3D Graphics API State of the Union

SIGGRAPH 2015

Outline

- **The Khronos 3D Ecosystem**
  - Neil Trevett – Khronos president, NVIDIA VP Mobile Ecosystem
- **Khronos data formats specification**
  - Andrew Garrard – Specification editor
- **What's new in OpenGL**
  - Barthold Lichtenbelt – OpenGL Working Group chair
- **What's new in Open GL ES**
  - Tom Olson – OpenGL ES Working Group chair
- **Vulkan status report**
  - Working Group members

The Khronos 3D Ecosystem

Neil Trevett | Khronos President
NVIDIA Vice President Mobile Ecosystem

Khronos Connects Software to Silicon

Open Consortium creating ROYALTY-FREE, OPEN STANDARD APIs for hardware acceleration

Defining the roadmap for low-level silicon interfaces needed on every platform

Graphics, compute and vision processing

Rigorous specifications AND conformance tests for cross-vendor portability

Acceleration APIs
BY the Industry
FOR the Industry

Well over a *BILLION* people use Khronos APIs
Every Day…

Khronos Open Standards for Graphics and Compute

			SIGGRAPH NEWS
1990's	Workhorse cross-platform professional 3D apps & gaming	OpenGL	New Extensions to enable latest desktop graphics capabilities
2000's	Ubiquitous mobile gaming & graphics apps	OpenGL ES	OpenGL ES 3.2 released today to bring AEP functionality to core
2005	Safety Critical Graphics	OpenGL SC	New Safety Critical Working Group - Call for Participation
2008	Heterogeneous parallel compute	OpenCL	OpenCL 2.0 specification update and C++ Headers released
2014	Portable intermediate representation for graphics and parallel compute	SPIR	Provisional Spec Update and significant open source activity
2015	High-efficiency GPU graphics and compute for performance critical apps	Vulkan	Adopted by Android and other platforms. Building ecosystem

KHRONOS GROUP

Next Generation GPU APIs

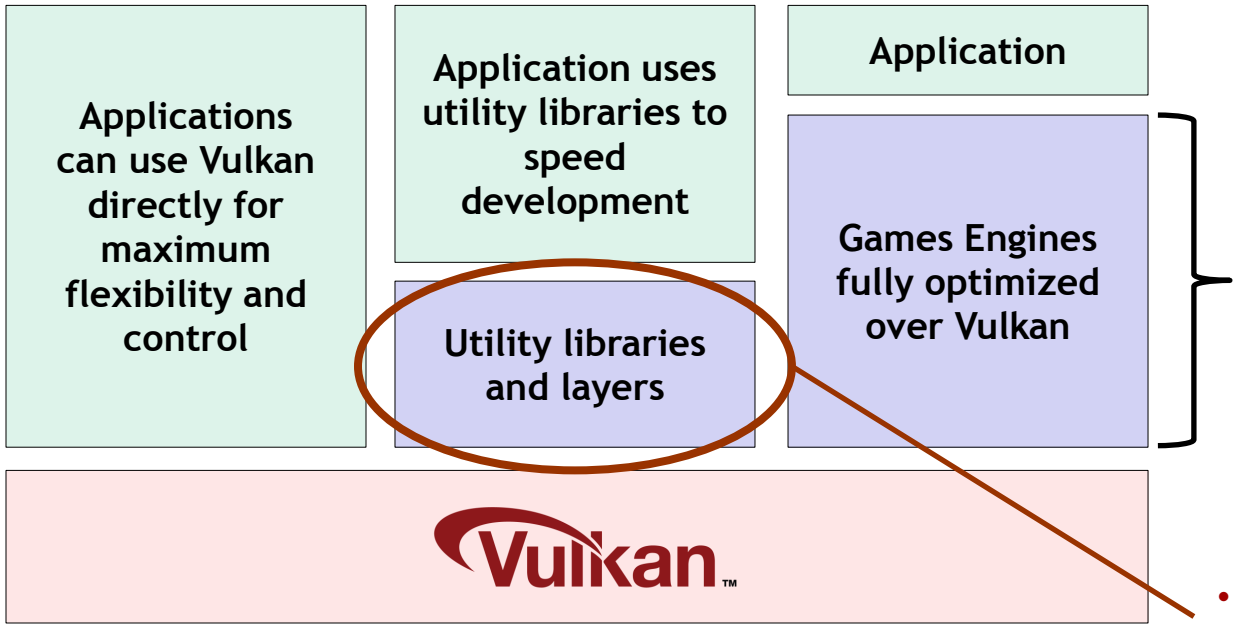
One
OS

One Platform
Vendor

Cross
Platform

No Compromise

The Power of a Three Layer Ecosystem



Developers can choose at which level to use the Vulkan Ecosystem

The industry's leading games and engine vendors are participating in the Vulkan working group

OXIDE · LUCASFILM Ltd · BLIZZARD ENTERTAINMENT · EA · unity · EPIC GAMES · VALVE

Rich Area for Innovation

- Many utilities and layers will be in open source
- Layers to ease transition from OpenGL
- Domain specific flexibility

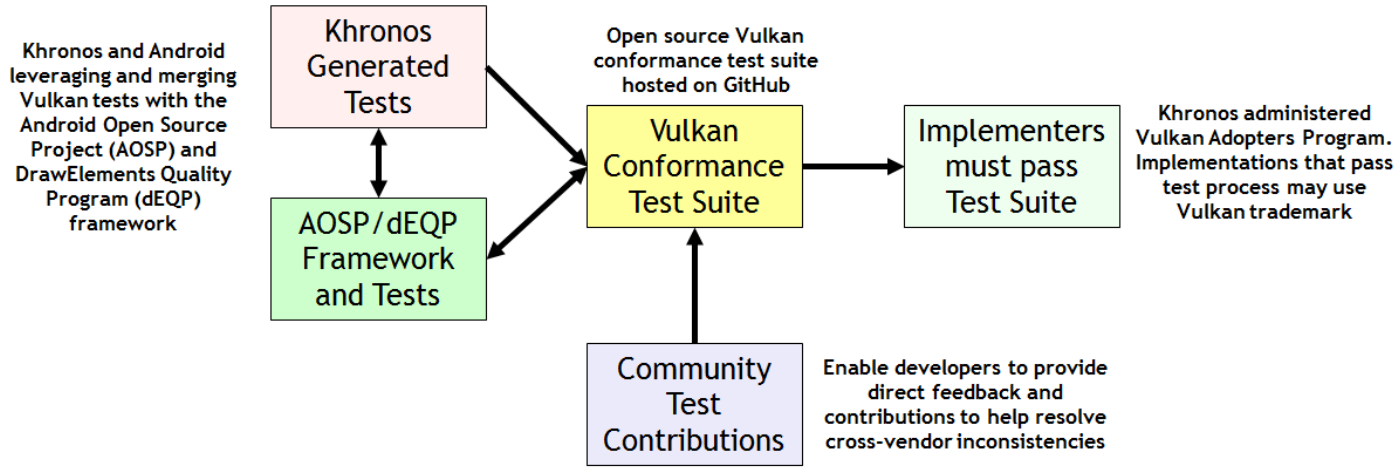
The same ecosystem dynamic as WebGL

A widely pervasive, powerful, flexible foundation layer enables diverse middleware tools and libraries

KHRONOS GROUP

Developing Ecosystem and Spec in Parallel

- Open sourcing Vulkan test suite to enable developer feedback and contributions
- Khronos supplied loader and layered tools architecture
- Open source layered tools – Valve/LunarG are the first
- Flexible Windows System Integration – working with platform vendors
- Example code, documentation and course notes
- SPIR-V for language innovation



SPIR-V Transforms the Language Ecosystem

- **First multi-API, intermediate language for parallel compute and graphics**
  - Native representation for Vulkan shader and OpenCL kernel source languages
  - https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf
- **Cross vendor intermediate representation**
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form

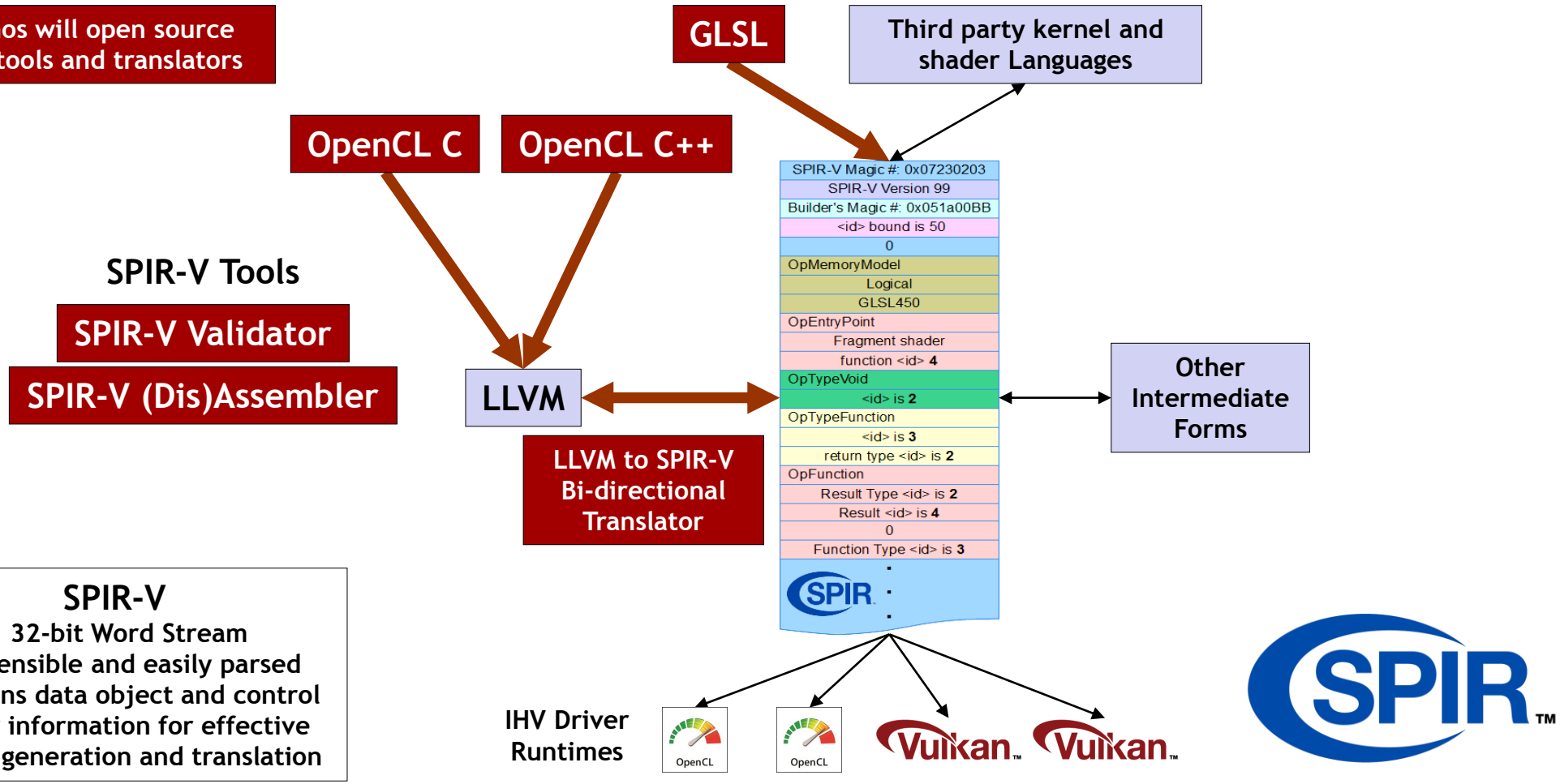
Multiple Developer Advantages

Same front-end compiler for multiple platforms
Reduces runtime kernel compilation time
Don't have to ship shader/kernel source code
Drivers are simpler and more reliable

Driving the SPIR-V Open Source Ecosystem

SPIR-V Provisional spec (V31) updated today!

Khronos will open source these tools and translators



SPIR-V

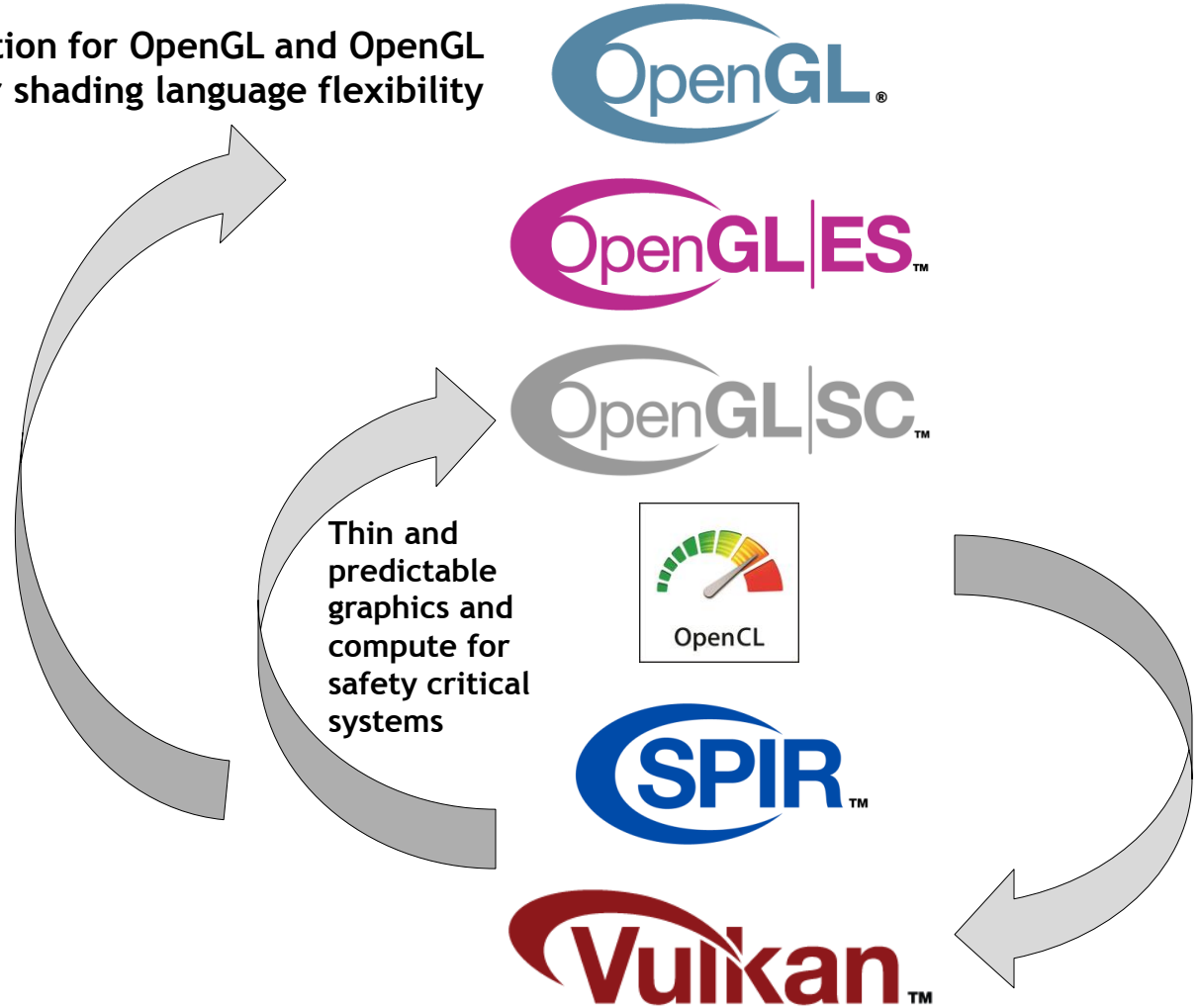
- 32-bit Word Stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

SPIR-V Open Source Community Activity

- **Python byte code to SPIR-V Convertor**
  - Write shaders or kernels in Python, Encode and decode SPIR-V in Python
  - Dis(Assembler) with high level human readable assembler syntax
- **.NET IL to SPIR-V Convertor**
  - Write and debug shaders or kernels using C# , SPIR-V interpreter
- **Shade SPIR-V virtual machine**
  - Test and debug SPIR-V binaries for binary correctness in human readable format
- **Otherside SPIR-V virtual machine**
  - Academic software rasterizer project to produce C code from SPIR-V
- **Rust (Dis)Assembler**
  - Encode and decode SPIR-V binaries in Rust
- **Go (Dis)Assembler**
  - Encode and decode SPIR-V in Go, SPIR-V represented in Go data structures
- **Haskell EDSL**
  - SPIR-V like language embedded in Haskell with significantly relaxed layout constraints
- **Lisp SPIR-V Specification**
  - Lisp readable SPIR-V specification
- **JSON SPIR-V specification**
  - Conversion of HTML SPIR-V specification to JSON format
- **This is just the start….**

Roadmap Possibilities

SPIR-V Ingestion for OpenGL and OpenGL ES for shading language flexibility

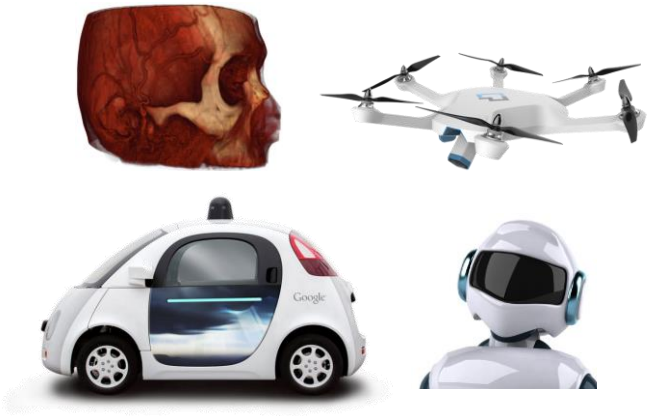


Thin and predictable graphics and compute for safety critical systems

1. C++ Shading Language
2. Single source C++ Programming from SYCL
3. OpenCL-class Heterogeneous Compute to Vulkan runtime

Safety Critical Working Group

Call for participation to create OpenGL SC 2.0 and future safety critical APIs!



OpenGL SC™ — 2005 — OpenGL SC 1.0 — Fixed function graphics subset

OpenGL SC™ — 2016 (planned) — OpenGL SC 2.0 — Programmable shader pipeline subset

OpenGL ES™ — 2003 — OpenGL ES 1.0 — Fixed function graphics

OpenGL ES™ — 2007 — OpenGL ES 2/3 — Programmable shader pipeline

SPIR™

Vulkan™

New Generation API for safety certifiable graphics AND compute

Many future safety critical use cases involve vision and compute acceleration (e.g. neural nets)

Data Format Specification
SIGGRAPH, August 2015

Andrew Garrard | Spec. Editor
Senior Software Engineer, Samsung

Should you ignore me?

- Are you writing a program?
- Does it work with images, textures, buffers, etc?
- Do you have that content in memory?
- Do you need to describe that content to anyone else?
- Do you have more than one type of content?
- Do you need to describe how hardware or software handles this stuff?

