GPU TECHNOLOGY CONFERENCE

Fast High-Quality Panorama Stitching

GTC 2010 , San Jose | James Fung & Timo Stich, NVIDIA

PRESENTED BY NVIDIA.

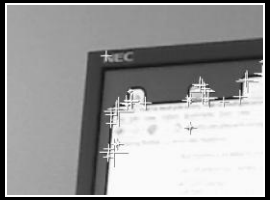
GPU Panorama Stitching

GPU Panorama Stitching

Image Pipeline: Panorama Stitching

Left Image

Right Image

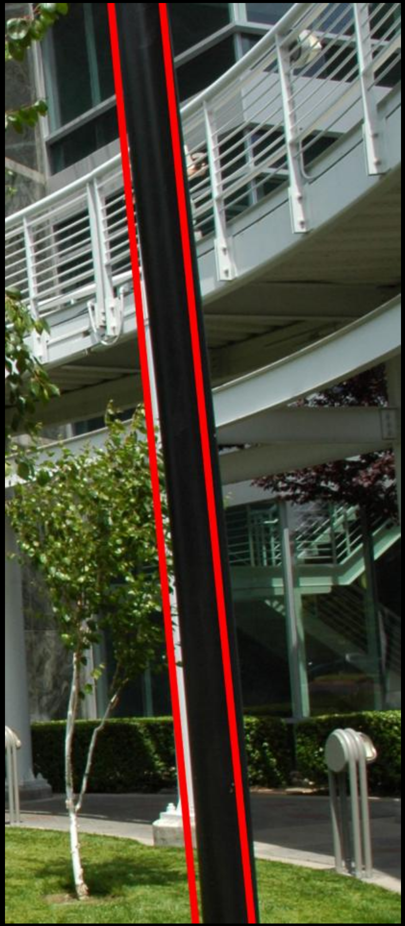


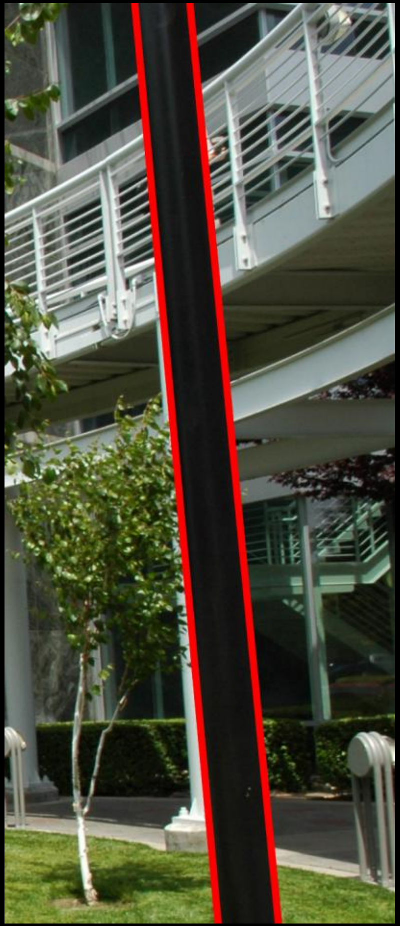
Radial Distortion Correction → **Keypoint Detection & Extraction (Shi-Tomasi/SIFT)** → Keypoint Matching → Recover Homography (RANSAC) → Projective Transform → Image Stitching

Radial Distortion Removal

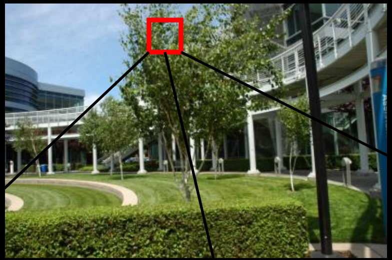
$$\delta x = x(\kappa_1 r^2 + \kappa_2 r^4 + \ldots)$$
$$\delta y = y(\kappa_1 r^2 + \kappa_2 r^4 + \ldots)$$

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**

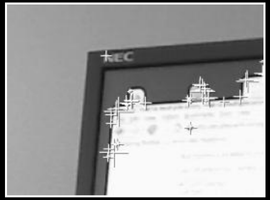
Input Image: 3008x2000 (6MP) RGB

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

Image Pipeline: Panorama Stitching

Left Image 

Right Image 


Radial Distortion Correction → **Keypoint Detection & Extraction (Shi-Tomasi/SIFT)** → Keypoint Matching → Recover Homography (RANSAC) → Projective Transform → Image Stitching

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)

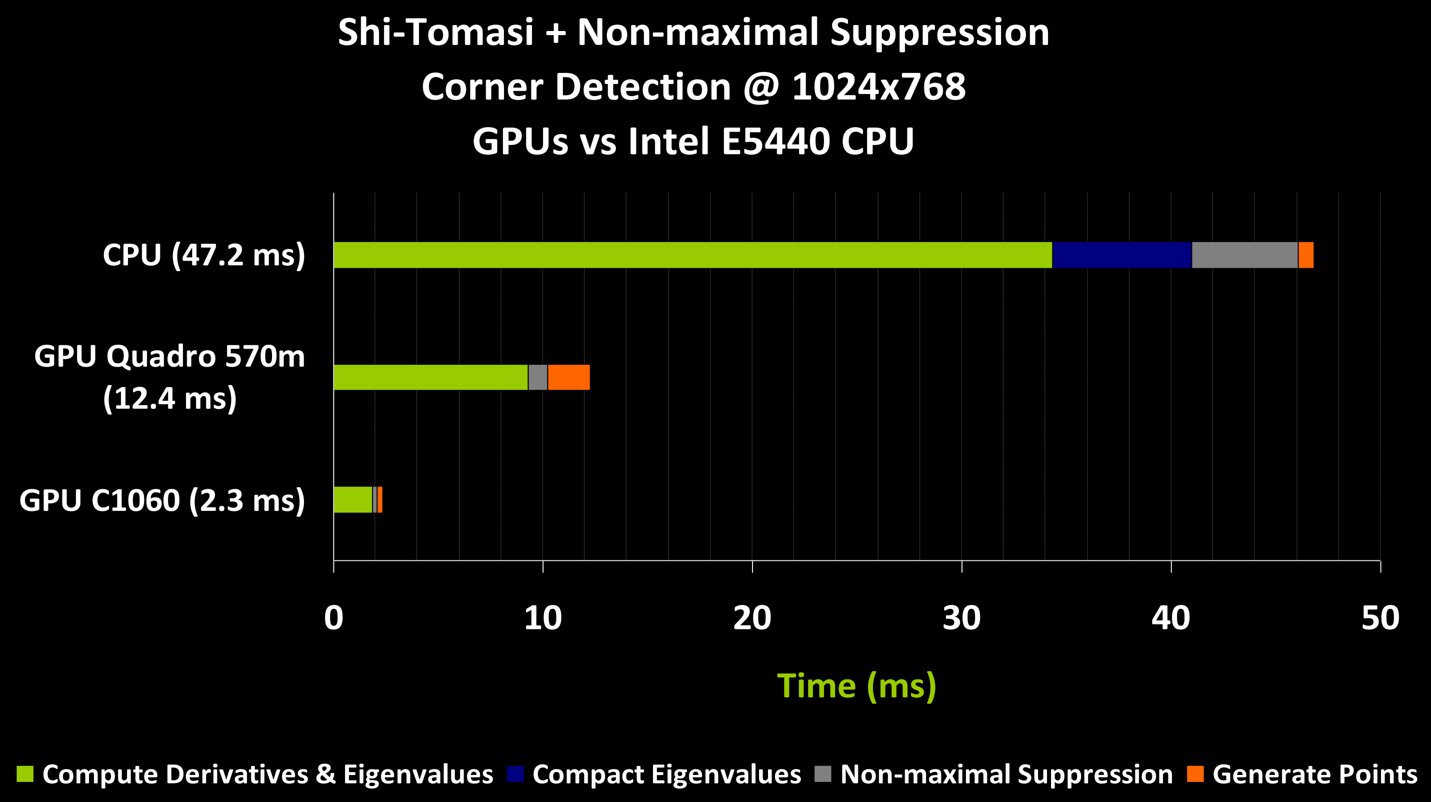
Harris: Compute Mc, (based on Eigenvalues of A, λ_1, and λ_2) by computing the determinant and trace of A

