Open Standards for Multicore Development
The State of the Union
Multicore Developers Conference, May 2015

Neil Trevett | Khronos President
NVIDIA Vice President Mobile Ecosystem
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, vision, sensor and camera processing

Rigorous specifications AND conformance tests for cross-vendor portability

Acceleration APIs BY the Industry FOR the Industry

Well over a BILLION people use Khronos APIs Every Day...
Multicore Topics Today

Framework for heterogeneous parallel programming
Low-level control over all system compute resources - now with C++!

Cross vendor Intermediate Language
Emerging foundational target for language compilers

Low power vision processing
Performance portability over diverse processor architectures

3D Graphics
Pervasive and safety critical 3D graphics

New generation 3D graphics
Direct control over GPUs for graphics and compute
OpenCL - Parallel Programming - HPC to Mobile

- Heterogeneous parallel programming of diverse compute resources
  - Targeting supercomputers -> embedded systems -> mobile devices
- One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware
  - Dynamically interrogate system load, balance work across available processors
- E.g. a single programming framework for all processors on a mobile SOC
  - Multi-core CPUs, GPUs, DSPs - even ISPs and specialized hardware blocks
OpenCL - Portable Heterogeneous Computing

- OpenCL = Two APIs and Two Kernel languages
  - C Platform Layer API to query, select and initialize compute devices
  - C Runtime API to build and execute kernels across multiple devices
  - OpenCL C and OpenCL C++ kernel languages to write parallel code
OpenCL 2.1 Provisional – March 2015

- New OpenCL C++ kernel language based on a subset of C++14
  - Significantly enhanced programmer productivity and code performance

- Support for the new Khronos SPIR-V intermediate language in core
  - SPIR-V used to ingest from C++ front-end - no C++ compiler in driver
  - OpenCL C ingestion still supported to preserve kernel code investment

- Runs on any OpenCL 2.0-capable hardware
  - Only driver update required

OpenCL C++ Shading language
SPIR-V in Core
Subgroups into core
Subgroup query operations
clCloneKernel
Low-latency device timer queries

Dec08
OpenCL 1.0 Specification
18 months

Jun10
OpenCL 1.1 Specification
18 months

Nov11
OpenCL 1.2 Specification
24 months

Nov13
OpenCL 2.0 Specification
16 months

Mar15
OpenCL 2.1 Specification (Provisional)

3-component vectors
Additional image formats
Multiple hosts and devices
Buffer region operations
Enhanced event-driven execution
Additional OpenCL C built-ins
Improved OpenGL data/event interop

Device partitioning
Separate compilation and linking
Enhanced image support
Built-in kernels / custom devices
Enhanced DX and OpenGL Interop

Shared Virtual Memory
On-device dispatch
Generic Address Space
Enhanced Image Support
C11 Atomics
Pipes
Android ICD
OpenCL C++

- The OpenCL C++ kernel language is a static subset of C++14
  - Frees developers from low-level coding details without sacrificing performance

- C++14 features removed from OpenCL C++ for parallel programming
  - Exceptions, Allocate/Release memory, Virtual functions and abstract classes
  - Function pointers, Recursion and goto

- Classes, lambda functions, templates, operator overloading etc..
  - Fast and elegant sharable code - reusable device libraries and containers
  - Templates enable meta-programming for highly adaptive software
  - Lambdas used to implement nested/dynamic parallelism

- C++11-based standard library optimized for data-parallel programming
  - Atomics, meta-programming & type traits, math functions...
  - Plus new library features: Work-item & Work-group functions,
    Dynamic parallelism, Image & Pipe functions...

