Vision Processing BOF
OpenVX Overview...
...and the Vision API Jungle

Neil Trevett
Vice President Mobile Ecosystem at NVIDIA | President of Khronos
SIGGRAPH, August 2015
Khronos Connects Software to Silicon

Open Consortium creating ROYALTY-FREE, OPEN STANDARD APIs for hardware acceleration

Defining the roadmap for low-level silicon interfaces needed on every platform

Graphics, compute and vision processing

Rigorous specifications AND conformance tests for cross-vendor portability

Acceleration APIs BY the Industry FOR the Industry

Well over a BILLION people use Khronos APIs Every Day…
Visual Computing = Graphics AND Vision

Virtual glass

High-Quality Reflections, Refractions, and Caustics in Augmented Reality and their Contribution to Visual Coherence

P. Kán, H. Kaufmann, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, Austria

https://www.youtube.com/watch?v=i2MEwVZzDaA

New mobile visual sensors for MORE DATA

Advanced mobile hardware for MORE PROCESSING

Enables closer intertwining of real and virtual worlds

Real time demo on GPU-enabled laptop

© Copyright Khronos Group 2015 - Page 3
Mobile Vision Acceleration = New Experiences

Need for advanced sensors and the acceleration to process them

- Computational Photography and Videography
- Face, Body and Gesture Tracking
- 3D Scene/Object Reconstruction
- Augmented Reality
Embedded Vision API Jungle

- There are programming frameworks dedicated to classic vision processing
  - Widely used - many aimed at different use cases

- But general compute acceleration APIs can be used for vision processing
  - When should use general programmable frameworks instead of libraries?

- Lets compare and contrast three frameworks
  - How do they relate to each other?
OpenCL - Portable Heterogeneous Computing

- Portable Heterogeneous programming of diverse compute resources
  - Targeting supercomputers -> embedded systems -> mobile devices

- One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware
  - Dynamically interrogate system load and balance work across available processors

- OpenCL = Two APIs and Kernel language
  - C Platform Layer API to query, select and initialize compute devices
  - C Runtime API to build and execute kernels across multiple devices

- Use OpenCL to:
  - Code vision functionality ‘from the ground up’
  - Accelerate compute tasks such as Neural Networks
OpenCV

- Extensive and widely used open source vision library - 1,000s of functions
  - 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!

OpenCV Application

- CPU Thread
- CPU Thread
- CPU Thread

ocl::Queue
ocl::Queue
ocl::Queue

ocl::Device
ocl::Device
ocl::Device

ocl::Context

- 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!
Vision Processing Power Efficiency

- E.g. AR wearables will need ‘always-on’ vision
  - With smaller thermal limit / battery than phones!
- GPUs have x10 imaging power efficiency over CPU
  - GPUs architected for efficient pixel handling
- Dedicated Hardware/DSPs can be even more efficient
  - With some loss of generality
- Mobile SOCs have space for more transistors
  - But can’t turn on at same time = Dark Silicon
  - Can integrate more gates ‘for free’ if careful
  how and when they are used

Potential for dedicated sensor/vision silicon to be integrated into Mobile Processors
But how will they be programmed for PORTABILITY and POWER EFFICIENCY?
OpenVX - Vision Acceleration

• Royalty-free open standard API
  - Reliably accelerated by hardware vendors
  - Tightly defined conformance tests
• Targeted at low-power, real-time applications
  - Mobile and embedded platforms
• Portability across diverse heterogeneous processors
  - ISPs, Dedicated hardware, DSPs and DSP arrays, GPUs, Multi-core CPUs ...
• Doesn’t require high-power CPU/GPU Complex
  - Low-power host can setup and manage frame-rate vision processing pipeline

OpenVX extends easily re-usable vision acceleration to very low power domains
OpenVX Graphs - The Key to Efficiency

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

- Graph enables implementations to optimize for power and performance
  - 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

- Minimizes host interaction during frame-rate graph execution
  - Host processor can setup graph which can then execute almost autonomously
# OpenVX and OpenCV are Complementary

<table>
<thead>
<tr>
<th></th>
<th>OpenCV</th>
<th>OpenVX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation</td>
<td>Community driven open source library</td>
<td>Open standard API designed to be implemented by hardware vendors</td>
</tr>
<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</td>
<td>Tight focus on core hardware accelerated functions for mobile vision - but extensible</td>
</tr>
<tr>
<td></td>
<td>Multiple camera APIs/interfaces</td>
<td>Uses external/native camera API</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Memory-based architecture</td>
<td>Graph-based execution</td>
</tr>
<tr>
<td></td>
<td>Each operation reads and writes to memory</td>
<td>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>
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
OpenVX and OpenCL are Complementary

