# Facial Feature Localization using GPU Accelerated Correlation Filters

Nicholas Vandal

nvandal@cmu.edu

Final Project Report

16-811: Math Fundamentals for Robotics

## 1.1 Abstract

This is the final project report for CMU 16-811: Math Fundamentals for Robotics, Fall 2009. It summarizes work done in implementing a GPU-based Eye Detector using MACE Filters.

## 1.2 Introduction and Problem Statement

Locating specific features (ie. nose, mouth, eyes, etc.) in an image of a face is an important precursor for many higher level computer vision tasks. More generally, pattern recognition is obviously a very important task with numerous applications, and as such a wide range of techniques, theory and research has gone into this problem. Especially in the realm of biometrics, accurately locating the features of a face is often crucial for facial geometry analysis or iris recognition algorithms. This project focused specifically on the task of locating the left and right eyes in an image of face that has already been normalized to a specific scale size. Throughout this project, face detection and normalization was done by manually cropping high resolution images of faces and rescaling to a fixed template size so as to eliminate additional complexity and overhead; however, in a real life system, detecting the face would typically be accomplished first by a separate face detector.

One standard methodology for pattern recognition is to extract features from an image and perform classification based on these features using a statistical classifier such as a SVM or other machine learning techniques. This project instead is concerned with the techniques of Correlation Pattern Recognition (CPR) which use correlation filters designed to operate directly on the pixels of an image. The basic idea behind CPR is to synthesize a reference signal and compare the degree to which an input signal resembles the reference signal using the correlation operator.

$$c(t) \triangleq \int_{-\infty}^{\infty} f(\tau) h^*(t+\tau)\, d\tau$$

Correlation can be performed in either the time/spatial domain or in the frequency domain.(1) When dealing with real world applications, there is often a premium placed on speed and achieving real-time performance. Especially when there are multiple correlation filters (potentially to detect or discriminate between several different classes of objects or to increase robustness), to be applied to a stream of high resolution images, it can be difficult to achieve high frame rates using conventional single-core general purpose CPUs. This difficulty is compounded by the fact that often in integrated biometrics systems, correlation filters are often but one component of a more complex scheme. Clearly, as in every computational problem, there is much impetus to implement CPR techniques as fast as possible. Implementations on parallel architectures are typically the best means of achieving the dramatic speedups necessary to achieve real-time performance for larger images and more pattern recognition algorithms. At the extreme, you have the intrinsic hardware level parallelism of ASICS, FPGAs and Optical Correlation Devices; however, while these hardware solutions are unparalleled in their speed, they are typically much less flexible, operate on fixed length data, and are more difficult to design and debug. In recent years, driven by the insatiable market demand for realistic 3D games, the Graphics Processing Unit (GPU) has evolved into a highly parallel, multithreaded, many-core processor of tremendous power. With the advent of increasingly more programmable GPUs, General-Purpose Computing on Graphics Processing Units (GPGPU) has emerged as an architecture ideally suited to algorithms with data-parallel operations and high arithmetic intensity. Nvidia's CUDA (Compute Unified Device Architecture) is a parallel programming model and software environment that leverages the computational horsepower of GPUs for general purpose computing through a C interface. (2)

The primary goal of this project is to investigate accelerating pattern recognition using MACE filters, a specific type of correlation filter, by using the GPGPU programming paradigm.

## 2.1 Correlation Filters

For all classes of correlation filters, the correlation output for an image in the spatial domain is given by the following equation (Eq 1.2), where *f(x,y)* is the input image and *h(x,y)* is the filter template.

$$c(x,y) = \iint f(\tau_x, \tau_y) h(x+\tau_x, y+\tau_y) d\tau_x d\tau_y$$

This cross-correlation defined above is the inner product of the filter and the input image for all possible shifts. Equivalently, the correlation plane *c(x,y)* can also be obtained using the Convolution Theorem, which states that a time/spatial domain convolution or correlation is equivalent to a element-wise product of Fourier transforms. This can be seen in (Eq 1.3) where *F(u,v)* is the 2-D Fourier transform of *f(x,y)* and *H(u,v)* is the 2-D Fourier transform of *h(x,y)*.

