Khronos Standards: Poweering the Future of Embedded Vision

Neil Trevett, Khronos President
NVIDIA VP Developer Ecosystems
ntrevett@nvidia.com | @neilt3d
Khronos Connects Software to Silicon

Open, royalty-free interoperability standards to harness the power of GPU, multiprocessor and XR hardware

3D graphics, augmented and virtual reality, parallel programming, inferencing and vision acceleration

Non-profit, member-driven standards organization, open to any company

Well-defined multi-company governance and IP Framework

Founded in 2000

>150 Members ~ 40% US, 30% Europe, 30% Asia
Khronos Active Initiatives

3D Graphics
Desktop, Mobile and Web

3D Assets
Authoring and Delivery

Portable XR
Augmented and Virtual Reality

Parallel Computation
Vision, Inferencing, Machine Learning

Safety Critical APIs

This work is licensed under a Creative Commons Attribution 4.0 International License
Khronos Compute Acceleration Standards

Higher-level Languages and APIs
- Streamlined development and performance portability

- Single source C++ programming with compute acceleration
- Graph-based vision and inferencing acceleration

Lower-level APIs
- Direct Hardware Control

- GPU rendering + compute acceleration
- Intermediate Representation (IR) supporting parallel execution and graphics
- Heterogeneous compute acceleration

Increasing industry interest in parallel compute acceleration to combat the ‘End of Moore’s Law’

SYCL and SPIR were originally OpenCL sub projects
Vulkan: An API for Direct GPU Control

Complex drivers cause overhead and inconsistent behavior across vendors
- Difficult to optimize
- Error handling is always active
- Full GLSL preprocessor and compiler in driver
- Different APIs for desktop/mobile

**Application**
- Single thread per context

**High-level Driver Abstraction**
- Error detection
- Context management
- Memory allocation
- Full GLSL compiler
- Synchronization

**GPU**
- A Graphics API

**Front-end Compiler(s)**
- GLSL, HLSL etc.

**Application**
- Memory allocation
- Thread management
- Explicit synchronization
- Multi-threaded generation of command buffers

**Loadable debug and validation layers**

**Thin Driver**
- SPIR-V compiler back end

**GPU**
- A GPU API

**Simpler drivers - application has the best knowledge for holistic optimization - no ‘driver magic’**
- Explicit creation of API objects before usage - efficient, predictable execution
- Easier portability - no fighting with different vendor heuristics
- Validation and debug layers loaded only when needed
- SPIR-V intermediate language: shading language flexibility
- Unified API across mobile, embedded and desktop platforms
- Multithread/Multicore friendly
APIs to Reduce Safety Certification Costs

- Industry safety standards and best practices
- API Design Guidelines to reduce system certification costs:
  - Reduce API surface area
  - Ensure deterministic behavior
  - Define robust error handling
  - C API - MISRA C Compliance
- Embedded Industry Requirements:
  - Decoupling software and hardware components
  - Cross-generation reusability
  - Cross-platform reusability
  - Field OTA upgradability
- Khronos APIs designed to reduce system safety certification costs:
  - Safety Critical variants: OpenGL SC and Vulkan SC
  - Safety Critical profiles: OpenVX 1.3 and OpenCL (potential)
- Open invitation to the industry to join at no cost
  - https://www.khronos.org/advisors/kscaf

This work is licensed under a Creative Commons Attribution 4.0 International License
Safety Critical GPU API Evolution

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Programmable Shaders

OpenGL SC 1.0 - 2005
Fixed function graphics safety critical subset

OpenGL SC 2.0 - 2016
Programmable Shaders safety critical subset

Vulkan 1.2 - 2020
Explicit Graphics and Compute

Vulkan is a Compelling Starting Point
- Widely adopted, royalty-free open standard
- Smaller surface area than OpenGL
- Not burdened by debug functionality
- Very little internal state
- Well-defined thread behavior
- Ingests SPIR-V IR - no runtime front-end compiler

