More on Vulkan and SPIR-V: The future of high-performance graphics

KHRONOS™ GROUP

Outline

- **Welcome**
  - Neil Trevett, Khronos President (NVIDIA)
- **Vulkan project overview**
  - Tom Olson, GL Next committee chair (ARM)
- **Vulkan applications**
  - Graham Sellers, Vulkan specification co-editor (AMD)
- **SPIR-V provisional specification**
  - John Kessenich, SPIR-V specification editor (LunarG)
- **Member progress reports and demos**
  - Various members
- **Next steps**
  - Tom again
- **Q&A / Panel discussion**

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, rich media, vision, sensor and camera processing

Rigorous specifications AND conformance tests for cross-vendor portability

Acceleration APIs
BY the Industry
FOR the Industry

BOARD OF PROMOTERS

KHRONOS GROUP

Over 100 members worldwide
any company is welcome to join

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

Khronos News at GDC

- **Vulkan - next generation graphics API**
  - Low overhead, high-efficiency graphics and compute on GPUs
  - Formerly discussed as Next Generation OpenGL Initiative
  - Technical overview and demos today – spec later this year

- **SPIR-V - new shader IR supporting both graphics and compute constructs**
  - Adopted by both Vulkan *and* OpenCL 2.1
  - Provisional specification available today

Vulkan Project Overview

Tom Olson, ARM
GDC, March 2015

Vulkan project history / status

- **June to August 2014**
  - Next Generation OpenGL project launch
  - Unprecedented commitment from all sectors of the industry
  - Project disclosure and call for participation at SIGGRAPH

- **Since then…**
  - Intense focus and a lot of hard work
  - Vulkan unveil at GDC 2015

- **Status**
  - Broad agreement on basic shape and semantics of the API
  - 'alpha' header file enabling experiments
  - API spec drafting is under way
  - SPIR-V spec drafting basically complete – provisional spec available

KHRONOS™ GROUP

Vulkan vision and goals

- **An open-standard, cross-platform 3D+compute API for the modern era**
  - Compatibility break with OpenGL
  - Start from first principles

- **Goals**
  - Clean, modern architecture
  - Multi-thread / multicore-friendly
  - Greatly reduced CPU overhead
  - Architecture-neutral – full support for tile-based as well as direct renderers
  - Predictable performance through ***explicit control***
  - Improved reliability and consistency between implementations

KHRONOS GROUP

Vulkan in a nutshell

- **Modern architecture**
  - GL Context replaced by separate command buffers and dispatch queues
- **Thread-friendly**
  - Most object types are free-threaded
  - Application is responsible for synchronization
- **Low CPU overhead**
  - Error checking and dependency tracking are the application’s job
  - Can opt in to a validation layer
- **Explicit control of when work is done**
  - Shader compilation and command generation happen at predictable times
  - Immutable state specified early to move driver work away from dispatch time

KHRONOS GROUP

But first…

- **Huge thanks to the whole Vulkan team!**
  - New members are always welcome

AMD

THANKS AMD!

Vulkan Applications

Graham Sellers, AMD
GDC, March 2015

Hi! I'm Graham Sellers

- AMD's OpenGL and Vulkan architect
- Represent AMD at OpenGL ARB
- Contributor of many OpenGL features and extensions
- Author of OpenGL SuperBible
- Spent the last year or so working on Vulkan
- I'm going to whip through a complete Vulkan application from startup to tear down

This is pseudo-code, not final API
We are still finalizing some details

https://www.khronos.org/vulkan

@grahamsellers

Vulkan Application Startup

- **Vulkan is represented by an “instance”**
- **Application can have multiple Vulkan instances**
  - Each is independent
  - Eases middleware, subsystems, etc.
- **Instance is owned by the loader**
  - Aggregates drivers from multiple vendors
  - Responsible for discovery of GPUs
  - Makes multiple drivers look like one big driver supporting many GPUs

```
VK_APPLICATION_INFO appInfo = { ... };
VK_ALLOC_CALLBACKS allocCb = { ... };
VK_INSTANCE instance;

vkCreateInstance(&appInfo, &allocCb, &instance);
```

Vulkan GPUs

- **Vulkan instance creation takes:**
  - Application info – tell Vulkan about your application
  - Allocation callbacks – Vulkan will allocate system memory using *your* allocator
- **Once you have an instance, ask it about GPUs**

```
uint32_t gpuCount;
VK_PHYSICAL_GPU gpus[10];

vkEnumerateGpus(instance, ARRAYSIZE(gpus), &gpuCount, gpus);
```

- **Produces a list of GPUs, and a count**
- **GPUs can be from different vendors**
  - Integrated + discrete
  - Multiple discrete GPUs in one system
  - Cross-GPU resource sharing and explicit multi-GPU support is in API

KHRONOS GROUP

Vulkan GPU Info

- **Query information about a GPU**

```
VK_SOME_GPU_INFO_STRUCTURE info;
uint32_t infoSize = sizeof(info);

vkGetGpuInfo(gpu[0], VK_GPU_INFO_WHATEVER, &infoSize, &info);
```