$$c(x,y) = \iint F(u,v) H^*(u,v) e^{i2\pi(ux+vy)} dudv$$

Using the FFT algorithm to compute the Fourier transforms of both *h(x,y)* and *f(x,y)*, performing element-wise multiplication, and then performing the inverse Fourier transform, is generally the preferred method to compute the correlation plane, as the FFT has an asymptotic complexity of $O(n\log_2 n)$, where *n* is the number of elements in both the image and the filter (assuming the filter and test image have the same dimensions for simplicity). Therefore, performing correlation in the frequency domain is dominated by the FFT run time, while in the spatial domain the complexity of convolution is $O(n^2)$. All further references to convolution in this project are performed in the frequency domain for efficiency and speed when processing large images.

Pattern recognition is performed by processing the convolution plane output and searching for well defined peaks in correlation energy. The locations of said peaks indicate the relative position of the detected object. The figure below illustrates the general steps for performing correlation in with the frequency domain with FFTs. This testing process remains the same for the various classes of filters, which differ only in how they are synthesized from the training images.

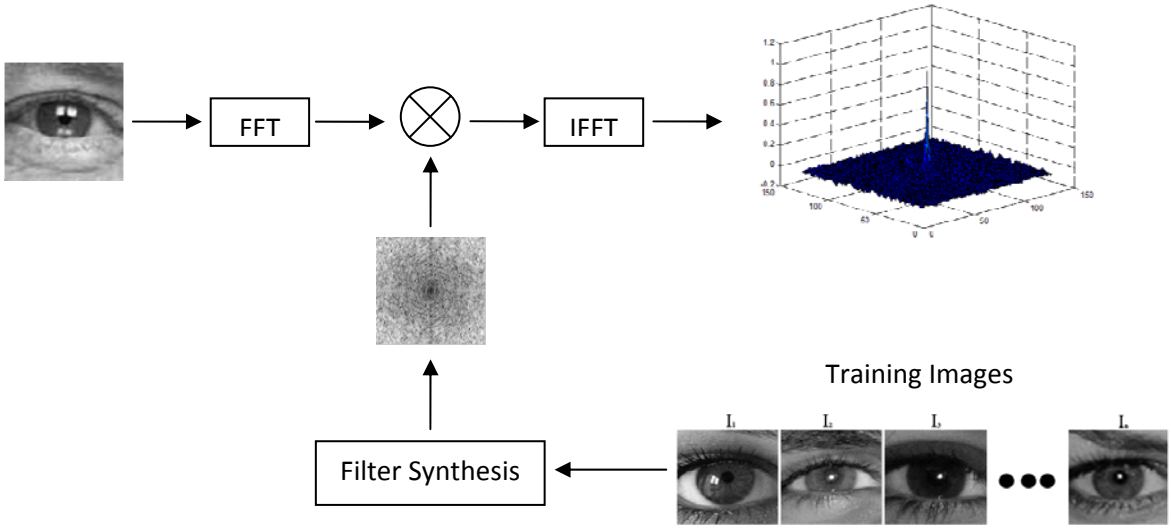



**Figure 1: Block diagram of FFT based correlation**

## 2.2 Matched Spatial Filters (MSF)

The Matched Spatial Filter is simply a replica of the reference image that one is looking for in the test image. This is the most basic type of CPR filter, and is ideal for detecting the reference image in a test image that has been corrupted by Gaussian noise. However, the MSF performs very poorly in response in any variation within a class, as well as scale and rotational distortions. Obviously it is not feasible to have a MSF for the eye of each person.(3)(4) Thus a large family of filters have been developed that attempt address this shortcoming.

## 2.3 Minimum Average Correlation Energy (MACE) Filters

The MACE filter is a class of filter developed by Mahalanobis et al (5) which minimizes the average correlation energy of all of the training images subject to the constraint that the correlation peak at the origin is fixed to a hard constraint (nominally 1 if the object is in the class). This minimization produces a correlation for training images within a class that have a very sharp peak at the origin of the correlation plane. MACE filters have a closed form solution that can be derived as follows.

