Parallel Video Object Tracker
Project Milestone Report

Eric Li
Carnegie Mellon University
Pittsburgh, PA 15213
`shizhuol@cs.cmu.edu`

Yanxin Jiang
Carnegie Mellon University
Pittsburgh, PA 15213
`yanxinj@andrew.cmu.edu`

1 URL

`https://github.com/askEric0/Parallel-Video-Object-Tracker.git`

2 SUMMARY

We are developing a high-performance visual object tracker implemented in `C++` and `CUDA`. This system aims to achieve high performance on video object tracking, both in speed of processing and GPU utilization. During the past one week and half, we have successfully built multiple baselines for the comparisons of our own tracker. These baselines includes:

1. Tracker from Opencv library (CPU)
2. Tracker from Opencv library (GPU)
3. Tracker of self-annotated CPU baseline with KCF Algorithm

We have also successfully implemented preliminary final tracker based on NCC, both CPU version and CUDA version.

2.1 Opencv CPU Tracker

This baseline uses OpenCV's built-in tracking API (e.g., CSRT/KCF). It provides a high-quality reference implementation but is treated as a black box. It runs entirely on the CPU, uses hand-engineered features + correlation filters and is highly robust to occlusion and deformation.

2.2 Opencv GPU Tracker Naive

OpenCV's GPU tracking support is limited to optical flow and does not include full correlation-filter trackers on GPU.

2.3 CPU + KCF + FFT

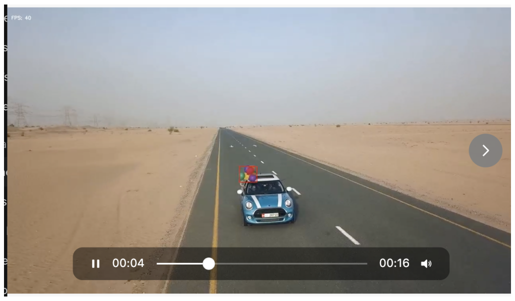
It is similar to opencv cpu version with KCF and FFT, whereas not robust as opencv's polished version.

2.4 Our NCC Tracker

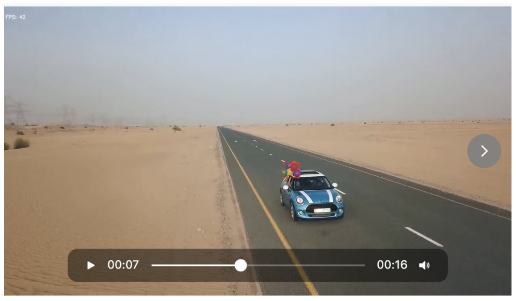
This is our final high-performance tracker, based on Normalized Cross-Correlation (NCC). Instead of FFT-based correlation filters, we directly compute normalized cross-correlation between: the current frame and a fixed (or slowly updated) template. Each output location is computed independently, making it ideal for massive GPU parallelism.

3 PROGRESS RESPECT TO PROPOSAL

We have finished implementation half of our final goal, that is finish the baseline and finishing the implementation parts of final tracker. The baselines performed well and we have also built our own baseline. In the final version implementation, our current implementation are tested and works well in video object tracking not only in easy scenes but also in some complex scenes (the object is shortly blocked behind or half blocked behind an obstacle).



(a) Objecting Tracking 1



(b) Objecting Tracking 2

Figure 1: Side-by-side comparison of runtime performance for CPU baselines vs. our GPU NCC implementation.

Do we still expect to meet our original deliverables? Our initial goal of achieving a 5× speedup over the CPU baseline is unlikely to be met as it would require much more complicated kernel-level optimization, which cannot be achieved only in one week. However, we expected to achieve a modest improvement of about 1.2–1.5× over our current CUDA implementation through batching and other small optimizations. We may also be able to achieve a stable real-time or near-real-time tracking on the short videos, which still aligned with the main goal of our proposal.

Updated Deliverables for Poster Session

- Functional CUDA-based NCC tracker with batching.
- Comparison against OpenCV CPU and OpenCV GPU (naive) baselines.
- Real-time object tracking in both CPU and GPU mode for a short video.

4 ISSUES AND QUESTION MARKS

Questions remain despite good progress.

1. The OpenCV CPU tracker is very strong and impressive in robustness and speed. It has been polished for many years by developers hence fully optimized, it is impossible for our customized version (even GPU version) to surpass this baseline. The Naive GPU version of OpenCV is easy to surpass though, as it only uses optical flows.
2. As in the proposal, we are adding batching to object tracking, which will significantly improve the speed of processing as it processes many frames at the same time. However, batching will significantly decrease the robustness of tracking as it batches the template updates as well.
3. We found improvement of GPU version compared to CPU version rather small, especially given the video is short, and tracking box (affect kernel size) is small. Launching CUDA kernels will even produce more overhead.

5 ADJUSTMENT OF GOAL

1. We plan to drop off the section of further optimization beyond batched frame processing.
2. We will compare mostly against OpenCV Naive GPU tracker, and seek to improve upon that. We are also further optimizing our final tracker with FFT features, if possible.

3. We will compute the frame process per second time to visualize comparison of speedup across different implementations.

6 PLAN OF PRESENTATION

We plan to show on the poster some frames of our video object tracking, and use laptop to do real-time video playing demo. Apart from that, we will provide graph of memory handling and structural design of our tracker.

7 DETAILED SCHEDULE

7.1 Dec 1 – Dec 4

- Implement batched frame processing to improve GPU utilization. (Eric & Yanxin)
- Try to apply lightweight kernel optimizations, such as basic shared-memory tiling, if it indeed helps optimization. (Eric & Yanxin)
- Try to achieve a modest speedup of 1.2–1.5$\times$ over the current CUDA version. (Eric & Yanxin)

7.2 Dec 5 – Dec 8

- Clean up and finalize the code for the final report. (Eric & Yanxin)
- Demonstrate stable real-time or near-real-time tracking on representative video examples. (Eric & Yanxin)