Update on Khronos Open Standard APIs for Vision Processing

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 and vision 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...
Khronos Standards for Vision Processing

Low-power, portable vision processing

Image: ScreenMedia

GPU Graphics and GPU Compute

Heterogeneous Parallel Processing for Vision and Neural Net acceleration
OpenGL ES Roadmap

1.0    1.1    2.0    3.0    3.1    3.2

- Fixed function pipeline
- Driver update
- Silicon update

- Programmable vertex and fragment shaders
- 32-bit integers and floats
- NPOT, 3D/depth textures
- Texture arrays
- Multiple render targets

- Compute shaders

- Tessellation and geometry shaders
- ASTC texture compression
- Floating point render targets
- Debug and robustness for security

Epic’s Rivalry demo using full Unreal Engine 4
https://www.youtube.com/watch?v=jRr-G95GdaM

The most widely deployed 3D graphics API in history
Industry will ship >1.7 billion OpenGL ES-enabled devices in 2015

http://hwstats.unity3d.com/mobile/gpu.html

© Copyright Khronos Group 2015 - Page 4
OpenGL ES Fragment Shaders

- Fragment Shaders in OpenGL 2.0 were the original ‘GPGPU’ technique (2004)
  - Fragment shaders executed as part of graphics pipeline
  - Need to configure inputs/outputs as textures and images

- Mobile fragment shaders arrived in OpenGL ES 2.0 (2007)
  - Now pervasively available on almost ANY mobile device or OS

- Easy integration of compute shaders into graphics apps - no API interop needed
  - Program kernels (shaders) in GLSL not C
  - Good for small kernels NOT complete apps
  - Limited to acceleration on a single GPU

<table>
<thead>
<tr>
<th>OpenGL ES Fragment Shaders</th>
<th>OpenGL ES 2.0+ Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
<td>Khronos</td>
</tr>
<tr>
<td>Acceleration Devices</td>
<td>GPU</td>
</tr>
<tr>
<td>Scope</td>
<td>Graphics+Compute</td>
</tr>
<tr>
<td>Explicit Memory/Execution</td>
<td>No</td>
</tr>
<tr>
<td>Host Bindings</td>
<td>C</td>
</tr>
<tr>
<td>Kernel Language</td>
<td>GLSL</td>
</tr>
<tr>
<td>Availability</td>
<td>OpenGL ES 2.0+</td>
</tr>
<tr>
<td>Precision</td>
<td>NO IEEE 754 (highp)</td>
</tr>
<tr>
<td></td>
<td>mediump / lowp</td>
</tr>
</tbody>
</table>
OpenGL ES Compute Shaders

- New class of shader introduced in OpenGL 4.4 (2012) and OpenGL ES 3.1 (2014)
  - Separated from graphics pipe - can use any buffer, image or texture
- Much more flexibility on how compute shader is executed and uses memory
  - But still use GLSL to write kernels and limited to execution on GPU only
- OpenGL ES 3.1 is less pervasive than fragment shaders
  - But mandated in Android Lollipop and Marshmallow - so growing rapidly
OpenCL

- Heterogeneous parallel programming of diverse compute resources
  - Targeting supercomputers -> mobile devices -> embedded systems
- One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware
  - Distribute work across all available processors in a system
- Can represent function of hardware ‘Custom Devices’ as built-in kernels
  - Control hardware from OpenCL run-time: e.g. video encode/decode, Camera ISP
- Robust framework for coding complete applications
  - One source with both CPU and accelerated paths