Let $E_i$ be the energy of the *i*th spatial correlation plane $c_i(x,y)$ and $C_i(u,v)$ be the frequency equivalent.

$$E_i = \sum_{x=0}^{d-1}\sum_{y=0}^{d-1}\left|c_i(x,y)\right|^2 = \frac{1}{d^2}\sum_{u=0}^{d-1}\sum_{u=0}^{d-1}\left|C_i(u,v)\right|^2 \qquad \text{By Parseval's Theorm}$$

$$= \frac{1}{d^2}\sum_{u=0}^{d-1}\sum_{u=0}^{d-1}\left|H(u,v)\right|^2\left|F_i(u,v)\right|^2$$

$$E_i = \mathbf{h}^{\mathrm{T}}\mathbf{D}_i\mathbf{h}$$

Where we assume that the input images $f_i(x,y)$, the correlation outputs, and the resulting filter are all of size *dxd* and the there are *i=1,2,3….N* training images. $F_i(u,v)$ is the 2D-DFT of the *i*th training image, $\mathbf{D}_i$ is diagonal matrix of size $d^2$ x $d^2$ containing the power spectrum (the square of the magnitude of the DFT) of training image *i* along its diagonal, and **h** is a $d^2$x1 column vector that contains the rearranged 2-D correlation filter $H_i(u,v)$.

We seek to minimize:

$$E_{average} = \frac{1}{N}\sum_{i=1}^{N} E_i$$

This can be expressed also as:

$$E_{average} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{h}^{+}\mathbf{D}_i\mathbf{h} = \mathbf{h}^{+}\mathbf{D}\mathbf{h}$$

$$\text{where} \quad \mathbf{D} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{D}_i$$

Subject to the constraint: $\mathbf{X}^{+}\mathbf{h} = \mathbf{u}$

Where **X** is a $d^2$x*N* complex valued matrix whose columns contain the rearranged 2-D Fourier transform of the *ith* training image. The minimization problem with the linear constraint can be solved using Lagrange multipliers:

$$\Lambda(\mathbf{h},\lambda) = \mathbf{h}^{+}\mathbf{D}\mathbf{h} - \lambda\left(\mathbf{X}^{+}\mathbf{h} - \mathbf{u}\right)$$

$$\nabla\Lambda(\mathbf{h},\lambda) = \begin{pmatrix} \dfrac{\partial\Lambda(\mathbf{h},\lambda)}{\partial\mathbf{h}} \\ \dfrac{\partial\Lambda(\mathbf{h},\lambda)}{\partial\lambda} \end{pmatrix} = \begin{pmatrix} 2\mathbf{D}\mathbf{h} - \mathbf{X}\lambda \\ \mathbf{u} - \mathbf{X}\mathbf{h} \end{pmatrix} = 0$$

$$2\mathbf{D}\mathbf{h} - \mathbf{X}\lambda = 0 \rightarrow \mathbf{D}\mathbf{h} = \mathbf{X}\frac{\lambda}{2} \rightarrow \mathbf{h} = \mathbf{D}^{-1}\mathbf{X}\frac{\lambda}{2}$$

$$\mathbf{X}^{+}\left(\mathbf{D}^{-1}\mathbf{X}\frac{\lambda}{2}\right)=\mathbf{u}\rightarrow\mathbf{X}^{+}\mathbf{D}^{-1}\mathbf{X}\frac{\lambda}{2}=\mathbf{u}\rightarrow\frac{\lambda}{2}=\left(\mathbf{X}^{+}\mathbf{D}^{-1}\mathbf{X}\right)^{-1}\mathbf{u}$$

$$\boxed{\mathbf{h}=\mathbf{D}^{-1}\mathbf{X}\left(\mathbf{X}^{+}\mathbf{D}^{-1}\mathbf{X}\right)^{-1}\mathbf{u}}$$

Where **h** is the frequency domain MACE filter arranged in a $d^2$x*1* column vector, and + is the complex conjugate transpose. (4)

2.4 Unconstrained Minimum Average Correlation Energy (UMACE) Filters

The UMACE filter is a variant on the MACE filter, which seeks to minimize the ACE while maximizing the average correlation values at the origin instead of constraining them to a specific value. (6) This filter also has a closed form solution that can be derived as follows:

The average correlation peak of all of the training images can be expressed as the inner product of the frequency domain filter **h** and **m** is the rearranged column vector of the 2D DFT of the average of all of the training images.

$$\left|\mathbf{h}^{+}\mathbf{m}\right|^{2}=\mathbf{h}^{+}\mathbf{m}\mathbf{m}^{+}\mathbf{h}$$

Minimizing the ACE while maximizing the average correlation peak therefore is written as the optimization function:

$$J\left(\mathbf{h}\right)=\frac{\mathbf{h}^{+}\mathbf{m}\mathbf{m}^{+}\mathbf{h}}{\mathbf{h}^{+}\mathbf{D}\mathbf{h}}$$

Setting the derivative equal to zero and solving for critical points:

$$\frac{\partial J\left(\mathbf{h}\right)}{\partial\mathbf{h}}=\frac{2\mathbf{h}^{+}\mathbf{D}\mathbf{m}\mathbf{m}^{+}\mathbf{h}-2\mathbf{h}^{+}\mathbf{m}\mathbf{m}^{+}\mathbf{h}\mathbf{D}\mathbf{h}}{\left|\mathbf{h}^{+}\mathbf{D}\mathbf{h}\right|^{2}}=0$$

$$\frac{\mathbf{h}^{+}\mathbf{D}\mathbf{h}\mathbf{m}\mathbf{m}^{+}\mathbf{h}}{\mathbf{h}^{+}\mathbf{D}\mathbf{h}}-\frac{\mathbf{h}^{+}\mathbf{m}\mathbf{m}^{+}\mathbf{h}\mathbf{D}\mathbf{h}}{\mathbf{h}^{+}\mathbf{D}\mathbf{h}}=0$$

$$\frac{\mathbf{h}^{+}\mathbf{D}\mathbf{h}\mathbf{m}\mathbf{m}^{+}\mathbf{h}}{\mathbf{h}^{+}\mathbf{D}\mathbf{h}}-\left(\frac{\mathbf{h}^{+}\mathbf{m}\mathbf{m}^{+}\mathbf{h}}{\mathbf{h}^{+}\mathbf{D}\mathbf{h}}\right)\mathbf{D}\mathbf{h}=0$$

$$\frac{\left(\mathbf{h}^{+}\mathbf{D}\mathbf{h}\right)\mathbf{m}\mathbf{m}^{+}\mathbf{h}}{\left(\mathbf{h}^{+}\mathbf{D}\mathbf{h}\right)}-J\left(\mathbf{h}\right)\mathbf{D}\mathbf{h}=0$$

$$\mathbf{m}\mathbf{m}^{+}\mathbf{h}-J\left(\mathbf{h}\right)\mathbf{D}\mathbf{h}=0$$

This is a generalized eigenvalue/eigenvector problem.

Letting $\lambda=\mathbf{J}\left(\mathbf{h}\right)$ then $\mathbf{m}\mathbf{m}^{+}\mathbf{h}=\lambda\mathbf{D}\mathbf{h}$

Since we are maximizing $\mathbf{J}\left(\mathbf{h}\right)$ we solve the eigenvalue problem and select the eigenvector corresponding to the largest eigenvalue. However, we also observe that $\mathbf{m}\mathbf{m}^{+}$ is an outer product of a vector with itself and therefore $\mathbf{m}\mathbf{m}^{+}$ has a rank of one and therefore there is only one nonzero eigenvalue and only one valid eigenvector solution

to this problem. Rewriting $\mathbf{mm}^{+}\mathbf{h} = \lambda\mathbf{Dh}$ yields $\mathbf{h} = \frac{\alpha}{\lambda}\mathbf{D}^{-1}\mathbf{m}$. We can ignore this constant as the final metric of peak sharpness PSR is invariant to a constant scaling.

$$\boxed{\mathbf{h} = \mathbf{D}^{-1}\mathbf{m}}$$

Where. Observe that **D** in both the MACE and UMACE Filters is a diagonal matrix, so the inverse operation is trivial.