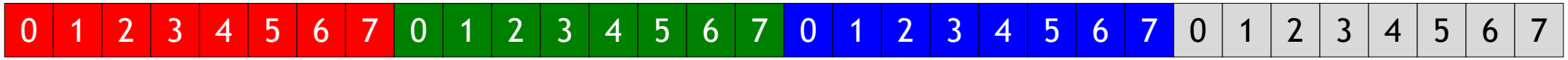
This is about formats

- You've got some bits that correspond to a pixel, or buffer element
- What do they mean?
- "Oh, RGB, obviously…"

When you said that…

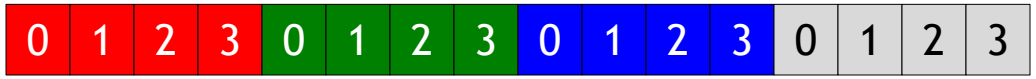
- *What* RGB?

32bpp red, green, blue, alpha 8888



16bpp red, green, blue 565

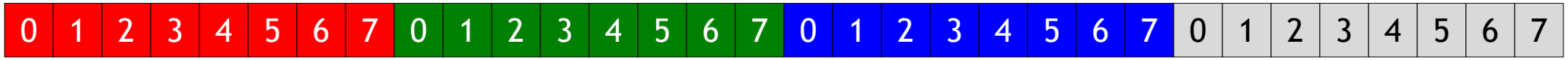
16bpp red, green, blue, alpha 4444



When you said that…

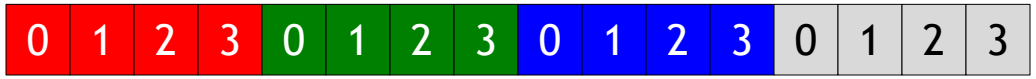
- *What* RGB?

32bpp red, green, blue, alpha 8888



16bpp red, green, blue 565

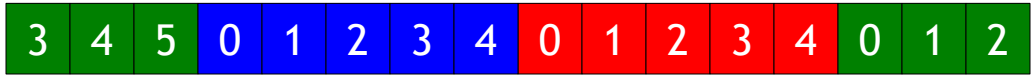
16bpp red, green, blue, alpha 4444



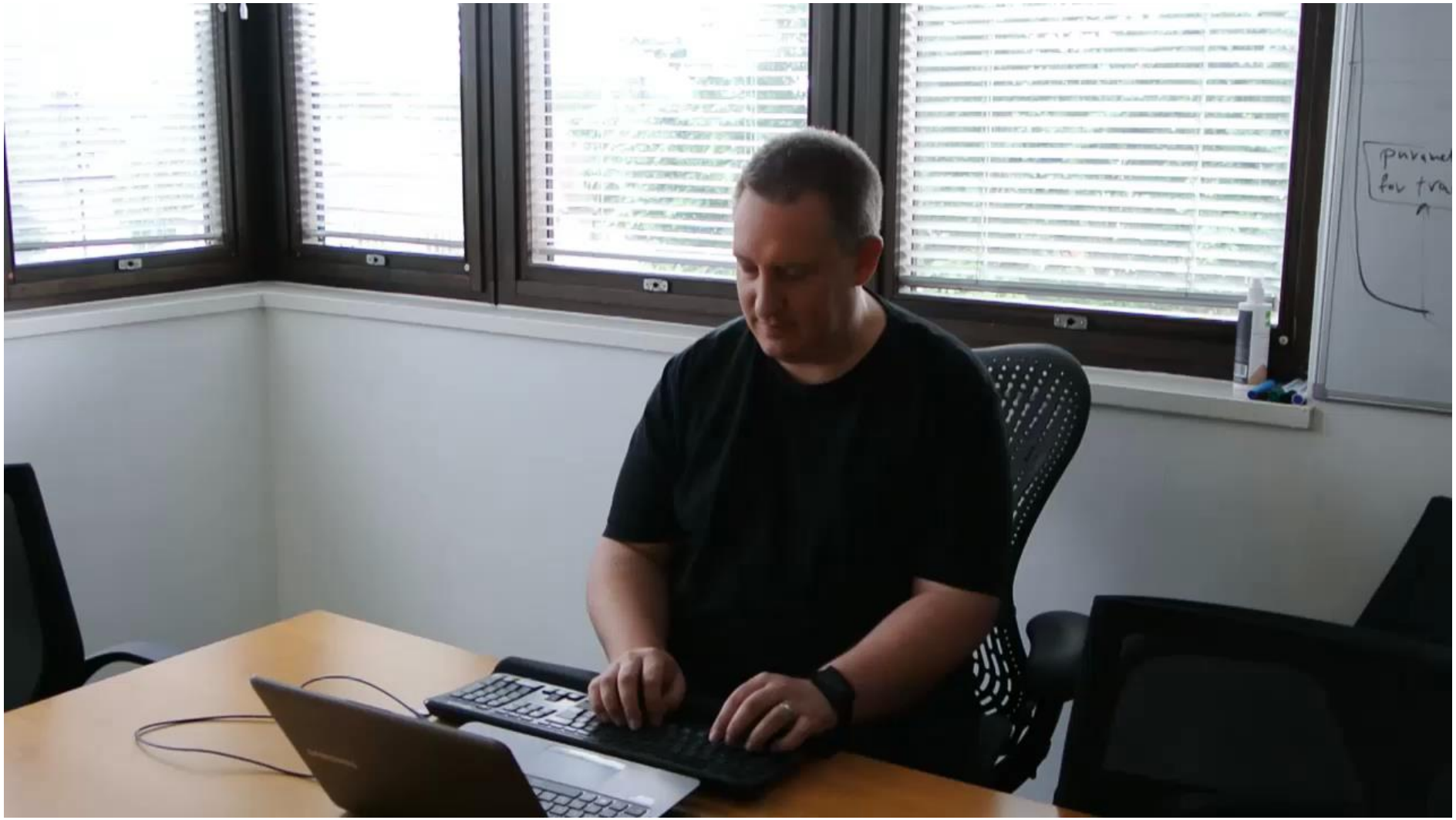
- What order? (Which API's convention?)

32bpp blue, green, red, alpha 8888

16bpp red, green, blue 565, swapped endian



This is when…



Ow. And then…

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”

Ow. And then…

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”
- …range?

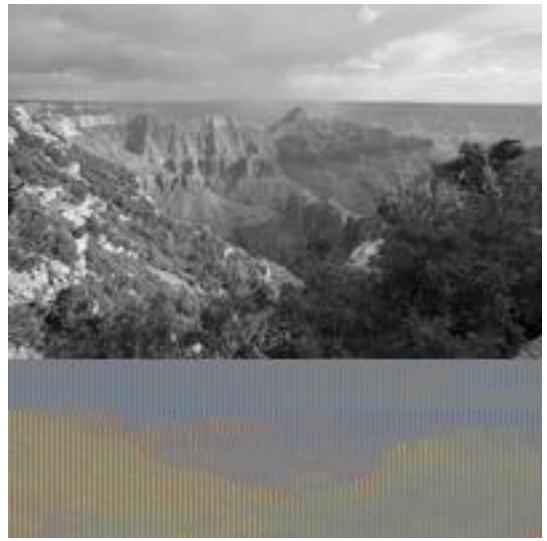
Ow. And then…

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”
- …range?
- …gamma?
- sRGB?
- TV output?
- *Which* TV output?

And if you guess…

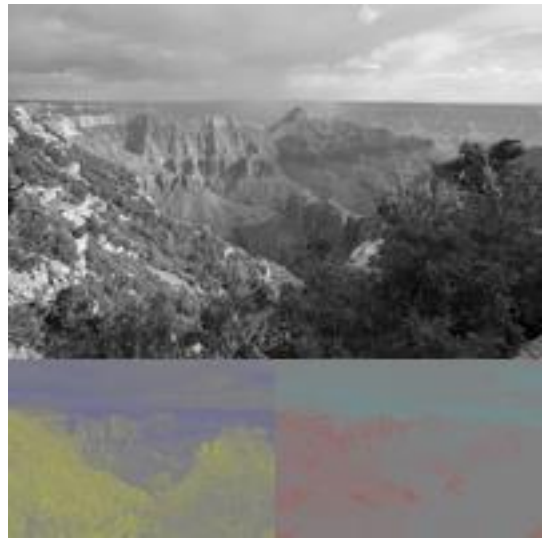
Then you want to output to video

- Ooh, YUV



Then you want to output to video

- Ooh, YUV
- …YUV
- …or used compressed formats

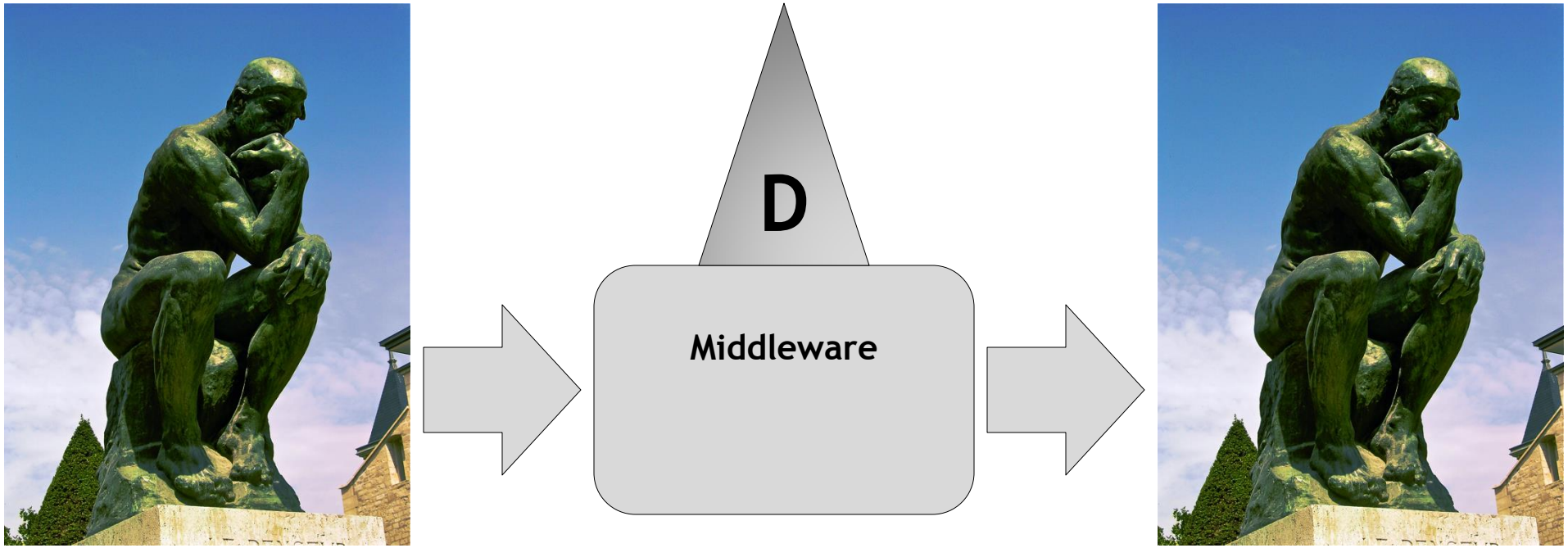


And read from a camera

- Bayer!
- Metadata
- Etc…

When you do have this right…

- You end up with a library that gets confused and data disappears



Application code

Output library

Around this time…

Nothing does this right…

- Describing a format is really easy when you know your problem space
- At some point, you're using something that doesn't
- Everyone rolls their own
- No problems until they have to work together
- Problem spaces aren't as disjoint as you'd think

The Khronos Data Format Specification

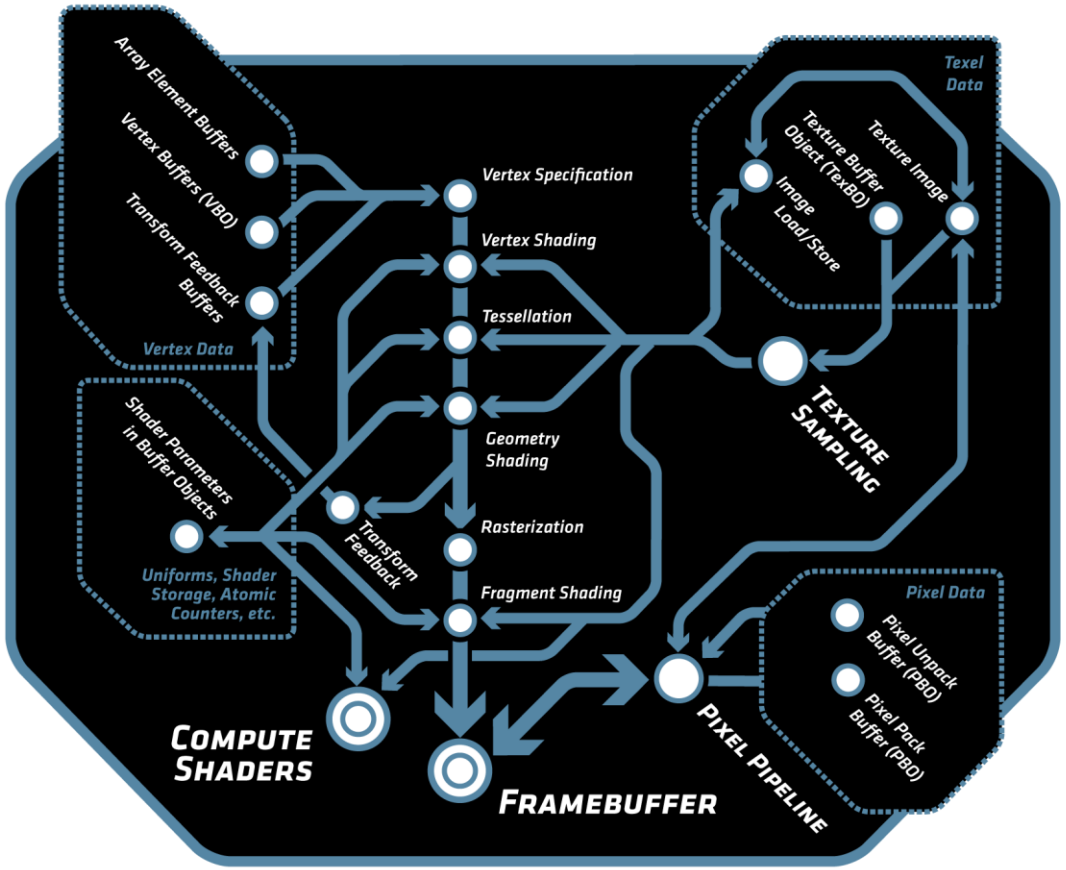
- Just released
- A really dull thing done right so you don't have to think about it
- Descriptive
- Extensible
- Versioned
- Flexible
- Not big
- No conformance
- Not tied to any other Khronos spec
- The press don't understand what it is

- www.khronos.org/dataformat/

Fellow sufferers…

Acute qwertyitis: keyboard rash from impact damage
c.f. chronic qwertyitis, caused by keyboard-as-pillow

