New and Emerging Standards for Embedded Vision

Neil Trevett
Vice President Developer Ecosystem, NVIDIA | President, Khronos
ntrevett@nvidia.com | @neilt3d
Khronos is an International Industry Consortium of over 100 companies creating royalty-free, open standard APIs to enable software to access hardware acceleration for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks and Vision Processing.
Khronos Open Standards

3D for the Web
- VR/AR and games in-browser
- Efficiently delivering runtime 3D assets

Real-time 2D/3D
- Virtual and Augmented Reality displays
- Cross-platform gaming and UI
- CAD and Product Design

Vision, Neural Networks, VR/AR
- Tracking and odometry
- Scene analysis/understanding
- Neural Network inferencing
- VR/AR system portability

Parallel Computation
- Machine Learning acceleration
- Embedded vision processing
- High Performance Computing (HPC)
OpenVX - Efficient Vision Acceleration

- Vision acceleration for real-time, mobile and embedded platforms
  - High performance AND low power consumption are key

- Higher abstraction than OpenCL for performance portability across diverse architectures
  - Multi-core CPUs, GPUs, DSPs and DSP arrays, ISPs, FPGAs, Dedicated hardware...

- Extends portable vision acceleration to very low power domains
  - Doesn’t require high-power CPU/GPU Complex or OpenCL precision
OpenVX - Graph-Level Abstraction

- OpenVX developers express a graph of image operations (‘Nodes’)
  - Using a C API
- Nodes can be executed on any hardware or processor coded in any language
  - Implementers can optimize under the high-level graph abstraction
- Graphs are the key to run-time power and performance optimizations...

![OpenVX Graph](image-url)
OpenVX - Efficiency AND Rapid Development

- Graphs enable automatic optimizations
  - Especially tiling and kernel fusion
- Automatic selection of kernels
  - Optimized by data type at graph verification time
- User does NOT need to handle:
  - Hardware features: like scatter-gather, custom ISPs, hardware blocks
  - Custom kernels: intrinsics, assembly code
  - Custom data movement: tiling, local memory management, DMA, memory hierarchy
- The OpenVX framework does all of this for you

![Relative Performance Chart]

<table>
<thead>
<tr>
<th>Category</th>
<th>OpenCV (CPU)</th>
<th>OpenVX (GPU accelerated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>1.1</td>
<td>2.9</td>
</tr>
<tr>
<td>Analysis</td>
<td>2.9</td>
<td>8.7</td>
</tr>
<tr>
<td>Filter</td>
<td>8.7</td>
<td>15</td>
</tr>
<tr>
<td>Geometric</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>Overall</td>
<td>2.5</td>
<td>10</td>
</tr>
</tbody>
</table>

NVIDIA implementation experience. Geometric mean of >250 primitves, grouped into each category, running at different image sizes and parameter settings.
Simple Edge Detector in OpenVX

```c
vx_image input = vxCreateImage(1920, 1080);
vx_image output = vxCreateImage(0, 0);
vx_image horiz = vxCreateVirtualImage();
vx_image vert = vxCreateVirtualImage();
vx_image mag = vxCreateVirtualImage();

vx_graph g = vxCreateGraph();
vxSobel3x3Node(g, input, horiz, vert);
vxMagnitudeNode(g, horiz, vert, mag);
vxThresholdNode(g, mag, THRESH, output);

status = vxVerifyGraph(g);
status = vxProcessGraph(g);
```

 Declare Input and Output Images
 Declare Intermediate Images
 Construct the Graph
 Compile the Graph
 Execute the Graph
OpenVX Evolution

OpenVX 1.0
Spec released October 2014

OpenVX 1.1
Spec released May 2016

Conformant Implementations

AMD
Cadence
CEVA
Imagination
Texas Instruments
NVIDIA
SoCionext
Synopsys
VeriSilicon

New Functionality
- Expanded Nodes Functionality
- Enhanced Graph Framework

AMD OpenVX Tools
- Open source, highly optimized for x86 CPU and OpenCL for GPU
- “Graph Optimizer” looks at entire processing pipeline and removes, replaces, merges functions to improve performance and bandwidth
- Scripting for rapid prototyping, without re-compiling, at production performance levels
http://gpuopen.com/compute-product/amd-openvx/

OpenVX 1.2
Spec released May 2017

Extensions
- Neural Network Acceleration
- Graph Save and Restore
- 16-bit image operation

Safety Critical
OpenVX 1.1 SC for safety-certifiable systems

New Functionality
Under Discussion
- NNEF Import
- Programmable user kernels with accelerator offload

