Accessing the GPU & the GPUImage Library

Instructor - Simon Lucey

16-423 - Designing Computer Vision Apps
Today

- Motivation
- GPU
- OpenGL
- GPUImage Library
Correlation Filters with Limited Boundaries

Hamed Kiani Galoogahi
Istituto Italiano di Technologia
Genova, Italy
hamed.kiani@iit.it

Terence Sim
National University of Singapore
Singapore
tsim@comp.nus.edu.sg

Simon Lucey
Carnegie Mellon University
Pittsburgh, USA
slucey@cs.cmu.edu

Abstract

Correlation filters take advantage of specific properties in the Fourier domain allowing them to be estimated efficiently: \(O(N D \log D)\) in the frequency domain, versus \(O(D^3 + N D^2)\) spatially where \(D\) is signal length, and \(N\) is the number of signals. Recent extensions to correlation filters, such as MOSSE, have reignited interest in their use in the vision community due to their robustness and attractive computational properties. In this paper we demonstrate, however, that this computational efficiency comes at a cost. Specifically, we demonstrate that only \(\frac{D}{2}\) proportion of shifted examples are unaffected by boundary effects which has a dramatic effect on detection/tracking performance. In this paper we propose a novel approach to correlation filter estimation that: (i) takes advantage of inherent computational redundancies in the frequency domain, (ii) dramatically reduces boundary effects, and (iii) is able to implicitly exploit all possible patches densely extracted from training examples during learning process. Impressively object tracking and detection results are presented in terms of both accuracy and computational efficiency.

1. Introduction

Correlation between two signals is a standard approach to feature detection/matching. Correlation touches nearly every facet of computer vision from pattern detection to object tracking. Correlation is rarely performed naively in the spatial domain. Instead, the fast Fourier transform (FFT) affords the efficient application of correlating a desired template/filter with a signal.

Correlation filters, developed initially in the seminal work of Hester and Casasent [15], are a method for learning a template/filter in the frequency domain that rose to some prominence in the 80s and 90s. Although many variants have been proposed [15, 18, 20, 19], the approach’s central tenet is to learn a filter, that when correlated with a set of training signals, gives a desired response, e.g. Figure 1 (b). Like correlation, one of the central advantages of the approach is that it attempts to learn the filter in the frequency domain due to the efficiency of correlation in that domain. Interest in correlation filters has been reignited in the vision world through the recent work of Bolme et al. [5] on Minimum Output Sum of Squared Error (MOSSE) correlation filters for object detection and tracking. Bolme et al.’s work was able to circumvent some of the classical problems...
// 5. Now apply some OpenCV operations
cv::Mat gray; cv::cvtColor(cvImage, gray, CV_RGBA2GRAY); // Convert to grayscale
cv::GaussianBlur(gray, gray, cv::Size(5,5), 1.2, 1.2); // Apply Gaussian blur
cv::Mat edges; cv::Canny(gray, edges, 0, 50); // Estimate edge map using Canny edge detector
**SIMD Vector Extensions**

- **What is it?**
  - Extension of the ISA
  - Data types and instructions for the parallel computation on short (length 2, 4, 8, …) vectors of integers or floats
  - Names: MMX, SSE, SSE2, …

- **Why do they exist?**
  - **Useful:** Many applications have the necessary fine-grain parallelism
  - **Then:** speedup by a factor close to vector length
  - **Doable:** Relative easy to design; chip designers have enough transistors to play with

**Register width**

- 64 bit
- 128 bit
- 256 bit

**Intel x86 Processors**

- 8086
- 286
- 386
- 486
- Pentium
- Pentium MMX
- Pentium III
- Pentium 4
- Pentium 4E
- Pentium 4F
- Core 2 Duo
- Penryn
- Core i7 (Nehalem)
- Sandy Bridge
- Haswell
- Core i7 (Haswell)

**AVX (Advanced Vector Extensions)**

- AVX2

**What is SIMD?**

- Single Instruction, Multiple Data (SIMD)

**4-way**
The GPU shares the system bus with the CPU. This is especially important for handheld devices, where the GPU is often used as a purpose accelerator. The transformation of a 2D image is done by applying 1D FFT to its rows and columns consecutively. The precision of 1/256 is used for pixel processing, while an 8-bit fixed point format is used for per-vertex computations. The GPU is 3 times faster than the CPU for various image and signal processing algorithms, on a Nvidia Tegra CPU, 1GB of RAM, a core ARM Cortex A9 CPU, and GPU. The transform of a 2D image is done by applying 1D FFT to its rows and columns consecutively. The GPU is 3 times faster than the CPU for various image and signal processing algorithms. The transformation of a 2D image is done by applying 1D FFT to its rows and columns consecutively. The GPU is 3 times faster than the CPU for various image and signal processing algorithms.
Today