Thank you



KHRONOS GROUP

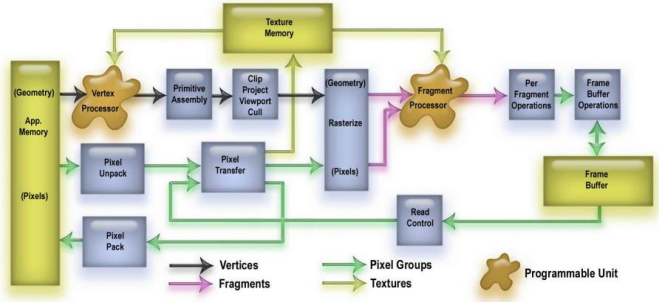
OpenGL®

Siggraph 2015

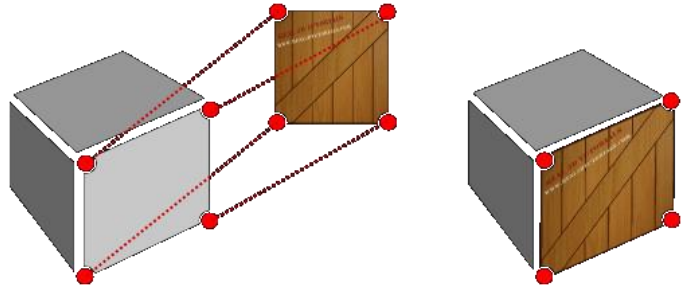
Barthold Lichtenbelt
OpenGL ARB chair

Announcing 13 new OpenGL ARB extensions

- **NEW graphics pipeline operation**
  - 5 ARB extensions



- **NEW texture mapping functionality**
  - 3 ARB extensions



- **NEW shader functionality**
  - 6 ARB extensions

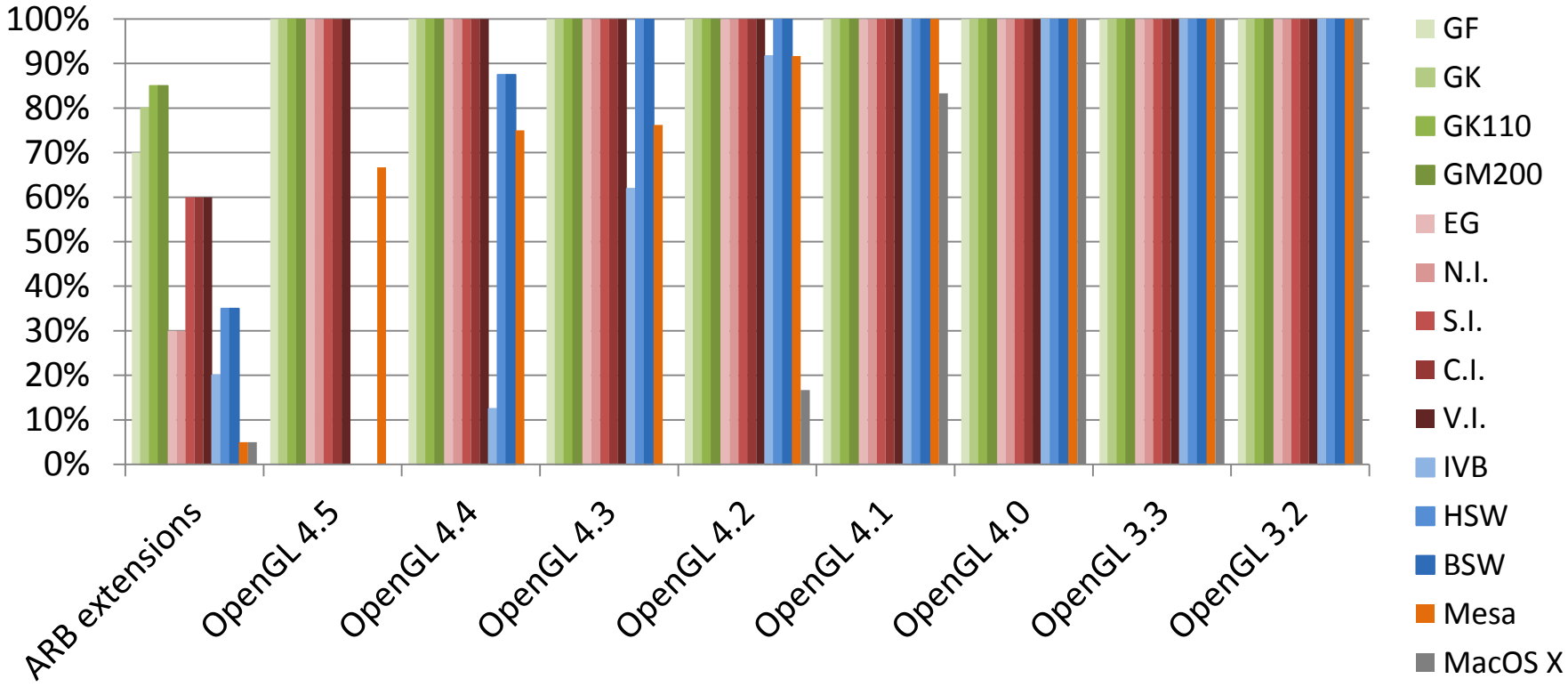
OpenGL®

KHRONOS GROUP

OpenGL Driver Support since last year

AMD from OpenGL 4.4 to 4.5

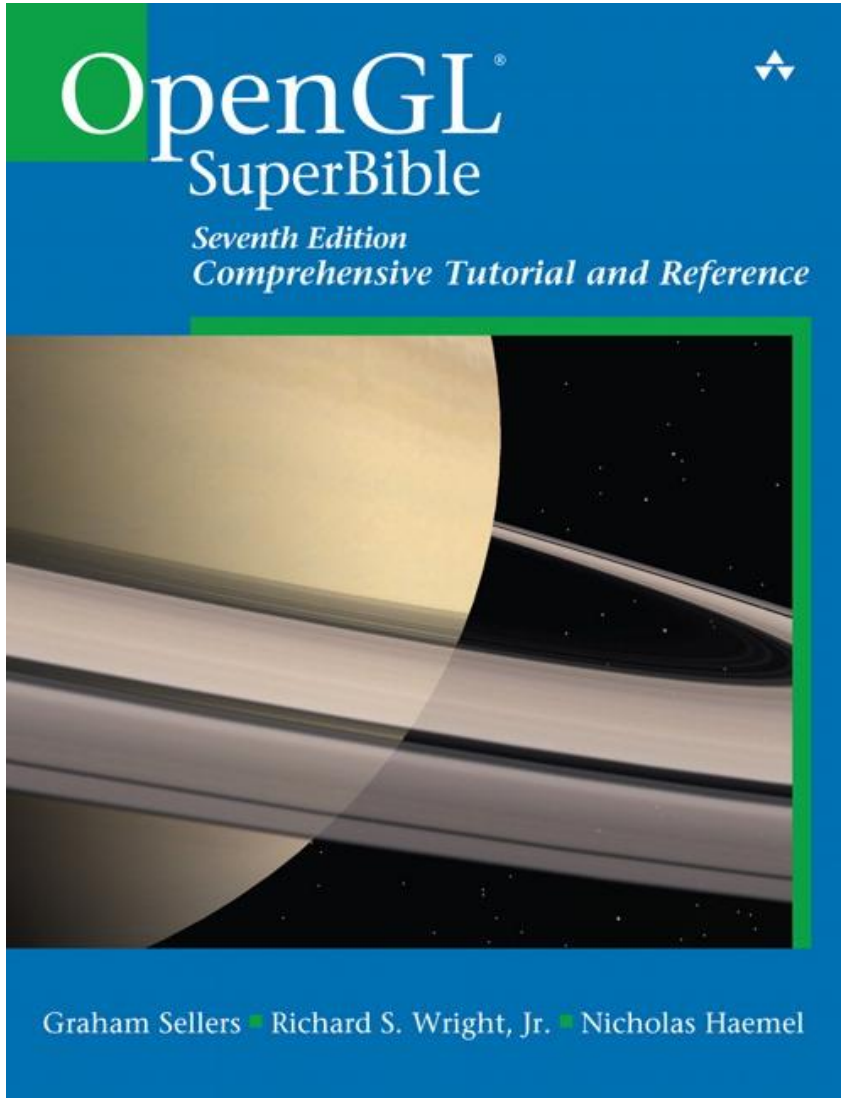
Mesa from OGL 3.3 to OGL 4.1



Intel added OGL 4.3 support for IVB
Significant progress on OGL 4.4

NVIDIA added support for 13 ARB extensions this week
Supported OpenGL 4.5 since Siggraph 2014

Seventh Edition of the OpenGL SuperBible



Glslang Reference Validator update

- Virtually all of ESSL 3.1 + AEP and most of GLSL 4.x
- Nearing completion of arrays of arrays
- SPIR-V translation, disassembly, compression
- Moved to github, seeing community involvement
- Increased portability, robustness, and performance

test.comp

```
#version 310 es

layout(local_size_z = 1000) in;

void main()
{
    barrier();
}
```

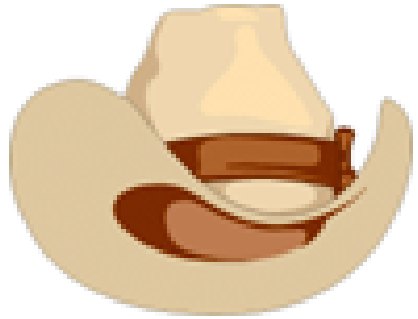
```
$ glslangValidator  test.comp
ERROR: 0:3: 'local_size' : too large; see gl_MaxComputeWorkGroupSize
ERROR: 1 compilation errors.  No code generated.
```

OpenGL Conformance Tests

- **Conformance submissions required for GL 4.4 and 4.5 implementations**
  - encouraged for earlier driver versions
- **Continued improvements to the code**
  - Significant OpenGL 3.x and 4.x coverage added
- **Shared codebase with OpenGL ES 3.2 CTS**
  - additional desktop-specific tests

KHRONOS GROUP

GLEW Support Available NOW



- **GLEW = The OpenGL Extension Wrangler Library**
  - Open source library
    - http://glew.sourceforge.net/
  - Your one-stop-shop for API support for all OpenGL extension APIs

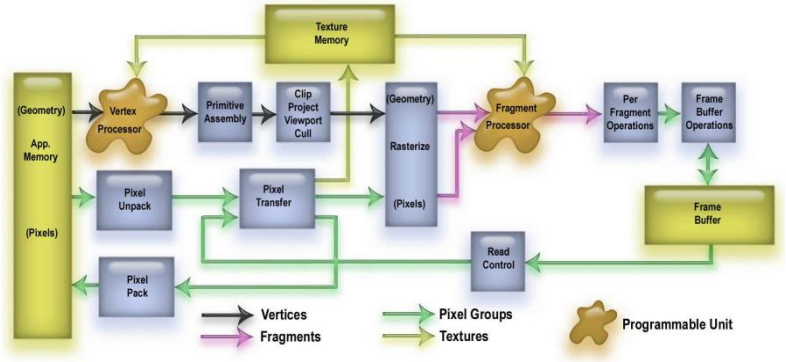
- **GLEW 1.13.0 provides API support for all 13 extensions NOW**

- **Thanks to Nigel Stewart**

KHRONOS GROUP

NEW Graphics Pipeline Operation

- Fragment shader interlock
  - ARB_fragment_shader_interlock
- Programmable sample positions for rasterization
  - ARB_sample_locations
- Post-depth coverage version of sample mask
  - ARB_post_depth_coverage
- Vertex shader viewport & layer output
  - ARB_shader_viewport_layer_array
- Tessellation bounding box
  - ARB_ES3_2_compatibility



Details…

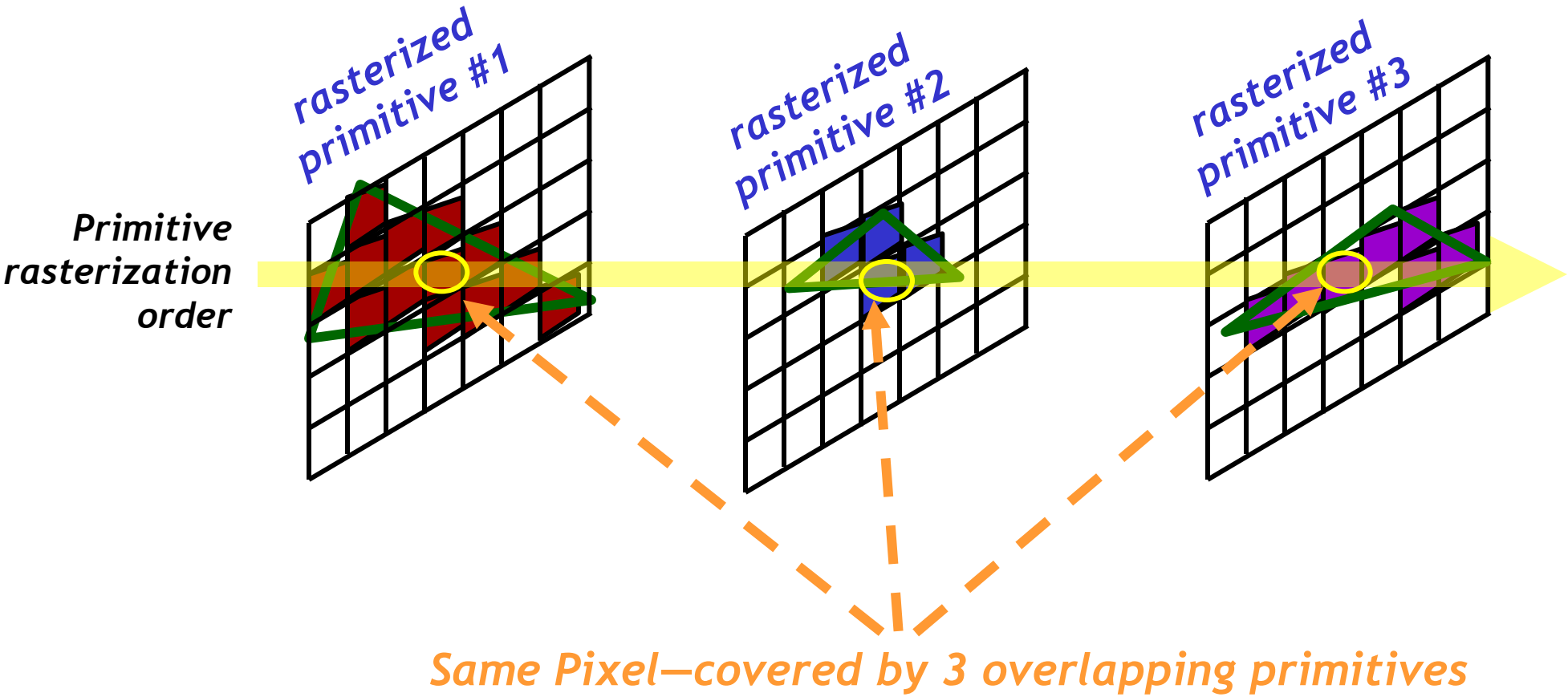
Fragment Shader Interlock

- **NEW extension: ARB_fragment_shader_interlock**
  - Provides reliable means to read/write fragment's pixel state within a fragment shader
    - GPU managed, no explicit barriers needed
- **Uses**
  - Custom blend modes
  - Deferred shading algorithms
    - E.g. screen space decals
- **Adds GLSL functions to begin/end interlock**
  - void beginInvocationInterlockARB(void);
  - void endInvocationInterlockARB(void);
- **Why is a fragment shader interlock needed? ...**

Shared exponent (rgb9e5) format blending via fragment shader interlock

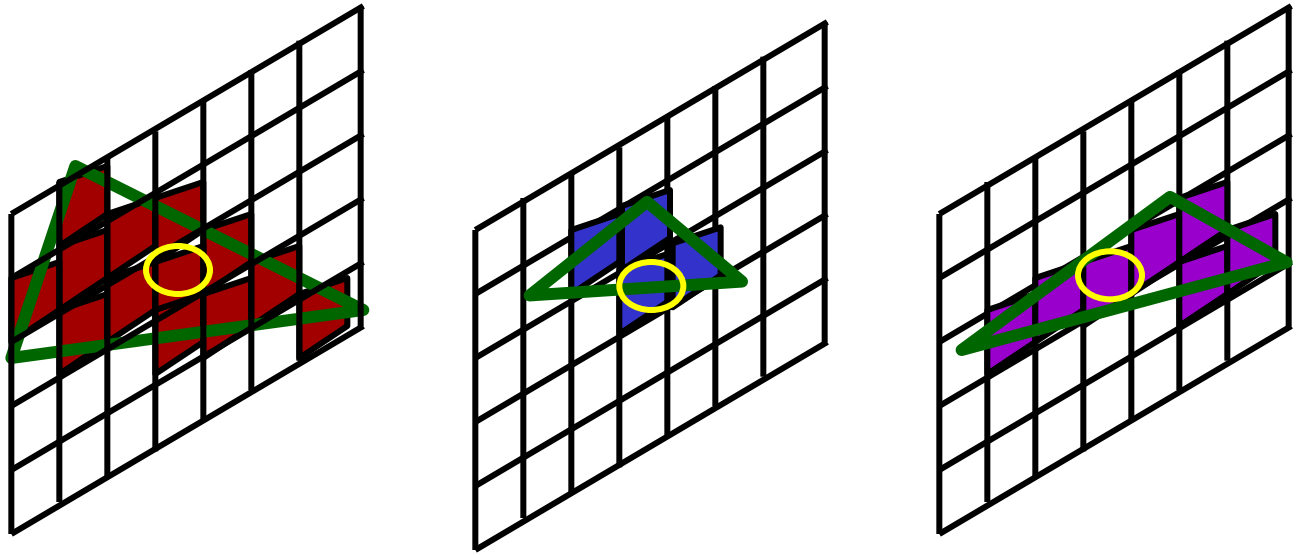
Image credit: David Bookout (Intel), *Programmable Blend with Pixel Shader Ordering*

Pixel Update Preserves Primitive Rasterization Order



OpenGL requires stencil/depth/blend operations be observed to match rendering order, so:

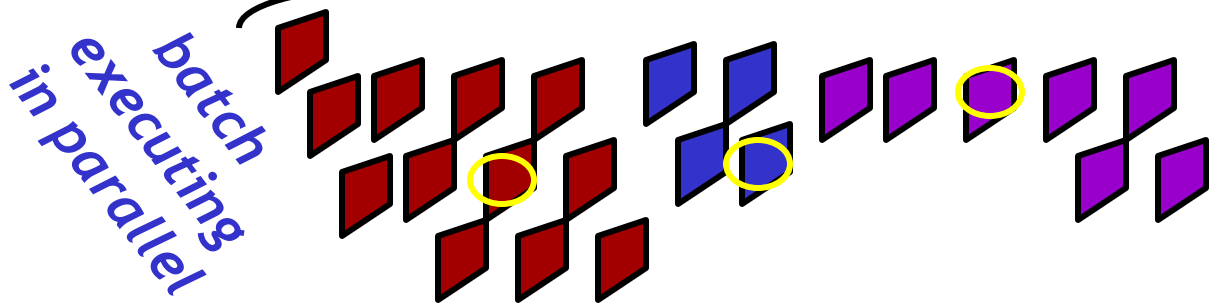
Yet Fragment Shading is Massively Parallel



Conventional Approach
Batch as many fragments in parallel as possible, maximum efficiency

+ scores of other primitives

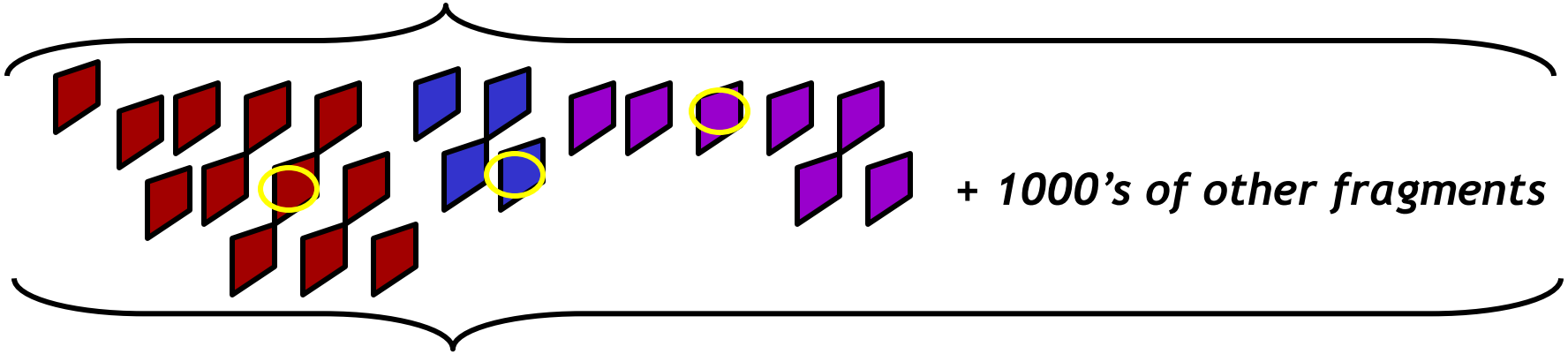
GPU Fragment Shading: parallel execution of fragment shader threads



