Accelerating Vision Processing

Neil Trevett
Vice President Mobile Ecosystem at NVIDIA
President of Khronos and Chair of the OpenCL Working Group
SIGGRAPH, July 2016
Khronos Open Standards

Khronos is an Industry Consortium of over 100 companies creating royalty-free, open standard APIs to enable software to access hardware acceleration for graphics, parallel compute and vision.
Accelerated Vision API Jungle

Vision Frameworks
Neural Net Libraries

cuDNN

OpenCV

clBLAS

OpenVX

Language-based
Acceleration Frameworks

Explicit Kernels

OpenGL ES

Vulkan

DirectX 12

OpenCL

GPU

FPGA

DSP

Dedicated Hardware
OpenGL ES

Fixed function Pipeline

Vertex and fragment shaders

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

Compute Shaders

Driver Update
Silicon Update
Silicon Update
Driver Update
Silicon Update

2003
1.0
2004
1.1
2007
2.0
2012
3.0
2014
3.1
2015
3.2

OpenGL ES 2.0

OpenGL ES 3.x

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

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=jRrG95GdaM

Vertex and Fragment Shaders

Compute Shaders
New Generation GPU APIs

- **DirectX 12**: Only Windows 10
- **Vulkan**: ‘Half Way New Gen’
  - Retains Traditional Binding Model
  - Mixes OpenGL ES 3.1/OpenCL 1.2
  - C++11-based kernel language
  - Objective-C or Swift
  - Cross Platform

Vulkan 1.0 launched in February
Shipping now on Windows, Linux, Android platforms from multiple vendors
Vulkan Explicit GPU Control

**High-level Driver Abstraction**
- Context management
- Memory allocation
- Full GLSL compiler
- Error detection
- Layered GPU Control

**GPU**

**Application**
- Single thread per context

**Thin Driver**
- Explicit GPU Control
- Memory allocation
- Thread management
- Synchronization
- Multi-threaded generation of command buffers

**Language Front-end Compilers**
- Initially GLSL
- SPIR-V pre-compiled shaders

**Vulkan Benefits**
- Resource management in app code:
  - Less hitches and surprises
- Simpler drivers:
  - Improved efficiency/performance
  - Reduced CPU bottlenecks
  - Lower latency
  - Increased portability
- Command Buffers:
  - Command creation can be multi-threaded
  - Multiple CPU cores increase performance
- Graphics, compute and DMA queues:
  - Work dispatch flexibility
- SPIR-V Pre-compiled Shaders:
  - No front-end compiler in driver
  - Future shading language flexibility
- Loadable Layers:
  - No error handling overhead in production code

**Vulkan 1.0** provides access to
OpenGL ES 3.1 / OpenGL 4.X-class GPU functionality
but with increased performance and flexibility
OpenCL

- Heterogeneous parallel programming of diverse compute resources
  - One code tree can be executed on CPUs, GPUs, DSPs and FPGA
- 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
- New in OpenCL 2.2 - OpenCL C++ kernel language is a static subset of C++14
  - Adaptable and elegant sharable code - great for building libraries
  - Templates enable meta-programming for highly adaptive software
  - Lambdas used to implement nested/dynamic parallelism
OpenCL 2.2 - Top to Bottom C++

**Single Source C++ Programming**
Full support for features in C++14-based Kernel Language

**API and Language Specs**
Brings C++14-based Kernel Language into core specification

**Portable Kernel Intermediate Language**
Support for C++14-based kernel language e.g. constructors/destructors

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++ Kernel Language
SPIR-V 1.1 with C++ support
SYCL 2.2 for single source C++

SPIR-V in Core
Subgroups into core
Subgroup query operations
cCloneKernel
Low-latency device
timer queries