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

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

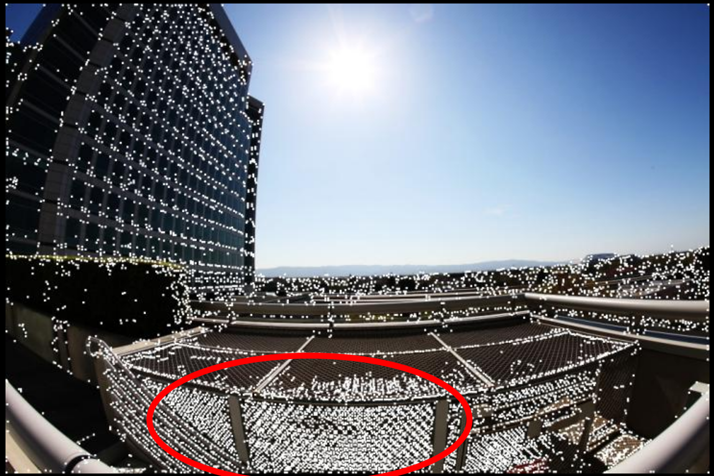
$$\lambda_1, \lambda_2 = \frac{\operatorname{tr}(A) \pm \sqrt{\operatorname{tr}(A)^2 - 4\det(A)}}{2}$$

Feature Detection



Corner Detection

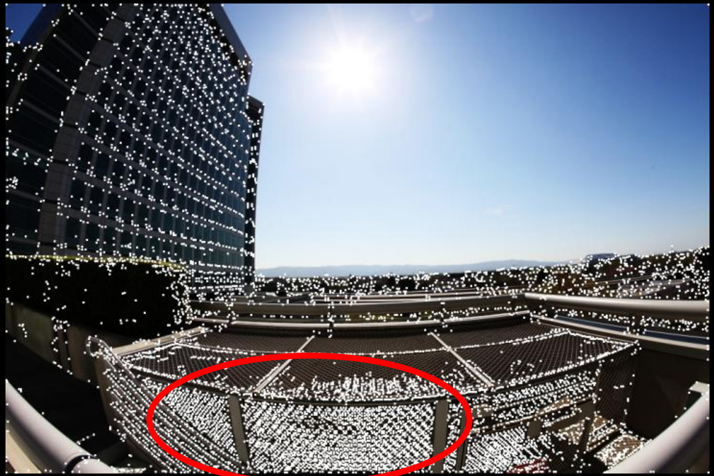
Corner Detection

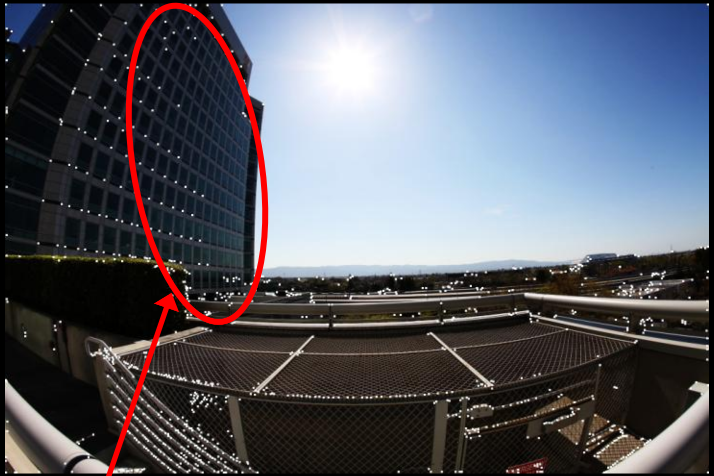


Indistinct

- No single threshold can eliminate clutter and maintain weaker features

Corner Detection



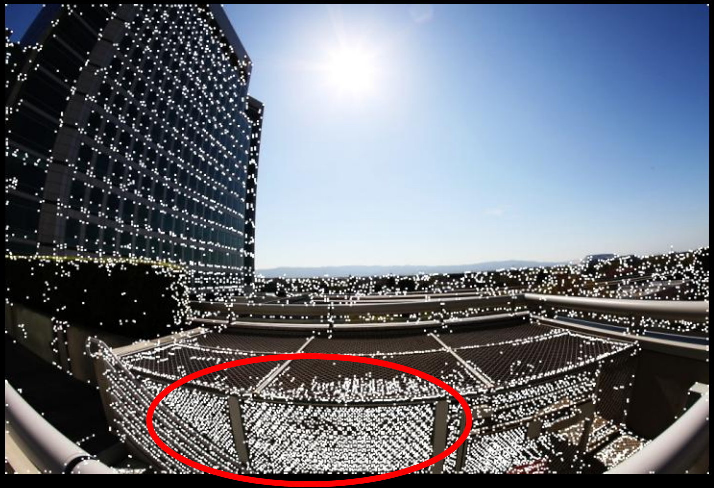


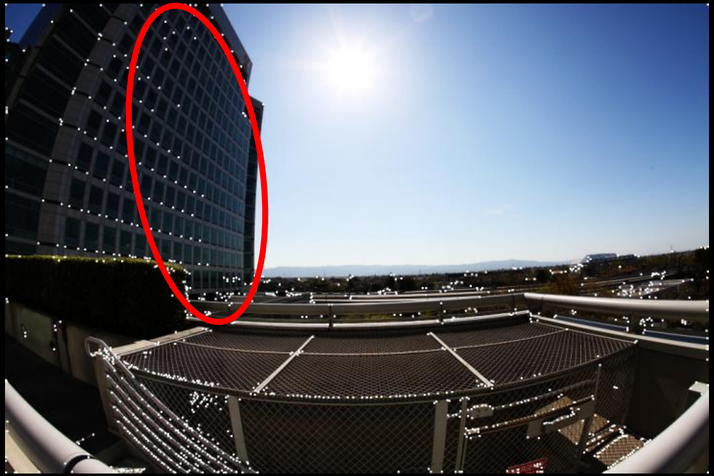
Indistinct cluttered features

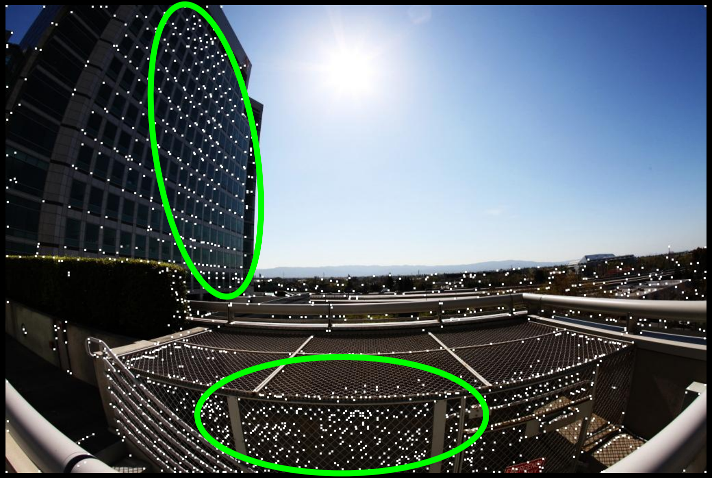
Loss of salient points

- No single threshold can eliminate clutter and maintain weaker features

Modified Shi-Tomasi







- Take ratio of min(λ1, λ2) 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