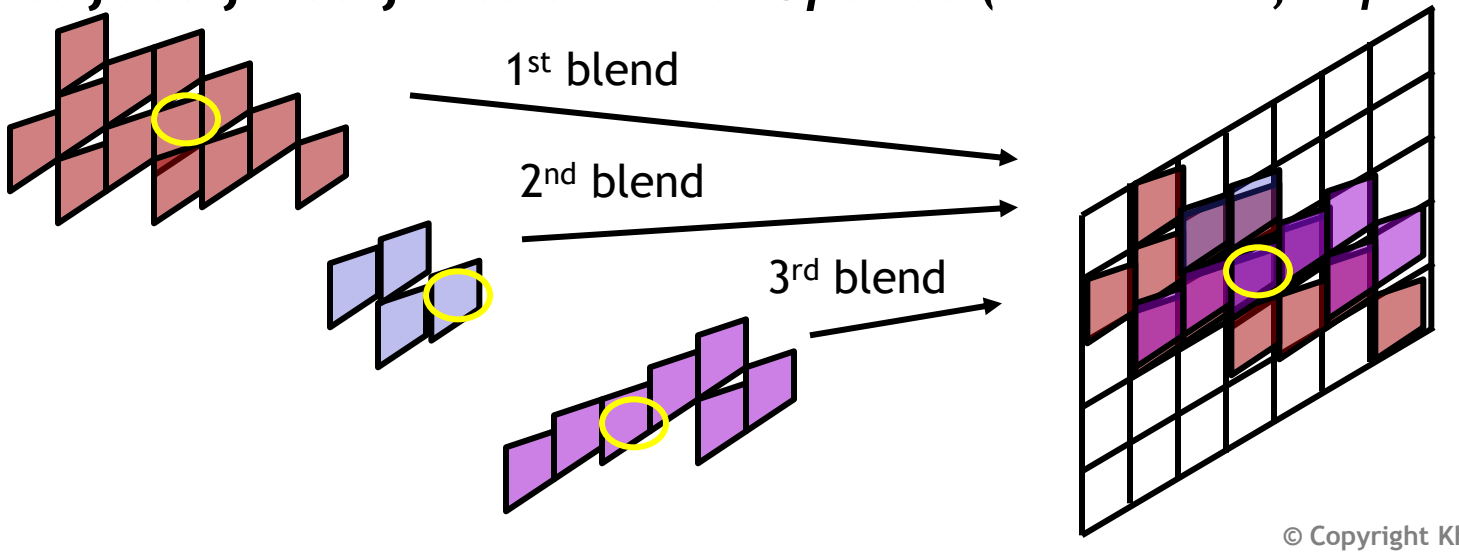
+ 1000's of other fragments

Post-Shader Pixel Updates Respect Rasterization Order

Fragment Shading: parallel execution of fragment shader threads

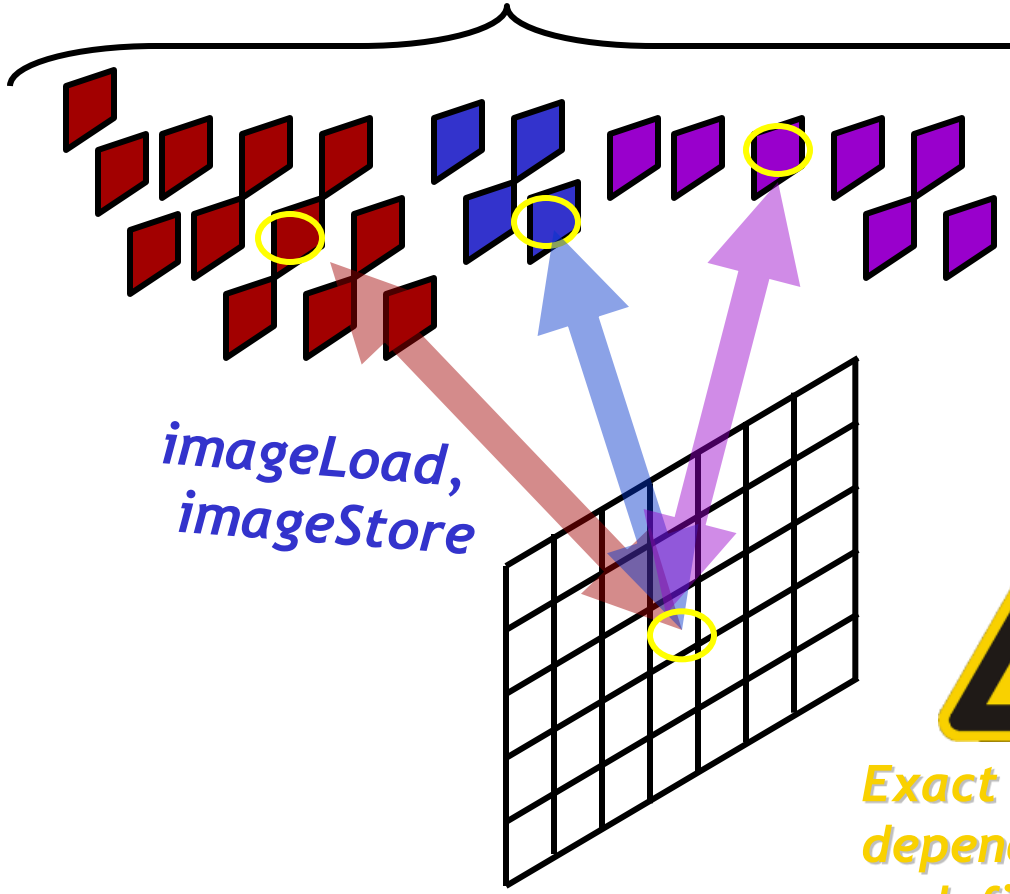


Shader results feed fixed-function Pixel Update (stencil test, depth test, & blend)



However, Shader Access to Framebuffer Unsafe!

GPU Fragment Shading: parallel execution of fragment shader threads

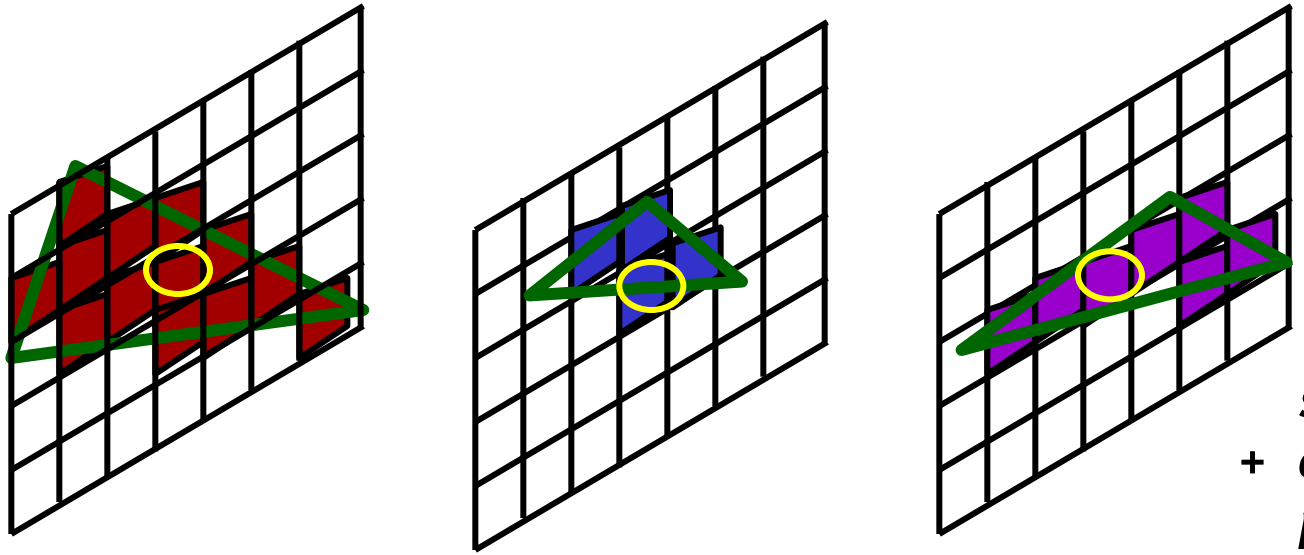


+ 1000's of other fragments

Pixel updates by fragment shader instances executing in parallel cannot guarantee primitive rasterization order!

Exact behavior varies by GPU and timing dependent for any particular GPU—so both undefined & unreliable

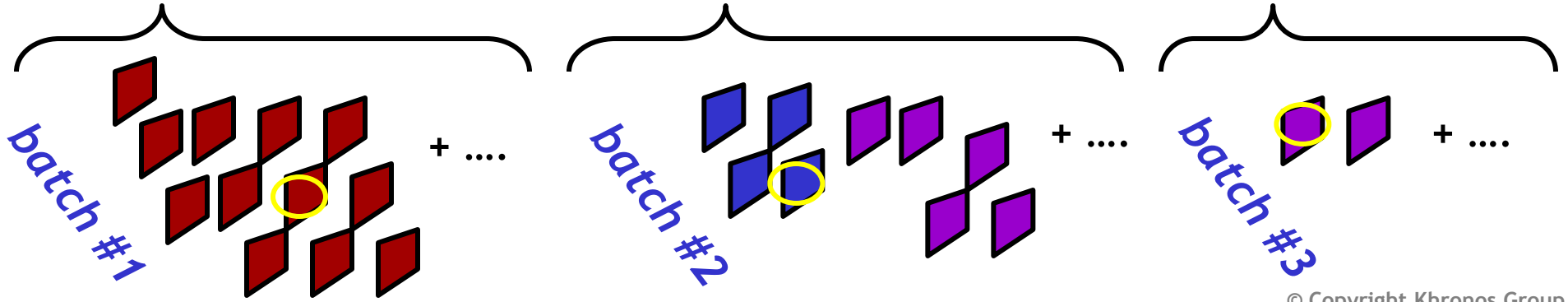
Interlock Guarantees Pixel Ordering of Shading



Interlock Approach
Batch but disallow fragments for same pixel in parallel execution of fragment shader interlock

+ scores of other primitives

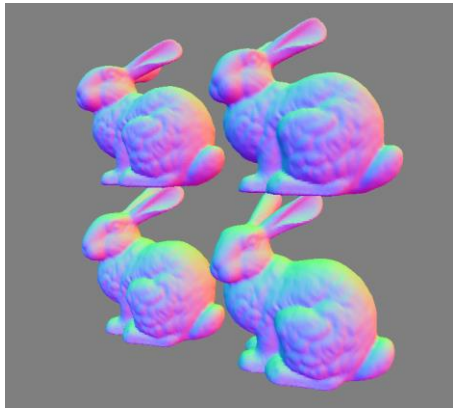
GPU Fragment Shading: parallel execution of fragment shader threads



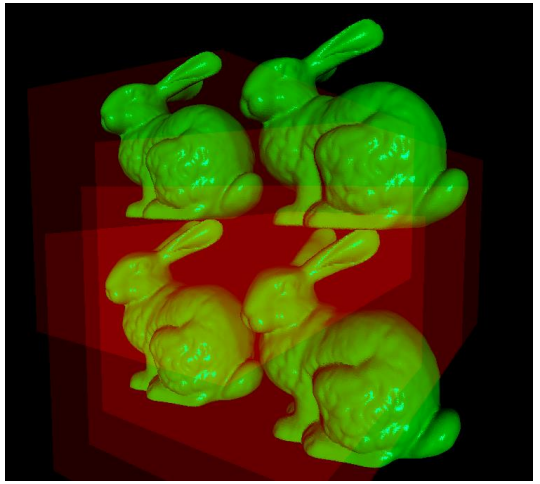
KHRONOS GROUP

Screen Space Decal Approach Visualized

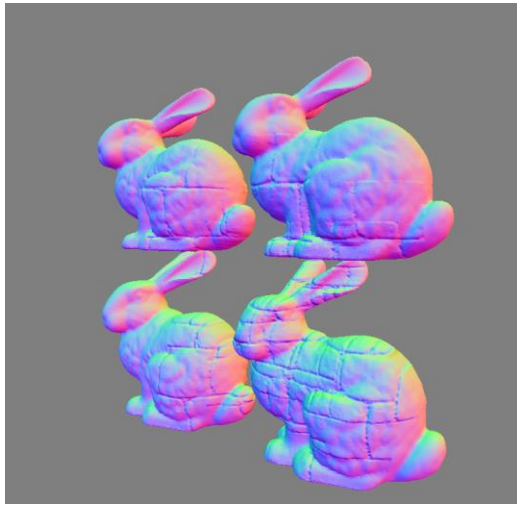
"Normal image"
before *blended*
normal decals



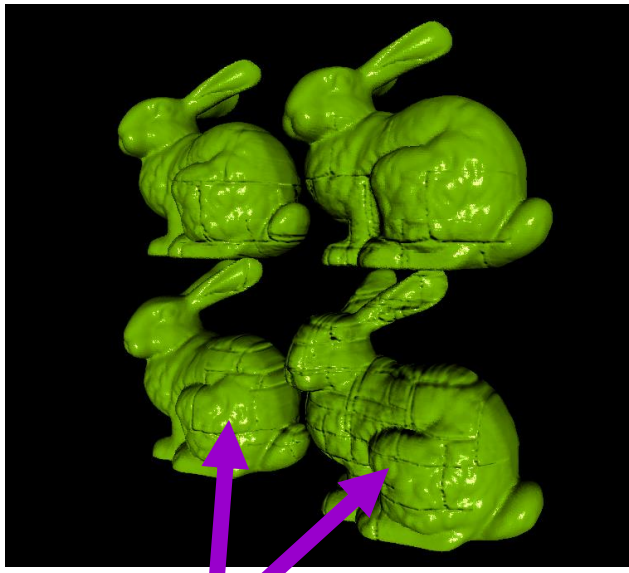
Visualization of decal
boxes overlaid on scene



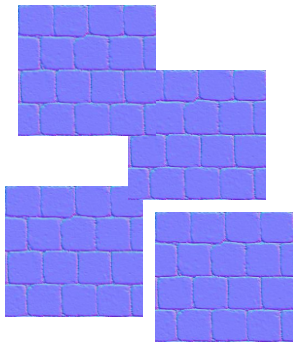
"Normal image"
after *blended*
normal decals



Final shaded color result



Brick pattern
normal map decals
applied to decal
boxes



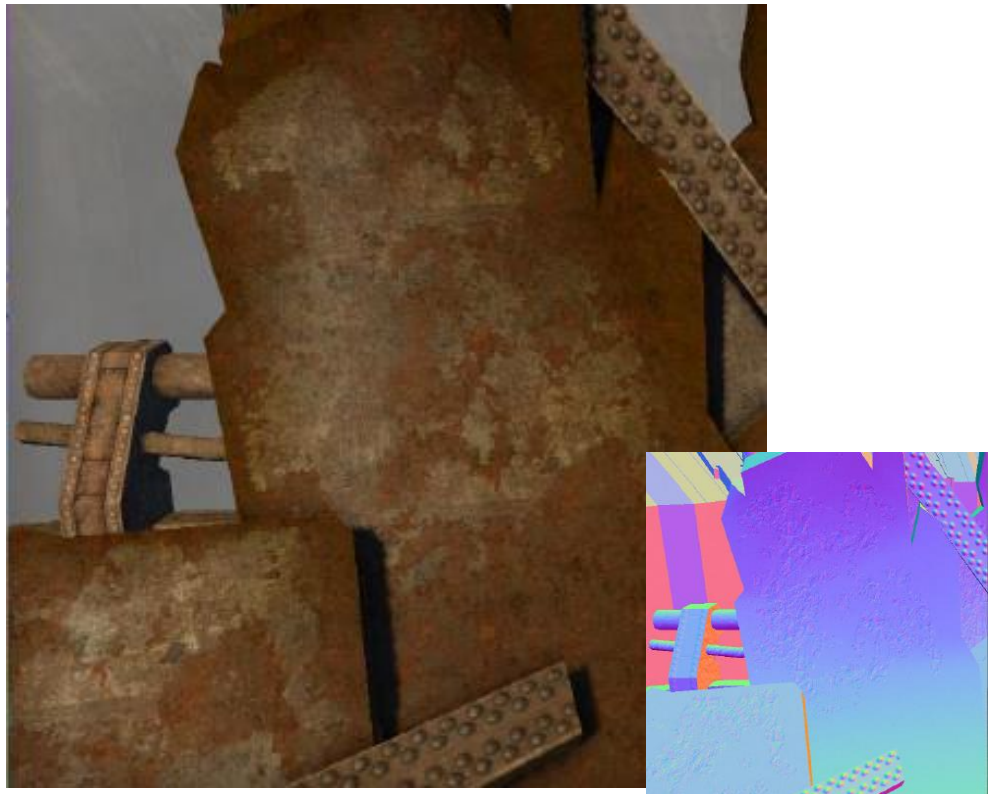
brick normals
blended with
fragment shader
interlock

Bunny shading
includes brick pattern

Motivation: Bullet holes and dynamic scuffs

- **Desire: Dynamically add apparently geometric details as "after effects"**

Without screen-space decals

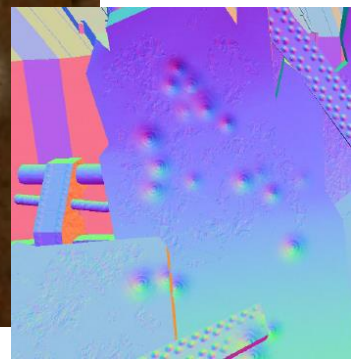


Shaded color result

Normal Map

With screen-space decals

Shaded color result



Normal Map

Image credit: Pope Kim, *Screen Space Decals in Warhammer 40,000: Space Marine*

GLSL Fragment Interlock Usage

- Fragment interlock portion of surface space decal GLSL fragment shader

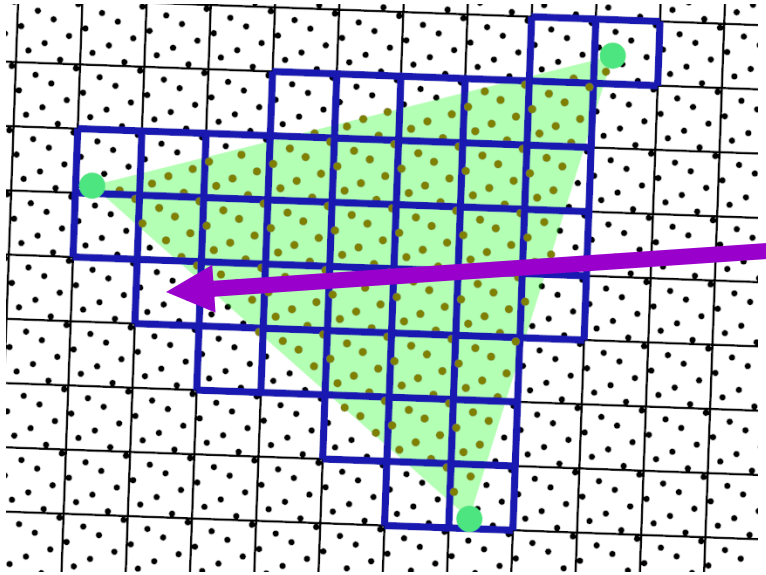
```
beginInvocationInterlockARB(); {

  // Read “normal image” framebuffer's world space normal
  vec3 destNormalWS = normalize(imageLoad(uNormalImage, ivec2(gl_FragCoord.xy)).xyz);
  // Read decal's tangent space normal
  vec3 decalNormalTS = normalize(textureLod(uDecalNormalTex, uv, 0.0).xyz * 2 - 1);
  // Rotate decal's normal from tangent space to world space
  vec3 tangentWS = vec3(1, 0, 0);
  vec3 newNormalWS = normalize(mat3x3(tangentWS,
                                      cross(destNormalWS, tangentWS),
                                      destNormalWS) * decalNormalTS);
  // Blend world space normal vectors
  vec3 destNewNormalWS = normalize(mix(newNormalWS, destNormalWS, uBlendWeight));
  // Write new blended normal into “normal image” framebuffer
  imageStore(uNormalImage, ivec2(gl_FragCoord.xy), vec4(destNewNormalWS,0));

} endInvocationInterlockARB();
```

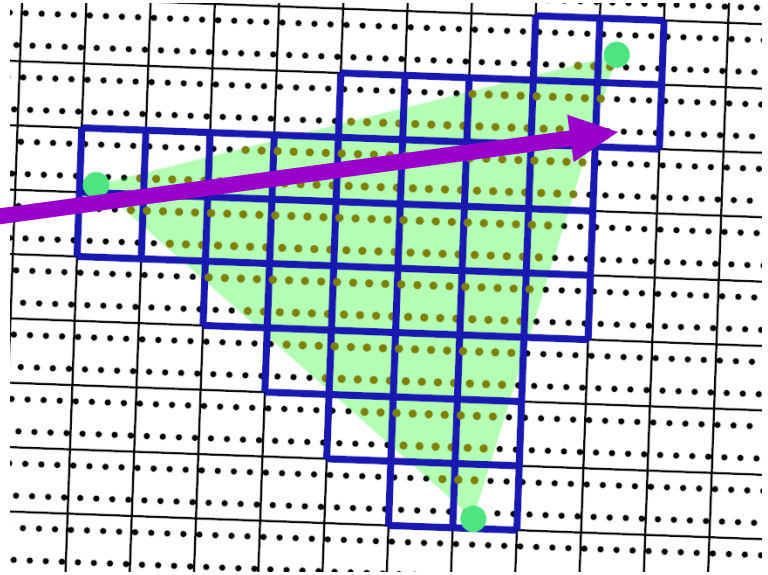
Programmable Sample Positions

- **Conventional OpenGL**
  - Multisample rasterization has fixed sample positions
- **NEW ARB_sample_locations extension**
  - glFramebufferSampleLocationsfvARB specifies sample positions on sub-pixel grid



