Advances in GPU-based Image Processing and Computer Vision

James Fung

Talk Objectives

- Introduce GPU hardware and features available for imaging & vision
- Present new algorithm mappings on the GPU
  - GPU vision is not just pixel operations!
- Introduce libraries and resources dedicated to GPU imaging & vision

Evolution of GPU Computing (Vision)

GPU Technology

Fixed Function Pipeline

Texture Combiners

Programmable Shaders
Single Precision Floating Point

Framebuffer Objects, Render to Texture

CUDA Architecture
C- for CUDA

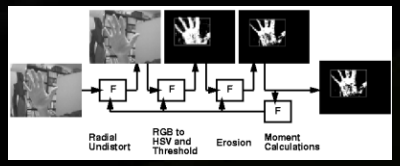
Double Precision

Vision Mappings

Image Derivatives, simple edge detection

Image Projections

On-GPU Math
Feature Detection
Image Filtering,
Bayer Demosaicing

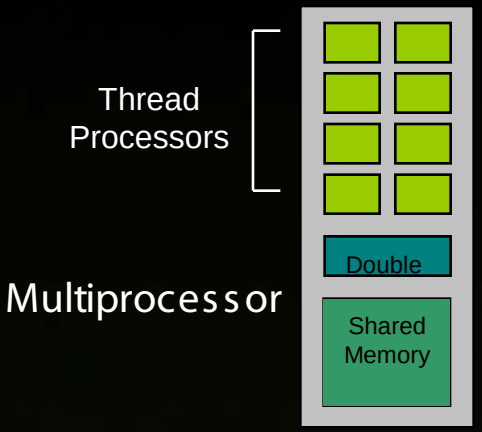


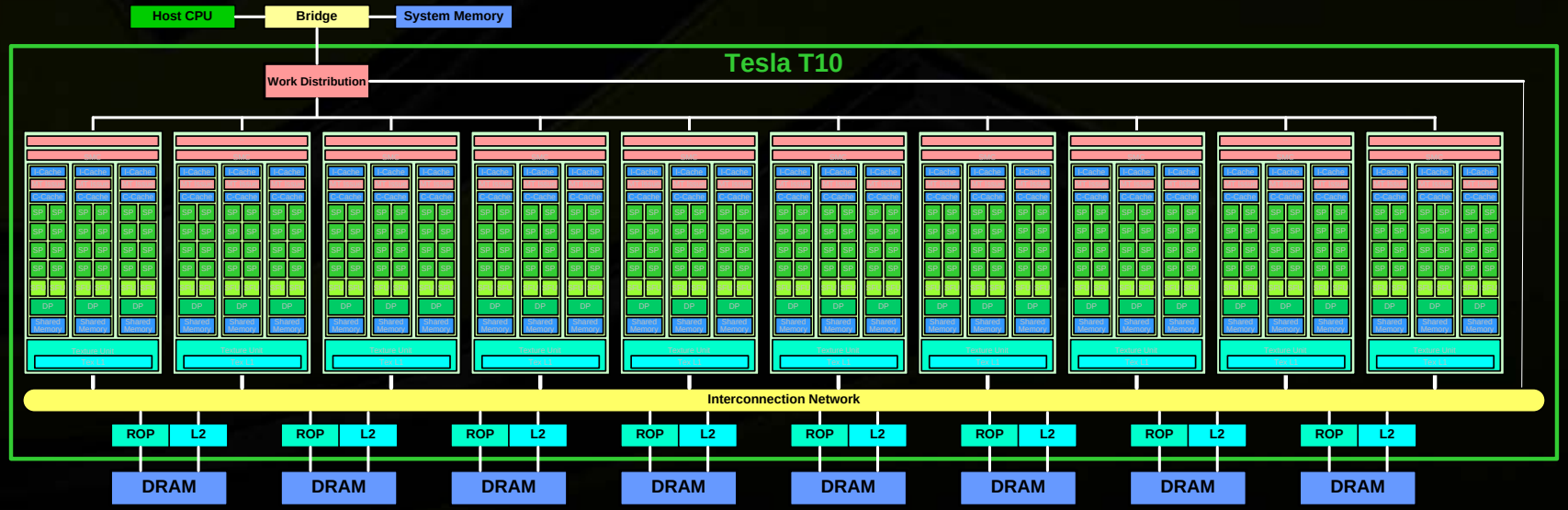
Imaging Pipelines
Fast “Gather” and (Global) Reduction Operations

General Numerical Computing
Full “On-GPU” algorithms

10-Series Architecture

- 240 thread processors grouped into 30 streaming multiprocessors (SMs) with 4.0 GB of RAM
  - 1 TFLOPS single precision
  - 87 GFLOPS double precision (IEEE 754 floating point)
  - Each SM:
    - 8 Thread Processors
    - 1 Double Precision Unit
    - 16 KB Shared Memory, 16394 Registers

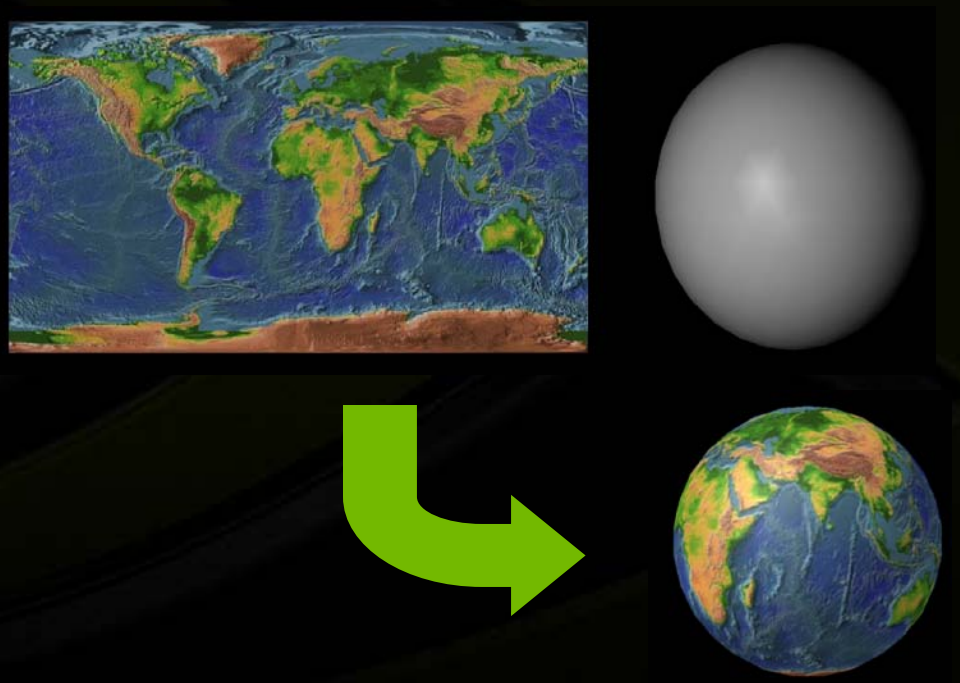




CUDA Features for Image and Video

Texture Hardware

Texture engine is a custom hardware block for image data access



Texture Features

- Cached
- Bilinear interpolation
- Data-type conversion
- Boundary clamping
- Multi-channel aware
- 1, 2, or 3D

When to Use?

- Achieve near-optimal performance with 2D spatially local data access
- Anytime image access requires interpolated data
- 1-D Look-up-tables. E.g. Gamma and tone mapping

CUDA Features for Image and Video Processing

Shared Memory (SMEM)

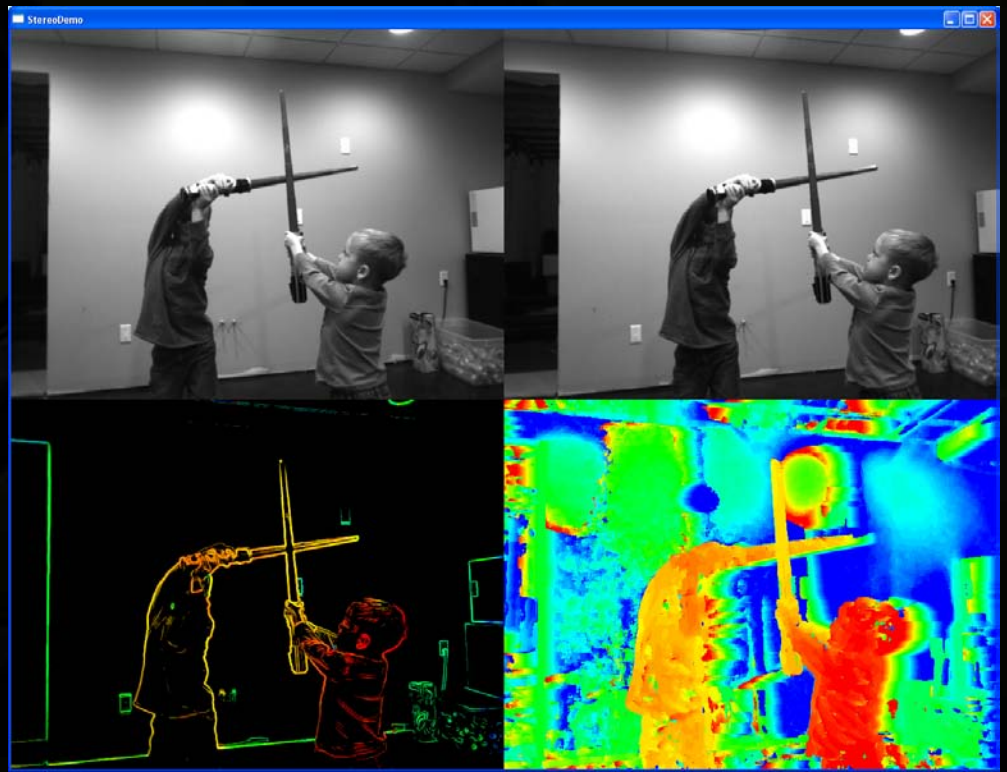
- Ultra-high speed “on-chip” memory
- Load an image tile into SMEM and share pixels between threads

CUDA Featuresfor Image and Video Processing

Visualization: Graphics Interop.

