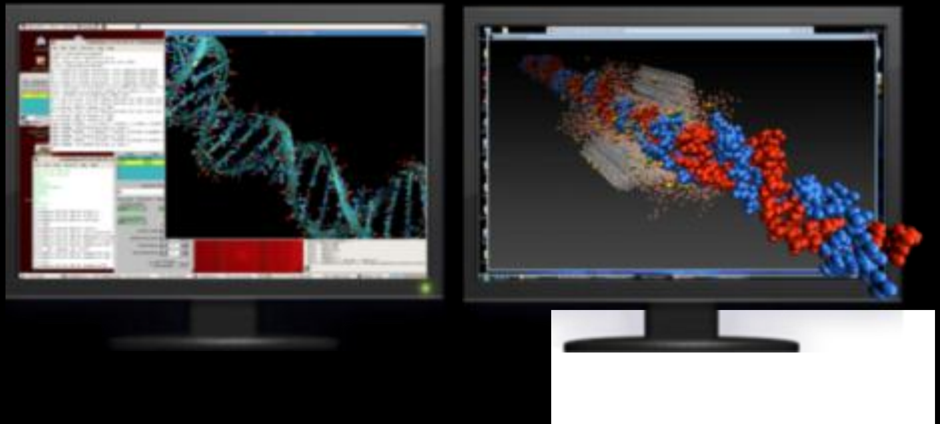
- New extension
  - GL_EXT_x11_sync_object
- Bridges the X Synchronization Extension with OpenGL 3.2 “sync” objects (ARB_sync)
- Introduces new OpenGL command
  - GLintptr sync_handle;
  - GLsync glImportSyncEXT (GLenum *external_sync_type*, GLintptr *external_sync*, GLbitfield *flags*);
    - *external_sync_type* must be GL_SYNC_X11_FENCE_EXT
    - *flags* must be zero

Other Linux Updates

- GL_CLAMP behaves in conformant way now
  - Long-standing work around for original Quake 3
- Enabled 10-bit per component X desktop support
  - GeForce 8 and better GPUs
- Support for 3D Vision Pro stereo now

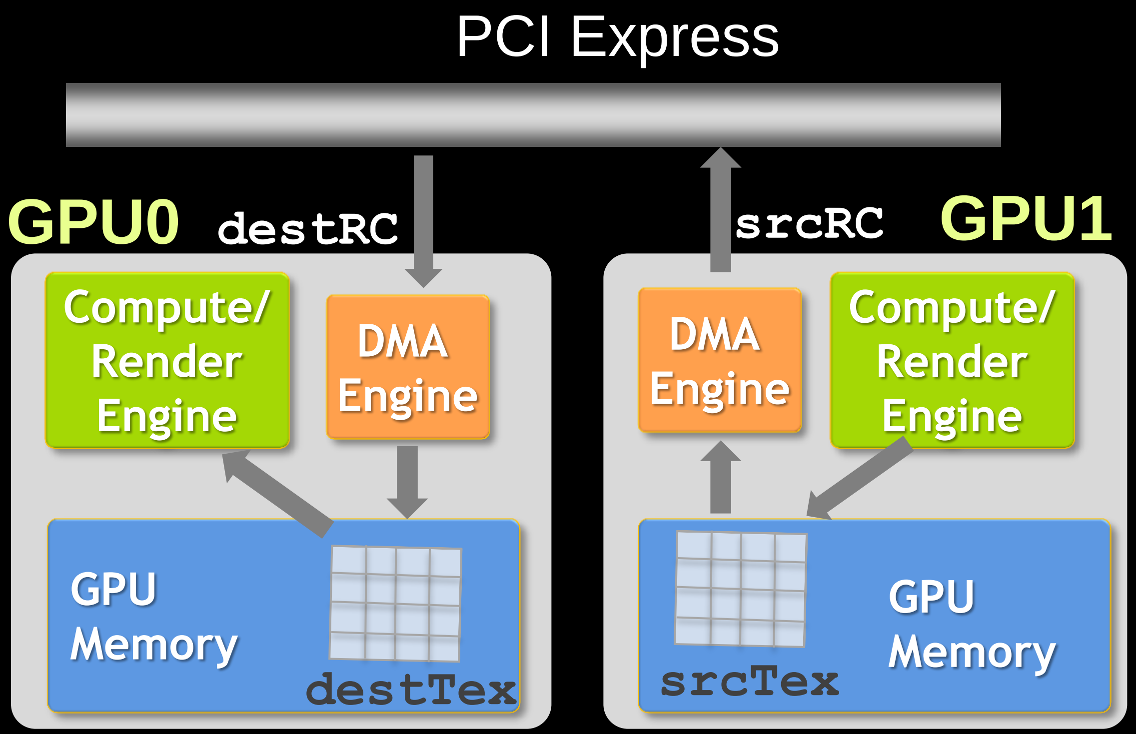
What is 3D Vision Pro?

- For Professionals
- All of 3D Vision support, plus
  - Radio frequency (RF) glasses, Bidirectional
  - Query compass, accelerometer, battery
  - Many RF channels - no collision
  - Up to ~120 feet
  - No line of sight needed to emitter
  - NVAPI to control



