Next Generation Graphics GPU Shader and Compute Libraries

INTRODUCTION

The graphics and display technology industries are going through major change, rapidly adopting the new Vulkan® graphics and compute technology. Current open standards such as OpenGL® and OpenCL® have served the industry well for many years, so where is the incentive for integrators to make the change to Vulkan?

This white paper provides a history of graphics and compute standards, as well as graphics technology, and discusses the new Vulkan graphics and compute libraries available for specialist industries; those with more stringent safety requirements such as aerospace, automotive, and transportation.

A HISTORY OF GRAPHIC SYSTEMS

The use of graphics systems first appeared in the early 1990's on Silicon Graphics Workstations using a proprietary graphics library called IRIS GL. IRIS GL lead to the creation of Open Graphics Library (OpenGL), which was formalized in 1992. The first IBM PC graphics—or text mode displays—eventually evolved into "real graphics" with the advent of extended Video Graphics Array (VGA) standard. Today, compute capabilities are a hot topic. With compute technology, the GPU can perform normal graphics output, as well as make use of the many available GPU cores as concurrent processing nodes.

When OpenGL was first formalized in 1992, there were Industry Standard Architecture (ISA) bus graphics cards in use with VGA graphics. Today, AMD and NVIDIA PCI Express graphics cards facilitate smooth, detailed game play. The availability of ample free compute cores on modern GPUs allow them to be used for gaming, cryptocurrency mining, deep machine learning, and many more advanced scenarios. Since 1992, there has been a technology drive from ISA [1], through to PCI, AGP, and now PCI Express[2] to increase graphics throughput. Core graphics technologies have also evolved from earlier, very primitive versions to today's modern graphics architectures. Figure 1 and 2 below illustrate the technology differences.

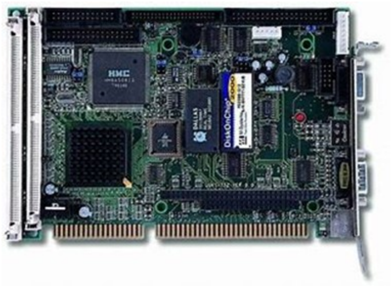


Figure 1: ISA BUS VGA Graphics card, circa 1990's

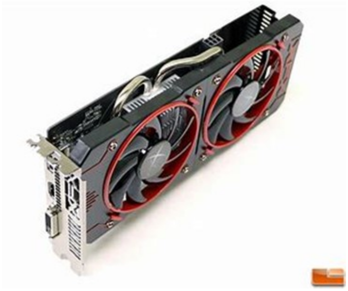


Figure 2: PCI Express with 1k+ cores circa now

The new graphics chipsets are a total phase shift in design, but still support OpenGL standards. In commercial industries, preferences have been evolving to favour new generation technology like Vulkan that offer big leaps in performance. This adoption is already underway in the gaming industry for next generation games such as Artifact, Doom, Hitman, and Quake, which use new AMD and NVIDIA GPUs. An eventual shift from OpenGL, OpenCL, and Compute Unified Device Architecture (CUDA) to Vulkan could mean the end of an era for these technologies.

To understand the graphic library standards, we must first examine their origins and progression, as well as the reasons opposing graphics libraries have been created along the way.

OpenGL

The OpenGL standard took a number of years to evolve. Competition in the computer industry forced secrecy and the creation of different standards amongst the key players of the day: Silicon Graphics, Sun Microsystems, IBM, and Hewlett-Packard. Eventually, these manufacturers and others agreed to participate in the OpenGL Architecture Review Board (ARB), and the first major OpenGL 2.0 specification was released in 2004. The ARB passed control of the specification to the Khronos Group[4], which is a non-profit member funded consortium focused on open standard royalty-free Application Programming Interfaces (APIs).

OpenGL's design allowed graphical applications to use a wide range of programming languages using language bindings, including 'C', Java, and JavaScript, to access the OpenGL library functions. The JavaScript based web browser graphics library (WebGL), which is based on OpenGL ES 2.0, allowed 3D rendering from within a web browser to enhance users' web browsing experiences.