Image and Video frequently need interactive display

CUDA Interop with OpenGL and D3D allows display without additional data transfer overhead



Programming for CUDA

Standard C Code

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

```
__global__ void saxpy_parallel(int n, float a, float *x,
float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

See http://www.nvidia.com/cuda

GPU Panorama Stitching

- **Example imaging pipeline**

Panorama generated from three 3840x2880 images completed in 0.577s on a GTX280 GPU

GPU Panorama Stitching

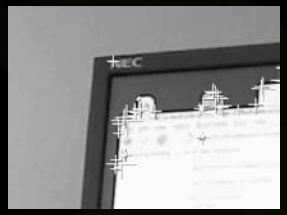
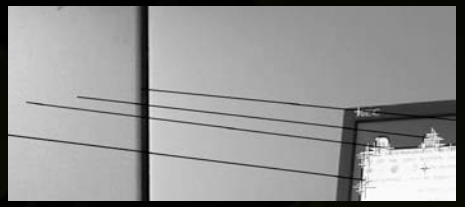
- **Example imaging pipeline**

Panorama generated from three 3840x2880 images completed in 0.577s on a GTX280 GPU

Imaging Pipeline: Panorama Stitching

Left Image 

Right Image 

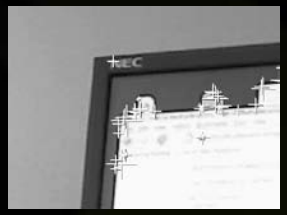
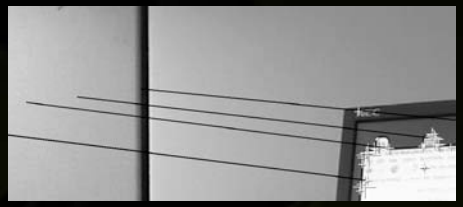





Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

Imaging Pipeline: Panorama Stitching

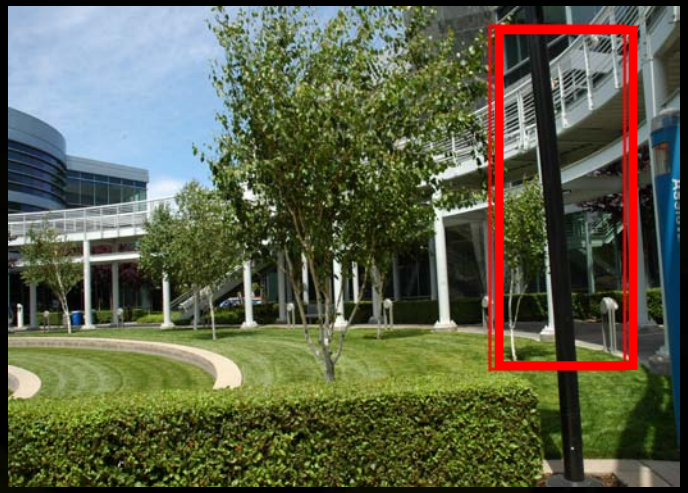
Left Image 

Right Image 


Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

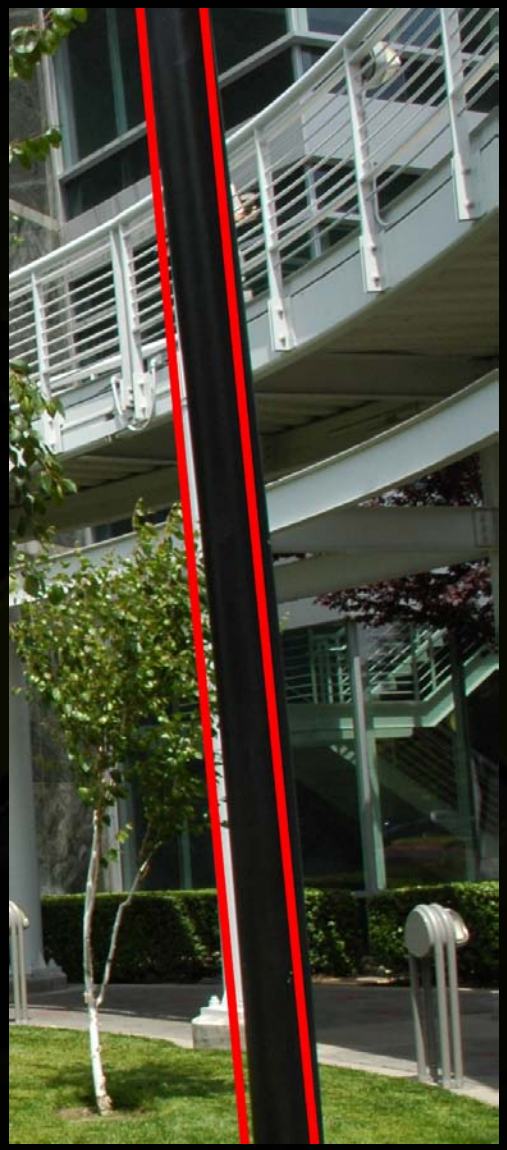
Radial Distortion Removal

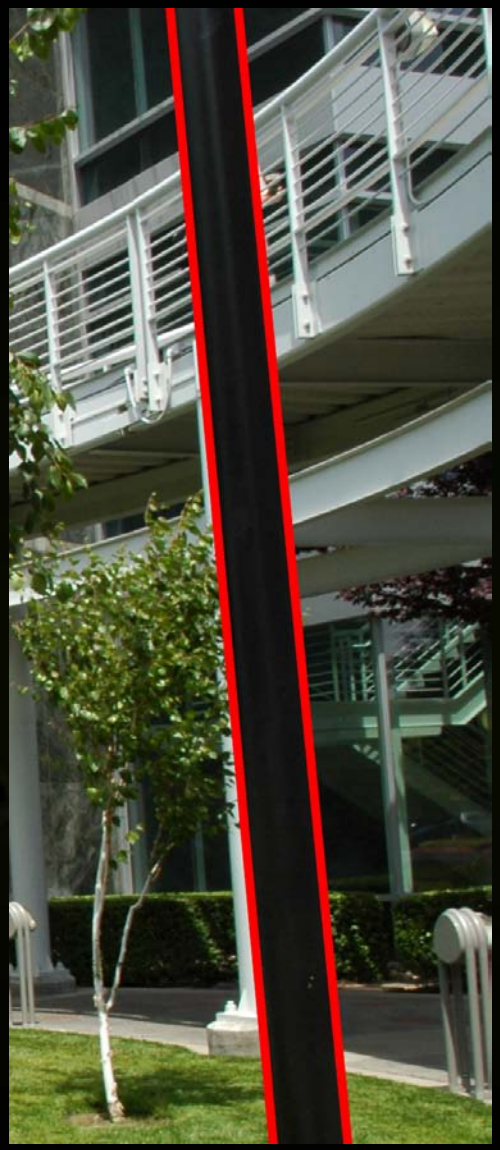


$$\delta x = x(\kappa_1 r^2 + \kappa_2 r^4 + \ldots)$$

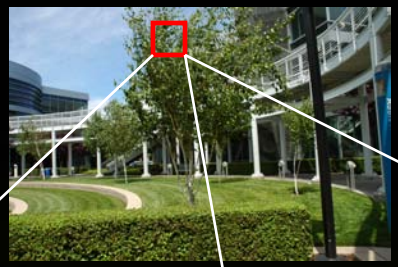
$$\delta y = y(\kappa_1 r^2 + \kappa_2 r^4 + \ldots)$$

Images taken with a Nikon D70 18-70mm NIKKOR Lens





Radial Distortion Removal



Point Samples

Bilinearly Interpolation (hw)

Bicubic Interpolation

Radial Distortion Removal

Interpolation Method	Time (ms)	
	Quadro 570m 4 SMs	Tesla C1060 30 SMs
Nearest Neighbor	70.99 ms	8.31 ms
Linear Interpolation	71.06 ms	8.39 ms
Bicubic (4 samples)	107.26 ms	11.77 ms

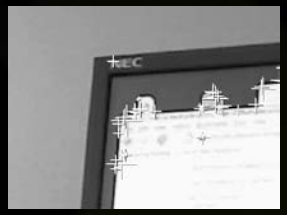
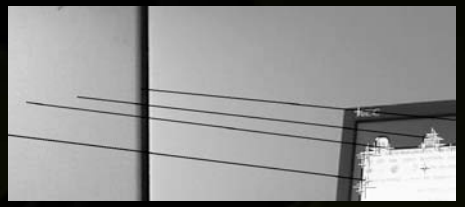
Input Image: 3008x2000 (6MP) RGB

- Linear Interpolation is “Free”!
- Apply hardware linear interpolation to approximate higher order interpolation *(see Simon Green’s “Bicubic” SDK example)*
- Excellent Texture Cache behaviour

Imaging Pipeline: Panorama Stitching

Left Image 

Right Image 


Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

Corner Detection

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Compute matrix A, in region (u,v) around a point (x,y), of Gaussian weighted (w(u,v,)) image derivatives (I_x, I_y)

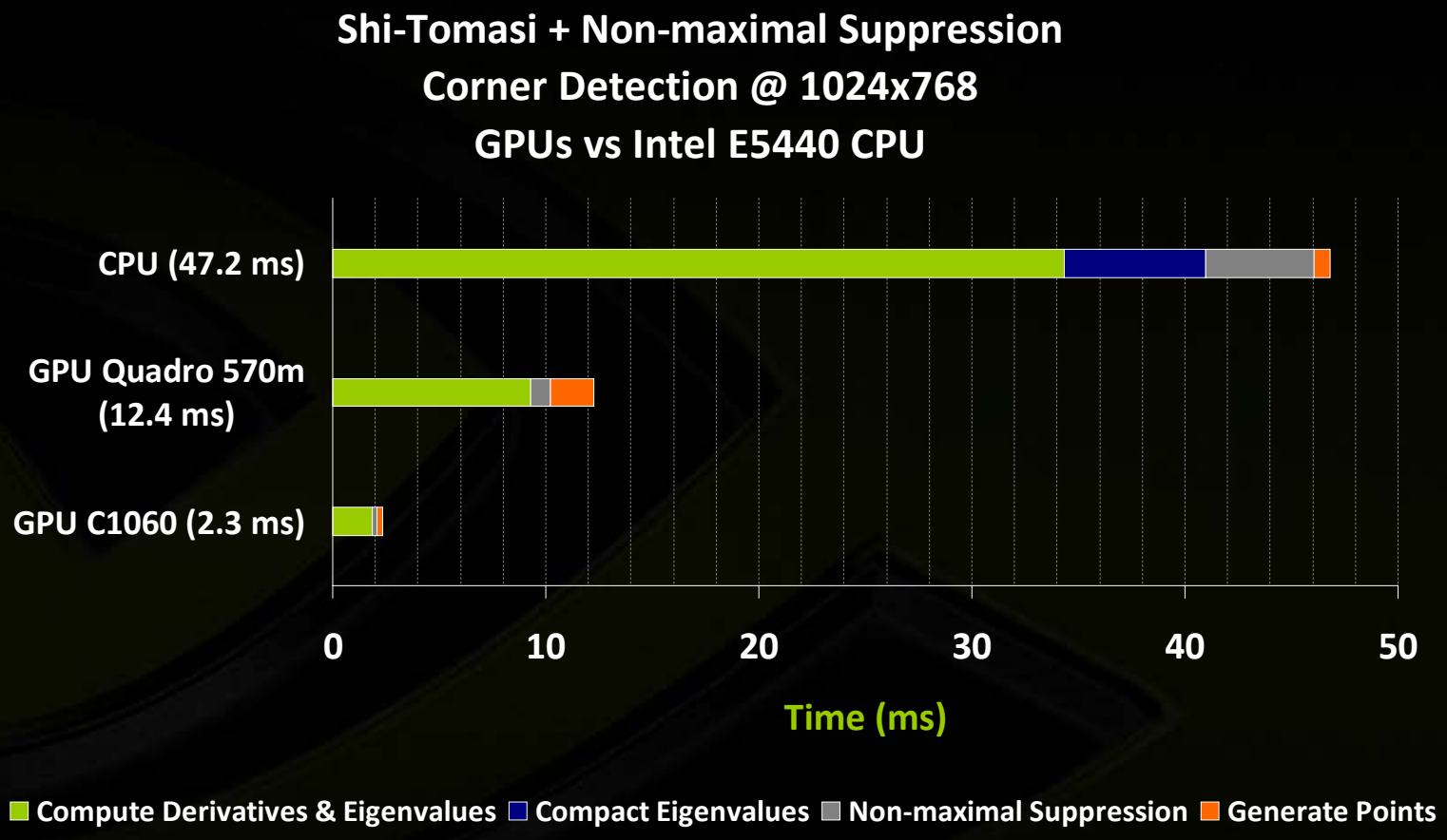
Harris: Compute Mc, (based on Eigenvalues of A, λ_1, and λ_2) by computing the determinant and trace of A