- Motivation
- GPU
- OpenGL
- GPUImage Library
UP TO 84x FASTER

2010

2014

iPhone 6

GPU PERFORMANCE
GFX Bench - Manhattan 3.0
ARM processor - FHD (1920x1080) devices 2015/05/24 [link]

<table>
<thead>
<tr>
<th>Device</th>
<th>Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Tango</td>
<td>1798</td>
</tr>
<tr>
<td>Apple iPhone 6</td>
<td>1793</td>
</tr>
<tr>
<td>NVIDIA Shield</td>
<td>1675</td>
</tr>
<tr>
<td>Apple iPad Air 2</td>
<td>1671</td>
</tr>
<tr>
<td>Apple iPhone 5S</td>
<td>1665</td>
</tr>
<tr>
<td>Sony Experia Z3</td>
<td>1621</td>
</tr>
<tr>
<td>Opus E350</td>
<td>1548</td>
</tr>
<tr>
<td>ZTE Grand S Pro</td>
<td>1373</td>
</tr>
<tr>
<td>Google Nexus 9</td>
<td>1351</td>
</tr>
<tr>
<td>ZTE NX510J</td>
<td>1337</td>
</tr>
<tr>
<td>HTC One M9</td>
<td>1259</td>
</tr>
<tr>
<td>LG G Flex 2</td>
<td>1232</td>
</tr>
</tbody>
</table>

(Taken from YouTube Tango Talk 2015)
OpenCL versus CUDA

- **Open Computing Language (OpenCL)**
  - OpenCL is the currently the dominant open general-purpose GPU computing language, and is an open standard.
  - OpenCL is actively supported on Intel, AMD, Nvidia and ARM platforms.
  - OpenCL is based on the C99 language.

- **Compute Unified Device Architecture (CUDA)**
  - Dominant proprietary (NVIDIA) framework.
  - Designed to work with well known languages such as C, C++ and Fortran.
  - OpenCV 3.0 now has support for both.
  - Neither are supported in iOS, so we cannot use them :(. 

Today

- Motivation
- GPU
- OpenGL
- GPUImage Library
What is OpenGL?

- OpenGL is a graphics API
  - Portable software library (platform-independent)
  - Layer between programmer and graphics hardware
  - Uniform instruction set (hides different capabilities)
- OpenGL can fit in many places
  - Between application and graphics system
  - Between higher level API and graphics system
- Why do we need OpenGL or an API?
  - Encapsulates many basic functions of 2D/3D graphics
  - Think of it as high-level language (C++) for graphics
  - History: Introduced SGI in 92, maintained by Khronos
  - Precursor for DirectX, WebGL, Java3D etc.
- OpenGL is platform independent.
OpenGL

• Since 2003, can write vertex/pixel shaders.
• Fixed function pipeline special type of shader.
• Like writing C programs.
• Performance >> CPU (even used for non-graphics).
• Operate in parallel on all vertices or fragments.
OpenGL ES

- ES stands for Embedded Systems (ES).
- Subset of OpenGL API
  - Libraries GLUT and GLU not available.
- Designed for embedded systems like smart devices.
- Released in 2003, also maintained by Khronos.
The “World” is Triangular!!!
The “World” is Triangular!!!
OpenGL Pipeline

- **Vertex array**: location of vertex in 3D space.
- **Vertex Shader**: at a minimum calculates the projected position of the vertex in screen space.
- **Triangle Assembly**: connects the projected vertices.
- **Rasterization**: breaks the remaining visible parts into pixel-sized fragments.
- **Fragment Shader**: texture mapping and lighting.
- **Testing & Blending**: discards fragments from objects that are behind the ones already drawn.
- **Framebuffers**: final destination for the rendering job.

Taken from: [http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-1.-The-Graphics-Pipeline.html#gl1-pipeline](http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-1.-The-Graphics-Pipeline.html#gl1-pipeline)
Programmable Shaders

- A **Shader** is a user-defined program designed to run on some stage of a graphics processor.
  - Its purpose is to execute one of the programmable stages of the rendering pipeline.
  - Since shaders are programmable, they are increasingly been used for non-graphics applications - such as computer vision operations.
- Running custom filters on the GPU using OpenGL ES requires a lot of code to set up and maintain :(.
- Much of the code is boilerplate, however, it is extremely cumbersome to build up a full application to test out ideas in vision using OpenGL ES.
Why the GPU?