Default 8x multisample pattern

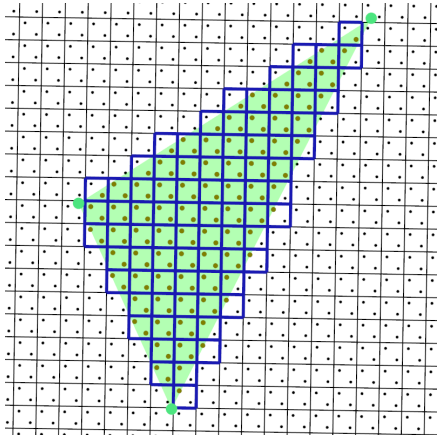
Same triangle but covers sample patterns differently



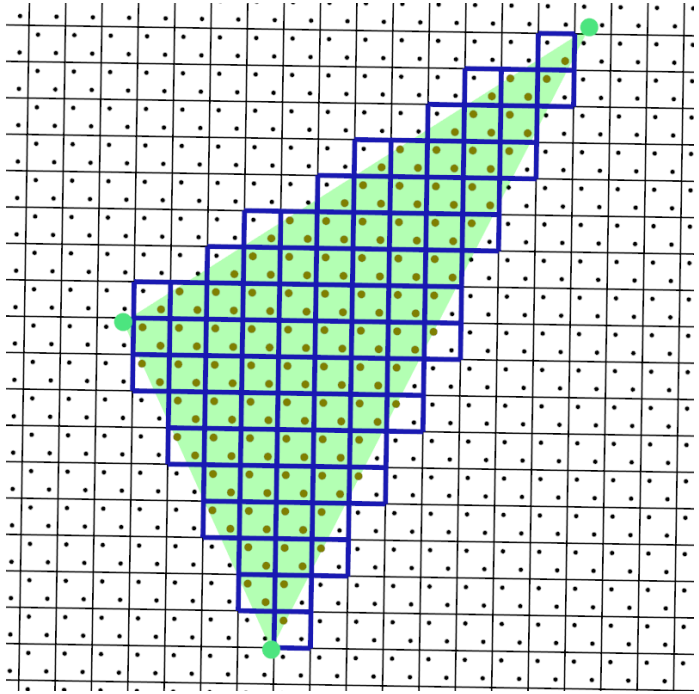
Application-specified 8x multisample pattern, oriented for horizontal sampling

Application: Temporal Antialiasing

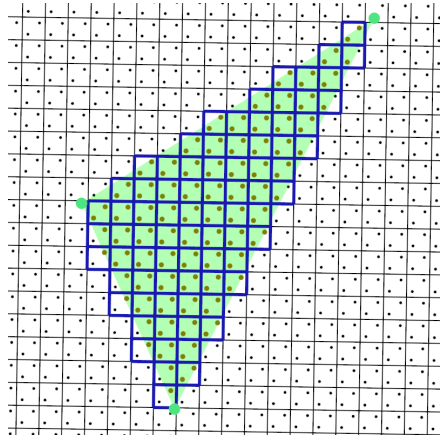
- **Reprogram samples different every frame and render continuously**



Default 2x multisample pattern



Temporal virtual 4x antialiasing



Alternative 2x multisample pattern

- **Done well, can double effective antialiasing quality "for free"**
  - Needs vertical refresh synchronization
  - And app must render at rate matching refresh rate (e.g. 60 Hz)

Vertex Shader Viewport & Layer Output

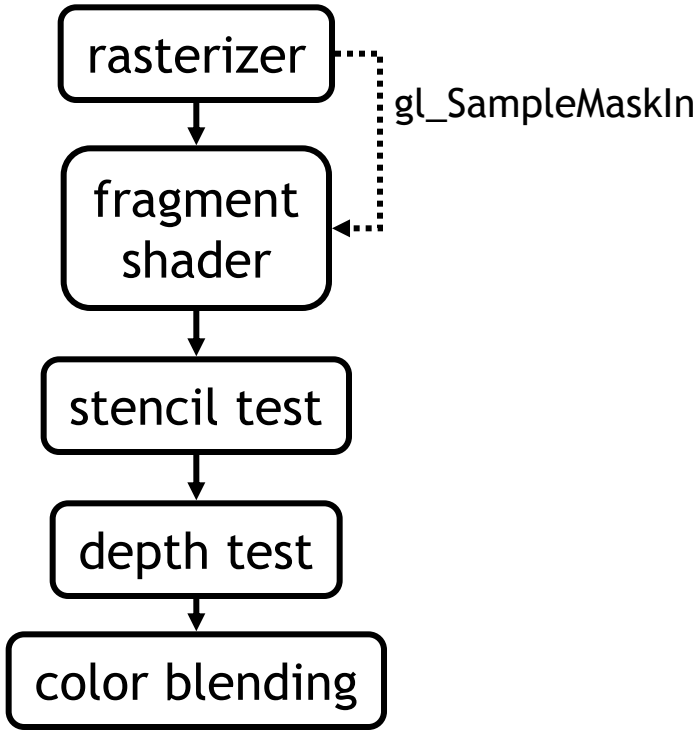
- **NEW extension ARB_shader_viewport_layer_array**

- **Previously geometry shader needed to write viewport index and layer**
  - Forced layered rendering to use geometry shaders
  - Even if a geometry shader wasn't otherwise needed

- **New vertex shader (or tessellation evaluation shader) outputs**
  - out int gl_ViewportIndex
  - out int gl_Layer

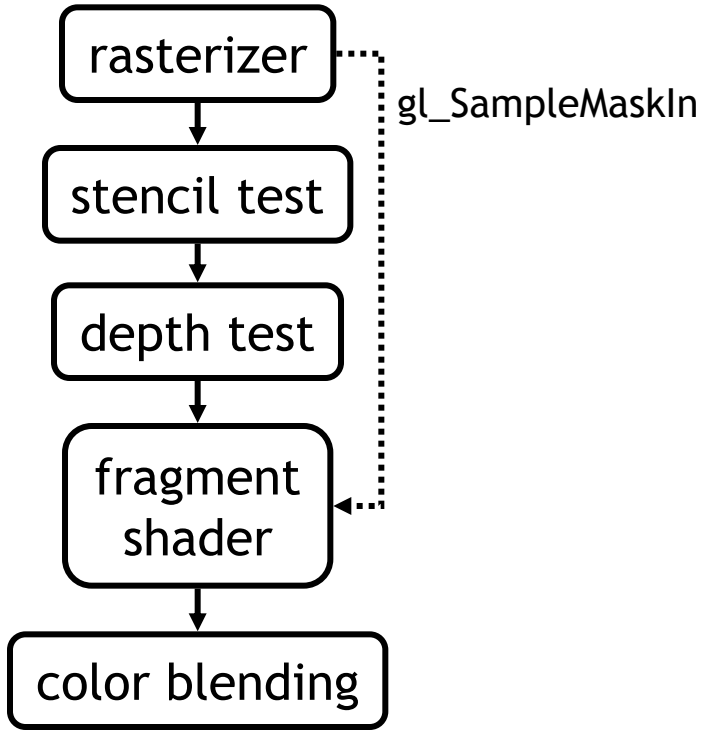
Early Fragment Tests & Post Depth Coverage

Default behavior



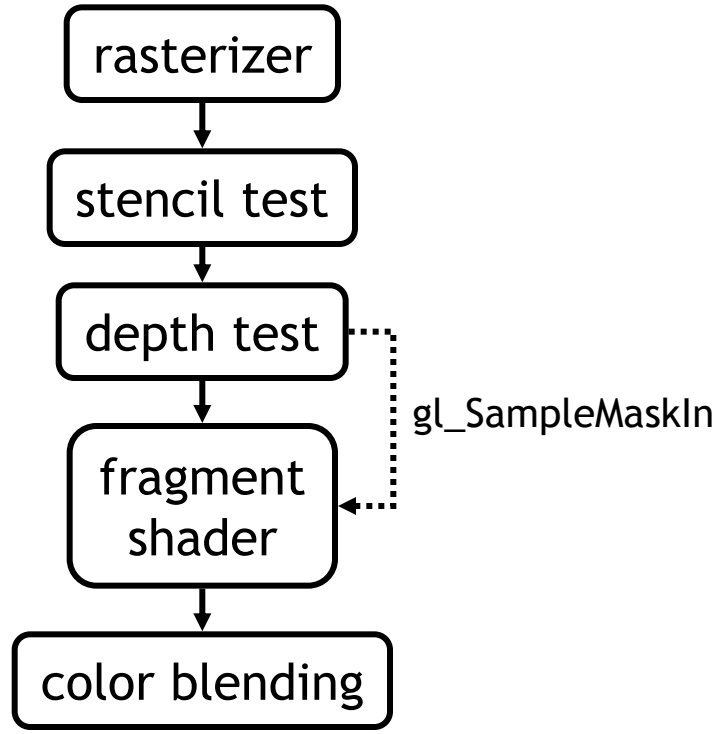
- *Late stencil-depth tests*
- *Rasterizer determines sample mask*

`layout(early_fragment_tests) in;`



- *Early stencil-depth tests*
- *Rasterizer determines sample mask*

```
layout(early_fragment_tests) in;
layout(post_depth_coverage) in;
```



- NEW **ARB_post_depth_coverage**
- *Early stencil-depth tests*
- *Post-depth coverage determines mask*

ES 3.2 Compatibility (tessellation, queries)

- **NEW extension ARB_ES3_2_compatibility**

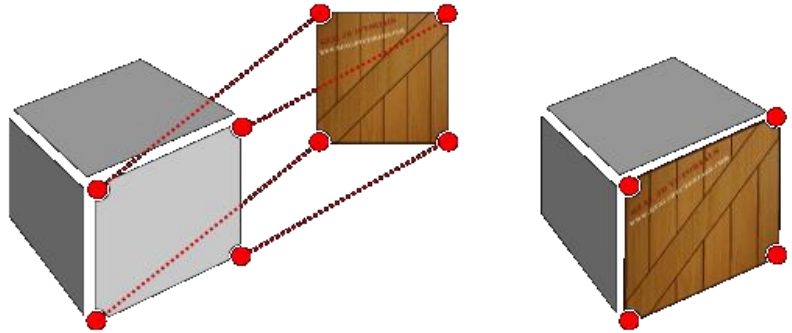
- **Adds Command to specify bounding box for evaluated tessellated vertices in Normalized Device Coordinate (NDC) space**
  - glPrimitiveBoundingBoxARB(float minX, float minY, float minZ,
    float maxX, float maxY, float maxZ)
  - Initial space accepts entirety of NDC space (effectively not limiting tessellation)
  - Implementations may be able to optimize performance, assuming accurate bounds
  - ES 3.2 added this to make tessellation more friendly to mobile use cases
    - **Hint:** Expect today's desktop GPUs are likely to simply ignore this but API matches ES 3.2

- **Adds two implementation-dependent constants related to multisample line rasterization**
  - GL_MULTISAMPLE_LINE_WIDTH_RANGE_ARB
  - GL_MULTISAMPLE_LINE_WIDTH_GRANULARITY_ARB
  - Same toke values as ES 3.2

- **Adds support for OpenGL ES 3.20 shading language**

NEW Texture Mapping Functionality

- **Texture Reduction Modes: Min/Max**
  - ARB_texture_filter_minmax
- **Sparse Textures, done right**
  - ARB_sparse_texture2
- **Sparse Texture Clamping**
  - ARB_sparse_texture_clamp



Details…

New Texture Reduction Modes: Min/Max

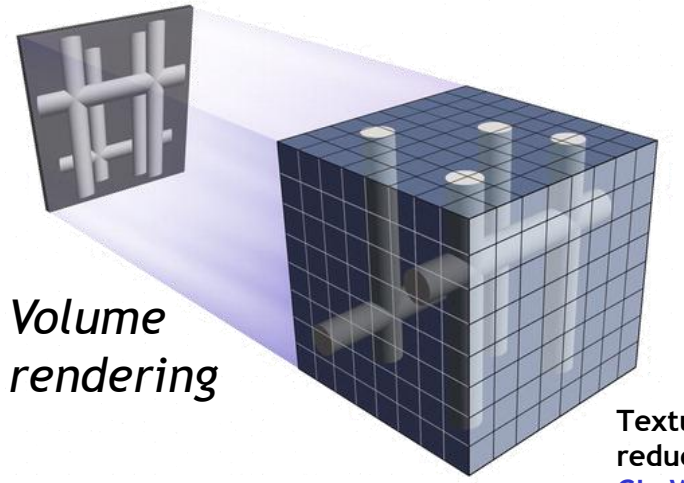
- **NEW extension: ARB_texture_filter_minmax**
  - Texture fetch result = minimum or maximum of all sampled texel values

- **Adds NEW "reduction mode" for texture parameter**
  - **Choices:** GL_WEIGHTED_AVERAGE_ARB (initial state), GL_MIN, or GL_MAX
  - Use with glTexParameteri, glSamplerPatameteri, etc.

- **Example applications**
  - Estimating variance or range when sampling data in textures
  - Conservative texture sampling
    - E.g. Maximum Intensity Projection for medical imaging

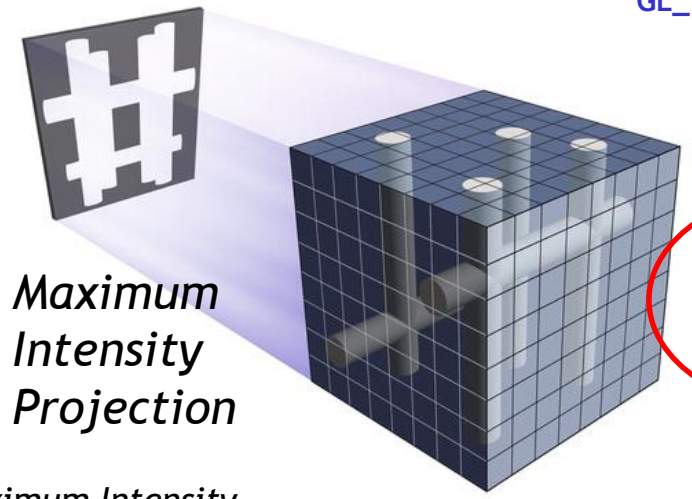
Application: Maximum Intensity Projection

- **Radiologist interpret 3D visualizations of CT scans**
- **Volume rendering simulates opacity attenuated ray casting**
  - Good for visualizing 3D structure
- **Maximum Intensity Projection (MIP) rendering shows maximum intensity along any ray**
  - Good for highlighting features without regard to occlusion
  - Avoids missing significant features



Volume rendering

Texture reduction mode
GL_WEIGHTED_AVERAGE_ARB



Maximum Intensity Projection

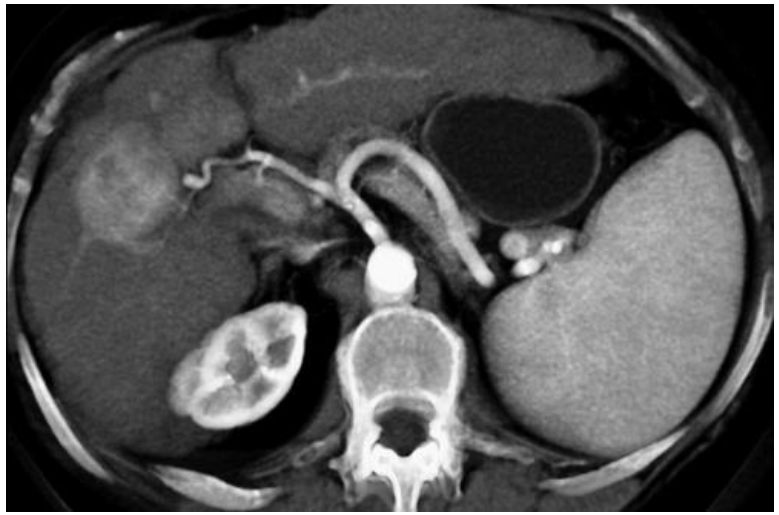
Texture reduction mode
GL_MAX

Image credit: Fishman et al. *Volume Rendering versus Maximum Intensity Projection in CT Angiography: What Works Best, When, and Why*

Maximum Intensity Projection vs. Volume Rendering Visualized

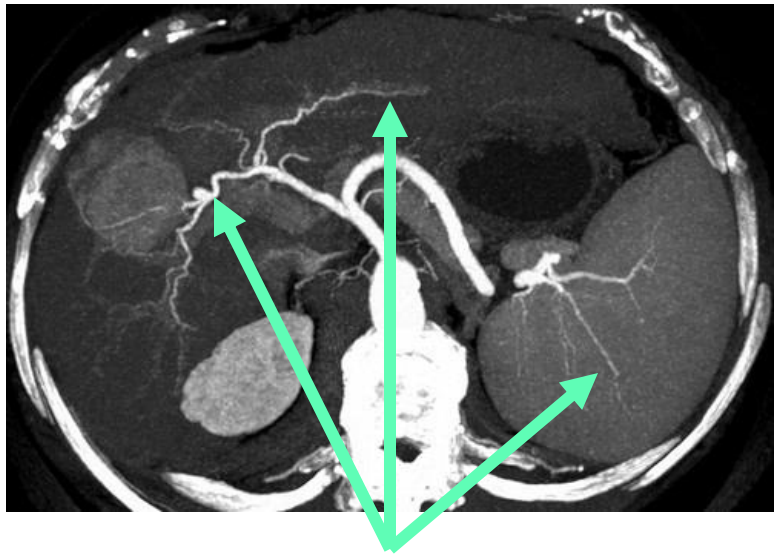
Axial view of human middle torso

Volume Rendering



Provides more 3D feel by accounting for occlusion

Maximum Intensity Projection

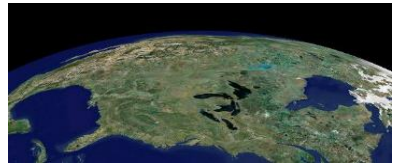


Good at mapping arterial structure, despite occlusion

Image credit: Fishman et al. *Volume Rendering versus Maximum Intensity Projection in CT Angiography: What Works Best, When, and Why*

Sparse Textures Visualized

- **Textures can be HUGE**
  - Think of satellite data
  - Or all the terrain in a huge game level
  - Or medical or seismic imaging



- **We don't never expect to be looking at everything at once!**
  - When textures are huge, can we just make resident what we need?
  - YES, that's sparse texture

- **ARB_sparse_texture standardized in 2013**
  - Reflected limitations of original sparse texture hardware implementations
  - Now we can do better…