## 2.5 Peak-to-Sidelobe Ratio (PSR)

This metric is the metric used to quantify how close of a match the input image is to the filter. It is calculated as follows where the sidelobe is an annular region surrounding the peak value of the spatial correlation plane:

$$PSR = \frac{peak - \mu_{sidelobe}}{\sigma_{sidelobe}}$$

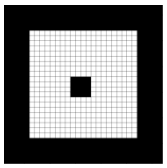


**Figure 2: Sidelobe Region**

## 3.1 CPU/GPU Implementation of FFT Based MACE Filters

Initial development and debugging of filter training and detector scripts was performed in MATLAB due to ease of development and rich libraries. MATLAB implementation was very slow, so to provide a fair comparison of to C for CUDA code, implemented detector code in C, using the open source FFTW library for FFT routines and OpenGL for displaying the input image and detection locations. The GPU implementation was derived from the single threaded C code. It is written in C for CUDA and uses the CUFFT library for executing FFT transforms and OpenGL for display purposes.

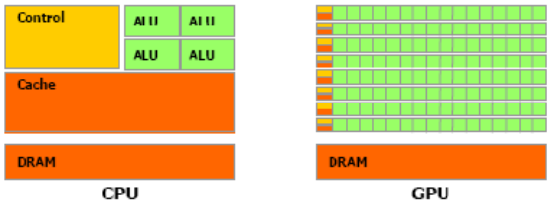



**Figure 3: CPU/GPU Architecture. Illustrates the far greater number of transistors devoted data processing**

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

**Figure 4: Example element-wise addition kernel**

```
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
```

**Figure 5: Kernel launch host code**

C for CUDA extends C by allowing the developer to define C functions known as kernels, which when called are executed in parallel by N separate threads on the GPU by individual scalar processor cores. Nvidia terms this architecture SIMT (single instruction, multiple thread). The CUDA programming model is designed to provide fine-grained data and thread parallelism nested inside of coarse-grain data and task parallelism through a hierarchy of thread groups, shared memory, and barrier synchronization. (2) There are two major conceptual differences between the host system and the device where the kernels are executed that are important to understand for maximum performance of GPU code: threads, which on the GPU are extremely lightweight (virtually zero penalty for a context switch) and numerous (up to 30,000 actively running, potentially billions in queue) ; and memory, which on the device is physically and virtually divided into different types (registers, shared, local, global, constant, and texture) that each have a specific purpose and properties. (7)

The CUDA implementation of the FFT filtering remains at an early level of optimization. While CUDA has a low learning curve, and enables rapid GPU code production, to get maximum speedup requires fine tuning, and most crucially taking advantage of the memory hierarchy. Filter training remains as MATLAB code as it is not a time critical application. Additionally, filter synthesis is performed using circular correlation (ie. not adequately padding

the training images of eye regions. In the localization code, the smaller eye-sized filter is padded in the spatial domain and transformed back into the frequency domain to get filter that will perform correlation over the entire larger image. Additionally both the image and the frequency domain filter are padded to the next higher power of two greater than ($I_{dim}$+$F_{dim}$-1). This padding process performs linear correlation of the filter with the image. See code appendix for additional details.

## 3.2 Results

The primary focus and purpose of this project was the investigation speedups possible with heterogeneous GPU computation and not necessarily achieving peak pattern recognition performance with correlation filters or evaluating said performance against other pattern recognition techniques. A set of filters were synthesized that worked nominally well, but priority of effort was given towards coding fast CPU and GPU implementations of the correlation test stage.

Below is a screen shot from one run of the CUDA implementation using a UMACE filter and the corresponding correlation plane for the left eye. The window title bar indicates execution time for this image (both eyes), including all host-device memory transfers, as well as the frame rate. It should be noted that the eyes extracted from this example image were one of 44 used to synthesize this UMACE filter—resulting in the very sharp and high peak in the correlation plane.