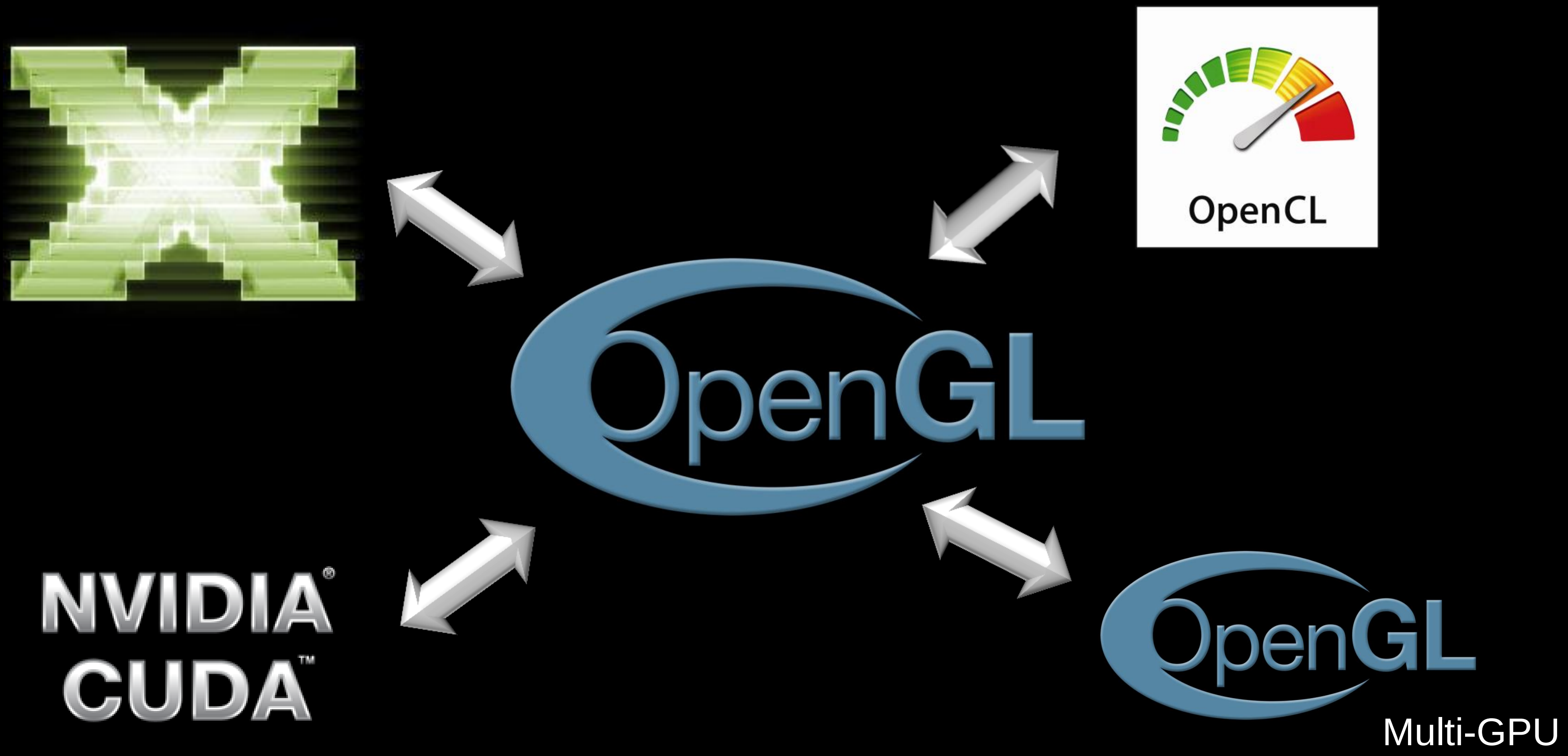
OpenGL Inter-GPU Communication

- NV_copy_image extension
  - Quadro-only
  - (Asynchronous) copy between GPU's
  - Does not require binding textures or state changes
  - App handles synchronization

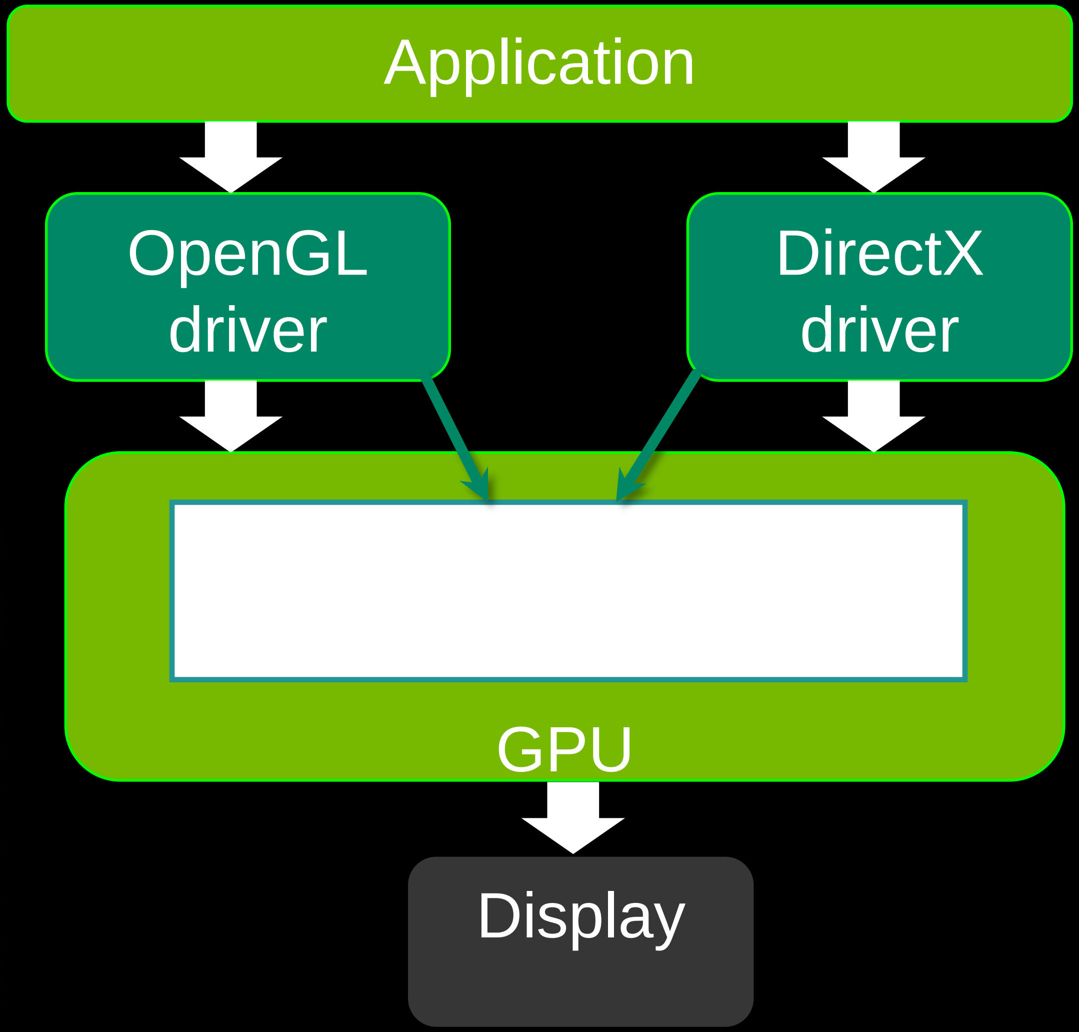



```
wglCopyImageSubDataNV(srcRC, srcTex,
  GL_TEXTURE_2D,0, 0, 0, 0, destRC, destTex,
  GL_TEXTURE_2D, 0, 0, 0, 0,
  width, height, 1);
```

OpenGL Interoperability



DirectX / OpenGL Interoperability



- DirectX interop is a GL API extension (WGL_NVX_dx_interop)
- Premise: create resources in DirectX, get access to them within OpenGL
- Read/write support for DirectX 9 textures, render targets and vertex buffers
- Implicit synchronization is done between Direct3D and OpenGL
- Soon: DirectX 10/11 support

DirectX / OpenGL Interoperability

```
// create Direct3D device and resources the regular way
direct3D->CreateDevice(..., &dxDevice);

dxDevice->CreateRenderTarget(width, height, D3DFMT_A8R8G8B8,
                             D3DMULTISAMPLE_4_SAMPLES, 0,
                             FALSE, &dxColorBuffer, NULL);

dxDevice->CreateDepthStencilSurface(width, height, D3DFMT_D24S8,
                                    D3DMULTISAMPLE_4_SAMPLES, 0,
                                    FALSE, &dxDepthBuffer, NULL);
```

DirectX / OpenGL Interoperability

```
// Register DirectX device for OpenGL interop
HANDLE gl_handleD3D = wglDXOpenDeviceNVX(dxDevice);

// Register DirectX render targets as GL texture objects
GLuint names[2];
HANDLE handles[2];

handles[0] = wglDXRegisterObjectNVX(gl_handleD3D, dxColorBuffer,
               names[0], GL_TEXTURE_2D_MULTISAMPLE, WGL_ACCESS_READ_WRITE_NVX);
handles[1] = wglDXRegisterObjectNVX(gl_handleD3D, dxDepthBuffer,
              names[0], GL_TEXTURE_2D_MULTISAMPLE, WGL_ACCESS_READ_WRITE_NVX);

// Now textures can be used as normal OpenGL textures
```

