SS12

GPU Programming Lab Project

Optical Flow and Super Resolution

Ross Kidson 03627521
Oliver Dunkley 03631802

Introduction

Contents

- Implementation environment, methods
- Optical Flow
  - example
  - theory
  - implementation
  - results & performance
- Super Resolution
  - theory
  - implementation
  - results

Implementation

- nsight, ubuntu
- two code bases
- focused on benchmarking/comparing memory types
- python scripts & open office

Benchmarking methods

- GPU - NVidia Visual Profiler
- CPU - custom 'benchmark' class to reproduce the same data

```
Benchmark::instance()->addEvent(Benchmark::start, "add_flow_fields");
  for(unsigned int p=0;p<nx_fine*ny_fine;p++){
     _u1[p] += _u1lvl[p];
     _u2[p] += _u2lvl[p];
}
Benchmark::instance()->addEvent(Benchmark::end, "add_flow_fields");

     .
     .
     .

Benchmark::instance()->doBenchmark(); //Save collected data to file
```

Optical flow

"Pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and the scene"

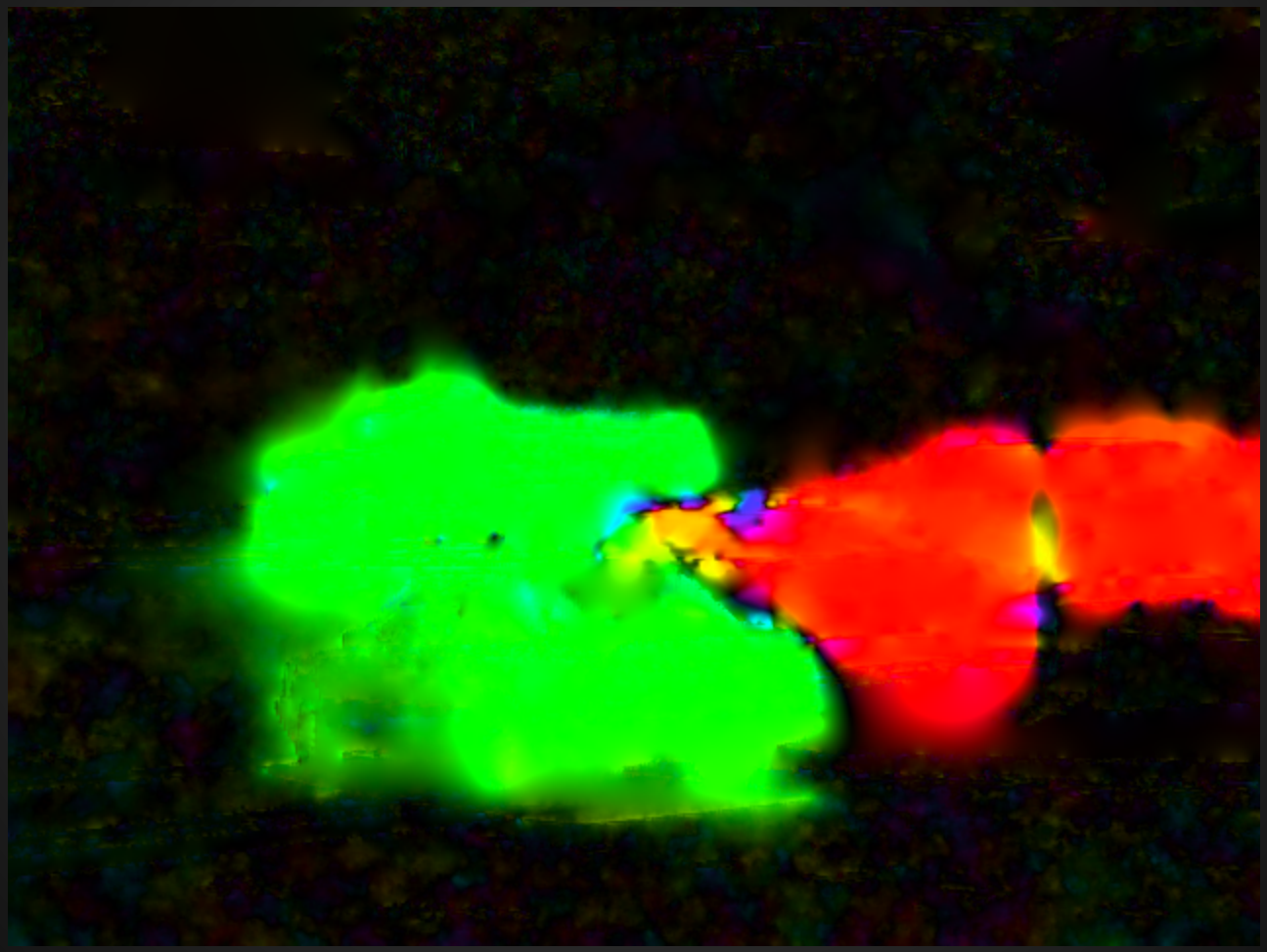
Usages

- motion detection
- object segmentation
- time-to-collision
- motion compensated encoding
- stereo disparity measurement

Optical flow: Example input street 1

Optical flow: Example input street 2

Optical flow: Example output



Optical flow

Optical flow Example input Tron 1

Optical flow Example input Tron 2

Optical flow Overlayed output

Optical flow: Formulation

Given two images I_1 and I_2 one computes a field vector u that matches image intensities:

$$I_2(\mathbf{x} + u(\mathbf{x})) = I_1(\mathbf{x})$$

Formulate as an energy function to minimize

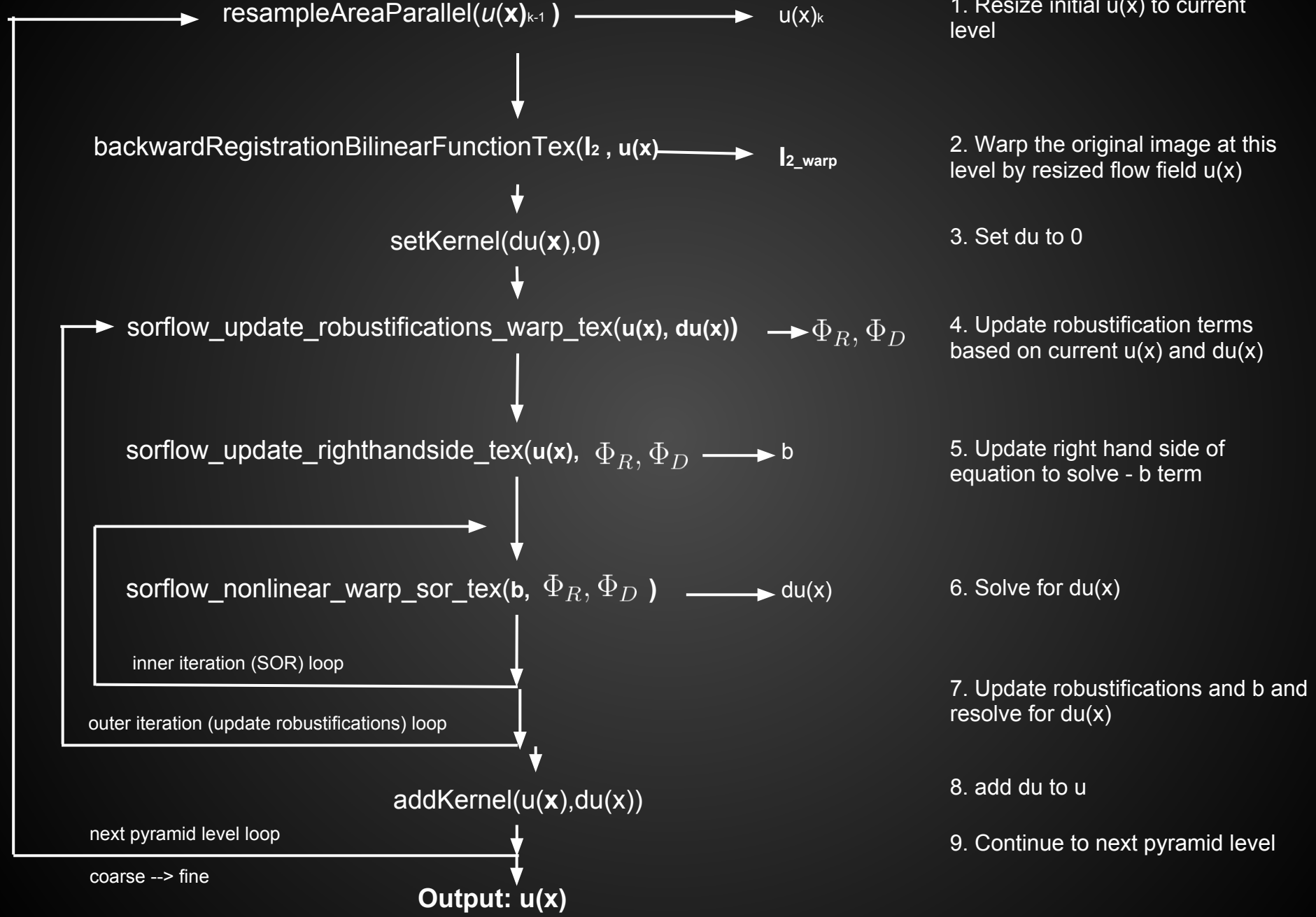
$$E(u) = \int_\Omega (I_2(\mathbf{x} + u(\mathbf{x})) - I_1(\mathbf{x}))^2 + \lambda \mid \bigtriangledown u \mid^2 d\mathbf{x}$$