Vulkan SC Goals and Design
- Reduce driver size and complexity
  -> Offline pipeline creation, no dynamic display resolutions
- Deterministic Behavior
  -> No ignored parameters or undefined behaviors
- Static memory management
- Robust Error Handling
  -> Enhanced fault handling callback
- C API - MISRA C Compliance

Industry Need
GPU Acceleration APIs designed to ease system safety certification e.g. to ISO 26262 / ASIL-D
OpenCL - Low-level Parallel Programming

Programming and Runtime Framework for Application Acceleration
Offload compute-intensive kernels onto parallel heterogeneous processors
CPUs, GPUs, DSPs, FPGAs, Tensor Processors
OpenCL C or C++ kernel languages

Platform Layer API
Query, select and initialize compute devices

Runtime API
Build and execute kernels programs on multiple devices

Explicit Application Control
Which programs execute on what device
Where data is stored in memories in the system
When programs are run, and what operations are dependent on earlier operations

Complements GPU-only APIs
Simpler programming model
Relatively lightweight run-time
More language flexibility, e.g., pointers
Rigorously defined numeric precision
OpenCL is Widely Deployed and Used

The industry's most pervasive, cross-vendor, open standard for low-level heterogeneous parallel programming

Desktop Creative Apps
- Adobe
- Sony
- Modo
- Autodesk
- CHADGROUP
- CyberLink
- NVIDIA
- Realflow
- ArcSoft
- LightWave
- SideFX
- Blackmagic Design
- Silhouette

Parallel Languages
- OpenACC
- SYCL
- OpenVINO
- Intel C++
- NVIDIA CUDA
- PyOpenCL

Linear Algebra Libraries
- SYCL-BLAS
- ViennaCL
- CLBlast

Machine Learning Libraries and Frameworks
- SYCL-DNN
- Caffe
- Acuity
- MetaWare EV
- TI DL Library (TIDL)

Accelerated Implementations
- Intel cIDNN
- NVIDIA cuDNN
- Xiaomi MiLib
- ARM MACE
- VeriSilicon
- Synopsys

Molecular Modelling Libraries
- CHARMM
- OpenMM
- GROMACS

Math and Physics Libraries
- ForceBalance
- Wolfram Mathematica

Vision, Imaging and Video Libraries
- VisionCpp
- Matplotlib
- Halide
- FFmpeg

Machine Learning Compilers
- TensorFlow
- OpenVINO
- OpenVX

Compiler Frameworks
- LLVM
- Clang
- OpenMP

Machine Learning Libraries and Frameworks
- MetaWare EV
- NNAPI

Mobile and Embedded
- Apple Metal
- ARM Mali
- NVIDIA CUDA
- AMD Radeon
- Intel Xeon Phi
- Xilinx Zynq

Cloud and Server
- Google TensorFlow
- Microsoft Cognitive Services
- IBM Watson

This work is licensed under a Creative Commons Attribution 4.0 International License

OpenCL 3.0

**Increased Ecosystem Flexibility**
All functionality beyond OpenCL 1.2 queryable plus macros for optional OpenCL C language features
New extensions that become widely adopted will be integrated into new OpenCL core specifications

**OpenCL C++ for OpenCL**
Open-source C++ for OpenCL front end compiler
combines OpenCL C and C++17 replacing
OpenCL C++ language specification

**Unified Specification**
All versions of OpenCL in one specification for easier maintenance, evolution and accessibility
Source on Khronos GitHub for community feedback, functionality requests and bug fixes

**Moving Applications to OpenCL 3.0**
OpenCL 1.2 applications - no change
OpenCL 2.X applications - no code changes if all used functionality is present
Queries recommended for future portability

**C++ for OpenCL**
Supported by Clang and uses the LLVM compiler infrastructure
OpenCL C code is valid and fully compatible
Supports most C++17 features
Generates SPIR-V kernels
Asynchronous DMA Extensions

