GPU Accelerated Multiple-object Tracking using Correlation Filters

Siddhesh Mehra
CyLab Biometrics Center
Carnegie Mellon University
Pittsburgh PA 15232
Email: siddheshmehra@cmu.edu

Nicholas A. Vandal
CyLab Biometrics Center
Carnegie Mellon University
Pittsburgh PA 15232
Email: nvandal@cmu.edu

***Abstract*—In this paper we introduce the Smoothed Average Phase Filter, a correlation filter built using multiple phase-only filters. An adaptive implementation of this filter achieves reliable tracking performance. In order to achieve peak tracking speeds while maintaining robust results, we present a system that exploits the 'embarrassingly parallel' nature of tracking independent objects by performing computations on Graphics Processing Units to perform real-time tracking of multiple objects using various families of correlation filters. We employ Nvidia's CUDA, a general purpose parallel computing architecture and software environment to achieve real-time performance in mega-pixel images. We propose to compare the results of matched, MACE, UMACE, ASEF and SAPF filters on the following tasks: (a) Pedestrian tracking, (b) Face tracking, (c) Vehicle tracking.**

***Index Terms*—Correlation Filters, CUDA, GPGPU, GPU, Tracking, Real Time, SAPF**

I. Introduction

Tracking in general is a complex and difficult task that can be defined in its simplest form as assigning consistent labels to an object as it moves around an image plane across successive frames in a video sequence. The task is made difficult by the presence of noise in the image, unknown scale due to projection from our 3D world onto the 2D image, variable scene illumination, complex motion of both the tracked object and the camera, full and partial occlusions and changes in appearance of the target[1].
While there are many classes of algorithms for tracking, we focus on a tracking algorithm class known as kernel tracking, which is performed by computing the motion of a primitive object region from on frame to the next[1]. We introduce the Smoothed Average Phase Filter, a correlation filter built using multiple phase-only filters. We use various classes of correlation filters to represent our tracked regions and explore the robustness and accuracy of tracking achievable with each class of correlation filter.
There are numerous obvious applications of tracking in the fields of security and surveillance, traffic monitoring, and vehicle navigation to name but a few, but a common constraint amongst most real systems is the requirement to perform tracking in real-time. This performance challenge is exacerbated by very real world tasks such as tracking many independent objects simultaneously and/or working with multi-megapixel video streams at high frame rates–it is in this problem domain that we have chosen to focus.
In order to achieve peak tracking speeds while maintaining robust results, we present a tracking system that exploits the 'embarrassingly parallel' nature of tracking independent objects as well as the data-parallel computations within an individual track by shifting most of the processing from the CPU to the Graphic Processor Unit (GPU). We employ Nvidia's CUDA, a general purpose parallel computing architecture and software environment, that greatly eases the burden of developing on GPU hardware that is designed for the compute-intensive, highly parallel task of graphics rendering[2].

II. Background and related work

A. Correlation filter theory

Correlation can be mathematically expressed as

$$c(t) = \int_{-\infty}^{\infty} f(x)g^{*}(x)dx \tag{1}$$

In terms of images, $f(x)$ may be considered as the test image, $g(x)$ as the 'template' image, and * denotes the complex conjugate. Our aim in correlation filter based tracking is to design a template which will maximize its correlation output of a true test image.
Correlation filters have been a popular method in the task of object detection since the introduction of the Synthetic Discriminant Functions[3]. The MACE[4] and UMACE[5] filters alleviate the problem of false correlation peaks by maximizing the correlation output at the origin while simultaneously minimizing the average correlation energy. Optimal Tradeoff filters [6] incorporate noise tolerance to correlation filters.
Henceforth in this paper, we refer to the 'template' as a 'filter' during the design process, and as the 'template' during the test process. These terms, in our case, are interchangeable. Also, general concepts of filter theory are referred to keeping in mind their application to images.
Let us also describe some of the terminology used in the next few sections. x_i refers to the i^{th} vectorized template image, X is a dxn matrix of n images stacked column-wise, h refers to the actual correlation filter.

1) Matched filters: In its simplest form, a matched filter is a replica of the object to be detected. They are optimal in case the test image has only been corrupted using Additive White Gaussian Noise. However, their performance degrades rapidly with slight distortion from the template. Average templates of multiple matched filters sometimes perform better, but also increase the false detection rate.