Optical flow: Formulation

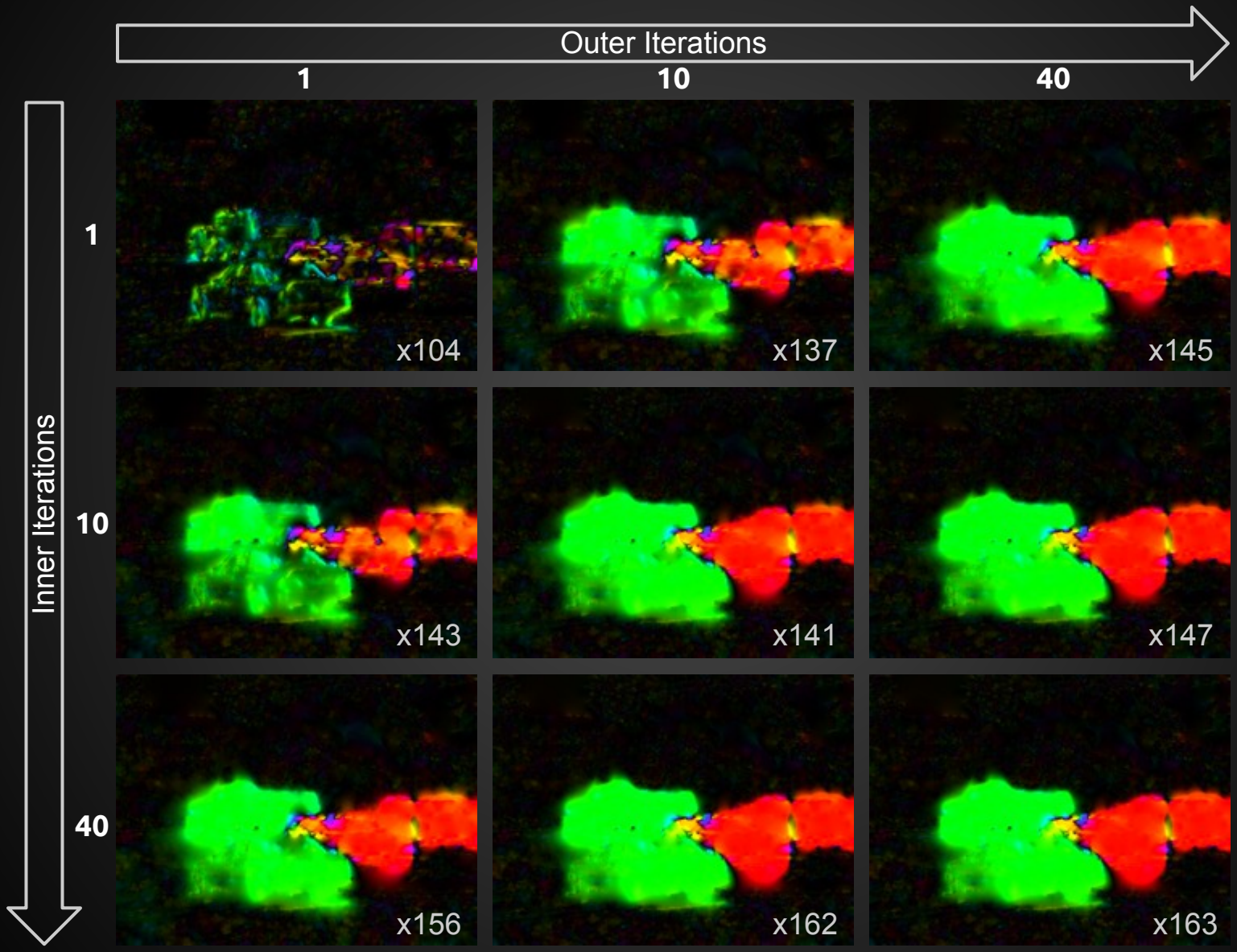
Apply taylor series expansion and robust penalty term: $\Phi(s^2) = \sqrt{\epsilon + s^2}$

$$E(u) = \int_{\Omega} (\Phi_D((\bigtriangledown I^T u + I_t)^2) + \lambda \Phi_R(|\bigtriangledown u|^2) d\mathbf{x}$$

This can be solved with Euler-Lagrange equations and reformulated into ax = b form, which can then be solved using SOR

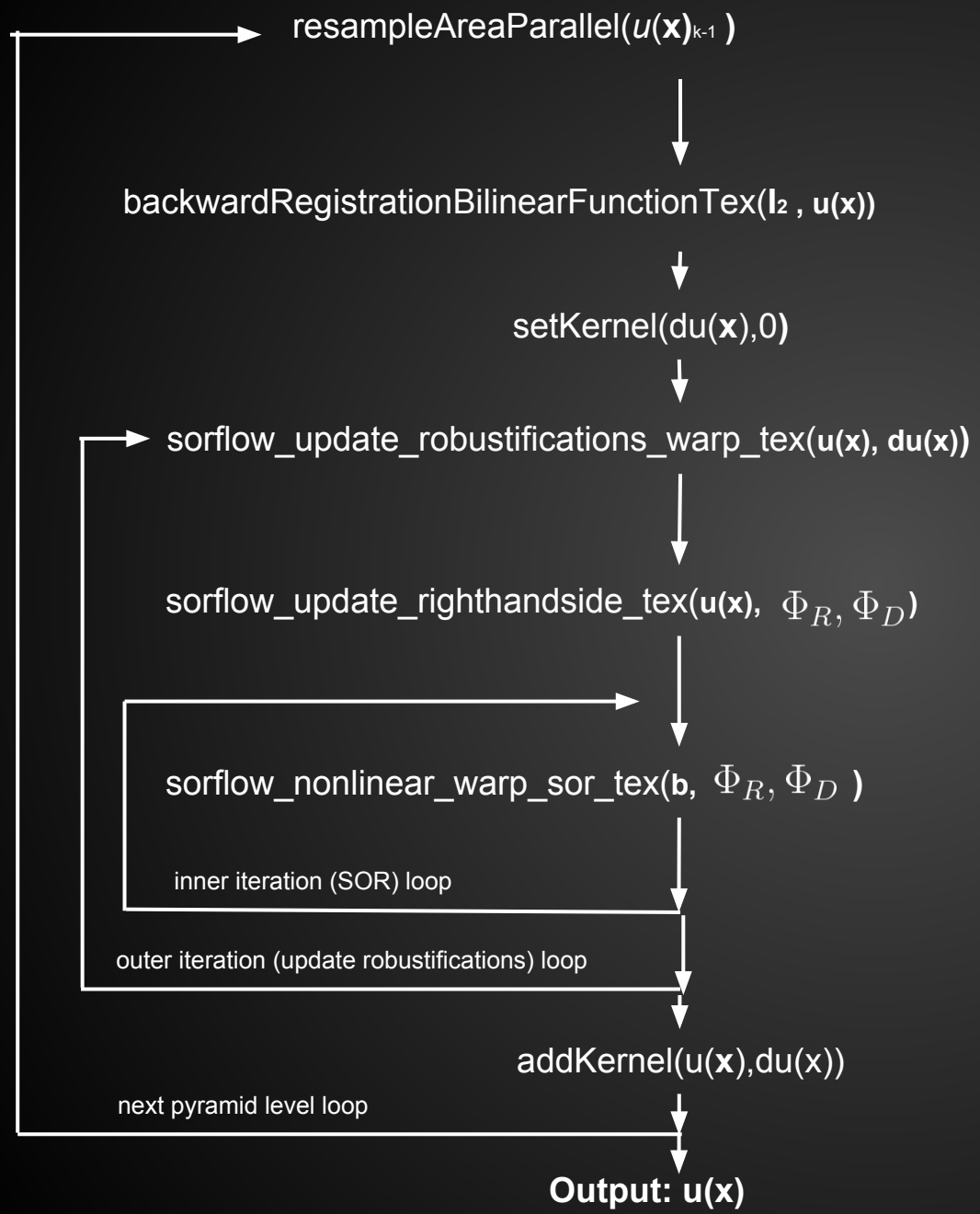

resampleAreaParallel(u(x)k-1)
u(x)k
1. Resize initial u(x) to current level
backwardRegistrationBilinearFunctionTex(I2 , u(x)
I2_warp
2. Warp the original image at this level by resized flow field u(x)
setKernel(du(x),0)
3. Set du to 0
sorflow_update_robustifications_warp_tex(u(x), du(x))
ΦR, ΦD
4. Update robustification terms based on current u(x) and du(x)
sorflow_update_righthandside_tex(u(x), ΦR, ΦD
b
5. Update right hand side of equation to solve - b term
sorflow_nonlinear_warp_sor_tex(b, ΦR, ΦD)
du(x)
6. Solve for du(x)
inner iteration (SOR) loop
7. Update robustifications and b and resolve for du(x)
outer iteration (update robustifications) loop
addKernel(u(x),du(x))
8. add du to u
next pyramid level loop
9. Continue to next pyramid level
coarse --> fine
Output: u(x)

Optical Flow Results



Applying GPU Techniques

- registers
- constant memory
- shared memory
- texture memory
- global memory
- kernels, blocks, pitch, warp



