BLINK: A GPU-Enabled Image Processing Framework

Mark Davey
Lead HPC Engineer
The Foundry

The Foundry and HPC

- The Foundry
  - Founded in 1996
  - We develop award-winning visual effects, computer graphics and design software used globally by leading artists and designers
- HPC
  - We create frameworks to make best use of all available compute devices – “make things go faster”
  - Initial target: 2D Image Processing

2D Image Processing

- A fundamental component in many Foundry products.

  Used in such effects as:

  - Noise reduction
  - Keying
  - Motion and disparity estimation
  - Colour correction/grading
  - Panoramic stitching
  - 3D texture creation

Need to make it as fast as possible!

Moving to GPUs

- Traditionally used the CPU for image processing
- Lots of legacy code
- GPUs are great at image processing
- Our customers often have powerful GPUs but not always (e.g. render farms)
- Need a fallback CPU path
- Do not want to write same code multiple times (debugging, maintenance, new hardware, etc.)

The Solution - BLINK

- “Write once, deploy everywhere”
- Image processing algorithms expressed as kernels
- Kernels written in a C++ like, domain-specific language
- Kernels run over an iteration space
- Metadata expresses access patterns, image formats, boundary conditions, etc.
- Kernels are translated into different back-ends
- JIT Compilation for many paths

BLINK - Features

- Multiple back-ends supported
- Consistent results across devices
- Range of image formats and layouts available
- Kernel execution strategy left to framework
- Profiling (execute and transfer)

BLINK Back-ends

- CUDA (4.2, Compute Capability 2.0)
- OpenCL (1.1)
- GLSL (1.2)
- x86 (Scalar, SSE2, SSE4.1, AVX, AVX2)

BLINK Example

```
class GainImage: ImageComputationKernel<eComponentWise>
{
    param:
      Image<eRead, ePoint> src;
      Image<eWrite, ePoint> dst;
      float gain;

    void define(){
  defineParam(gain, “myGain” , 1.0f);
    }

    void process(){
      dst() = src() * gain;
    }
};
```

BLINK Example

```
    class GainImage: ImageComputationKernel<eComponentWise>
{
   param:
     Image<eRead, ePoint> src;
     Image<eWrite, ePoint> dst;
     float gain;

   void define(){
  defineParam(gain, "myGain" , 1.0f);
   }

   void process(){
     dst() = src() * gain;
   }
};
```

THE FOUNDRY

BLINK Example

```
class GainImage: ImageComputationKernel<eComponentWise>
{
   param:
     Image<eRead, ePoint> src;
     Image<eWrite, ePoint> dst;
     float gain;

   void define(){
 defineParam(gain, “myGain” , 1.0f);
   }

   void process(){
     dst() = src() * gain;
   }
};
```

BLINK - The Foundry

Nuke – Post Production Compositing Software

- Many key plug-ins written using BLINK
- BlinkScript
  - Customers can create kernels within Nuke for GPU and CPU
  - Multi-GPU support on selected configurations
- OCULA 4 – Stereoscopic Toolset