<table>
<thead>
<tr>
<th>Use Case</th>
<th>General Heterogeneous programming</th>
<th>Domain targeted vision processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of Use</strong></td>
<td>General-purpose math libraries with no built-in vision functions</td>
<td>Fully implemented vision operators and framework ‘out of the box’</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Language-based - needs runtime compilation</td>
<td>Library-based - no online compiler required</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>Full IEEE floating point mandated</td>
<td>Requires minimal floating point or fixed point-optimized for vision operators</td>
</tr>
<tr>
<td><strong>Target Hardware</strong></td>
<td>‘Exposed’ architected memory model which can impact performance portability</td>
<td>Abstracted node and memory model - diverse implementations can be optimized for power and performance</td>
</tr>
</tbody>
</table>
OpenVX Status

- Finalized OpenVX 1.0 specification released October 2014
  - OpenVX 1.0.1 spec maintenance update released June 2015  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

- Full conformance test suite and Adopters Program available
  - Test suite exercises graph framework and functionality of each OpenVX 1.0 node

- Commercial conformant products
  - NVIDIA, Synopsis, Vivante
Speakers

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Neil Trevett</td>
<td>Khronos President</td>
<td>NVIDIA</td>
<td>The Vision API Jungle</td>
</tr>
<tr>
<td>Andrew Garrard</td>
<td>Senior Software Engineer</td>
<td>Samsung</td>
<td>Khronos Data Format for Vision Processing</td>
</tr>
<tr>
<td>Radha Giduthuri</td>
<td>Design Engineer</td>
<td>AMD</td>
<td>OpenVX Technical Overview</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>Wrap-up and Questions</td>
</tr>
</tbody>
</table>
Should you ignore me?

- Are you writing a program?
- Does it work with images, textures, buffers, etc?
- Do you have that content in memory?
- Do you need to describe that content to anyone else?
- Do you have more than one type of content?
- Do you need to describe how hardware or software handles this stuff?
This is about formats

- You’ve got some bits that correspond to a pixel, or buffer element
- What do they mean?
- “Oh, RGB, obviously…”
When you said that...

- *What* RGB?

**32bpp red, green, blue, alpha 8888**

```
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
```

**16bpp red, green, blue 565**

```
0 1 2 3 4 0 1 2 3 4 5 0 1 2 3 4
```

**16bpp red, green, blue, alpha 4444**

```
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
```
When you said that...

- **What RGB?**

  32bpp red, green, blue, alpha 8888
  
  
  16bpp red, green, blue 565
  
  16bpp red, green, blue, alpha 4444

- **What order? (Which API’s convention?)**

  32bpp blue, green, red, alpha 8888

  16bpp red, green, blue 565, swapped endian
This is when...
Ow. And then...

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”
Ow. And then...

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”
- ...range?
Ow. And then...

- Now we know which bit is which channel
- But what do the channels mean?
- “The obvious colors”
- ...range?
- ...gamma?
- sRGB?
- TV output?
- Which TV output?
And if you guess...
Then you want to output to video

- Ooh, YUV
Then you want to output to video

- Ooh, YUV
- ...YUV
- ...or used compressed formats
And read from a camera

- Bayer!
- Metadata
- Etc...
When you do have this right...

- You end up with a library that gets confused and data disappears
Around this time...
Nothing does this right...

- Describing a format is really easy when you know your problem space
- At some point, you’re using something that doesn’t
- Everyone rolls their own
- No problems until they have to work together
- Problem spaces aren’t as disjoint as you’d think
The Khronos Data Format Specification

- Just released
- A really dull thing done right so you don’t have to think about it
- Descriptive
- Extensible
- Versioned
- Flexible
- Not big
- No conformance
- Not tied to any other Khronos spec
- The press don’t understand what it is

www.khronos.org/dataformat/
Fellow sufferers...

Acute qwertyitis: keyboard rash from impact damage
c.f. chronic qwertyitis, caused by keyboard-as-pillow
Thank you
Technical Overview
August 2015

Radhakrishna Giduthuri
Design Engineer, AMD
OpenVX - Design Overview

- **Software Landscape**
  - direct use by applications
  - acceleration layer for higher-level vision frameworks, engines or platform APIs

- **Design Objectives**
  - framework of standard computer vision functions to run on wide variety of platforms
    - abstraction for commonly-used vision functions
    - abstraction for aggregations of functions
  - provide implementer the opportunity to accelerate and minimize the run-time overhead
OpenVX - Function Overview

- Core data structures
  - Images and Image Pyramids
  - Processing Graphs, Kernels, Parameters

- Image Processing
  - Arithmetic, Logical, and statistical operations
  - Multichannel Color and BitDepth Extraction and Conversion
  - 2D Filtering and Morphological operations
  - Image Resizing and Warping

- Core Computer Vision
  - Pyramid computation
  - Integral Image computation

- Feature Extraction and Tracking
  - Histogram Computation and Equalization
  - Canny Edge Detection
  - Harris and FAST Corner detection
  - Sparse Optical Flow

OpenVX specification is extensible. Khronos maintains an extension registry.