<table>
<thead>
<tr>
<th>Governance</th>
<th>Khronos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration Devices</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>Scope</td>
<td>Compute</td>
</tr>
<tr>
<td>Explicit Memory/Execution</td>
<td>Yes</td>
</tr>
<tr>
<td>Host Bindings</td>
<td>C/C++</td>
</tr>
<tr>
<td>Kernel Language</td>
<td>C/C++/SPIR-V</td>
</tr>
<tr>
<td>Availability</td>
<td>Any OS</td>
</tr>
<tr>
<td>Precision</td>
<td>Full IEEE 754</td>
</tr>
<tr>
<td></td>
<td>IEEE 754 Subset</td>
</tr>
<tr>
<td></td>
<td>IEEE 754 Relaxed</td>
</tr>
</tbody>
</table>
OpenCL - Portable Heterogeneous Computing

- OpenCL = Two APIs and Two Kernel languages
  - C Platform Layer API to query, select and initialize compute devices
  - OpenCL C and (soon) OpenCL C++ kernel languages to write parallel code
  - C Runtime API to build and execute kernels across multiple devices

- One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware
  - Dynamically balance work across available processors
OpenCL Implementations

Vendor timelines are first implementation of each spec generation

- **OpenCL 1.0 Specification**
  - **Dec08**: AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
  - **Jun10**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
  - **Nov11**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

- **OpenCL 1.1 Specification**
  - **Dec14**: Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
  - **Mar15**: Texas Instruments, STI, Mediatek

- **OpenCL 1.2 Specification**
  - **Mar15**: Texas Instruments, STI, Mediatek
  - **Nov13**: Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
  - **Dec13**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

- **OpenCL 2.0 Specification**
  - **Nov13**: Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
  - **Dec14**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

- **OpenCL 2.1 Specification**
  - **Mar15**: Texas Instruments, STI, Mediatek

**Desktop**
- **1.0 | May09**: AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.1 | Jul11**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.2 | Jun12**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

**Mobile**
- **1.0 | May09**: AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.1 | Jun10**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.2 | Aug12**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

**Embedded**
- **1.0 | Jan10**: AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.1 | May10**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.1 | Aug12**: Apple, IBM, AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante

**FPGA**
- **1.0 | Jul13**: AMD, Intel, NVIDIA, ARM, Imagination, Qualcomm, Vivante
- **1.2 | May15**
- **1.2 | Aug15**
Embedded Needs Driving OpenCL Roadmap

- OpenCL roadmap discussions focused on mobile and embedded markets
  - Very different needs and constraints to HPC/desktop

Different Bottlenecks to HPC
Mobile GPU have less processing throughput than desktop
GPU often busy with rendering hi-res screen
Mobile memory bandwidth is often perf bottleneck

Always-on vision / Neural Nets are key use cases
Must run at very low power levels
Thermal conditions often limit performance

Vision Applications Must Not Drop Frames
Real-time / guaranteed QOS

App performance portability is much more critical
More diverse architectures in mobile than HPC
1000s of different devices

Many diverse hardware processing blocks
DSP, ISP, dedicated hardware

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

Optimize Energy Efficiency
Reduce precision - even integer-only profiles?
Autonomous frame processing - no host involvement?

Extended Execution Model
Pre-emption, yielding

Higher-level portability frameworks and engines
Built over powerful low-level APIs

Expand support for Custom Devices
Extended pipes and graphs in OpenCL run-time?
OpenCL used to Accelerate OpenCV

- Extensive and widely used open source vision library - 1,000s of functions
  - Released under a free-use BSD license
  - Written in optimized C/C++

- C++, C, Python and Java interfaces
  - Windows, Linux, Mac OS
  - iOS and Android

- Increasingly taking advantage of heterogeneous processing using OpenCL
  - OpenCV 3.0 Transparent API - single API entry for each function/algorithm
  - Dynamically loads OpenCL runtime if available; otherwise falls back to CPU code
  - Runtime Dispatching - no recompilation!