Mipmap chain of a sparse texture

Only limited number of pages are resident

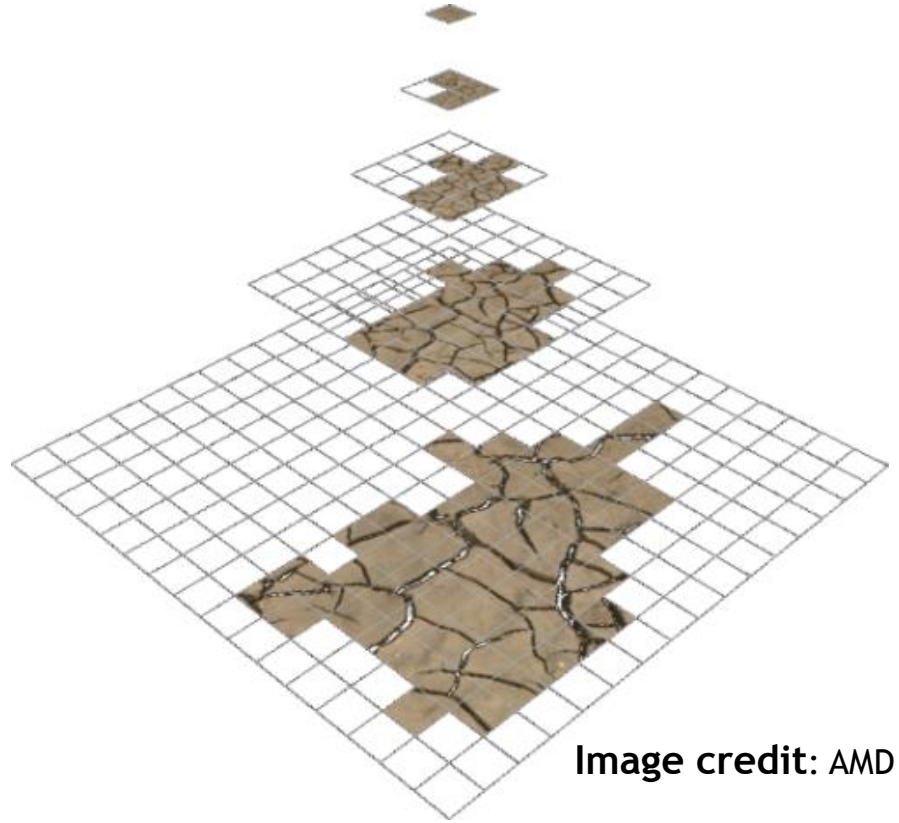


Image credit: AMD

Sparse Textures, done right

- **NEW extension ARB_sparse_texture2**
  - Builds on prior ARB_sparse_texture (2013) extension
    - Limitation:
      - Fetching non-resident data returned undefined results without indication
        - So no way to know if non-resident data was fetched
      - This reflected hardware limitations of the time, fixed in newer hardware

- **Sparse Texture version 2 detects non-resident access**
  - Fetch of non-resident data now returns zero
  - spareTexture*ARB GLSL texture fetch functions return residency information integer
  - sparseTexelsResidentARB GLSL function maps returned integer as Boolean residency
  - Now supports sparse multisample and multisample texture arrays

Sparse Texture, done even better

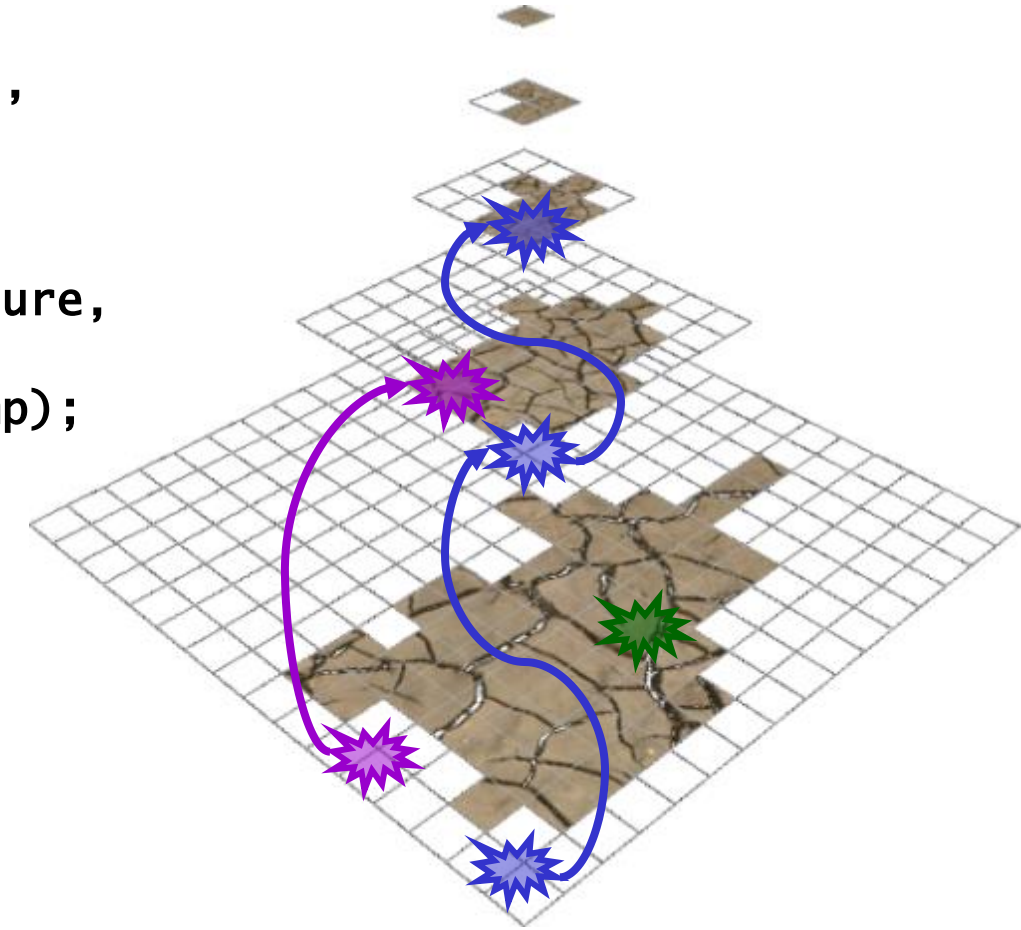
- **NEW extension ARB_sparse_texture_clamp**
- **Adds new GLSL texture fetch variant functions**
  - Includes additional level-of-detail (LOD) parameter to provide a per-fetch floor on the hardware-computed LOD
  - Sparse texture variants
    - sparseTextureClampARB, sparseTextureOffsetClampARB, sparseTextureGradClampARB, sparseTextureGradOffsetClampARB
  - Non-sparse texture versions too
    - textureClampARB, textureOffsetClampARB, textureGradClampARB, textureGradOffsetClampARB

- **Benefit for sparse texture fetches**
  - Shaders can avoid accessing unpopulated portions of high-resolution levels of detail
  - when knowing texture detail is unpopulated
    - Either from a priori knowledge
    - Or feedback from previously executed "sparse" texture lookup functions

Sparse Texture Clamp Example

- Naively fetch sparse texture until you get a valid texel

```
vec4 texel;
int code = sparseTextureARB(spare_texture,
                            uv, texel);
float minLodClamp = 1;
while (!sparseTexelsResidentARB(code)) {
  code = sparseTextureClampARB(sparseTexture,
                               uv, texel,
                               minLodClamp);
  minLodClamp += 1.0f;
}
```



NEW Shader Functionality

- OpenGL ES 3.2 Shading Language Compatibility
  - ARB_ES3_2_compatibility
- Parallel Compile & Link of GLSL
  - ARB_parallel_shader_compile
- 64-bit Integers Data Types
  - ARB_gpu_shader_int64
- Shader Atomic Counter Operations
  - ARB_shader_atomic_counter_ops
- Query Clock Counter
  - ARB_shader_clock
- Shader Ballot and Broadcast
  - ARB_shader_ballot

Details…

ES 3.2 Compatibility (shader support)

- **NEW extension ARB_ES3_2_compatibility**

- **Just say `#version 320 es` in your GLSL shader**
  - Develop and use OpenGL ES 3.2's GLSL dialect from regular OpenGL
  - Helps desktop developers target mobile and embedded devices

- **ES 3.2 GLSL adds functionality already in OpenGL**
  - KHR_blend_equation_advanced, OES_sample_variables, OES_shader_image_atomic, OES_shader_multisample_interpolation, OES_texture_storage_multisample_2d_array, OES_geometry_shader, OES_gpu_shader5, OES_primitive_bounding_box, OES_shader_io_blocks, OES_tessellation_shader, OES_texture_buffer, OES_texture_cube_map_array, KHR_robustness
  - Notably Shader Model 5.0, geometry & tessellation shaders

Parallel Compile & Link of GLSL

- **NEW extension ARB_parallel_shader_compile**
  - Facilitates OpenGL implementations to distribute GLSL shader compilation and program linking to multiple CPU threads to speed compilation throughput
  - Allows apps to better manage GLSL compilation overheads
    - Benefit: Faster load time for new shaders and programs on multi-core CPU systems

- **Part 1: Tells OpenGL's GLSL compiler how many CPU threads to use for parallel compilation**
  - void glMaxShaderCompilerThreadsARB(GLuint threadCount)
  - Initially allows implementation-dependent maximum (initial value 0xFFFFFFFF)

- **Part 2: Shader and program query if compile or link is complete**
  - Call glGetShaderiv or glGetProgramiv on GL_COMPLETION_STATUS_ARB parameter
  - Returns true when compile is complete, false if still compiling
  - Unlike other queries, will not block for compilation to complete.

64-bit Integer Data Types in GLSL

- **NEW extension** ARB_gpu_shader_int64
  - adds 64-bit integers
- **New data types**
  - Signed: int64_t, i64vec2, i64vec3, i64vec4,
  - Unsigned: uint64_t, u64vec2, u64vec3, u64vec4
  - Supported for uniforms, buffers, transform feedback, and shader input/outputs
- **Standard library extended to 64-bit integers**
- **Programming interface**
  - Uniform setting
    - glUniform{1,2,3,4}i{,v}64ARB
    - glUniform{1,2,3,4}ui{,v}64ARB
  - Direct state access (DSA) variants as well
    - glProgramlUniform{1,2,3,4}i{,v}64ARB
    - glProgramlUniform{1,2,3,4}ui{,v}64ARB
  - Queries for 64-bit uniform integer data

Shader Ballot and Broadcast

- **NEW extension ARB_shader_ballot**
  - Assumes 64-bit integers

- **Concept**
  - Group of invocations (shader threads) which execute in lockstep can do a limited forms of cross-invocation communication via a group broadcast of a invocation value, or broadcast of a bitarray representing a predicate value from each invocation in the group
  - Allows efficient collective decisions within a group of invocations

- **New built-in data types**
  - Uniform: gl_SubGroupSizeARB
  - Integer input: gl_SubGroupInvocationARB
  - Mask input: gl_SubGroupEqMaskARB, gl_SubGroupGeMaskARB, gl_SubGroupGtMaskARB, gl_SubGroupLeMaskARB, gl_SubGroupLtMaskARB
- **New GLSL functions**
  - uint64_t ballotARB(bool value)

Shader Atomic Counter Operations in GLSL

- **NEW extension ARB_shader_atomic_counter_ops**
  - Builds on ARB_shader_atomic_counters extension (2011, OpenGL 4.2)
    - Original atomic counters quite limited
      - Could only increment, decrement, and query

- **New operations supported on counters**
  - Addition and subtraction: atomicCounterAddARB, atomicCounterSubtractARB
  - Minimum and maximum: atomicCounterMinARB, atomicCounterMaxARB
  - Bitwise operators (AND, OR, XOR, etc.)
    - atomicCounterAndARB, atomicCounterOrARB, atomicCounterXorARB
  - Exchange: atomicCounterExchangeARB
  - Compare and Exchange: atomicCounterCompSwapARB

Query Clock Counter in GLSL

- **NEW extension ARB_shader_clock**

- **New functions query a free-running "clock"**
  - 64-bit monotonically incrementing shader counter
  - uint64_t clockARB(void)
  - uvec2 clock2x32ARB(void)
    - Avoids requiring 64-bit integers, instead returns two 32-bit unsigned integers

- **Similar to Win32's QueryPerformanceCounter**
  - But within the GPU shader complex

- **Can allow shaders to monitor their performance**
  - Details implementation-dependent

Thanks

- **Multi-vendor effort!**

- **Particular thanks to specification leads**
  - Pat Brown (NVIDIA)
  - Piers Daniell (NVIDIA)
  - Slawomir Grajewski (Intel)
  - Daniel Koch (NVIDIA)
  - Jon Leech (Khronos)
  - Timothy Lottes (AMD)
  - Daniel Rakos (AMD)
  - Graham Sellers (AMD)
  - Eric Werness (NVIDIA)

OPENGL SUPPORT IN UNITY 5.3

KHRONOS API SUPPORT

- OpenGL core profile 3.2 to 4.5 (desktop)
- OpenGL ES 2.0 to 3.1aep (desktop & mobile)
- WebGL 1.0 and 2.0 (desktop)

- Feature parity with Direct3D11

- Why OpenGL ES on desktop?

OPEN PROBLEMS IN REAL-TIME RENDERING

- Siggraph 2015 course
- *Panel: Production Cost in Games: What Technology is Needed?*

OPEN PROBLEMS IN REAL-TIME RENDERING

- Siggraph 2015 course
- *Panel: Production Cost in Games: What Technology is Needed?*

Why OpenGL ES on desktop?
To improve Unity developers interaction cycle

unity

OPENGL SUPPORT IN UNITY 5.3

- December 8th 2015

OpenGL demo

Christophe Riccio, Unity

What's new in OpenGL ES

Tom Olson, ARM
OpenGL ES Working Group chair

Outline

- **Introduction and status**
- **OpenGL ES 3.2 overview**
- **Open sourcing the OpenGL ES conformance test**
- **What the future holds**
- **Member news**
  - NVIDIA – Piers Daniell
  - Imagination Technologies – Tobias Hector
  - Kishonti Informatics – Zoltan Hortsin

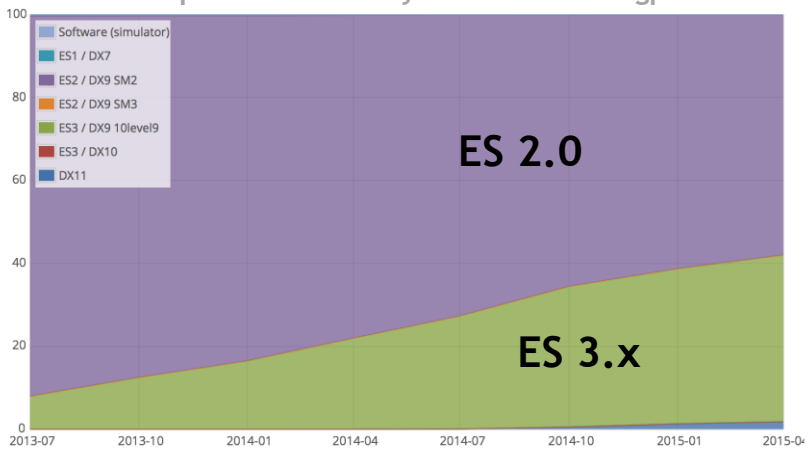
OpenGL ES today

- **The most widely deployed 3D graphics API in history**
  - Industry will ship >1.7 billion devices in 2015

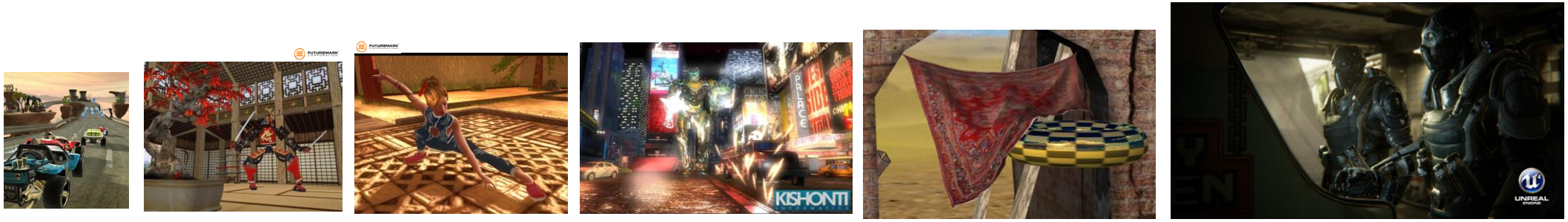
- **ES 2.0 is still shipping in large volumes**
  - Entry level phones / tablets, smart TV, automotive…
  - Being adopted rapidly in new ultra-low-power form factors
  - Basis for WebGL, possibly OpenGL ES SC

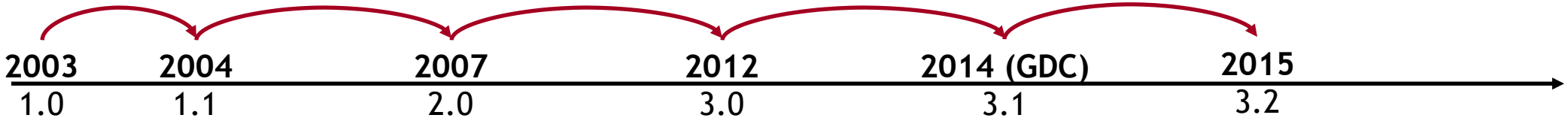
- **ES 3.x is gaining share**
  - *de facto* standard for high-end smartphones
  - 40% share in latest Unity mobile device stats

http://hwstats.unity3d.com/mobile/gpu.html



The evolution continues…





- **What's new**
  - OpenGL ES 3.2 released this week
  - Conformance test and ecosystem updates

Introducing OpenGL ES 3.2

- **Goals**
  - Absorb Android Extension Pack (AEP) into core
  - Add other useful features IF universally supported
  - Minimize effort – don't impact Vulkan development schedule

- **What is AEP?**
  - A 'meta-extension' rolling up 20 EXT/OES extensions and features

```
KHR_debug                                   EXT_draw_buffers_indexed
KHR_texture_compression_astc_ldr            EXT_geometry_shader
KHR_blend_equation_advanced                 EXT_gpu_shader5
OES_sample_shading                          EXT_primitive_bounding_box
OES_sample_variables                        EXT_shader_io_blocks
OES_shader_image_atomic                     EXT_tessellation_shader
OES_shader_multisample_interpolation        EXT_texture_border_clamp
OES_texture_stencil8                        EXT_texture_buffer
OES_texture_storage_multisample_2d_array    EXT_texture_cube_map_array
EXT_copy_image                              EXT_texture_sRGB_decode
```

OpenGL ES 3.2 features

- **New pipeline stages: Tessellation and Geometry**
  - OES_tessellation_shader
  - OES_geometry_shader
  - OES_shader_io_blocks
  - OES_primitive_bounding_box

- **Extended compute functionality**
  - OES_shader_image_atomic
  - Compute operations in fragment shader

- **Per-sample processing**
  - OES_sample_shading
  - OES_sample_variables
  - OES_shader_multisample_interpolation

OpenGL ES 3.2 features continued…

- **Extended blending operations**
  - OES_draw_buffers_indexed
  - KHR_blend_equation_advanced

- **Ease of getting your code working**
  - KHR_debug

- **Texturing functionality**
  - KHR_texture_compression_astc_ldr
  - OES_texture_cube_map_array
  - OES_texture_buffer
  - OES_texture_border_clamp
  - OES_texture_stencil8
  - OES_texture_storage_multisample_2d_array
  - EXT_texture_sRGB_decode

OpenGL ES 3.2 features continued…

- **Extended blending operations**
  - OES_draw_buffers_indexed
  - KHR_blend_equation_advanced

- **Ease of getting your code working**
  - KHR_debug

- **Texturing functionality**
  - KHR_texture_compression_astc_ldr
  - OES_texture_cube_map_array
  - OES_texture_buffer
  - OES_texture_border_clamp
  - OES_texture_stencil8
  - OES_texture_storage_multisample_2d_array
  - ~~EXT_texture_sRGB_decode~~

Other new features

- **KHR_robustness**
  - Safe queries (no buffer overrun)
  - Enable ‘graceful’ recovery from GPU reset
  - Deterministic behavior on out-of-range index accesses

- **OES_draw_elements_base_vertex**
  - Add an offset to every index in an indexed draw call

- **EXT_color_buffer_float**
  - Required support for 16F and 32F render targets: R, RG, RGBA
  - Required support for R11F_G11F_B10F render targets

OpenGL ES 3.2 status

- **Specifications released on Monday**
  - API specification editor – Jon Leech (Khronos)
  - GLSL ES specification editor – Rob Simpson (Qualcomm)

- **Man pages available**
  - Man page editor – Ben Bowman (Imagination Technologies)

- **GLSLang reference compiler in progress**
  - John Kessenich (LunarG)
  - Currently almost complete for AEP

- **Conformance test in progress**
  - Contractor selection underway
  - *Will be managed as an open source project starting later in 2015*

Why open source conformance?

