Navigating the Vision API Jungle:
Which API Should You Use and Why?
Embedded Vision Summit, May 2015

Neil Trevett | Khronos President
NVIDIA Vice President Mobile Ecosystem
Embedded Vision API Jungle

- Many APIs for embedded vision processing - industry confusion
- Overview and categorization to help developers decide which to use
- Latest updates and roadmaps for Khronos APIs

Use Acceleration Frameworks to:
1. Code vision functionality ‘from the ground up’ - including Neural Networks
2. Accelerate computer vision libraries and engines

Lets compare and contrast the available Acceleration Frameworks first...
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
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

---

<table>
<thead>
<tr>
<th>OpenGL ES</th>
<th>OpenGL ES Compute Shaders</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>Yes</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 3.1+</td>
</tr>
<tr>
<td>Precision</td>
<td>IEEE 754 Subset (highp) mediump / lowp</td>
</tr>
</tbody>
</table>
Google RenderScript

- Easy, high-level compute offload for Java and NDK Android apps
  - Design target is code portability across all Android devices

- C99-based kernel language for work offload to CPU and GPU
  - Each user script gets a Java “glue” class with accessors
  - JIT Compilation for host and device portability

- Built-in high-level Intrinsics library - the most popular feature today
  - Image processing (Blur, YuvToRGB, etc.)

- Transparent CPU fallback
  - No explicit control where code runs

- Android only
NVIDIA CUDA

- The industry’s original dedicated GPU Compute language
  - C/C++ language extensions for ‘single source’ programming
- Easy programmability and low level access to GPU
  - Unified Memory, Virtual Addressing, Dynamic Parallelism etc.
- Mature and optimized tools and compute / imaging libraries
  - NPP, cuFFT, cuBLAS, cuda-gdb, nvprof etc.
- NVIDIA only, GPU only, main focus is desktop/HPC
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 tree with CPU and accelerated paths

<table>
<thead>
<tr>
<th>OpenCL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
<td>Khronos</td>
</tr>
<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 - How it Works

- 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

OpenCL C++ Kernel Language
- Static subset of C++14
- Classes, lambda functions, templates, operator overloading..
- Fast and elegant sharable code - reusable device libraries and containers with PERFORMANCE PORTABILITY
- Template meta-programming for highly adaptive software
- Lambdas used for nested/dynamic parallelism

OpenCL 2.1 Provisional - March 2015
OpenCL C++ kernel language
SPIR-V in Core - no C++ front-end in driver
Runs on any OpenCL 2.0-capable hardware
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**
  - 2.0 | Jul14
  - 1.2 | May15

- **Mobile**
  - 2.0 | Dec14
  - 1.2 | May15

- **Embedded**
  - 1.2 | Sep13
  - 1.2 | Apr14

- **FPGA**
  - 1.0 | Dec14
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 / sensor processing key use case
- 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?
SYCL for OpenCL

- Single-source heterogeneous programming using **STANDARD C++**
  - Use C++ templates and lambda functions for host & device code

- Can use existing and new C++ compilers and layers over OpenCL run-time
  - Builds on C++11, with additional support for C++14
  - Will enable C++17 Parallel STL programs in the future

- SYCL 1.2 final spec released May 2015
  - Including open source triSYCL from AMD
  - [https://github.com/amd/triSYCL](https://github.com/amd/triSYCL)
OpenCL as Parallel Language Backend

Approaching 200 languages, frameworks and projects using OpenCL C 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

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

SPIR-V is a significant convergence point in the language ecosystem for graphics and parallel computation
Apple Metal

- First shipping ‘new generation graphics API’ with explicit GPU access
- Application constructs command buffers for graphics, compute and DMA
  - Mix and match of OpenGL and OpenCL functionality
- C++11-based kernel language - using AIR as intermediate language
- Higher-level memory allocation for ease-of-use
- Only Apple, only iOS 8, only Objective-C or Swift

<table>
<thead>
<tr>
<th>Governance</th>
<th>Apple</th>
</tr>
</thead>
<tbody>
<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>Yes</td>
</tr>
<tr>
<td>Host Bindings</td>
<td>Objective-C/Swift</td>
</tr>
<tr>
<td>Kernel Language</td>
<td>C++</td>
</tr>
<tr>
<td>Availability</td>
<td>iOS 8 / A7 Hardware</td>
</tr>
<tr>
<td>Precision</td>
<td>IEEE 754 Subset</td>
</tr>
</tbody>
</table>
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
  - Layered architecture so validation and debug layers unloaded when not needed

- Multi-threading friendly - create graphics, compute and DMA command buffers
  - General model - could be extended to heterogeneous processing in the future

<table>
<thead>
<tr>
<th>VULKAN</th>
<th>Vulkan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
<td>Khronos</td>
</tr>
<tr>
<td>Acceleration Devices</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>Scope</td>
<td>Graphics+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>SPIR-V</td>
</tr>
<tr>
<td>Availability</td>
<td>Any OS</td>
</tr>
<tr>
<td>Precision</td>
<td>Full IEEE 754 IEEE 754 Subset</td>
</tr>
</tbody>
</table>

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
Unified API for mobile, desktop, console and embedded platforms
Layered architecture so validation and debug layers unloaded when not needed