Variables tested

Already in Texture Memory

- Image gradients Ix, Iy, It
- backwardRegistration - $\mathbf{I_2}$

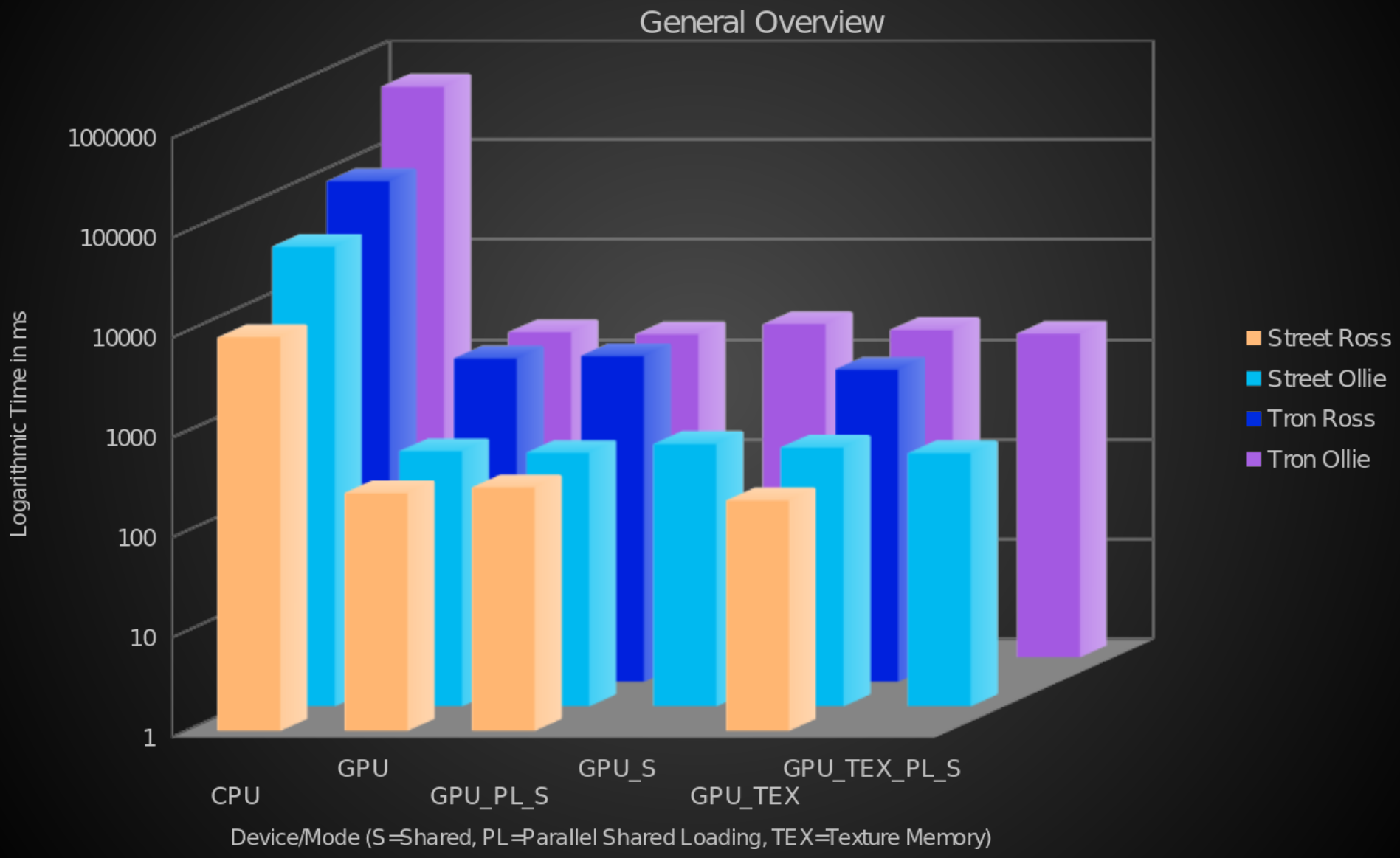
Added to Texture Memory

-
- $\Phi_R(x), \Phi_D$

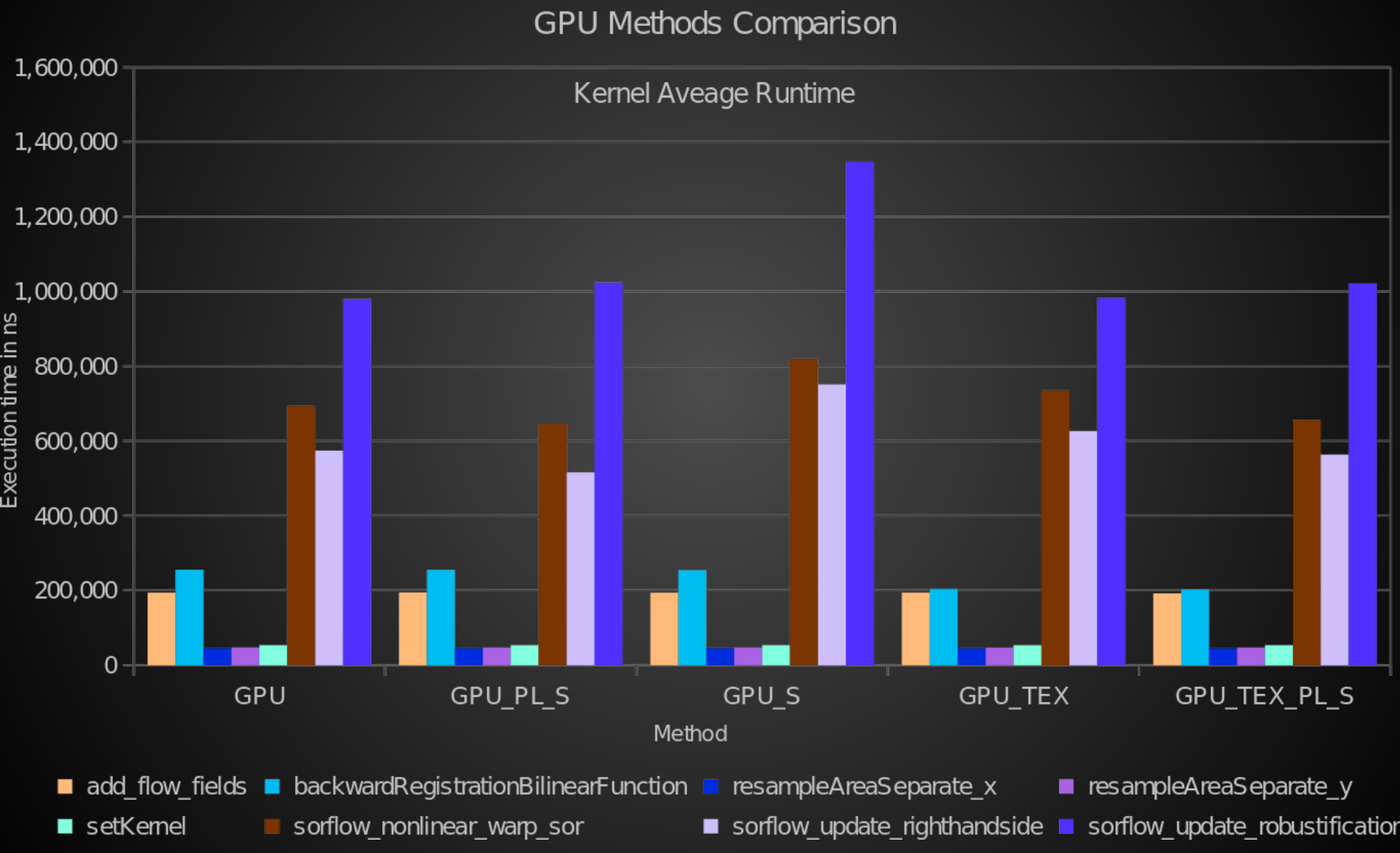
Added to Shared Memory

- u(x)
- du(x)