- **Lots of information available about GPU**
  - Manufacturer, relative performance, memory sizes, queue types, etc.
- **Cross-GPU compatibility query**

```
VK_GPU_COMPATIBILITY_INFO compatInfo;

vkGetMultiGpuCompatibility(gpuA, gpuB, &compatInfo);
```

- **Compatibility info indicates**
  - Full sharing, sharing of specific resources, or no compatibility at all

KHRONOS GROUP

Vulkan Devices

- **Construct a device instance from a GPU**

```
VK_DEVICE_CREATE_INFO info = { ... };
VK_DEVICE device;

vkCreateDevice(gpu, &info, &device);
```

- **Creation info contains information about:**
  - Number and type of queues required
  - Which extensions you want to use
    - Extensions are 'opt-in' – cannot accidentally use an extension
  - Level of validation
    - Drivers generally will not include much, if any, error checking
    - Layers above can validate at various levels
    - Drivers may include multiple layers to validate vendor-specific behavior

Vulkan Queues

- **Get queue handles from the device**

```
VK_QUEUE queue;

vkGetDeviceQueue(device, 0, 0, &queue);
```

- **Queues are represented using two indices**
  - Node ordinal
    - Node ordinal represents a “family” of queues, which are directly compatible
  - Queue index
    - Each queue family can have many queue instances
- **Queues encapsulate**
  - Functionality – graphics, compute, DMA
  - Scheduling – independently scheduled, asynchronous

KHRONOS GROUP

Vulkan Command Buffers

- GPU commands are batched in command buffers

```
VK_CMD_BUFFER_CREATE_INFO info;
VK_CMD_BUFFER cmdBuffer;

vkCreateCommandBuffer(device, &info, &cmdBuffer);
```

- Create as many command buffers as you need
- Command buffer creation info includes
  - Which queue family you want to submit commands to (node ordinal)
  - Information about how aggressively drivers should optimize for GPU performance
  - etc.

Vulkan Commands

- **Commands are inserted into command buffers**

```
VK_CMD_BUFFER_BEGIN_INFO info = { ... };
vkBeginCommandBuffer(cmdBuf, &info);

vkCmdDoThisThing(cmdBuf, ...);
vkCmdDoSomeOtherThing(cmdBuf, ...);

vkEndCommandBuffer(cmdBuf);
```

- **Driver heavy lifting happens here**
  - Build many command buffers from many threads
  - Re-use command buffers
  - Spend time here optimizing work, not last minute right before draw
  - Big packages of immutable state make the workload less regardless

Vulkan Shaders

- **Vulkan shaders are compiled up-front**

```
VK_SHADER_CREATE_INFO info = { ... };
VK_SHADER shader;

vkCreateShader(device, &info, &shader);
```

- **Shader creation info contains**
  - Pointer to shader source
    - SPIR-V – portable, vendor-neutral, *open*, *extensible* shader binary
    - Other IRs could be supported through the same interfaces
  - Additional optional information

- **Compile shaders from multiple threads**
  - Driver will do as much work as it can right here

Vulkan Pipeline State

- **Pipeline state is fully compiled**

```
VK_GRAPHICS_PIPELINE_CREATE_INFO info = { ... };
VK_PIPELINE pipeline;

vkCreateGraphicsPipeline(device, &info, &pipeline);
```

- **Creation info contains**
  - Compiled shaders
  - Blend, depth, culling, stencil state, etc.
  - List of states that need to be mutable
- **Pipelines can be serialized and deserialized**

```
uint32_t dataSize = DATA_SIZE;
void* data = malloc(DATA_SIZE);

vkStorePipeline(pipeline, &dataSize, data);
...
vkLoadPipeline(device, dataSize, data, &pipeline)
```

Vulkan Mutable State

- Some pipeline state is mutable or *dynamic*
- Represented by smaller state objects

```
VK_DYNAMIC_VP_STATE_CREATE_INFO vpInfo = { ... };
VK_DYNAMIC_VP_STATE_OBJECT vpState;

vkCreateDynamicViewportState(device, &vpInfo, &vpState);

VK_DYNAMIC_DS_STATE_CREATE_INFO dsInfo = { ... };
VK_DYNAMIC_DS_STATE dsState;

vkCreateDynamicDepthStencilState(device, &dsInfo, &dsState);
```

KHRONOS GROUP

Vulkan Resources

- Resources have a CPU and a GPU component
- CPU side is allocated using a vkCreate* function:

```
VK_IMAGE_CREATE_INFO imageInfo = { ... };
VK_IMAGE image;
vkCreateImage(device, &imageInfo, &image);

VK_BUFFER_CREATE_INFO bufferInfo = { ... };
VK_BUFFER buffer;
vkCreateBuffer(device, &bufferInfo, &buffer);
```

- It is the application's responsibility to allocate GPU memory for resources…

KHRONOS GROUP

Vulkan GPU Memory

- **Query objects for their memory requirements**

```
VK_IMAGE_MEMORY_REQUIREMENTS reqs;
size_t reqsSize = sizeof(reqs);

vkGetObjectInfo(image,
                VK_INFO_TYPE_IMAGE_MEMORY_REQUIREMENTS,
                &reqsSize, &reqs);
```