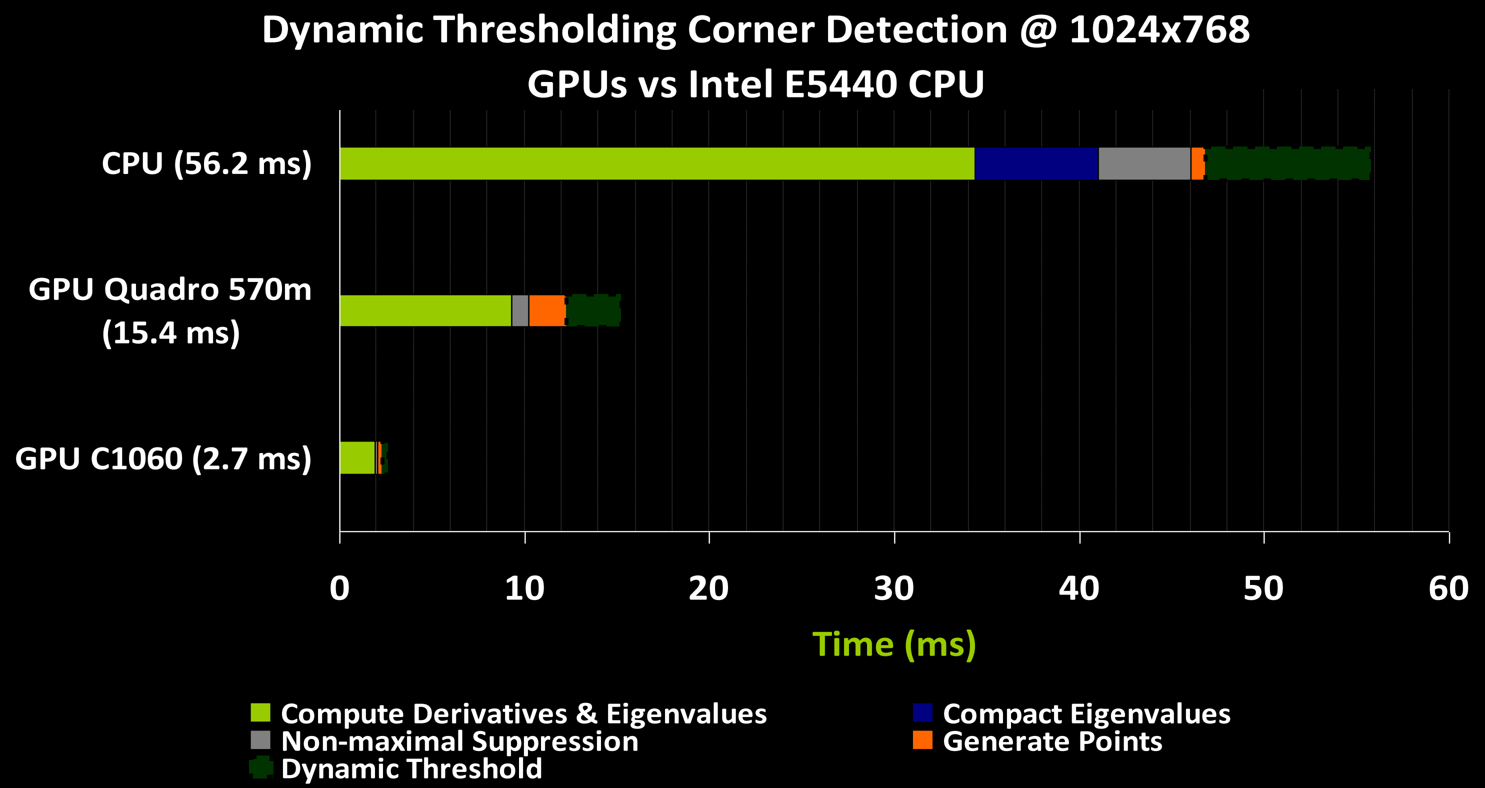
$$\lambda_1, \lambda_2 = \frac{\mathrm{tr}(A) \pm \sqrt{\mathrm{tr}(A)^2 - 4\det(A)}}{2}$$

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



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

www.mpii.de/~gziegler

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)

$$
\begin{matrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{matrix}
\qquad
\begin{matrix}
2 & 1 & 2 & 1 \\
0 & 1 & 0 & 2 \\
1 & 1 & 0 & 4 \\
0 & 0 & 0 & 2
\end{matrix}
$$

- 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)

$$
\begin{matrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{matrix}
\quad \longrightarrow \quad
\begin{matrix}
2 & 1 & 2 & 1 \\
0 & 1 & 0 & 2 \\
1 & 1 & 0 & 4 \\
0 & 0 & 0 & 2
\end{matrix}
\quad \longrightarrow \quad
\begin{matrix}
4 & 5 \\
2 & 6
\end{matrix}
$$

- 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

4	5
2	6

17

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

Holds points 1..4

Holds points 5..9:
Point 5 must be inside (below) this quadrant

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

1	0	1	0	$\boxed{1}$	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	$\boxed{2}$	1
0	1	0	2
1	1	0	4
0	0	0	2

4	$\boxed{5}$
2	6

17

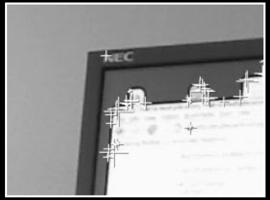
Holds Points 5,6: point 5 must be inside this quadrant

This must be point 5

Image Pipeline: Panorama Stitching

Left Image

Right Image



Radial Distortion Correction → **Keypoint Detection & Extraction (Shi-Tomasi/SIFT)** → Keypoint Matching → Recover Homography (RANSAC) → Projective Transform → Image Stitching

Generating Descriptors

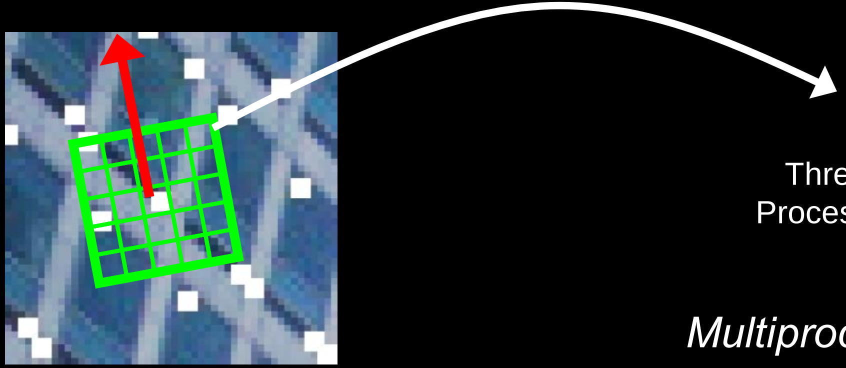
- What's a Feature Descriptor?
  - Distinct numerical representation of an image point for matching
- Our example: SIFT
  - "Scale Invariant Feature Transform"
  - 128 element floating point vector ("key")
  - Nearest Euclidean distance between keys is the best match