$$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(A) - \kappa \mathrm{trace}^2(A)$$

Shi-Tomasi: Compute Eigenvalues, λ_1, and λ_2 and threshold on min(λ_1, λ_2)

$$\lambda_1, \lambda_2 = \frac{\mathrm{tr}(A) \pm \sqrt{\mathrm{tr}(A)^2 - 4\det(A)}}{2}$$

Feature Detection



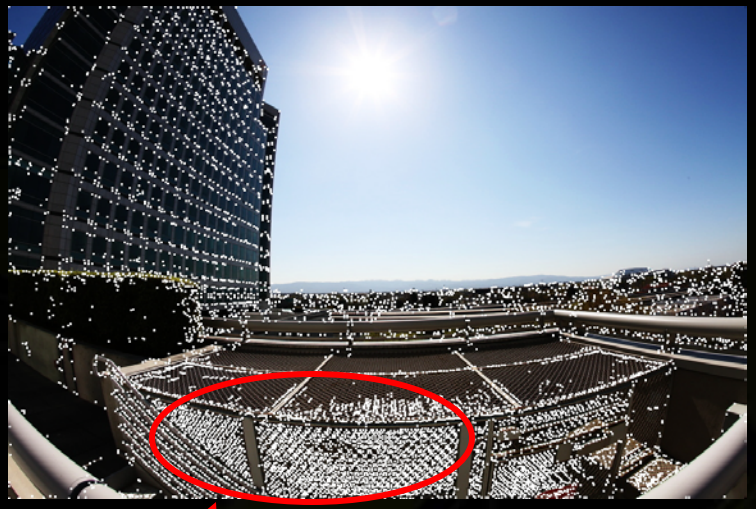
Corner Detection

Weak Threshold

Strong Threshold

- No single threshold can eliminate clutter and maintain weaker features

Corner Detection



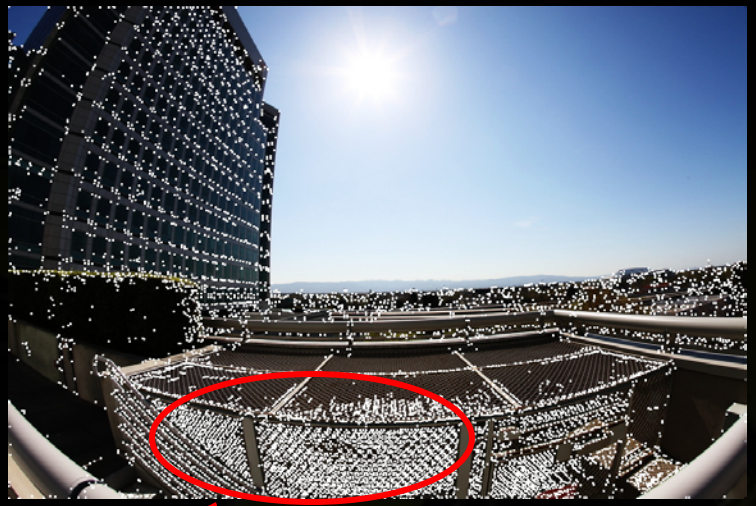
Weak Threshold

Strong Threshold

Indistinct cluttered features

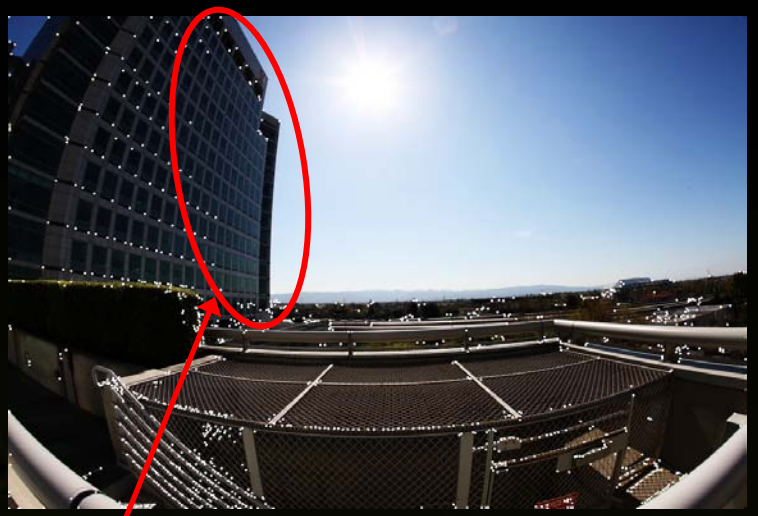
- No single threshold can eliminate clutter and maintain weaker features

Corner Detection



Weak Threshold

Indistinct cluttered features

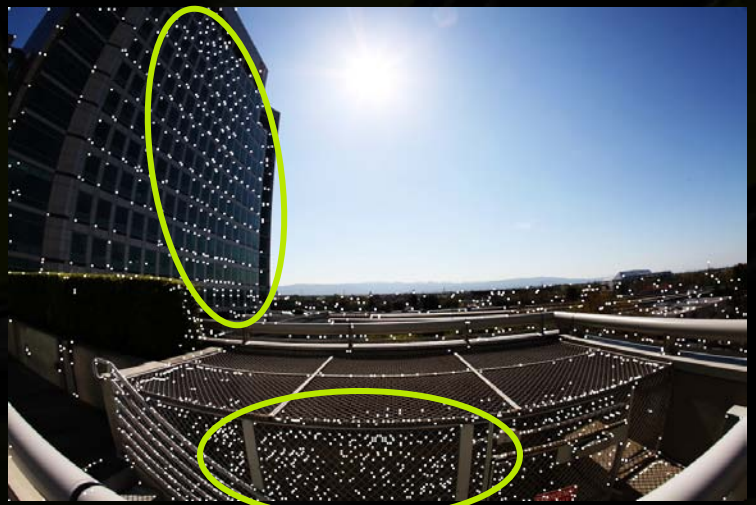


Strong Threshold

Loss of salient points

- No single threshold can eliminate clutter and maintain weaker features

Modified Shi-Tomasi



Dynamic Threshold

- Take ratio of min(λ_1, λ_2) to its neighbourhood
- Reduces clutter, maintains distinctive (though weaker) features

Corner Detection: Dynamic Method

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Compute matrix A, in region (u,v) around a point (x,y), of Gaussian weighted (w(u,v,)) image derivatives (I_x, I_y)

Dynamic Thresholding: Compute Eigenvalues

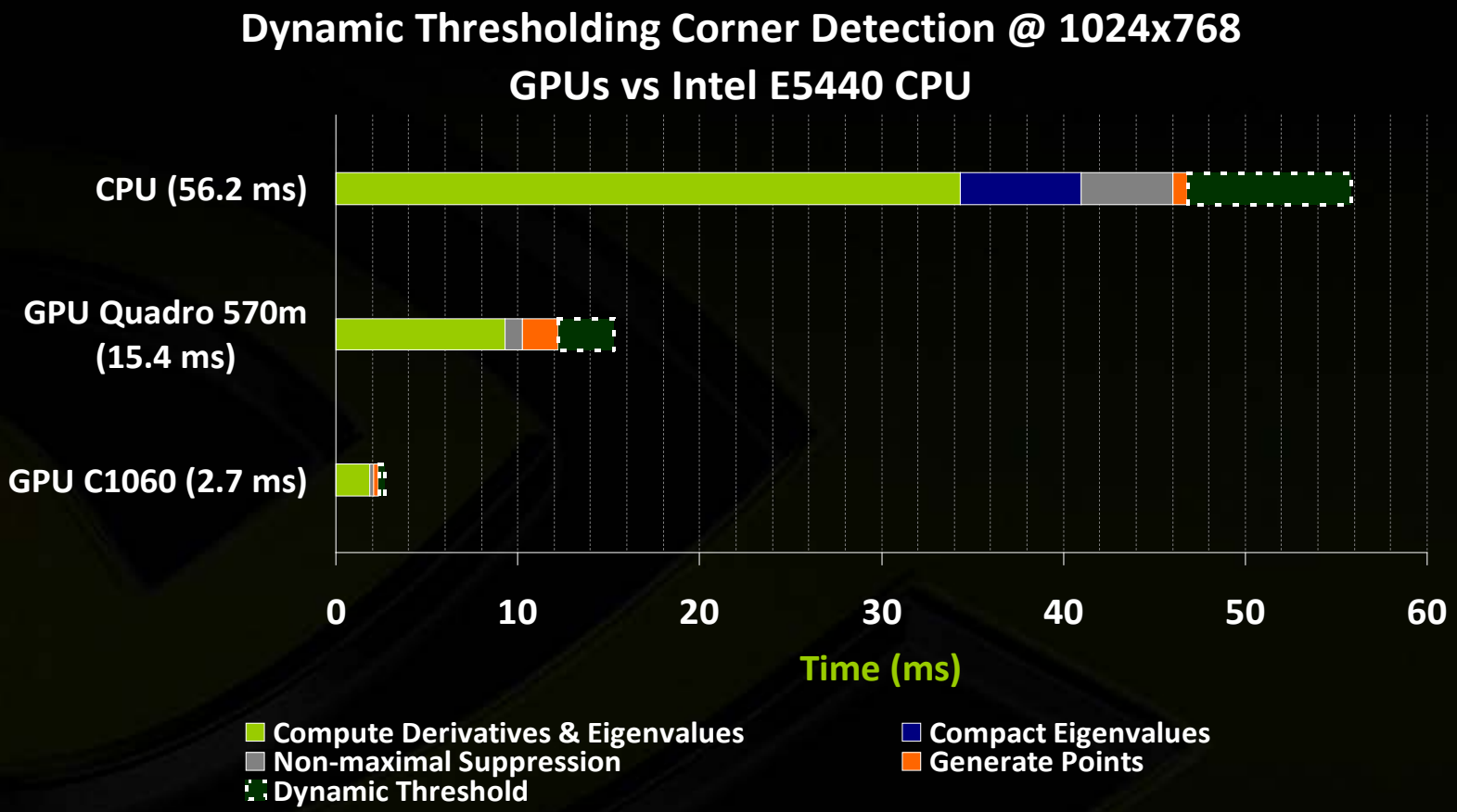
$$\lambda_1, \lambda_2 = \frac{\mathrm{tr}(A) \pm \sqrt{\mathrm{tr}(A)^2 - 4\det(A)}}{2}$$

Dynamic Threshold: Compare min(λ_1, λ_2) to regional (u,v) minimum

$$M_r = \frac{\min(\lambda_1, \lambda_2)}{\sum_u \sum_v \min(\lambda_1, \lambda_2)}$$

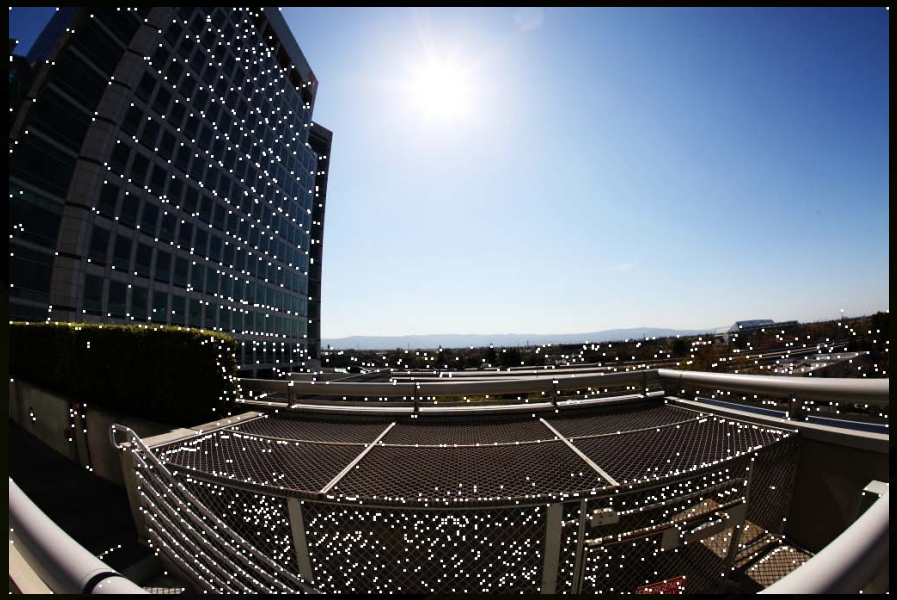
Additional Computational Cost: One 2D convolution and a division/comparison

Dynamic Feature Detection



- More complex algorithms
  → better quality/performance with GPU

Pixels to Points: *HistoPyramids*



(0,0)
(24,20)
(48,20)
(124,30)
(148, 44)
(24,45)
(56, 48)
(347,569)
(271,756)
(273,881)
…

- How to go from pixels to (x,y) point coordinates, *on the GPU*?

GPU HistoPyramids

- Based on papers by Ziegler et al. (NVIDIA)
- Able to generate a list of feature coordinates completely on the GPU
- Determines:
  - What is the location of each point?
  - How many points are found?
- Applicable for quadtree data structures

Gernot Ziegler, Art Tevs, Christian Theobalt, Hans-Peter Seidel, GPU Point List Generation through HistogramPyramids
Technical Report, June 2006
See http:// www.mpii.de/~gziegler for full information

HistoPyramids

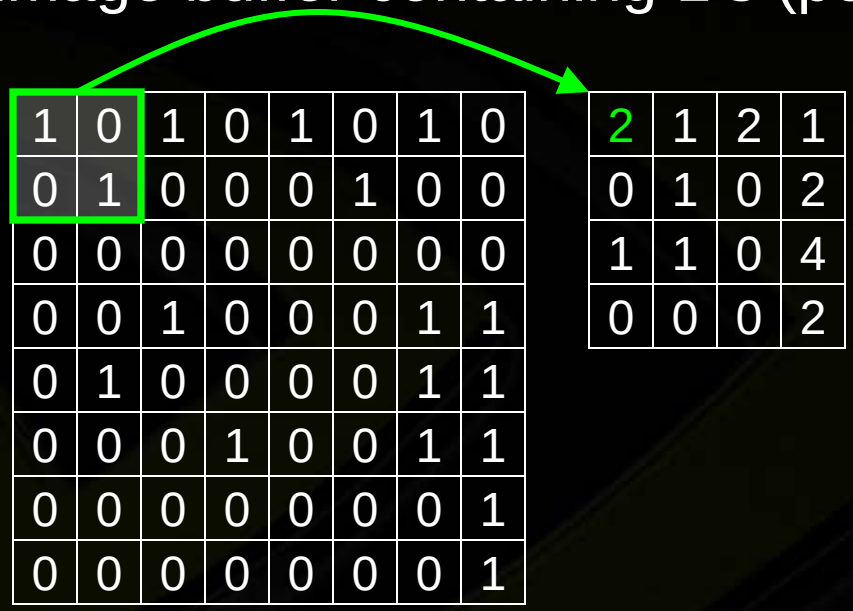
