Next Generation Graphics GPU Shader and Compute Libraries

Introduction

The graphics and display technology industries are going though major change, rapidly adopting the new Vulkan® graphics and compute technology. Since current open standards such as OpenGL® and OpenCL® have served the industry so well for many years, why should integrators make the change to Vulkan graphics?

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 such as aerospace, automotive and transportation, that have more stringent safety requirements.

A History of Graphics Systems

The earliest use of graphics systems appeared in the early 1990's on Silicon Graphics Workstations using IRIS GL, which was a proprietary Graphics Library. IRS GL lead to the creation of OpenGL, which was formalised in 1992. Also in recent memory is the excitement of the first IBM PC graphics, or rather text mode displays, which eventually became "real graphics" with the advent of extended VGA standard (Video Graphics Array). In present day, compute capabilities have become a very 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 formalised in 1992, there were ISA bus (Industry Standard Architecture) graphics cards in use with VGA graphics, while now we have AMD and Nvidia PCI Express graphics cards that can facilitate smooth detailed game play. Today's availability of ample free compute cores on the GPU allow it 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 **ref[1]**, through to PCI, to AGP and now to PCI Express **ref[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 concept, but still support OpenGL standard(s). In commercial industries, preferences have been evolving to favour new generation technology like Vulkan that offer big leaps in performance. With the new AMD and Nvidia GPUs, that adoption is already happening in the gaming industry for next generation games such as Artifact, Doom, Hitman and Quake to name a few. This means an eventual shift from OpenGL, OpenCL and CUDA to Vulkan, which could spell the end of an era for these technologies in the future.

In order to understand the graphic library standards, we must look at their origins, progression and of course the reasons why other opposing graphics libraries have been created along the way.

OpenGL

The OpenGL standard took a number of years to evolve. Due to the nature of the computer industry, competition forced secrecy and different standards between the key players of the day Silicon Graphics, Sun Microsystems, IBM and Hewlett-Packard. Eventually, all key and other manufacturers participated in the OpenGL Architecture Review Board (ARB), where the first major OpenGL 2.0 specification was released in 2004. ARB passed control of the specification to the Khronos Group **ref[4]** who are 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 for those including 'C', Java, and Javascript in order to access the OpenGL library functions. The Javascript based web browser Graphics Library (WEBGL), based on OpenGL ES 2.0, allowed 3D rendering from within a web browser, which enhances users' web browsing experiences.

OpenGL was not intended for the final GL contexts, windowing, video capture or physical display of the graphics using the GPU hardware, but instead was defined as a common set of rendering API, which may be called by the application to render without hardware dependence. However, hardware vendors are also able to add their own extensions to OpenGL, which are defined in the 'OpenGL Registry' maintained by the Khronos Group. This allows for new hardware functions 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 colour within an image and there also exists 2D, 3D, pixel, vertex, geometry and compute shaders, which are typically compiled and run on graphics hardware, **ref[3].** Figure 3 illustrates the "Graphics Pipeline" which results in all commands being executed on the GPU frame buffer. It's interesting to 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.