GPU TECHNOLOGY CONFERENCE

DirectX / OpenGL Interoperability

```
// Rendering example: DirectX and OpenGL rendering to the
// same render target
direct3d_render_pass(); // D3D renders to the render targets as usual

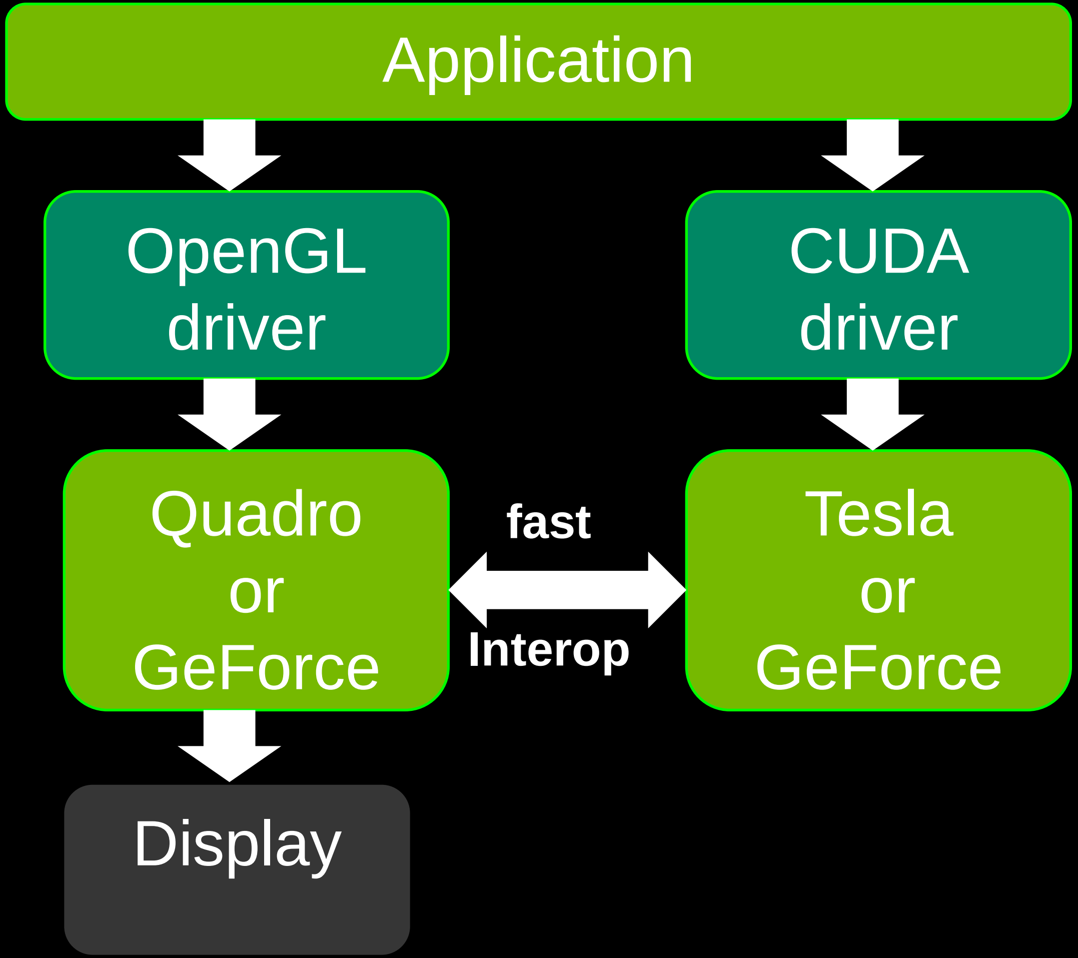
// Lock the render targets for GL access
wglDXLockObjectsNVX(handleD3D, 2, handles);

opengl_render_pass(); // OpenGL renders using the textures as render
                      // targets (e.g., attached to an FBO)

wglDXUnlockObjectsNVX(handleD3D, 2, handles);

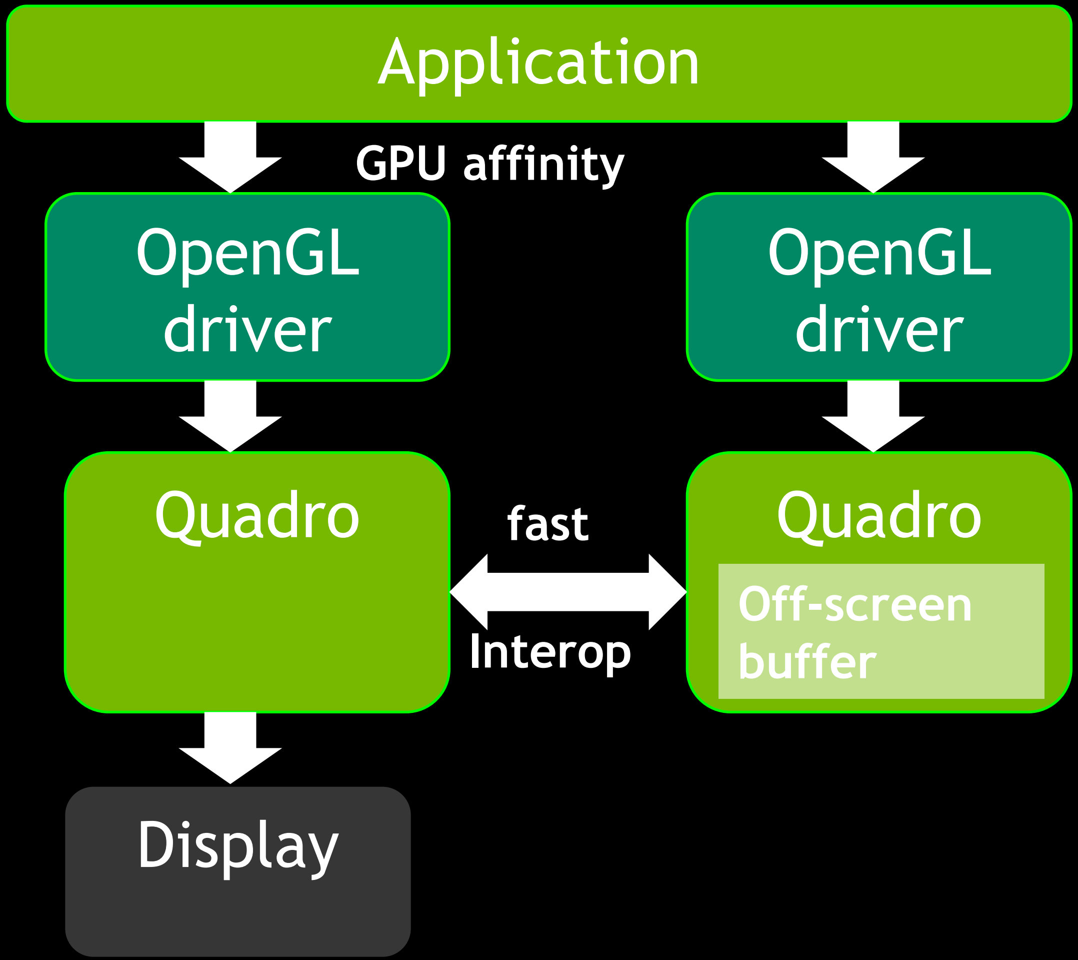
direct3d_swap_buffers(); // Direct3D presents the results on the screen
```

CUDA Interoperability example



- Interop APIs are extensions to OpenGL and CUDA
- Multi-card interop 2x faster on Quadro / Tesla. Transfer between cards, minimal CPU involvement
- GeForce requires CPU copy