- How do we generate a list of points on the GPU from an image buffer containing 1's (points) and 0's (non-points)

1	0	1	0	1	0	1	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1
0	1	0	0	0	0	1	1
0	0	0	1	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

- Do a reduction: each level is the sum of 2x2 region "below" it
- The top level is the number of points total

HistoPyramids

- How do we generate a list of points on the GPU from an image buffer containing 1’s (points) and 0’s (non-points)



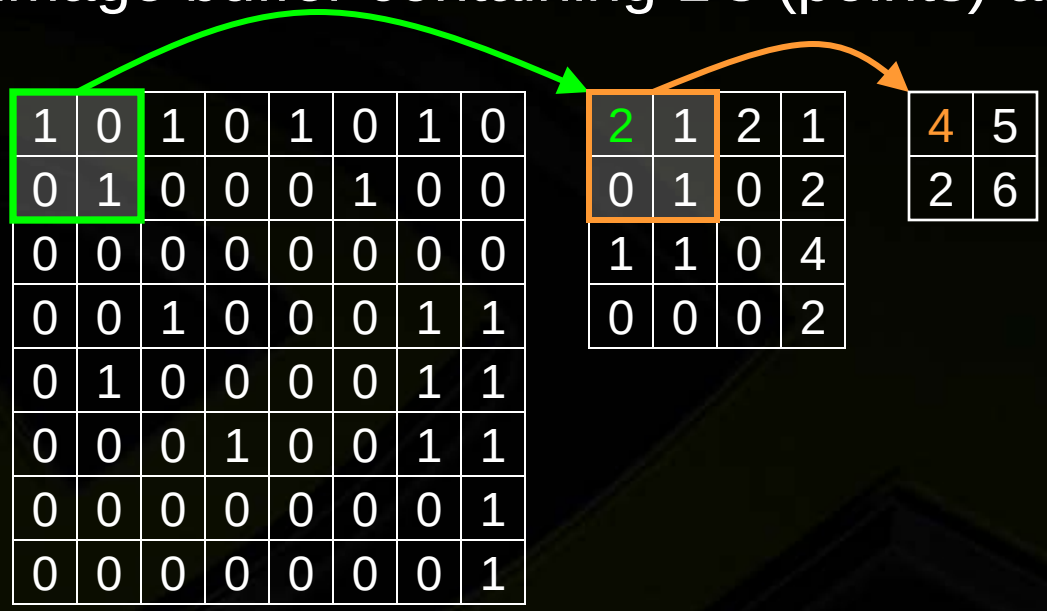
1	0	1	0	1	0	1	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1
0	1	0	0	0	0	1	1
0	0	0	1	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

2	1	2	1
0	1	0	2
1	1	0	4
0	0	0	2

- Do a reduction: each level is the sum of 2x2 region “below” it
- The top level is the number of points total

HistoPyramids

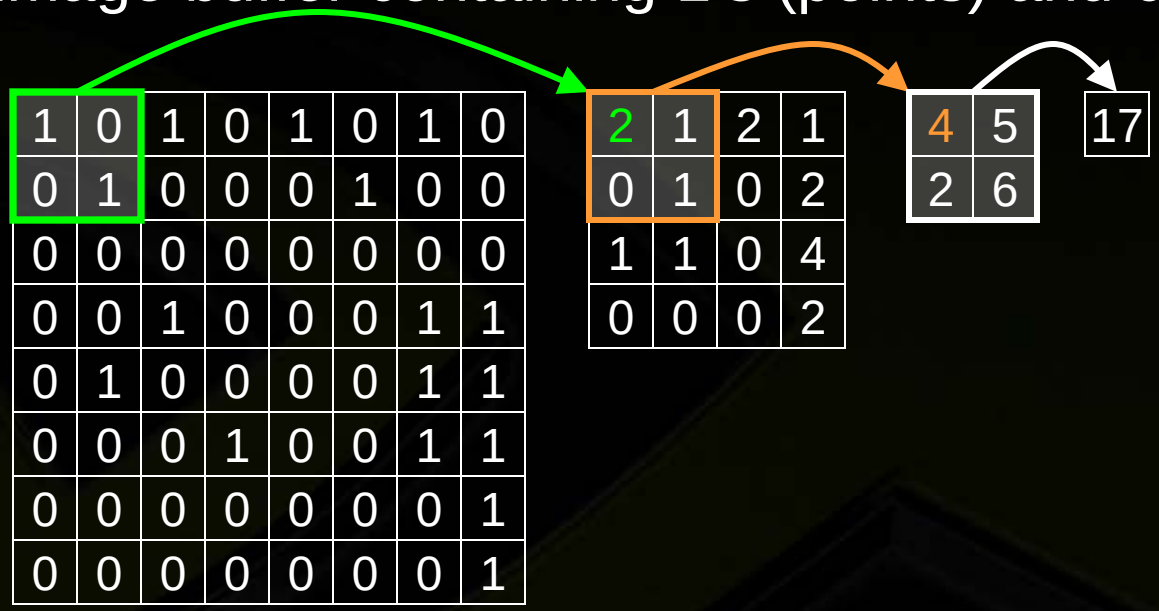
- How do we generate a list of points on the GPU from an image buffer containing 1's (points) and 0's (non-points)



- Do a reduction: each level is the sum of 2x2 region "below" it
- The top level is the number of points total

HistoPyramids

- How do we generate a list of points on the GPU from an image buffer containing 1's (points) and 0's (non-points)



- Do a reduction: each level is the sum of 2x2 region "below" it
- The top level is the number of points total

HistoPyramid

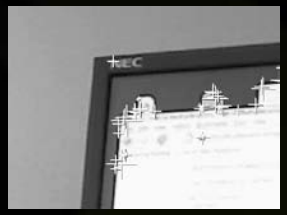
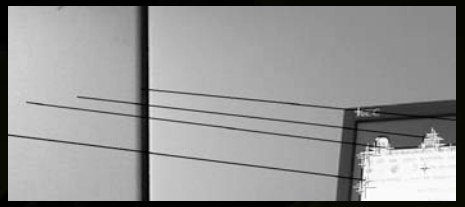
- The pyramid is now a *map* to where the point locations are
- Traverse down the pyramid, counting past points, and populate the list
- Example: at what coordinates is the 5th point in the list

Time: (C1060 GPU)
1024x1024: 0.27 ms
4096x4096: 2.1 ms
8192x8192: 7.7 ms

Imaging Pipeline: Panorama Stitching

Left Image 

Right Image 


Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

Generating Descriptors

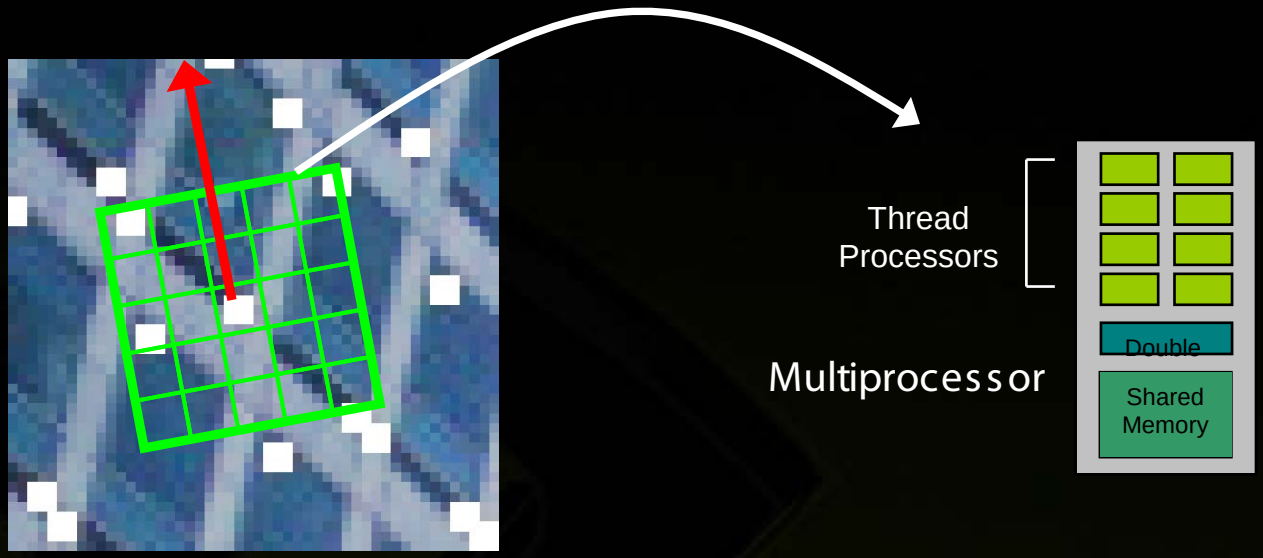
- What's a Feature Descriptor?
  - Distinct numerical representation of an image point for matching
- Our example: SIFT
  - "Scale Invariant Feature Transform"
  - 128 element floating point vector ("key")
  - Nearest Euclidean distance between keys is the best match

Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision* 60 (2): 91–110.

Generating Descriptors

- Sparse point processing
- HistoPyramid tree organization gives good spatial locality!
- Previous fragment shaders could do this but limited to one fragment processor (thread) per point
  - Inefficient use of parallel processing architecture on GPU