- One queue and one OpenCL device per CPU thread
- Different CPU threads can share a device but use different queues
- OpenCL kernels are executed asynchronously
OpenCL as Parallel Language Backend

- JavaScript binding for initiation of OpenCL C kernels
- Language for image processing and computational photography
- MulticoreWare open source project on Bitbucket
- Single Source C++ Programming for OpenCL
- Java language extensions for parallelism
- River Trail Language extensions to JavaScript
- Compiler directives for Fortran, C and C++
- PyOpenCL Python wrapper around OpenCL
- Harlan High level language for GPU programming

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

This trend will be significantly accelerated by the availability of SPIR-V which is specifically designed to be a compiler target
OpenCL 2.1 Released - November 2015

- Support for the SPIR-V 1.0 intermediate language in core
  - E.g. SPIR-V used to ingest from diverse language front-ends
  - OpenCL C ingestion still supported to preserve kernel code investment

- OpenCL API updates
  - E.g. subgroups and subgroup queries in core

- Runs on any OpenCL 2.0-capable hardware
  - Only driver update required
SPIR-V Transforms the Language Ecosystem

- First multi-API, intermediate language for parallel compute and graphics
  - Native representation for Vulkan shader and OpenCL kernel source languages
- 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

**Multiple Developer Advantages**
- Same front-end compiler for multiple platforms
- Reduces runtime kernel compilation time
- Don’t have to ship shader/kernel source code
- Drivers are simpler and more reliable

**Standard Portable Intermediate Representation**

**Diverse Languages and Frameworks**

**Tools for analysis and optimization**

**Hardware runtimes on multiple architectures**
Driving the SPIR-V Open Source Ecosystem

SPIR-V Tools
- OpenCL C
- OpenCL C++
- SPIR-V Validator
- SPIR-V (Dis)Assembler
- LLVM to SPIR-V Bi-directional Translator
- SPIR-V (Dis)Assembler
- SPIR-V Validator
- SPIR-V Tools

LLVM

Third party kernel and shader Languages

GLSL

Other Intermediate Forms

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

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

Khronos plans to open source these tools soon

IHV Driver Runtimes

© Copyright Khronos Group 2015 - Page 15
Vulkan - Portable, Nexgen GPU Access

- Ground-up design - open standard for high-efficiency GPU graphics and compute
  - Simpler drivers for low-overhead efficiency and cross vendor consistency
- Unified API for mobile, desktop, console and embedded platforms
- Multi-threading friendly - create graphics, compute and DMA command buffers
  - General model - could be extended to heterogeneous processing
Next Generation GPU APIs

<table>
<thead>
<tr>
<th>DirectX 12</th>
<th>Only Windows 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Apple</td>
<td>Cross Platform</td>
</tr>
<tr>
<td>Any GPU with OpenGL ES 3.1/OpenGL 4.X and up</td>
<td></td>
</tr>
<tr>
<td>Any OS</td>
<td></td>
</tr>
</tbody>
</table>

Vulkan Target Availability

Vulkan Committed Platform Adoption
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

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

Driver processes full shading language source

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

Unified API for mobile, desktop, console and embedded platforms

Simpler drivers for low-overhead efficiency and cross vendor consistency

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

Vulkan delivers the maximized performance and cross platform portability needed by sophisticated engines, middleware and apps
No Compromise

Retains OpenGL Binding Model
(but missing functionality such as Tessellation and Geometry Shaders)

Amount of work to port from traditional OpenGL and OpenGL ES
The Power of a Three Layer Ecosystem

Applications can use Vulkan directly for maximum flexibility and control.

Application uses utility libraries to speed development.

Utility libraries and layers.

Application.

Games Engines fully optimized over Vulkan.

The industry’s leading games and engine vendors are participating in the Vulkan working group.

- Many utilities and layers will be in open source
- Layers to ease transition from OpenGL
- Domain specific flexibility

Developers can choose at which level to use the Vulkan Ecosystem.