- **Application allocates GPU memory**

```
VK_MEMORY_ALLOC_INFO memInfo = { ... };
VK_GPU_MEMORY mem;
vkAllocMemory(device, &memInfo, &mem);
```

- **Bind application-owned GPU memory to objects**

```
vkBindObjectMemory(image, 0, mem, 0);
```

Vulkan Descriptors

- Vulkan resources are represented by *descriptors*
- Descriptors are arranged in *sets*
- Sets are allocated from *pools*
- Each set has a layout, which is known at pipeline creation time
  - Layout is shared between sets and pipelines and must match
  - Layout represented by object, passed at pipeline create time
- Can switch pipelines which use sets of the same layout
- Many sets of various layouts are supported in one pipeline in a *chain*

```
vkCreateDescriptorPool(...);
vkCreateDescriptorSetLayoutChain(...);
vkCreateDescriptorSetLayout(...);
vkAllocDescriptorSets(...);
```

KHRONOS GROUP

Vulkan Render Passes

- Render passes represent logical phases of a frame
- Render passes are explicit objects

```
VK_RENDER_PASS_CREATE_INFO info = { ... };
VK_RENDER_PASS renderPass;

vkCreateRenderPass(device, &info, &renderPass);
```

- Render pass contains a lot of information about rendering
  - Layout and types of framebuffer attachments
  - What to do when the render pass begins and ends
  - The region of the framebuffer that the render pass may effect
- Vitally important information for tile-based and deferred renderers
  - … but also very helpful for traditional forward-renderers!

Vulkan Drawing

- Draws are placed inside render passes
- Executed in the context of a command buffer

```
VK_RENDER_PASS_BEGIN beginInfo = { renderPass, ... };

vkCmdBeginRenderPass(cmdBuffer, &beginInfo);

vkCmdBindPipeline(cmdBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);
vkCmdBindDescriptorSets(cmdBuffer, ...);
vkCmdDraw(cmdBuffer, 0, 100, 1, 0);

vkCmdEndRenderPass(cmdBuffer, renderPass);
```

- Pipelines, dynamic state objects and other resources bound to command buffers
- All draw types supported
  - Indexed and non-indexed, direct and (multi-)indirect, compute dispatches, etc.

Vulkan Synchronization

- **Work is synchronized using event primitives**

```
VK_EVENT_CREATE_INFO info = { ... };
VK_EVENT event;

vkCreateEvent(device, &info, &event);
```

- **Events can be set, reset, queried and waited on**

```
vkSetEvent(...);
vkResetEvent(...);
vkGetEventStatus(...);
vkCmdSetEvent(...);
vkCmdResetEvent(...);
vkCmdWaitEvents(...);
```

- **Command buffers can signal events as they complete execution**

Vulkan Resource State

- **Operations in command buffers are demarked by pipeline barriers**
- **Barriers can wait on and signal events**
- **Barriers can transition resources from state to state**
  - Renderable
  - Readable as texture
  - etc.

```
VK_IMAGE_MEMORY_BARRIER imageBarrier = { ... };
VK_PIPELINE_BARRIER barrier = { ..., 1, &imageBarrier };

vkCmdPipelineBarrier(cmdBuffer, &barrier);
```

- **Drivers do not track state**
  - Applications are responsible for state tracking
  - If you get it wrong, we will happily render garbage or crash
  - Validation layer will track state (slowly) and scream at you when you screw up

Vulkan Work Enqueue

- **Work is executed on queues belonging to devices**
- **Completed command buffers are sent to queues for execution**

```
VK_CMD_BUFFER commandBuffers[] = { cmdBuffer, ... };
vkQueueSubmit(queue, 1, commandBuffers, fence);
```

- **Queues own memory residency**
  - Driver will not track memory residency for you

```
vkQueueAddMemReference(queue, mem);
vkQueueRemoveMemReference(queue, mem);
```

- **Queues can also signal and wait on semaphores for object ownership**

```
vkQueueSignalSemaphore(queue, semaphore);
vkQueueWaitSemaphore(queue, semaphore);
```

Vulkan Presentation

- **Presentation is how we get images to the screen**
- **Displayable resource represented by a special kind of image**
  - Bindable to framebuffers
  - Created by platform-specific modules called WSI (Window System Interface)
- **Defining a small number (~2?) of WSI bindings**
  - One for compositing systems where the compositor owns the displayable surface
  - One for systems that allow presentation of application-owned surfaces
- **WSI also deals with things like:**
  - Enumerating display devices and video modes
  - Going full screen
  - Controlling vsync
- **Presentations enqueued along with command buffers**

Vulkan Teardown

- **Application responsible for object destruction**
  - Must be correctly ordered
  - No reference counting
  - No implicit object lifetime
- **Do not delete objects that are still in use!**
- **Most objects destroyed with:**

```
vkDestroyObject(object);
```

- **Some objects are “special”:**

```
vkDestroyDevice(device);
vkDestroyInstance(instance);
```

Vulkan AZDO?

