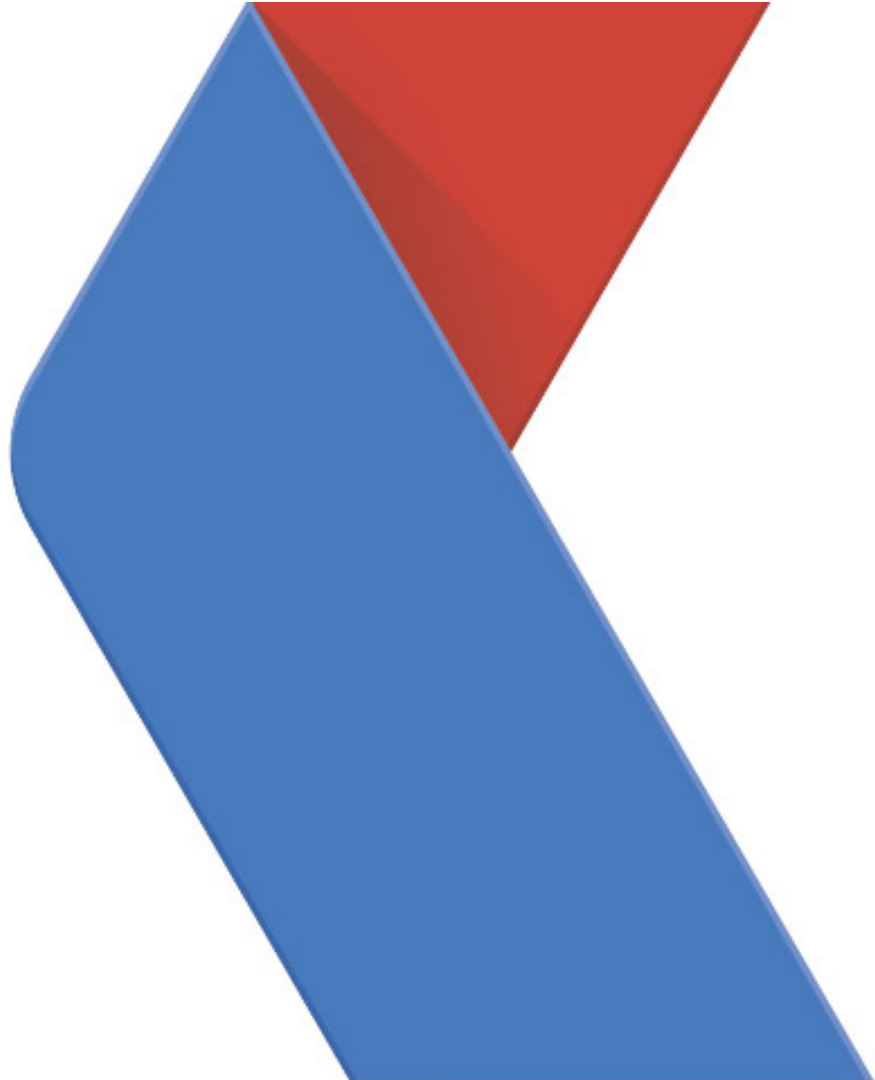


Vulkan: High- Performance 3D Graphics for Android

December 2016



About This Talk

Why a New Graphics API?

What is Vulkan?

Getting Started on Android

Tips and Best Practices for Mobile/Android

Is Vulkan Right for My App?

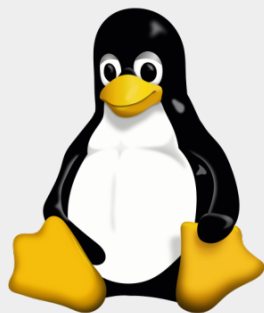
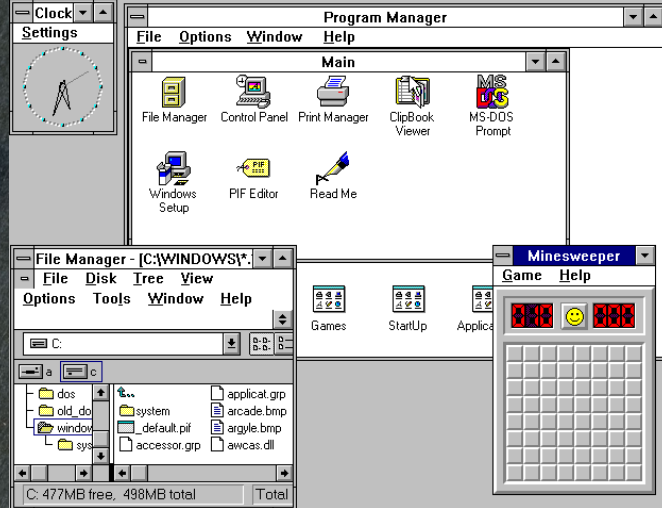