- Parallel processing of single descriptor now possible via thread cooperation
- Thread Cooperation a feature of CUDA achieved via:
  - Shared Memory
  - Thread Synchronization

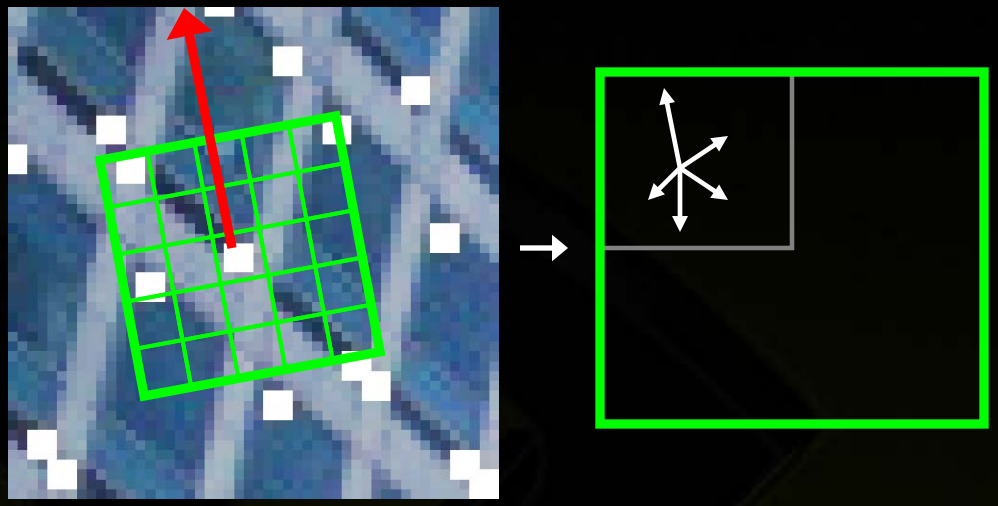
Feature Descriptor Computation



1. Calculate feature orientation
   (shared memory reduction)
2. Lookup rotated samples
   (texture cache)

- Made possible by Thread Cooperation and data dependent array indexing in compute shaders
- Good texture cache usage, constant cache (Gaussian weights)

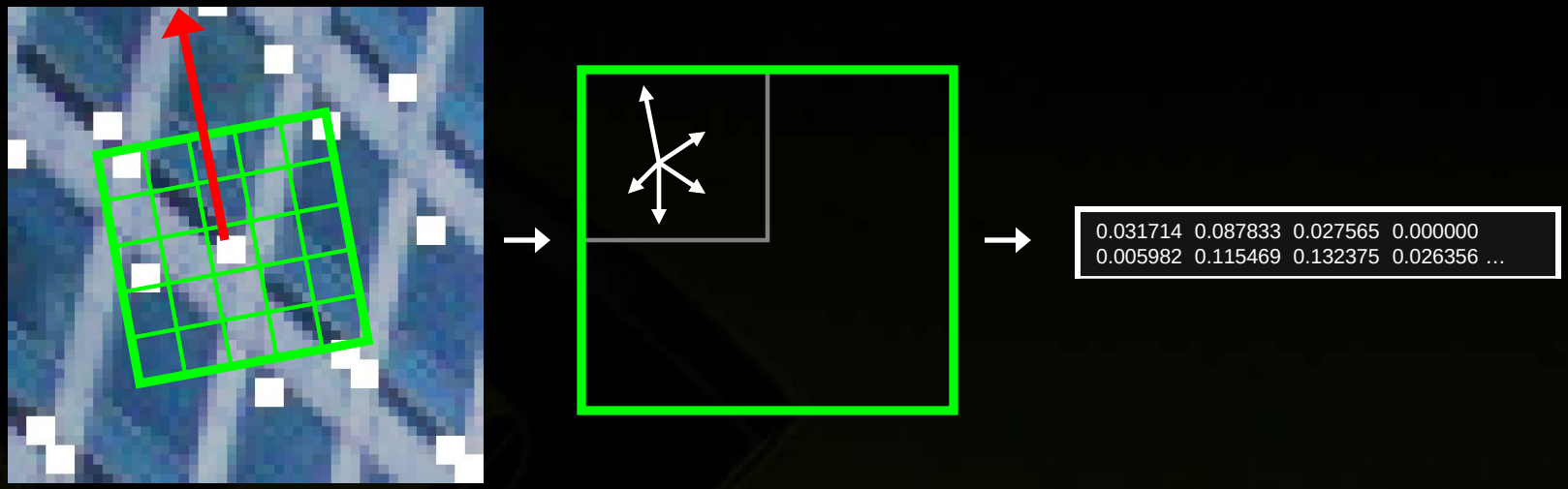
Feature Descriptor Computation



1. Calculate feature orientation *(shared memory reduction)*
2. Lookup rotated samples (texture cache)
3. Generate local orientation histograms (one thread per histogram, shared memory, pointer indexing)

- Made possible by Thread Cooperation and data dependent array indexing in compute shaders
- Good texture cache usage, constant cache (Gaussian weights)

Feature Descriptor Computation



1. Calculate feature orientation *(shared memory reduction)*
2. Lookup rotated samples (texture cache)
3. Generate local orientation histograms (one thread per histogram, shared memory, pointer indexing)
4. Normalize Histogram *(shared memory reduction)*
5. Threshold
6. Re-normalize Histogram *(shared memory reduction)*

- Made possible by Thread Cooperation and data dependent array indexing in compute shaders
- Good texture cache usage, constant cache (Gaussian weights)

Feature Matching (SIFT Descriptors)

GPU	Rate (key pair comparisons/ms)
Tesla C1060	34,983

- **Matching Operation: Best match is nearest neighbour (nearest distance between vectors)**

- **See Also: V. Garcia, E. Debreuve, and M. Barlaud, "Fast K nearest neighbor search using GPU," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08***

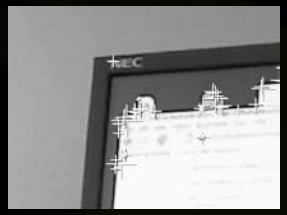
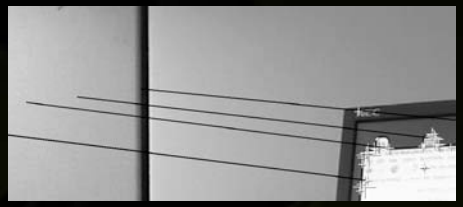
0.031714 0.087833 0.027565 0.000000 …
0.031714 0.087833 0.027565 0.000000 …
0.000000 0.134660 0.049070 0.000000 …
0.011031 0.045146 0.034813 0.000000 …
0.000000 0.087889 0.089118 0.000000 …
0.011031 0.089875 0.007983 0.000000 …
0.019691 0.162070 0.083896 0.000000 …
0.000000 0.120853 0.049070 0.000000 …
0.000000 0.020656 0.055590 0.075415 …
0.000000 0.224569 0.082681 0.000000 …
0.000000 0.000000 0.124618 0.075472 …
0.000000 0.033847 0.028859 0.075415 …
0.000000 0.000000 0.224569 0.155564 …
0.005982 0.115469 0.132375 0.026356 …
0.000000 0.051259 0.176472 0.000000…
0.005982 0.123400 0.089118 0.011102…
…

0.031714 0.087833 0.027565 0.000000 …
0.031714 0.087833 0.027565 0.000000 …