OpenGL was not intended for the final GL contexts, windowing, video capture, or physical display of graphics using the GPU hardware. Rather, it was defined as a common set of rendering API, which may be called by the application to render without hardware dependence. However, hardware vendors can also add their own extensions to OpenGL, which are defined in the OpenGL Registry maintained by the Khronos Group. This allows new hardware functions to be introduced by newer GPU hardware architectures.

OpenGL introduced shader language support to provide more control of the graphics pipeline without using assembly language or hardware specific languages (a requirement in the early days) to make shader applications. A shader is a type of computer program used for shading levels of light, darkness, and color within an image. 2D, 3D, pixel, vertex, geometry, and compute shaders also exist, which are typically compiled and run on graphics hardware[3]. Figure 3 illustrates the Graphics Pipeline that results in all commands being executed on the GPU frame buffer. Note that OpenGL is only concerned with the frame buffer and does not deal with any other peripherals that may be available for use by the display subsystem.

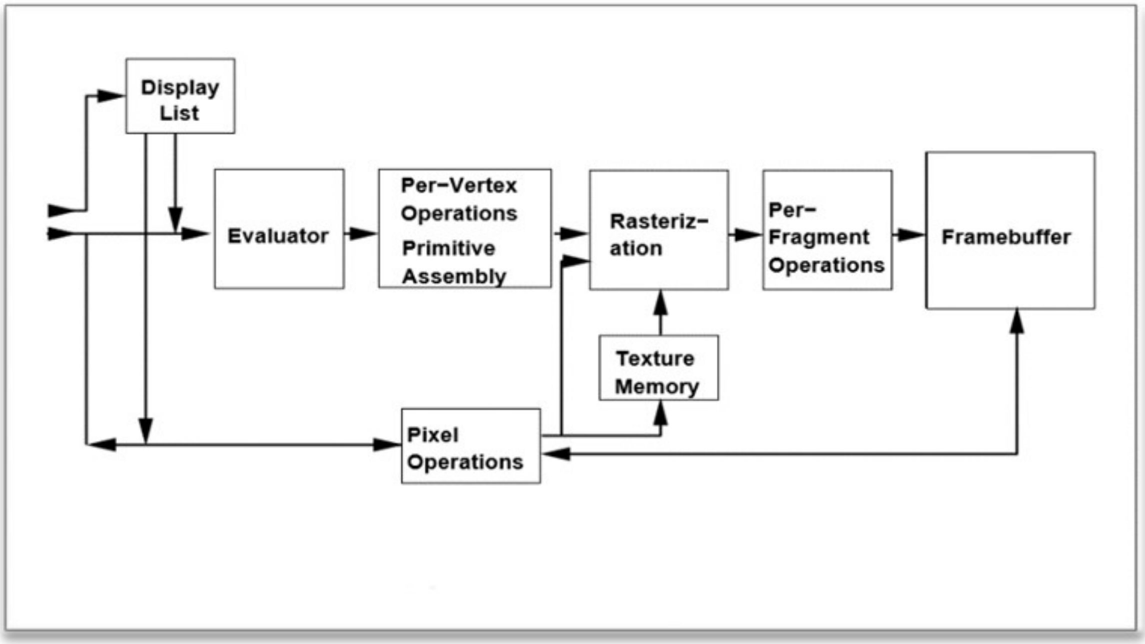


Figure 3: OpenGL Graphics Pipeline

OpenCL and CUDA

OpenCL and CUDA were derived differently than OpenGL. CUDA allowed the use of GPU cores by processor intensive software algorithms, and OpenCL was aimed firmly at programs that could execute across heterogeneous platforms to include CPUs, GPUs, DSP, FPGA and other hardware accelerator platforms. CUDA defined the term General Purpose Computing on Graphics Processing Unit (GPGPU), where software algorithms may be run on GPU cores to perform intensive processing, leaving the CPU cores free to perform other tasks.

CUDA, which is proprietary to NVIDIA, was created solely for NVIDIA hardware and is still being developed today, with emphasis on High Performance Computing (HPC), game development, cryptography, and many other applications. NVIDIA also provides conformance of GPU products to OpenCL, where the purpose is firmly based in heterogeneous computing. Figure 4 illustrates the CUDA development and runtime system. Note that software functions must be compiled for both the CPU architecture and GPU architecture because GPGPU algorithms are designed to run on the GPU cores, which are a different model from cores on a processor, that may be X86, ARM, or Power Architecture. Figure 4 illustrates the CUDA process for GPGPU compute.