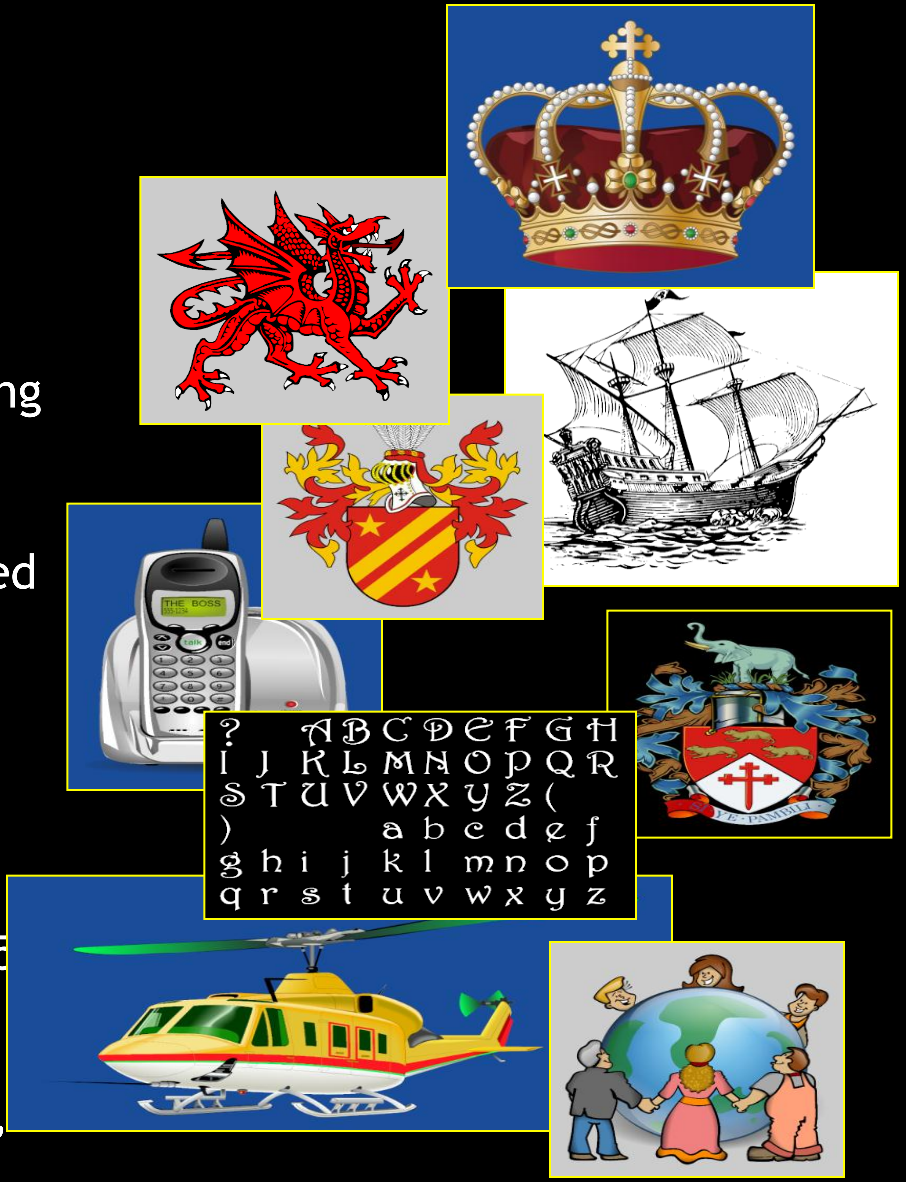
Multi-GPU OpenGL Interoperability



- Interop using NV_copy_image OpenGL extension
- Transfer directly between cards, minimal CPU involvement
- Quadro only

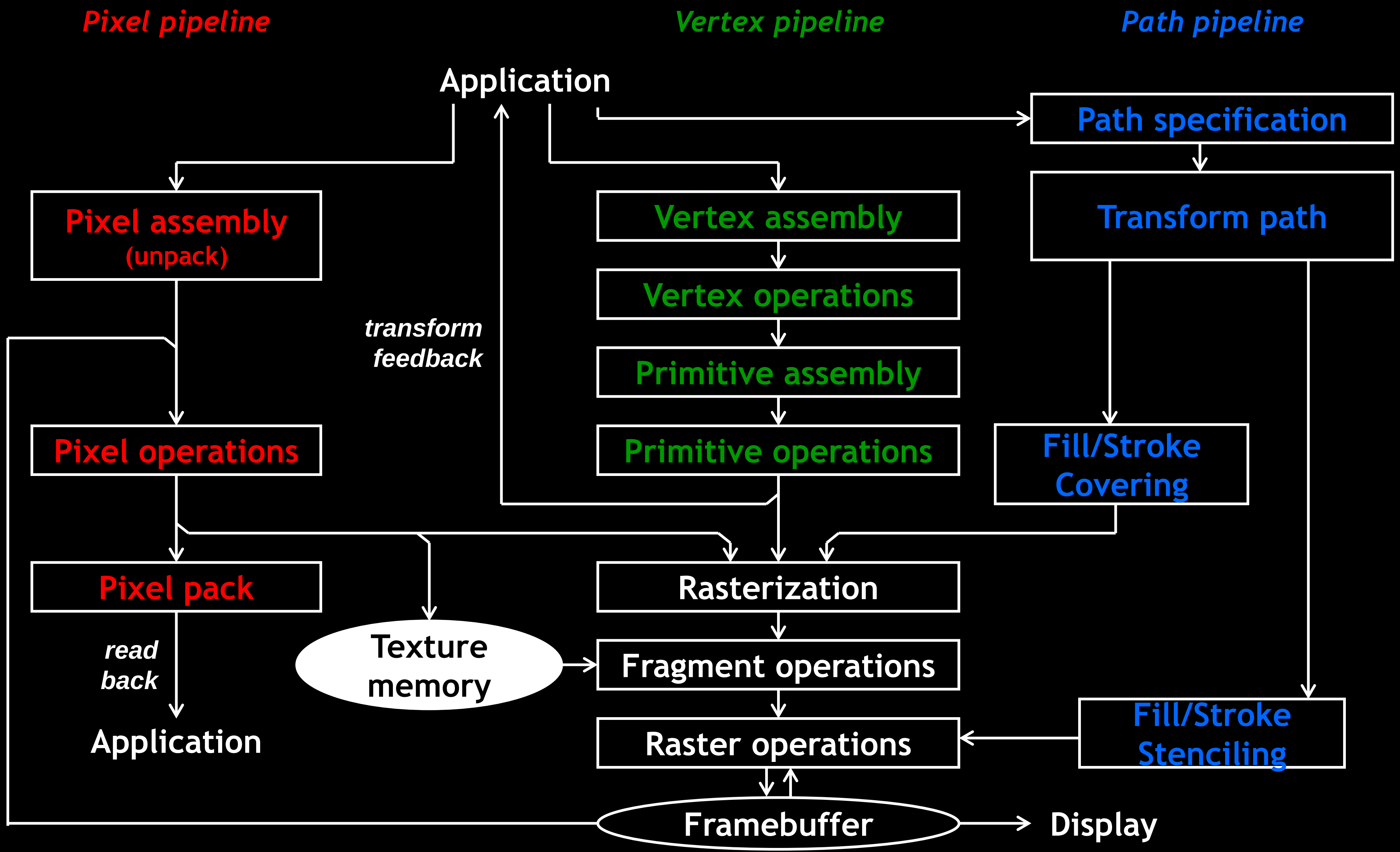
What is path rendering?

- 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.
- Standards
  - Content: PostScript, PDF, TrueType fonts, Flash, Scalable Vector Graphics (SVG), HTML5 Canvas, Silverlight, Office drawings
  - APIs: Apple Quartz 2D, Khronos OpenVG, Microsoft Direct2D, Cairo, Skia, Qt::QPainter, Anti-grain Graphics,



What is NV_path_rendering?