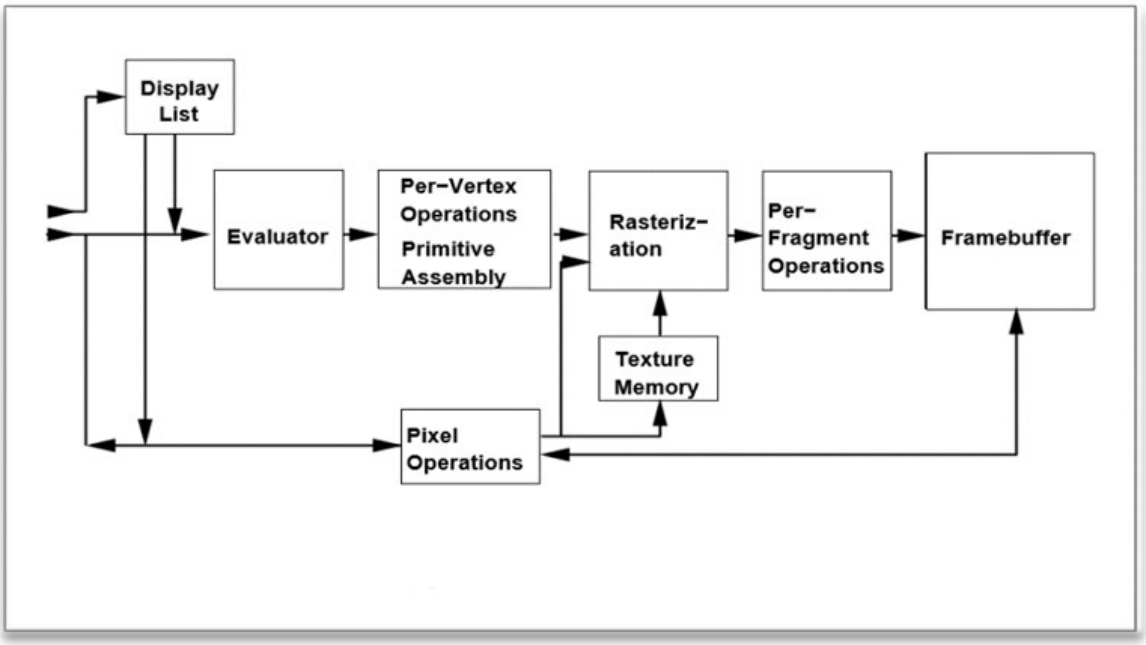

Figure 3: OpenGL Graphics Pipeline

OpenCL and CUDA

Open Computing Language (OpenCL) and Compute Unified Device Architecture (CUDA) were derived through a different path 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 GPGPU or **G**eneral **P**urpose computing on **G**raphics **P**rocessing **U**nit, where software algorithms may be run on GPU core(s) 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. However, NVIDIA also provides conformance of GPU products to OpenCL, where the purpose is firmly based on heterogeneous computing. Figure 4 illustrates the CUDA development and runtime system, where it's important to note that software functions need to be compiled for both the CPU architecture and GPU architecture. This is 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 PPC. Figure 4 illustrates the CUDA process for GPGPU compute.

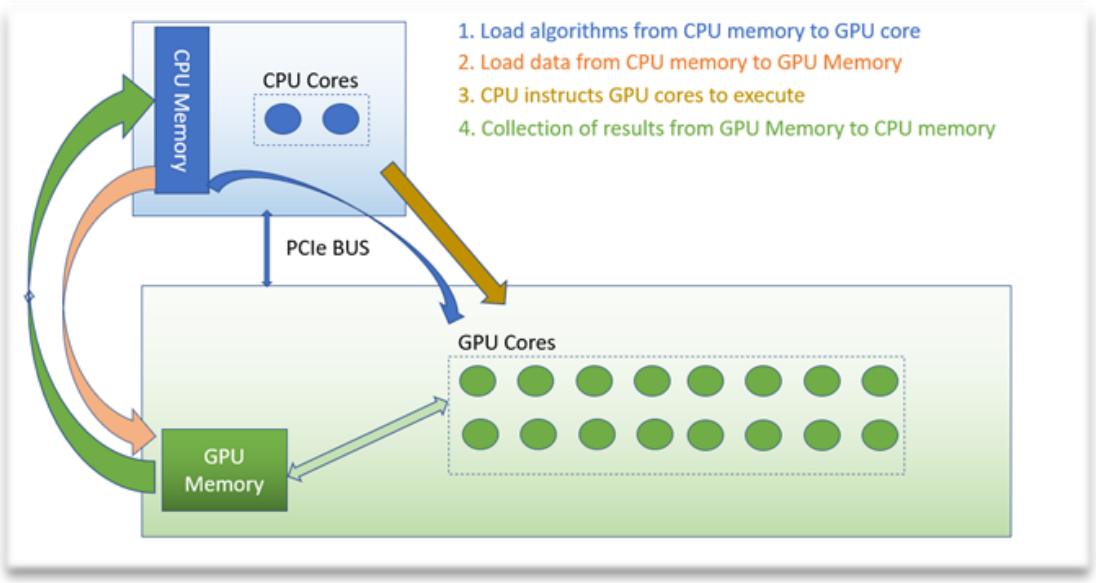

Figure 4: Illustration of CUDA GPGPU Compute

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

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

SPIR 1.2 release was defined to map OpenCL 1.2 to LLVM 3.2 (Low Level Virtual Machine), 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.