OpenCL embraces a new class of Embedded Processors
Many DSP-like devices have Direct Memory Access hardware

Transfer data between global and local memories via DMA transactions
Transactions run asynchronously in parallel to device compute enabling wait for transactions to complete
Multiple transactions can be queued to run concurrently or in order via fences

OpenCL abstracts DMA capabilities via extended asynchronous workgroup copy built-ins
(New!) 2- and 3-dimensional async workgroup copy extensions support complex memory transfers
(New!) async workgroup fence built-in controls execution order of dependent transactions
New extensions complement the existing 1-dimensional async workgroup copy built-ins

Async Fence controls order of dependent transactions
All transactions prior to async_fence must complete before any new transaction starts, without a synchronous wait

Async 3D-3D Copy Transaction
Reshaping possible
$V_{global} = V_{local}$

The first of significant upcoming advances in OpenCL to enhance support for embedded processors
## ML Compiler Steps

**Import Formats**
- Caffe, Keras, MXNet, ONNX
- TensorFlow Graph, MXNet, PaddlePaddle, Keras, ONNX
- PyTorch, ONNX
- TensorFlow Graph, PyTorch, ONNX

**Front-end / IR**
- NNVM / Relay IR
- nGraph / Stripe IR
- Glow Core / Glow IR
- XLA HLO

**Output**
- OpenCL
- LLVM, CUDA, Metal
- OpenCL
- LLVM, CUDA
- OpenCL
- LLVM
- OpenCL
- LLVM, TPU IR, XLA IR
- TensorFlow Lite / NNAPI (inc. HW accel)

### Embedded NN Compilers
- CEVA Deep Neural Network (CDNN)
- Cadence Xtensa Neural Network Compiler (XNNC)

### Consistent Steps
1. Import Trained Network Description
2. Apply graph-level optimizations e.g., node fusion, node lowering and memory tiling
3. Decompose to primitive instructions and emit programs for accelerated run-times

### Fast progress but still area of intense research
If compiler optimizations are effective - hardware accelerator APIs can stay ‘simple’ and won’t need complex metacommands (e.g., combined primitive commands like DirectML)
SPIR-V enables a rich ecosystem of languages and compilers to target low-level APIs such as Vulkan and OpenCL, including deployment flexibility: e.g., running OpenCL C kernels on Vulkan.
SYCL Single Source C++ Parallel Programming

- **C++ Libraries**: Standard C++ Application Code
  - SYCL Compiler
  - Other Backends
  - CPU Compiler
  - CPU
  - Other Backends

- **SYCL Compiler**: C++ Template Libraries
  - SYCL BLAS
  - SYCL-Eigen
  - SYCL-DNN
  - SYCL Parallel STL

- **ML Frameworks**: Complex ML frameworks can be directly compiled and accelerated
  - C++ templates and lambda functions separate host & accelerated device code

- **CPU**: Accelerated code passed into device OpenCL compilers
  - Other Backends
  - CPU
  - GPU
  - FPGA
  - DSP
  - AI/Tensor HW
  - Custom Hardware

- **SYCL**: C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding

- **OpenCL**: SYCL is ideal for accelerating larger C++-based engines and applications with performance portability

This work is licensed under a Creative Commons Attribution 4.0 International License © The Khronos® Group Inc. 2021 - Page 14
SYCL 2020 Launched February 2021

Expressiveness and simplicity for heterogeneous programming in modern C++
Closer alignment and integration with ISO C++ to simplify porting of standard C++ applications
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Backend acceleration API independent

New Features
Unified Shared Memory | Parallel Reductions | Subgroup Operations | Class template Argument Deduction

Significant SYCL adoption in Embedded, Desktop and HPC Markets
SYCL Implementations in Development

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies.

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute.