2) Synthetic Discriminant Filters: Another approach to build a filter from multiple template images of the object is to control the correlation output peak at the origin. Consider an nx1 vector c which defines the desired correlation output of each template image. The Equal Correlation Peak - Synthetic Discrimant Filter can be obtained from the equation

$$h = X(X^T X)^{-1} c \tag{2}$$

B. MACE and UMACE filters

The SDF filter tries to constrain only the output correlation peak at the origin. This sometimes results in false peaks besides the origin. MACE[4] and UMACE[5] filters try to solve this problem by minimizing the average correlation energy while constraining the output at the origin.
The MACE filter specifies the peak output at origin in a similar way to SDFs, while minimizing the average correlation energy. It can be obtained from the equation

$$h = D^{-1} X(X^{+} D^{-1} X)^{-1} c \tag{3}$$

The UMACE filter relaxes the constraints by not exactly specifying the value at the origin. This results in a simpler equation for the UMACE filter.

$$h = D^{-1} m \tag{4}$$

In the above equations, D is a dxd diagonal matrix with the average power spectrum of the template image along the diagonal, the $^{+}$ sign denotes the complex conjugate transpose, and m is the average template image.

MACE filters perform well, but are prone to overfitting and are computationally expensive due to the inverse calculation. UMACE filters solve these problems, but with lesser noise tolerance.

C. ASEF filters

ASEF filters[7] approach the filter training problem in a different manner. Each Synthetic Exact Filter (SEF) is the element-wise division of an image of a Gaussian centered at the target point with the complete image containing the template . An average of multiple SEFs results in the ASEF filter, which demonstrates good performance with a large number of training images.

$$H_{SEF} = \frac{G_i(u,v)}{F_i(u,v)} \tag{5}$$

Here, $G_i(u,v)$ is the Gaussian image, $F_i(u,v)$ is the training image and the division is element-wise. Notice the use of $F(u,v)$ instead of x implies that $F(u,v)$ is an entire image

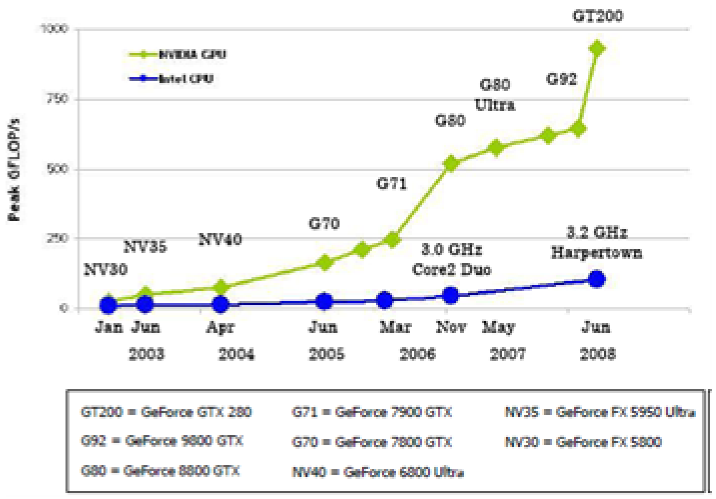

Fig. 1. Comparison of Floating-Point Operations per Second (FLOPS) for the GPU and CPU.[2]

which contains the template in the region where it overlaps with the Gaussian image.

$$h = \frac{1}{n} \sum_{i=1}^{n} (h_{SEFi}(u,v)) \tag{6}$$

D. General Purpose Computation on GPU and CUDA

Driven by the insatiable market demand for realistic 3D games, the GPU has evolved into a highly parallel, multi-threaded, many-core processor of tremendous power. In terms of peak FLOPS, modern commodity GPUs overshadow CPUs as shown in Fig 1. The growth rate of computational capabilities of GPUs have been growing at a average yearly rate of 1.7 (pixels/second) to 2.3(vertices/second), which is a significant margin over the average yearly rate of roughly 1.4 for CPU performance[9]. The reason for this discrepancy is that while improvements in semiconductor fabrication technology benefit both the CPU and GPU equitably, fundamental architectural differences favor the GPU in terms of peak computational throughput. The CPU dedicates a large portion of its die space to cache and flow control hardware such as branch predication and out-of-order execution–it is optimized for high performance in executing sequential code, whereas the GPU is optimized for executing highly data-parallel rendering code, and is able to devote more transistors directly for computation[9]. However, there is a definite trend towards a convergence of the GPU and CPU as CPUs add more and more cores, and GPUs have evolved from fixed rendering pipelines to today's fully programmable architectures.

Prior to the introduction of Nvidia's CUDA (Compute Unified Device Architecture) API and ATI's equivalent FireStream in 2006, performing general purpose computation on GPUs was challenging work, requiring programmers to express problems in terms of rendering textures on shader units and other graphics primitives. CUDA allows the programmer to develop in C or other high-level languages and dramatically lowers the barrier to developing dramatically accelerated applications.