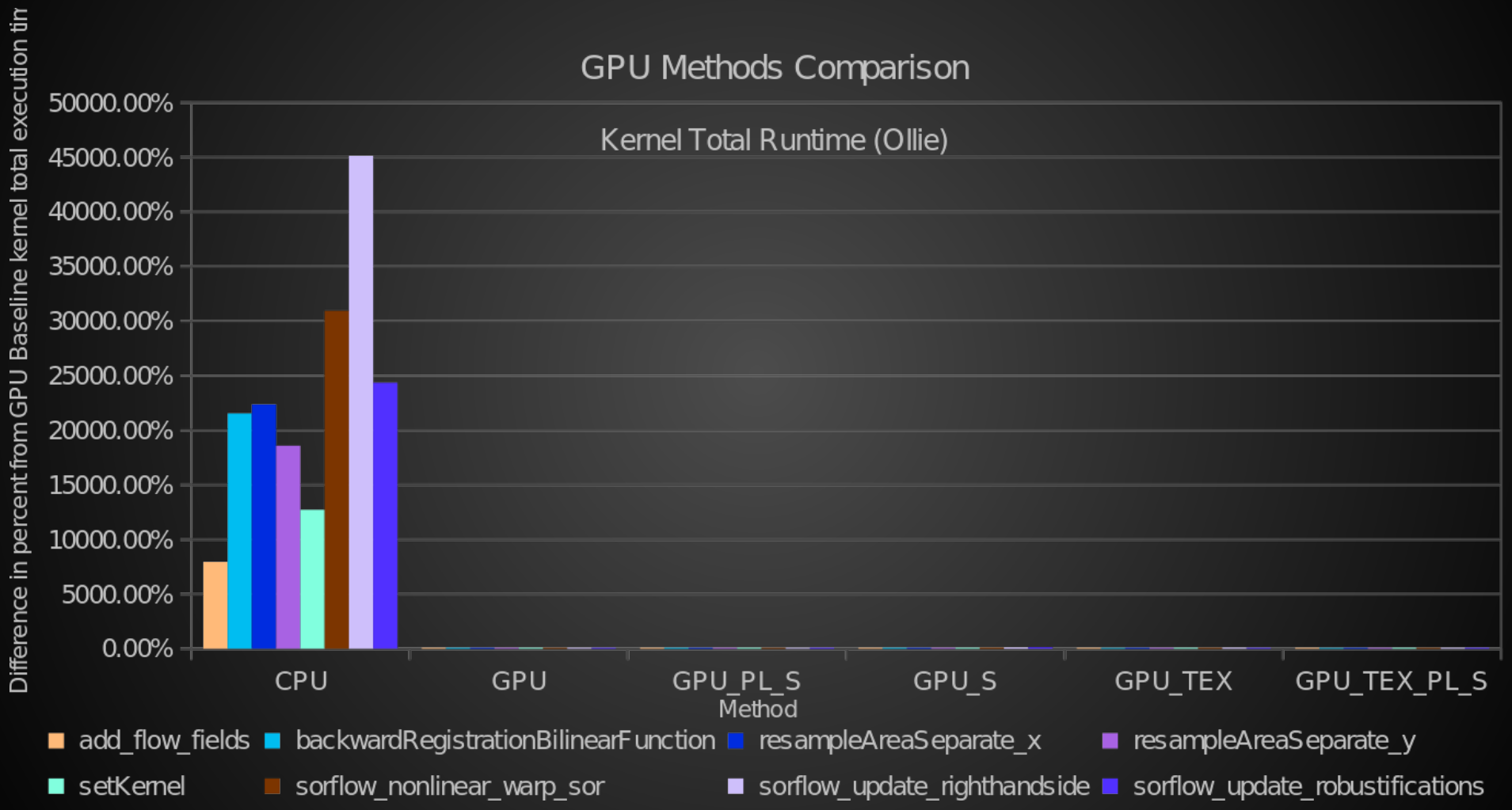
Optical Flow Results



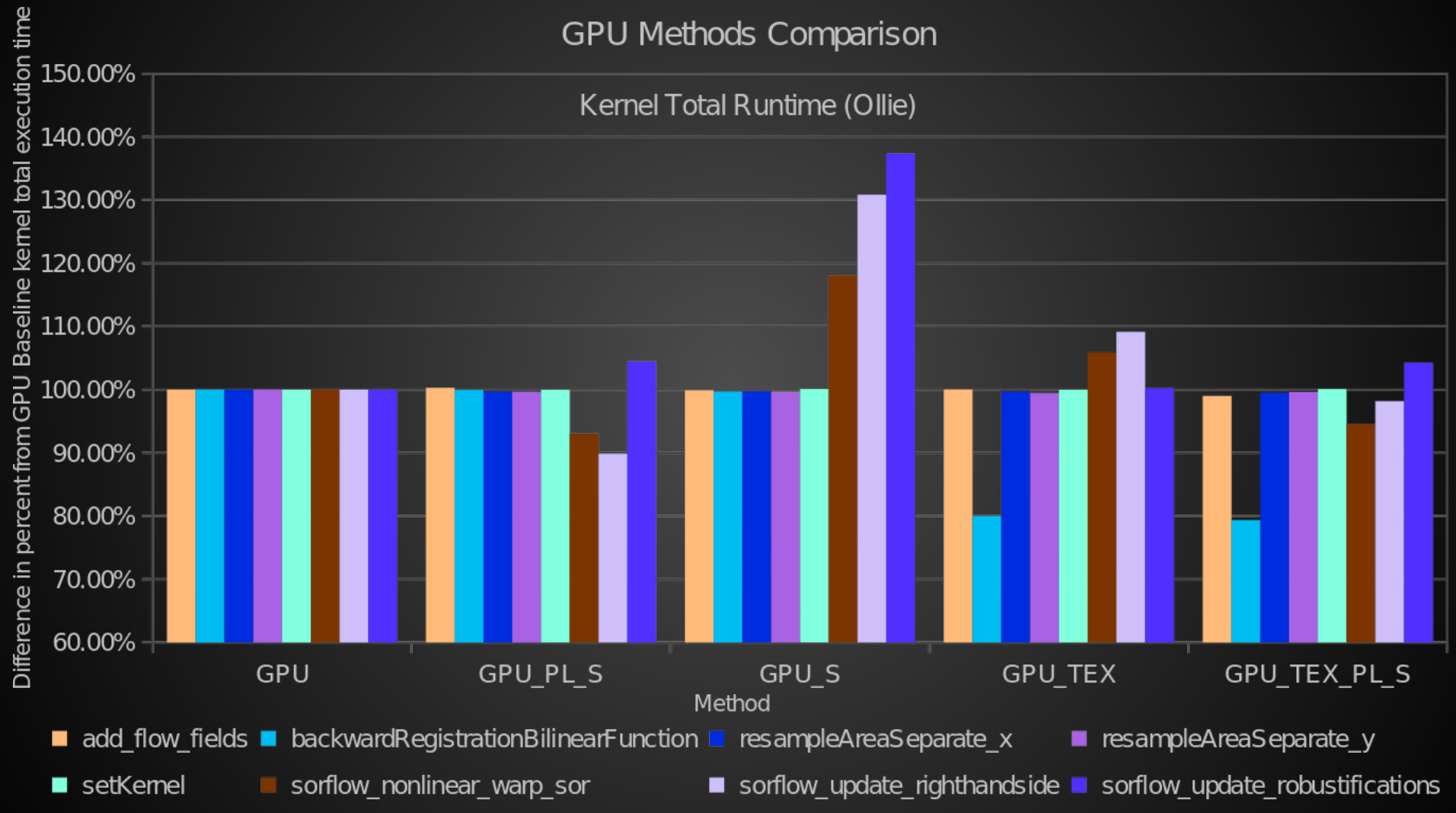
Optical Flow Results



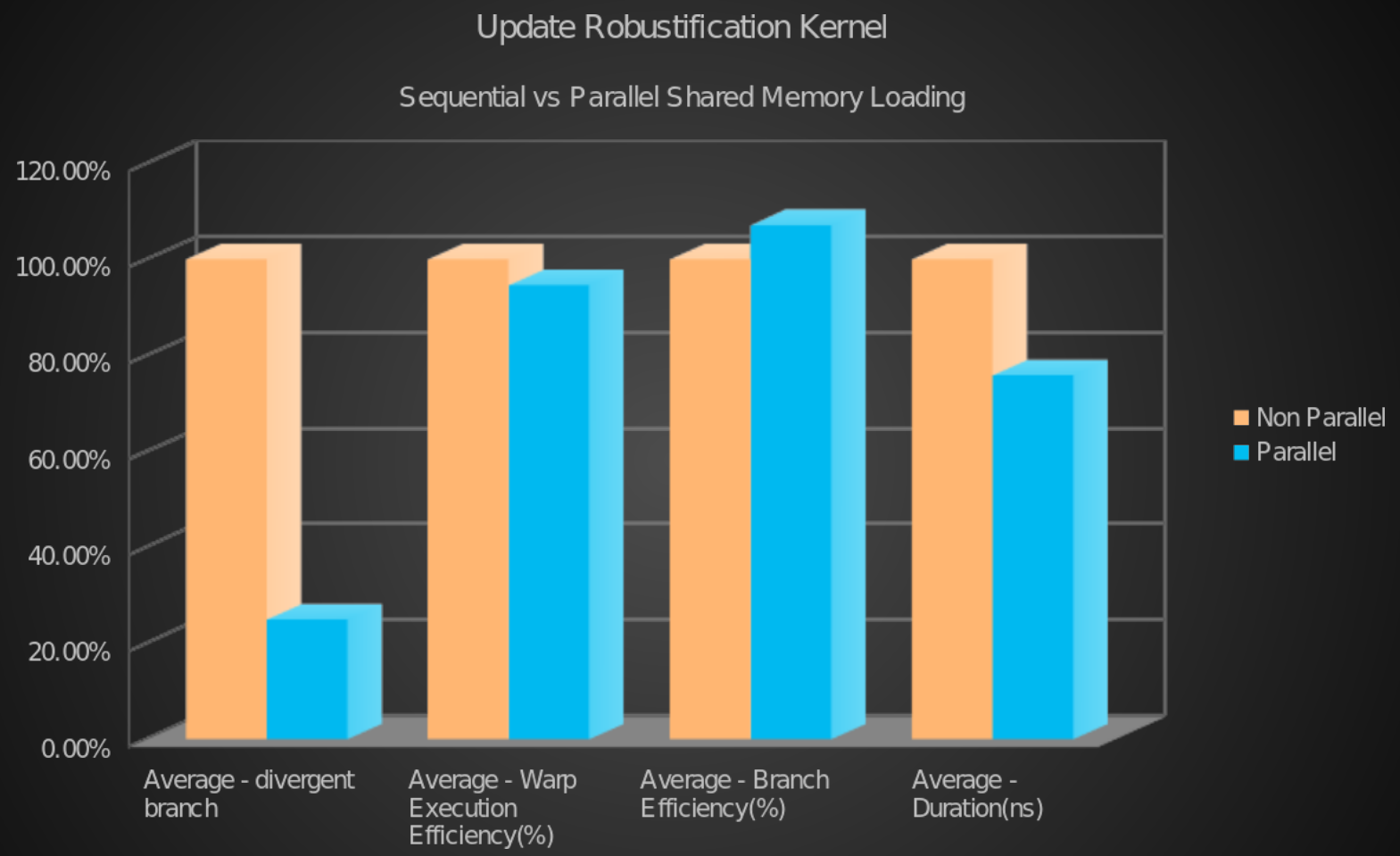
Optical Flow Results



Optical Flow Results

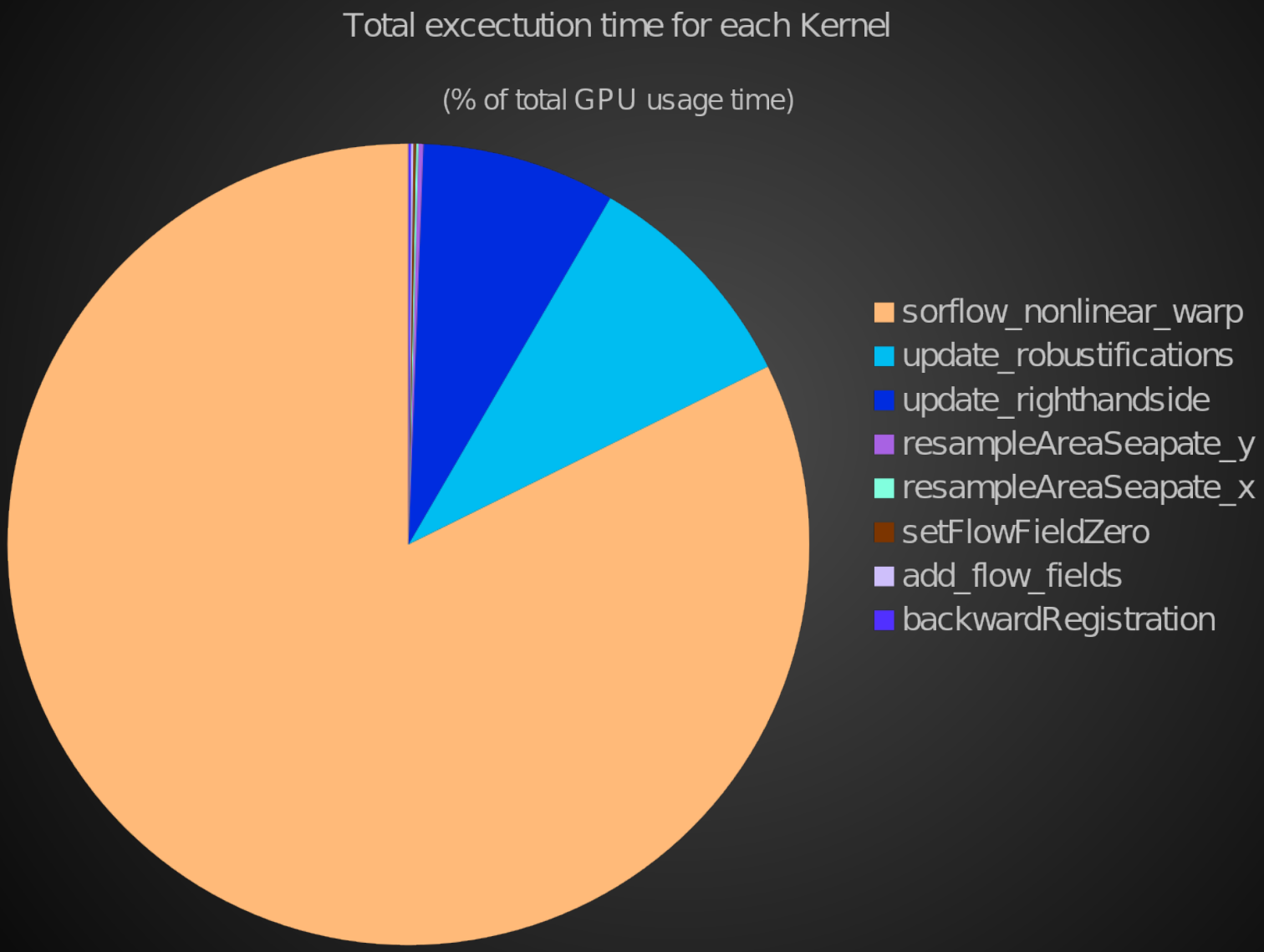


GPU Techniques - Small Hack 1

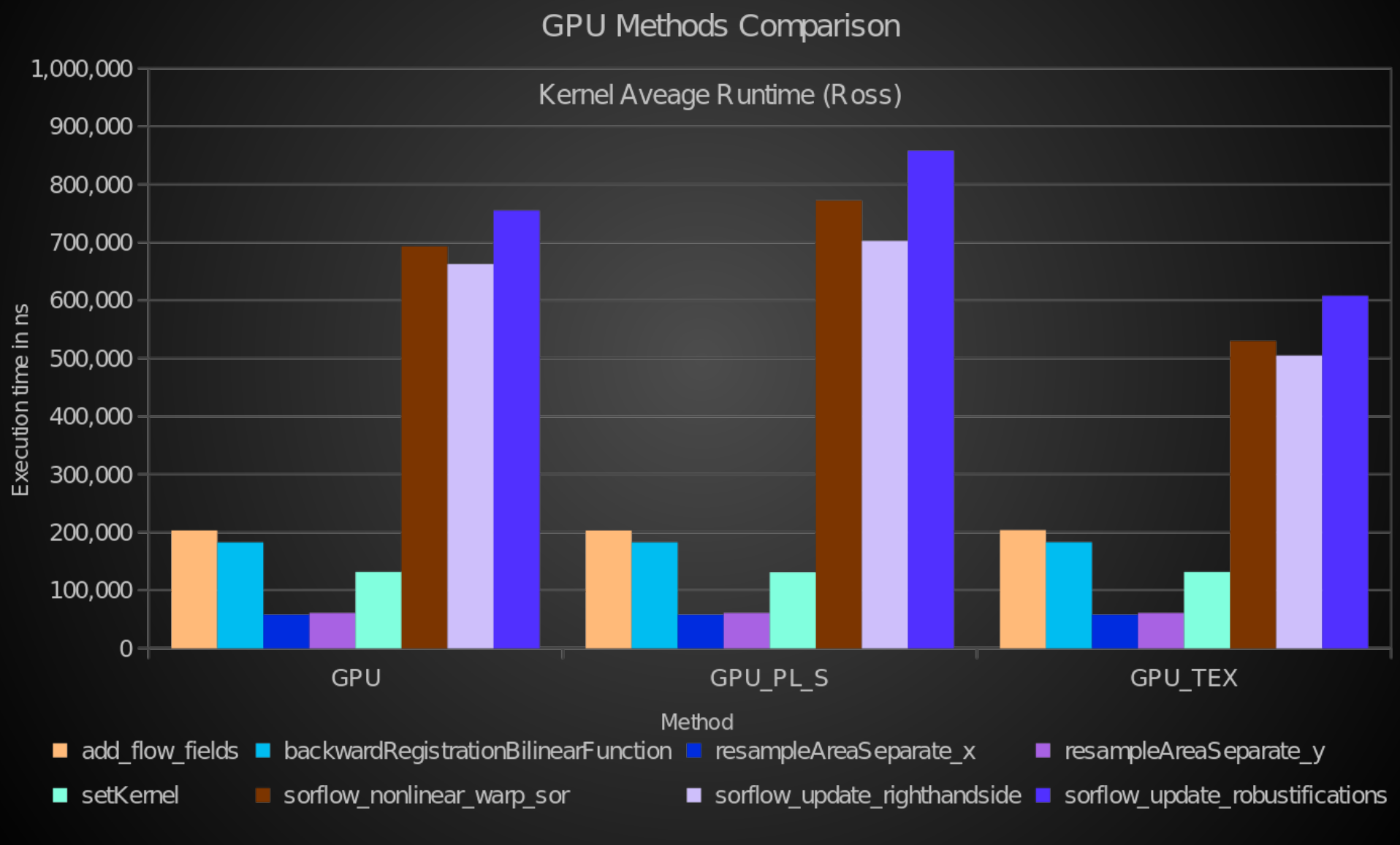



```
if (x == 0) shared_mem[0][ty] = shared_mem[tx][ty];          //one at a time
                                  vs
if (x == 0){   shared_mem1[0][ty] = shared_mem1[tx][ty];     //1 to N in parallel