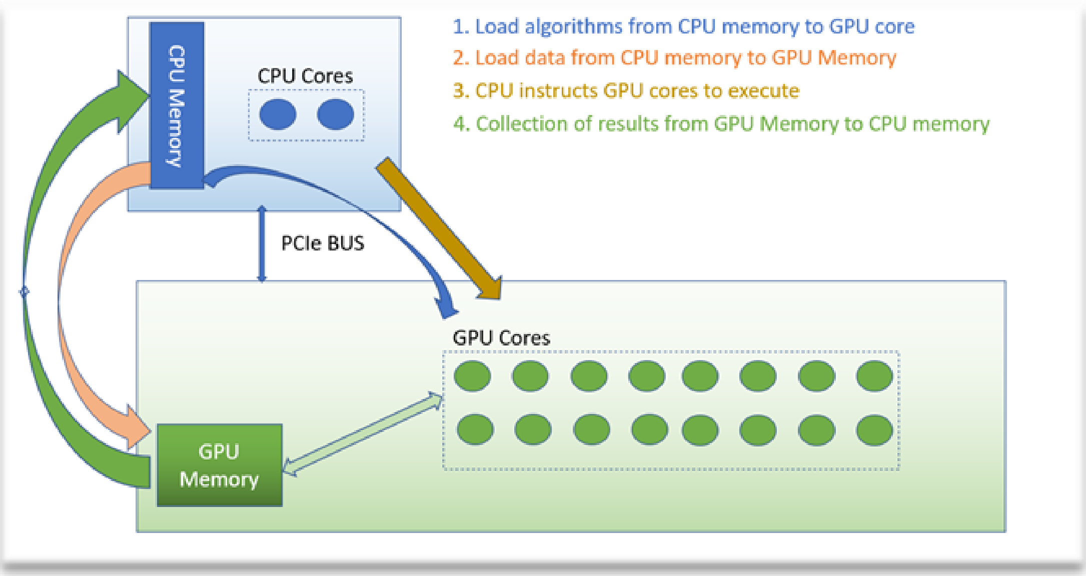

1. Load algorithms from CPU memory to GPU core
2. Load data from CPU memory to GPU Memory
3. CPU instructs GPU cores to execute
4. Collection of results from GPU Memory to CPU memory
CPU Memory
CPU Cores
PCIe BUS
GPU Cores
GPU Memory

Figure 4: Illustration of CUDA GPGPU Compute

The Khronos Group currently manages OpenGL as well as OpenCL. We will examine OpenCL, which is supported by industry standard GPU manufacturers where it is possible to use OpenCL with several different GPUs at the same time, as independent compute unit (CU) devices.

OpenCL is a framework for parallel programming with no graphics aspect, where applications intended for CU devices are written in Standard Portable Intermediate Representation (SPIR). SPIR is an open standard managed by Khronos that applies to OpenCL as well as Vulkan, where SPIR-V is defined for the integration of Vulkan and aspects of OpenGL and OpenCL.

The SPIR 1.2 release was designed to map OpenCL 1.2 to Low Level Virtual Machine (LLVM 3.2), where LLVM was originally developed by Apple to provide a method of compiling different languages into a binary object compatible for parallel processing on a range of compute units. Compilations of object code to run on cores can be run offline or during target runtime.

The goal of SPIR is to be vendor neutral, portable, and to allow management of the specification as an open standard for the future, as well as to encourage the use of SPIR with OpenCL.

Figure 5 illustrates a typical OpenCL system.

