The Vulkan SDK

From the Vulkan API Launch to Today

Karen Ghavam
CEO and Engineering Director
LunarG, Inc
Today’s Talk

- Creation of the Vulkan SDK
- Vulkan API and the Vulkan Developer Tools
- Creation of Present Day LunarG
Important Context
Who is LunarG
Who is Khronos
What is Vulkan
Why Vulkan?
9 Years Ago

The Existential Event

Important Context
- Who is LunarG
- Who is Khronos
- What is Vulkan
- Why Vulkan?
9 Years Ago

The Existential Event

The Vulkan Developer Tools
The Vulkan SDK

Important Context
Who is LunarG
Who is Khronos
What is Vulkan
Why Vulkan?
Who is LunarG?

- Independent, privately owned software consultancy
- Passionate about 3D graphics & compute technology
- Industry leading, 3D-graphics software experts with decades of experience
  - Vulkan, OpenXR, OpenGL, Direct3D, Metal, …
  - Developer tools, drivers, performance tuning…
- Developers of proprietary and open source drivers, tools, & software solutions
- Founded in 2009 – Headquarters in Fort Collins, CO
- Delivers the Vulkan SDK
What is Vulkan?

- Cross-platform, Cross-vendor Graphics and Compute API
  - PCs, consoles, mobile phones, embedded platforms
- Vulkan API Specification Created by the Khronos Group
  - Member driven consortium for the creation and maintenance of open standards
    - GPU vendors
    - Platform vendors
    - SoC Integrators
    - Tool developers
    - Game Engine developers
    - ...
- Low level and explicit API
  - Specification in essence defines a GPU
A Brief History of Vulkan

SIGGRAPH in Vancouver
- Khronos call for participation in defining the "glNext" API
  - OpenGL, Direct3D were mature with minor feature updates
  - A need to scrape away the abstractions included in OpenGL and Direct3D
  - Mantle, Direct3D 12, Metal all demonstrated the needs of the future
- Features
  - High-efficiency access to graphics and compute on modern GPUs
  - Abstraction removal – explicit GPU and CPU control over workloads
  - Multithreading-friendly API with reduced overhead
  - Common shader programming intermediate language (SPIR-V)
A Brief History of Vulkan

August 2014

First Vulkan Proof of Concept
- Vulkan ILO Driver (Linux, Intel GPU)
- Valve Source2 Engine
- Key feedback for the Vulkan 1.0 Specification

March 2015

February 2016
A Brief History of Vulkan

August 2014

March 2015

GDC

- Technical Previews
- Valve Source2 Engine
- Vulkan ILO Driver

February 2016
A Brief History of Vulkan

August 2014

March 2015

February 2016

Public Launch
Why Vulkan?
Why Vulkan? Cross-platform support

Same API for Mobile, desktop, (and Apple platforms)
Why Vulkan? Improved Cross-vendor Compatibility

One API usage validator used by all (Vulkan-ValidationLayer)
Why Vulkan? Improved Performance

• Explicit application control over GPU and CPU workloads
• Multithreading-friendly API
• No more error checking in the Vulkan driver
Why Vulkan? Shader Language Flexibility

Standardized Intermediate Language (SPIR-V)

- Eliminates front-end compilers from drivers
  - Reduce driver complexity
- Front-end language flexibility
  - Improve portability

![Diagram showing various shader languages and SPIR-V](image-url)
Why Vulkan? Open Standard

Strengthened ecosystem focus

- Embrace and engage with the ISVs
- Open conformance test suite - more rigor
- More control put in the developer's hands
Vulkan is a Layered Architecture

- **Vulkan Loader**
  - Library that finds and loads Drivers & Layers

- **Vulkan Layer**
  - “Plugin interface to the Vulkan API”
  - Intercepts Vulkan API calls made by applications
  - Enables mechanism for valuable cross-vendor debugging tools

API calls work their way through the loader, layers, and driver in order
OpenGL vs. Vulkan

Application
Single thread per context

Graphics Driver
Context management
Memory allocation
Full GLSL compiler
Error Detection

Graphics Driver
Explicit GPU Control

GPU

Application
Memory allocation
Thread management
Explicit Synchronization
Multi-threaded
Generation of command buffers

Loadable Layers
(e.g. Validation Layer)

Multiple Front-end Compilers
GLSL, HLSL, etc.

GPU
The Forming of Present-day LunarG

9 Years Ago

The Existential Event

Important Context
Who is LunarG
Who is Khronos
What is Vulkan
Why Vulkan?
When did "The Vulkan Journey" start for Karen?

