Using OpenCL for Performance-Portable, Hardware-Agnostic, Cross-Platform Video Processing

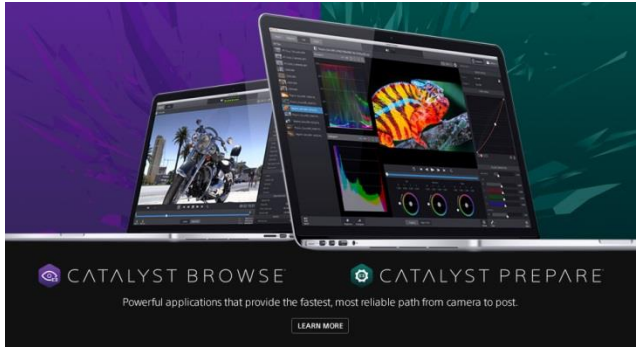
GTC 2015 S5592

Dennis Adams, Director of Technology

Sony Creative Software Inc.

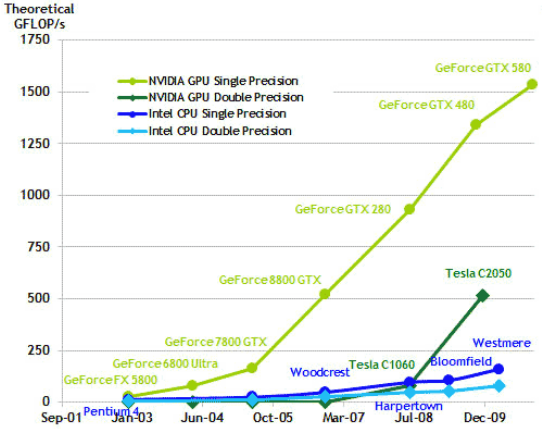
What we make

- Sony Creative Software makes digital content creation tools
  - Audio & video editing
  - Music creation
  - Media preparation
- GPU accelerated
  - Vegas Pro & Movie Studio
  - Catalyst Browse & Prepare



Our move to GPU computing

- Hardware video processing acceleration
  - Fast but limited
  - Out-classed over time
  - Not a good development to benefit ratio
- GPU Computing
  - Interesting, broader alternative
  - More and more customers had a powerful GPU sitting in their system
  - Ride the curve brought by gaming and HPC



Why OpenCL?

- Cross-vendor and cross-platform
  - Open standard
  - Multiple vendor API → Best use of development resources
  - One set of work → NVIDIA, AMD, and Intel

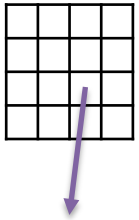
- Aligned very well with our needs
  - Most image processing is extremely parallel
  - OpenCL C
    - Very approachable
    - Excellent image processing support
    - Easy to port CPU implementations

OpenCL basics

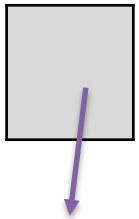
- Initialization
  - Host discovers what devices are available
  - Creates device contexts and command queue
  - Compiles kernels
- Processing
  - Makes data available to device
  - Runs kernels over 1D, 2D, or 3D global work sizes
  - Kernel executes a single work item

Design choice: Buffers or Images?

- Buffers
  - Raw memory
  - Fastest with best-case (coalesced) access patterns
  - Slowest with less-than-ideal access patterns
- Images
  - Abstracted storage
  - Fairly good with any access pattern that has locality
    - Due to texture caching
  - Better align with our image processing needs
    - Can use float4 regardless of underlying image format
    - Bilinear filtering “for free”
    - Border handling



```
uchar v = buffer[y*p+x];
```



```
float4 v = read_imagef(img, sampler, coord);
```

Simple color blend kernel

```
1   __constant sampler_t imageSampler = CLK_NORMALIZED_COORDS_FALSE |
2                                       CLK_ADDRESS_CLAMP |
3                                       CLK_FILTER_NEAREST;
4
5   __kernel void Blend(__read_only  image2d_t imageInput,
6                       __write_only image2d_t imageOutput,
7                       const float  flBlendAmount,
8                       const float4 colorBlend)
9   {
10     int x = get_global_id(0);
11     int y = get_global_id(1);
12
13     float4 inp = read_imagef(imageInput, imageSampler, (int2) (x,y));
14
15     float4 out = inp + (colorBlend - inp) * flBlendAmount;
16
17     write_imagef(imageOutput, (int2) (x,y), out);
18  }
```