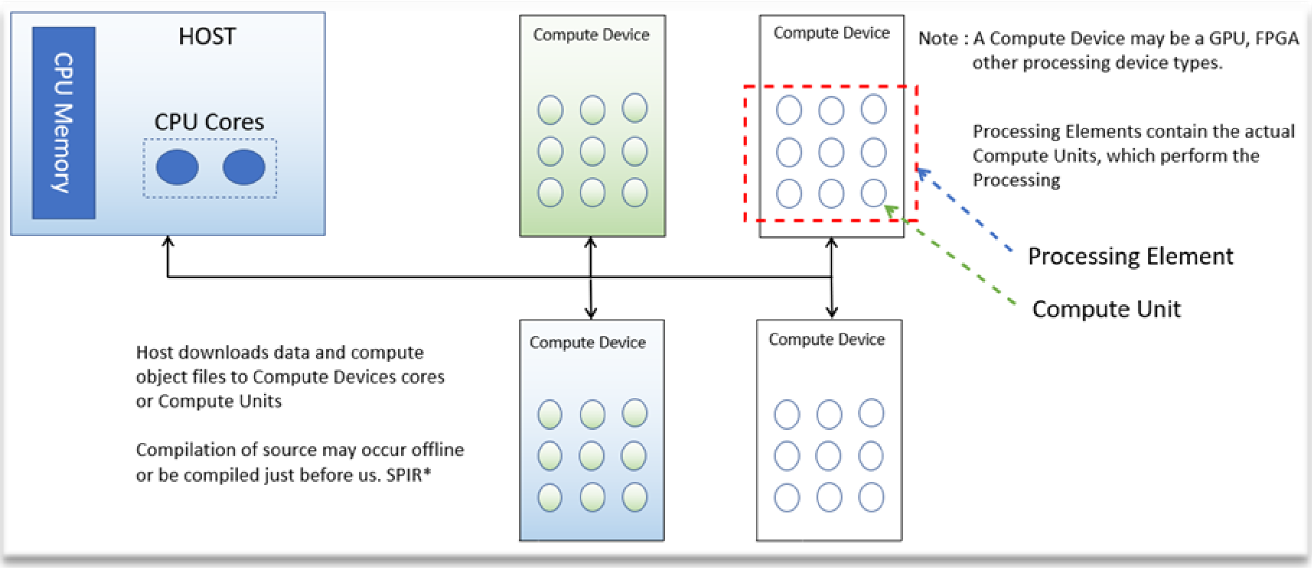

HOST
CPU Memory
CPU Cores
Compute Device
Compute Device
Note : A Compute Device may be a GPU, FPGA other processing device types.
Processing Elements contain the actual Compute Units, which perform the Processing
Processing Element
Compute Unit
Host downloads data and compute object files to Compute Devices cores or Compute Units
Compilation of source may occur offline or be compiled just before us. SPIR*
Compute Device
Compute Device

Figure 5: Illustration of OpenCL Compute

The goal of OpenCL is to make all compute devices logical with respect to the host processor, where each compute device has processing elements with one or more compute units. The compilation of the compute software algorithms is typically carried out on the development machine, but it is still possible to compile at runtime using the SPIR system on embedded platforms.

VULKAN: THE NEXT GEN OF GRAPHICS AND COMPUTE

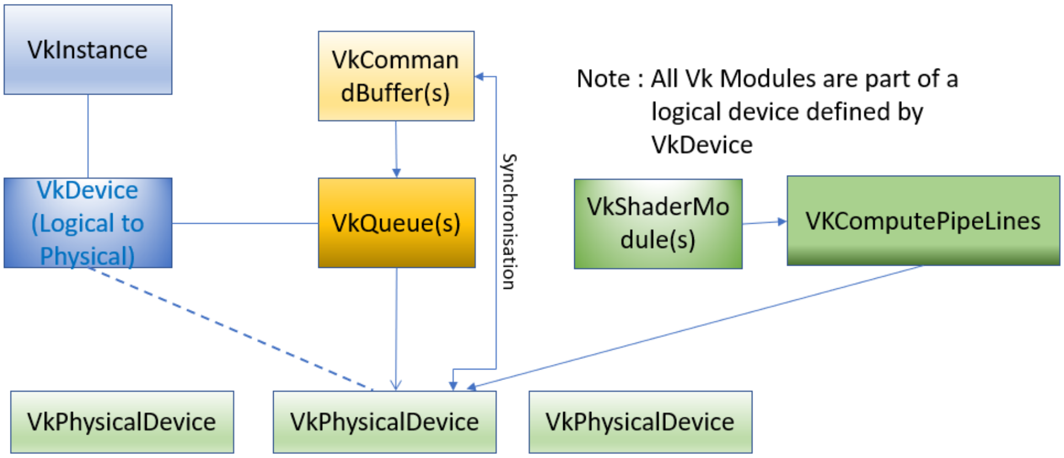
Vulkan is the next generation graphics and compute library combining capabilities from both OpenGL and OpenCL. It is managed as an open source library to allow the industry to evolve the standard though Khronos. This is fundamental for industries in which safety certification is critical. The Vulkan API is a whole new world for OpenGL users as, similar to OpenCL and CUDA, it allows low level access to the GPU. As the result of its inherent performance boost, Vulkan initially appealed largely to game developers; however, the Khronos Group is in the process of defining Vulkan safety critical libraries that have captured the attention of aerospace, automotive, and transportation industries in which OpenGL SC is used and certification is required.