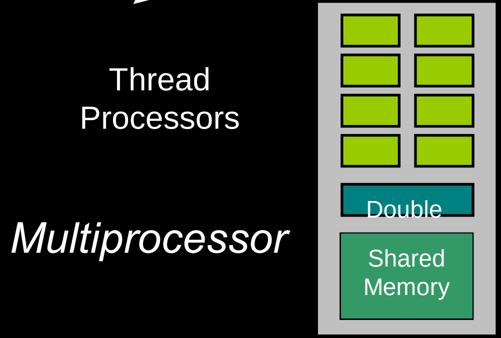
Distinctive Image Features from Scale-Invariant Keypoints

Generating Descriptors

- Sparse point processing
- HistoPyramid tree organization gives good spatial locality!
- Parallel processing of single descriptor with thread cooperation
  - Shared Memory
  - Thread Synchronization

Feature Descriptor Computation

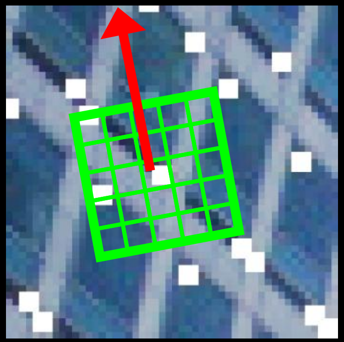


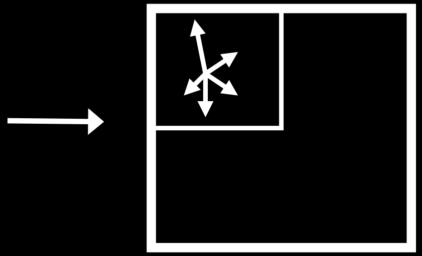


1. Calculate feature orientation *(shared memory reduction)*
2. Lookup rotated samples (texture cache)

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

Feature Descriptor Computation

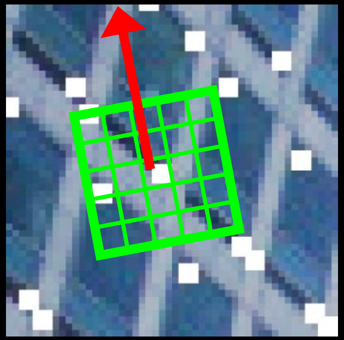




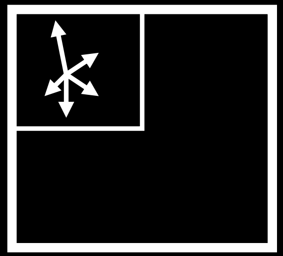
1. Calculate feature orientation *(shared memory reduction)*
2. Lookup rotated samples (texture cache)
3. Generate local orientation histograms (one thread per histogram,shared memory, pointer indexing)

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

Feature Descriptor Computation



→



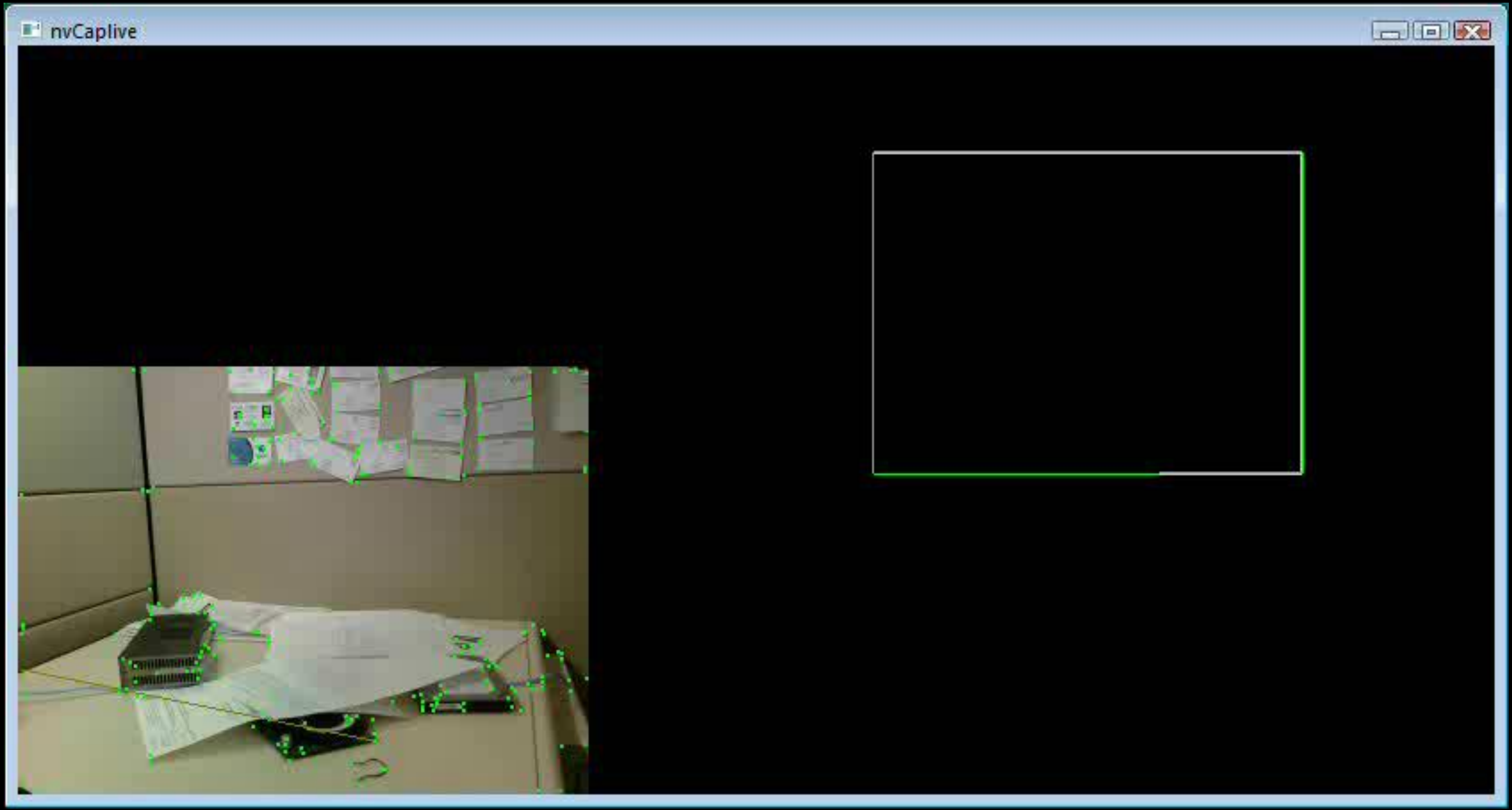
→

0.031714 0.087833 0.027565
0.000000 0.005982 0.115469
0.132375 0.026356 …

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)