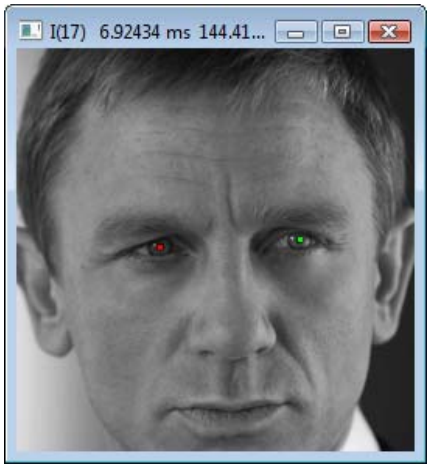



**Figure 6: Eye locations detected**

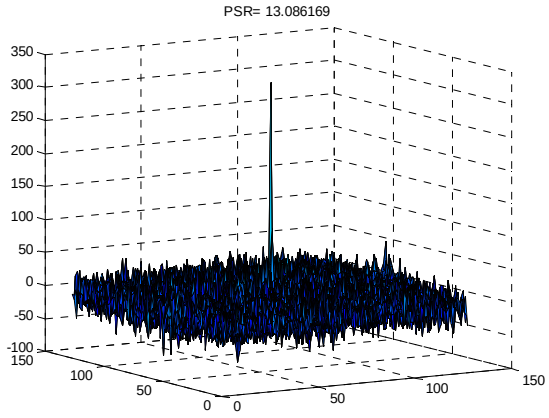
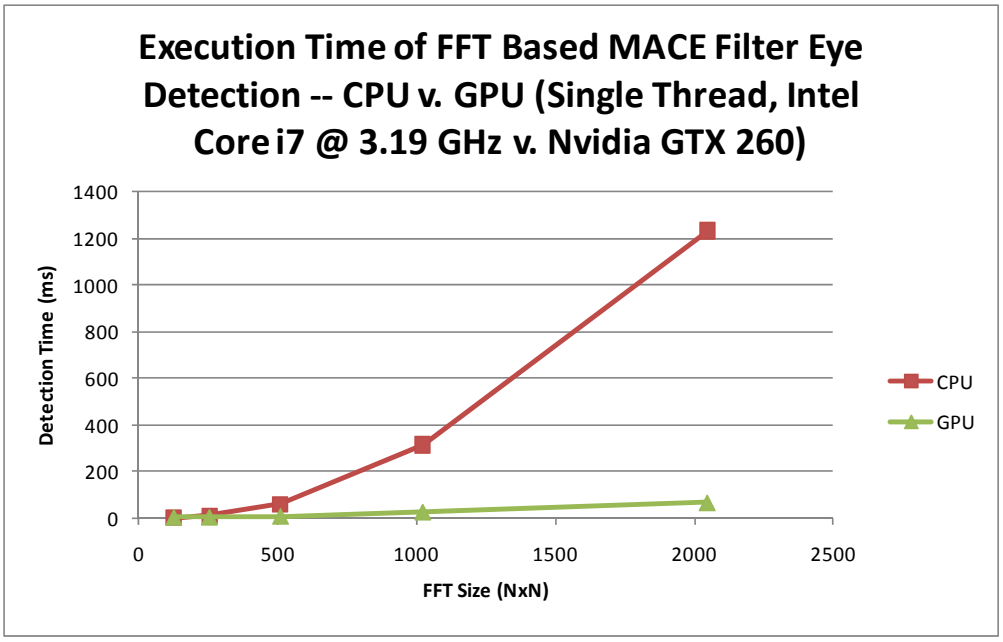
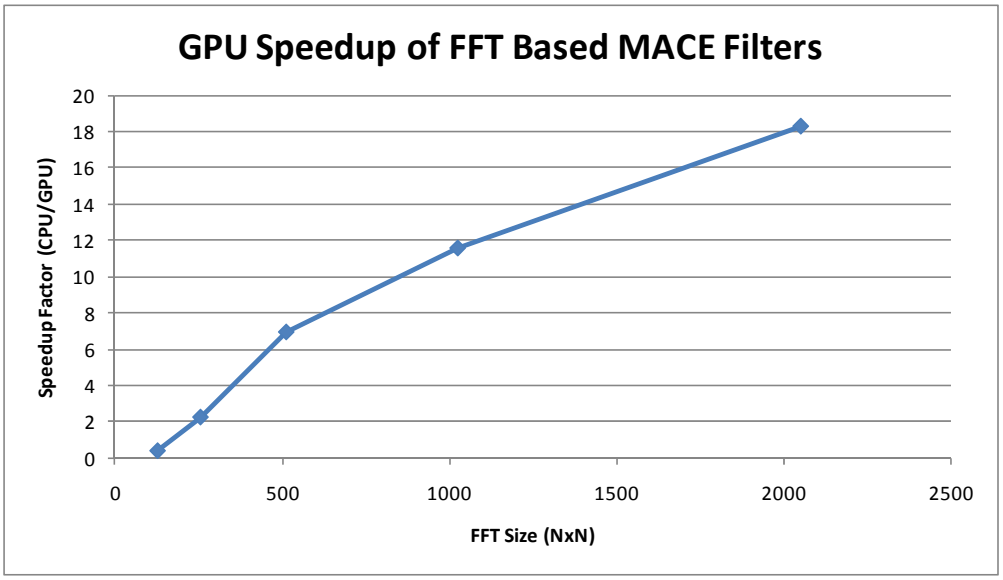