OpenVX Roadmap

New Functionality
- Conditional node execution
- Feature detection
- Classification operators
- Expanded imaging operations

OpenVX Evolution
Spec released October 2014
Conformant Implementations
New OpenVX 1.2 Functions

- **Feature detection**: find features useful for object detection and recognition
  - Histogram of gradients - HOG
  - Local binary patterns - LBP
- **Classification**: detect and recognize objects in an image based on a set of features
  - Import a classifier model trained offline
  - Classify objects based on a set of input features
- **Image Processing**: transform an image
  - Generalized nonlinear filter: Dilate, erode, median with arbitrary kernel shapes
  - Non maximum suppression: Find local maximum values in an image
  - Edge-preserving noise reduction
- **Conditional execution & node predication**: Selectively execute portions of a graph based on a true/false predicate
- Many, many minor improvements
- **New Extensions**
  - **Import/export**: compile a graph; save and run later
  - **16-bit support**: signed 16-bit image data
  - **Neural networks**: Layers are represented as OpenVX nodes

If A then S ← B else S ← C
Khronos NNEF (Neural Net Exchange Format)

- Range of Neural Network tools and inferencing architectures is rapidly increasing
- NNEF encapsulates neural network formal semantics
  - Structure, Data formats
  - Commonly used operations (such as convolution, pooling, normalization, etc.)
- Cross-vendor Neural Net file format removes industry friction
  - Simple exchange between tools and inferencing engines
  - Unified format for network optimizations

Every Tool Needs an Exporter to Every Accelerator
OpenVX 1.2 and Neural Net Extension

- Convolution Neural Network topologies can be represented as OpenVX graphs
  - Layers are represented as OpenVX nodes
  - Layers connected by multi-dimensional tensors objects
  - Layer types include convolution, activation, pooling, fully-connected, soft-max
  - CNN nodes can be mixed with traditional vision nodes

- Import/Export Extension
  - Efficient handling of network Weights/Biases or complete networks

- OpenVX will be able to import NNEF files into OpenVX Neural Nets

![An OpenVX graph mixing CNN nodes with traditional vision nodes](image-url)
Safety Critical APIs

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

Experience and Guidelines

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

OpenVX SC 1.1 Released 1st May 2017
Restricted “deployment” implementation executes on the target hardware by reading the binary format and executing the pre-compiled graphs

Khronos SCAP
‘Safety Critical Advisory Panel’ Guidelines for designing APIs that ease system certification. Open to Khronos member AND industry experts. If interested to join contact ntrevett@nvidia.com

Vulkan SC being discussed
Small driver size
Advanced functionality
Graphics and compute

© Copyright Khronos Group 2017 - Page 12
OpenVX SC - Safety Critical Vision Processing

- OpenVX 1.1 - based on OpenVX 1.1 main specification
  - Enhanced determinism
  - Specification identifies and numbers requirements
- MISRA C clean per KlocWorks v10
- Divides functionality into “development” and “deployment” feature sets
  - Adds requirement to support import/export extension

**OpenVX SC Development Feature Set (Create Graph)**
- Entire graph creation API

**Binary format**
- Implementation-dependent format

**OpenVX SC Deployment Feature Set (Execute Graph)**
- No graph creation API

- Verify Export
- Import
Layered Vision Processing Ecosystem

Implementers may use OpenCL 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 for lower power implementations.
OpenCL - Low-level Parallel Programming

- Low level programming of heterogeneous parallel compute resources
  - One code tree can be executed on CPUs, GPUs, DSPs and FPGA

- OpenCL C language to write kernel programs to execute on any compute device
  - Platform Layer API - to query, select and initialize compute devices
  - Runtime API - to build and execute kernels programs on multiple devices

- New in OpenCL 2.2 - OpenCL C++ kernel language - 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

### New Features
- 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

### Timeline
- **Dec08**: OpenCL 1.0 Specification
- **Jun10**: OpenCL 1.1 Specification
- **Nov11**: OpenCL 1.2 Specification
- **Nov13**: OpenCL 2.0 Specification
- **Nov15**: OpenCL 2.1 Specification
- **May16**: OpenCL 2.2 PROVISIONAL
Khronos SYCL - Single Source C++

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

- Kernel Fusion in C++ is a widely used compiler technique - proven to work
  - Halide, Eigen, Boost.Compute, ...
  - Optimization at the C++, not assembly, level
  - Achieves better performance on complex software than hand-coding

- Rapid optimization of multiple libraries - more information at http://sycl.tech
  - SYCLBLAS
  - SYCL Eigen
  - SYCL TensorFlow
  - SYCL GTX
  - triSYCL
  - ComputeCpp
  - VisionCpp
  - ComputeCpp SDK
Graph Programming - Fusion Results

- C++ Kernel fusion provides optimization benefits
  - Tiled operations in local memory
  - Reduced bandwidth to off-chip memory

Without fusion, each operation takes roughly the same amount of time on the accelerator (an AMD APU in this case) but the overhead varies a little.