               shared_memN[0][ty] = shared_mem2[tx][ty]; }
```

Optical Flow Results



Optical Flow Results



Super Resolution: Formulation

Given a number of degraded observations of image *I* that have been transformed by linear operations T^i a set of linear equations can be obtained:

$$I_F^1 = FT^1 I$$
$$I_F^2 = FT^2 I$$

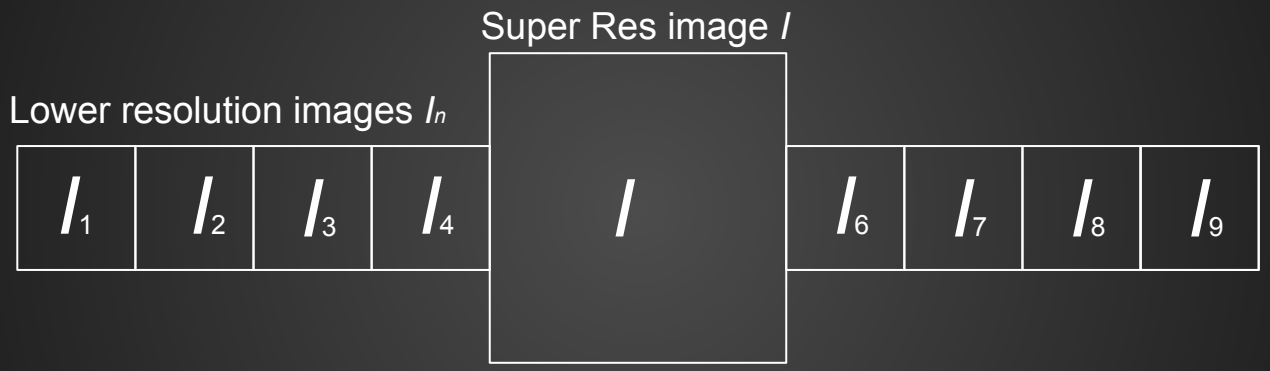
$$I_F^n = FT^n I$$

Super Resolution: Formulation

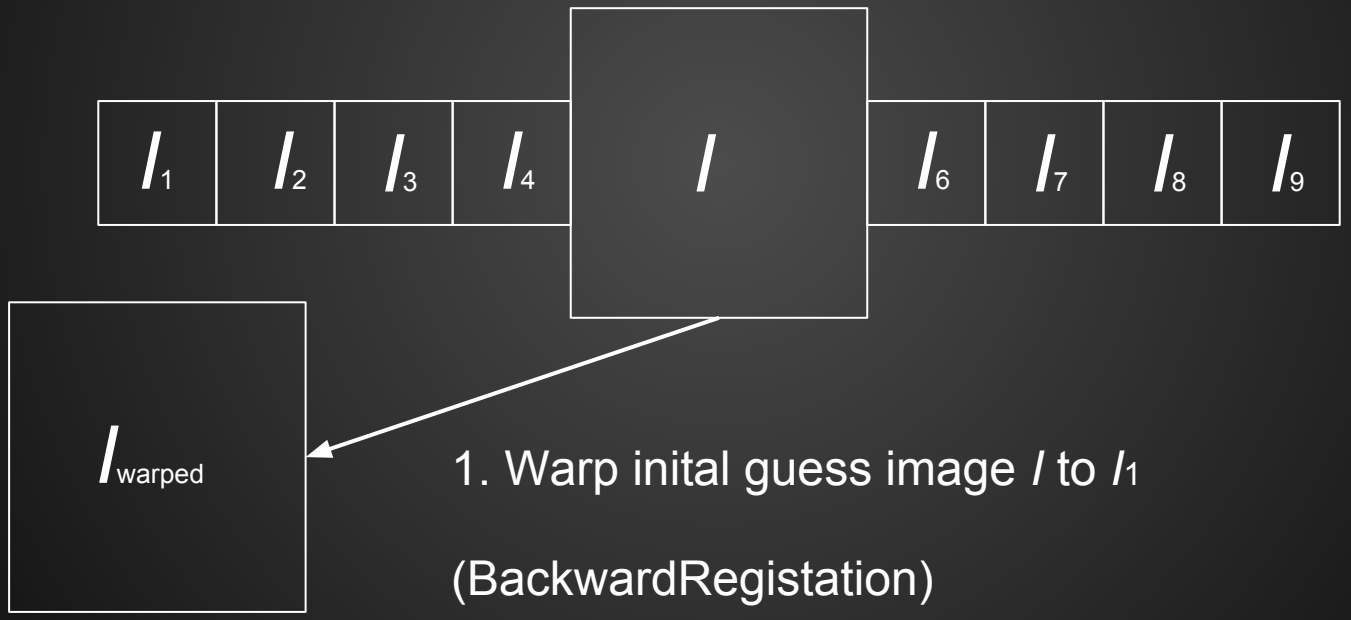
Energy function with regularity penalty term:

$$E(I) = \sum_{i=1}^{n} \mu ||FT^i I - I_F^i||_1 + \lambda ||\nabla I||_1$$

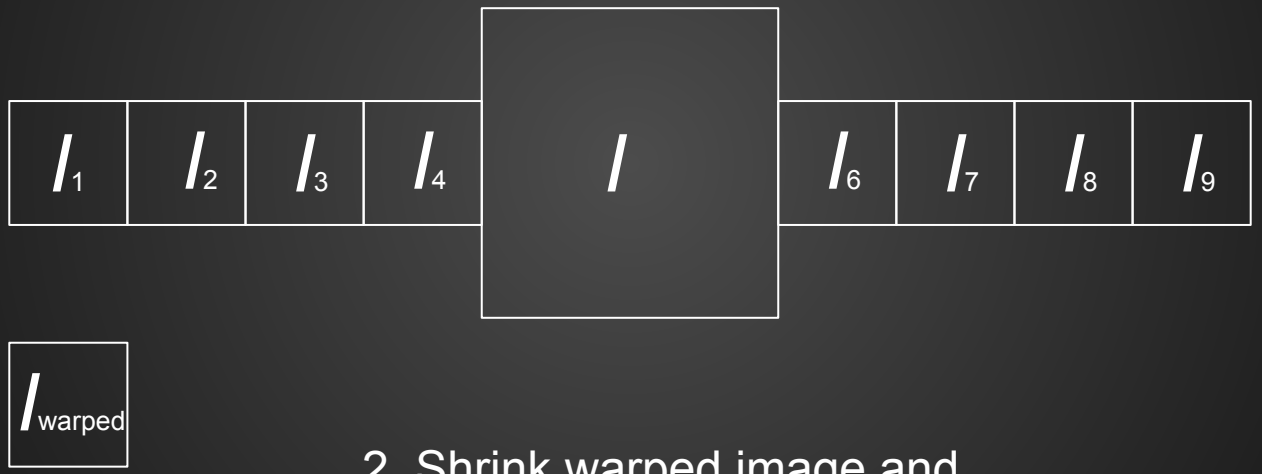
Super Resolution: General Method



Super Resolution: General Method

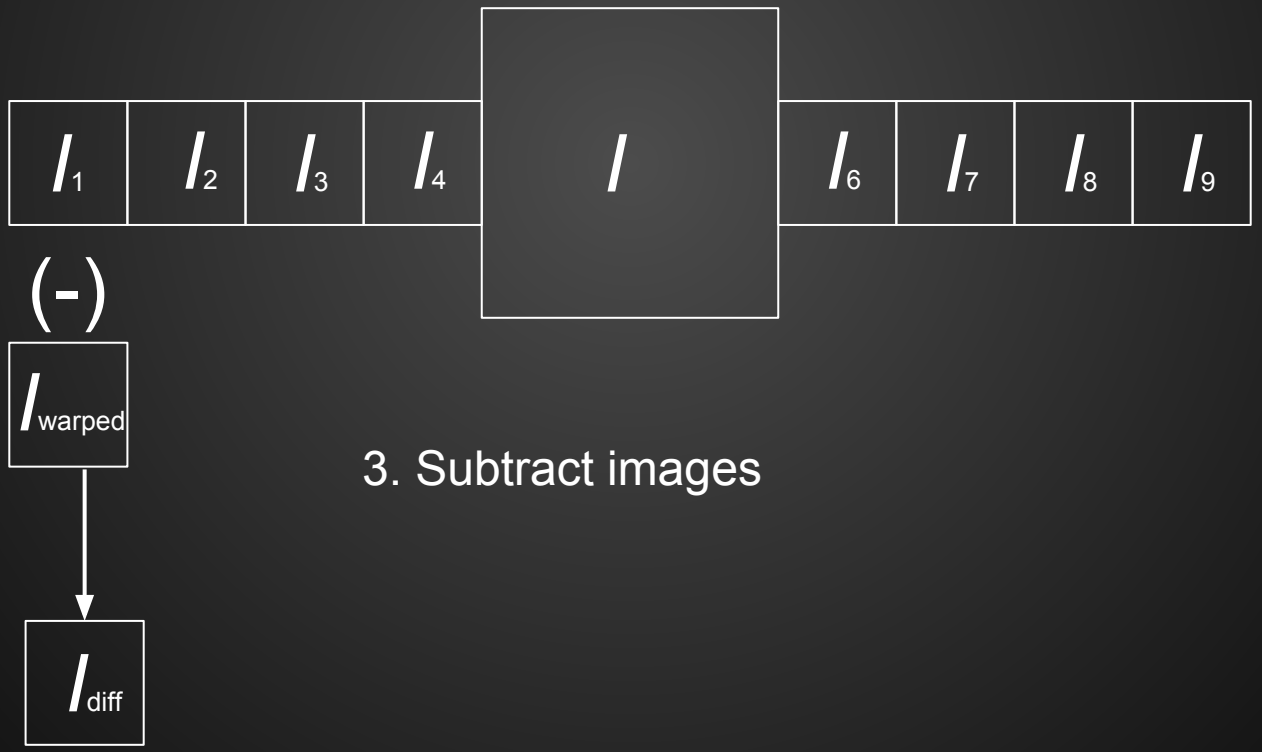


Super Resolution: General Method

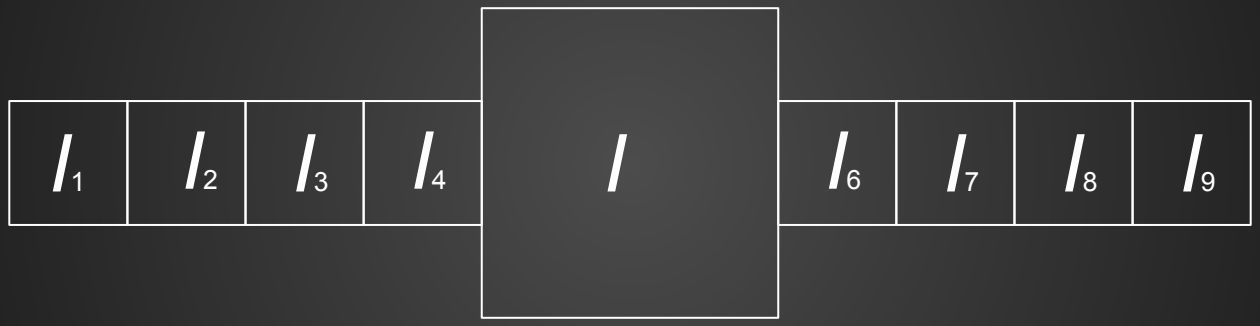


2. Shrink warped image and perform gaussian blur

Super Resolution: General Method

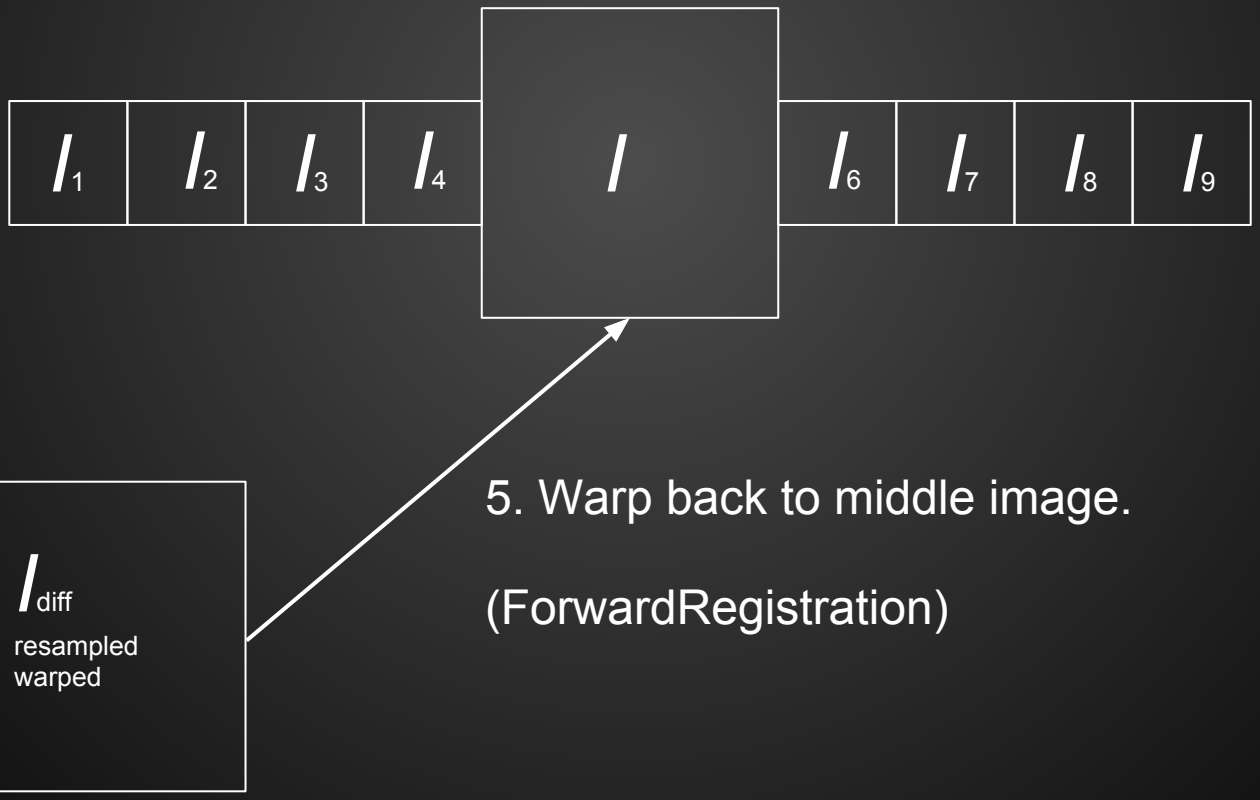


Super Resolution: General Method



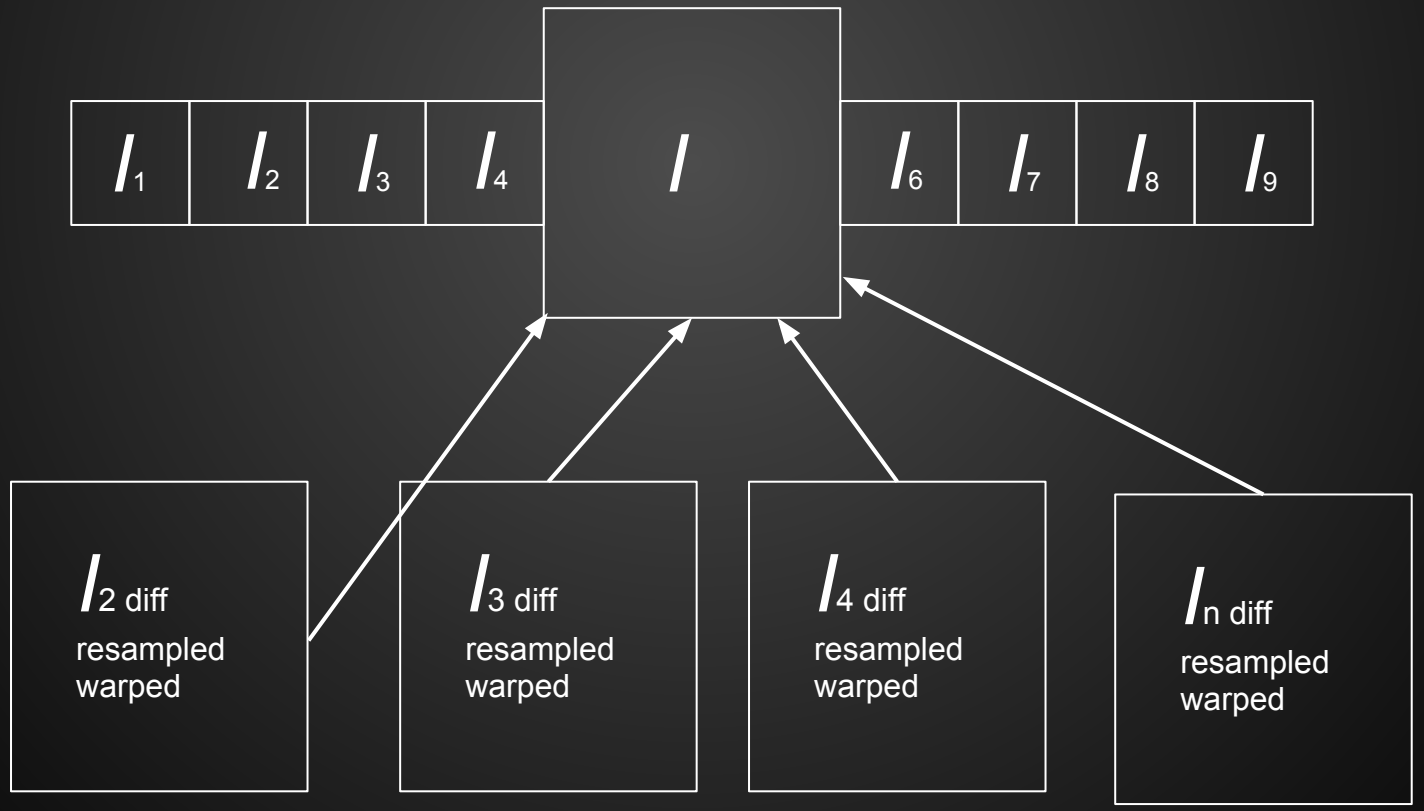
4. Resample difference image back to larger resolution