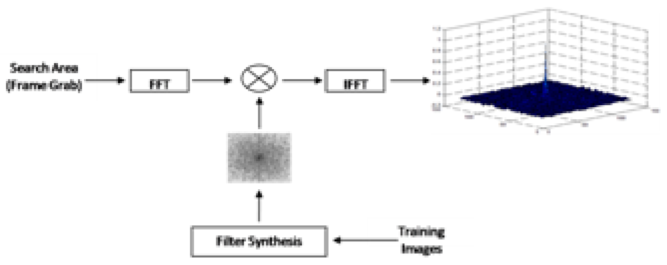

Fig. 2. High level block diagram of current filter tracking system demonstrates that normalized correlation of the filter with the search area is performed in the frequency domain for maximal efficiency.

III. Smoothed Average Phase Filters

The Smoothed Average Phase Filter is a correlation filter built using multiple phase-only filters. The phase of the Fourier transformed image carries key shape features[11] which are sufficient to track the object in the later frames. The Smoothed Phase Filter (SPF) incorporates the features of the Phase Only Filter[10], while Gaussian smoothing is applied to smooth suppress any high frequency noise.

$$H_{SPF}(u,v) = G(u,v).\frac{F(u,v)}{|F(u,v)|} \tag{7}$$

$$h_i(u,v) = \alpha(H_{SPFi}(u,v)) + (1-\alpha)(H_{i-1}) \tag{8}$$

Equation 8 corresponds to updating the filter in each frame by exponential smoothing. Averaging multiple SPFs is reminiscent of the ASEF filters, but we show that SAPF requires lesser number of training samples than the ASEF.

IV. Tracking system architecture

Our tracking system is fairly general, allowing a wide variety of correlation filters to be essentially dropped into place. A high level system architecture is shown in Fig. 2. A video sequence from either a camera or an offline stream of frames is feed into the system, where it is double buffered in host memory to reduce IO latency, and then copied onto a CUDA capable graphics card. Once a frame has been loaded into the device memory, it is displayed to the user via OpenGL. In our system, the user selects a region of interest in the image of arbitrary size that he wishes to track, but this could easily be adapted to automatically track the output of an object detector such as a Viola-Jones detector.

Selecting a region of interest in the image causes the system to allocate required memory resources and initialize a filter from that ROI in the current frame. We perform normalized correlation of a subregion of each frame with the synthesized filters. By making the assumption that the target doesn't move too far between frames (quite true in case the target is much smaller than the image itself), the search space for the next frame is restricted by extracting an area of a fixed multiple of the filter size centered at the current predicted location of the target, and the peak correlation output is designated as the location of target in frame $t+1$ provided that the peak is above a threshold. All correlation and filter design is performed in the frequency domain using Nvidia's CUFFT library. The subregion of the image centered on the correlation peak in frame $t+1$ is extracted and used to update the filter for the subsequent frame.

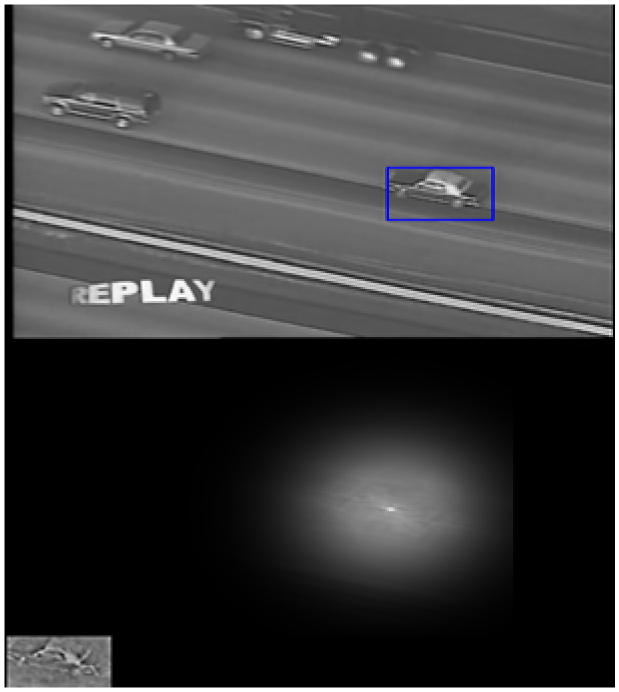

Fig. 3. Typical CUDA Tracking System output on a car chase video includes the tracked object location (top), the normalized correlation plane (middle), and the spatial domain representation of the filter used for tracking (bottom)

A Kalman filter is used to estimate the position, velocity and acceleration (P-A-V) of each tracked object and as noted above, the predicted position of each object at time $t{+}1$ is used to extract a search area. Additionally, the predicted velocity of the object is used to generate a prior PDF $\sim N(0,\sigma)$ mask that is is multiplied with the correlation plane to further restrict the tracker spatially.