- **Vulkan is already PDCTZO (Pretty Darn Close to Zero Overhead)!**
  - Very little validation unless you opt-in
  - You manage everything – virtually no driver funky business
  - Much better abstraction of the hardware – no complex mapping of API to silicon
- **Submit the same command buffer many times**
  - Amortized cost of building command buffer literally approaches zero
- **Bindless**
  - Debatable need – descriptor sets can be of arbitrary size
  - Explicit memory residency already in API
- **Sparse**
  - Yes
- **MultiDrawIndirect**
  - Yes
- **Shader Draw Parameters**
  - Yes

Vulkan Summary

- **Vulkan is not "low level" - just a better abstraction of modern hardware**
- **Vulkan is very low overhead**
  - Reduced CPU utilization means more cycles for your application
  - Explicit threading support means you can go wide without worrying about graphics APIs
  - Building command buffers once and submitting many times means low amortized cost
- **Cross-platform, cross-vendor**
  - Not tied to single OS (or OS version)
  - Not tied to single GPU family or vendor
  - Not tied to single architecture
    - Desktop + mobile, forward and deferred, tilers all first class citizens
- **Open, extensible**
  - Khronos is an open standards body
    - Collaboration from a wide cross-section of industry, IHVs + ISVs, games, CAD, AAA + casual
  - Full support for extensions, layering, debuggers, tools
  - SPIR-V fully documented – *write your own compiler!*

Vulkan™

Thanks!

https://www.khronos.org/vulkan

@grahamsellers

Vulkan™

SPIR™ SPIR-V Provisional

GDC, San Jose
March 2015

Standard
Portable
Intermediate
Representation

Goal:

1) Portable binary representation of shaders and compute kernels for GPUs and parallel computers
2) Target for OpenCL C/C++, GLSL, and other shader languages

Enables compiler ecosystem for more portable shaders

Why use SPIR?

Without SPIR:

- **Vendors shipping source**
  - Risk IP leakage
- **Limited Portability**
  - No ISV control over front end
  - Different front end semantics per vendor
- **Higher runtime compilation time**

With SPIR:

- **Ship a single binary**
  - Requires tools to decipher; protecting IP
- **Improved Portability**
  - ISV can create their own front end tool chain
  - Multiple ISVs can share a common front end
- **Reduced runtime compilation time**
  - Some steps are offloaded

Opportunity to unleash innovation:
Domain Specific Languages, C++ Compilers, ….

What is SPIR-V?

- **New intermediate language for input to Khronos graphics and compute APIs**
  - Fully specified Khronos-defined standard
  - Can natively represent Khronos graphics and compute idioms
    - E.g., implicit derivatives with control-flow constraints
  - Memory and execution models for all GLSL and OpenCL high-level languages

- **Core for Vulkan**
  - The only language accepted by the API
  - Exposes machine model for Vulkan
  - Fully supports the GLSL/ESSL shader languages
  - Other shading languages easily target SPIR-V

- **Core for OpenCL 2.1**
  - Supports OpenCL 1.2, 2.0, 2.1 kernel languages

SPIR-V shader-language support

- **Compiler chain split in two**
  - Front end compiler emits SPIR-V portable binary IL, offline
  - SPIR-V IL is compiled to machine-specific binary by driver, online

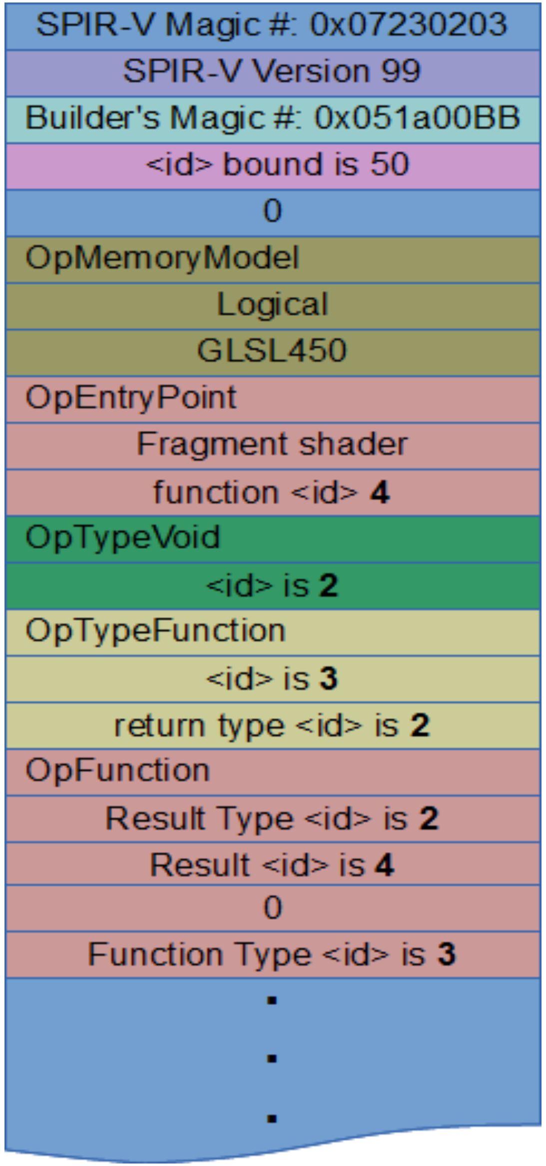
- **Front end NOT required in driver**
  - Khronos working on offline language front ends