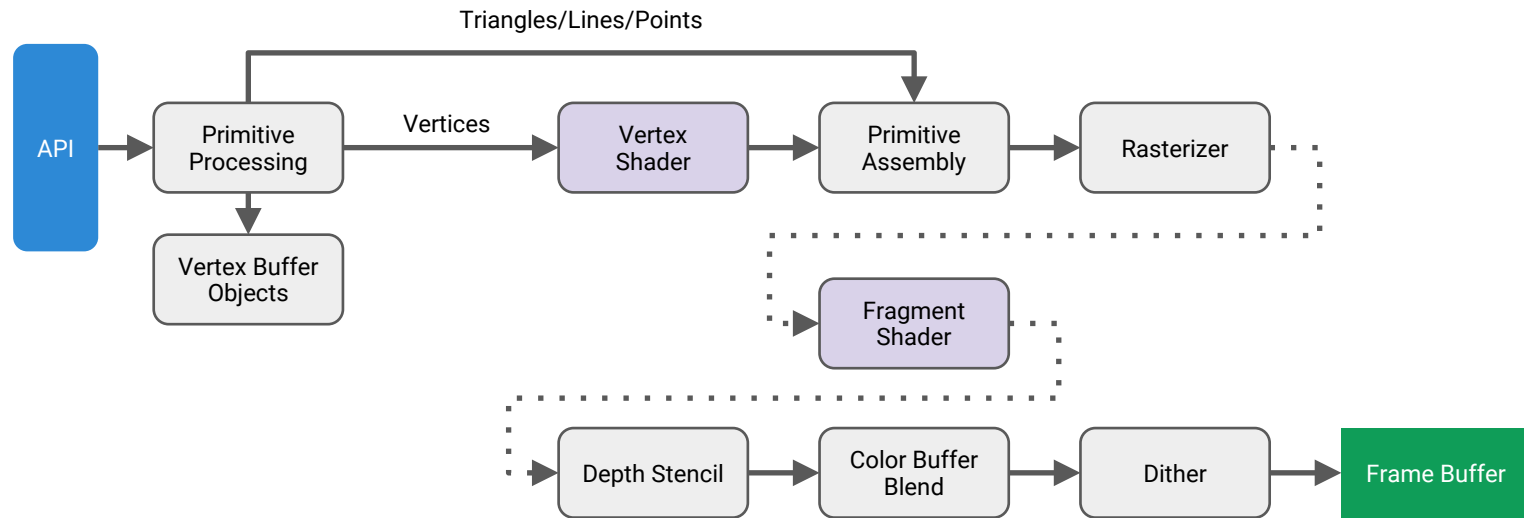
Starting Over

OpenGL

Released in 1992



OpenGL ES 2.0 Pipeline

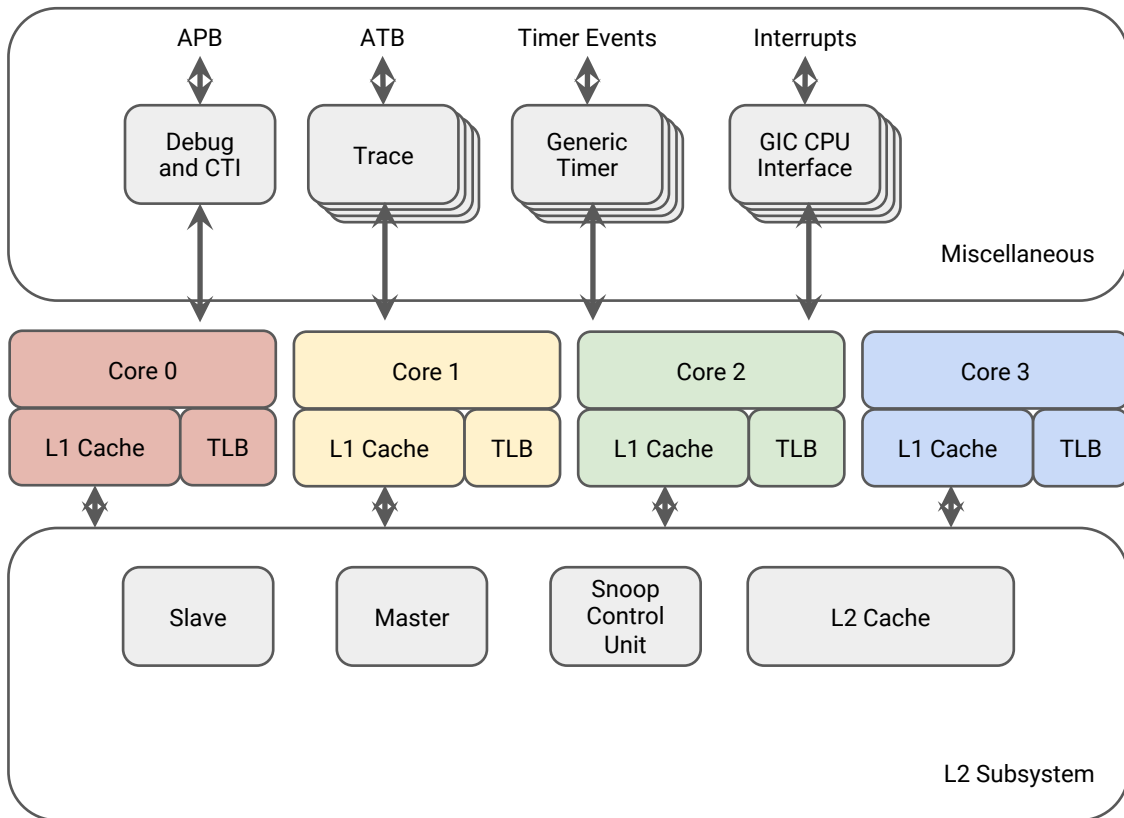


Mobile Computing Today

Multi-Core

> 2.0 GHz

Multi-Level Cache



Power and Clock Speed

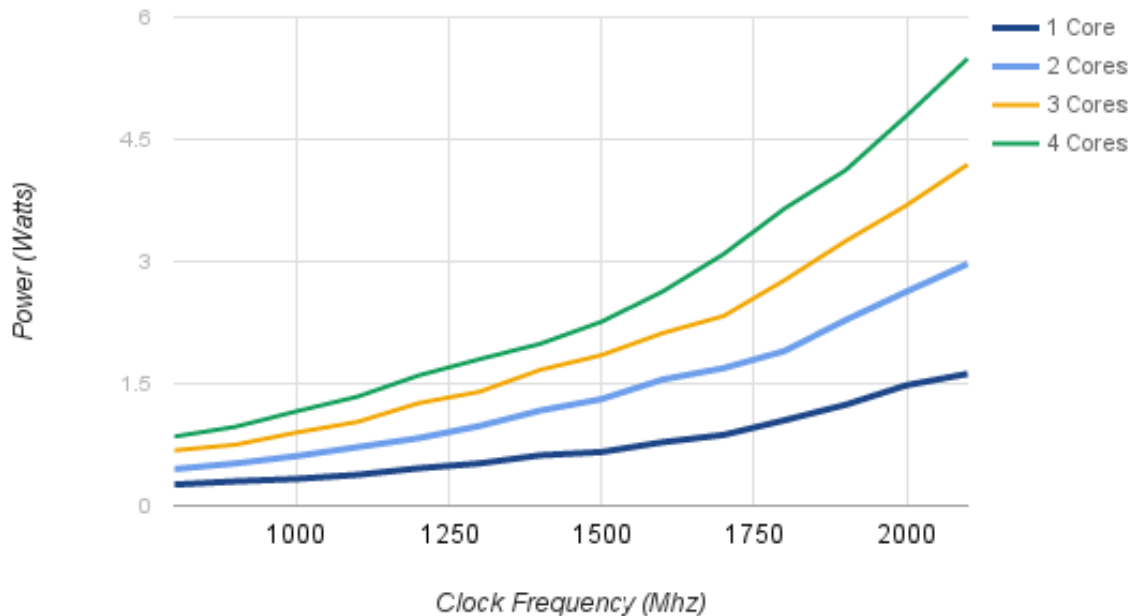
Multi-Core

> 2.0 GHz

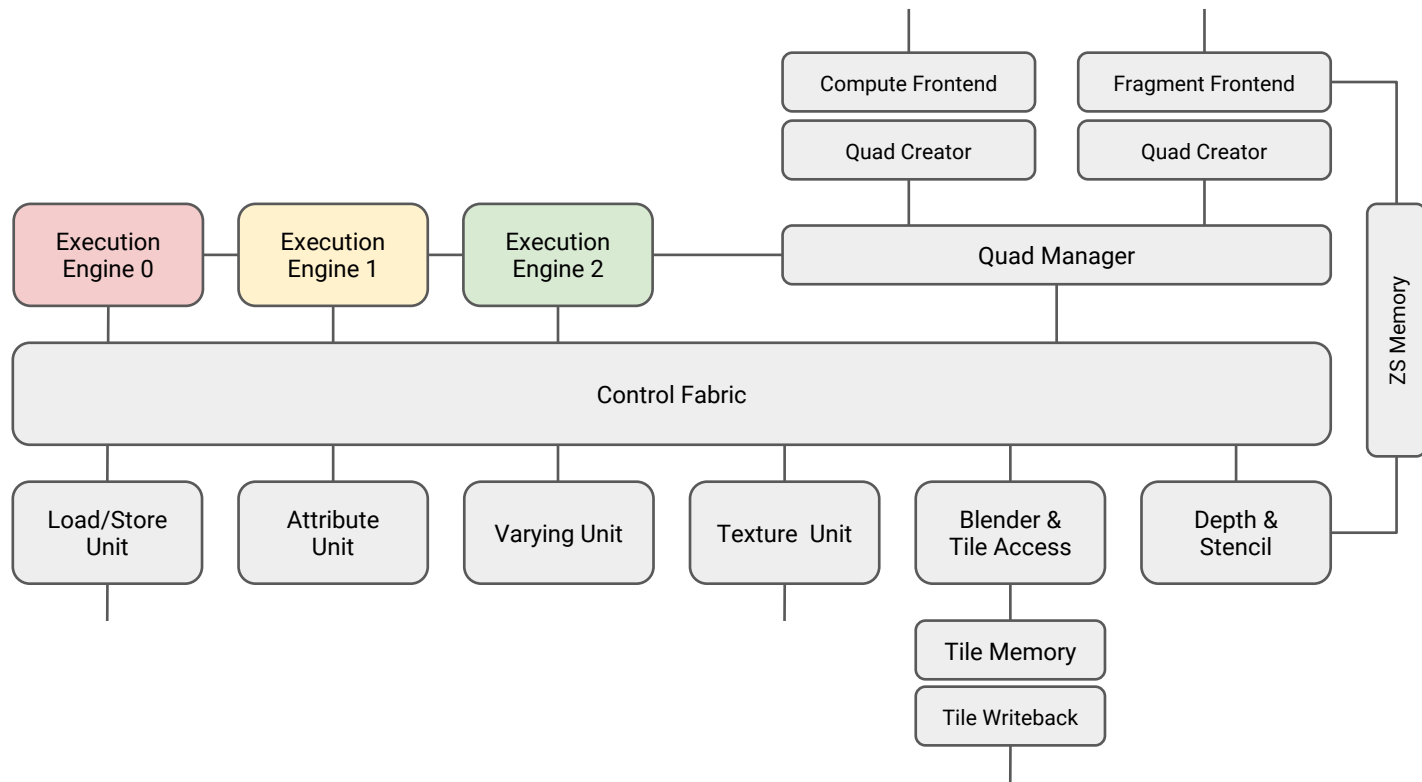
Multi-Level Cache

$$P=CV^2f$$

Clock Frequency and Power Scaling



Modern GPU Design





Enter Vulkan

Vulkan

- Explicit
- Modern
- Extensible



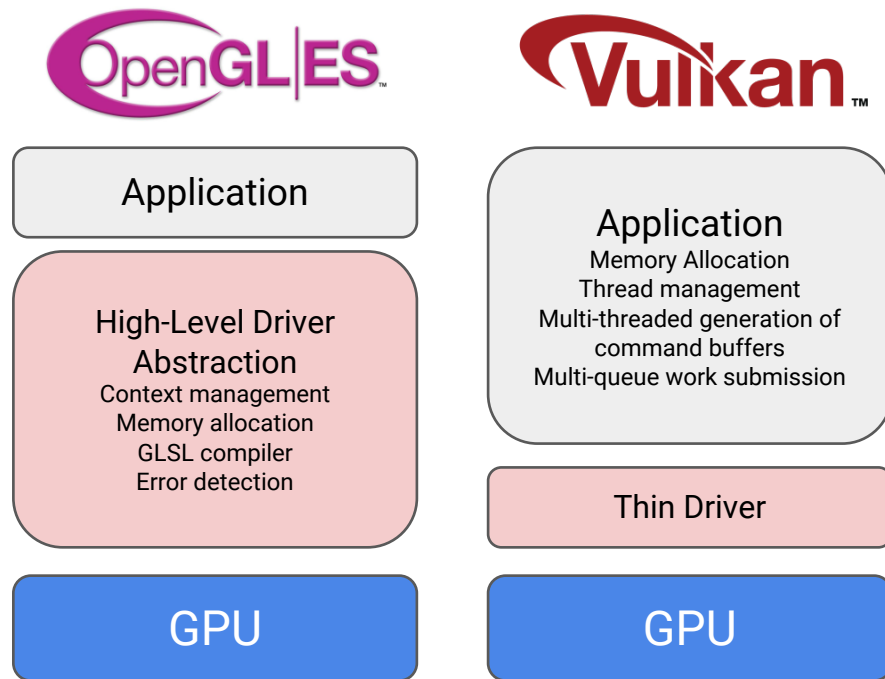
Vulkan, an Explicit API

- Explicit
 - Predictable behavior
 - Predictable performance

**PROGRAMMER
ADVISORY
EXPLICIT CONTENT**

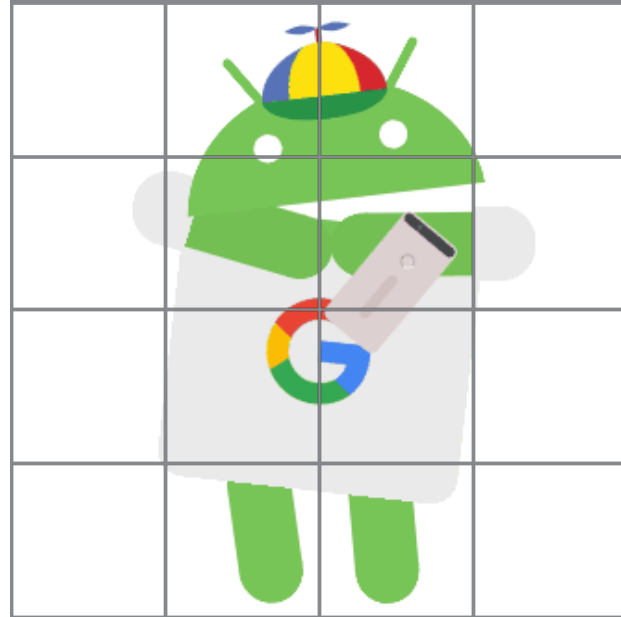