  - "Off the Grid"
  - Upon return, it all started…
- "Do you want to coast to retirement, or go out with a bang?"
- October 12, 2015 – First day at LunarG
The Existential Event

- Turmoil 3 months before the public launch!
The Existential Event

- Turmoil 3 months before the public launch!
The Existential Event

- Turmoil 3 months before the public launch!
The Existential Event

Engineering Team cut in half!!
9 Years Ago

**The Existential Event**

**The Vulkan Developer Tools**
- The Vulkan SDK

**Important Context**
- Who is LunarG
- Who is Khronos
- What is Vulkan
- Why Vulkan?

**Developer Tools and the Vulkan SDK**
Why Developer Tools?

- While the Vulkan API specification is absolutely necessary…
  - It isn't sufficient for the success of the Vulkan API

- Application developers need debugging tools!
Open-source Vulkan Developer Tools

Included in the Vulkan SDK

- Vulkan Loader
- vkconfig
- Validation Layer
- vulkaninfo
- Emulation Layers
- SPIR-V Optimizer
- SPIR-V Tools
- Crash Diagnostic Layer
- GPUInfo
- VOLK
- shaderc
- SPIR-V Validator
- Profiles Toolset
- GFX Reconstruct
- VKVIA
- DXC
- SPIR-V Reflect
- apidump
- GFX Reconstruct
- VKVIA
- SPIR-V Cross
- glslang
- Vulkan-HPP
- Screenshot
- VMA
- MoltenVK
- SPIR-V Visualizer
- SDL
- Monitor
- GLM
Open-source Vulkan Developer Tools

Included in the Vulkan SDK

Vulkan Desktop Loader

- Used by ALL on ALL desktop platforms
- Consistent method of finding Vulkan Drivers
- Allows multiple Vulkan drivers to exist without interference
- Provides the plug-in mechanism to enable Vulkan Layers

→ Having a single loader results in consistent runtime behaviors across platforms
Open-source Vulkan Developer Tools

Included in the Vulkan SDK

- **Vulkan Validation Layer**
  - Most valuable and critical debugging layer
  - Validate correct Vulkan API usage (drivers don’t do it anymore)
  - Once the application is error free, disable it and run at full speed!
  - One validation implementation → More consistency across vendors

- Vulkan Loader
- vkconfig
- Validation Layer
- Vulkan Validation Layer
- SPIR-V Cross
- glslang
- Vulkan-HP
- Screenshot
- VMA
- MoltenVK
- SPIR-V Visualizer
- SDL
- Monitor
- GLM
Open-source Vulkan Developer Tools

Included in the Vulkan SDK

Windows

MoltenVK
- Bringing Vulkan to Apple

 SPIR-V Optimizer
 SPIR-V Tools
 shadec
 SPIR-V Validator
 SPIR-V
 vulkdump
 Vulkan Loader

Validation Layer

vkconfig

Crash Diagnostic Layer

vulkaninfo

GPUInfo

GFX Reconstruct

Vulkan-HPP

Screenshot

VMA

SDK

Emulation Layers

VOLK

VolkVIA

Monitor

GLM
Open-source Vulkan Developer Tools
Included in the Vulkan SDK

- GFXReconstruct - API Capture and Replay
  - Cross-platform (Windows, Linux, Android, macOS)
  - Run Vulkan workloads during GPU development
  - Debug Vulkan applications
  - Regression testing using real application workloads
  - Underlying engine for profiling and debugging tools

- DXC
- SPIR-V Reflect
- apidump
- GFX Reconstruct
- VKVIA
- glslang
- Vulkan-HPP
- Screenshot
- VMA
- SPIR-V Cross
- MoltenVK
- SPIR-V Visualizer
- SDL
- Monitor
- GLM

Open-source Vulkan Developer Tools are included in the Vulkan SDK. They include tools such as GFXReconstruct for API capture and replay, which is cross-platform and can be used for running Vulkan workloads, debugging applications, regression testing, and as the underlying engine for profiling and debugging tools. The tools are integrated into the Vulkan SDK and can be used on various platforms including Windows, Linux, Android, and macOS.
Open-source Vulkan Developer Tools
Included in the Vulkan SDK

- Vulkan Loader
- SPV Tools
- Vulkan
- SPIR-V
- SPIR-V
- Optimizer
- Validation
- Layer
- vulkaninfo
- Emulation
- Layers
- MoltenVK
- SPIR-V
- Visualizer
- SDL
- Monitor
- GLM

Crash Diagnostic Layer
- Track down and identify the cause of GPU hangs and crashes
- Instruments command buffers with completion checkpoints
- Get a dump file
- Strong user demand. Debugging Device Lost errors very difficult!
The Vulkan SDK

Delivered by LunarG in close coordination with the Khronos Vulkan working group

- **Benefits**
  - Pre-built
  - Curated
  - Integrated
  - System Installation
  - vkconfig ready for use
  - License Registry

Delivered by LunarG in close coordination with the Khronos Vulkan working group
The Vulkan SDK