Super Resolution: General Method



Super Resolution: General Method

6. Repeat for all other images and add all difference images together



Super Resolution: General Method

7. Repeat the entire process until image converges

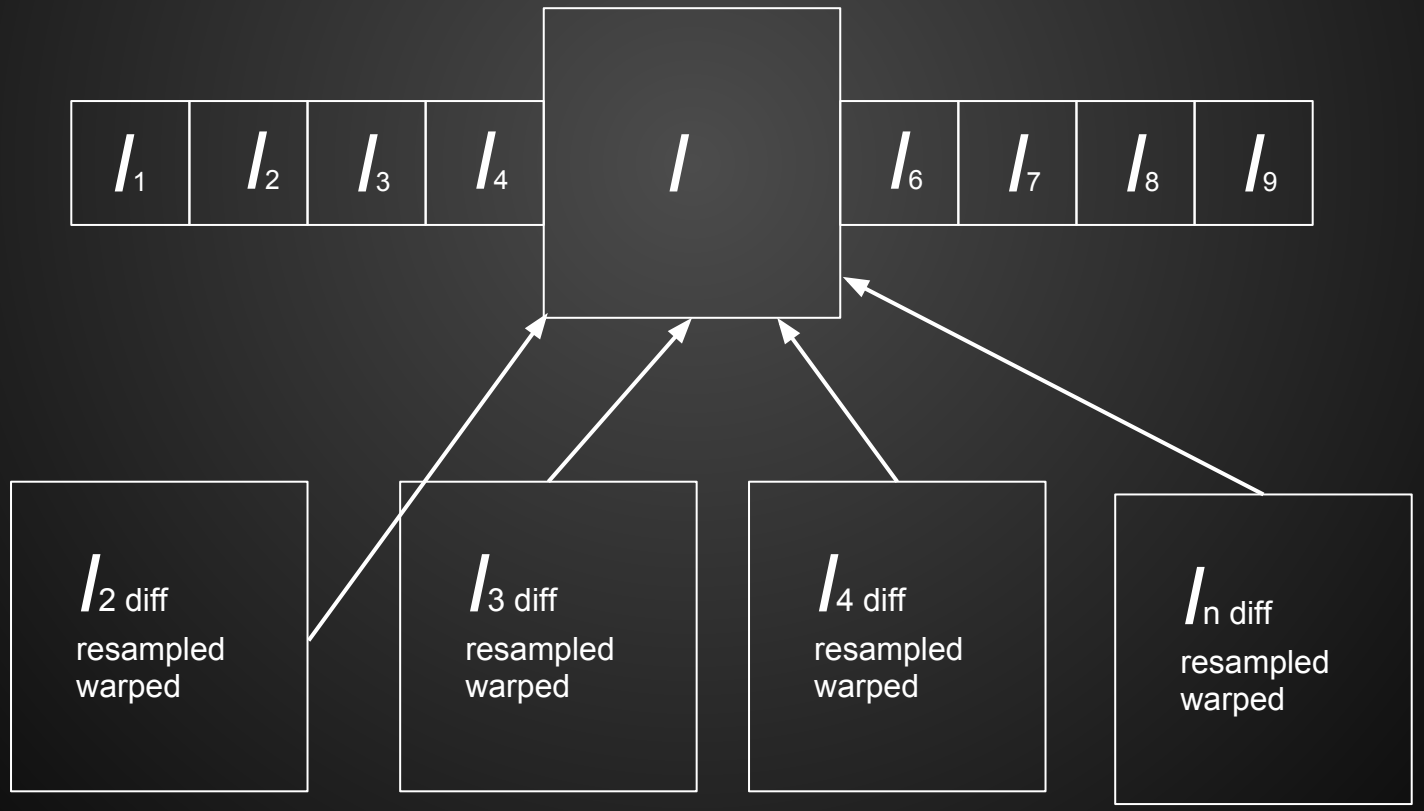


Image loop 1:Calculate diff images

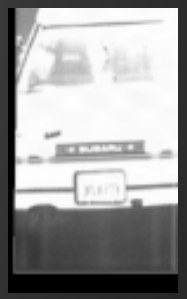
For each image I_n:

backwardRegistrationBilinearValueTex(uor(x), u(x))

gaussBlurSeparateMirrorGpu() ***[const memory!]***

resampleAreaParallelSeparate()

shrink



dualL1Difference()

Results in a vector of original sized difference images

Image loop 2: Warp difference images forward

For each difference image

resampleAreaParallelSeparateAdjoined()

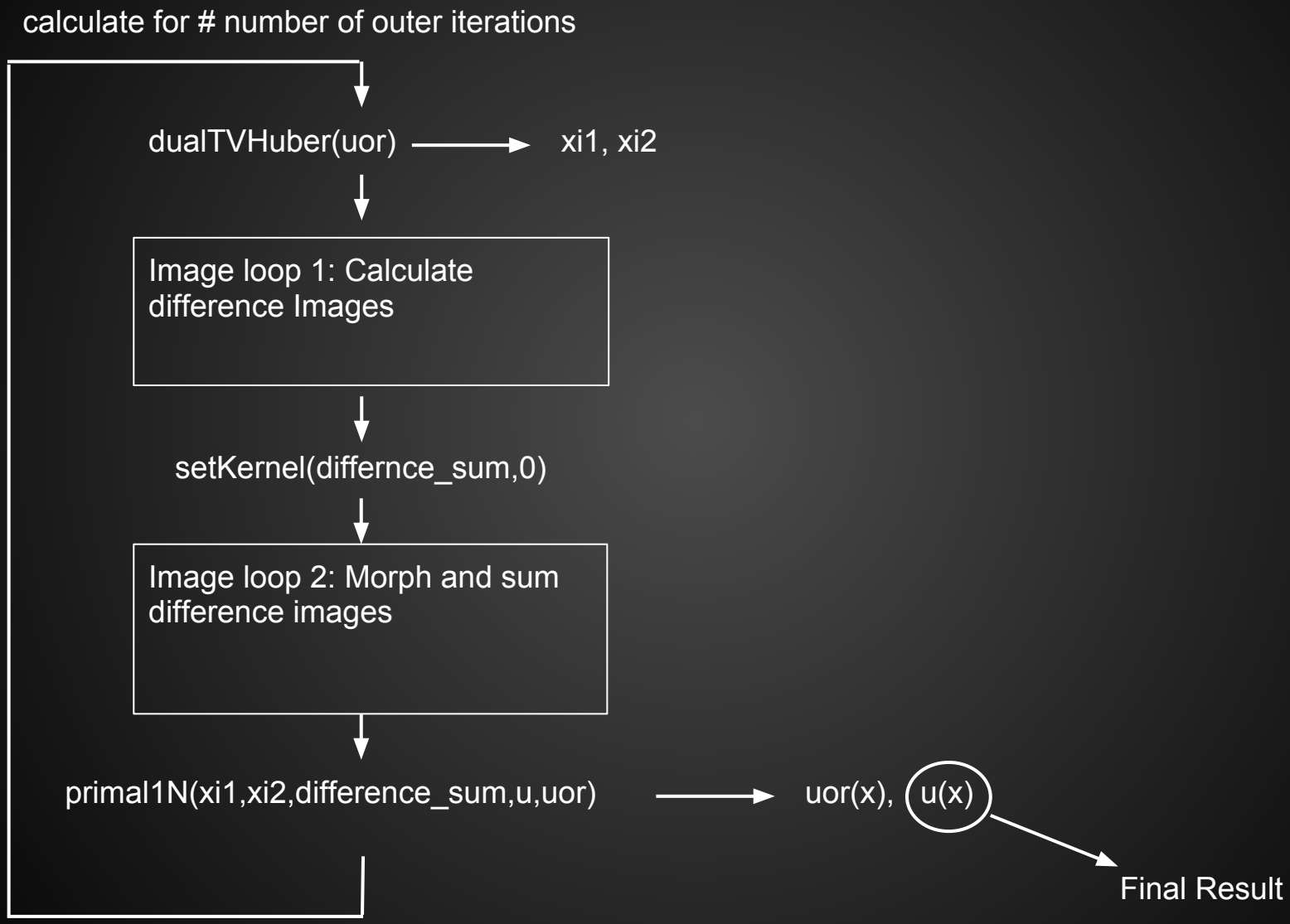
gaussBlurSeparateMirrorGpu()

forewardRegistrationBilinearAtomic()

addKernel(diff_image, temp_accumulator)

Results in an image of all summed differences together

Putting it together (for one image)

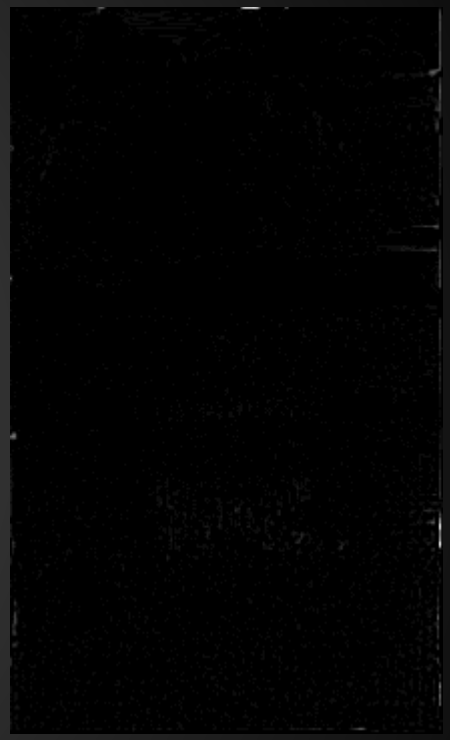


Super Resolution Output & Cuda Tex2d(...) differences

original

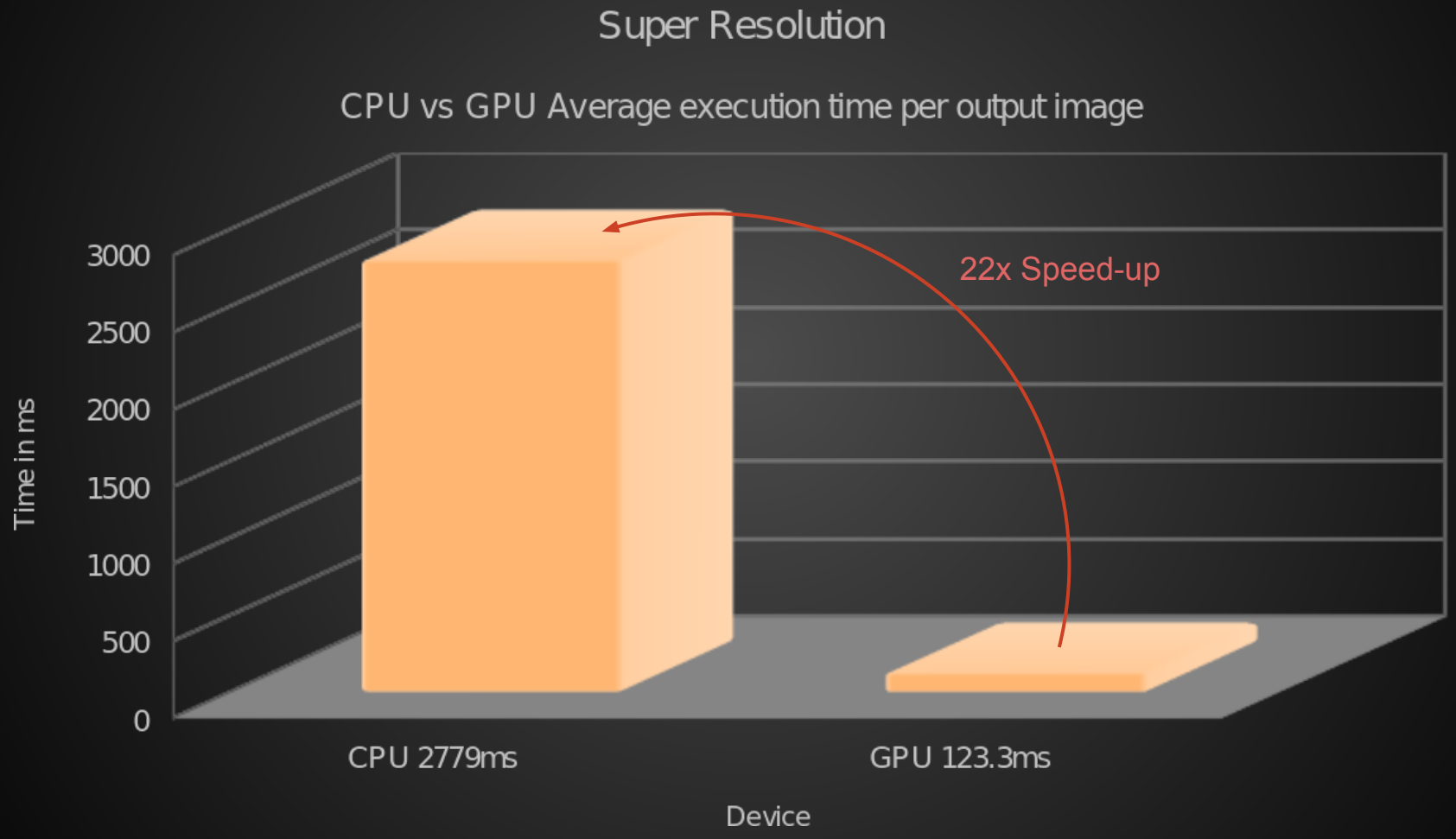
manual

cuda tex2d(...)