Matching Results



GPU TECHNOLOGY CONFERENCE

Image Pipeline: Panorama Stitching

Left Image

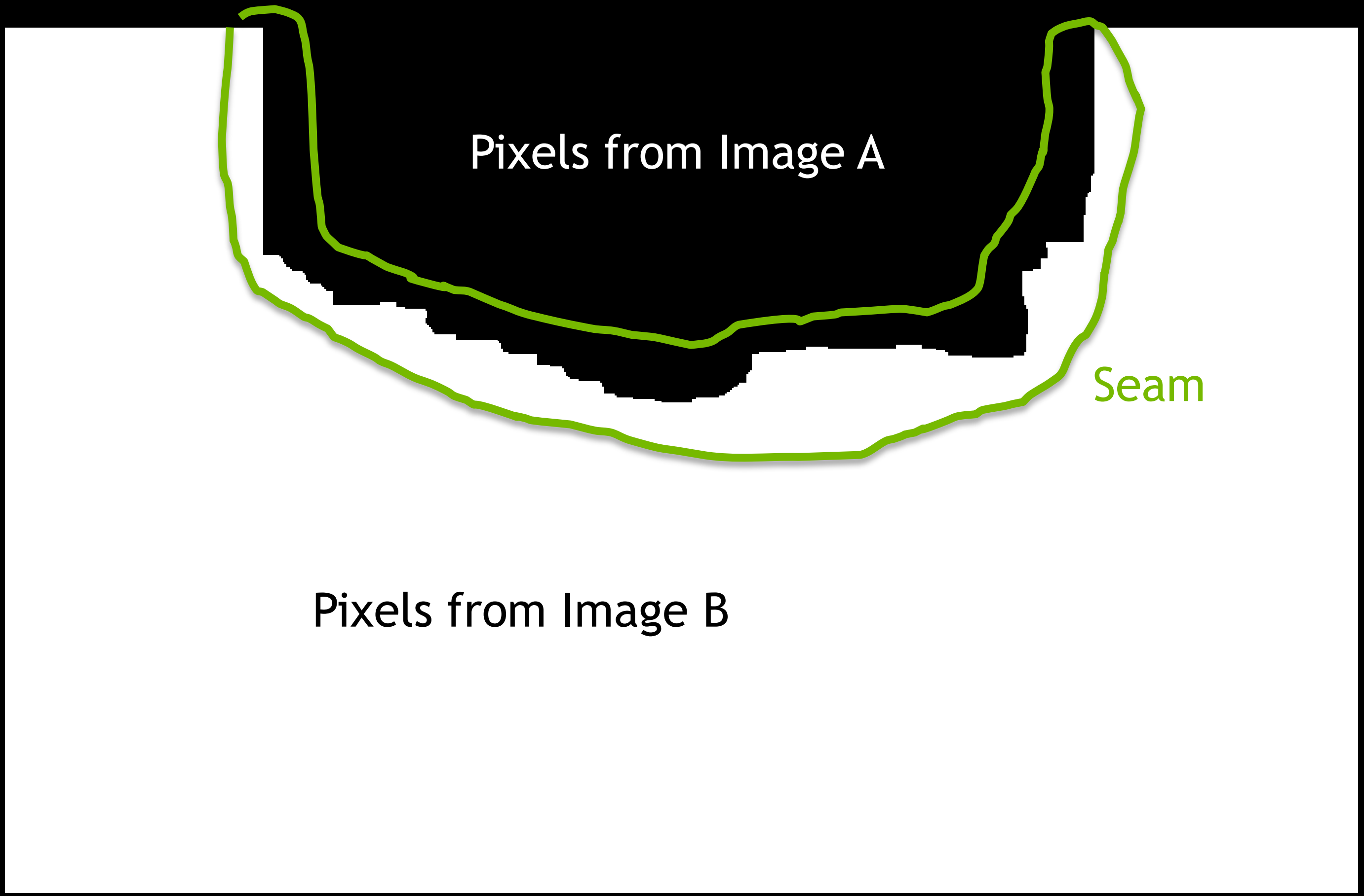
Right Image

Radial Distortion Correction → **Keypoint Detection & Extraction (Shi-Tomasi/SIFT)** → Keypoint Matching → Recover Homography (RANSAC) → Projective Transform → Image Stitching

Blending can cause artifacts

Better: Image Stitching

Binary Labeling



What defines a good seam?

- Intuition: Must not be noticeable
  - Avoid introducing new gradients
- Breaking it down on the pixel level:
  - Color differences between pixels of images should be minimal at seam pixels

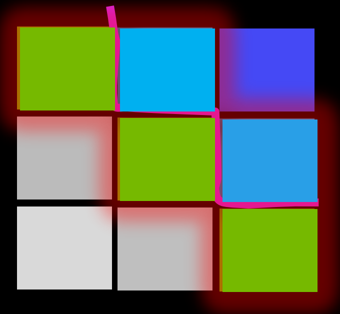


Image A

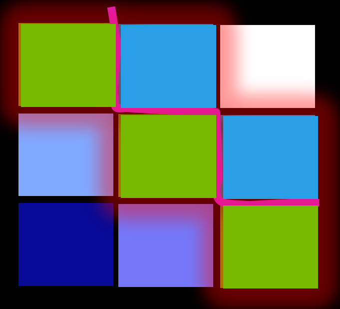
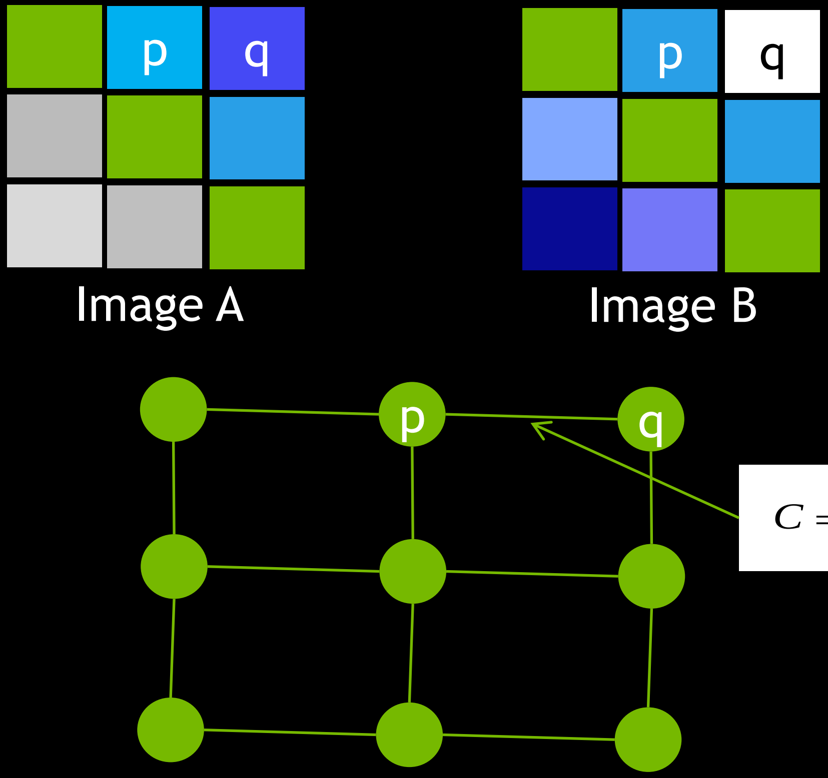