<table>
<thead>
<tr>
<th>Date</th>
<th>Specification</th>
<th>Timeframe</th>
<th>Date</th>
<th>Specification</th>
<th>Timeframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dec08</td>
<td>OpenCL 1.0 Specification</td>
<td>18 months</td>
<td>Jun10</td>
<td>OpenCL 1.1 Specification</td>
<td>18 months</td>
</tr>
<tr>
<td>Nov11</td>
<td>OpenCL 1.2 Specification</td>
<td>24 months</td>
<td>Nov13</td>
<td>OpenCL 2.0 Specification</td>
<td>24 months</td>
</tr>
<tr>
<td>Nov15</td>
<td>OpenCL 2.1 Specification</td>
<td>7 months</td>
<td>May16</td>
<td>OpenCL 2.2 PROVISIONAL</td>
<td></td>
</tr>
</tbody>
</table>
SPIR-V Ecosystem

- Khronos has open sourced these tools and translators
- Khronos plans to open source these tools soon

SPIR-V
- Khronos defined and controlled cross-API intermediate language
- Native support for graphics and parallel constructs
  - 32-bit Word Stream
  - Extensible and easily parsed
  - Retains data object and control flow information for effective code generation and translation

Other Intermediate Forms

Third party kernel and shader Languages

OpenCL C

OpenCL C++

LLVM

New with OpenCL 2.2 and SPIR-V 1.1

https://github.com/KhronosGroup/SPIRV-Tools

SPIR-V (Dis)Assembler

SPIR-V Validator

IHV Driver Runtimes

GLSL

HLSL

LLVM to SPIR-V Bi-directional Translator

SPIR-V Magic: # 0x07230203
SPIR-V Version: 99
Builder’s Magic: # 0x051a000B
{<id> bounds: 50}
OpMemoryModel
Logical
GLSL450
OpEntryPoint
Fragment shader
function <id> 4
OpTypeVoid
id 2
OpTypeFunction
{<id> is 3}
return type <id> 2
OpFunction
Result Type <id> is 2
Result <id> is 4
Function Type <id> is 3

© Copyright Khronos Group 2016 - Page 10
SYCL for OpenCL

- Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
- Aligns the hardware acceleration of OpenCL with direction of the C++ standard
  - C++14 with open source C++17 Parallel STL hosted by Khronos

Developer Choice
The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language
Low Level Control
‘GPGPU’-style separation of device-side kernel source code and host code

Single-source C++
Programmer Familiarity
Approach also taken by C++ AMP and OpenMP
OpenCL Roadmap Discussions...

**Desktop**
Use cases: Video and Image Processing, Gaming Compute
Roadmap: Vulkan interop, arbitrary precision for increased performance, pre-emption, collective programming and improved execution model

**Vulkan Compute can leverage OpenCL?**
Gaming Compute, Pixel Processing, Inference
Fine grain graphics and compute (no interop needed)
SPIR-V for shading language flexibility - C/C++
Low-latency, fine grain run-time
Google Android adoption
Competes well with Metal (=C++/OpenCL 1.2)
Roadmap: types, precision and accuracy
Pointers and address spaces, execution model

**Mobile**
Use case: Photo and Vision Processing
Roadmap: arbitrary precision for inference engine and pixel processing efficiency, pre-emption and QoS scheduling for power efficiency

**HPC, SciViz, Datacenter**
Use case: Numerical Simulation, Virtualization
Roadmap: enhanced streaming processing, enhanced library support

**FPGAs**
Use cases: Network and Stream Processing
Roadmap: enhanced execution model, self-synchronized and self-scheduled graphs, fine-grained synchronization between kernels, DSL in C++

**Embedded**
Use case: Signal and Pixel Processing
Roadmap: arbitrary precision for inference engine and pixel processing efficiency, hard real-time scheduling, asynch DMA

**Possible learnings from Vulkan Philosophy**
1. Explicit - provide direct access to hardware capabilities with thin driver
2. Feature Sets - enable diverse architectures to ship just relevant features
3. Open source conformance tests for deep community engagement
OpenCV

• Extensive and widely used open source vision library - written in optimized C/C++
  - Free-use BSD license

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

• Increasingly taking advantage of heterogeneous processing using OpenCL
  - OpenCV 3.X 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!