Vulkan Explicit GPU Control



Vulkan, a Modern API

- Modern
 - **Designed for Desktop and Mobile**
 - Data-oriented
 - Multithreading
 - Shader IR (SPIR-V)



Vulkan, a Modern API

- Modern
 - Designed for Desktop and Mobile
 - **Data-oriented**
 - Multithreading
 - Graphics, Compute, DMA Queues
 - Shader IR (SPIR-V)

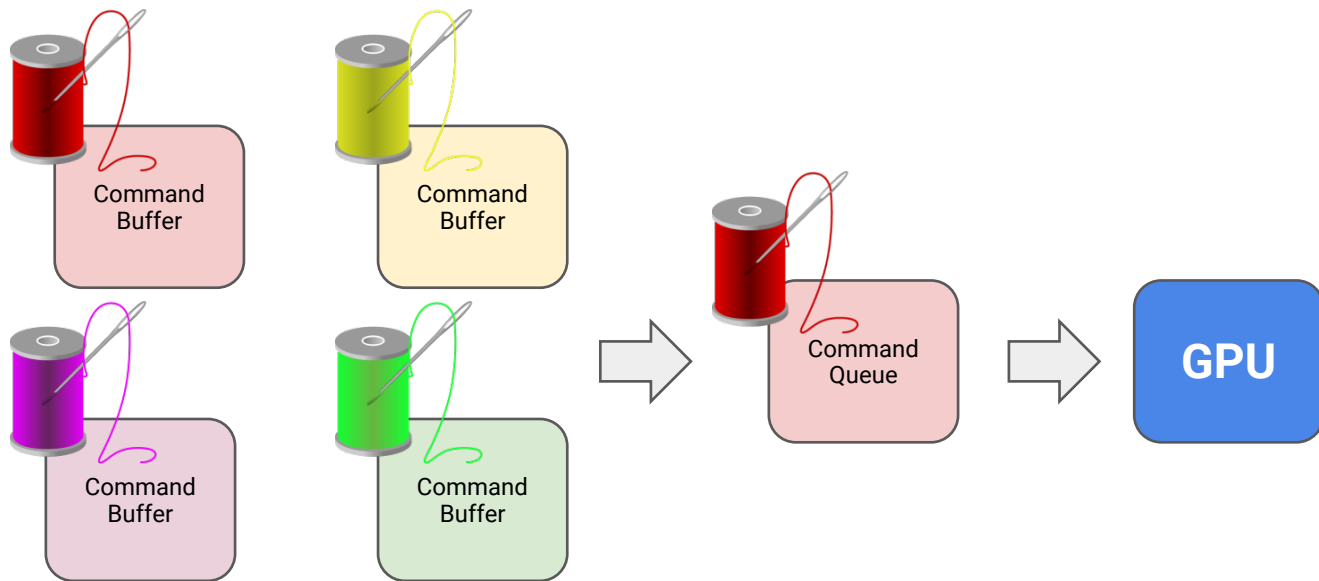


Vulkan, a Modern API

- Modern
 - Designed for Desktop and Mobile
 - Data-oriented
 - **Multithreading**
 - Graphics, Compute, DMA Queues
 - Shader IR (SPIR-V)



Vulkan Multithreading



Vulkan, a Modern API

- Modern
 - Designed for Desktop and Mobile
 - Data-oriented
 - Multithreading
 - **Graphics, Compute, DMA Queues**
 - **Shader IR (SPIR-V)**