SPIR-V: A Deeper Look

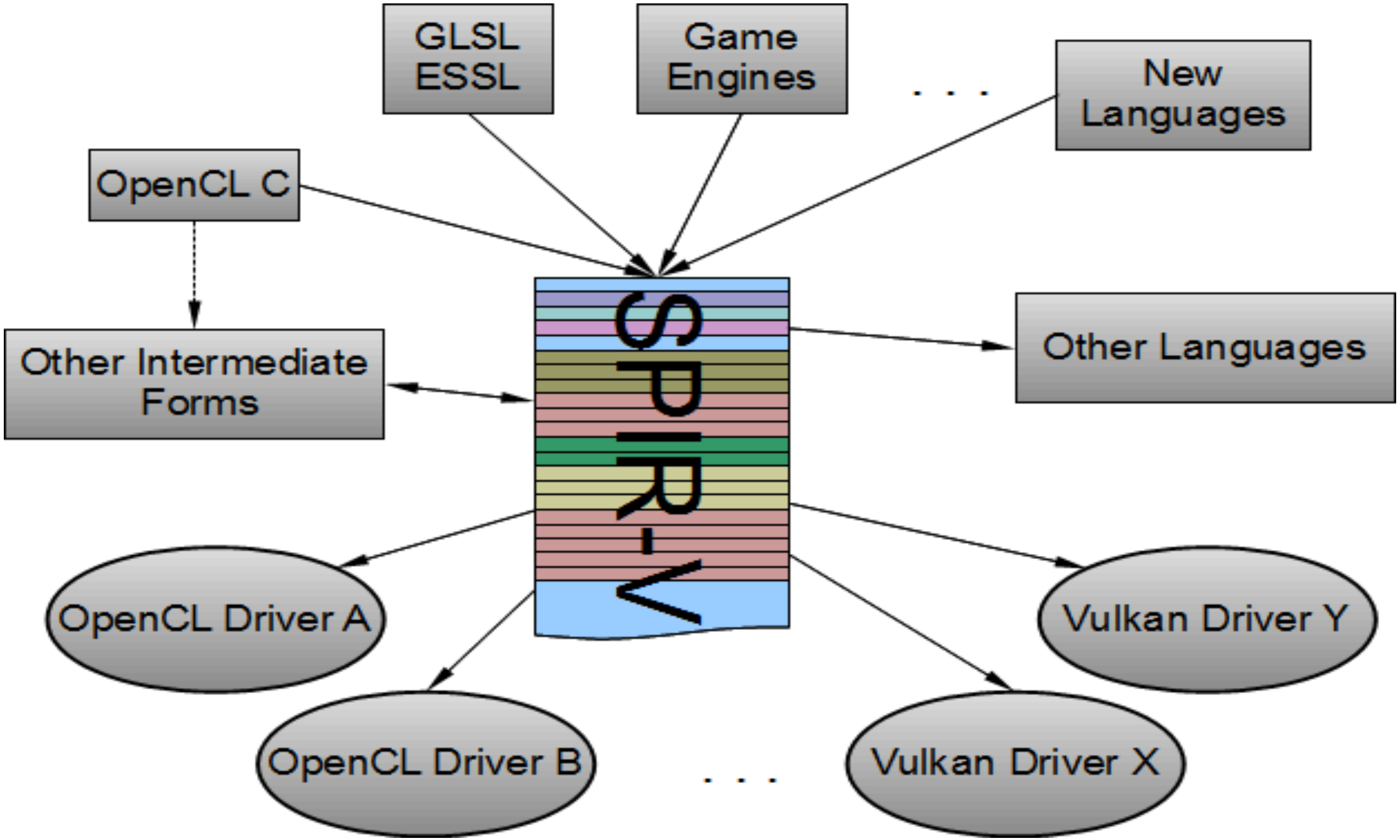
- **A Binary Intermediate Language**
  - A linear stream of words (32-bits)
- **Functions inside a module contain a CFG (control-flow graph) of basic blocks**
- **Load/Store instructions are used to access declared variables**
- **Intermediate results are represented using single static-assignment (SSA)**
- **Data objects are represented logically, with hierarchical type information**
  - e.g. No flattening of aggregates or assignment to physical registers
- **Selectable addressing model**
  - Allow usage of pointers, or dictate a memory model which is purely logical
- **Can be easily extended**
- **Support debug information that can be safely stripped without changing the semantics of SPIR-V modules.**

SPIR-V is a Binary Form

- Stream of words
- 32-bits wide
- Not a file format
  - This is the form passed through entry point
  - But, works well to start file with the magic number and directly store the stream
    - Deduce endianness from magic number