Image B

Computing good seams

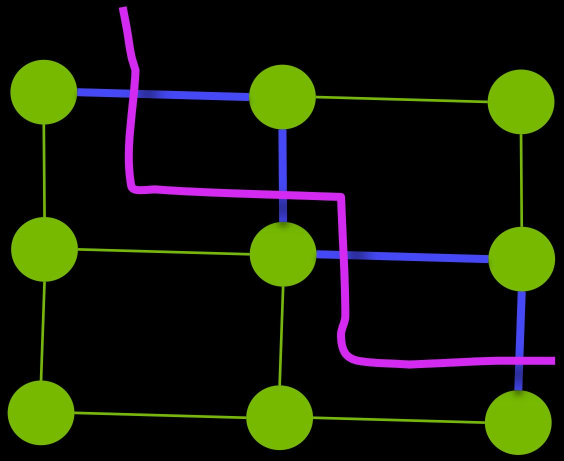


$$C = |I_A(p) - I_B(p)|^2 + |I_A(q) - I_B(q)|^2$$

Markov Random Field

Computing good seams

- Graph Cut to compute the minimal cost seam
  - Very fast, global optimal solver for binary label problems
  - NPP primitive (upcoming 3.2 release)



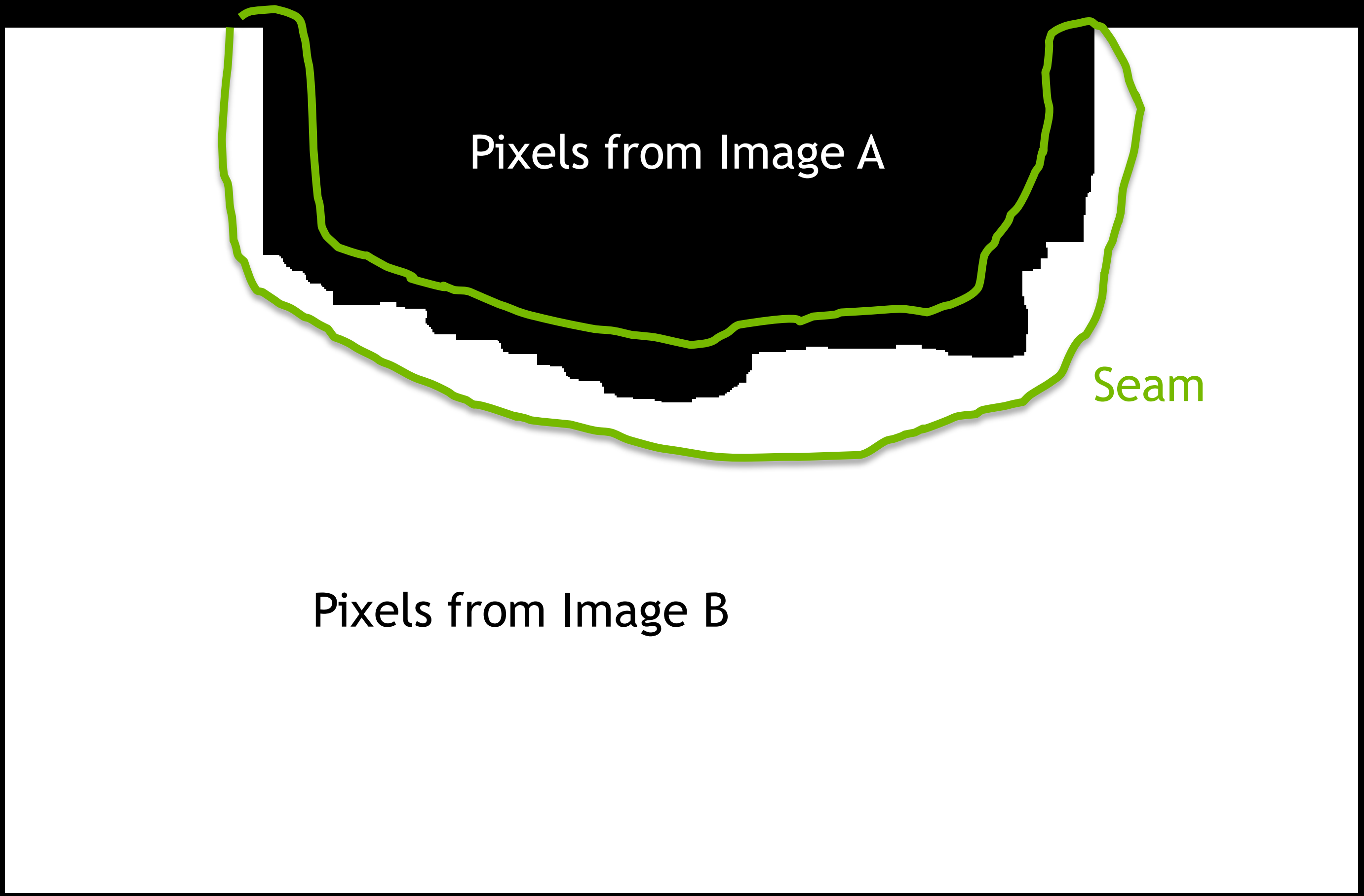
Best seam is the Minimum Cut

NPP Graphcut

- Graph is stored in Arrays
  - Weights to terminals in one array
  - Weights to neighbors in four arrays, horizontal edges are transposed

- `nppiGraphcut_32s8u(d_terminals, d_left_transposed, d_right_transposed, d_top, d_bottom, step, transposed_step, size, d_labels, label_step, pBuffer);`

Binary Labeling



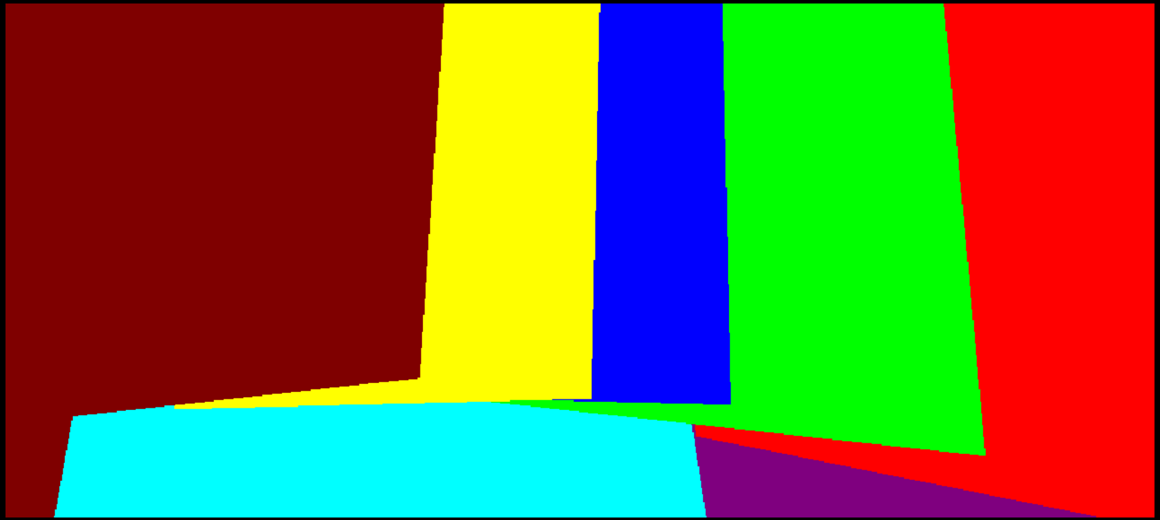
From two images to many

- One label for each image in the set: N labels
  - Assign each pixel in the panorama one label
  - Unfortunately this is NP hard problem ☹
- Alpha-Expansion algorithm to the rescue
  - Intuition: “Keep current label or change to alpha”
  - Again binary problem solvable with Graph Cut
  - Repeat for all labels until the optimal solution is found