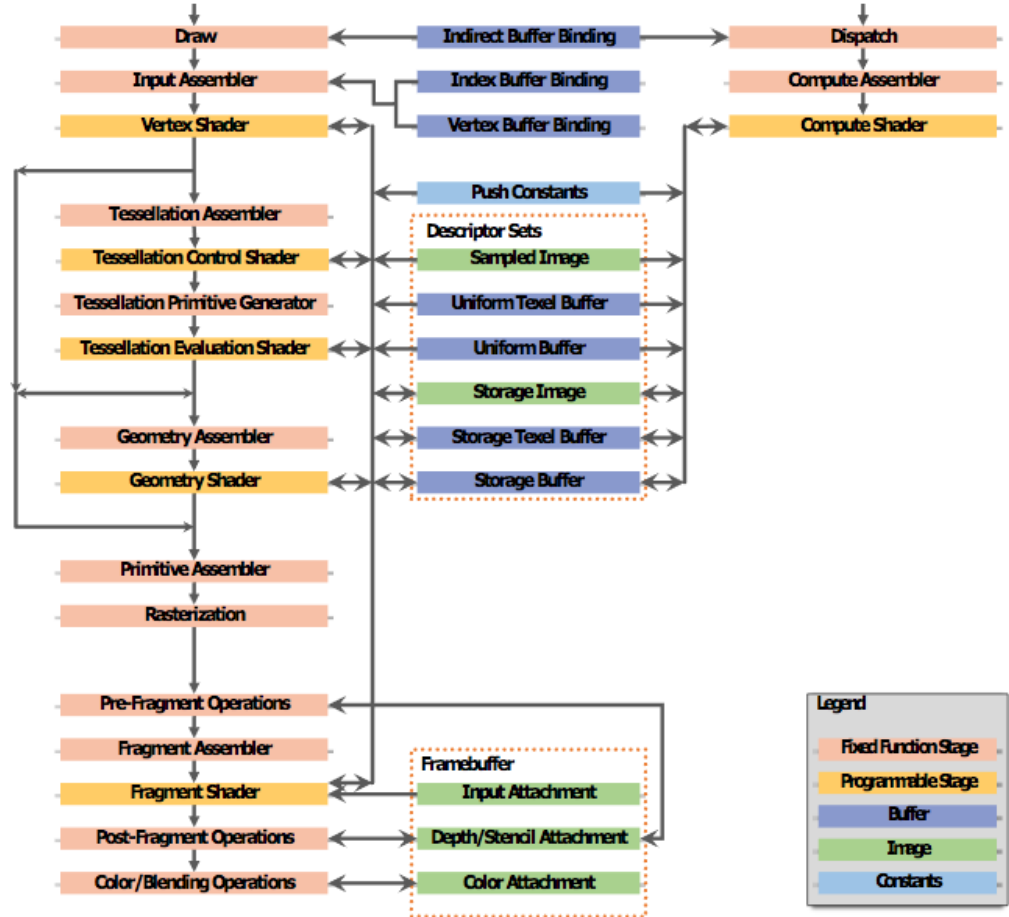


GLSL and **HLSL** are both supported by
ShaderC!



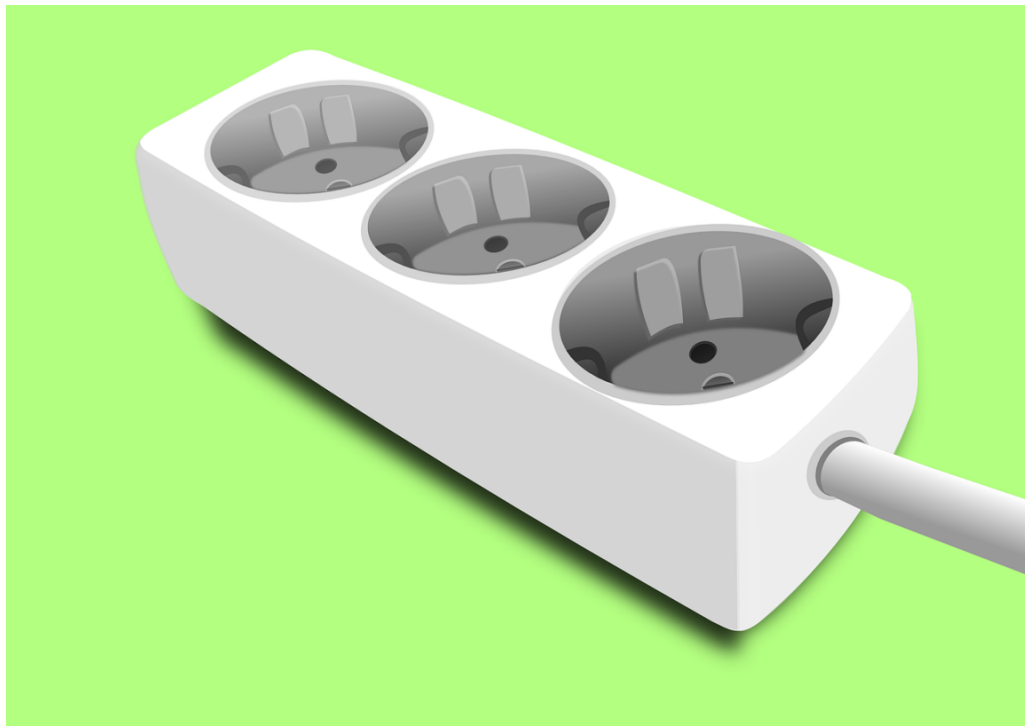
A Modern Pipeline

- Programmers are in the center of the pipeline
 - Design shaders
 - Configure fixed function units
 - Feed images/textures

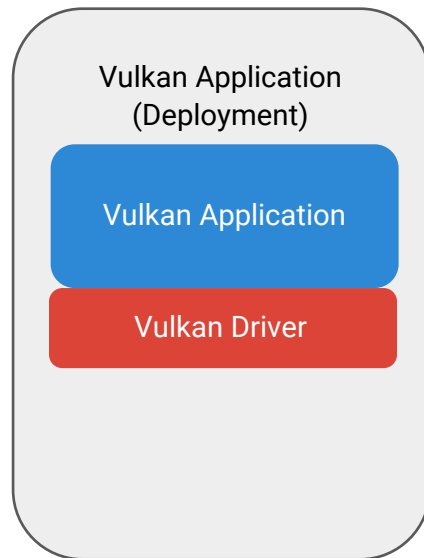
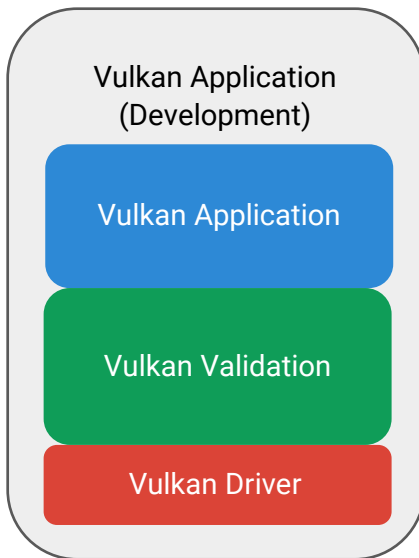
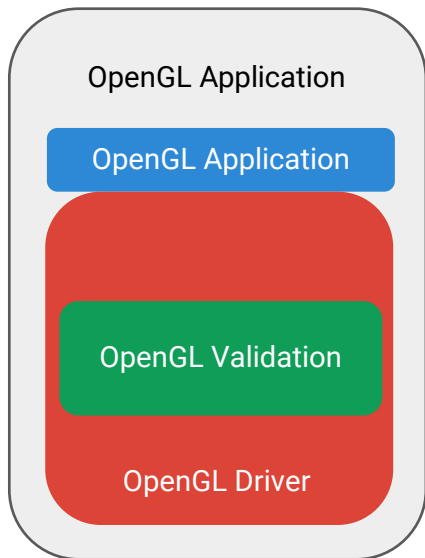


Vulkan, an Extensible API

- Extensible
 - **Layers and extensions**
 - Common loader interface



OpenGL and Vulkan



Vulkan, an Extensible API

- Extensible
 - Layers and extensions
 - **Common loader interface**



Vulkan, an Open Standard

- Open Source
 - Specification, Tests
 - Tools
 - Validation Layers
- Broad OS Support
 - Android
 - Windows 7,8, 10
 - Linux