**Figure 7: Correlation plane corresponding to left eye**

All filter FFTs, device and host memory allocations, and other aspects of the computation that remain constant across different images or multiple runs of the same image are pre-computed and pre-loaded into the GPU prior to the start of the main rendering loop. This simulates a realistic scenario in which a stream of video footage is processed. Execution times were averaged amongst a set of 45 images and used to generate the charts and plots shown below.

| Filter Size | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| Image Size | 64 | 128 | 256 | 512 | 1024 |
| Padded FFT Size | 128 | 256 | 512 | 1024 | 2048 |
| | | | | | |
| Detection Execution Time (ms) | | | | | |
| CPU | 1.882 | 10.877 | 59.8 | 315.6 | 1233.1 |
| GPU | 4.53 | 4.8 | 8.607 | 27.29 | 67.5 |
| | | | | | |
| GPU Speedup | 0.4154525 | 2.266042 | 6.947833 | 11.56468 | 18.26815 |




## 4.1 Conclusion/Future Work

The net result of this project is the development of a GPU-enabled FFT-correlation based detection framework that can essentially accommodate any frequency domain filter bank, not just MACE and UMACE filters. A speedup in the range of 18X versus a single threaded CPU implementation is a fairly significant increase, but represents the low end of what is possible with more fully optimized code GPU code that takes advantage of shared and texture memory. Future work will include more completely optimizing this code, integration of more advanced filter banks such as Average of Synthetic Exact Filters (ASEF) (3) which offers greater pattern recognition performance, and/or 'Corefaces' which replaces the single MACE filter template with a linear subspace expansion (4), and eventually integration into a GPU accelerated Viola-Jones face detector which I have been developing for the past semester. The end goal is a system capable of processing multi-megapixel HD video in real-time.

# References

1. B. V. K. Vijaya Kumar, Abhijit Mahalanobis, Richard D. Juday. *Correlation Pattern Recognition.* Cambridge : Cambridge University Press, 2005.

2. NVIDIA Corporation. *Nvidia CUDA Programming Guide.* s.l. : Nvidia, 2009.

3. *Average of Synthetic Exact Filters.* Bolme, David S., Draper, Bruce A., Beveridge Ross J. June 2009, s.l. : Computer Vision and Pattern Recognition. Colorado State University.

4. Savvides, Marios. *Reduced Complexity Face Recognition using Advanced Correlation Filters and Fourier Subspace Methods for Biometric Applications.* s.l. : Department of Electrical and Computer Engineering, Carnegie Mellon University, 2004.

5. *Minimum average correlation energy filters.* A. Mahalanobis, B.V.K. Vijaya Kumar, and D. Casasent. 26, s.l. : Appl. Opt., 1987, pp. 3633-3640.

6. *Efficient Design of Advanced Correlation Filters for Robust Distortion-Tolerant Face Recognition.* Marios Savvides, B.V.K. Vijaya Kumar. Miami, Florida : IEEE International Conference on Advanced Video and Signal Based Surveillance , 2003.

7. NVIDIA Corporation. NVIDIA CUDA Best Practices Guide. [Online] 2009.