Highly adaptive parallel software that delivers tuned performance across diverse platforms
OpenCL Implementations

Vendor timelines are first implementation of each spec generation

- **Dec08**  OpenCL 1.0 Specification
- **Jun10**  OpenCL 1.1 Specification
- **Nov11**  OpenCL 1.2 Specification
- **Nov13**  OpenCL 2.0 Specification
- **Mar15**  OpenCL 2.1 Specification

**Desktop**

- 1.0 | May09
- 1.1 | Jul11
- 1.2 | Jun12
- 2.0 | Dec14

**Mobile**

- 1.0 | Jun10
- 1.1 | Jun10
- 1.2 | Sep13
- 1.2 | Apr14
- 1.2 | Dec14
- 1.2 | Sep14

**Embedded**

- 1.0 | Jan10
- 1.1 | May10
- 1.1 | Aug12
- 1.2 | Sep13
- 1.2 | May15
- 1.2 | May15

**FPGA**

- 1.0 | Jul13
- 1.0 | Dec14
- 1.2 | Mar15
- 1.2 | Apr14
- 1.2 | Dec14
- 1.2 | Sep14

OpenCL 1.0 Specification

- **Dec08**
- **May09**
- **Aug09**
- **May10**
- **May10**
- **Feb11**
- **Mar11**
- **Jun10**
- **Aug12**
- **Nov12**
- **May13**
- **Aug13**
- **May14**

OpenCL 1.1 Specification

- **Jun10**
- **Jul11**
- **Aug11**
- **Mar11**
- **Jun11**
- **Feb11**
- **Mar11**
- **Nov11**
- **Aug12**
- **Nov12**
- **May13**
- **Apr12**
- **Mar13**

OpenCL 1.2 Specification

- **Jun12**
- **May12**
- **Aug12**
- **Dec12**
- **Sep13**
- **Apr14**
- **Dec14**
- **Sep14**
- **May15**

OpenCL 2.0 Specification

- **Nov13**
- **Dec14**
- **Mar15**

OpenCL 2.1 Specification

- **Mar15**
OpenCL and Embedded Platforms

- OpenCL roadmap discussions focused on mobile and embedded markets
  - Different needs to HPC/desktop

- Dynamic load-balancing
  - Instrumentation/control to route work to different compute resources depending on loading/thermal conditions?

- Vision and sensor processing is important use case
  - Reduce precision requirements?

- Real-time / guaranteed QOS
  - Pre-emption, yielding, extended execution models?

- Embedded workload topologies
  - Extended pipes and graphs in OpenCL run-time?
OpenCL Ecosystem

Implementers
Desktop/Mobile/FPGA/Embedded

Additional Working Group Members
Apps/Tools/Tests/Courseware

Single Source C++ Programming

Core API and Language Specs

Portable Kernel Intermediate Language
SYCL for OpenCL - Single-source C++

• Pronounced ‘sickle’ to go with ‘spear’ (SPIR)

• Royalty-free, cross-platform C++ programming layer
  - Builds on concepts portability & efficiency of OpenCL
  - Ease of use and flexibility of C++

• Single-source C++ development
  - C++ template functions can contain host & device code
    - e.g. parallel_sort<MyType> (myData);
  - Construct complex reusable algorithm templates using OpenCL for acceleration

• SYCL 1.2 Provisional spec released at GDC in March 2014
  - Updated at Supercomputing November 2014
  - Expecting finalization very soon

• Multiple implementations
  - Including open source triSYCL from AMD
  - https://github.com/amd/triSYCL
OpenCL as Parallel Language Backend

- WebCL
- Halide
- C++ AMP
- River Trail
- lpfork
- aparapi
- OpenACC
- Intel
- Embedded array language for Haskell
- Compiler directives for Fortran, C and C++
- PyOpenCL
- Harlan
- High level language for GPU programming

JavaScript binding for initiation of OpenCL C kernels
Language for image processing and computational photography
MulticoreWare open source project on Bitbucket
Embedded array language for Haskell
Java language extensions for parallelism
River Trail Language extensions to JavaScript
Compiler directives for Fortran, C and C++
PyOpenCL Python wrapper around OpenCL

Approaching 200 languages, frameworks and projects using OpenCL as a compiler target to access vendor optimized, heterogeneous compute runtimes

This trend will be accelerated by the support of SPIR-V in OpenCL 2.1 Core
SPIR-V Transforms the Language Ecosystem

• Cross vendor intermediate representation
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form

• SPIR-V - first multi-API, intermediate language for parallel compute and graphics
  - Native representation for Vulkan shader and OpenCL kernel source languages