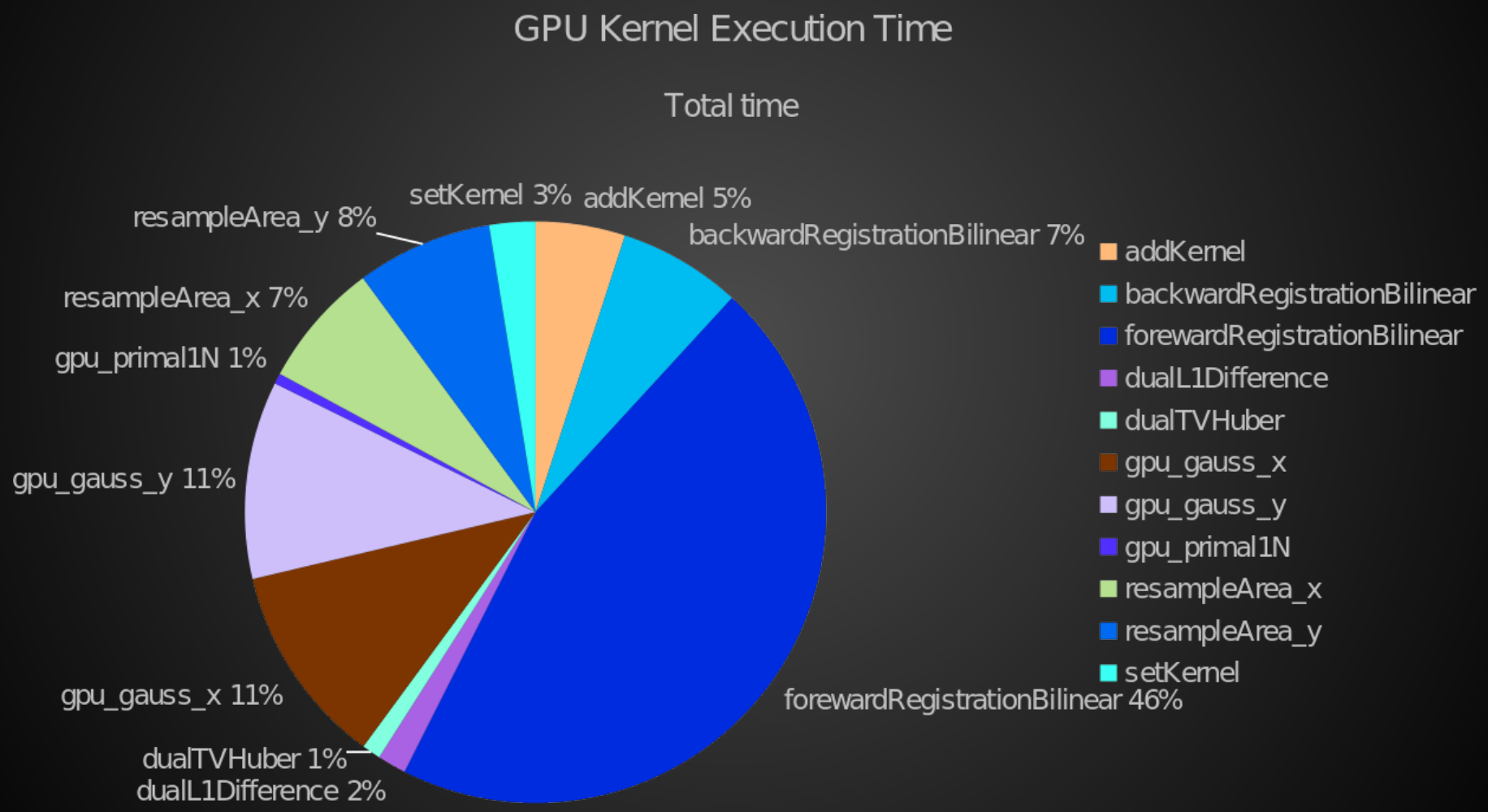


difference

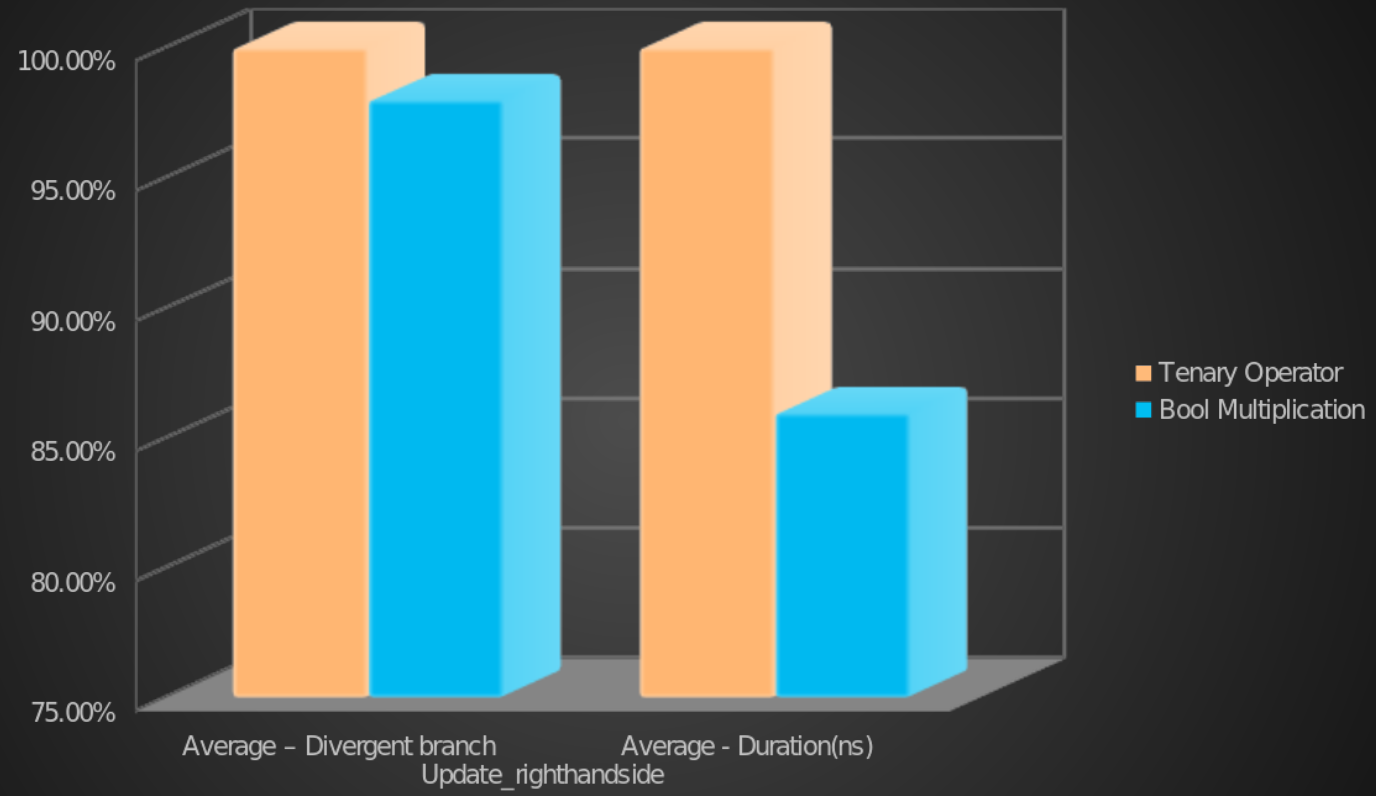
Super Resolution GPU Performance



Super Resolution Performance



GPU Techniques - Small Hacks 2




```
const int tx_1 = tx == 0 ? tx : tx - 1;
```

vs

```
const int tx_1 = tx - 1 * (x > 0);
```

Debugging techniques: Compare Image class

- Automatically compare cpu to gpu images
  - Compares pixel values
  - Errors not always visible
- Outputs which failed, stats, display image differences
- Facilitates code testing after optimizations/hacks
- Problem with texture memory (low similarity thresholds)

```
ImageComparison* ic = ImageComparison::instance();

CPU:
ic->addImage(_I1pyramid->level[rec_depth], "CI1",rec_depth, nx_fine, ny_fine,1, "CPU");
ic->addImage(_I2pyramid->level[rec_depth], "CI2",rec_depth, nx_fine, ny_fine,1, "CPU");
ic->dumpData("cpu_images"); //save data to disk
....
GPU:
ic->addImage(_I1pyramid->level[rec_depth], "CI1", rec_depth,nx_fine, ny_fine, pitch,"GPU");
ic->addImage(_I2pyramid->level[rec_depth], "CI2", rec_depth,nx_fine, ny_fine, pitch,"GPU");
....
ImageComparison::instance()->compareImages("cpu_images");
```

Difficulties

- Diving into the code
- Compiling on a local system
- Debugging segaults (printf, __LINE__, __FILE__ helped...)
- Texture offsets
- Incorrect in/out pitches
- Forgetting to *catchkernel* (oops)
- Evaluating massive amounts of data
- Maintaining two code bases
- Combining benchmark data

References:

Nils Papenberg, Andres Bruhn, Thomas Brox, Stephan Didas, and Joachim Weickert. 2006. **Highly Accurate Optic Flow Computation with Theoretically Justified Warping**. *Int. J. Comput. Vision 67, 2 (April 2006), 141-158.*

Markus Unger, Thomas Pock, Manuel Werlberger, and Horst Bischof. 2010. **A convex approach for variational super-resolution**. *In Proceedings of the 32nd DAGM conference on Pattern recognition*, Michael Goesele, Stefan Roth, Arjan Kuijper, Bernt Schiele, and Konrad Schindler (Eds.). Springer-Verlag, Berlin, Heidelberg, 313-322.

NVIDIA CUDA C Best Practices Guide DG-05603-001_v4.1 | January 2012