Tracking Objects Better, Faster, Longer

Assoc. Prof. Dr. Alptekin Temizel
atemizel@metu.edu.tr
Graduate School of Informatics, METU

18 March 2015
GPU Technology Conference
Real-time tracking of objects in video is an important problem in various domains such as:
- Robotics
- Defense
- Security
- Immersive applications

Many studies in the literature are based on short term tracking which often fails if the object is:
- Occluded
- Disappears from the field of view
- Changes its appearance rapidly
- Goes through a large displacement between consecutive frames.
Track the object in real-time
- The object location is expected to be provided by the tracker in most cases.

Learn its appearance
- The predicted location of the object is used by P-N experts in the learning component.

Detect when it reappears after an occlusion or disappearance
- when the detector has higher confidence than the tracker, the object is assumed to be at the location estimated by the detector and the tracker is reinitialized with this result.
Motivations for Optimization

- Increase the resolutions for which the algorithm can run in real-time,
- Allow running multiple instances of the algorithm to support multiple object tracking,
- Allow running the algorithm at higher accuracy.
  - Tuning the algorithm parameters for higher tracking accuracy requires higher computation power,
Detector needs to check 30,000 Bounding Boxes even in a 320x240 frame!
## Test Platform

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Windows 7 x64</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel i7 4770K 3.5 GHz, 4 Physical Cores, Hyper Threading Factor is 2</td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla K40c, Compute Capability 3.5</td>
</tr>
<tr>
<td></td>
<td>15 Streaming Multiprocessors (SM)</td>
</tr>
<tr>
<td></td>
<td>192 Cores per SM (total of 2880 cores)</td>
</tr>
<tr>
<td></td>
<td>2 Async. Copy Engine, Hyper-Q Enabled</td>
</tr>
<tr>
<td>RAM</td>
<td>32 GB DDR3</td>
</tr>
<tr>
<td>Serial Computer Expansion Bus</td>
<td>PCIe 2.1</td>
</tr>
<tr>
<td>CUDA Toolkit</td>
<td>6.0</td>
</tr>
<tr>
<td>CUDA Driver Version</td>
<td>6.0</td>
</tr>
<tr>
<td>CUDA Run time Version</td>
<td>6.0</td>
</tr>
<tr>
<td>OpenCV Version</td>
<td>2.4.9</td>
</tr>
<tr>
<td>OpenMP Version</td>
<td>2.0</td>
</tr>
</tbody>
</table>
### Analysis for various video resolutions

<table>
<thead>
<tr>
<th>Component</th>
<th>Time per call (ms)</th>
<th>Time for whole sequence (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>480x270</td>
<td>960x540</td>
</tr>
<tr>
<td><strong>Tracking</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LK Optical Flow</td>
<td>1.100</td>
<td>4.280</td>
</tr>
<tr>
<td>Normalized Cross Corr.</td>
<td>0.620</td>
<td>0.630</td>
</tr>
<tr>
<td><strong>Learning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Generation</td>
<td>0.010</td>
<td>0.020</td>
</tr>
<tr>
<td>Random Forest Update</td>
<td>0.440</td>
<td>1.200</td>
</tr>
<tr>
<td>Patch Warping</td>
<td>0.080</td>
<td>0.230</td>
</tr>
<tr>
<td>BB Overlap</td>
<td>0.020</td>
<td>0.060</td>
</tr>
<tr>
<td><strong>Detection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Recall</td>
<td>5.930</td>
<td>20.400</td>
</tr>
<tr>
<td>Integral Image</td>
<td>0.271</td>
<td>1.100</td>
</tr>
<tr>
<td>Image Blurring</td>
<td>1.685</td>
<td>6.509</td>
</tr>
</tbody>
</table>
Analysis for 1920x1080 video
Optimization Strategy

- Heterogeneous implementation
  - Serial parts are run asynchronously on the CPU
  - The most computationally costly parts are parallelized on the GPU

- Apply stream compaction

- Design the data structures to allow coalesced access

- Use shared memory whenever suitable.

- Load balancing - this is achieved by the proposed grouping of the data.
Lucas-Kanade Optical Flow

Pyramidal Lucas-Kanade is used to handle large motion

Open-CV’s GPU Module which has a large community support has been adopted
Patch Warping is the most computationally expensive part.

The other parts do not take significant processing time as they involve calculation for a limited number of BBs and learning is invoked intermittently. As such, implementation of these parts on GPU were considered infeasible.

Processing these parts on the CPU while processing patch warping on the GPU necessitates moving large amounts of data (i.e. warped patches) between CPU and GPU.

As a result, we have decided to keep the learning component purely on CPU.
Implementation: Detection

Start

GPU
Calculate Integral Images

Next Cluster
i = i+1

GPU
Run PV-Computation on Cluster_i

BB_PV > MIN_VAR

NO
Discard It

YES
Implementation: Detection

1. **GPU**
   - Run Stream-Compaction on Remaining BBs for Cluster_i

2. **GPU**
   - Run RFI Calculation on Cluster_i

3. **Compute Confidence Values for Cluster_i**

4. **CPU**
   - More Clusters?

5. **End**
   - NO
- Ensure chunks to have similar number of BBs to be processed.

- Exploitation of spatial locality of BBs is also important.
Stream Compaction

- Patches having low variance (marked with -1) need not to be transferred to the CPU.

- Stream compaction is performed by calculating the shift amounts by prefix-sum.

<table>
<thead>
<tr>
<th>BB₀</th>
<th>BB₁</th>
<th>BB₂</th>
<th>BB₃</th>
<th>BB₄</th>
<th>BB₅</th>
<th>BB₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BB₀</th>
<th>BB₁</th>
<th>BB₂</th>
<th>BB₃</th>
<th>BB₄</th>
<th>BB₅</th>
<th>BB₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
</tr>
</tbody>
</table>
Results

#40, Posterior 0.82; fps: 84.16, #numwindows:66875, Learning
Experimental Results

- **H-TLD** vs **TLD**
  - For 480x270 resolution: H-TLD takes 3.711 milliseconds, TLD takes 10.156 milliseconds.
  - For 960x540 resolution: H-TLD takes 6.193 milliseconds, TLD takes 37.520 milliseconds.
  - For 1920x1080 resolution: H-TLD takes 11.950 milliseconds, TLD takes 122.460 milliseconds.

Elapsed Time in milliseconds
The main bottleneck is the data transfers between the CPU and GPU memory spaces.

A further analysis of the framework reveals that approximately 45% of total recall calculation time is spent on RFI part; and approximately 78% of the RFI Calculation’s time is spent in moving the calculated RFIs to the host side.

If this data transfer could have been eliminated, a theoretical speed-up bound of 13.13x at 1920x1080 resolution would be obtained.

This theoretical analysis shows the potential impact of expected memory bandwidth enhancements and speed-up of data transfers between CPU and GPUs in the next generation architectures.
Questions

H-TLD library code repository
https://github.com/iliTheFallen/htld

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!

For further enquiries:
Dr. Alptekin Temizel
http://www.metu.edu.tr/~atemizel/
atemizel@metu.edu.tr