OpenCL Imaging on The GPU: Optical Flow

James Fung
Developer Technology, NVIDIA
Pyramidal Lucas Kanade Optical Flow in OpenCL

Algorithm by Jean-Yves Bouguet (Intel)
Implementation by James Fung (NVIDIA)
Optical Flow

• Calculate movement of selected points in pairs of images

• Applications:
  - Image stabilization
  - Feature tracking
  - Video encoding

• May be used to track
  - a few select points: sparse optical flow
  - All image points: dense optical flow
    - Computationally intensive!
Algorithm

- For each point \( u \) on image \( I \), find the corresponding point \( v \) on image \( J \)
- Calculate spatial gradient \( G \) in vicinity of each point \( u \)
- Iteratively solve for flow:

\[
\text{Calculate image mismatch vector } b \\
\text{Estimate motion } \quad v = G^{-1} b(x',y') \\
\text{Update position } \quad (x',y') = (x' + v_x, y' + v_y) \\
\text{Repeat for } N \text{ iterations or until convergence}
\]
Algorithm

Source image pair

Image I ➔ Image J

Precompute $G$ from $I$

$G_{2x2}$

Flow $(v)$

Downsample

$S$-charr Edge filter

Solve for flow

$v = G^{-1}b$

Iterate $v = G^{-1}b$

Flow Result

Iterate $v = G^{-1}b$

Iterate $v = G^{-1}b$
Optical Flow in OpenCL

• Embarrassingly parallel
• Computationally intensive

• Can take advantage of special-purpose GPU hardware:
  - Texture cache for 2D spatial locality
  - Hardware bilinear interpolation for sub-pixel lookups
  - Local Memory

• Can use OpenCL-OpenGL interoperability for visualization
Texture Cache

- Different areas of image have different amounts of motion, but motion is spatially coherent
  - Lookup window offset varies inside the image
  - Texture cache captures 2D locality

Large Motion (3-7 pixels)

Small motion (< 3 pixels)
Hardware Interpolation

• Sub-pixel accuracy & sampling is crucial
• Between iterations, $x,y$ is non-integer

$$
\partial I_k (x, y) = I^L (x, y) - J^L (x + g_x^L + v_x^L, y + g_y^L + v_y^L)
$$

$$
b_k = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \left[ \frac{\partial I_k (x, y) I_x (x, y)}{\partial I_k (x, y) I_y (x, y)} \right]
$$

• Use texture hardware linear interpolation

```c
sampler_t bilinSample = ...
    | CLK_FILTER_LINEAR;
...
float Jsample = read_imagef( J_float,
bilanSampler, Jidx+(float2)(i,j) ).x;
...```
Using OpenCL Local Memory

- Lookup locations for I, Ix and Iy are static per iteration
- Can be explicitly cached in Local Memory
  - Local memory is on-chip
  - Local memory is faster to access, useful for repeated access
Effect of moving from Texture to Local Memory Usage

Middlebury “Minicooper” data set
Frames: 10-11 Resolution: 640 x 480 Greyscale
Convergence: 0.0004px
Flow compute time for 3 pyramid levels shown
Optical Flow Demo

- Pyramidal Lucas Kanade Optical Flow
- Visualization done on GPU by sharing data between OpenGL & OpenCL
Optical Flow Performance

LK Pyramidal Optical Flow
GTX 460 & GTX 580

Middlebury “Minicooper” data set
Frames: 10-11 Resolution: 640x480 Greyscale
Convergence: 0.0004px
Pyramid 3-level flow compute time shown
OpenCL Performance Analysis with the CUDA Visual Profiler 4.0

- Local memory replay(%): 0.00
- Shared bank conflict replay(%): 11.33
- Shared memory bank conflict per shared memory instruction(%): 72.21

- Shared memory bank conflicts are high which causes serialization of threads. Using appropriate padding for data stored in shared memory so that the data is padded to a multiple of shared memory bank size helps on average across methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>L1 Cache Miss Rate</th>
<th>L2 Cache Miss Rate</th>
<th>Branch Conflicts</th>
<th>Texture Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86.1790</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97.5473</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96.0951</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82.5859</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93.3682</td>
</tr>
</tbody>
</table>
Thank You!

Contact:

James Fung
jfung@nvidia.com