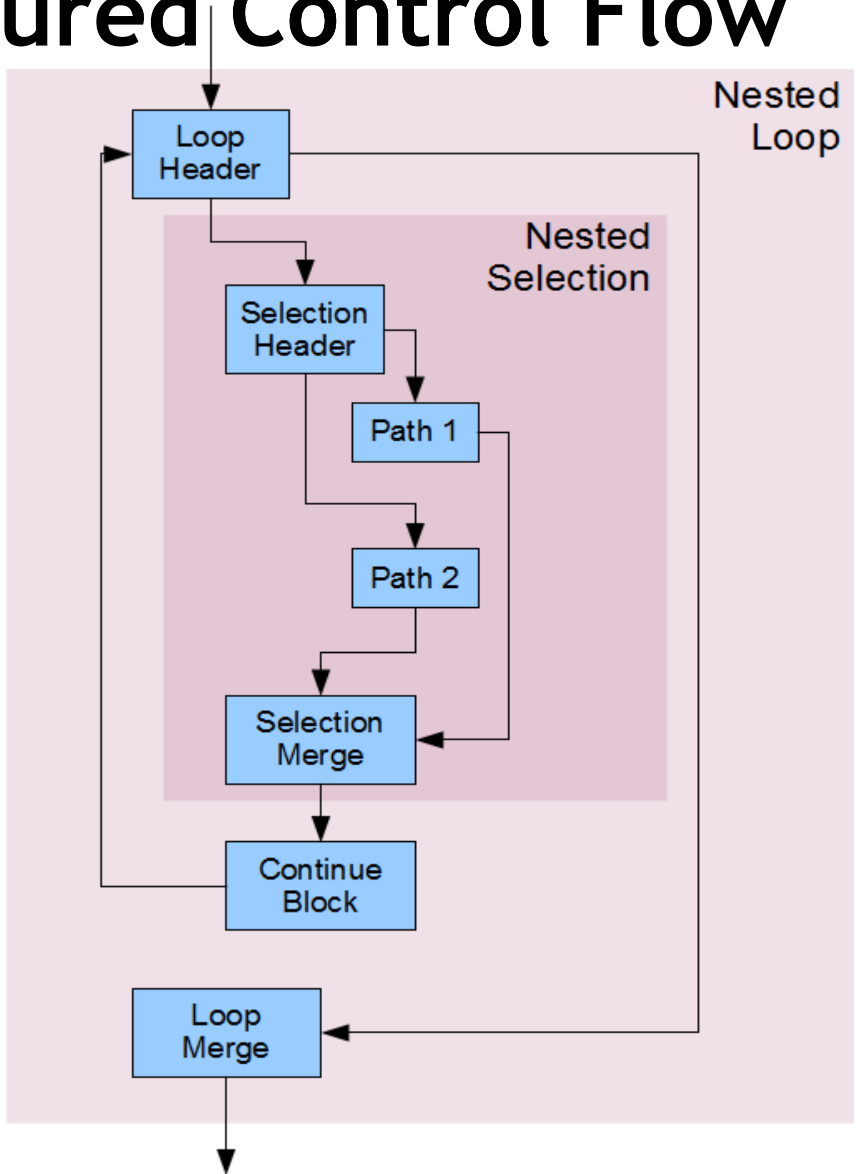


SPIR-V is a Common Intermediate Form



Structured Control Flow

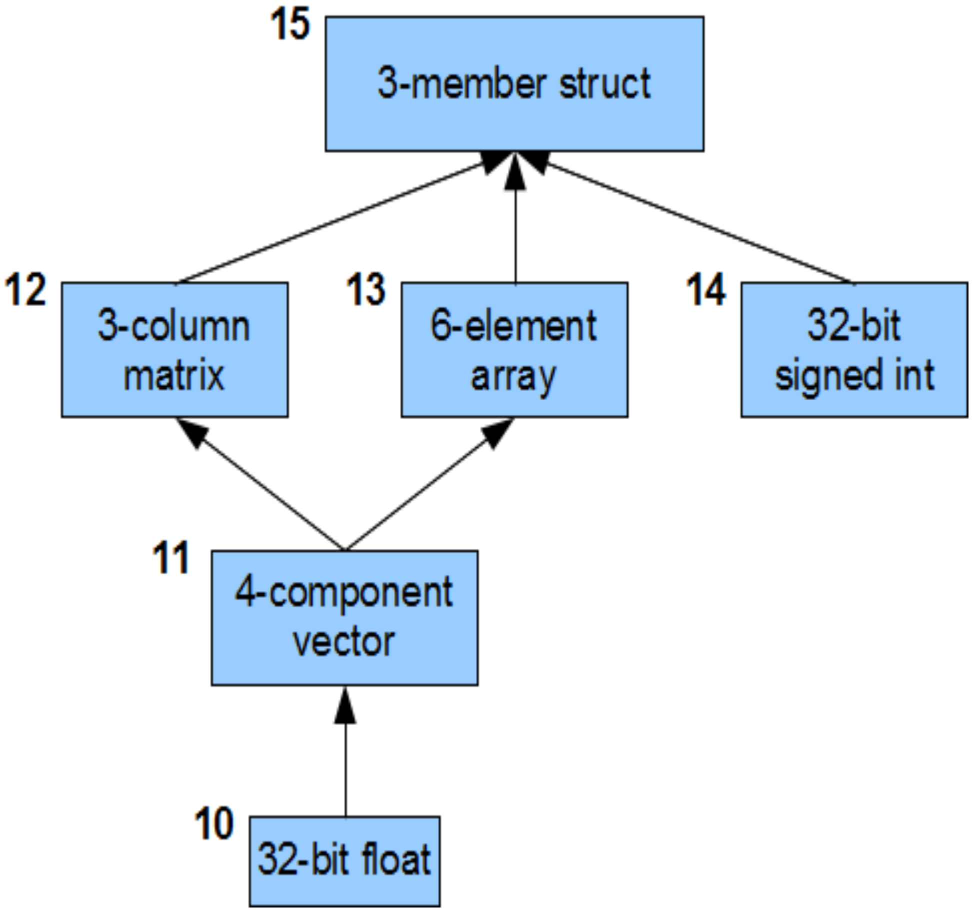
```
for (...) {

    if (...)

        ...
    else

        ...


    ...


}
```




```
11: Label
    ...
    LoopMerge 12 NoControl
    BranchConditional 18 19 12
19:   Label
22:   ...
      SelectionMerge 24 NoControl
      BranchConditional 22 23 28
23:     Label
        ...
        Branch 24
28:     Label
        ...
        Branch 24

24:   Label
      ...


      Branch 11


12: Label
```