Delivered by LunarG in close coordination with the Khronos Vulkan working group

- Vulkan Loader
- vkconfig
- Validation Layer
- Vulkaninfo
- Emulation Layers
- SPIR-V Optimizer
- SPIR-V Tools
- Crash Diagnostic Layer
- Vulkaninfo
- SPIR-V Tools
- Profiles Toolset
- GPUInfo
- VOLK
- shadec
- SPIR-V Validator
- apidump
- GFX Reconstruct
- VKVIA
- DXC
- SPIR-V Reflect
- Vulkan-HPP
- Screenshot
- VMA
- SPIR-V Cross
- glslang
- Vulkan-HPP
- Screenshot
- VMA
- MoltenVK
- SPIR-V Visualizer
- SDL
- Monitor
- GLM

NEW!

Delivered by LunarG in close coordination with the Khronos Vulkan working group
The Vulkan SDK

- LunarG Ownership
  - Initial creation
  - Ongoing enhancement and maintenance
- LunarG - contributor
- LunarG - Maintainer
Vulkan SDK Download Page (vulkan.lunarg.com)
Vulkan SDK Downloads are Healthy

Windows SDK

Mac SDK

Linux SDK

~38,000/week

~6000/week

~5800/week

Note: Numbers are for Linux “Tarball” only and don’t include Ubuntu packages also available from LunarG or other linux distros
The Existential Event

The Vulkan Developer Tools
The Vulkan SDK

The Mad Scramble
The Benefactor
Stabilization

9 Years Ago

Important Context
Who is LunarG
Who is Khronos
What is Vulkan
Why Vulkan?
The Mad Scramble

- Half the engineering team
- Deliver an SDK in 3 months
  - Vulkan Loader
  - Vulkan Validation Layer
- Launch the SDK download site
The Mad Scramble

The Good News
• The engineers knew what they were doing

The Bad News
• The challenge in front of us!
The Vulkan API Launch - February 16, 2016

- Coordinated Launch
  - Khronos Vulkan 1.0 API specification
  - First Vulkan SDK
How is this funded?
How is this funded?
How is this funded?
How is this funded?
Why Did We Succeed in the Beginning?

A Meaningful Purpose
- Our work matters, and will have a positive and broad industry impact
Why Did We Succeed in the Beginning?

A Meaningful Purpose
- Our work matters, and will have a positive and broad industry impact
- The Generosity of the Vulkan Ecosystem Benefactor

VALVE
Stabilization

- Strong Vulkan API adoption as a low-level standard
- More companies actively participating in building the ecosystem
  - Enabling benefits for ALL
- Listening to the Vulkan application developers
  - Yearly LunarG Developer Survey
  - Accountability to the developers
Stabilization

- A team of 3D graphics SW experts excited about the vision
  - Talented, skilled, enthusiastic
  - Naturally attracts the right people for the job
Stabilization

● A team of 3D graphics SW experts excited about the vision
  ○ Talented, skilled, enthusiastic
  ○ Naturally attracts the right people for the job

● And the LunarG purpose continues!
The First Vulkan SDK

- An INCOMPLETE Validation Layer implementation
- The first Vulkan Loader implementation
- Windows and Linux only
Validation Layer - Then and Now

June 2018
Validation Layer - Then and Now

June 2018

90K

June 2024

730K
Validation Layer and VUIDs

- **VUID - Valid Usage ID**
  - Assigned to each API usage
  - How that part of the API must be used

- **Validation Layer is validating the VUIDs**
  - “Error Checking”
The Validation Layer - Today

- Healthy open-source project with robust functionality
  - GPU-assisted validation - to support the bindless attributes of the Vulkan API
The Validation Layer - Today

- Healthy open-source project with robust functionality
  - GPU-assisted validation - to support the bindless attributes of the Vulkan API
  - Synchronization Validation
    - 2019 - Hazard detection within a single buffer
    - 2022 - Hazard detection within and between queue submissions and across queues
    - These two versions enable baseline functionality and does not cover all Vulkan extensions. More to do!
The Validation Layer - Today

- CI Test Farm
  - SW testing
    - Mock ICD
  - GPU HW
    - Nvidia
    - AMD
    - Intel
    - Android
  - Windows, Linux, Android, macOS
The Validation Layer

We aren't done yet!
Vulkan API continues to evolve!
9 Years Ago

The Existential Event

The Vulkan Developer Tools
The Vulkan SDK

The Mad Scramble
The Benefactor
Stabilization

Important Context
Who is LunarG
Who is Khronos
What is Vulkan
Why Vulkan?