SPIR-V is a significant convergence point in the language ecosystem for graphics and parallel computation
SPIR-V at the Center of Language Ecosystem

* Khronos considering developing open source implementations of these translators

Other Intermediate Forms

OpenCL C

OpenCL C++

GLSL

New kernel and shader Languages

SPIR-V
- 32-bit Word Stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

OpenCL C++

OpEntryPoint
- Fragment shader
- function <id> 4

OpTypeVoid
- <id> is 2

OpTypeFunction
- <id> is 3
- return type <id> is 2

OpFunction
- Result Type <id> is 2
- Result <id> is 4
- 0
- Function Type <id> is 3

Vulkan Driver X

Vulkan Driver Y

OpenCL Driver A

OpenCL Driver B

Other Languages

LLVM

Other Intermediate Forms
SPIR-V Advantages for Developers

- Developers can use same front-end compiler across multiple platforms
  - Eliminating major source of cross-vendor portability
- Reduces runtime shader/kernel compilation time
  - Driver only has to process SPIR-V not full source language
- Don’t have to ship shader/kernel source code
  - Provides a measure of IP protection
- Drivers are simpler and more reliable
  - No need to include front-end compilers

SPIR-V Whitepaper
OpenVX - Vision Acceleration

- Royalty-free open standard API
  - Reliably accelerated by hardware vendors
  - Tightly defined conformance tests

- Targeted at low-power, real-time applications
  - Mobile and embedded platforms

- Portability across diverse heterogeneous processors
  - ISPs, Dedicated hardware, DSPs and DSP arrays, GPUs, Multi-core CPUs ...

- Doesn’t require high-power CPU/GPU Complex
  - Low-power host can setup and manage frame-rate vision processing pipeline

OpenVX extends easily re-usable vision acceleration to very low power domains
OpenVX Graphs - The Key to Efficiency

- OpenVX developers express a graph of image operations (‘Nodes’)
  - Nodes can be on any hardware or processor coded in any language

- Graph enables implementations to optimize for power and performance
  - E.g. Nodes may be fused by the implementation to eliminate memory transfers
  - E.g. Processing can be tiled to keep data entirely in local memory/cache

- Minimizes host interaction during frame-rate graph execution
  - Host processor can setup graph which can then execute almost autonomously

Example OpenVX Graph
Layered Vision Processing Ecosystem

• The higher abstraction level of OpenVX protects app from hardware differences
  - Increased application *functional* and *performance* portability

• Lower-level APIs can be used to implement OpenVX nodes
  - Depending on the available processors

• Layered software model enables powerful ecosystems
  - E.g. WebGL, Vulkan - and now OpenVX

---

<table>
<thead>
<tr>
<th>Application Software</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engines/frameworks</td>
<td>OpenVX</td>
</tr>
<tr>
<td>Powerful, flexible</td>
<td>C/C++</td>
</tr>
<tr>
<td>low-level APIs / languages</td>
<td></td>
</tr>
<tr>
<td>Processor Hardware</td>
<td>Dedicated Vision Hardware</td>
</tr>
</tbody>
</table>

Programmable Vision Processors

Implementers may choose to use OpenCL or Compute Shaders to *implement* OpenVX nodes on programmable processors

... And then use OpenVX to enable a developer to easily *connect* those nodes into a graph
OpenVX and OpenCV are Complementary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Community driven open source</th>
<th>Accelerated open standard API implemented by hardware vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conformance</td>
<td>No formal conformance Adopters program</td>
<td>Implementations must pass defined conformance test suite to use trademark</td>
</tr>
<tr>
<td>Consistency</td>
<td>Available functions can vary depending on implementation / platform</td>
<td>All core functions must be available in all conformant implementations</td>
</tr>
<tr>
<td>Scope</td>
<td>Very wide 1000s of imaging and vision functions Multiple camera APIs/interfaces</td>
<td>Tight focus on hardware accelerated functions for mobile vision Use external camera API</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Memory-based architecture Each operation reads and writes to memory</td>
<td>Graph-based execution Optimizable computation, data transfer</td>
</tr>
<tr>
<td>Typical Use Case</td>
<td>Rapid experimentation and prototyping Re-usable code</td>
<td>Production development &amp; deployment Callable Library</td>
</tr>
</tbody>
</table>
OpenVX Status