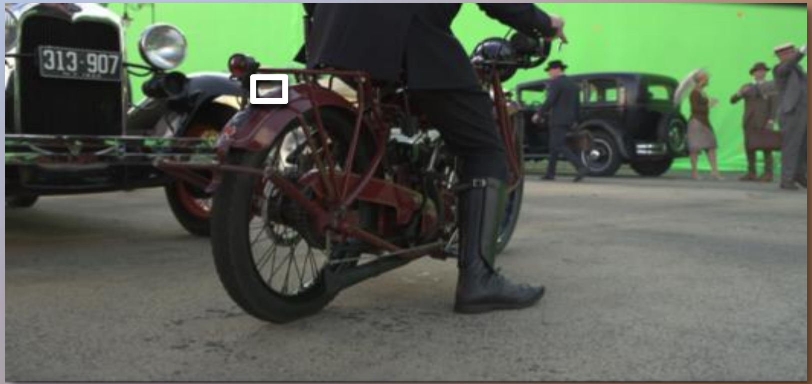
Projects

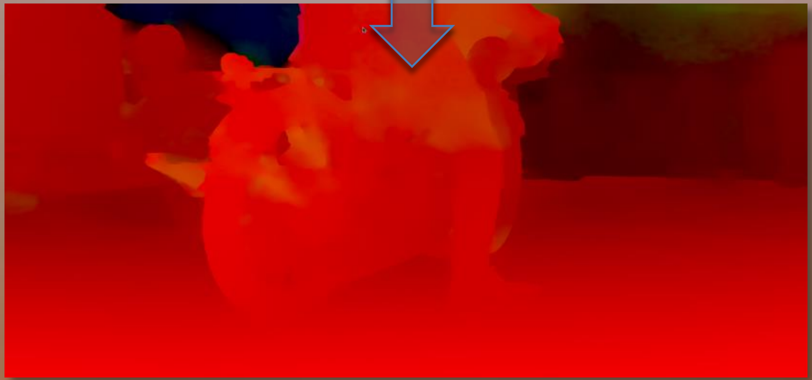
- ASAP – A Scalable 2D/3D Architecture for Cross Media Virtual Production
- Dreamspace – Advancements in Virtual Production Frameworks

OCULA

- A collection of Nuke tools to handle stereoscopic imagery
- Vector Disparity Generator at its heart
  - Correct colour and focus, automatically correct alignment, retime
- Latest version (4) written using BLINK
- Over 12K kernel calls per frame!