Technology Challenges
Today and the Future
Validation Layer - Vulkan Synchronization

Semaphores
- Main cross-queue synchronization mechanism

Events and Barriers
- Synchronization of commands submitted to a single queue

Fences
- Synchronize work between the device and the host

Validation Layer Improvement Opportunity:
- High Performance Overhead due to required volume of state tracking
- Ongoing improvement opportunity: Performance tuning
Validation Layer – So Many Vulkan Objects!

- The Sheer Number of Vulkan Objects – complexity
- Different functions and usages
  - Rules for how can they be used
  - Rules for order of creation

→ Complexity in the validation layer
Validation Layer - Descriptor Indexing Validation

- Descriptors invoked from shaders
  - Only used descriptors required to be valid
  - Might only use “10” out of millions

- Initial validation implementation
  - Slowed app from 100+ FPS to a fractional value!
  - All descriptors were being validated, regardless if used!

- Performance Improvement!
  - Using instrumented shaders on the GPU
    - Detect which descriptors are actually used
  - Only validate used descriptors
Validation Layer – GPU-AV Performance

- GPU-AV requires instrumenting shaders
- Shaders become bloated; impacting performance
  - Pipeline compile times
  - Runtime shader execution
Validation Layer – Latency in Error Reporting

- Errors detected well after the Vulkan API call that caused them (aka at vkQueueSubmit time)
- Difficult to provide meaningful error messages
- Opportunity to improve error messages:
  - Storing information for later use without unbearable performance impacts
GFXReconstruct - Vulkan Swapchain

- Different swapchain modes present and return images in different order
  - From run to run
- No swapchain presentation mode guarantees return order!
- GFXReconstruct Opportunity: How can we display the correct image during replay?
  - Solution: Implemented a virtual swapchain
  - Same number of images in replay as in capture
  - Use the indices in the same order from capture to replay
GFXReconstruct - API Explicitness

● Portability Challenge
  ○ Vulkan API is explicit
  ○ Hence captures from one GPU can’t be replayed on another GPU

● Conflicting Use Cases
  ○ Exact API calls needed for analysis
  ○ Use existing captures on newer/different GPUs

● Opportunity: How to enable some portability of captures
  ○ Collect additional data?
  ○ Translation layer?
The GPU-centric Universe

- GPUs - no longer "Graphics Processing Units"
  - Efficient processing of large blocks of data simultaneously
  - Compute - AI and ML

- Less Graphics API usage on the CPU
  - Rendering complexity still increasing via GPU driven rendering

- Many workloads moving to the GPU
  - Maximize utilization of GPU features
  - Reduce CPU interaction
D3D12 Work Graphs – GPU Autonomy

- **GPU Autonomy**
  - GPU Feeds itself
- **Dynamic Work Expansion**
  - Shader threads (producers) requesting work to run (consumers)
- Removes round trips to CPU
- Currently not available in Vulkan

![D3D12 Work Graphs – DirectX Developer Blog. March 11, 2024](image_url)
GFXReconstruct - GPU Autonomy

- Information no longer known at a function device call from the CPU side
- Addresses baked into capture content
  - Needs to be a different address during replay
GPU-Centric Universe: Developer Tools

Implications

- Debugging on a CPU vs GPU
  - CPUs provide the Instruction Set Architecture (ISA) and ability to step thru code
  - GPUs can be a black box and intrinsically different
    - Imagine stepping through 1 of a million items in a massive parallelism environment!

- Cross-GPU open-source tools are useful today
  - Evolve the tools for the GPU-centric universe
  - Cooperation needed from many parties
    - IHVs
    - Specification definitions
    - Tool writers
An Example API “hook”

- Vulkan “bufferDeviceAddressCaptureReplay”
  - Enable in driver during capture
  - Query memory location upon allocation
  - Can use that same memory allocation during replay
  - Current limitation: Not guaranteed to work from one vendor to another
From the launch of Vulkan to Today…

● There is ONE Industry-standard Vulkan desktop SDK
  ○ Wide adoption
  ○ Strong satisfaction
  ○ Open and free for all developers
  ○ Cross-platform SDK: Windows-x64/x86, Windows on arm, Linux, Apple platforms

● Valuable developer tools
  ○ Robust in features and reliability
  ○ Providing real value to Vulkan application developers
From the launch of Vulkan to Today…

● There is ONE Industry-standard Vulkan SDK
  ○ Wide adoption
  ○ Strong satisfaction
  ○ Open and free for all developers
  ○ Cross-platform: Windows, Linux, Android, Apple platforms

● Valuable developer tools
  ○ Robust in features and reliability
  ○ Providing real value to Vulkan application developers

LunarG Purpose Continues!
Evolve the tools for a GPU-centric universe!
Karen’s Reflection on the LunarG Journey

- The Power of being “Purpose Driven”
  - The ability to overcome adverse conditions to achieve amazing results!
- A Gift to the Vulkan Ecosystem
  - Useful
  - Impactful
  - Lasting and can be carried forward