V. Results and future work

Quantitative measures of filter performance between the various filters (MACE, UMACE, ASEF, SAPF) have not been performed yet.

Acknowledgment

We would like to thank all members of Carnegie Mellon CyLab Biometrics Center for their invaluable assistance and support.

References

[1] A. Yilmaz, O. Javed, and M. Shah, *Object tracking: A survey,* ACM Computing Surveys, vol. 38, no. 4, pp. 13+, December 2006. [Online]. Available: http://dx.doi.org/10.1145/1177352.1177355
[2] NVIDIA Corporation, *NVIDIA CUDA Programming Guide*, [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/CUDA_GDB_v3-0.pdf
[3] C.F. Hester and D. Casasent, *Multivariant technique for multiclass pattern recognition,* Appl. Opt. 19, pp.1758-1761, 1980.

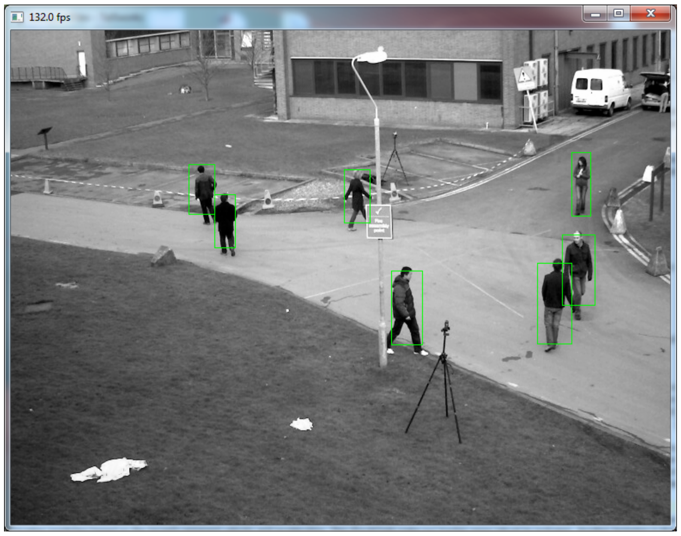

Fig. 4. Screen shot demonstrating simultaneously tracking 7 independent objects at 132 fps using GPU accelerated SAPFs.

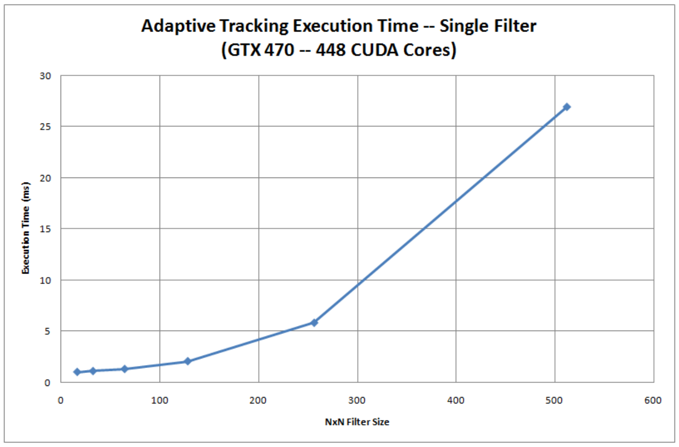

Fig. 5. Execution time for adaptive tracking using a single filter as a function of filter size (NxN) on a GTX 470 GPU.

[4] A. Mahalanobis, B.V.K. Vijaya Kumar, and D. Casasent, *Minimum average correlation energy filters,* Appl. Opt. 26, pp. 3633-3630, 1987.

[5] A. Mahalanobis, B.V.K. Vijaya Kumar, S. Song, S. Sims and J. Epperson, *Unconstrained correlation filters,* Appl. Opt. 33, pp. 37513759, 1994.

[6] J. Figue and Ph. Refrégiér, *Optimality of trade-off filters,* Appl. Opt. 32, pp. 1933-1935, 1993.

[7] D.S. Bolme, B.A. Draper, J.R. Beveridge, *Average of Synthetic Exact Filters,* Computer Vision and Pattern Recognition, June 2009.

[8] D. Kirk, and W. Hwu, *Programming Massively Parallel Processors,* Morgan Kaufmann Pub, 2010. [Book]

[9] M. Ekman, F. Warg, and J., *An in-depth look at computer performance growth,* ACM SIGARCH Computer Architecture News 33, 1, pp. 144-147, Mar 2005.

[10] J.L. Horner and P.D. Gianino, *Phase-only matched filtering,* Appl. Opt. 23, pp. 812-816, 1984.

[11] A.V. Oppenheim and J.S. Lim, *The importance of phase in signals,* Proceedings of the IEEE 69, pp. 529-541, 1981.