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 Landscape Overview

Vision Frameworks
Neural Net Libraries

cuDNN

Language-based
Acceleration Frameworks

Explicit Kernels

OpenCL

OpenVX Neural Net Extension

OpenVX

cBLAS

OpenCV

Sycl

NVIDIA
CUDA

DirectX 12

OpenGL ES

Vulkan

GPU

FPGA

DSP

Dedicated Hardware
### OpenGL ES

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>ES 1.0</td>
<td>Fixed function pipeline, vertex and fragment shaders</td>
</tr>
<tr>
<td>2004</td>
<td>ES 1.1</td>
<td>Driver update, silicon update</td>
</tr>
<tr>
<td>2007</td>
<td>ES 2.0</td>
<td>32-bit integers and floats, NPOT, 3D/depth textures, texture arrays, multiple render targets</td>
</tr>
<tr>
<td>2012</td>
<td>ES 3.0</td>
<td>Driver update</td>
</tr>
<tr>
<td>2014</td>
<td>ES 3.1</td>
<td>tessellation and geometry shaders, ASTC texture compression, floating point render targets</td>
</tr>
<tr>
<td>2015</td>
<td>ES 3.2</td>
<td>Driver update</td>
</tr>
<tr>
<td>2016</td>
<td>AEP</td>
<td>Multiple render targets, debug and robustness for security</td>
</tr>
</tbody>
</table>

---

**Epic's Rivalry demo** using full Unreal Engine 4: [https://www.youtube.com/watch?v=jRr-G95cd4M](https://www.youtube.com/watch?v=jRr-G95cd4M)

---

Close to 2 Billion OpenGL ES devices shipped in 2015

---

[http://hwstats.unity3d.com/mobile/gpu.html](http://hwstats.unity3d.com/mobile/gpu.html)
Three New Generation GPU APIs

- **DirectX 12**: Only Windows 10
- **Vulkan**: Cross Platform

Vulkan is extensible – just like OpenGL - and so is the new only generation API where hardware vendors can deliver innovations to market whenever they need.

- **‘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
- **Only Apple**
Vulkan Explicit GPU Control

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

Vulkan Benefits

- Resource management in app code: Less driver hitches and surprises
- Simpler drivers: Improved efficiency/performance, Reduced CPU bottlenecks, Lower latency, Increased portability
- Multi-threaded 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
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 Diagram]

- Kernel code compiled for devices
- GPU
- DSP
- CPU
- FPGA
- Devices
- Runtime API loads and executes kernels across devices
- CPU
- Host
# 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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-component vectors</td>
<td>Additional image formats</td>
</tr>
<tr>
<td>Multiple hosts and devices</td>
<td>Buffer region operations</td>
</tr>
<tr>
<td>Enhanced event-driven execution</td>
<td>Additional OpenCL C built-ins</td>
</tr>
<tr>
<td>Improved OpenGL data/event interop</td>
<td></td>
</tr>
<tr>
<td>Device partitioning</td>
<td>Separate compilation and linking</td>
</tr>
<tr>
<td>Enhanced image support</td>
<td>Built-in kernels / custom devices</td>
</tr>
<tr>
<td>Enhanced DX and OpenGL Interop</td>
<td></td>
</tr>
<tr>
<td>Shared Virtual Memory</td>
<td>On-device dispatch</td>
</tr>
<tr>
<td>Generic Address Space</td>
<td>Enhanced Image Support</td>
</tr>
<tr>
<td>C++11 Atomics</td>
<td>Pipes</td>
</tr>
<tr>
<td>Android ICD</td>
<td></td>
</tr>
</tbody>
</table>

## Timeline

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Duration</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL 1.0</td>
<td>Dec08</td>
<td>18 months</td>
<td></td>
</tr>
<tr>
<td>OpenCL 1.1</td>
<td>Jun10</td>
<td>18 months</td>
<td></td>
</tr>
<tr>
<td>OpenCL 1.2</td>
<td>Nov11</td>
<td>24 months</td>
<td></td>
</tr>
<tr>
<td>OpenCL 2.0</td>
<td>Nov13</td>
<td>24 months</td>
<td></td>
</tr>
<tr>
<td>OpenCL 2.1</td>
<td>Nov15</td>
<td>7 months</td>
<td></td>
</tr>
<tr>
<td>OpenCL 2.2</td>
<td>May16</td>
<td>7 months</td>
<td></td>
</tr>
</tbody>
</table>

- **OpenCL C++ Kernel Language**: SPIR-V 1.1 with C++ support
- **SYCL 2.2**: for single source C++
SPIR-V Ecosystem

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

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

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

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

SPIR-V (Dis)Assembler

SPIR-V Validator

Other Intermediate Forms

LLVM

ARB_gl_spirv extension

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
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...

- 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

Applications

Vendor-supplied and open source middleware

- Math Libraries
- Language Front-ends
- Tool Layers

Dial-able types and precision

Installable tool & validation layers

Thin, explicit run-time with rigorous memory/execution model. Low-latency, fine-grain pre-emption and synchronization

Real-time Pre-emption and QoS scheduling
Explicit Asynch DMA
Self-synchronized, self-scheduled graphs
Stream Processing...

Features that can be enabled for particular target markets
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...

OpenVX Graphs Diagram

- Color Conversion
- Channel Extract
- Image Pyramid
- Optical Flow
- Harris Track
- Array of Keypoints
- Array of Features
- RGB Frame
- YUV Frame
- Gray Frame

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

Relative Performance

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
<th>NVIDIA implementation experience. Geometric mean of &gt;2200 primitives, grouped into each categories, running at different image sizes and parameter settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>1.1</td>
<td>OpenCV (GPU accelerated)</td>
</tr>
<tr>
<td>Analysis</td>
<td>2.9</td>
<td>OpenVX (GPU accelerated)</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>2.5</td>
<td></td>
</tr>
</tbody>
</table>
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.

AMD OpenVX
- 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 1.0 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 2nd May 2016 at Embedded Vision Summit
  - Expands node functionality AND enhances graph framework
- Roadmap Discussions
  - More nodes, programmable nodes (OpenCL or SPIR-V)
  - Feature Sets to enable market-targeting without fragmentation
- OpenVX is EXTENSIBLE
  - Implementers can add their own nodes at any time to meet customer and market needs
OpenVX 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

- The specification is provisional
  - Welcome feedback from the deep learning community
Safety Critical APIs

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

OpenGL SC 1.0 - 2005
Fixed function graphics

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Shader programmable pipeline subset

Khronos Launches Safety Critical Advisory Panel - and Invites Industry Experts to Participate

Experienced practitioners in the field of safety critical system design are invited to apply for Advisory Panel membership simply by sending an email to khronos_scap_apply@Khronos.org. Please include your contact information, a short history of your experience along with why you feel you could help us set the future direction of safety critical APIs.

Read the press release


Small driver size
Safety Critical heterogeneous compute
Advanced functionality
Graphics and compute
Thanks - and Please 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 embedded 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