Getting Started with Vulkan on Android



Some Vulkan Devices



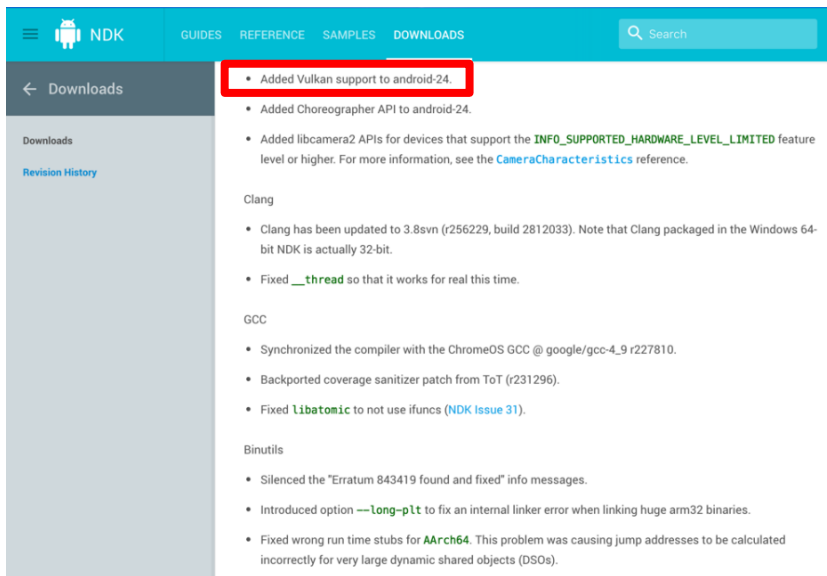
Android Nougat



d.android.com/about/versions/nougat

NDK Headers in R12

d.android.com/ndk/downloads



The screenshot shows the NDK Downloads page. The top navigation bar is blue with the NDK logo and links for GUIDES, REFERENCE, SAMPLES, and DOWNLOADS. A search bar is on the right. The left sidebar has a 'Downloads' link and a 'Revision History' link. The main content area lists updates, with the first item, 'Added Vulkan support to android-24', highlighted by a red rectangle. Other updates include 'Added Choreographer API to android-24', 'Added libcamera2 APIs for devices that support the INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED feature level or higher', 'Clang' updates (Clang 3.8svn, fixed __thread), 'GCC' updates (synchronized with ChromeOS GCC, backported sanitizer patch, fixed libatomic), and 'Binutils' updates (silenced Erratum 843419, introduced --long-plt option, fixed wrong run time stubs for AArch64).

NDK

GUIDES REFERENCE SAMPLES DOWNLOADS

Search

← Downloads

Downloads

[Revision History](#)

- Added Vulkan support to android-24.
- Added Choreographer API to android-24.
- Added libcamera2 APIs for devices that support the `INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED` feature level or higher. For more information, see the [CameraCharacteristics](#) reference.

Clang

- Clang has been updated to 3.8svn (r256229, build 2812033). Note that Clang packaged in the Windows 64-bit NDK is actually 32-bit.
- Fixed `__thread` so that it works for real this time.

GCC

- Synchronized the compiler with the ChromeOS GCC @ google/gcc-4.9 r227810.
- Backported coverage sanitizer patch from ToT (r231296).
- Fixed `libatomic` to not use ifuncs ([NDK Issue 31](#)).

Binutils

- Silenced the "Erratum 843419 found and fixed" info messages.
- Introduced option `--long-plt` to fix an internal linker error when linking huge arm32 binaries.
- Fixed wrong run time stubs for `AArch64`. This problem was causing jump addresses to be calculated incorrectly for very large dynamic shared objects (DSOs).



Vulkan Samples

github.com/googlesamples/vulkan-basic-samples.git

```
# Clone the repository
git clone https://github.com/googlesamples/vulkan-basic-samples.git

# Navigate to the LunarGSamples directory

# Update the glslang source
$ ./update_external_sources.sh -s -g

# Build the sample Android Studio projects
cd API-samples
$ cmake -DANDROID=ON -DANDROID_ABI=[armeabi-v7a|arm64-v8a|
x86|x86_64|all(default)]
```

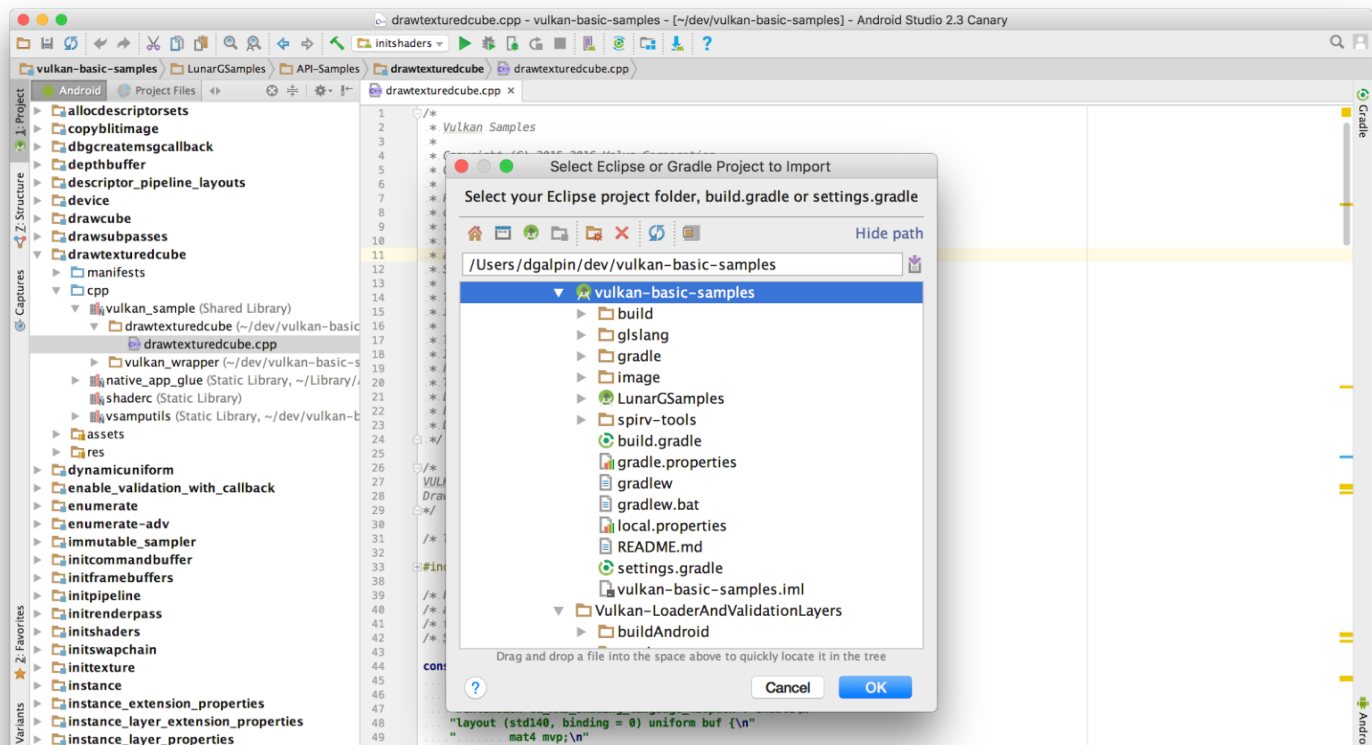
SH



Google Developer Day

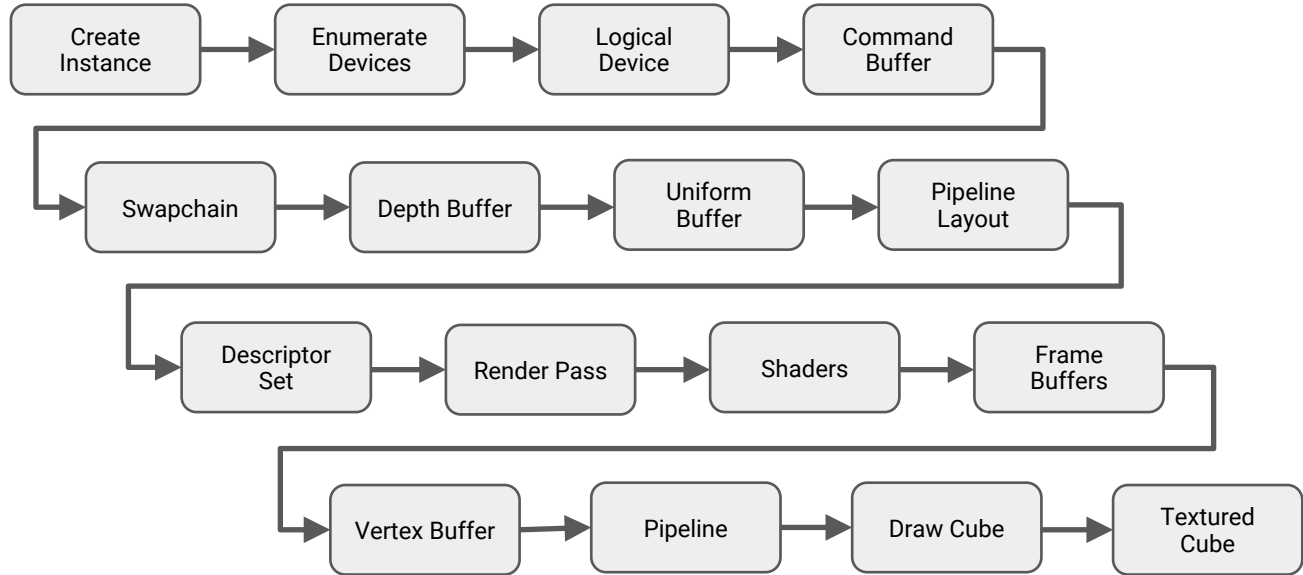
Vulkan Samples in Android Studio

Open project at root of repo



Vulkan Samples in Android Studio

Sample Progression Tutorial (LunarXchange)