OpenCV is active open source - not an API specification
A strength and a weakness!
Production deployment often needs tightly defined callable API
Vision Pipeline Challenges and Opportunities

Growing Camera Diversity
Flexible sensor and camera control to GENERATE an image stream

Diverse Vision Processors
Use efficient acceleration to PROCESS the image stream

Sensor Proliferation
Combine vision output with other sensor data on device
OpenVX - Low Power Vision Acceleration

- Precisely defined API for production deployment of vision acceleration
  - Targeted at real-time mobile and embedded platforms

- Higher abstraction than OpenCL for 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 or OpenCL precision
  - 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
- Graphs can execute almost autonomously
  - Possible to Minimize host interaction during frame-rate graph execution
- Graphs are the key to run-time optimization opportunities...

Feature Extraction Example Graph
OpenVX Efficiency through Graphs..

**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
Example Relative Performance

<table>
<thead>
<tr>
<th>Category</th>
<th>Relative Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>1.1</td>
</tr>
<tr>
<td>Analysis</td>
<td>2.9</td>
</tr>
<tr>
<td>Filter</td>
<td>8.7</td>
</tr>
<tr>
<td>Geometric</td>
<td>1.5</td>
</tr>
<tr>
<td>Overall</td>
<td>2.5</td>
</tr>
</tbody>
</table>

OpenCV (GPU accelerated)
OpenVX (GPU accelerated)

NVIDIA implementation experience. Geometric mean of >2200 primitives, grouped into each category, running at different image sizes and parameter settings.
Layered Vision Processing Ecosystem

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

And then developers can use OpenVX to enable a developer to easily *connect* those nodes into a graph.

OpenVX enables the graph to be *extended* to include hardware architectures that don’t support programmable APIs.

The OpenVX graph enables implementers to *optimize* execution across diverse hardware architectures and drive to lower power implementations.
OpenVX 1.0 Shipping, OpenVX 1.1 Released!

- Multiple OpenVX 1.0 Implementations shipping - spec in October 2014
  - Open source sample implementation and conformance tests available
- OpenVX 1.1 Specification released 2\textsuperscript{nd} May 2016 at Embedded Vision Summit
  - Expands node functionality AND enhances graph framework
  - Sample source and conformance tests will be updated to OpenVX 1.1 in 1H16
- OpenVX is EXTENSIBLE
  - Implementers can add their own nodes at any time to meet customer and market needs
What’s New in OpenVX 1.1?

- Expanded node functionality AND enhanced graph framework
  - Plus many minor improvements and clarifications

- Laplacian pyramids
  - Computational photography use cases

- Targets - for execution flexibility on heterogeneous devices
  - Application can control on which accelerator to run nodes

- Median, erode and dilate image filters
  - Including custom patterns

- Improved read and write data to and from OpenVX objects
  - Easier to use and less error prone

- Improved API for extending OpenVX with user kernels
  - More convenience and flexibility
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>
Safety Critical APIs

OpenGL SC 1.0 - 2005
Fixed function graphics subset

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

OpenGL ES 2.0 - 2007
Shader programmable pipeline

Experience and Guidelines

New Generation APIs for safety certifiable vision, graphics and compute
E.g. ISO 26262 and DO-178B/C

Small driver size
Advanced functionality
Graphics and compute
Safety Critical Advisory Panel

- Safety Critical Advisory Panel Announced Today!
  - Generating API design guidelines on how to best design acceleration APIs that ease system certifications
- Open to Khronos members and invited experts
  - https://www.khronos.org/openglsc/
OpenVX Roadmap Discussions

- Safety critical vision API - OpenVX SC?
- Significantly broaden node functionality
- In-graph neural nets
- Programmable nodes (OpenCL or SPIR-V?)
- Market-specific feature sets

- What do YOU need from OpenVX?
Get Involved!

- A diverse set of vision APIs in the industry
  - Developer choice is good - but need to choose wisely!

- Many APIs originally created to program GPUs
  - But vision processing needs are increasingly driving API roadmaps

- Industry will tend to consolidate around leading APIs
  - Working toward a multi-layer API ecosystem
  - Powerful foundational hardware APIs enabling rich middleware APIs and libraries

- Any company or organization is welcome to join Khronos for a voice and a vote in any of its standards
  - www.khronos.org

- Neil Trevett
  - ntrevett@nvidia.com
  - @neilt3d