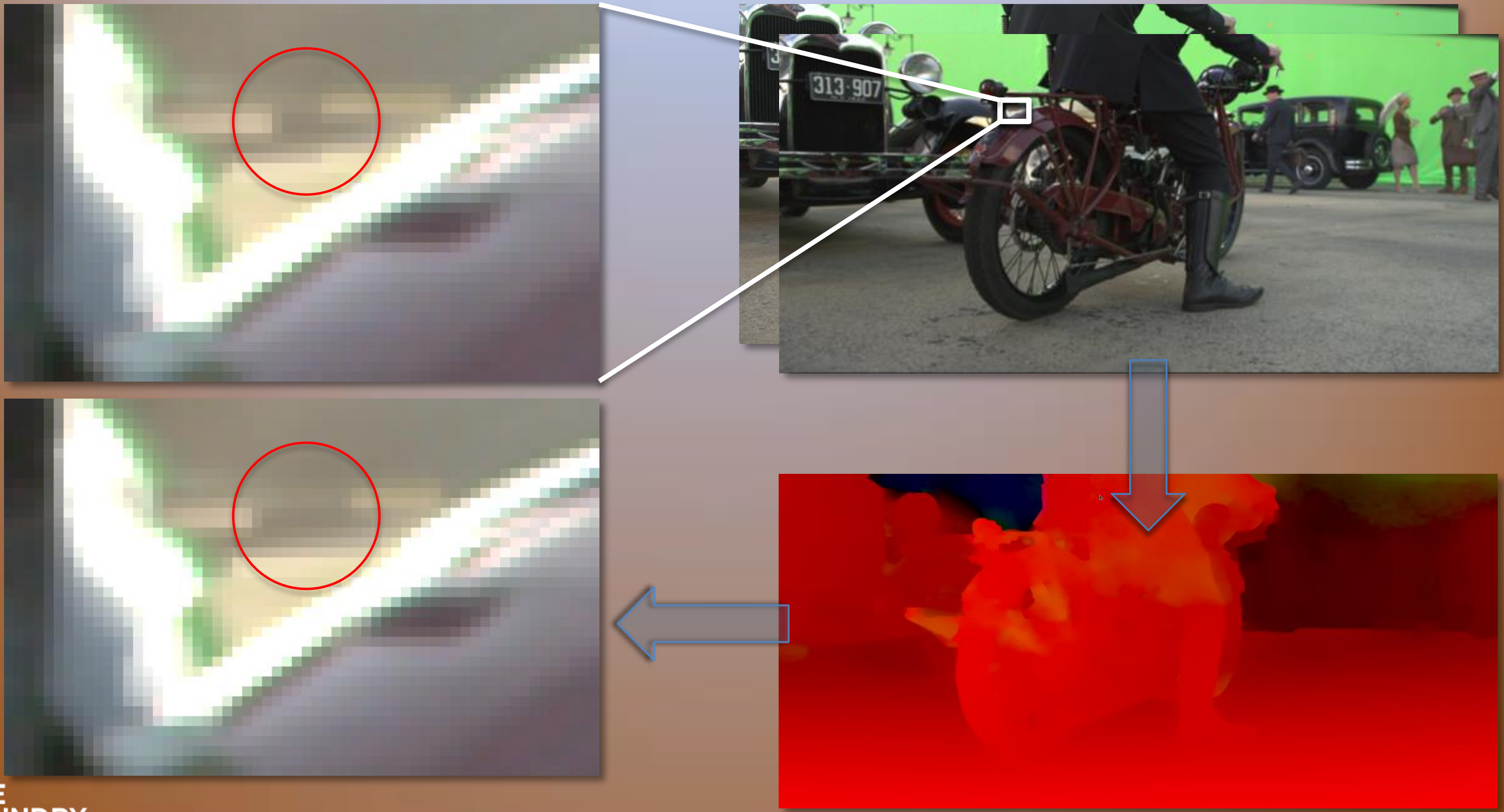
OCULA 4 – Disparity Generation





THE FOUNDRY

OCULA 4 – Different Devices



THE FOUNDRY

Numerical Identity I

- Our customers need visually identical results when processing on different devices.
- Some algorithms are extremely sensitive to small differences in mathematical results (e.g. OCULA!)
- Need to ensure numerical identity to guarantee visual identity

Numerical Identity – General Overview

- Disable fast math - to prevent compiler from reordering math operations.
- Force floating point literals to single precision - different compilers treat double literals differently giving inconsistent results.
- Disable Fused-Multiply-Add (FMA)
- Implement unified math library for all code paths
  - Algebraic functions — sqrt, hypot …
  - Transcendental functions — sin, exp …
  - Integral rounding functions — ceil, floor …
  - IEEE standard functions — fmod, fabs …
  - Matrices and operators — transpose, inverse …
  - Vectors and operators — dot, cross …
  - Others — min, max …

Numerical Identity – Platform Specifics

CUDA (nvcc flags)

- Disable “Flush Denormals To Zero” (--ftz=false)
- Disable “Fused Multiply Add” (--fmad=false)
- Enable precise square root and divide (--prec-sqrt=true --prec-div=true)

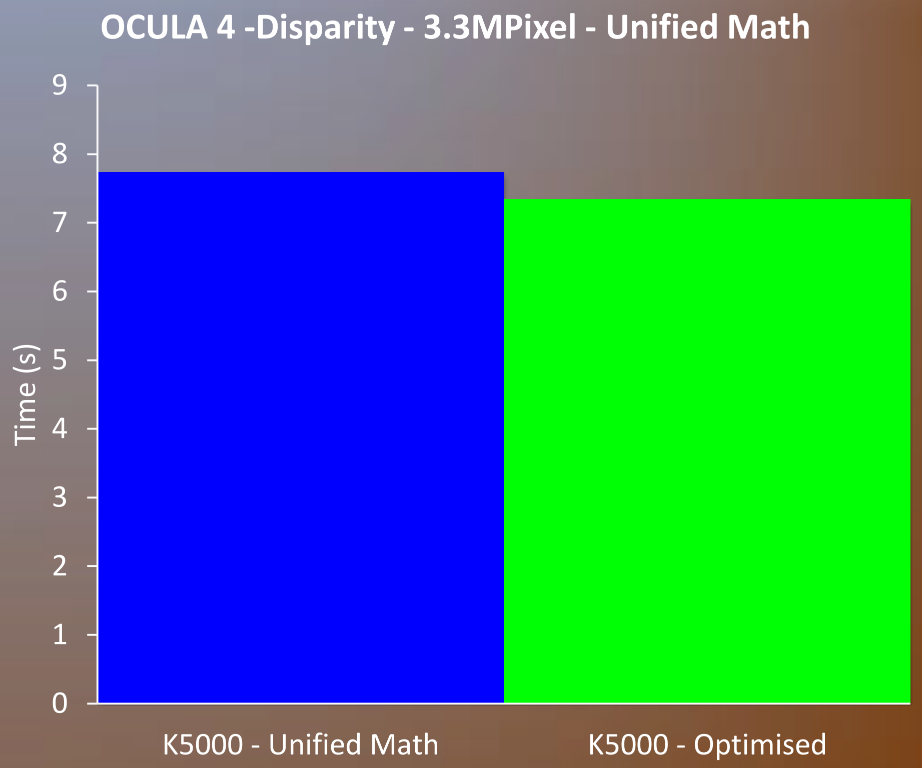
CPU:

- Precisely control FPU control register for rounding, denormal handing, etc (using _mm_setcsr intrinsic)
- Implement vector types (float1..float4, int1..int4,...)