0.000000 0.134660 0.049070 0.000000 …
0.011031 0.045146 0.034813 0.000000 …
0.000000 0.087889 0.089118 0.000000 …
0.011031 0.089875 0.007983 0.000000 …
0.019691 0.162070 0.083896 0.000000 …
0.000000 0.120853 0.049070 0.000000 …
0.000000 0.020656 0.055590 0.075415 …
0.000000 0.224569 0.082681 0.000000 …
0.000000 0.000000 0.124618 0.075472 …
0.000000 0.033847 0.028859 0.075415 …
0.000000 0.000000 0.224569 0.155564 …
0.005982 0.115469 0.132375 0.026356 …
0.000000 0.051259 0.176472 0.000000…
0.005982 0.123400 0.089118 0.011102…
…

Imaging Pipeline: Panorama Stitching

Left Image 

Right Image 


Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

GPU Laplacian Pyramids

Source Image

L1 Image

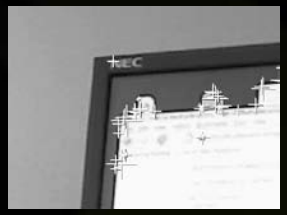
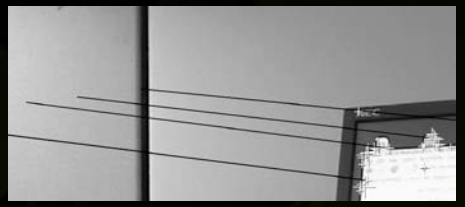
Hardware: Intel Xeon E5440 @ 2.83 GHz IPP 5.0, Generate L1 (from G0 and G1), C1060 GPU

Image Size.Type	CPU Time (sub. in place)	CPU Rate, Million Pixels/s (sub. in place)	GPU Time (SDK Convolution)	GPU Speedup (SDK Convolution)
3888x2592 32f_C1R (10 MP)	71.76 ms	140 M/s	7.11 ms	**10.1x**
3008x1960, 32f_C1R (6 MP)	45.979 ms	128 M/s	4.12 ms	**11.2x**
1024x1024, 32f_C1R (1 MP)	6.632 ms	158 M/s	0.8 ms	**8.29x**

Imaging Pipeline: Panorama Stitching

Left Image 

Right Image 


Radial Distortion Correction → Keypoint Detection & Extraction (Shi-Tomasi/SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

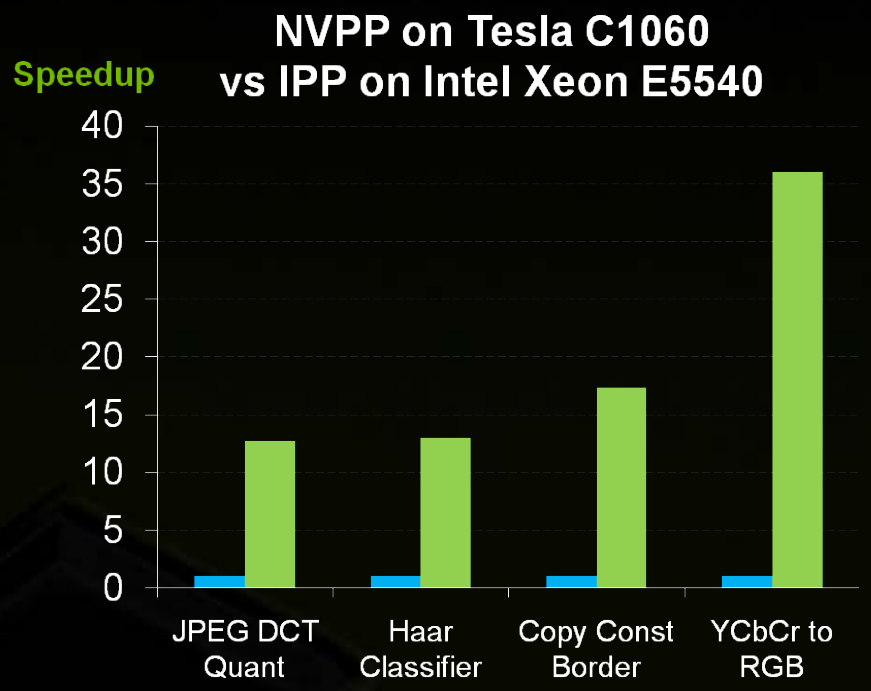
NVPP: Image Processing Primitives

- **Library of high performance image processing primitives**
- **http://www.nvidia.com/nvpp**

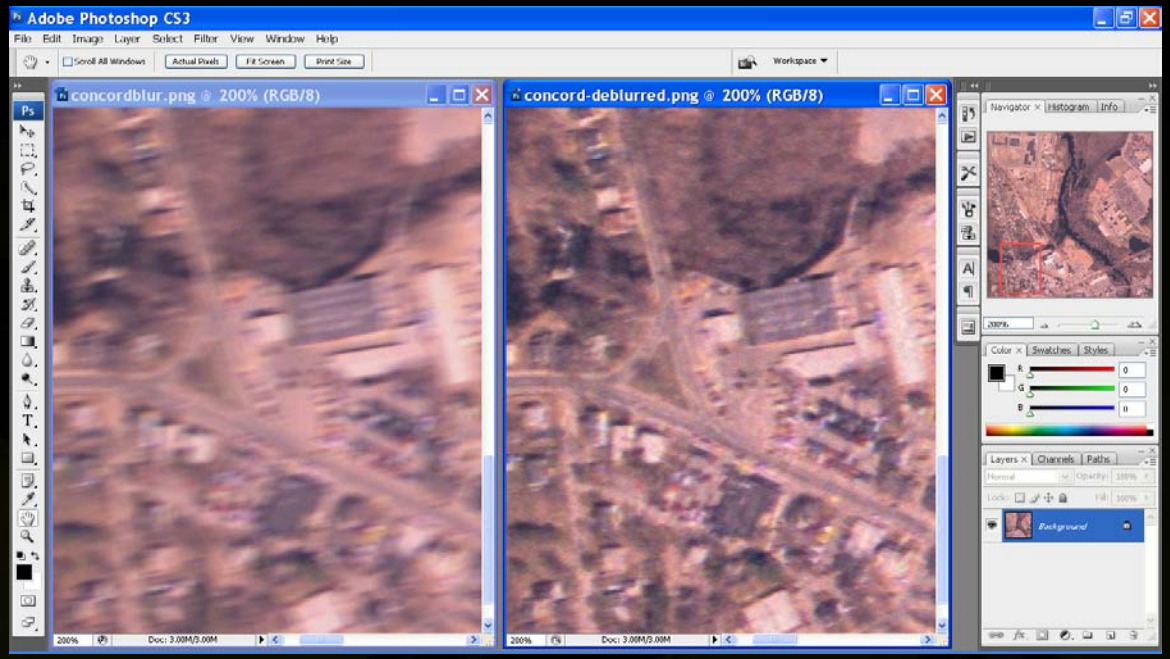
- **Data exchange & initialization**
  - Set, Convert, CopyConstBorder, Copy, Transpose, SwapChannels
- **Arithmetic & Logical Ops**
  - Add, Sub, Mul, Div, AbsDiff
- **Threshold & Compare Ops**
  - Threshold, Compare
- **Color Conversion**
  - RGB To YCbCr (& vice versa), ColorTwist, LUT_Linear
- **JPEG**
  - DCTQuantInv/Fwd, QuantizationTable
- **Filter Functions**
  - FilterBox, Row, Column, Max, Min, Median, Dilate, Erode, SumWindowColumn/Row
- **Geometry Transforms**
  - Mirror, WarpAffine / Back/ Quad, **WarpPerspective** / Back / Quad, Resize
- **Statistics**
  - **Mean, StdDev, NormDiff, MinMax,** Histogram, SqrIntegral, RectStdDev
- **Computer Vision**
  - **ApplyHaarClassifier, Canny**

NVPP: Performance

- Platform
  - NVPP on Tesla C1060
  - IPP (multi-threaded) on Intel Quad-core Xeon E5540 (Nehalem) 2.53 GHz
- Speedups from 10x to 36x

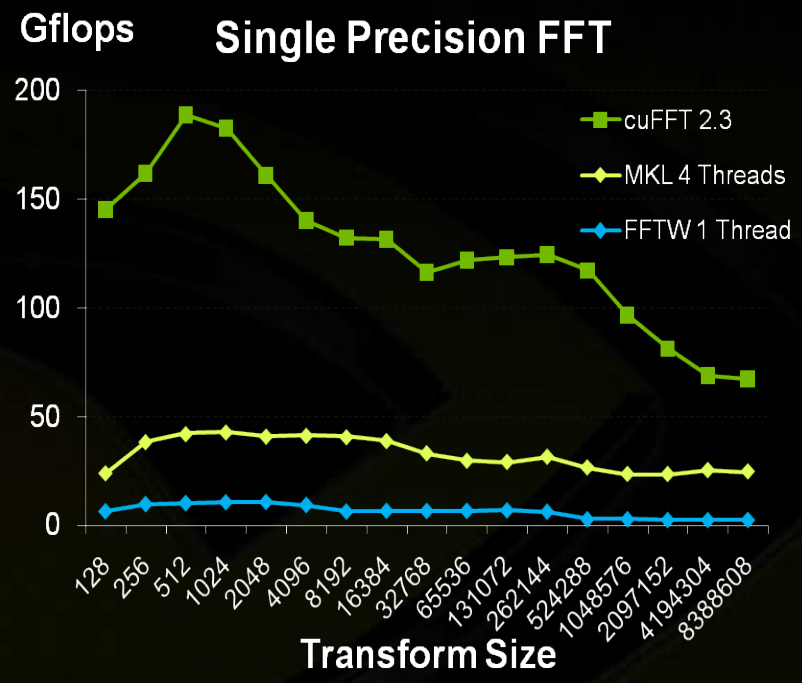