Images in and out

Blending parameters

Image coordinate

Read float4 RGBA

Process in float4

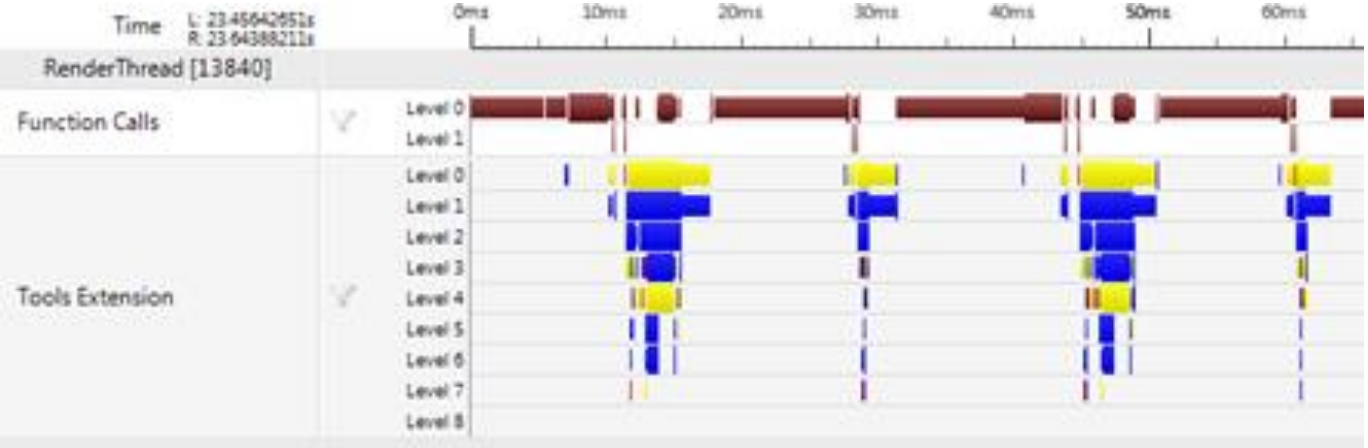
Write result

Welding it on

- Add GPU support
  - One piece at a time
  - Without breaking the application
- Image object extended
  - Automatic data movement
- Image processing functions extended
  - GPU path added one at a time
- No GPU support yet? → CPU code still worked

Tools

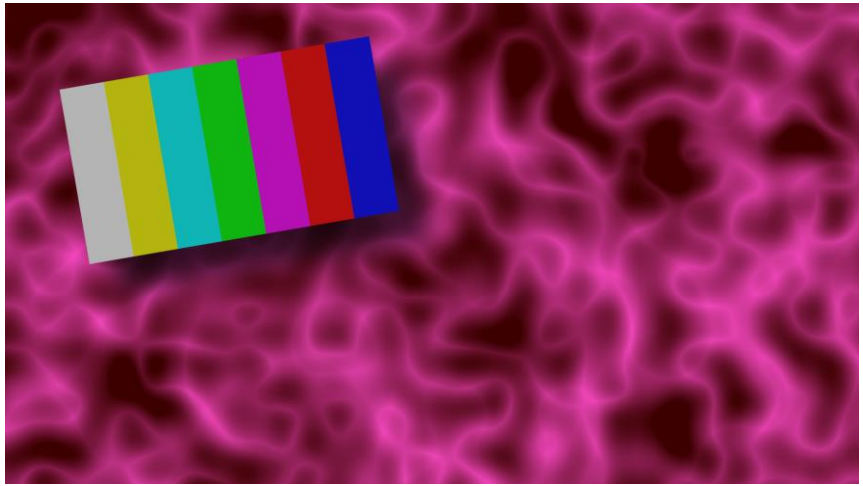
- NVIDIA Parallel Nsight and AMD APP Profiler for timeline traces
  - OpenCL API timing
  - Data upload/download timing
  - Kernel timing
  - Hierarchical host thread time ranges