- OpenGL extension to GPU-accelerate path rendering
- Uses "stencil, then cover" (StC) approach
  - Create a path object
  - Step 1: "Stencil" the path object into the stencil buffer
    - GPU provides fast stenciling of filled or stroked paths
  - Step 2: "Cover" the path object and stencil test against its coverage stenciled by the prior step
    - Application can configure arbitrary shading during the step
  - More details later
- Supports the union of functionality of all major path rendering standards
  - Includes all stroking embellishments
  - Includes first-class text and font support
  - Allows this functionality to mix with traditional 3D and programmable shading



Pixel pipeline
Vertex pipeline
Path pipeline
Application
Path specification
Pixel assembly
(unpack)
Vertex assembly
Transform path
Vertex operations
transform
feedback
Primitive assembly
Pixel operations
Primitive operations
Fill/Stroke
Covering
Pixel pack
Rasterization
read
back
Texture
memory
Fragment operations
Application
Raster operations
Fill/Stroke
Stenciling
Framebuffer
Display

Talk on Tuesday

- "GPU-Accelerated Path Rendering"
  - Tuesday, May 15
  - 14:00-14:50, Room A3
- Teaser scene

GPU TECHNOLOGY CONFERENCE

Bindless Graphics

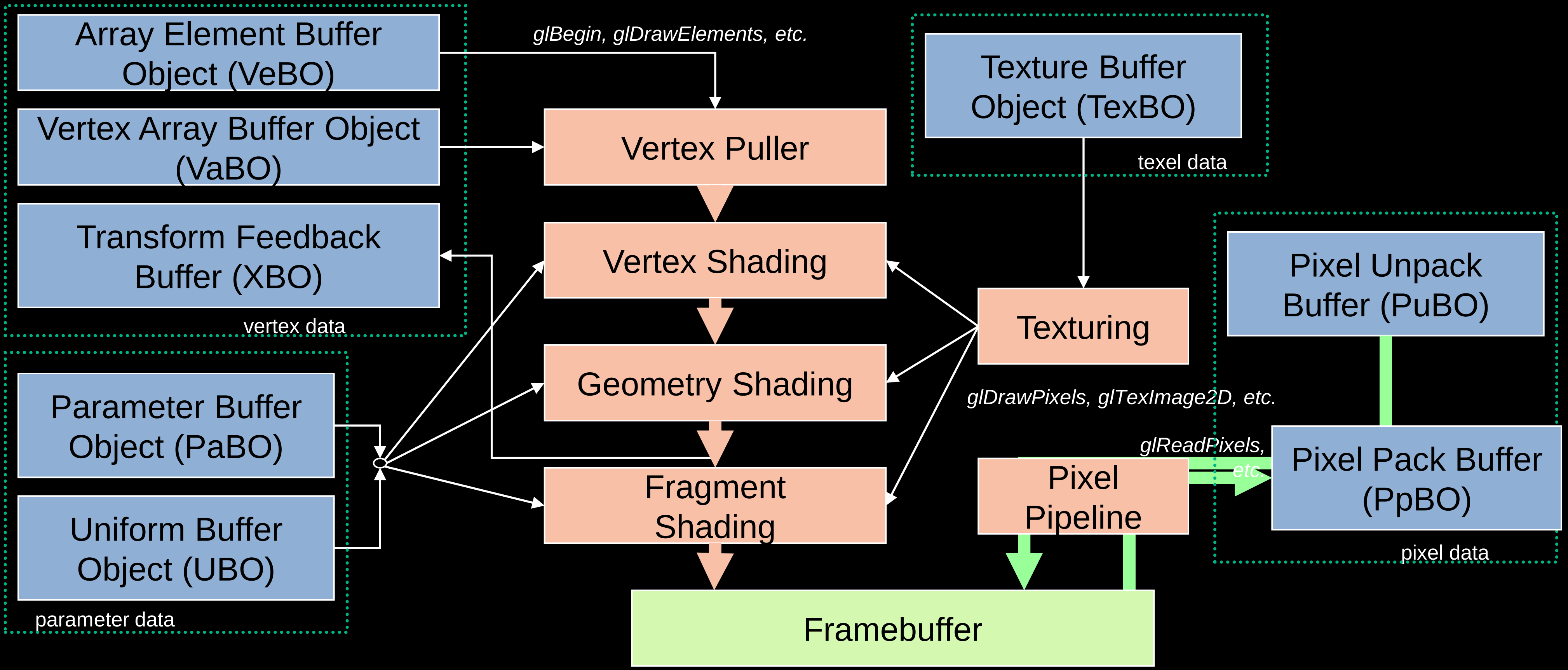
- Problem: Binding to different objects (textures, buffers) takes a lot of validation time in driver
  - And applications are limited to a small palette of bound buffers and textures
  - Approach of OpenGL, but also Direct3D
- Solution: Exposes GPU virtual addresses
  - Let shaders and vertex puller access buffer and texture memory by its virtual address!

Prior to Bindless Graphics

- Traditional OpenGL
  - GPU memory reads are "indirected" through bindings
    - Limited number of texture units and vertex array attributes
  - glBindTexture—for texture images and buffers
  - glBindBuffer—for vertex arrays

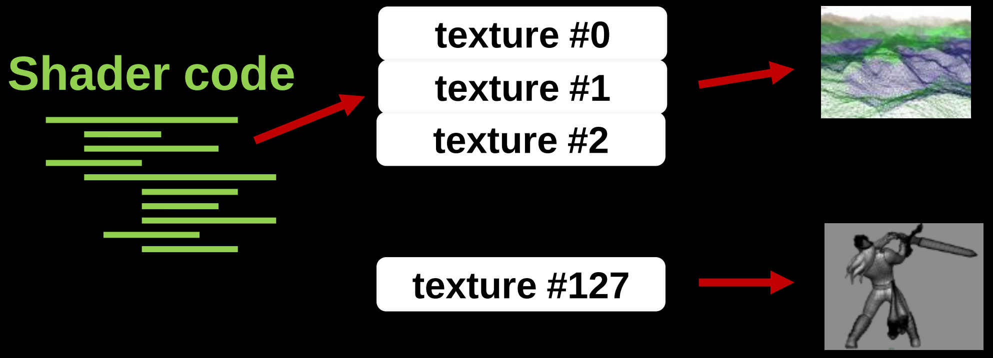
Buffer-centric Evolution

- Data moves onto GPU, away from CPU
  - – Apps on CPUs just too slow at moving data otherwise

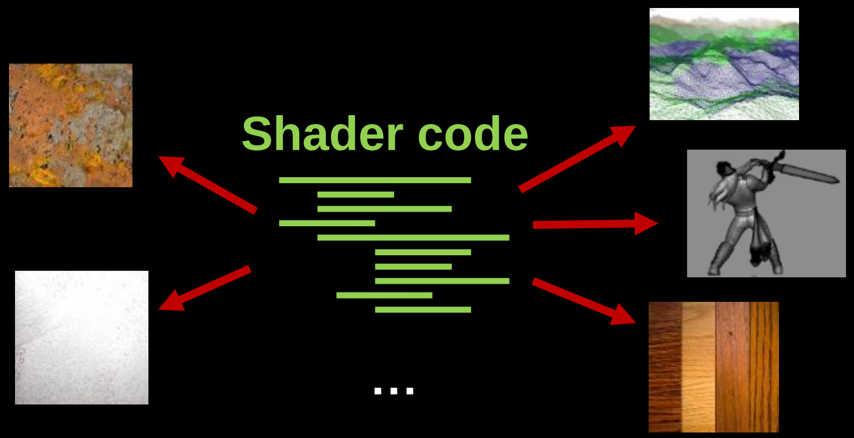


Kepler - Bindless Textures

- Enormous increase in the number of unique textures available to shaders at run-time
- More different materials and richer texture detail in a scene



Pre-Kepler texture binding model



Kepler bindless textures
over 1 million unique textures

Kepler - Bindless Textures

GPU TECHNOLOGY CONFERENCE

Pre-Kepler texture binding model

CPU

Load texture A
Load texture B
Load texture C
Bind texture A to slot I
Bind texture B to slot J
Draw()

GPU

Read from texture at slot I
Read from texture at slot J

CPU

Bind texture C to slot K
Draw()

GPU

Read from texture at slot K

Kepler bindless textures

CPU

Load textures A, B, C
Draw()