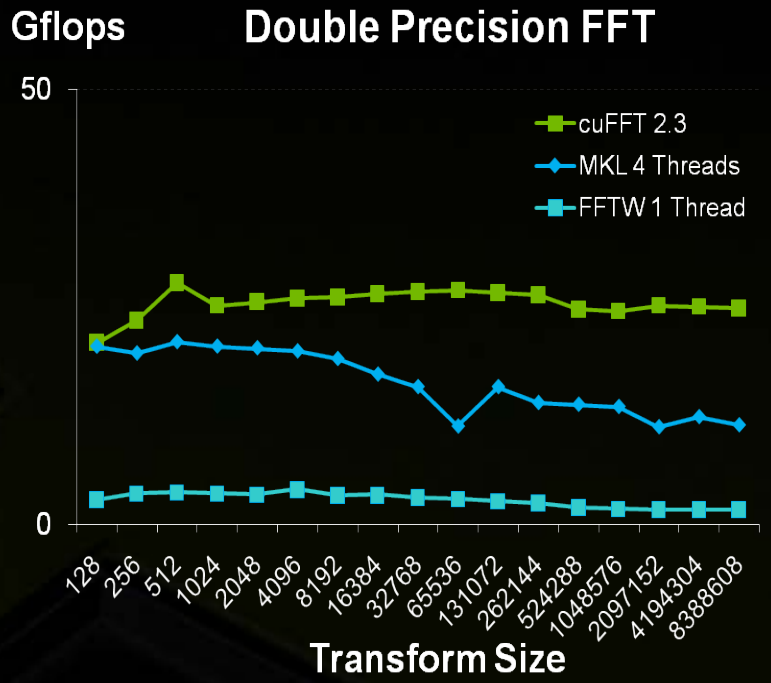
Frequency Domain Processing



- Application: Deconvolution FFT based processing (deblurring)
- CUDA FFT Library available
- Photoshop Plugin programming example available
- http://www.nvidia.com/cuda →Documentation→Windows

cuFFT: CUDA FFT Libraries





cuFFT 2.3: NVIDIA Tesla C1060 GPU

MKL 10.1r1: Quad-Core Intel Core i7 (Nehalem) 3.2GHz

Data resident on GPU (PCI-e transfer times not included)

GrabCuts: CUDA Graph Cuts

- Comparison between Boykov (CPU), CudaCuts and our GrabCuts implementation (by Timo Stich, NVIDIA)
  - Intel Core2 Duo E6850 @ 3.00 GHz
  - NVIDIA Tesla C1060

Dataset	Boykov (CPU)	CudaCuts (GPU)	Our (GPU)	Speedup Our vs CPU
Flower (600x450)	**191 ms**	**92 ms**	**20 ms**	9.5x
Sponge (640x480)	**268 ms**	**59 ms**	**14 ms**	19x
Person (600x450)	**210 ms**	**78 ms**	**35 ms**	6x

Average speedup over CPU is 11x

Additional Implementations:

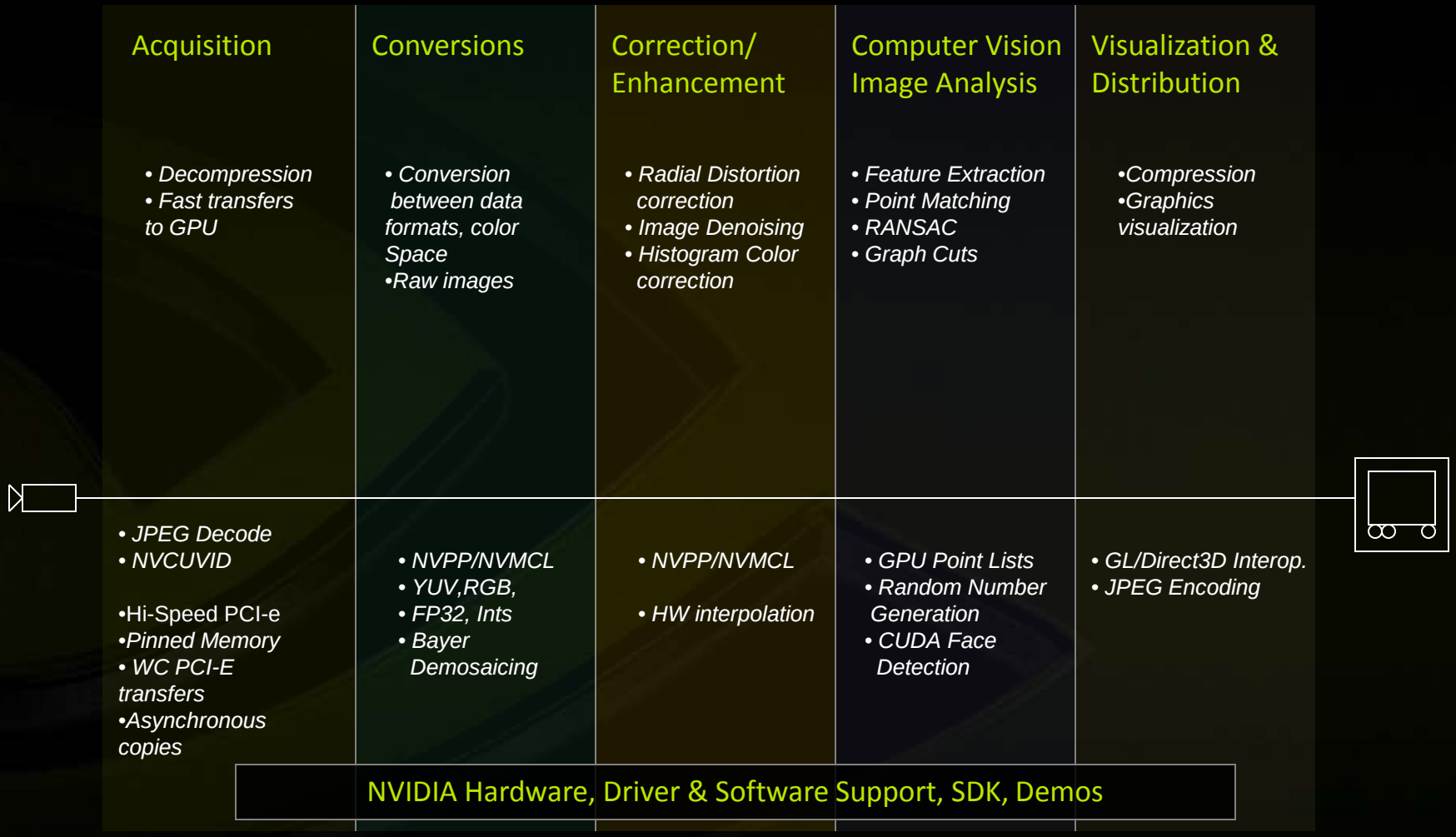
Mohamed Hussein, Amitabh Varshney, and Larry Davis, “On Implementing Graph Cuts on CUDA,” First Workshop on General Purpose Processing on Graphics Processing Units 2007.

Vibhav Vineet, P. J. Narayanan, "CUDA cuts: Fast graph cuts on the GPU," ***Computer Vision and Pattern Recognition Workshop***, pp. 1-8, 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008.

3rd Party Research & Libraries

- Level-Set segmentation with CUDA
  - http://code.google.com/p/cudaseg/
- Video segmentation with CUDA
  - http://code.google.com/p/gpuwire/
- Multiclass SVM implementation in CUDA
  - http://code.google.com/p/multisvm/
- CUDA implementation of pedestrian detection algorithm – MIT Project
  - http://code.google.com/p/cudapeddet/
- GPU4Vision
  - http://gpu4vision.icg.tugraz.at/
- MinGPU: A minimum GPU library for Computer Vision
  - http://server.cs.ucf.edu/~vision/MinGPU/
- CUDA SIFT
  - http://www.csc.kth.se/~celle/
  - http://www.cs.unc.edu/~ccwu/siftgpu/
- 90+ Imaging Papers/Resources at http://www.nvidia.com/cuda
- … lots more out there!

GPU Imaging

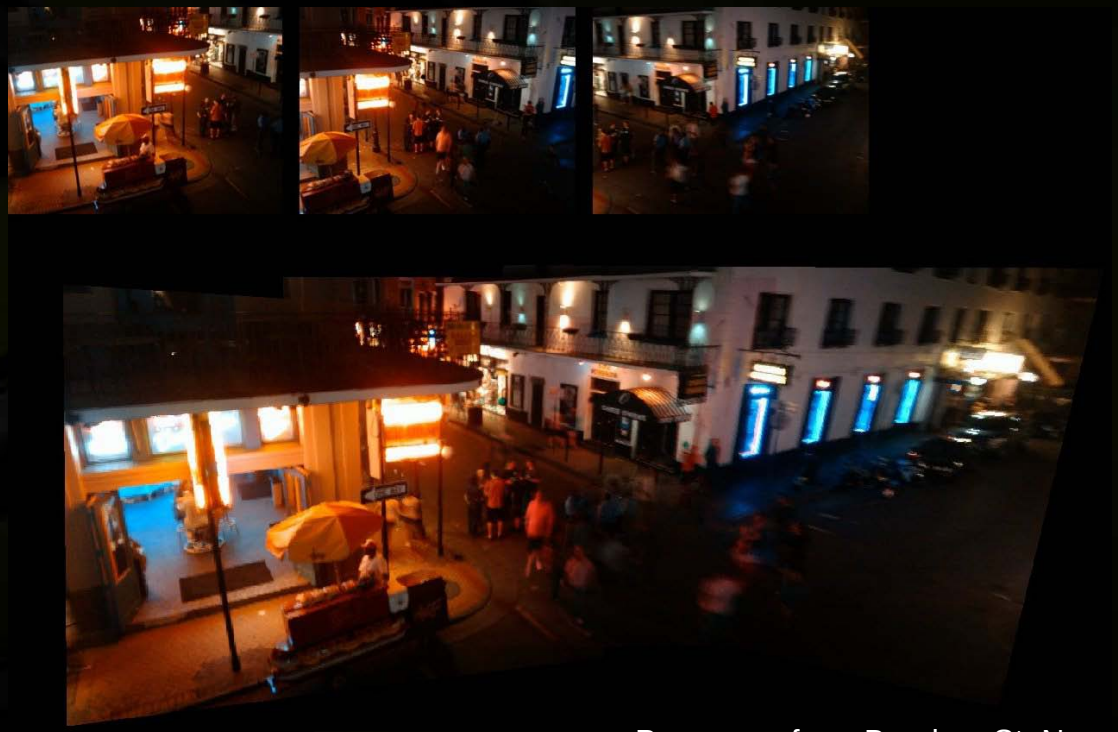

Acquisition
• Decompression
• Fast transfers to GPU
Conversions
• Conversion between data formats, color Space
•Raw images
Correction/ Enhancement
• Radial Distortion correction
• Image Denoising
• Histogram Color correction
Computer Vision Image Analysis
• Feature Extraction
• Point Matching
• RANSAC
• Graph Cuts
Visualization & Distribution
•Compression
•Graphics visualization
• JPEG Decode
• NVCUVID
•Hi-Speed PCI-e
•Pinned Memory
• WC PCI-E transfers
•Asynchronous copies
• NVPP/NVMCL