- **Why not?**
  - Seems like the right thing to do
- **Broaden range of contributors**
  - Enable contributions based on real applications
- **Share code with the community**
  - dEQP CTS (AOSP)
  - Piglit?
- **Issues**
  - Khronos gitlab vs github
  - Need to factor out components that can't be open-sourced

What does the future hold? (my view…)

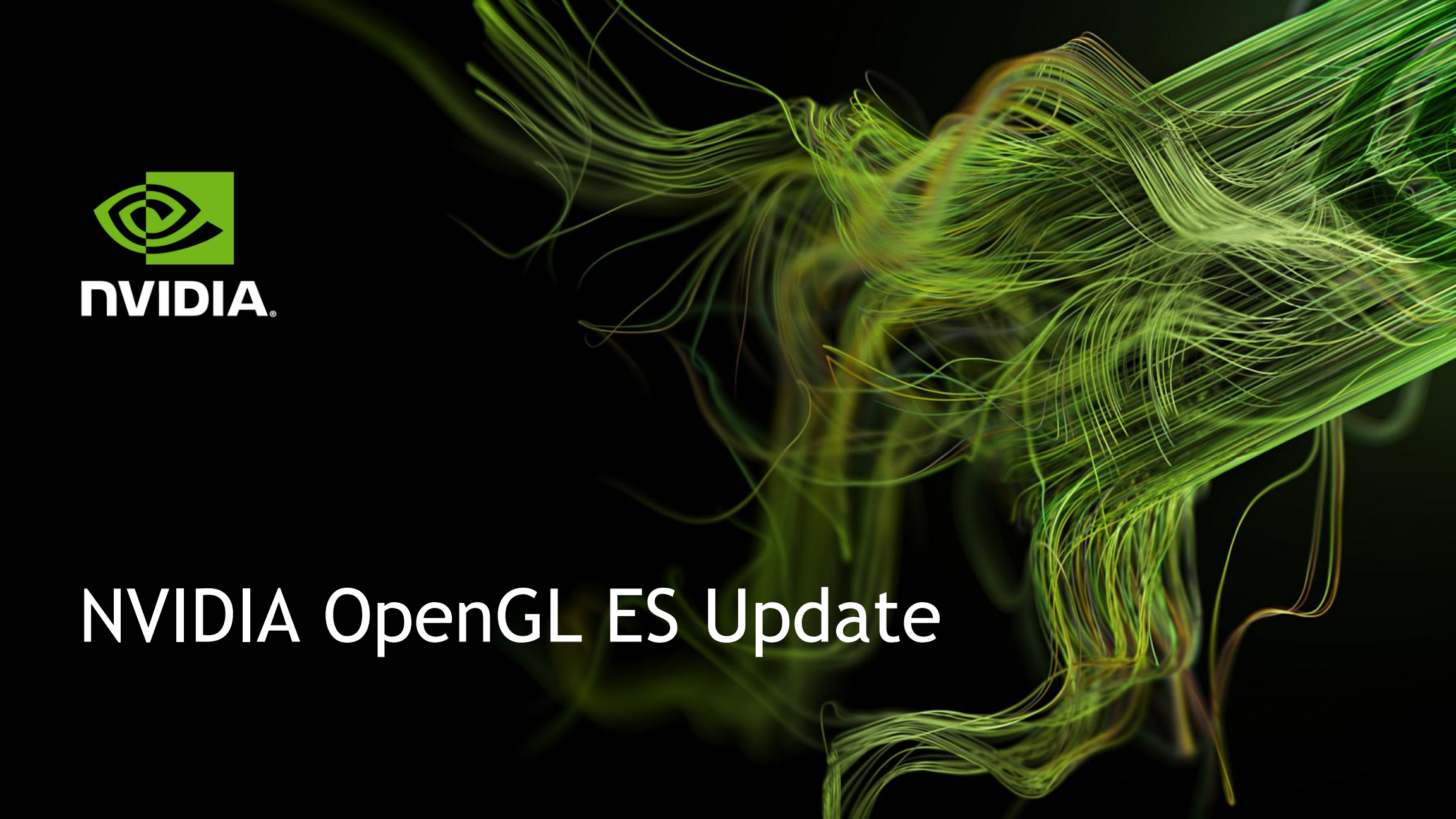
- **Especially performance- or latency-sensitive apps will migrate to Vulkan**
  - Better control of frame rate and latency
  - Easier to max out the hardware

- **OpenGL ES will continue to be the API of choice for a wide range of applications**
  - Shortest path to a functionally correct application
  - Reaches the widest range of platforms / largest number of eyeballs

- **OpenGL ES technical evolution will go on as long as needed**
  - Will continue to expose new hardware capabilities

OpenGL ES – Member News

OpenGL ES Working Group members



NVIDIA OpenGL ES Update

OpenGL ES 3.2 Driver

Available today

OpenGL ES 3.2 driver for Windows and Linux *

https://developer.nvidia.com/opengl-driver

Supported by GeForce 4xx series (Fermi) and up

Develop your OpenGL ES 3.2 content on desktop now

Deploy on mobile when available

OpenGL ES 3.2 for Shield Android TV coming soon via OTA update

Shield Tablet later

** Product is based on a published Khronos specification and is expected pass the Khronos Conformance Process when available. Current conformance status can be found at www.khronos.org/conformance.*

OpenGL ES 3.2 Update

OpenGL ES 3.2

- Supported by Series6XT Cores onwards
  - AEP in current drivers
  - ES 3.2 available shortly*

- Great new features!
  - Floating Point Rendering
  - Debugging
  - More blend modes
  - Dynamic indexing

** Product is based on a published Khronos specification and is expected pass the Khronos Conformance Process when available. Current conformance status can be found at www.khronos.org/conformance.*

KHRONOS GROUP

Vulkan Update
SIGGRAPH 2015

Outline

- **Introduction and status**
  - Tom Olson (ARM), Vulkan WG chair
- **Working Group progress report**
  - Loaders and Layers: Vulkan SDK – Jens Owen (LunarG / Valve)
  - Window system integration – Alon Or-bach (Samsung), Vulkan WSI chair
  - Vulkan API changes since GDC – Jesse Barker (ARM)
- **Why Vulkan is great**
  - Tobias Hector (Imagination Technologies)
- **Bringing Vulkan to the 3D ecosystem**
  - WG members

A Vulkan project timeline

- **2014 July – Project launch**
- **2014 August – Plan disclosed at SIGGRAPH**
- **2015 January – Most technical issues closed**
- **2015 March – Disclosure at GDC**
  - Discussions / feedback cycles with external developer community under NDA
  - Thanks guys!
- **2015 June – Internal “soft freeze”**
  - Spec writing
  - SDK construction
  - Conformance test work
  - Continued discussion and refinement

Project status

- **On track to deliver a specification by e/o 2015**

- **Specification writing is the critical path**
  - Thanks to our heroic editors:
    - Graham Sellers, AMD
    - Bill Licea-Kane, Qualcomm
    - Jon Leech, Khronos / Valve
  - …plus working group members

- **Conformance test creation is staffed, planned, and well under way**
  - Thanks to
    - Pyry Haulos, Google – CTS project manager / tech lead
    - Code contributions from Imagination Technologies, Intel, Qualcomm, Samsung

Loaders and Layers: Vulkan SDK

Jens Owen and Courtney Goeltzenleuchter
LunarG

Introduction to the LunarG SDK

- Valve funding LunarG to support Vulkan ecosystem
  - Cross platform tools will be made available as open source
  - LunarG providing SDK and support to developers via LunarXchange
    - Subscribe for updates at http://lunarg.com/vulkan

sign up: http://LunarG.com/Vulkan/

KHRONOS GROUP

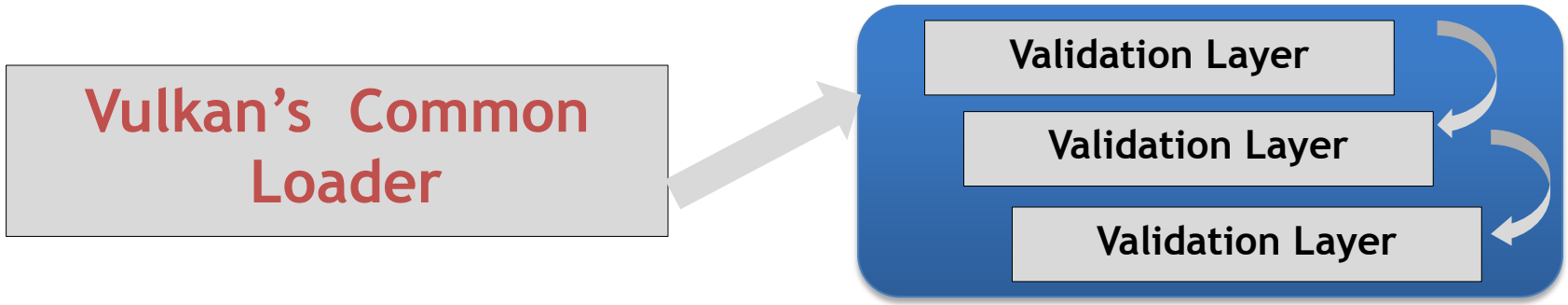
Introduction to the Vulkan Loader

- Common Loader used to enable use of layers
  - Add layers without affecting the app or the ICD
  - Provides a plug-n-play experience –
    - i.e. multiple Vulkan devices can coexist on a system peacefully
    - Aggregates drivers from multiple vendors

- Handler of…
  - Driver management
  - Layer libraries
  - Instance extensions

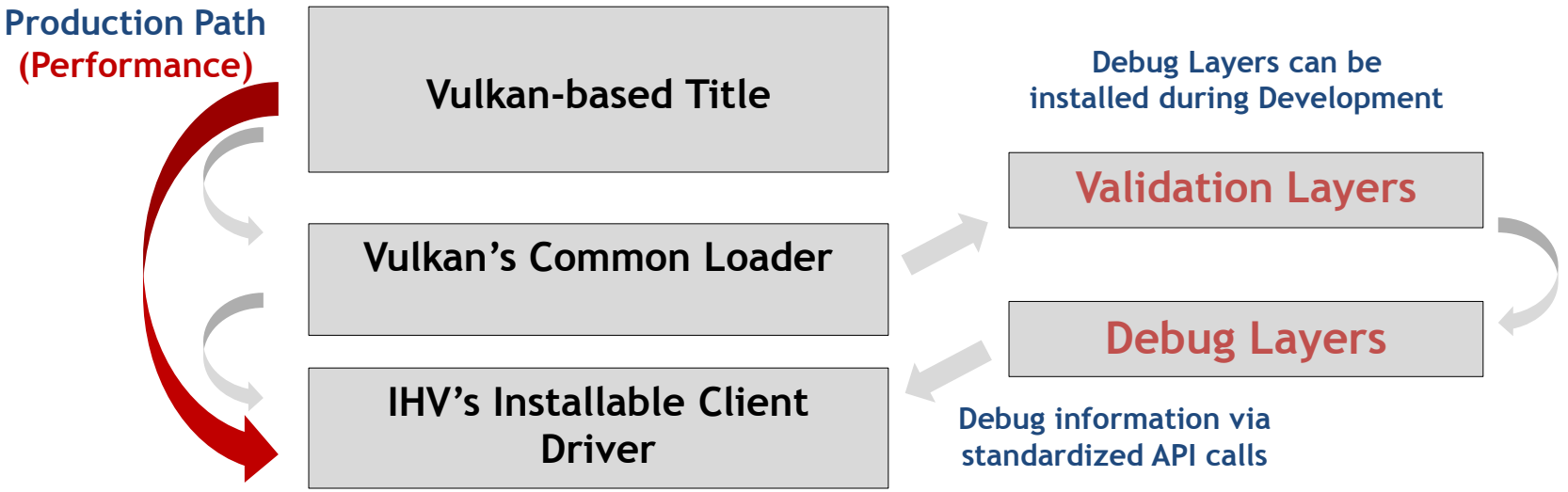
Vulkan Loader

- Loader ensures only supported layers and extensions are enabled at creation
  - Layers can be activated (enabled) either explicitly or implicitly
- Loader controls the layer call sequence
  - Chaining layers together in the proper sequence
    - Helps each layer determine where it needs to jump to next
  - Aggregates various device queries
- For IHVs, the loader:
  - Provides a helpful way to capture/debug app behavior
  - Reduces bloat from unnecessary error checking in the driver



Vulkan Tools Architecture for Layers

- Layered design for cross-vendor tools innovation and flexibility
  - IHVs plug into a common, extensible architecture for code validation, debugging and profiling during development without impacting production performance



Vulkan Validation Layers

- Validation above driver
  - Drivers generally will not include much error checking
  - Generic layers validate correct use of Vulkan across devices
  - Drivers may include multiple layers to validate vendor-specific behavior

- Vulkan supports intercepting or hooking API entry points
  - Built into the framework
  - Layers can intercept a subset of, or all, Vulkan API entry points
  - Multiple layers may be chained together to cascade their functionality and appear as a single, larger layer

LunarG Vulkan SDK Layers

Layer Name	Description
APIDump	Print API calls and their parameters and values
DrawState	Validate the descriptor set, pipeline state and dynamic state
Image	Validate texture formats and render target formats
MemTracker	Track & validate GPU memory, its binding to objects & command buffers
ObjectTracker	Track all Vulkan objects and flag invalid objects and object memory leaks
ParamChecker	Validate API parameter values
ShaderTracker	Validate the interfaces between SPIR-V modules and the graphics pipeline
Threading	Check validity of multi-threaded API usage

Vulkan Feature Sets

- **Vulkan supports hardware with a wide range of hardware capabilities**
  - Mobile OpenGL ES 3.1 up to desktop OpenGL 4.5 and beyond
- **One unified API for desktop, mobile, console, and embedded**
  - No "Vulkan ES" or "Vulkan Desktop"
- **Vulkan precisely defines a set of "fine-grained features"**
  - Features are specifically enabled at device creation time (similar to extensions)
- **Vulkan provides the mechanism but does not mandate policy**
  - Market or platform specific profiles can be defined by Khronos, platform vendors, or other interested parties

KHRONOS GROUP

Vulkan™

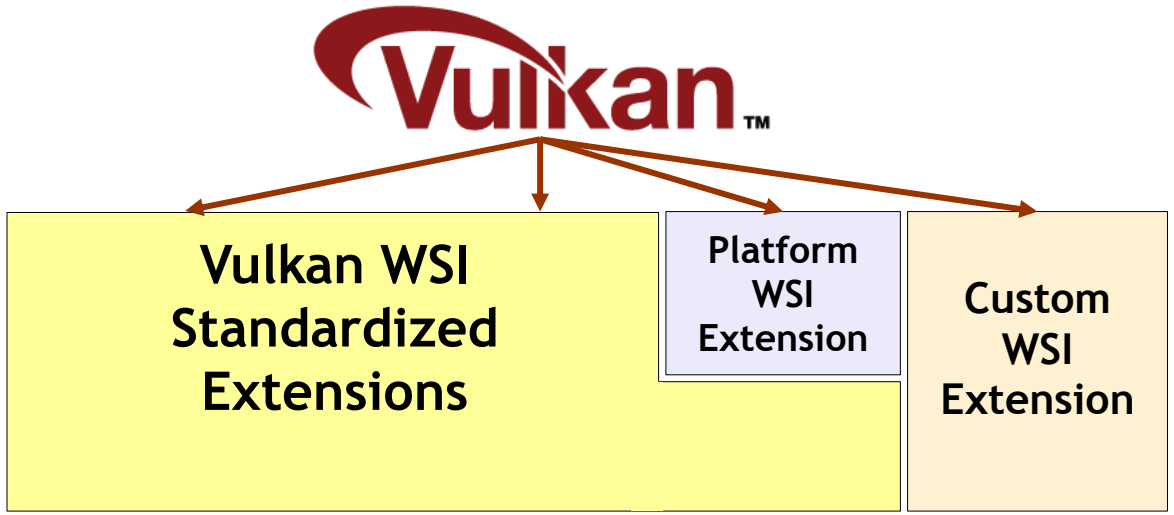
Vulkan Window System Integration Overview