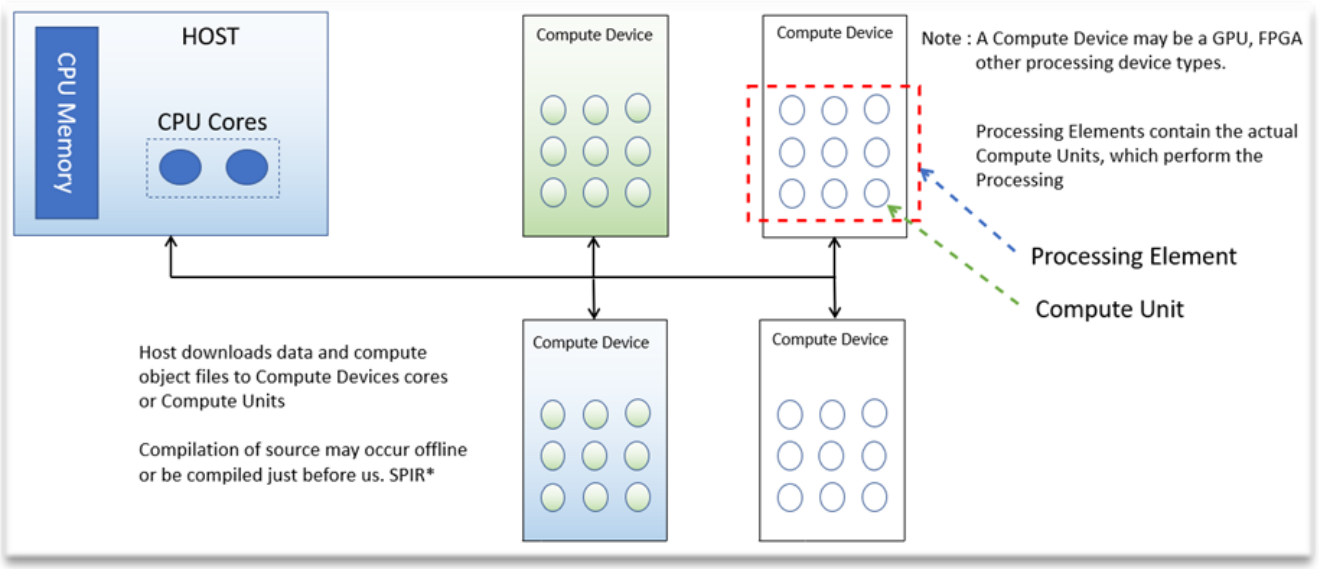

Figure 5: Illustration of OpenCL Compute

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's still possible to compile at run time 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 and is managed as an open source library to allow the industry to evolve the standard though Khronos. This is fundamental for certain industries where safety certification is concerned. The Vulkan API is a whole new world for OpenGL users as it allows low level access to the GPU, similar to OpenCL and CUDA. Due to the performance increase that can be achieved, Vulkan initially appealed largely to game developers. However, the Khronos Group is in the process of defining Vulkan safety critical libraries, which has captured the attention of aerospace, automotive and transportation industries where 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 (Standard Portable Intermediate Representation for Vulkan), which provides the compiler system for GPU environments. Currently, there are several low level guides on 'how to program' Vulkan, but a lack of high level information on the Vulkan software architecture.

The most important change between OpenGL/OpenCL and Vulkan is that Vulkan opens up the world of shader compute, as well as GPGPU compute. Shader compute allows for the creation of graphics directly using the GPU, where as OpenGL used the host processor to create the graphics pipeline before display. This means that Vulkan is a dimension faster than OpenGL and drastically reduces 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 **ref[4]**. This OpenGL shading language is supported by SPIR-V.