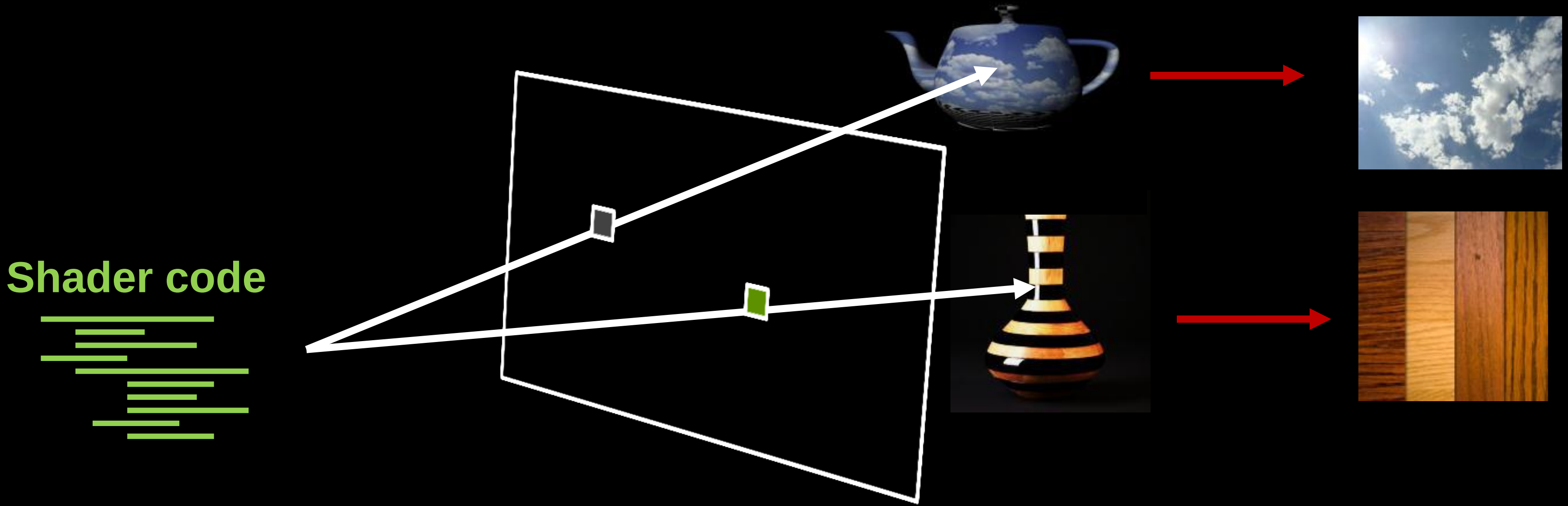
GPU

Read from texture A
Read from texture B
Read from texture C

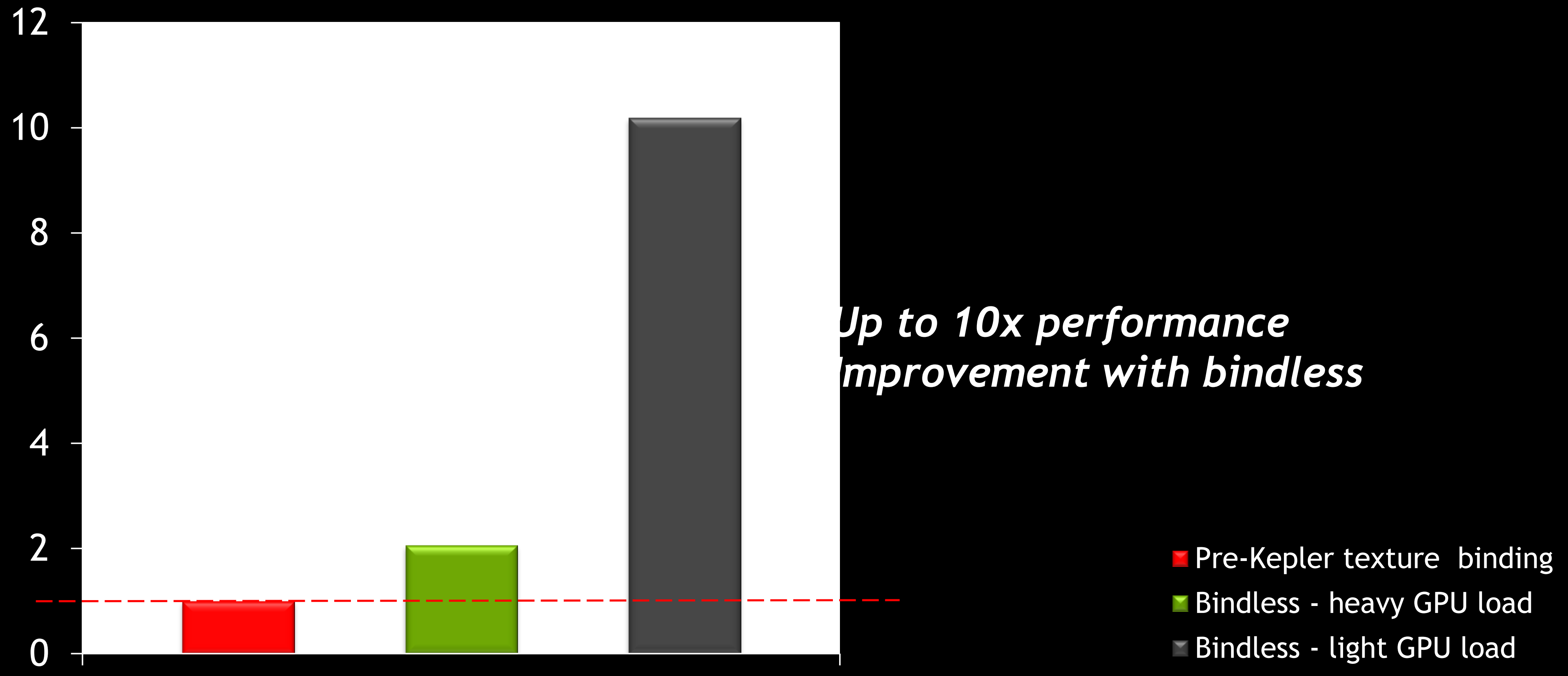
Bindless model reduces CPU overhead and improves GPU access efficiency

Bindless Textures

- Apropos for ray-tracing and advanced rendering where textures cannot be "bound" in advance



Bindless performance benefit



Up to 10x performance improvement with bindless

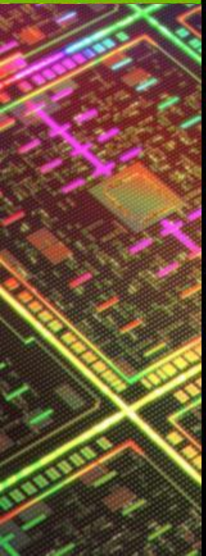
Numbers obtained with a directed test

More Information on Bindless Texture

- Kepler has new NV_bindless_texture extension
  - Texture companion to
    - NV_vertex_buffer_unified_memory for bindless vertex arrays
    - NV_shader_buffer_load for bindless shader buffer reads
- API specification publically available
  - http://developer.download.nvidia.com/opengl/specs/GL_NV_bindless_texture.txt

API Usage to Initialize Bindless Texture

- Make a conventional OpenGL texture object
  - With a 32-bit GLuint name
- Query a 64-bit texture handle from 32-bit texture name
  - GLuint64 glGetTextureHandleNV(GLuint);
- Make handle resident in context's GPU address space
  - void glMakeTextureHandleResidentNV(GLuint64);