Tips for Developing in Vulkan

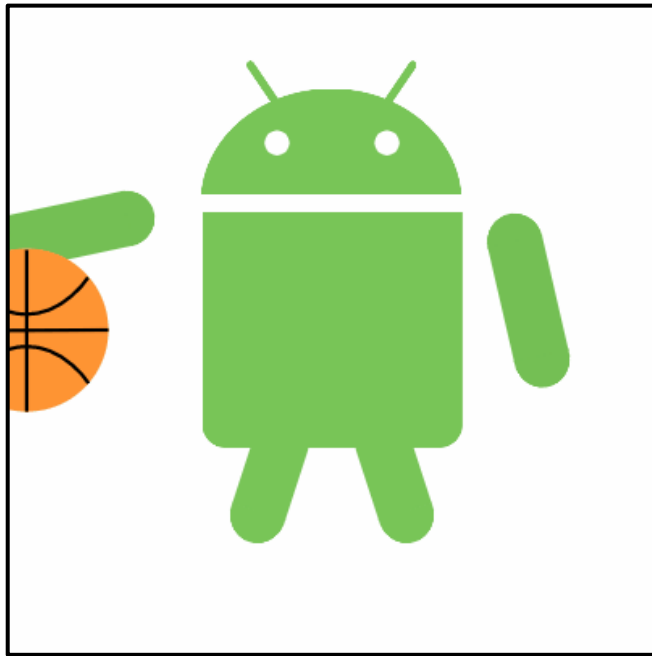


#1: Use Validation

Vulkan Validation Layers

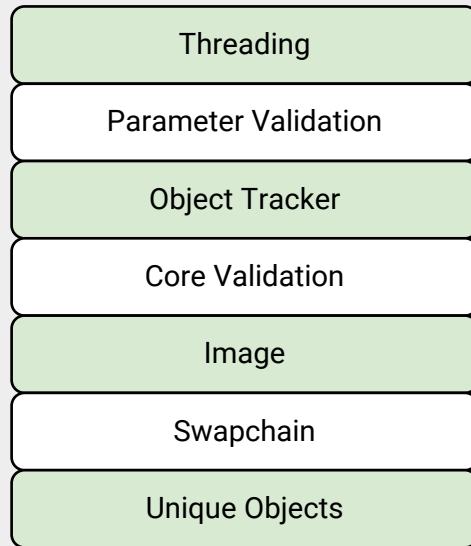
Your Defense Against Fragmentation

- Detects and reports incorrect API usage
- Capable of much more, including performance analysis
- Every early Vulkan developer tells us that using them is critical to their success



Vulkan Standard Validation Layers

```
const char*instance_layers[] = {  
    "VK_LAYER_LUNARG_standard_validation"  
};
```



Vulkan Standard Validation Layers

LunarGSamples/layers

CMake

```
add_vk_layer(threading threading.cpp thread_check.h vk_layer_table.cpp)
add_vk_layer(parameter_validation parameter_validation.cpp parameter_validation.h
vk_layer_table.cpp)
add_vk_layer(object_tracker object_tracker.cpp vk_layer_table.cpp)
add_vk_layer(core_validation core_validation.cpp vk_layer_table.cpp)
add_vk_layer(image image.cpp vk_layer_table.cpp)
add_vk_layer(swapchain swapchain.cpp vk_layer_table.cpp)
add_vk_layer(unique_objects unique_objects.cpp vk_layer_table.cpp vk_safe_struct.cpp)
```

Threading

Parameter Validation

Object Tracker

Core Validation

Image

Swapchain

Unique Objects



Google Developer Day

Vulkan Standard Validation Layers

LunarGSamples/layers

C++

```
const char *instance_layers[] = {  
    "VK_LAYER_GOOGLE_threading",  
    "VK_LAYER_LUNARG_parameter_validation",  
    "VK_LAYER_LUNARG_object_tracker",  
    "VK_LAYER_LUNARG_core_validation",  
    "VK_LAYER_LUNARG_image",  
    "VK_LAYER_LUNARG_swapchain",  
    "VK_LAYER_GOOGLE_unique_objects"  
};
```

Threading

Parameter Validation

Object Tracker

Core Validation

Image

Swapchain

Unique Objects



Enabling Instance Validation

```
const char *instance_layers[] = {  
    "VK_LAYER_GOOGLE_threading",  
    "VK_LAYER_LUNARG_parameter_validation",  
    "VK_LAYER_LUNARG_object_tracker",  
    "VK_LAYER_LUNARG_core_validation",  
    "VK_LAYER_LUNARG_image",  
    "VK_LAYER_LUNARG_swapchain",  
    "VK_LAYER_GOOGLE_unique_objects"  
};  
uint32_t instance_layer_request_count = sizeof(instance_layers) / sizeof(instance_layers[0]);  
// Pass desired instance layers into vkCreateInstance  
VkInstanceCreateInfo instance_info = {};  
instance_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;  
instance_info.enabledLayerCount = instance_layer_request_count;  
instance_info.ppEnabledLayerNames = instance_layers;
```

C++



Google Developer Day

Enabling Instance Validation

C++

```
static VKAPI_ATTR VkBool32 VKAPI_CALL DebugReportCallback(  
    VkDebugReportFlagsEXT msgFlags,  
    VkDebugReportObjectTypeEXT objType,  
    uint64_t srcObject, size_t location,  
    int32_t msgCode, const char * pLayerPrefix,  
    const char * pMsg, void * pUserData )  
{  
    if (msgFlags & VK_DEBUG_REPORT_ERROR_BIT_EXT) {  
        __android_log_print(ANDROID_LOG_ERROR,  
            "AppName",  
            "ERROR: [%s] Code %i : %s",  
            pLayerPrefix, msgCode, pMsg);  
    } else if (msgFlags & VK_DEBUG_REPORT_WARNING_BIT_EXT) {  
        ...  
    }  
}
```



Vulkan Validation Layers

github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/tree/master/layers


GitHub

[Explore](#) [Features](#) [Enterprise](#) [Pricing](#) [Sign up](#) [Sign in](#)









[KhronosGroup](#) / [Vulkan-LoaderAndValidationLayers](#) Watch 66 Star 138 Fork 35

[Code](#) [Issues 39](#) [Pull requests 3](#) [Pulse](#) [Graphs](#)

Branch: **master** [Vulkan-LoaderAndValidationLayers / layers /](#) [New file](#) [Find file](#) [History](#)

 **dustin-lunarg** layers: Add pNext parameter checking ...

Latest commit 2949a5a 5 days ago

..		
 linux	Bump api_version to 1.0.5 in json files	5 days ago
 windows	Bump api_version to 1.0.5 in json files	5 days ago
 .clang-format	layers: clang-format layers directory	5 days ago
 CMakeLists.txt	misc: Remove lunarg_debug_marker extension	11 days ago
 README.md	layers: GH LVL#49, Updated layer docs/README for new layers	14 days ago
 device_limits.cpp	layers: clang-format layers directory	5 days ago
 device_limits.h	layers: clang-format layers directory	5 days ago
 draw_state.cpp	layers: Fix overlapping attachment aliasing dependency check in draw_...	3 days ago



Google Developer Day

Building the Validation Layers

```
# First clone the repository from GitHub (not shown)  
# Then get the glslang peer directory
```

```
./update_external_sources.sh -g
```

```
# Generate the Android makefiles
```

```
cd build—android  
sh android-generate.sh  
ndk-build
```