Hierarchical Types, Constants, and Objects

```
struct {
    mat3x4;
    vec4[6];
    int;
};
```




```
10: OpTypeFloat 32
11: OpTypeVector 10 4
12: OpTypeMatrix 11 3
13: OpTypeArray 11 6
14: OpTypeInt 32 1
15: OpTypeStruct 12 13 14
```

SPIR-V: A Deeper Look (Summary)

- **A Binary Intermediate Language**
  - A linear stream of words (32-bits)
- **Functions inside a module contain a CFG (control-flow graph) of basic blocks**
- **Load/Store instructions are used to access declared variables**
- **Intermediate results are represented using single static-assignment (SSA)**
- **Data objects are represented logically, with hierarchical type information**
  - e.g. No flattening of aggregates or assignment to physical registers
- **Selectable addressing model**
  - Allow usage of pointers, or dictate a memory model which is purely logical
- **Can be easily extended**
- **Support debug information that can be safely stripped without changing the semantics of SPIR-V modules.**

Call to Action

- **Seeking feedback now on SPIR-V provisional**
  - A Provisional specification, subject to change based on your feedback
  - Spec available at www.khronos.org/spir
  - Provide feedback at https://www.khronos.org/spir_feedback_forum
  - White paper https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf

- **Innovate on the front end**
  - New languages, abstractions
  - Target production quality Back ends

- **Innovate on the back end**
  - New target platforms: Multi core, Vector, VLIW…
  - Reuse production quality frontends
  - Other high-level languages and IRs/ILs

- **Innovate on Tooling**
  - Program analysis, optimization

Member Progress Reports and Demos

Vulkan™ meets Mali™

Jesse Barker
Software Engineer, ARM

Vulkan investigations at ARM

- Prototype Vulkan driver for ARM® Mali™ Midgard GPU architecture
  - Intended to verify that Vulkan is a good fit to the architecture
  - Initial port on Arndale Octa (4+4 ARM Cortex™ A-15/7, Mali T-628 MP6)

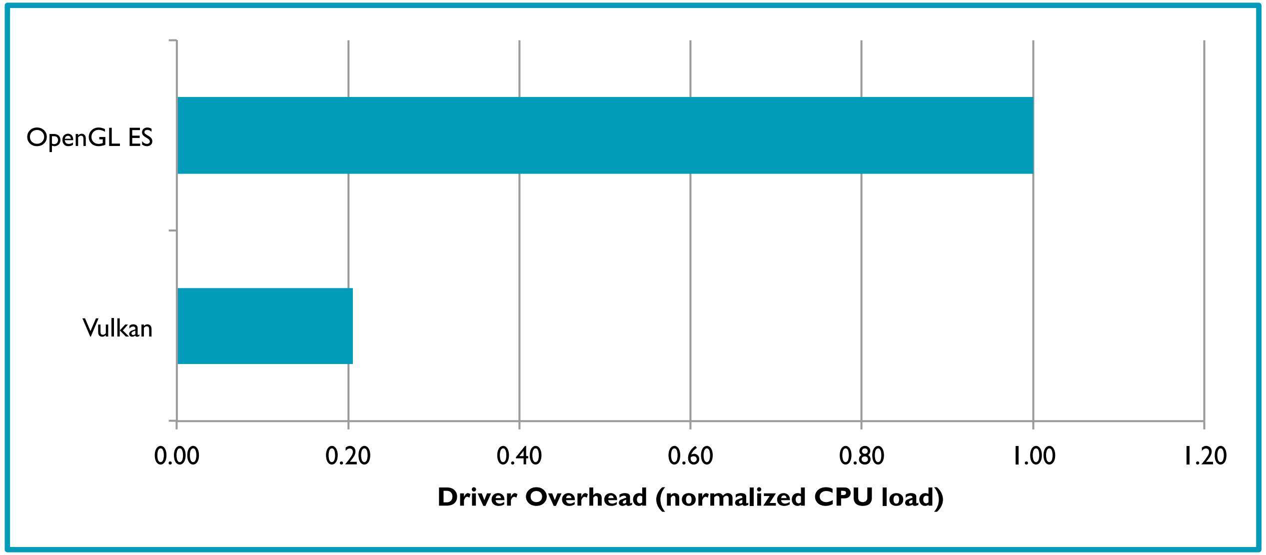
- Caveats
  - Partial implementation – critical functions only, and some shortcuts
  - Built on top of an OpenGL ES / OpenCL HAL, not optimized for Vulkan