• Finalized OpenVX 1.0 specification released October 2014
  - www.khronos.org/openvx

• Khronos open source sample implementation of OpenVX 1.0 released
  - https://www.khronos.org/registry/vx/sample/openvx_sample_20141217.tar.gz

• Full conformance test suite and Adopters Program available
  - Working group reviews submitted results
  - Test suite exercises graph framework and functionality of each OpenVX 1.0 node
  - Approved Conformant implementations can use the OpenVX trademark
Access to 3D on Over 2 BILLION Devices

300M Desktops / year
Windows, Mac, Linux

1.9B Mobiles / year

1B Browsers / year

Source: Gartner (December 2013)
OpenGL ES and OpenGL SC Roadmap

2003
1.0
Driver Update

2004
1.1
Silicon Update

2007
2.0
Silicon Update

2011
3.0
Driver Update

2012
3.0

Compute Shaders

32-bit integers and floats
NPOT, 3D/depth textures
Texture arrays
Multiple Render Targets

2014
3.1
Spec at GDC March 2014
Standard in Android L

OpenGL SC 1.0 - Safety Critical Profile
- Minimum driver size and complexity for DO-178B certification
- Removes gaming functionality unneeded by instrumentation displays
- Restores functionality for legacy/auto-generated applications: e.g. display lists, paletted textures

70 functions
30 functions
The Need for Vulkan

Ground-up design of a modern open standard API for driving high-efficiency graphics and compute on GPUs used across diverse devices

In the twenty two years since OpenGL was invented - the architecture of GPUs and platforms has changed radically

GPUs being used for graphics, compute and vision processing on a rapidly increasing diversity of platforms - increasing the need for cross-platform standards
Vulkan Explicit GPU Control

Complex drivers lead to driver overhead and cross vendor unpredictability

Error management is always active

Driver processes full shading language source

Separate APIs for desktop and mobile markets

Application

Traditional graphics drivers include significant context, memory and error management

Application responsible for memory allocation and thread management to generate command buffers

Direct GPU Control

Simpler drivers for low-overhead efficiency and cross vendor consistency

Layered architecture so validation and debug layers can be unloaded when not needed

Run-time only has to ingest SPIR-V intermediate language

Unified API for mobile, desktop, console and embedded platforms

Vulkan delivers the maximized performance and cross platform portability needed by sophisticated engines, middleware and apps
Vulkan Multi-threading Efficiency

1. Multiple threads can construct Command Buffers in parallel. Application is responsible for thread management and synch.

2. Command Buffers placed in Command Queue by separate submission thread.

Vulkan can provide Compute queues without Graphics. Opportunity to evolve Vulkan for direct access to generalized compute resources over time.
Vulkan Status

• Rapid progress since project start in June 2014
  - Significant proposals and IP contributions received from members

• Participants come from all segments of the graphics industry
  - Including an unprecedented level of participation from game engine ISVs

• Initial specs and implementations expected this year
  - Will work on any GPU hardware that supports OpenGL ES 3.1/OpenGL 4.X and up
  - Can ship on any OS - including previous versions of Windows

Working Group Participants
OpenGL SC Nexgen Opportunity

- Avionics market experience with minimal fixed function 3D API
- GLSL Programmable shaders and bringing desktop functionality to mobile
- Significantly thinner and simpler driver model for GPU graphics and compute
- Well-defined intermediate language to avoid compiler complexity in driver

A good time to define next generation OpenGL SC with modern functionality in minimal driver footprint!
Summary

• Khronos is creating royalty-free open standards to access compute resources
  - For graphics, heterogeneous compute and vision processing
• Khronos standards are key to many emerging markets such as gaming, mobile and embedded
  - Advanced next generation capabilities for ALL platforms
• Any company is welcome to join Khronos for a voice and a vote in the evolution of these standards
  - $15K annual membership fee for access to all Khronos API working groups
  - Well-defined IP framework protects your IP and conformant implementations

• More Information
  - www.khronos.org
  - ntrevett@nvidia.com
  - @neilt3d