SH



Google Developer Day

Checking for Validation Layers (Instance)

C++

```
// Get instance layer count using null pointer as last parameter
uint32_t instance_layer_present_count = 0;
vkEnumerateInstanceLayerProperties(&instance_layer_present_count, nullptr);

// Enumerate instance layers with valid pointer in last parameter
VkLayerProperties* layer_props = malloc(instance_layer_present_count *
sizeof(VkLayerProperties));
vkEnumerateInstanceLayerProperties(instance_layer_present_count, layer_props);
```



Beyond Validation: Testing

Leverage the Desktop

- AMD, Intel, and NVIDIA all support Vulkan
 - Test on different GPU architectures
 - Some GPUs/drivers don't support features you might expect
- Mobile and Desktop code 99% the same
 - Window management will take different paths

Bleeding Edge

- Be prepared for driver issues
 - That can be worked around





#2: Distributing Vulkan

Dynamically Bind to Vulkan

Consider using `vulkan_wrapper.cpp` from samples

C++

```
void* libvulkan = dlopen("libvulkan.so", RTLD_NOW | RTLD_LOCAL);  
if (!libvulkan)  
    return 0;
```

```
// Vulkan supported, set function addresses
```

```
vkCreateInstance = reinterpret_cast<PFN_vkCreateInstance>(dlsym(libvulkan, "vkCreateInstance"));  
vkDestroyInstance = reinterpret_cast<PFN_vkDestroyInstance>(dlsym(libvulkan, "vkDestroyInstance"));  
vkEnumeratePhysicalDevices = reinterpret_cast<PFN_vkEnumeratePhysicalDevices>(dlsym(libvulkan,  
"vkEnumeratePhysicalDevices"));
```

```
...
```



Google Developer Day

Vulkan and MultiAPK

XML

```
<uses-feature  
    android:name="android.hardware.vulkan.version"  
    android:required="true"/>
```



Google Developer Day

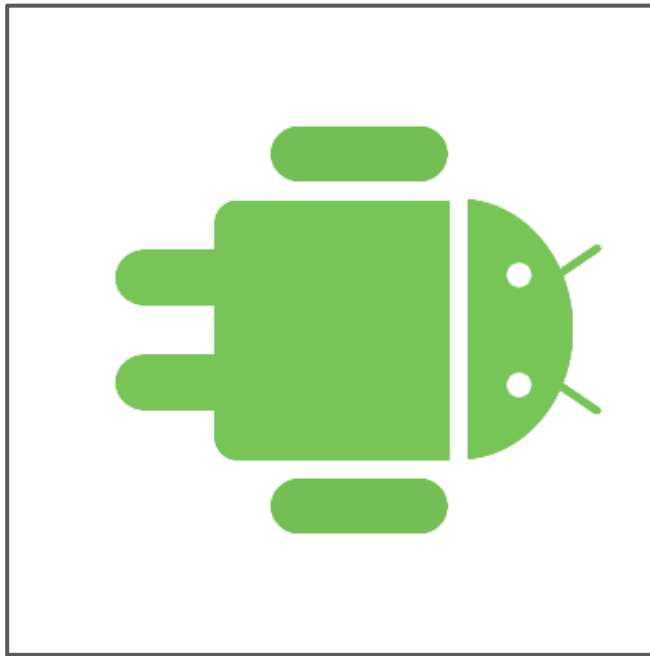


#3: Display Rotation

Apply Display Rotation During Rendering

Rotation Isn't Free on Android

- Android has a natural display orientation
- Letting the compositor rotate uses extra power/bandwidth
- Rotating while rendering the swapchain image is (essentially) free



Apply Display Rotation During Rendering

Rotation Isn't Free on Android

- `VkSurfaceCapabilitiesKHR::currentTransform` shows compositor rotation
- `VkSwapchainCreateInfoKHR::preTransform` is used to indicate to Vulkan that the app handles its own rotation



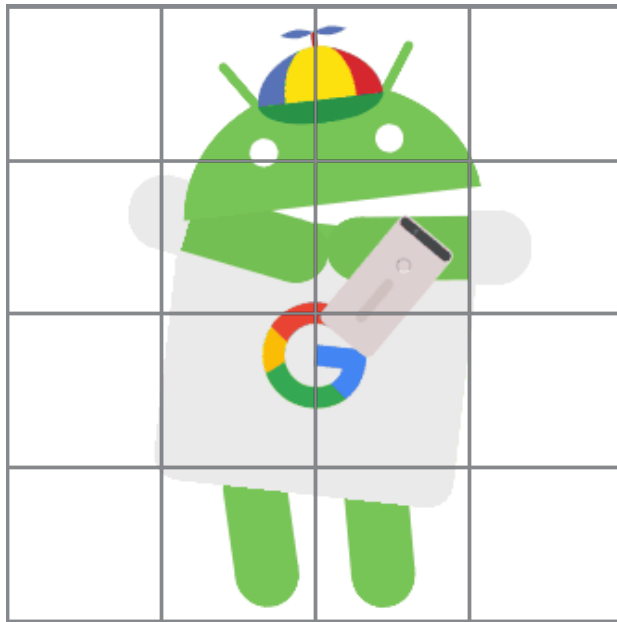
A large, stylized blue geometric shape, resembling a thick chevron or a stylized letter 'V', is positioned on the left side of the slide. It is composed of two overlapping parallelogram-like shapes, one in a darker blue and one in a lighter blue, creating a 3D effect.

#4: Minimize Render Passes

Tile-based GPU Architecture

Most mobile GPUs are tile-based

- Render target split into separately rendered “tiles”
- Resources needed to render a single tile stored in on-chip memory
- Bandwidth for reading off-chip is limited



Immediate Renderers

You draw...

Triangle 1



then

Triangle 2



then

Triangle 3



GPU Renders



Triangle 1



Triangle 2



Triangle 3



Tiling Renderers

You draw...