Alpha-Expansion

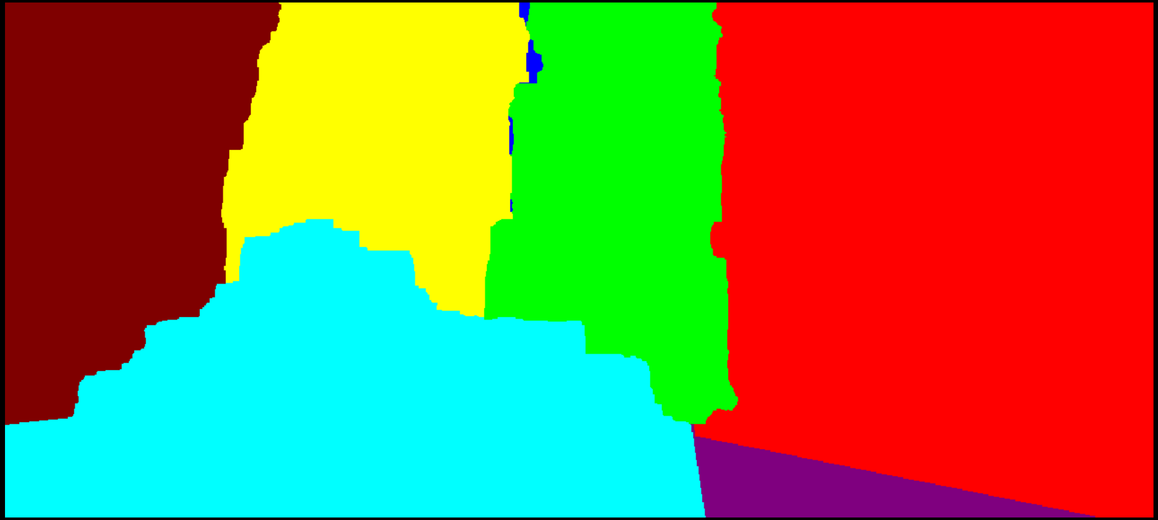
- Iteration for each label alpha:
  - Compute data term for alpha
  - Compute neighborhood terms for alpha
  - Solve binary Graph Cut
  - Use binary solution to get expanded multi-label solution
  - Compute total energy of expanded solution
  - If energy has decreased make this the current solution, otherwise discard

Alpha-Expansion



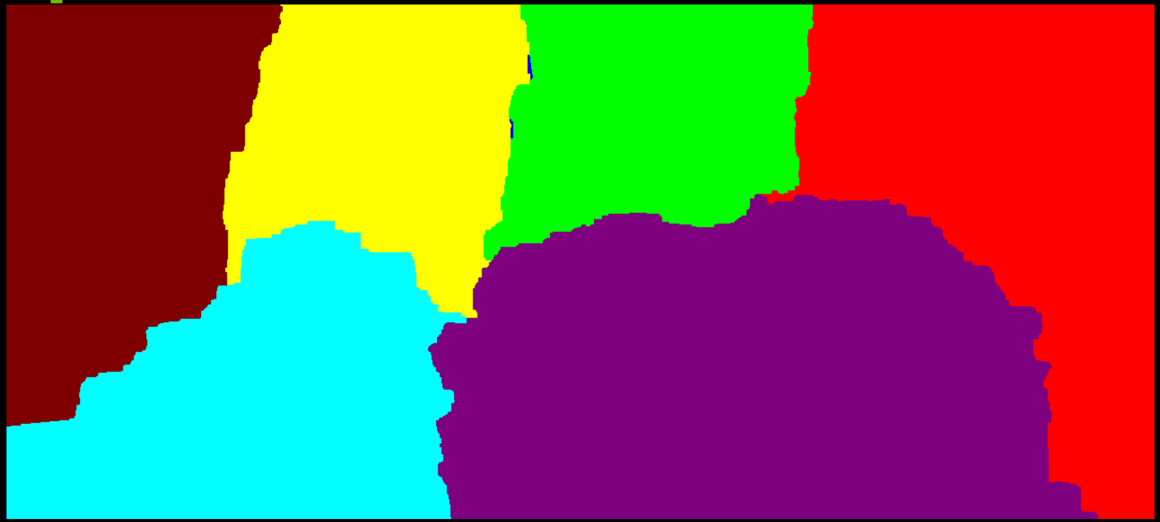
Initial Solution

Alpha-Expansion



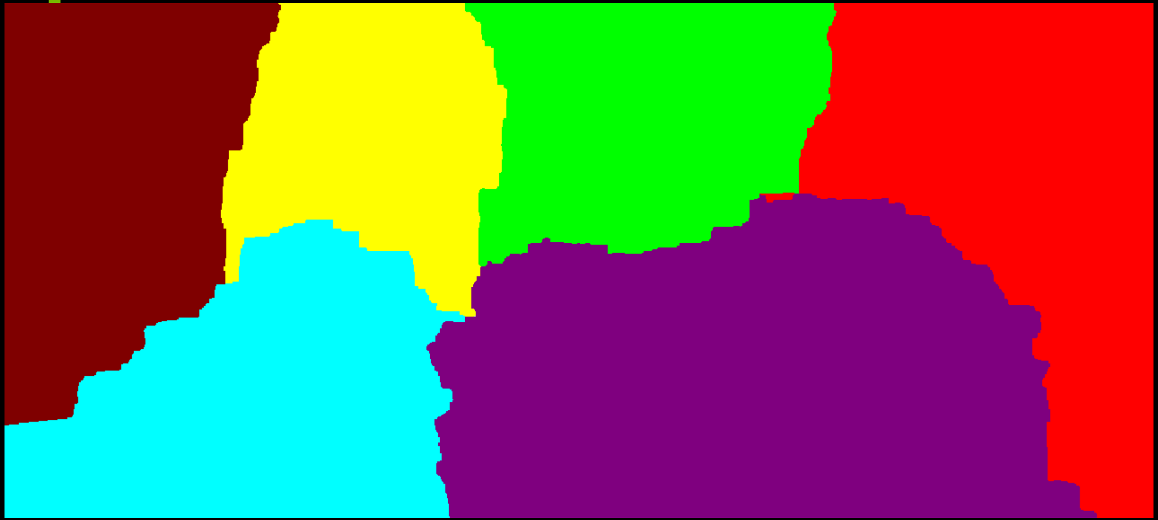
After 6 Expansion Steps

Alpha-Expansion



After 12 Expansion Steps

Alpha-Expansion



After 18 Expansion Steps

Final Result

Image Stitching Notes

- In this example only 6 out of the 7 input images contribute to the final result
  - Image Stitching reduces the image set

- Quality improves with each iteration
  - The current result is a preview that converges to the final result