OpenVX 1.0 defines a framework for creating, managing, and executing graphs.

Focused set of widely used functions that are readily accelerated.

Implementers can add functions as extensions.

Widely used extensions are adopted into future versions of the core.
OpenVX Example: Key-point Detector

OpenVX Graph

RGB Camera

Display Results

OpenVX Node

OpenVX Data Object

org.khronos.openvx.color_convert

org.khronos.openvx.channel_extract

org.khronos.openvx.harris_corners

VX_DF_IMAGE_RGB

VX_DF_IMAGE_IYUV

VX_DF_IMAGE_U8

VX_CHANNEL_Y

configuration

parameters

VX_TYPE_KEYPOINT

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX

OpenVX - Objects

• Object-Oriented Behavior
  - strongly typed at compile-time for safety critical applications (typedef’d type)
  - strongly typed at run-time for dynamic applications (vx_type_e enumerations)
  - any object may be down-cast to a vx_reference

• OpenVX Data Objects (opaque)
  - vx_image, vx_array, vx_pyramid, vx_scalar, vx_threshold, vx_convolution, vx_matrix, vx_distribution, vx_lut, vx_delay, vx_remap

• OpenVX Framework Objects
  - vx_context - object domain for all OpenVX objects
  - vx_kernel - abstract representation of a computer vision, such as, Harris Corners
  - vx_graph - a set of vision function nodes connected in a directed acyclic fashion
  - vx_node - an instance of a kernel paired with a specific set of references
  - vx_parameter - abstract input, output, or bidirectional object passed to a node and contains information about parameter usage
OpenVX - Data Objects

**vx_image**

- Image formats:
  - VX_DF_FORMAT_U8/S16/.../U32,
  - VX_DF_FORMAT_IYUV/NV12/...,
  - VX_DF_FORMAT_RGB/RGBX,
  - ...

**vx_pyramid**

- Array element data types:
  - vx_keypoint_t,
  - vx_rectangle_t,
  - vx_coordinates2d_t,
  - vx_coordinates3d_t,
  - user defined structs...

**vx_array**

- Array element data types:
  - vx_keypoint_t,
  - vx_rectangle_t,
  - vx_coordinates2d_t,
  - vx_coordinates3d_t,
  - user defined structs...

**vx_scalar**

- UINT8
- INT16
- INT32
- UINT32
- FLOAT32
- ...

**vx_delay**

- Delay container types:
  - vx_image, vx_pyramid, vx_array, vx_scalar,
  - vx_matrix, vx_distribution, vx_remap, vx_lut, vx_threshold
OpenVX - Framework

- Context
  - Data Objects
  - Kernels
- Graphs
  - Virtual Data Objects
  - Nodes
- Extensions
OpenVX - Graph Concepts

- Graphs are composed of one or more nodes that are added to the graph through node creation functions and nodes are linked together via data dependencies
  - only a single writer of an object may exist in a given graph
  - all writers shall produce output data before any reader of that data accesses it

- Graphs must be created ahead of processing time and verified by the implementation, after which they can be processed as many times as needed
  - maximize its optimization potential because all the operations of the graph and its dependencies are known ahead of time

- Virtual Data Objects give flexibility and enable vendors the ability to optimize
  - Scoped within the graph and may be declared to have no dimensions or format
  - Inaccessible through access/commit APIs

- Execution in synchronous block mode as well as asynchronous mode

- Node execution independence
OpenVX - Graph Life Cycle

• Construction
  - Create graph and connect nodes together by data objects:
    vxCreateGraph() and node creation API

• Verification
  - Check for consistency, correctness, and other conditions:
    vxVerifyGraph()

• Execution
  - Synchronous blocking mode:
    vxProcessGraph()
  - Asynchronous non-blocking mode:
    vxScheduleGraph() and vxWaitGraph()