Vulkan key components are execution units, work queues, command buffers, pipelines, subgroups, memory buffers, and the Vulkan device (or VkDevice), to which all commands are applicable. Vulkan also makes use of SPIR-V, which provides the compiler system for GPU environments. Currently, there are several low level guides on ‘how to program’ Vulkan, but a notable deficiency of high level information on the Vulkan software architecture.

The most important difference between OpenGL/OpenCL and Vulkan is that Vulkan opens up the world of shader and GPGPU compute. Shader compute allows for the creation of graphics directly using the GPU, whereas OpenGL used the host processor to create the graphics pipeline before display. This means that Vulkan is a dimension faster than OpenGL, dramatically reducing the host processor in the graphics system, which in turn reduces host CPU load and improves temperature performance.

There are extensions for OpenCL and other aspects of the Vulkan APIs to allow use of custom hardware features of GPUs[4]. This OpenGL shading language is supported by SPIR-V.

Figure 6 provides a high level overview of the Vulkan software architecture.



Note : All Vk Modules are part of a logical device defined by VkDevice

Note : Defines the PCI data associated with the physical GPU, such as Device and Vendor and device enumeration.

Figure 6. High Level Vulkan Software Architecture Overview

SAFETY CRITICAL VULKAN

Vulkan SC is a new specification being created to support aerospace, automotive, and transportation industries to provide Vulkan technology inside new certified platforms. Currently, Vulkan SC supports a small number of platforms, including the AMD Radeon™ E9171 GPU and the NXP i.MX 8 SoC with Vivante's GPU; however, the Vulkan ecosystem will expand and improve as adoption by safety critical projects increases.

The key differences between the new Vulkan (1.1) release and Vulkan SC release include:

- Vulkan 1.0 API is implemented with changes
- SPIR-V shader compiling and linking is offline and can only be performed on a host development system
- Pipeline cache and pipeline derivative functionality differs
- Freeing of memory is not allowed in Vulkan SC
- Vulkan SC will have additional callback mechanisms to deal with command buffer memory exhaustion and fatal error handling.

The Vulkan SC API is under review and is expected to be ratified soon. CoreAVI's VkCore® SC graphics and compute driver based on the Vulkan SC API is available now. CoreAVI has also created application libraries— VkCoreGL™ SC1 and VkCoreGL™ SC2—to provide support for OpenGL SC 1.0.1 and SC 2.0 libraries, which allows existing HMI tools such as ANSYS® SCADE Suite, Presagis VAPS XT, DiSTI's GL Studio®, ENSCO's IDATA®, and more to build safety certifiable OpenGL displays using the Vulkan SC library. This also allows shader and general purpose compute access using the Vulkan SC library.

Figure 7 provides an overview of the Vulkan SC library from CoreAVI.