A widely pervasive, powerful, flexible foundation layer enables diverse middleware tools and libraries. Likely to include vision libraries!
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
  - Driver upgrades will enable Vulkan on lots of current hardware
Compute API Summary

<table>
<thead>
<tr>
<th></th>
<th>OpenCL ES</th>
<th>OpenCL ES</th>
<th>Android</th>
<th>CUDA</th>
<th>Khronos</th>
<th>Khronos</th>
<th>Apple</th>
<th>Khronos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
<td>Khronos</td>
<td>Khronos</td>
<td>Google</td>
<td>NVIDIA</td>
<td>Khronos</td>
<td>Khronos</td>
<td>Apple</td>
<td>Khronos</td>
</tr>
<tr>
<td>Acceleration Devices</td>
<td>GPU</td>
<td>GPU</td>
<td>CPU + GPU</td>
<td>GPU</td>
<td>Heterogeneous</td>
<td>Heterogeneous</td>
<td>GPU</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>Scope</td>
<td>Graphics+Compute</td>
<td>Graphics+Compute</td>
<td>Compute</td>
<td>Compute</td>
<td>Compute</td>
<td>Compute</td>
<td>Graphics+Compute</td>
<td>Graphics+Compute</td>
</tr>
<tr>
<td>Explicit Memory/Execution</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Host Bindings</td>
<td>C</td>
<td>C</td>
<td>C/C++/Java</td>
<td>C/C++/Fortran</td>
<td>C/C++</td>
<td>C++</td>
<td>Objective-C/Swift</td>
<td>C/C++</td>
</tr>
<tr>
<td>Kernel Language</td>
<td>GLSL</td>
<td>GLSL</td>
<td>C</td>
<td>Single Source</td>
<td>C/C++/SPIR-V</td>
<td>Single Source</td>
<td>C++</td>
<td>SPIR-V</td>
</tr>
<tr>
<td>Availability</td>
<td>OpenGL ES 2.0+</td>
<td>OpenGL ES 3.1+</td>
<td>Android</td>
<td>NVIDIA GPU</td>
<td>Any OS</td>
<td>Any OS</td>
<td>iOS and MacOS</td>
<td>Any OS</td>
</tr>
</tbody>
</table>

Pervasive
Becoming Pervasive especially in mobile
Easy to use offload if don’t need explicit control
Mature toolchain for GPU Compute
Portable, low-level access to any heterogeneous compute resource
Low-barrier to entry, high-level C++ layer over OpenCL
‘Newgen’ API with integrated graphics and compute
Open, cross platform ‘newgen’ API with graphics and compute

Will adopt heterogeneous compute capabilities in the future?

Now to consider an actual vision framework...
OpenVX - Low Power Vision Acceleration

- Higher level abstraction API
  - Targeted at real-time mobile and embedded platforms
- Performance portability across diverse architectures
  - Multi-core CPUs, GPUs, DSPs and DSP arrays ISPs, Dedicated hardware...
- Extends portable vision acceleration to very low power domains
  - Doesn’t require high-power CPU/GPU Complex
  - Lower precision requirements than OpenCL
  - Low-power host can setup and manage frame-rate graph
OpenVX Graphs

- OpenVX developers express a graph of image operations (‘Nodes’)
  - Nodes can be on any hardware or processor coded in any language
  - E.g. on GPU nodes may implemented in OpenCL

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

![OpenVX Graph Diagram](image-url)
OpenVX Framework Efficiency..

**Graph Scheduling**
- Split the graph execution across the whole system: CPU / GPU / dedicated HW
- Faster execution or lower power consumption

**Memory Management**
- Reuse pre-allocated memory for multiple intermediate data
- Less allocation overhead, more memory for other applications

**Kernel Merge**
- Replace a sub-graph with a single faster node
- Better memory locality, less kernel launch overhead

**Data Tiling**
- Execute a sub-graph at tile granularity instead of image granularity
- Better use of data cache and local memory
Layered Vision Processing Ecosystem

- Lower-level compute APIs *can* be used to *implement* OpenVX nodes
  - Depending on the available processors
  - E.g. use OpenCL or OpenGL Compute Shaders