• Deconstruction
  - All the nodes in the graph are released:
    vxReleaseGraph()
OpenVX - Graph Example

```c
vx_context context = vxCreateContext();
vx_image images[] = {
    vxCreateImage(context, 640, 480, VX_DF_IMAGE_UYVY),
    vxCreateImage(context, 640, 480, VX_DF_IMAGE_S16),
    vxCreateImage(context, 640, 480, VX_DF_IMAGE_U8),
};
vx_graph graph = vxCreateGraph(context);
vx_image virts[] = {
    vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
    vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
    vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
    vxCreateVirtualImage(graph, 0, 0, VX_DF_IMAGE_VIRT),
};
vxChannelExtractNode(graph, images[0], VX_CHANNEL_Y, virts[0]),
vxGaussian3x3Node(graph, virts[0], virts[1]),
vxSobel3x3Node(graph, virts[1], virts[2], virts[3]),
vxMagnitudeNode(graph, virts[2], virts[3], images[1]),
vxPhaseNode(graph, virts[2], virts[3], images[2]),
status = vxVerifyGraph(graph);
if (status == VX_SUCCESS) status = vxProcessGraph(graph);
vxReleaseContext(&context); /* this will release everything */
```
OpenVX - Key-point Detector Example

#include <VX/vx.h>

// create OpenVX context: object domain for all OpenVX objects
vx_context context = vxCreateContext();

// Create data objects...
vx_image frameRGB = vxCreateImage(context, width_, height_, VX_DF_IMAGE_RGB);
vx_image frameYUV = vxCreateImage(context, width_, height_, VX_DF_IMAGE_IYUV);
vx_image frameGray = vxCreateImage(context, width_, height_, VX_DF_IMAGE_U8);
vx_array corners = vxCreateArray(context, VX_TYPE_KEYPOINT, 2000);

// create and verify the graph...
vx_graph graph = vxCreateGraph(context);
vxColorConvertNode(graph, frameRGB, frameYUV);
vxChannelExtractNode(graph, frameYUV, VX_CHANNEL_Y, frameGray);
vxHarrisCornersNode(graph, frameGray, strength_thresh, min_distance, k_sensitivity, gradient_size, block_size, corners, NULL);
if (vxVerifyGraph(graph) != VX_SUCCESS) { ... error ... }

// execute graph
while(...there is input...) {
    ... copy input RGB frame from camera into ‘frameRGB’ using vxAccessImagePatch() and vxCommitImagePatch() ...
    vxProcessGraph(graph);
    ... copy results from ‘corners’ using vxAccessArrayRange() and vxCommitArrayRange() ...
}

// release context
vxReleaseContext(&context);
OpenVX Graphs - The Key to Efficiency

- Vision processing directed graphs for power and performance efficiency
  - Each Node can be implemented in software or accelerated hardware
  - Nodes may be fused by the implementation to eliminate memory transfers
  - Processing can be tiled to keep data entirely in local memory/cache

- Minimizes host interaction during graph execution
  - Host processor can setup graph which can then execute almost autonomously

- EGLStreams can provide data and event interop with other Khronos APIs
  - BUT use of other Khronos APIs are not mandated

Example OpenVX Graph
OpenVX - Other Key Features

• Image special cases
  - Uniform images: uniform value in all pixels
  - Image ROI: rectangular region of another image
  - Images with externally allocated buffers

• Callbacks

• Hints and Directives

• User Kernels

• Extending OpenVX

• VXU Utility Library for access to single nodes
  - Easy way to start using OpenVX by calling each node independently
OpenVX 1.0 extensions

- Tiling extension: more efficient processing of graphs with user nodes
  - Provisional spec released
- XML Schema extension: cross-platform graph saving and loading
  - Provisional spec released
- S16 extension: wider support for S16 data type across OpenVX vision kernels
  - Provisional spec released
OpenVX - Summary

• Features functional and performance portability
  - Tightly defined specification
  - Full conformance tests
• Out-of-the-Box power efficient vision acceleration frameworks
• Performance portability across diverse HW
  - Higher-level abstraction hides hardware details
    - ISPs, Dedicated hardware, DSPs and DSP arrays, GPUs, Multi-core CPUs ...
• Extendable API
Questions and Call to Action

• These slides will be posted online later today
  - www.khronos.org

• Khronos welcomes feedback on OpenVX open forum

• Any company or organization is welcome to join Khronos
  - For a voice and a vote in this important emerging standard

• More Information
  - www.khronos.org
  - ntrevett@nvidia.com
  - @neilt3d
Another Example: Feature tracking Graph

- **Image capture API**
- **Display API**

**RGB frame**
- **color convert**
- **frameYUV**
- **channel extract**
- **frameGray**
- **pyramid**
- **pyr_delay**
- **ptr_delay**
- **pyr -1**
- **pyr 0**
- **pts -1**
- **pts 0**
- **optical flow pyrLK**

**Array of keypoints**
Yet Another Example - Stereo Machine Vision

Tiling extension enables user nodes (extensions) to also optimally run in local memory