Writing GLSL for Bindless Textures

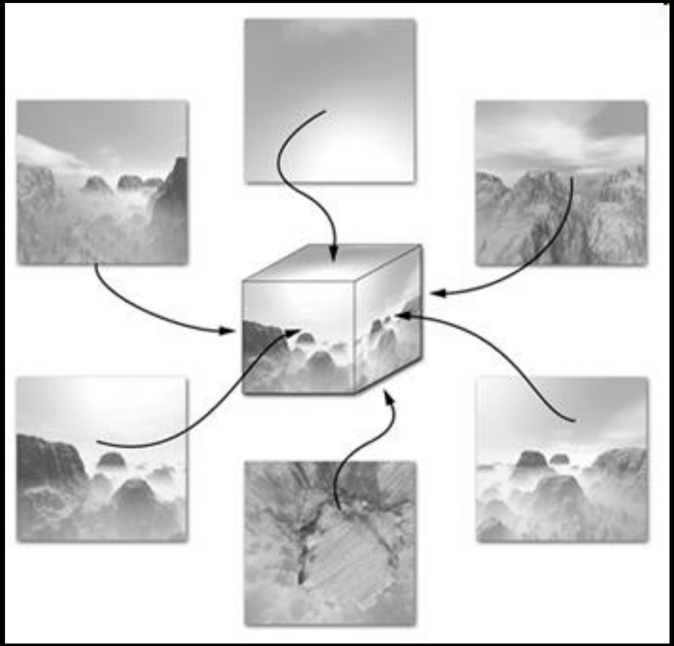
- Request GLSL to understand bindless textures
  - #version 400 // or later
  - #extension GL_NV_bindless_texture : require
- Declare a sampler in the normal way
  - in sampler2D bindless_texture;
- Alternatively, access bindless samplers in big array:
  - ```
    uniform Samplers {
      sampler2D lotsOfSamplers[256];
    }
    ```
  - Exciting: 256 samplers exceeds the available texture units!
    ```

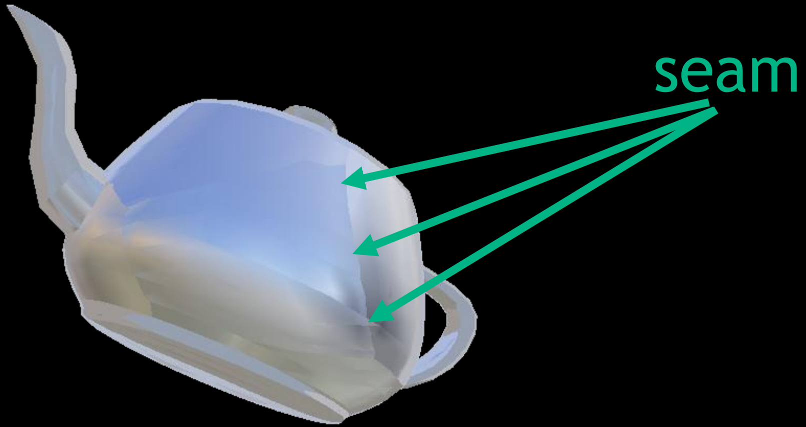
Update Sampler Uniforms with Bindless Texture Handle

- Get a location for a sampler or image uniform
  - GLint loc = glGetUniformLocation(program, “bindless_texture”);
  - GLint loc_array = glGetUniformLocation(program, “lotsOfSamplers”);
- Then set sampler to the bindless texture handle
  - glProgramUniformHandleui64NV(program, location, 1, &bindless_handle);

Seam-free Cube Map Edges

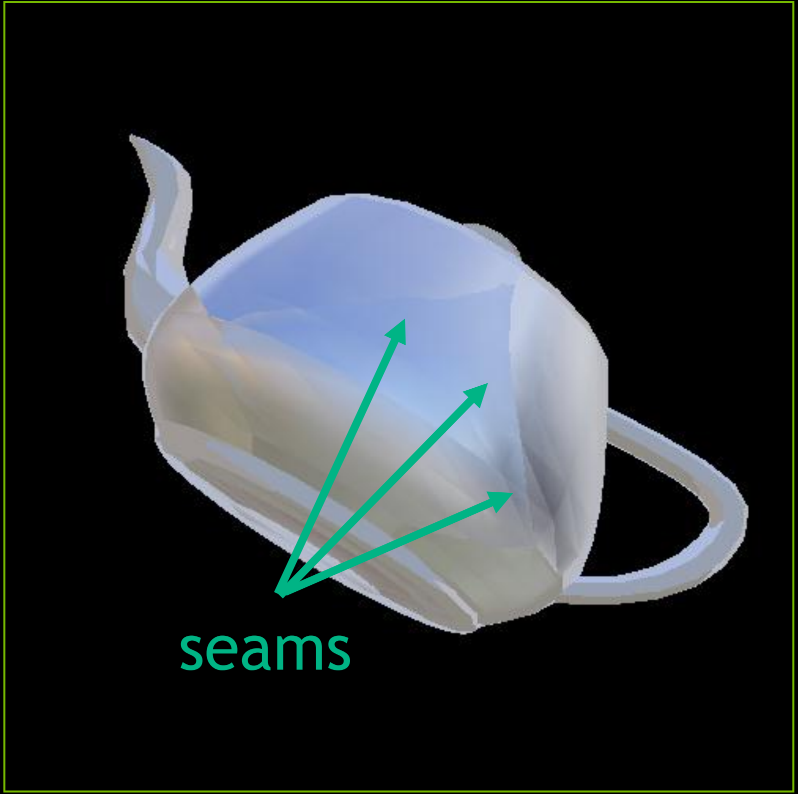
- Cube maps have edges along each face
  - Traditionally texture mapping hardware simply clamps to these seam edges
- Results in "seam" artifacts
  - Particularly when level-of-detail bias is large
    - Meaning very blurry levels
    - But seams appear sharply
- Use glEnable(GL_TEXTURE_CUBE_MAP_SEAMLESS) to mitigate these artifacts on context-wide basis
  - Applies to all cube maps





Seamless Cube Maps: Before and After

- Before: with edge seams



- After: without

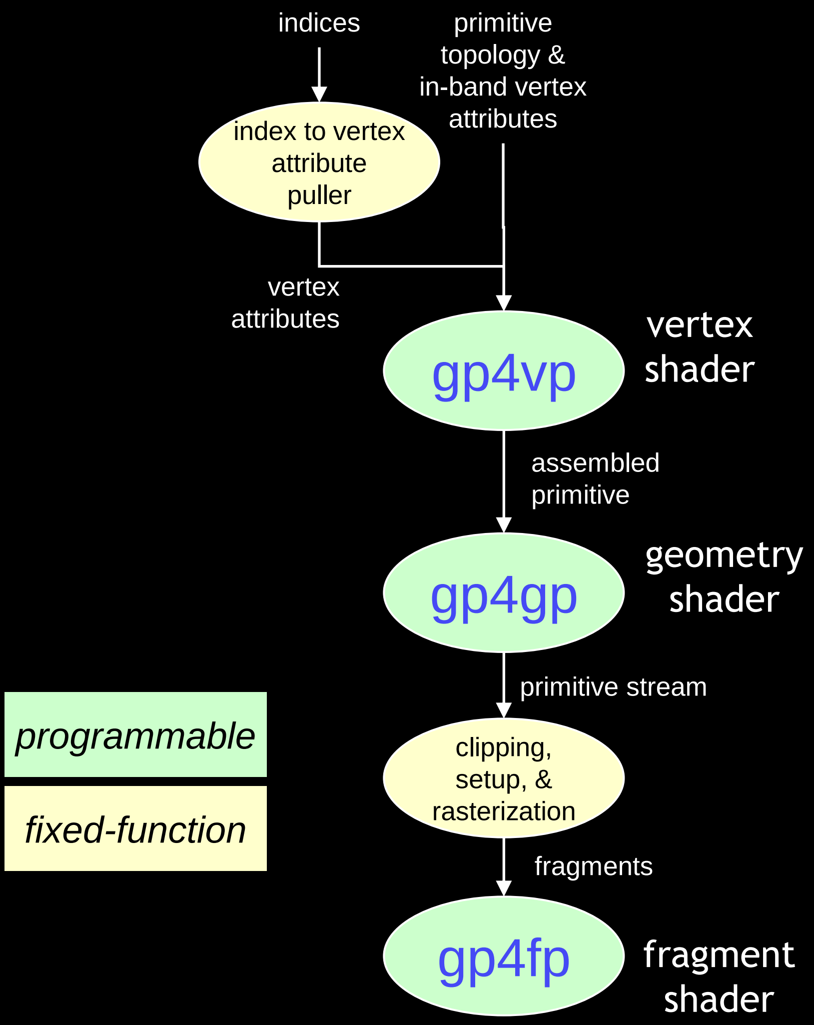
Kepler Provides Per-texture Seamless Cube Map Support

- Kepler GPUs support
  - GL_AMD_seamless_cubemap_per_texture
  - Provides per-texture object parameters
    - glTexParameteriv(GL_TEXTURE_2D,
      GL_TEXTURE_CUBE_MAP_SEAMLESS_ARB, GL_TRUE);
  - Not sure if you want a mix of seamless and seamed cube maps…
    - …but supported anyway

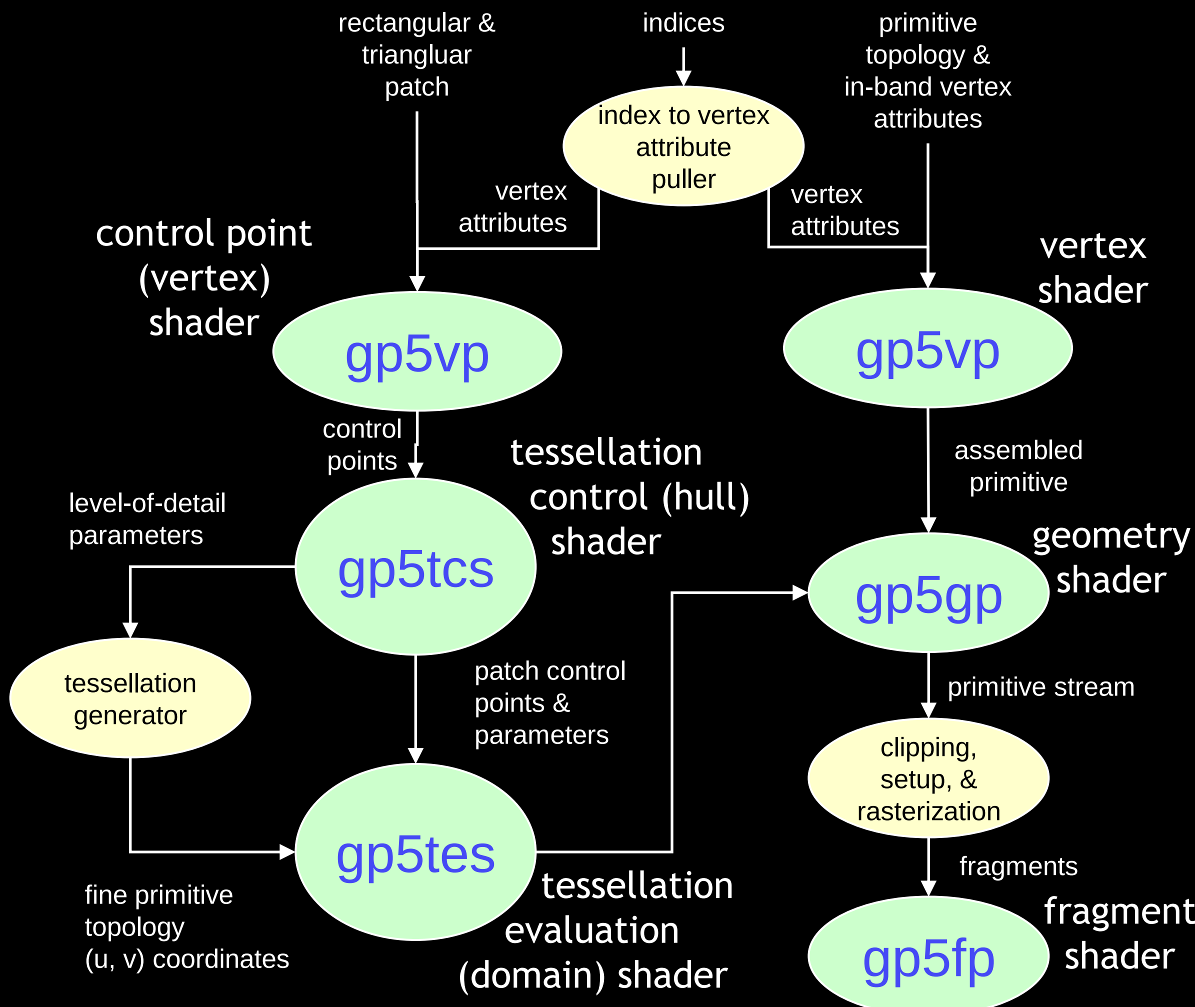
Cg Toolkit 3.1 Update

- Cg 3.0 = big upgrade
  - Introduced Shader Model 5.0 support for OpenGL
    - New gp5vp, gp5gp, and gp5fp profiles correspond to Shader Model 5.0 vertex, geometry, and fragment profiles
      - Compiles to assembly text for the NV_gpu_program5 assembly-level shading language
  - Also programmable tessellation profiles
    - gp5tcp = NV_gpu_program5 tessellation control program