Triangle 1



then

Triangle 2



then

Triangle 3



Google Developer Day

GPU Bins

Tri 1
Tri 2

Bin A

Tri 2

Bin C

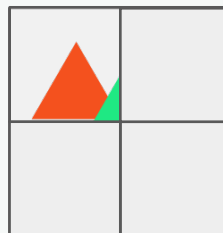
Tri 2
Tri 3

Bin B

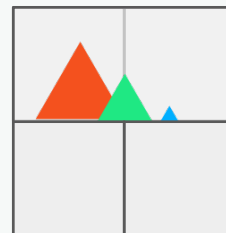
Tri 2
Tri 3

Bin D

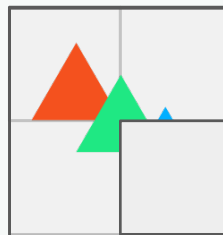
GPU Renders



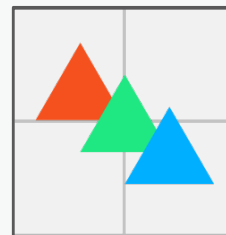
Tile A



Tile B



Tile C



Tile D

Multiple-Pass Rendering Techniques

Single-pixel Source Post-Process

Tone mapping

Exposure control

HDR

Multi-pixel Source Post-Process

Blur

Bloom

Focus control



Multiple-Pass Rendering Techniques

Deferred Shading

Moving material data on-and-off chip in buffers

Bandwidth & memory pressure can be painful

Volumetric Effects

Participating media & accumulation

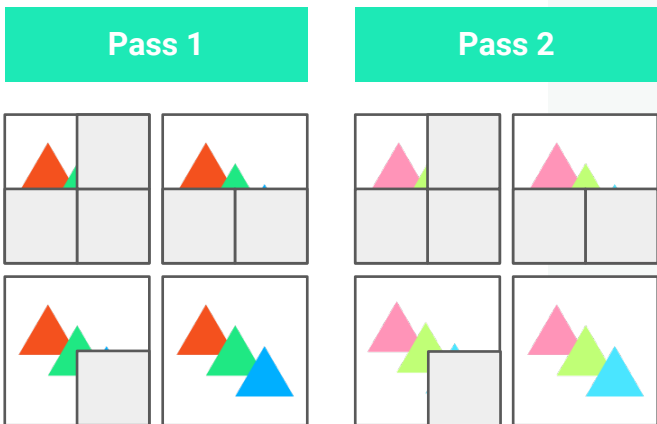
Soft particles, depth-sample techniques



Multiple Passes on a Tiling Renderer

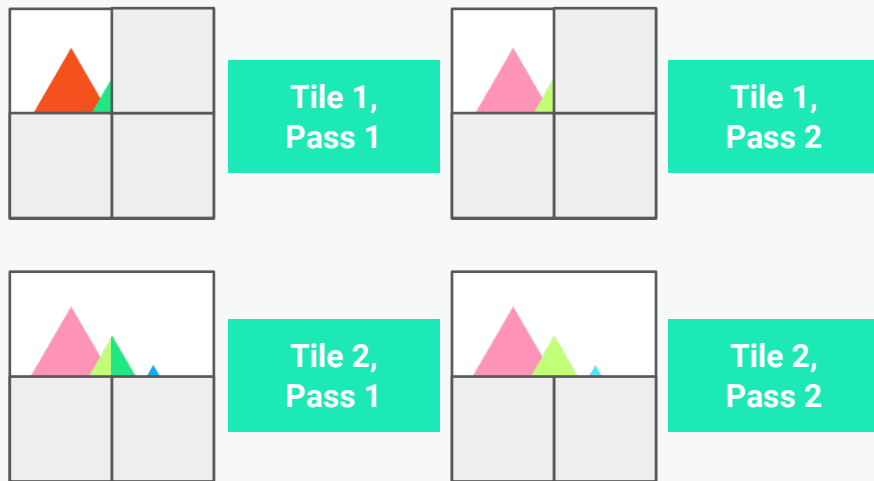
Without Vulkan Multipass

For each pass: render each tile and commit to memory. Later passes must reload. MSAA multiplies off-chip cost.



With Vulkan Multipass

For each tile: render each subpass. Relevant data stays on-chip. Resolve MSAA after last-pass, store once.



Tips for using subpasses

Attachments

Set load/store disposition correctly

Load: `VK_ATTACHMENT_LOAD_OP_DONT_CARE` if you're overwriting entire buffer,
`VK_ATTACHMENT_LOAD_OP_CLEAR` if you need it cleared

Store: `VK_ATTACHMENT_STORE_OP_DONT_CARE` if you're not using the contents again

Use `VK_ATTACHMENT_LOAD_OP_LOAD` and `VK_ATTACHMENT_STORE_OP_STORE` only
when you actually need to load or store!

Use transient attachments for never-stored data



Tips for using subpasses

Subpass dependencies

Specify **by-region**— otherwise it's a global dependency, and it waits on *all pixels* in prior pass

Single-pixel sources only (for now)

Each subpass can only access data for the same pixel in prior passes



Benefits

Efficiency, efficiency, efficiency

Tiling renderers save precious clock cycles and power by avoiding extraneous read/write in low-bandwidth situation

Opens the door for rendering techniques previously costly on mobile

Immediate renderers can still benefit

E.g.: transient attachments never have to live anywhere outside of cache memory



A large, stylized blue geometric shape, resembling a thick chevron or a folded ribbon, is positioned on the left side of the slide. It consists of two main parts: a darker blue upper section and a lighter blue lower section, both with sharp, angular edges.

#5: Memory Management

Choose Appropriate Memory

Applications Choose the Memory Type

- Different devices will have different memory types available
 - Memory types are ordered so applications can use a simple algorithm to pick the best type for each use
 - Mobile systems generally don't have separate physical memory heaps for CPU and GPU
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` is not as significant as it is on systems without shared memory.



Suballocate Device Memory

vkAllocMemory is Expensive

- Don't individually allocate small objects
- Alloc large blocks of memory, and then use a suballocator
- None Included - Expect this to be a common utility class



Effective Memory Management Strategies

Object Lifecycle tied to Rendering State

- Block allocate per-frame objects with a suballocator
 - Toss block into free heap after frames have rendered
- Use pools to avoid allocation during gameplay

vkDescriptorPool

- Not consistently implemented by drivers
 - Consider rolling your own



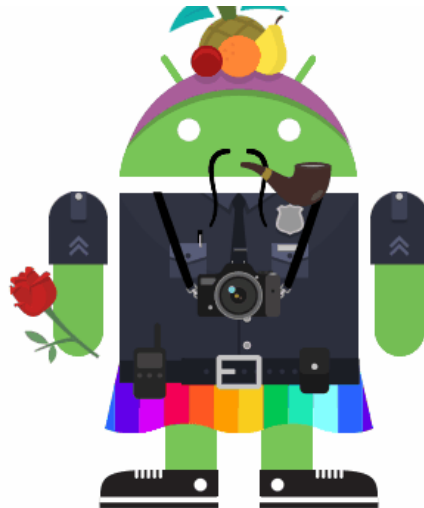


#6: Batch API Calls

Batch API Calls

Vulkan does Batches!

- Perform multiple operations with a single call!
- Examples include `vkQueueSubmit`, `vkAllocateCommandBuffers`, `vkUpdateDescriptorSets`, `vkCmdBindDescriptorSets`





#7: Use Multithreading

Multithread!

Many Operations can be Parallelized

Examples Include:

- Loading textures and shaders

- Building pipelines

- Building command buffers

Avoid Per-Thread Resources

- Leads to Memory Bloat

- Make sure to avoid cross-talk

Threads are Heavy-Weight

- Create a single-threaded path for debugging and for performance testing

- Bad multithreading is worse than no multithreading





#8: Synchronization

Synchronization

With Great Power Comes Great Responsibility

Vulkan Includes 5 Primitives

Fence

Semaphore

Event

Memory barrier

Pipeline barrier



Google Developer Day

Rendering to a Texture

C++

```
const VkImageMemoryBarrier barrier = {
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER, nullptr,
    ... // lots of parameters here in usual Vulkan fashion
    image, { // VkImage image
        VK_IMAGE_ASPECT_COLOR_BIT, // .aspectMask
        0, 1, // .baseMipLevel, .levelCount
        0, 1, // .baseArrayLayer, .layerCount
    },
};

vkCmdPipelineBarrier(cmdbuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // VkDependencyFlags
    0, nullptr, // memory barriers
    0, nullptr, // buffer memory barriers
    1, &barrier); // image memory barriers
```



Google Developer Day



#9: Pipelines

Pipelines

Are Expensive to Create

- Pre-create if at all possible
 - Consider using dynamic analysis and tracing
 - Easiest way to drop frames and stutter



Pipelines and SPIRV-Cross

Enables Shader Reflection

- At Runtime
 - Determine Shader Resources
 - Query Push Constants

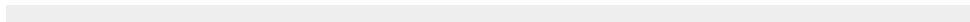
Enables GL Support for SPIRV Shaders

- Supports decompilation
- Emulation of Push Constants with uniforms





#10: Debugging



Capture/Replay Tools

VulkanTools

- Uses validation layers
 - vktrace
- Replay back on host
 - vkreplay

RenderDoc

- Windows Only GUI
- Beloved Game development tool
- Requires modified APK

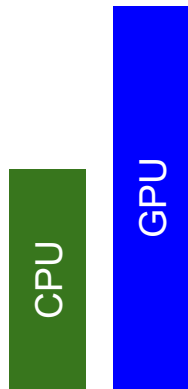




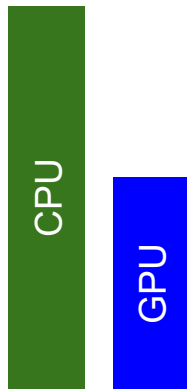
Should My Application Use Vulkan?

Vulkan Considerations

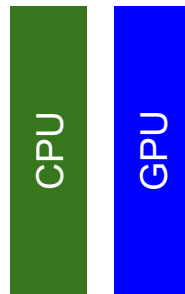
GPU Bound Rendering



CPU Bound Rendering



Thermally Bound Rendering



Vulkan in Game Engines



4.14 Released



In Preview



Google Developer Day

In Conclusion

You Can Start Now

Vulkan is here and coming to more devices soon!

developer.android.com/ndk/downloads

github.com/LunarG/VulkanSamples

github.com/KhronosGroup/SPIRV-Cross



Google Developer Day

Google

Thank you!