OpenCV does not fuse in this case, but Halide and SYCL do. The fused kernels are significantly faster than non-fused when using C++ programming to achieve fusion.

Courtesy Codeplay:
https://www.slideshare.net/AndrewRichards28/open-standards-for-adas-andrew-richards-codeplay-at-autosens-2016-66476890
Convergence with Standard ISO C++

- SYCL Aligns the hardware acceleration of OpenCL with direction of the C++ standard
  - C++14 with open source C++17 Parallel STL hosted by Khronos

- Khronos working with others on bringing proposals to ISO C++ for:
  - Executors - for scheduling work
  - “Managed pointers” or “channels” - for sharing data

- Hoping to target C++ 20
  - But timescales are tight
OpenCL as Parallel Language/Library Backend

Caffe - C++ based Neural network framework
Halide - Language for image processing and computational photography
C++ AMP - MulticoreWare open source project on Bitbucket
SYCL - Single Source C++ Programming for OpenCL
aparapi - Java language extensions for parallelism
OpenCV - Vision processing open source project
OpenACC - Compiler directives for Fortran, C and C++
TensorFlow - Open source software library for machine learning

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

Low Level Explicit APIs

© Copyright Khronos Group 2017 - Page 20
Future Directions for OpenCL Roadmap

- Tuned precision requirements for vision and inferencing markets
  - Will enable significant numbers of DSP implementations to become conformant

- Converge with Vulkan - expanding Vulkan beyond graphics + more processor types
  - Thin, powerful, explicit run-time for control and predictability
  - Feature sets and dial-able precision for target market agility
  - Installable tools and three layer ecosystem for flexibility and backwards compatibility
  - Vulkan renderpasses are already a way to enabled tiled processing

<table>
<thead>
<tr>
<th>Vendor-supplied and open source middleware</th>
<th>Math Libraries</th>
<th>Language Front-ends</th>
<th>Tool Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Definitions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dial-able types and precision</td>
<td>Thin, explicit run-time with rigorous memory/execution model. Low-latency, fine-grain pre-emption and synchronization</td>
<td>Real-time Pre-emption and QoS scheduling</td>
<td>Explicit Asynch DMA</td>
</tr>
<tr>
<td>Installable tool &amp; validation layers</td>
<td>Features that can be enabled for particular target markets</td>
<td>Self-synchronized, self-scheduled graphs</td>
<td>Stream Processing</td>
</tr>
<tr>
<td>Applications</td>
<td>SYCL</td>
<td>SPIR</td>
<td>...</td>
</tr>
</tbody>
</table>

© Copyright Khronos Group 2017 - Page 21
SPIR-V Ecosystem

**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

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

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

Third party kernel and shader Languages

- HLSL
- GLSL

‘glslang’ GLSL to SPIR-V compiler

OpenCL C

OpenCL C++

LLVM to SPIR-V Bi-directional Translator

LLVM

SPIR-V (Dis)Assembler

SPIR-V Validator

Other Intermediate Forms

IHV Driver Runtimes

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

https://github.com/KhronosGroup/SPIRV-Tools
Possible Convergence of Graph Technologies

- API-created graphs such as OpenVX benefit from flexibility of user-programmed nodes
  - OpenVX Tiling extension lets them participate in tiled/fused optimizations
  - But currently user-programmed nodes can run only on the CPU
- Perhaps use a C++ based language to program user nodes?
  - That can be offloaded and scheduled with the OpenVX graph
  - Perhaps use SPIR-V to define node capabilities and store portable Node programs

An OpenVX user programmed node in a C++ domain specific language may have its executable stored as SPIR-V
Key Takeaways and What’s Next?

- **Vision Tools and APIs are becoming increasingly sophisticated**
  - Ecosystem is layering libraries, language and run-times

- **Graph-based solutions are key to efficient vision processing**
  - Enable significant optimizations - often automatically

- **Compiler technologies also becoming increasingly important**
  - To enable C++ and language-based solutions

- **Safety-critical APIs becoming essential for many markets**
  - Many vision applications need system certification

- **Still no cross-vendor camera APIs?**
  - Is the time yet right?

- **Please join if your company interested helping to drive Khronos open standards!**
  - ntrevett@nvidia.com | @neilt3d