Performance

Test data set

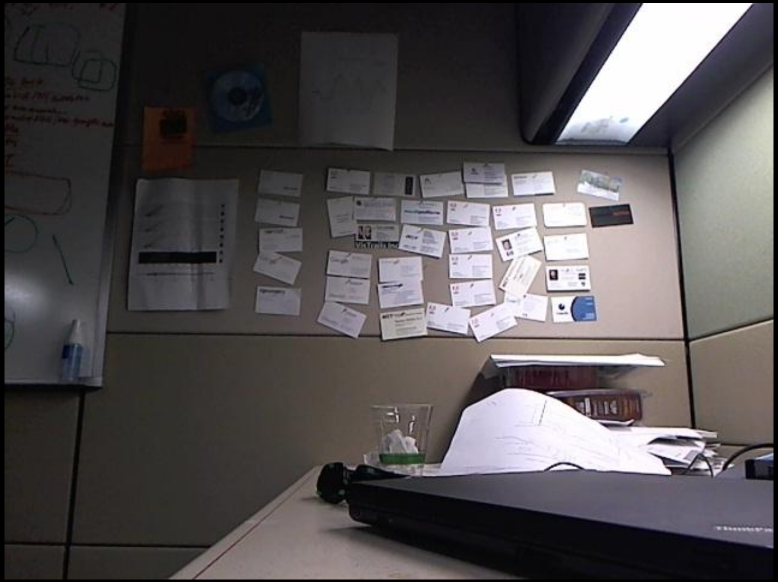


Image 0

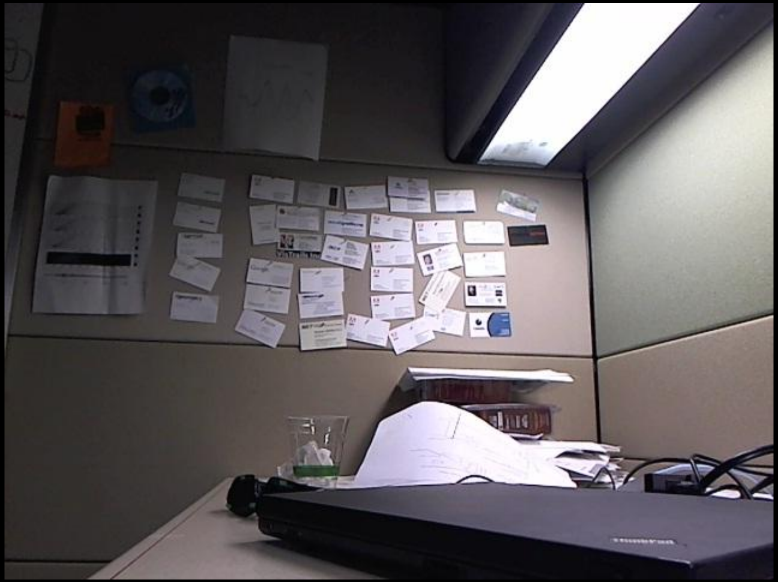


Image 1

Performance

- CUDA 3.0, Driver 260.16 wall clock times

Alpha Expansion Performance

Iteration	Time (ms)	
	GTX460	GTX 480
1	101.74	50.44
2	102.85	50.77
3	31.95	18.95
4	28.59	18.68
5	28.18	18.82
6	28.92	18.59

Image Pipeline: Panorama Stitching

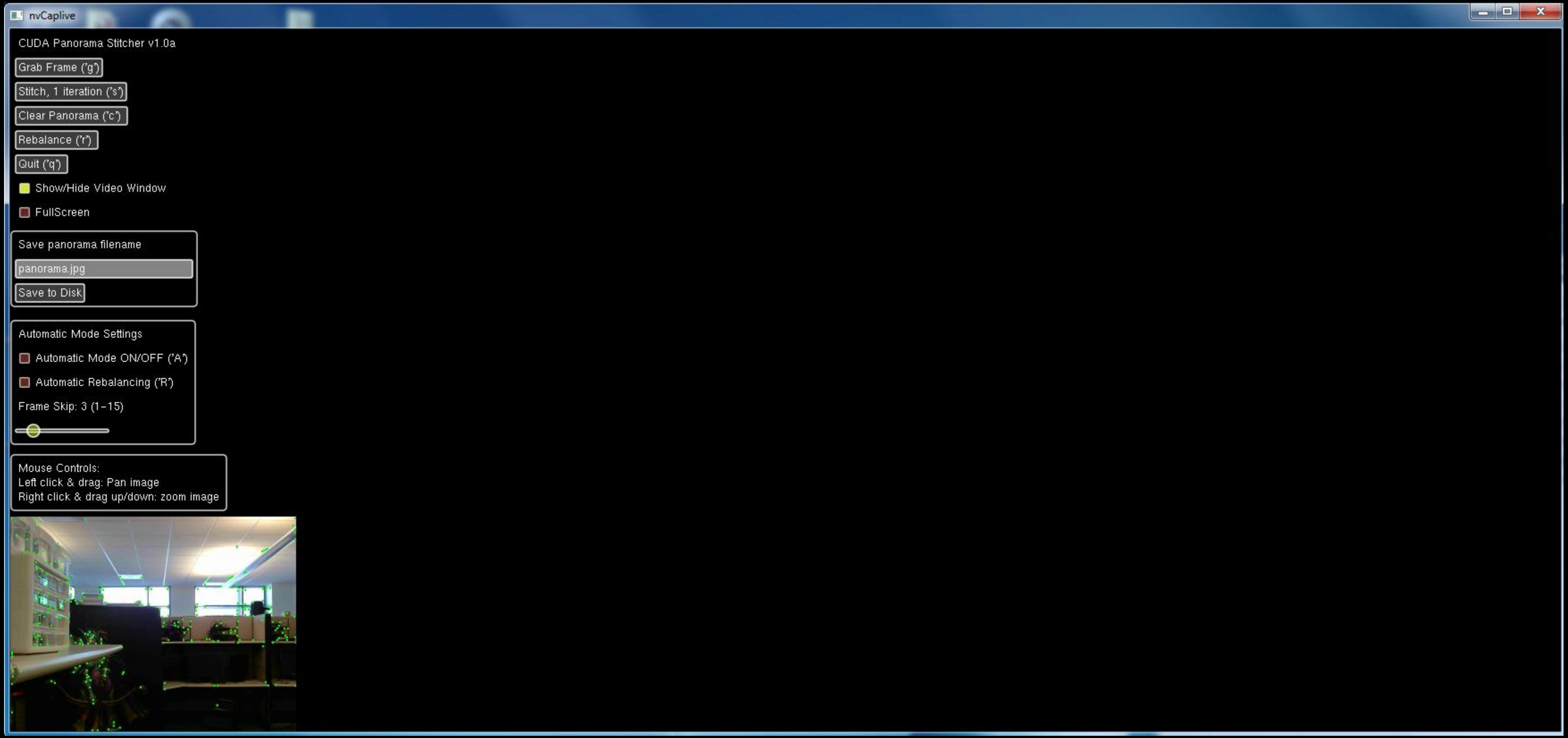
Left Image

Right Image

Radial Distortion Correction → **Keypoint Detection & Extraction (Shi-Tomasi/SIFT)** → Keypoint Matching → Recover Homography (RANSAC) → Projective Transform → Image Stitching

GPU TECHNOLOGY CONFERENCE

Example Application



Questions?

GPU TECHNOLOGY CONFERENCE

PRESENTED BY NVIDIA.