- Developers can then *use* OpenVX to easily connect those nodes into a graph
  - With portable performance

- High-level graph abstraction gives implementation flexibility
  - And significant optimization opportunities

---

**Application Software**

- OpenVX Runtime
- Flexible low-level APIs / languages
- Processor Hardware

---

**Application**

- **OpenVX**
- C/C++
- Programmable Vision Processors
- Dedicated Vision Hardware

---

© Copyright Khronos Group 2015 - Page 26
Example Relative Performance

- NVIDIA early implementation experience
  - Geometric mean of >2200 primitives, grouped into each categories, running at different image sizes and parameter settings

```
Example Relative Performance

<table>
<thead>
<tr>
<th>Category</th>
<th>OpenCV (GPU accelerated)</th>
<th>OpenVX (GPU accelerated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>1.1</td>
<td>2.5</td>
</tr>
<tr>
<td>Analysis</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>8.7</td>
<td></td>
</tr>
<tr>
<td>Geometric</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

© Copyright Khronos Group 2015 - Page 27
## OpenVX and OpenCV are Complementary

| Implementation | OpenCV | OpenVX
|----------------|--------|--------|
| Community driven open source library | Community driven open source library | Open standard API designed to be implemented by hardware vendors

**Conformance**
- Extensive OpenCV Test Suite but no formal Adopters program
- Implementations must pass defined conformance test suite to use trademark

**Consistency**
- Available functions can vary depending on implementation / platform
- All core functions must be available in all conformant implementations

**Scope**
- Very wide
- 1000s of imaging and vision functions
- Multiple camera APIs/interfaces
- Tight focus on core hardware accelerated functions for mobile vision - but extensible
- Uses external/native camera API

**Embedded Deployment**
- Re-usable code
- Callable library implemented and optimized by silicon vendors

**Efficiency**
- Memory-based architecture
- Each operation reads and writes to memory
- Graph-based execution
- Optimizable computation and data transfer

**Typical Use Case**
- Rapid experimentation and prototyping - especially on desktop
- Production development & deployment on wide range of mobile and embedded devices
OpenVX Status

• Finalized OpenVX 1.0 specification released October 2014
  - OpenVX 1.0.1 spec maintenance update released June 2015 [www.khronos.org/openvx](http://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](https://www.khronos.org/registry/vx/sample/openvx_sample_20141217.tar.gz)

• Full conformance test suite and Adopters Program available
  - Test suite exercises graph framework and functionality of each OpenVX 1.0 node

• Commercial conformant products
  - Intel, Imagination, NVIDIA, Synopsis, Vivante and many more coming...

• Roadmap discussions
  - More nodes, node profile sets, programmable nodes (SPIR-V?), Neural Net nodes
Vision Pipeline Challenges and Opportunities

Growing Camera Diversity

Diverse Vision Processors

Sensor Proliferation

Flexible sensor and camera control to GENERATE an image stream

Use efficient acceleration to PROCESS the image stream

Combine vision output with other sensor data on device
Safety Critical Working Group

Many future safety critical use cases involve vision and compute acceleration (e.g. neural nets)

New Khronos Safety Critical Advisory Panel
Defining guidelines for creating specifications for ISO 26262 and DO-178B/C certification
Roadmap Possibilities in Discussion

1. C++ Shading Language
2. Single source C++ Programming from SYCL
3. OpenCL-class Heterogeneous Compute to Vulkan runtime

SPIR-V Ingestion in OpenVX and OpenGL ES for programmable node and shading language flexibility

Thin and predictable graphics and compute for safety critical systems
Summary

- Khronos standards are key to many emerging embedded vision markets
  - GPU graphics and compute, parallel computation and vision
  - Khronos APIs are royalty-free

- Any company is welcome to join Khronos to influence the direction of these standards - and propose new initiatives
  - $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](http://www.khronos.org)
  - ntrevett@nvidia.com
  - @neilt3d