Result

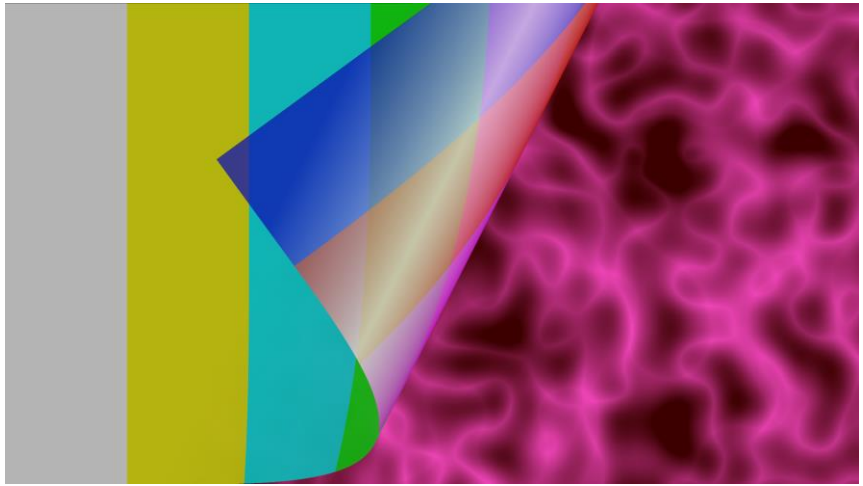
- Over 100 OpenCL kernels shipped
- Built-in functions

  YUV to RGB conversion, interlace handling, scaling, compositing, shadows, rotation, flips, cropping, fades, crossfades, etc.



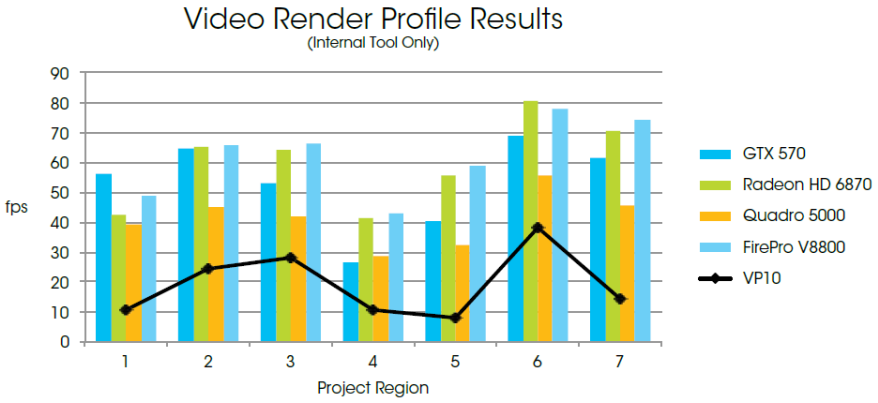
OpenFX plug-ins

- Over 60 GPU-accelerated OpenFX plug-ins
  - **Filters**
    Color Corrector, Blurs,
    Chroma Keyer, Lens Flare,
    Layer Dimensionality, etc.
  - **Transitions**
    Page Peel, Cross Effect,
    Clock Wipe, Zoom, etc.
  - **Generators**
    Noise Texture, Checkerboard
  - **Compositors**
    Bump Map, Layer Dimensionality
- Created OpenFX extension for getting OpenCL images
  - Now supported by multiple plug-in vendors



Wins

- 3-4x whole-pipeline performance
- Lightened load on CPU



- Later added OpenCL/OpenGL interop
  - Enabled 4K fullscreen realtime playback

Performance portability

- No vendor kernel differences
  - Bypass a few kernels on older drivers
- Very little vendor-specific host code
  - Mostly data transfer techniques

Pitfalls

- Early challenges
  - Buggy early drivers
  - Harsh learning curve
    - Why is my kernel crashing the driver?
  - No debugger
- Challenging algorithms
  - Took some time to get Gaussian Blur and Median filter fast

More recent challenges

- Vendor gap in OpenCL version support
  - We are very happy about NVIDIA's upcoming availability of OpenCL 1.2!
- Still finding the occasional driver bug

Next steps

New: **Catalyst Browse** and **Catalyst Prepare**

- Cross-platform
  - Windows/Mac OS X
- All-new video engine
  - OpenCL from the ground up

New video engine improvements

- Better Buffer and Image classes
- No fallback native-code CPU path
  - No compatible GPU? → OpenCL on the CPU
- Live GPU switching
  - Light up all eligible devices
  - Switch on the fly, even during playback
  - Paves the way for multi-GPU support

OpenCL performance improvements

- Free-pools
  - Reduce dynamic object allocation/deallocation
- Overlapped upload and compute
  - Compute on one frame while uploading next

Dynamic code generation

- OpenColorIO color management
  - Standard and consistent but slow
  - Has OpenGL GLSL shader code generation
    - Less accurate than CPU path

- Added OpenCL C kernel code generation
  - Produces the same results as CPU path
  - 100x faster than CPU path
  - Contributing back to open-source

Future

- Studying applications of OpenCL 2.x
  - Shared Virtual Memory
  - Dynamic Parallelism
  - Pipes
  - SPIR-V (2.1)

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!

SONY