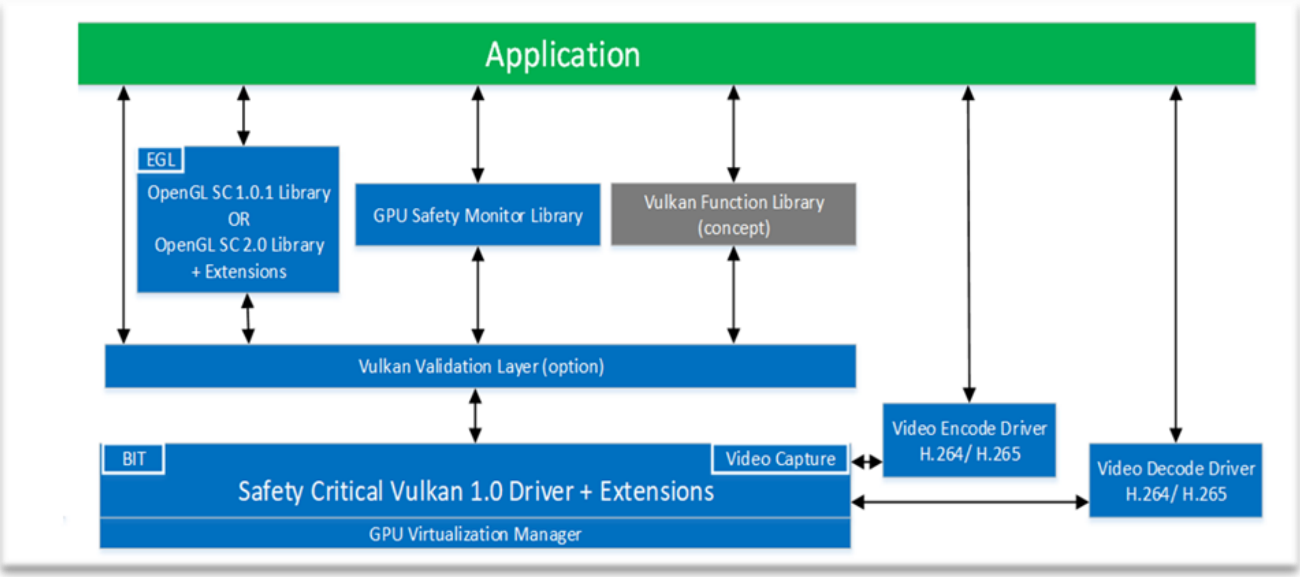


Figure 7: Vulkan SC Library Overview

VkCore SC provides many capabilities that are of interest to the safety certification community, notably the GPU safety monitor library, which is a software library instead of a traditional FPGA GPU safety monitor.

VkCore SC aims to incorporate future Vulkan Function Libraries that will provide ready-made, certifiable computer libraries to enable image edge detection, outline, color, and contrast adjustment, as well as Fast Fourier Transform (FFT) functionality.

CONCLUSION

Vulkan SC is the only graphics and compute library available for use in safety certifiable applications. CoreAVI's VkCore SC Vulkan-based safety critical graphics and compute driver, along with VkCoreGL SC1 and VkCoreGL SC2 OpenGL SC application libraries address next gen safety critical requirements for graphics and compute capabilities in avionics, automotive, and other transportation platforms. Contact Sales@CoreAVI for more information.

AUTHOR

Robert Pickles

Senior Field Application Engineer

Robert Pickles has almost 30 years of avionics software and systems level experience from the Royal Air Force, SBS Technologies, GE Intelligent Platforms and BAE Systems. Robert has previous OpenGL experience on certified avionics projects and is pleased to be working with CoreAVI and their customers on the new CoreAVI safety cert Vulkan graphics libraries. Robert was previously a solution architect at SYSGO, also directly involved with certification solutions for avionics and transportation system.

REFERENCES

[1] https://www.bing.com/images/search?view=detailV2&ccid=j3JTpHzu&id=8FA13BEE39CF188FC0468AB31A56FF8DE12BCC92&thid=OIP.j3JTpHzu18C1EPA6vrv0PQAAAA&mediaurl=http%3a%2f%2fwww.voxtechnologies.com%2fSBCs%2fimages%2ficp%2fjuki_750es1_1.jpg&exph=175&expw=236&q=isa+bus+vga&simid=608030650983579825&selectedIndex=11

[2] https://www.bing.com/images/search?view=detailV2&ccid=R19B3hcW&id=CD97B291D85790AF093DCF2640545499DD04E961&thid=OIP.R19B3hcW4hOl86FBxdT9awHaGM&mediaurl=https%3a%2f%2fwww.legitreviews.com%2fwp-content%2fuploads%2f2016%2f08%2fxfx-rx460-4gb-angle.jpg&exph=795&expw=950&q=Radeon+Graphics+Cards&simid=608032171402594626&selectedIndex=72

[3] https://en.wikipedia.org/wiki/File:CUDA_processing_flow_(En).PNG

[4] https://www.khronos.org/registry/spir-v/