• YUV,RGB,
• FP32, Ints
• Bayer Demosaicing
• NVPP/NVMCL
• HW interpolation
• GPU Point Lists
• Random Number Generation
• CUDA Face Detection
• GL/Direct3D Interop.
• JPEG Encoding
NVIDIA Hardware, Driver & Software Support, SDK, Demos

GPU Imaging & Vision Summary

- The Quality vs. Performance tradeoff is changing.
- CUDA's programming flexibility leads to more algorithms mapping efficiently on the GPU
  - Thread Cooperation: shared memory, synchronizations
- Non-Pixel operations mean fully GPU resident pipelines
  - Point Lists
  - Feature Processing
  - FFT
  - Random number generation (RANSAC?)
- New libraries & research on CUDA available now
  - NVPP, NVIDIA SDK, Academia

Thank You!

- See http://www.nvidia.com/cuda for complete information
- Questions?



Panorama from Bourbon St. New Orleans, SIGGRAPH 2009

GPU TECHNOLOGY CONFERENCE

GPU Technology Conference

Sept 30 – Oct 2, 2009 – The Fairmont San Jose, California

Learn about the latest breakthroughs developers, engineers and researchers are achieving on the GPU

- Learn about the seismic shifts happening in computing
- Preview disruptive technologies and emerging applications
- Get tools/techniques to impact mission critical projects now
- Network with experts and peers from across several industries

Special for SIGGRAPH Attendees!

Full Conference Pass for only $400!

(offer expires 8/21)

Register and Learn More: www.nvidia.com/gtc

Use registration code FC300SIG90

NVIDIA.

GPU TECHNOLOGY CONFERENCE

Confirmed Sessions @ GPU Tech Conf

Sept 30 – Oct 2, 2009 – The Fairmont San Jose, California

KEYNOTES:

Opening

Jen-Hsun Huang
Co-Founder / CEO
NVIDIA

Day 2

Hanspeter Pfister
Professor
Harvard

Day 3

Richard Kerris
CTO
Lucasfilm

GENERAL SESSIONS:

- **Hot Trends in Visual Computing: Computer Vision, Augmented Reality, Visual Analytics, Interactive Ray Tracing**
- **Breakthroughs in High Performance Computing: Energy, Medical Science, Supercomputing, Research**

Learn more and register: www.nvidia.com/gtc

Conference Sessions Covered Topics:

- 3D
- Algorithms & Numerical Techniques
- Astronomy & Astrophysics
- Computational Finance
- Computational Fluid Dynamics
- Computational Imaging
- Computer Vision
- Databases & Data Mining
- Embedded & Mobile
- Energy Exploration
- Film
- GPU Clusters
- High Performance Computing
- Life Sciences
- Machine Learning & Artificial Intelligence
- Medical Imaging & Visualization
- Molecular Dynamics
- Physical Simulation
- Programming Languages & APIs
- Advanced Visualization

NVIDIA.