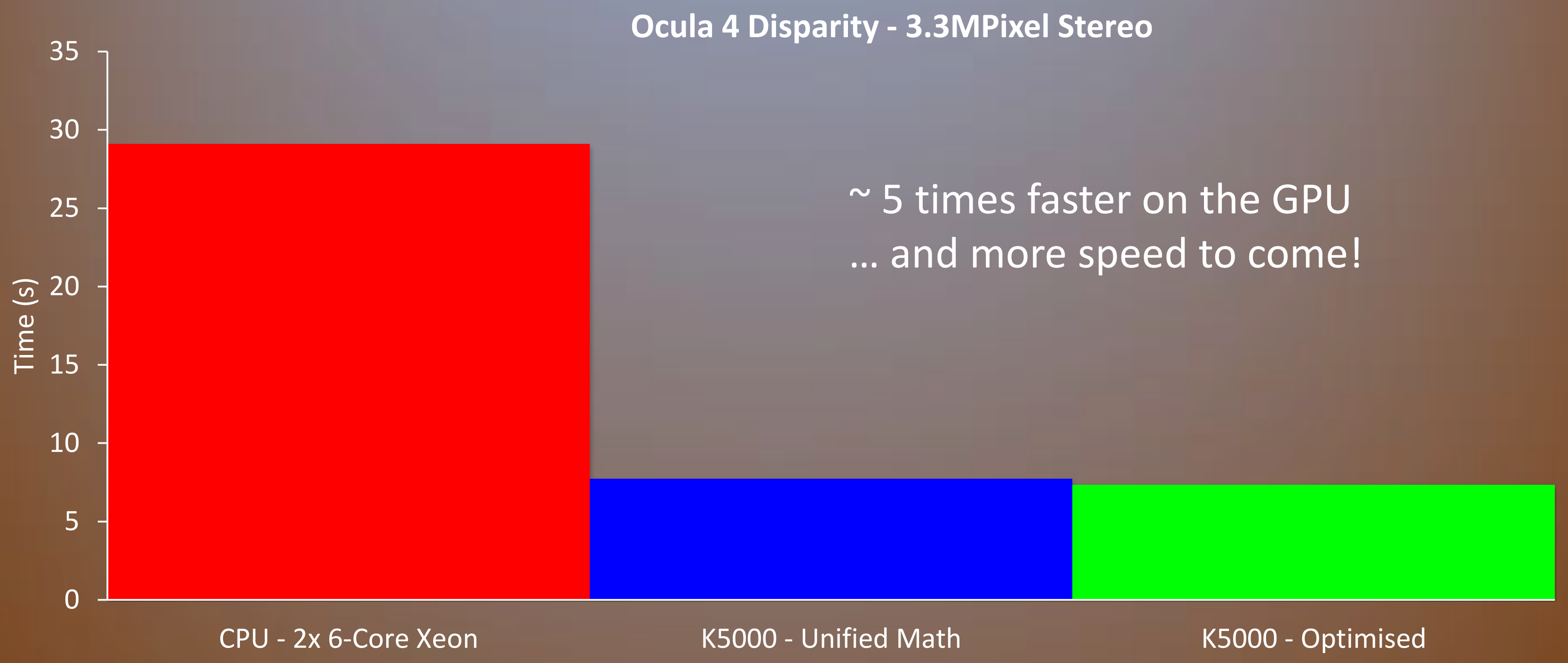
Also supported for OpenCL (NVIDIA GPUs only)

OCULA 4 - Results

- Disparity generation
- 3.3MPixel (2560x1350) frames
- End-to-end processing cost
- Only 5% overhead for Numerical Identity
- Many kernels are memory bound



OCULA 4 - Results



Under Development…Examples

- Heterogeneous Compute
  - Run graphs of kernels using scheduler
  - Target all available compute devices
  - Target data parallelism
- BLINK for Real-time
  - Export BLINK graphs from Nuke to run in BLINKPlayer
  - Kernels can be modified in BLINKPlayer
  - Parameters can be introspected from kernels and presented as GUI widgets
  - Composite live and rendered imagery

Thank You

Questions?