**Source Code**

**Computing Platforms**

- DPC++: Uses LLVM/Clang, Part of oneAPI
  - Any CPU
  - CUDA+PTX: NVIDIA GPUs
  - OpenCL+SPIR-V: Intel CPUs, Intel GPUs, Intel FPGAs

- ComputeCpp: Multiple Backends
  - Any CPU
  - OpenCL+PTX: NVIDIA GPUs
  - OpenCL+: SPIR-V: Intel CPUs, Intel GPUs, Intel FPGAs, AMD GPUs (depends on driver stack), Arm Mali, IMG PowerVR, Renesas R-Car

- triSYCL: Open source test bed
  - OpenCL+: SPIR-LLVM: XILINX FPGAs, POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more)

- hipSYCL: CUDA and HIP/ROCm
  - OpenMP
  - CUDA: NVIDIA GPUs
  - ROCm: AMD GPUs
  - VEO: Intel CPUs, NEC VEs

- neoSYCL: SX-AURORA TSUBASA

Multiple Backends in Development
SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL e.g., ROCm and CUDA

For more information: [http://sycl.tech](http://sycl.tech)
The Origin of OpenVX

Engines and Applications

Driver Model

3D Graphics API Driver

An open API standard enables multiple silicon vendors to ship drivers with their silicon

Silicon vendors can aggressively optimize drivers for their own silicon architecture

OpenVX is the industry’s only API standard enabling portable access to vendor-optimized vision drivers

High-level Abstraction

3D graphics is always accelerated by a GPU - so a low-level GPU-centric API still provides cross-vendor portability

Vision processing can be accelerated by a wide variety of hardware architectures

OpenVX needs a higher-level graph abstraction to enable optimized cross-vendor drivers

Engines and Applications

OpenVX Vision API Driver

Vision Processing Graph

Vision Node

Vision Node

Vision Node

Vision Node

Vision Proc essing Graph
OpenVX Cross-Vendor Vision and Inferencing

High-level graph-based abstraction for portable, efficient vision processing
- Optimized OpenVX drivers created, optimized and shipped by processor vendors
- Implementable on almost any hardware or processor with performance portability
- Graph can contain vision processing and NN nodes for global optimization
- Run-time graph execution need very little host CPU interaction

Vision Node

Native Camera Control

NNEF Import converts a trained Neural Network into OpenVX Graph
- Layers are represented as OpenVX nodes

OpenVX Graph

Downstream Application Processing

Vendors optimize and ship drivers for their platform
- Full list of conformant OpenVX implementations here:
  https://www.khronos.org/conformance/adaptors/confornant-products/openvx

Open Source Convertors

ONNX
TensorFlow
TensorFlowLite
Caffe

Stable Specification

Open Source Projects

http://github.com/KhronosGroup/NNEF-Tools

This work is licensed under a Creative Commons Attribution 4.0 International License
OpenVX Efficiency through Graphs..

**Graph Scheduling**
- Split 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 Fusion**
- 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

Performance comparable to hand-optimized, non-portable code
- Real, complex applications on real-world hardware
- Much lower development effort and higher portability than hand-optimized code
OpenVX 1.3 and Extensibility

**OpenVX 1.3 core specification defines market-targeted feature sets**
- Baseline Graph Infrastructure (enables other Feature Sets)
  - Default Vision Functions
  - Enhanced Vision Functions
- Neural Network Inferencing (including tensor objects)
  - NNEF Kernel import (including tensor objects)
- Binary Images
- Safety Critical (reduced features and graph import for easier safety certification)

**OpenVX is Extensible**
Fully accelerated custom nodes can be integrated into the OpenVX graph with OpenCL interop

---

**OpenVX data objects**

**OpenVX**

**Runtime**

**OpenCL Command Queue**

**cl_mem buffers**

**OpenCL**

**Runtime**

**Runtime**

**Application**

OpenVX user-kernels can access command queue and cl_mem objects to asynchronously schedule OpenCL kernel execution.

Copy or export cl_mem buffers into OpenVX data objects.

Map or copy OpenVX data objects into cl_mem buffers.