    - gp5tep = NV_gpu_program5 tessellation evaluation program
  - Also DirectX 11 profiles
- Cg 3.1 refines 3.0 functionality
  - Bug fixes, constant buffer support, better GLSL support

Cg 3.1 OpenGL Shader Model 5.0 Profiles



Cg 2.x support



Cg 3.x adds programmable tessellation

Cg 3.1 profiles

Shader domain	OpenGL Shader Model 4.0	OpenGL Shader Model 5.0	OpenGL Shading Language (GLSL)	DirectX 10	DirectX 11
Vertex (or control point)	gp4vp	gp5vp	glslv	vs_4_0	vs_5_0
Tessellation control (hull)	n/a	gp5tcp	n/a	n/a	hs_5_0
Tessellation evaluation (domain)	n/a	gp5tep	n/a	n/a	ds_5_0
Geometry	gp5gp	gp5gp	glslg	gs_4_0	gs_5_0
Fragment	gp4fp	gp5fp	glslf	ps_4_0	ps_5_0

Cg 3.1 Update

- Various improvements & bug fixes
  - Constant buffer fixes
  - GLSL translation for different GLSL versions
- Support for CENTROID, FLAT, and NOPERSPECTIVE fragment program interpolants
- Deprecated functionality
  - Removed Direct3D 8 support
  - Minimum Mac OS X version: 10.5 (Leopard)

Cg Off-line Compiler Compiles GLSL to Assembly

- GLSL is “opaque” about the assembly generated
  - Hard to know the quality of the code generated with the driver
  - Cg Toolkit 3.0 is a useful development tool even for GLSL programmers
- Cg Toolkit’s cgc command-line compiler actually accepts both the Cg (cgc’s default) and GLSL (with –oglsl flag) languages
  - Uses the same compiler technology used to compile GLSL within the driver
- Example command line for compiling earlier tessellation control and evaluation GLSL shaders
  - cgc -profile gp5tcp -oglsl -po InputPatchSize=3 -po PATCH_3 filename.glsl (assumes 3 input control points and 3 outputs points)
  - cgc -profile gp5tep -oglsl -po PATCH_3 filename.glsl (assumes 3 input control points)

Questions?

Other OpenGL-related Sessions at GTC

- S0024: GPU-Accelerated Path Rendering
  - Tuesday, 14:00, Room A3
- S0049: Using the GPU Direct for Video API
  - Tuesday, 15:00, Room J8
- S0356: Optimized Texture Transfers
  - Tuesday, 16:00, Room J2
- S0267A: Mixing Graphics and Compute with Multiple GPUs
  - Tuesday, 17:00, Room J2
- S0353: Programming Multi-GPUs for Scalable Rendering
  - Wednesday, 9:00, Room A1
- S0267B - Mixing Graphics and Compute with Multiple GPUs
  - Thursday, 17:30, Room L