Multi-threading friendly - create graphics, compute and DMA command buffers
General model - could be extended to heterogeneous processing in the future
SPIR-V at the Center of Language Ecosystem

* Khronos considering developing open source implementations of these translators

- 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
- OpenCL C++
- GLSL
- Vulkan Driver X
- Vulkan Driver Y
- OpenCL Driver A
- OpenCL Driver B
- Other Languages
- Other Intermediate Forms
- LLVM

New kernel and shader Languages

- LLVM
- Other Languages

* Khronos considering developing open source implementations of these translators

* Khronos considering developing open source implementations of these translators
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 Windows XP/7/8
## Compute API Summary

<table>
<thead>
<tr>
<th>Governance</th>
<th>Khronos Shaders</th>
<th>Khronos Shaders</th>
<th>Google</th>
<th>NVIDIA</th>
<th>Khronos</th>
<th>Khronos</th>
<th>Apple</th>
<th>Khronos</th>
</tr>
</thead>
<tbody>
<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>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/C++/Java</td>
<td>C/C++/Fortran</td>
<td>C/C++</td>
<td>C++</td>
<td>Objective-C/Swift</td>
<td>C/C++</td>
<td>SPIR-V</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 8 / A7 Hardware</td>
<td>Any OS</td>
</tr>
<tr>
<td>Precision</td>
<td>NO IEEE 754 (highp) med/lowp</td>
<td>IEEE 754 Subset (highp) med/lowp</td>
<td>Full IEEE 754</td>
<td>Full IEEE 754</td>
<td>Full IEEE 754</td>
<td>Full IEEE 754</td>
<td>Full IEEE 754</td>
<td>Full IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Pervasive</td>
<td>Becoming Pervasive</td>
<td>Easy to use offload if don’t need explicit control</td>
<td>Mature toolchain for GPU Compute</td>
<td>Portable, low-level access to any heterogeneous compute resource</td>
<td>Low-barrier to entry, high-level C++ layer over OpenCL</td>
<td>‘Newgen’ API with integrated graphics and compute</td>
<td></td>
</tr>
</tbody>
</table>

**Now to consider vision libraries ...**

© Copyright Khronos Group 2015 - Page 18
OpenCV

- Extensive and widely used open source vision library
  - 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
OpenCV and OpenCL Cooperation

• OpenCL is providing good acceleration platform for OpenCV 3.0 on desktop
  - First wave of OpenCL implementations were on PCs - now mature

• But OpenCL on mobile/embedded still maturing
  - Even though OpenCL is widely available from mobile SOC vendors
  - OpenCL mobile/embedded performance portability can be challenging

• Khronos welcomes input from OpenCV community
  - OpenCL roadmap largely driven by embedded vision processing needs

• Vision processing will not always be able to afford to run the full CPU/GPU complex - we need a way to extend vision performance portability to the ultra-low-power domain -> OpenVX!
OpenVX - Power Efficient Vision Acceleration

- Extend possible vision platforms to very small hosts and hardware/DSPs
  - Targeted at mobile and embedded platforms, low-power, real-time applications
  - May not want or be able to support OpenCL

- Need higher-level abstraction API for implementation flexibility
  - And performance portability

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

- Can minimize host interaction during frame-rate graph execution
  - Small host processor can setup graph which then executes almost autonomously

OpenVX graph enables power and performance optimization

- 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

Image stream from native camera API

Example OpenVX Graph
Layered Vision Processing Ecosystem

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

- Coding in OpenCL can provide portability across heterogeneous processors
  - ISPs, Dedicated hardware, DSPs and DSP arrays, GPUs, Multi-core CPUs ...

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

<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></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmable Vision Processes</th>
<th>Dedicated Vision Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL</td>
<td>GLES</td>
</tr>
</tbody>
</table>

© Copyright Khronos Group 2015 - Page 22
OpenVX Status

- Finalized OpenVX 1.0 specification released October 2014
  - [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
  - Approved Conformant implementations can use the OpenVX trademark
OpenVX and OpenCV are Complementary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Community driven open source library</th>
<th>Open standard API designed to be implemented by hardware vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conformance</td>
<td>Extensive OpenCV Test Suite but no formal 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 core hardware accelerated functions for mobile vision - but extensible Uses external/native 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 and data transfer</td>
</tr>
<tr>
<td>Typical Use Case</td>
<td>Rapid experimentation and prototyping - especially on desktop</td>
<td>Production development &amp; deployment on mobile and embedded devices</td>
</tr>
<tr>
<td>Embedded Deployment</td>
<td>Re-usable code</td>
<td>Callable library</td>
</tr>
</tbody>
</table>
Wrap-up

• Many developers will choose an out-of-the-box vision library: OpenCV or OpenVX
  - Choice depends on emphasis: research/production and target device
  - Khronos seeking to synergize OpenCV and OpenVX - not fragment

• Some will dive deeper into compute APIs if using programmable processors
  - Accelerating vision libraries or rolling own vision or neural net functionality
  - OpenGL ES is pervasive, OpenCL for cross-platform heterogeneous compute
  - Watch for Vulkan and OpenCL evolutionary cross-pollination

• Acknowledgement to Maxim Shevtsov’s presentation at SIGGRAPH Asia

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