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

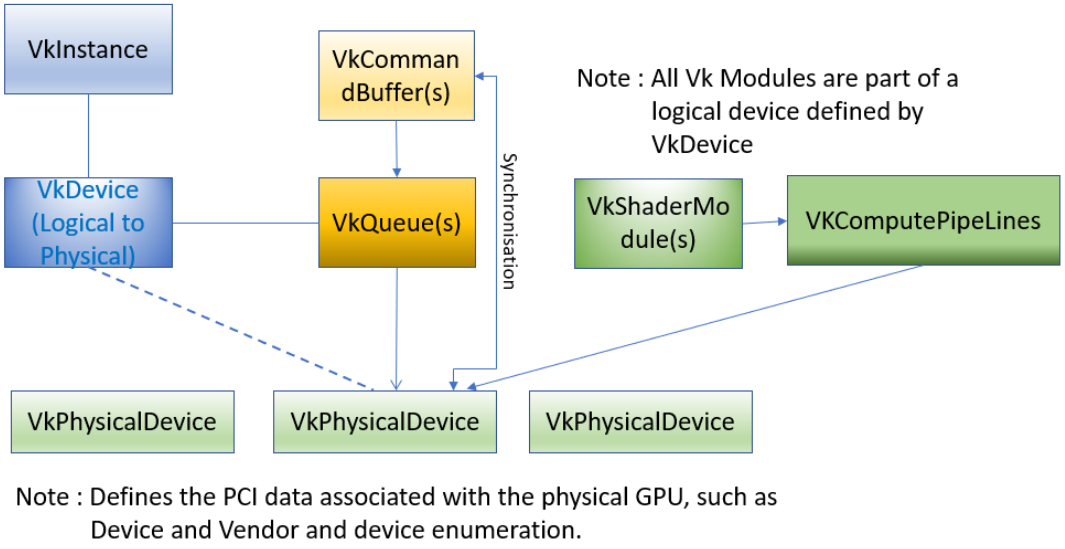

Figure 6. High Level Vulkan Software Architecture Overview

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 with adoption by safety critical projects.

The key differences between the new Vulkan (1.1) release and Vulkan SC release is as follows:

- The 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 scrutiny now and is expected to be ratified soon. CoreAVI's VkCore® SC graphics and compute driver based on the Vulkan SC API is now available. CoreAVI has also created application libraries called 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 others 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.

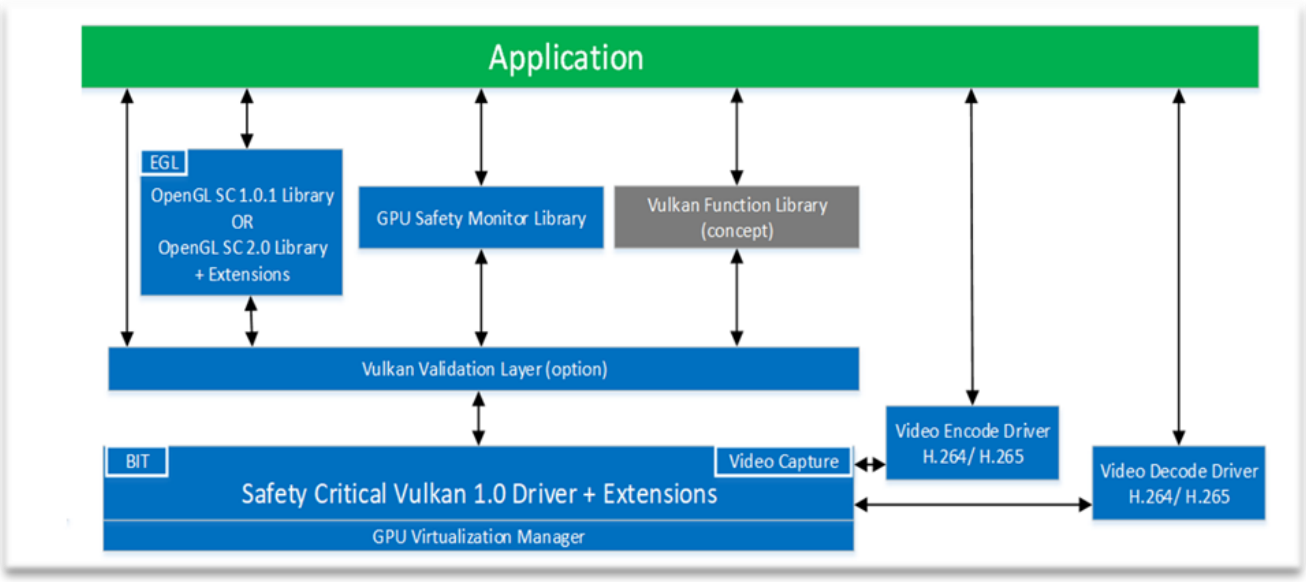

Figure 7: Vulkan SC Library Overview

VkCore SC provides many concepts 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, colour and contrast adjustment as well as FFT functionality (Fast Fourier Transform).

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/