Fully asynchronous host-device operations during data exchange.

---

This work is licensed under a Creative Commons Attribution 4.0 International License

© The Khronos® Group Inc. 2021 - Page 20
Open Source OpenVX & Samples

Fully Conformant
Open Source OpenVX 1.3
for Raspberry Pi
Raspberry Pi 3 and 4 Model B with Raspbian OS
Memory access optimization via tiling/chaining
Highly optimized kernels on multimedia instruction set
Automatic parallelization for multicore CPUs and GPUs
Automatic merging of common kernel sequences

Open Source OpenVX Tutorial and Code Samples
https://github.com/rgiduthuri/openvx_tutorial
https://github.com/KhronosGroup/openvx-samples

“Raspberry Pi is excited to bring the Khronos OpenVX 1.3 API to our line of single-board computers. Many of the most exciting commercial and hobbyist applications of our products involve computer vision, and we hope that the availability of OpenVX will help lower barriers to entry for newcomers to the field.”

Eben Upton
Chief Executive Raspberry Pi Trading
APIs for Embedded Compute

Networks trained on high-end desktop and cloud systems

Applications link to compiled inferencing code or call vision/inferencing API

Diverse Embedded Hardware
- Multi-core CPUs, GPUs
- DSPs, FPGAs, Tensor Cores
* Vulkan only runs on GPUs

Open industry standards, enable flexible integration and deployment of multiple acceleration technologies

Neural Network Training

Training Data

Compilation

Ingestion

Compiled Code

Vision / Inferencing Engine

C++ Application Code

Hardware Acceleration APIs

Sensor Data

GPU

FPGA

DSP

Dedicated Hardware

This work is licensed under a Creative Commons Attribution 4.0 International License
Need for Embedded Camera API Standards

Increasing Sensor Diversity
Including camera arrays and depth sensors such as Lidar

Cost and time to integrate and utilize sensors in embedded systems is a major constraint on innovation and efficiency in the embedded vision market

Multiple Sensors Per System
Synchronization and coordination become essential

Sophisticated Sensor Processing
Including inferencing. Sensor streams need to be efficiently generated and fed into acceleration APIs and processors

Proprietary Interfaces
Vendor-specific APIs to control cameras, sensors and close-to-sensor ISPs

Cost and time to integrate and utilize sensors in embedded systems is a major constraint on innovation and efficiency in the embedded vision market
Benefits of Embedded Camera API Standard

An effective open, cross-vendor open standard for camera, sensor and ISP control could provide multiple benefits:

- Cross-vendor portability of camera/sensor code for easier system integration of new sensors
- Preservation of application code across multiple generations of cameras and sensors
- Sophisticated control over sensor stream generation increases effectiveness of downstream accelerated processing

Development of Camera and sensor APIs may also generate new requirements for downstream vision and inferencing acceleration APIs.
Embedded Camera API Exploratory Group

Hosted by EMVA and Khronos

Online discussion forum and weekly Zoom calls, probably for a few months
Discuss industry requirements for open, royalty-free camera API(s)
No detailed design activity to protect participants IP
Explore if consensus can be built around an agreed Scope of Work document
Discuss what standardization activities can best execute actions in the Scope of Work

Any company is welcome to join
No cost or IP Licensing obligations
Project NDA to cover Exploratory Group Discussions

Proven Khronos Process to ensuring industry requirements are fully understood before starting standardization initiatives

Scope of Work Document
Agreed SOW document released from NDA and made public

Agreement with standardization bodies and/or open source projects on initiative(s) to execute the SOW under proven processes and IP Frameworks

Join and get involved!
https://www.khronos.org/embedded-camera/#getinvolved
Khronos for Global Industry Collaboration

Khronos membership is open to any company
Influence the design and direction of key open standards that will drive your business
Accelerate time-to-market with early access to specification drafts
Provide industry thought leadership and gain insights into industry trends and directions
Benefit from Adopter discounts

www.khronos.org