SIGGRAPH, Khronos BoF, August 2015

Alon Or-bach, Samsung Electronics
Chair, Vulkan WSI Technical Sub-Group

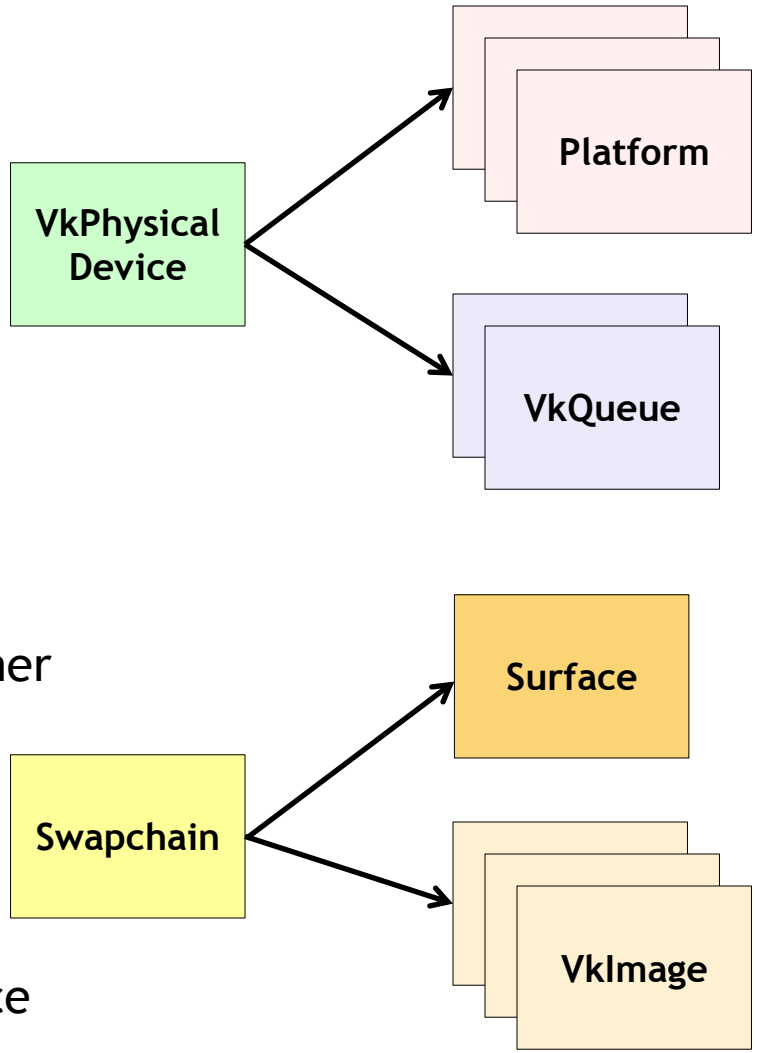
Vulkan Window System Integration (WSI)

- **Explicit control for acquisition and presentation of images**
  - Designed to fit the Vulkan API and today's compositing window systems
- **Standardized extensions - unified API for multiple window systems**
  - Works across Android, Mir, Windows (Vista and up), Wayland and X (with DRI3)
- **Platforms can extend functionality, define custom WSI stack, or have no display at all**
  - Cleanly separates device creation from window system
  - Decoupled from main API – allows progress at a different pace



Vulkan WSI: Key Concepts

- **Platform**
  - Our terminology for an OS / window system
  - e.g. Android, Windows, Wayland, X11 via XCB
- **Physical Device (VkPhysicalDevice)**
  - Exposes which Queues support presentation and which Platform(s) they support
- **Presentation Engine**
  - The Platform's compositor or display engine
- **Surface**
  - Abstraction for a Platform's window or other consumer
- **Presentable Image**
  - A VkImage created by the Platform
  - Most likely by the Presentation Engine
- **Swapchain**
  - Array of Presentable Images associated with a Surface

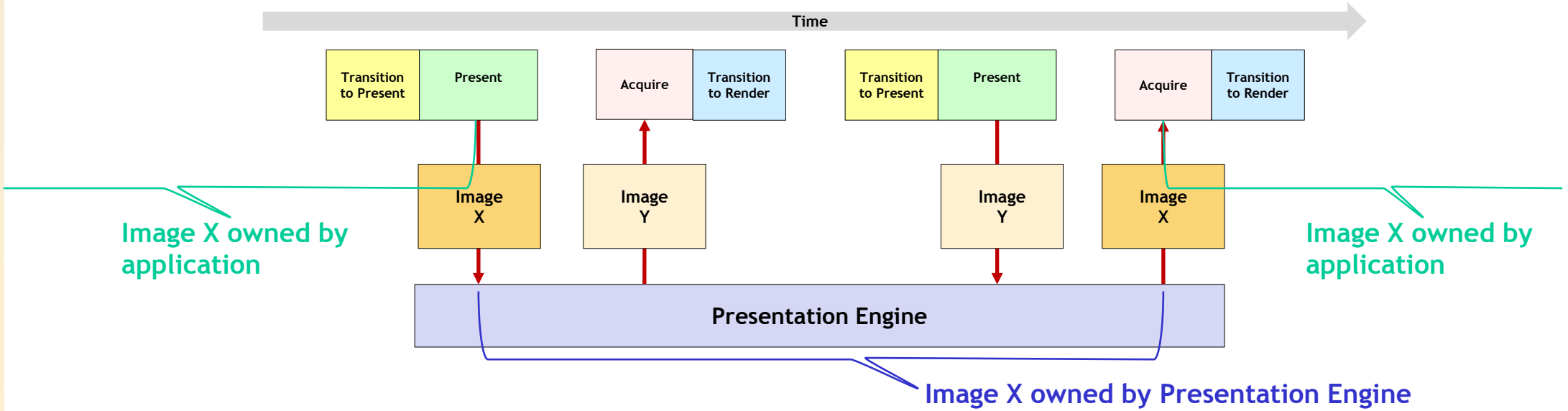


Vulkan WSI: Allocation Model

- **Several different allocation models considered**
  - Wanted a good fit with Vulkan API – upfront Command Buffer creation
  - Accommodate design of OS platforms compositors
  - Encourage greater control of swap chain where platforms support permits
- **Upfront allocation of Presentable Images**
  - Avoid last-minute surprise of which image is the render target
  - Application decides minimum number of images to request
  - Platform must allocate at least the number of images requested
- **Application has control over the order in which images are presented**
  - Any presentable image the application owns can be presented – once acquired!
  - Content of image preserved between presents
- **Clean mechanism to relay if desirable or necessary to recreate a Swapchain**
  - No surprising application with a changed image size
  - Platform informs application if current Swapchain no longer useable, or not optimal
  - Application is responsible to create a new Swapchain

Vulkan WSI: Ownership

- **Each presentable image is either owned by the application or the presentation engine**
  - This is always the case, with a clear transition of ownership. Never both simultaneously!
  - Application must only modify Presentable Images it owns
  - Presentation Engine must only display Presentable Images it owns
- **Presenting and acquiring are separate operations**
  - Presenting transfers ownership of a Presentable Image to the Presentation Engine
  - Acquiring transfers ownership of a Presentable Image to the application



Vulkan: API Changes Since GDC SIGGRAPH BoF, August 12, 2015

Jesse Barker | Principal Software Engineer
ARM

Overview

- **Developer Support**
  - Compile-Time Help for the Programmer
  - Heaps of Memory
- **Enhanced Features**
  - Pipeline Caches
  - Command Buffer Management
  - Multi-pass Render Passes

Developer Support

- Focus on type safety and const-ness
- Memory heaps are back

Pipeline Caches

- Supports efficient pipeline creation and switching
- Enables pipelines to share components
- “Save and restore” your favorites

Command Buffer Management

- **Command Pools**
  - Efficient command buffer building
  - Supports multiple usage models

- **2-Level Command Buffers**
  - Enables flexible command organization
  - Use a single level: command structure is flat
  - Use both levels: call commands in groups

Multi-pass Render Passes

- Extends the efficiency of the render pass
- Leverages two-level command buffers
- Leverages pixel locality
- What can I do with this?

Why Vulkan is Great

SIGGRAPH, Khronos BoF, August 2015

Tobias Hector
Imagination Technologies

Introduction - Topics

- **One API**
  - Vulkan is one API for all platforms
- **Efficient**
  - More efficient, CPU should not be the bottleneck
- **Parallelizable**
  - Distribute workloads, don't overload a single core
- **Explicit**
  - Tell the driver what to do - no more heuristics and guesswork
- **Architecture Positive**
  - Allow all architectures to expose their strengths

Introduction - Demo

- **Presenting with a demo**
  - OpenGL ES and Vulkan
  - Core API only

- **Aim to highlight Vulkan's advantages**
  - Many (simple) workloads to emphasize

- **Platform is a Nexus Player**
  - Android 5.1 (AOSP)
  - Mobile-class CPU/GPU
  - Will hit thermal limits

One API

- **All Platforms, All Architectures**
  - A single base API
  - Designed for modern systems
- **Capability Flags**
  - Minimum maximums, optional feature support
- **Extensions**
  - Vendor, Multi-Vendor, Khronos ratified
- **Feature Sets**
  - A range of HW exists today
  - Feature sets define functionality levels

Efficient – Why it's Important

- **GPU should not have to wait for the CPU**
  - Reduce the cost of the critical path

- **Less CPU = More GPU**
  - An SoC will have more thermal headroom

- **Increased Battery Life**
  - Less CPU usage = longer battery life!

Efficient – Command Buffer Re-Use

- **Static content doesn't need to be recalculated**
  - Just resubmit the same work
- **Commands baked into Command Buffers**
  - Can be re-used after submission
  - Can update resources
- **Cost of Queue Submission is low**

Efficient – Demo!

Parallelizable

- **Modern CPUs are multi-core**
  - Applications use multiple threads

- **Distribute workloads to multiple threads**
  - Application-managed threading
  - No global state
  - Separate command generation/submission

Parallelizable - Demo!

Explicit - Queue Submission

- **Explicit work submission**
  - Only Queue submission leads to GPU work
  - Command generation is separated

- **Application has the choice**
  - No heuristics or internal decisions about submission
  - Predictable results!

Explicit - Resource Management

- **Full explicit resource creation**
  - Ready to use when app decides

- **Resources - B.Y.O.M**
  - Explicitly allocate your own memory
  - Multiple heaps/types to allow informed choices

- **Allocation mapping to resources is a user choice**
  - Multiple resources can alias the same memory
  - Sparse features give even finer control

Architecture Positive

- **Designed for all architectures**
  - Emphasis on exposing each architecture's strengths

- **An example: Render Passes**
  - Makes render target loads/stores explicit

- **Sub-passes**
  - Multiple chained passes
  - Communicate via Pixel Local data

Architecture Positive - PLS!

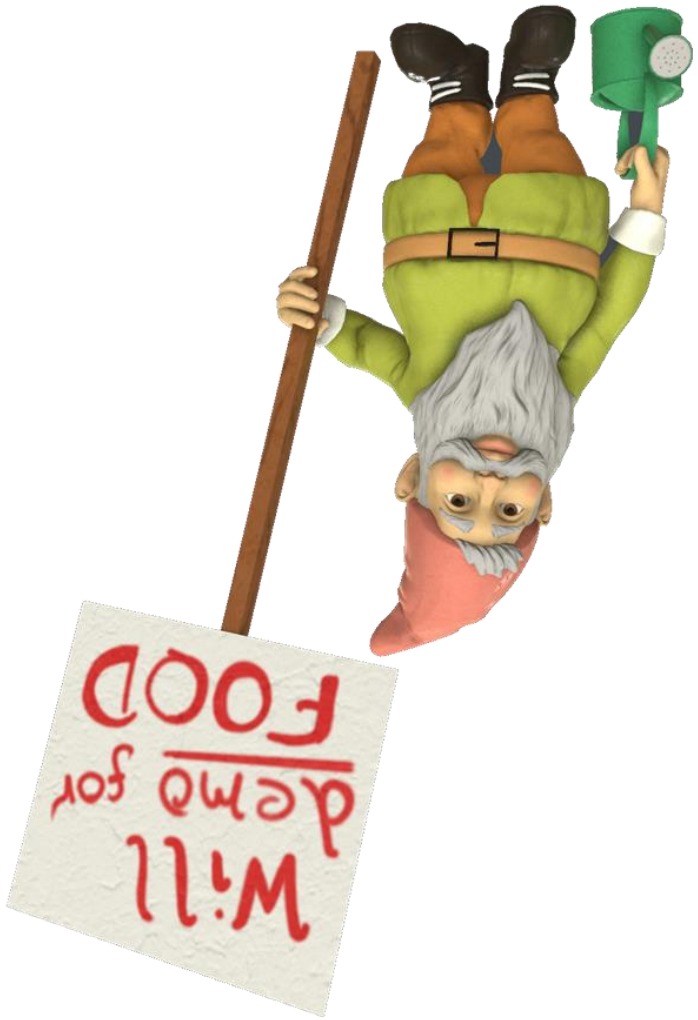
- **Designed for all architectures**
  - Emphasis on exposing each architecture's strengths

- **An example: Render Passes**
  - Makes render target loads/stores explicit

- **Sub-passes**
  - Multiple chained passes
  - Communicate via Pixel Local data

- **Pixel Local Storage in the Core API!**

Architecture Positive - Demo!



Summary

- **Vulkan gives us a lot of control**
  - At much greater efficiency
- **One API for all**
  - Architectures, Platforms
- **Come see the demo afterwards**
- **More Information**
  - khronos.org/vulkan
  - blog.imgtec.com
  - @tobskihectov
- **Questions?**

Bringing Vulkan to the 3D Ecosystem

Vulkan Working Group

android

Vulkan on Android

SIGGRAPH, August 2015

Jesse Hall | Google, Android Graphics

Vulkan: Better Mobile Graphics

- Multithreading and reduced CPU overhead:
  - Free up power and thermal headroom for the GPU
  - More draw calls and state changes per-frame per-core
  - Effectively scale to multiple CPU cores

- First-class support for tile-based architectures
  - App explicitly defines render pass boundaries and operations
  - Reusable RenderPass and Framebuffer objects amortize tiling setup
  - Allows merging render passes into a single tiling pass

Vulkan on Android

- Vulkan will be supported in a future Android release
  - Vulkan loader always present, even if device doesn’t support Vulkan
  - New <uses-feature> declaration in application manifest
  - NDK will include shader compiler, validation layers, and other tools
- Google is contributing to a comprehensive test suite
  - Open-source tests, available on all Vulkan platforms
  - Enforced for all Android devices and updates
- Android will continue to support OpenGL ES
  - Developers can choose which API best meets their needs
  - Continuing to contribute to and adopt new versions

android

Vulkan Update

Dan Ginsburg

VALVE

Vulkan Support

- Source 2 Engine ported
  - Dota 2 Reborn seeded to desktop IHVs
  - All IHVs have drivers running Dota 2 Reborn
- SDK collaboration with LunarG
  - Common loader
  - Samples
  - Reference implementation
  - SPIR-V tools
  - Trace capture and replay
  - Documentation

Why Vulkan is the Future

- DX12
  - Windows 10-only
- Metal
  - iOS/OSX-only
- Vulkan
  - Cross-platform: Windows 7/8/10, Linux, Android
  - Cross-vendor: NVIDIA, AMD, Intel, Qualcomm, Imagination Technologies, Samsung, ARM, Broadcom, Vivante.

NVIDIA Vulkan update

Why is it important to NVIDIA?

It’s open

API is designed to be extensible

- We can easily expose new GPU features

No single vendor or platform owner controls the API

Scales from low-power mobile to high-performance desktop

Can be used on any platform

It’s fast!

Vulkan driver update

Work in progress

- Supports the latest Vulkan API and LunarG SDK
- Alpha drivers available to Khronos members with NVIDIA NDA
  - Windows, Linux and Android
  - GeForce and Tegra
- Public beta available after Vulkan spec release

Demo: Vulkan cadscene

CPU overhead, multi-CPU scaling, pipeline changes

cadscene on Shield

Using the GameWorks cross-platform SDK

Same framework used for NVIDIA GameWorks samples

- https://github.com/NVIDIAGameWorks

Supports cross-platform development

- Code for Windows, Linux and Android

Demo: Vulkan cadscene on Shield

Interactive high-polygon count CAD models

Vulkan benchmarking with GFXBench 5

Zoltán Hortsin - Chief Rendering Engineer, Kishonti

Khronos BOF - August 12, 2015

Overview

- **First Vulkan benchmark**
- **Entirely new rendering engine**
  - In-house render API for Vulkan, Metal, DX12
    - Also on OpenGL 4.3+, ES 3.2, DX11
    - Same shaders using our translator
- **Concept**
  - Working title: Alien Beam
  - Huge amount of individual draw calls (2000+ per render pass)
  - Compute and render pipeline interop
- **State**
  - Currently WIP and Beta, RC expected by Q4
  - Currently v90, switch to v138 with Android as soon as possible

Alien Beam video

Rendering pipeline

- **Highlights:**
  - Deferred rendering (and forward rendering at transparents)
  - Adaptive HDR
  - Physically-based lighting
  - Raymarched volumetrics
  - Particles
- **Render passes:**
  1. Advect compute-based particles
  2. Deferred rendering: geometry pass & shading
  3. Forward rendering: transparencies (particles, light shafts)
  4. Post-process effects (including compute) and final composition

Future development plans

- **Rendering features**
  - Dynamic shadows for volumetrics
  - Image-based lighting to augment analytic lights
  - Enhanced fog
- **Additional post-effects**
  - Velocity-based motion blur & DoF
  - SSAO
  - Lens flares
- **Multi-threaded command recording**
- **Stereo rendering?**

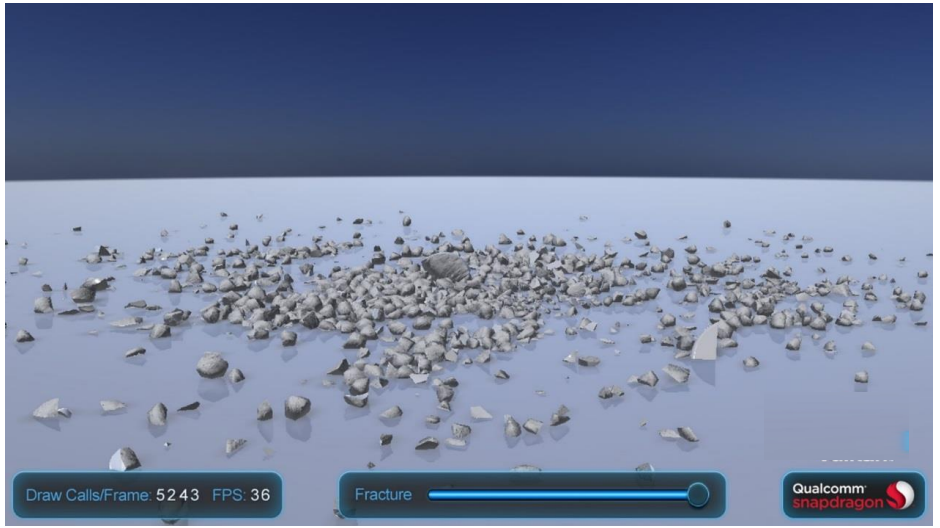
Vulkan on Adreno

Maurice Ribble, Qualcomm

Fracture Demo Using Vulkan™

Qualcomm® snapdragon™

- Preliminary Vulkan driver based on latest internal Khronos SDK header
- 10x number of draw calls over OpenGL® ES driver
- Vulkan prototype driver greatly reduces CPU overhead
  - Increases performance and reduces power consumption
- Three render passes:
  - Shadows
  - Reflections
  - Compositing
- FP16 HDR Lighting



"Qualcomm Technologies has been a major contributor to the development of Vulkan which we intend to support with our upcoming Qualcomm® Adreno™ GPUs for Qualcomm® Snapdragon™ processors."

Avinash Seetharamaiah, Senior Director of Engineering
Qualcomm Technologies, Inc.

KHRONOS™ GROUP

Fracture Demo

Intel demo

Slawomir Grajewski, Intel

KHRONOS™ GROUP

Questions?