Experiment

- Draw call microbenchmark
  - 1000 meshes, 3 materials
  - Minimal state change between meshes
  - Measure CPU cycles in driver
  - Compare to OpenGL ES

Results

- For this test case, 79% reduction in CPU cycles spent in driver!

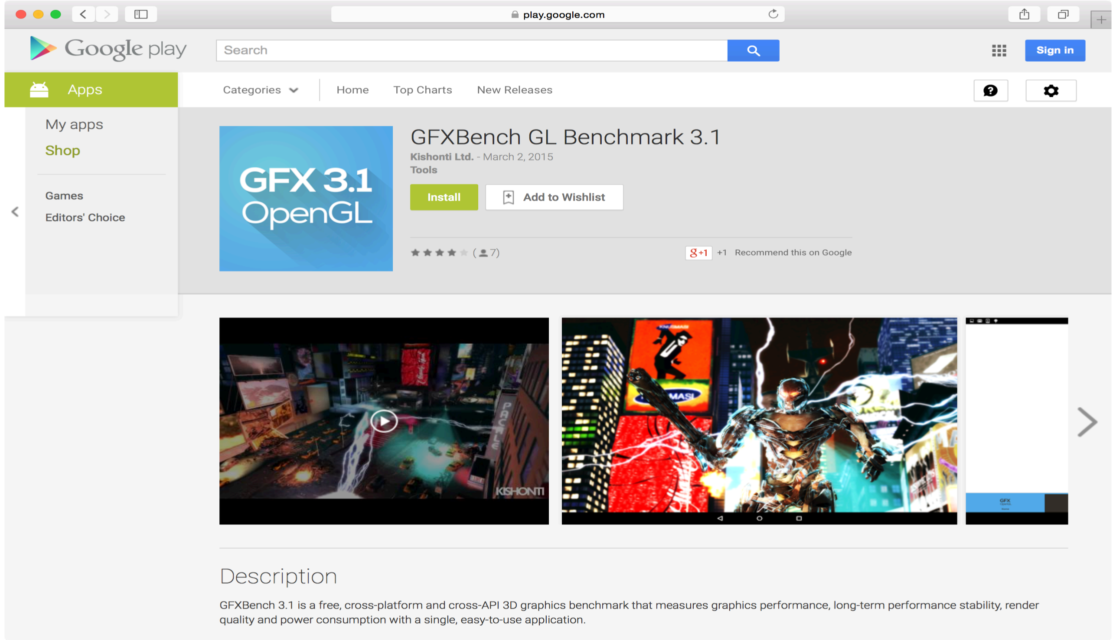


Kishonti Informatics

KHRONOS™ GROUP

GFXBench 3.1

- Graphics benchmark for OpenGL ES 3.1
- In Google Play store this week!
- Adds compute shaders and new high-precision low level tests



GFXBench 4.0

- Graphics benchmark to showcase OpenGL ES 3.1 with Android Extension Pack (AEP)
- Outdoor car chase scene with adaptive tessellation, HDR rendering, physically-based materials, compute post effects, dynamic reflections and shadows
- Also sports geometry shaders and ASTC texture compression
- Public release soon

GFXBench 5.0

- Entirely new engine aimed at benchmarking low-level graphics APIs (Vulkan, DX12, Metal)
- Concept is a night outdoor scene with aliens
- Still in pre-alpha, but shows the most important concepts
- Is showcased running Vulkan at Intel's GDC booth

Imagination Technologies

KHRONOS™ GROUP

Intel

KHRONOS™ GROUP


PHOTO: FLICKR IDLEMIND
Introducing:
Vulkan™
The Future of Graphics & Computing
Click here to
LEARN MORE

NVIDIA


PHOTO: FLICKR IDLEMIND
Introducing:
Vulkan
TM
The Future
of Graphics &
Computing
Click here to
LEARN MORE

Valve

GLAVE debugger

LunarG.com/Vulkan

Call to Action

- **Give us feedback on Vulkan and SPIR**
  - Links provided on Khronos forums
  - https://www.khronos.org/spir_v_feedback_forum
  - https://www.khronos.org/vulkan/vulkan_feedback_forum

- **Any company or organization is welcome to join Khronos for a voice and a vote in any of these standards**
  - www.khronos.org

- **Watch this space!**
  - Initial specs and implementations coming later this year

Q&A / Panel Discussion

KHRONOS™ GROUP