• Vertices, pixel fragments, and pixels are largely independent.
• Most of these entities can therefore be processed in parallel.
• For example,
  • 3 vertices of a triangle can be processed in parallel.
  • two triangles can be rasterized in parallel, etc.
• The rise of GPUs over the last two decades has been motivated by this inherent parallelism.
Today

• Motivation
• GPU
• OpenGL
• GPUImage Library
GPUImage Library

• BSD-licensed iOS library that lets you apply GPU-accelerated filters and other effects to images, live camera video and movies.
• Allows you to write your own custom filters in OpenGL-ES.
• Released in 2012 and developed by Brad Larson.
• GPUImage for Android now also exists.
GPUImage

- GPUImage can do many things OpenCV can do, but much faster through the GPU -
  - Color conversions (grayscale, RGB2HSV, etc.)
  - Image processing (image warping, cropping, blurring, edges, etc.)
  - Blending (drawing lines, points, etc.)
  - Visual effects (pixellate, sketch, etc.)
  - Computer vision (interest point detectors, hough transform, etc.)
- Check out - https://github.com/BradLarson/GPUImage for a full description of the capabilities.
GPUImage vs CoreImage

• There exists an internal framework in iOS called CoreImage that can do some of the things GPUImage can do.
• GPUImage is preferred in vision applications as,
  • You can seamlessly integrate filters with CPU C++ code using `GPUImageRawData` (more on this in later lectures)
  • All filters are written in OpenGL ES, so you can write custom filters if necessary.
  • Code is more portable (i.e. Android).

```cpp
varying highp vec2 textureCoordinate;
uniform sampler2D inputImageTexture;

void main()
{
  lowp vec4 textureColor = texture2D(inputImageTexture, textureCoordinate);
  lowp vec4 outputColor;
  outputColor.r = (textureColor.r * 0.393) + (textureColor.g * 0.769) + (textureColor.b * 0.189);
  outputColor.g = (textureColor.r * 0.349) + (textureColor.g * 0.686) + (textureColor.b * 0.168);
  outputColor.b = (textureColor.r * 0.272) + (textureColor.g * 0.534) + (textureColor.b * 0.131);
  outputColor.a = 1.0;

  gl_FragColor = outputColor;
}
GPUImage
GPUImage in Xcode

```
// Intro_GPUImage

// Created by Simon Lucey on 9/23/15.
// Copyright (c) 2015 CMU_16432. All rights
//

#import "ViewController.h"

#import <GPUImage/GPUImage.h>

@interface ViewController ()
    // Setup the view (this time using GPUImage)
    GPUImageView *imageView_
@end

@end
```
Playing with GPUImage

• We are now going to have a play with GPUImage.
• On your browser please go to the address,

https://github.com/slucey-cs-cmu-edu/Intro_GPUImage

• Or better yet, if you have git installed you can type from the command line.

$ git clone https://github.com/slucey-cs-cmu-edu/Intro_GPUImage.git
#import "ViewController.h"
#import <GPUImage/GPUImage.h>

@interface ViewController () {
    // Setup the view (this time using GPUImageView)
    GPUImage *imageView_
}
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // Setup GPUImage (not we are not using UIImageView here)........
    imageView_ = [[GPUImage alloc] initWithFrame:CGRectMake(0, 0, 0, self)

    // Important: add as a subview
    [self.view addSubview:imageView_];
}
@end
Playing with GPUImage

```
// Read in the image (of the famous Lena)
UIImage *inputImage = [UIImage imageNamed:@"lena.png"];

// Initialize filters
GPUImagePicture *stillImageSource = [[GPUImagePicture alloc] initWithImage:inputImage];
GPUImageSepiaFilter *stillImageFilter = [[GPUImageSepiaFilter alloc] init];

// Daisy chain the filters together (you can add as many filters as you like)
[stillImageSource addTarget:stillImageFilter];
[stillImageFilter addTarget:imageView_];

// Process the image
[stillImageSource processImage];

@end
```
GPUImage for Movies

```objc
// ViewController.m
// Movie_GPUImage
// Created by Simon Lucey on 9/24/15.
// Copyright © 2015 OMU_16432. All rights reserved.

#import "ViewController.h"
#import <GPUImage/GPUImage.h>

@interface ViewController ()
@end

@implementation ViewController

- (void)viewDidLoad {
    // Do any additional setup after loading the view, typically from a nib.

    // Setup GPUImage (not we are not using UIimageView here).......
    // videoView_ = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.width, self.view.frame.size.height)];
    // Important: add as a subview
    // [self.view addSubview:videoView_];

    // Set the movie file to read
    NSURL *sampleURL = [[NSBundle mainBundle] URLForResource:@"simon" withExtension:@"mov"];
    GPUImageMovie *movieFile = [[GPUImageMovie alloc] initWithURL:sampleURL];
    movieFile.runBenchmark = NO;
    movieFile.playAtActualSpeed = YES;

    // Initialize filters
    GPUImageSepiaFilter *sepiaFilter = [[GPUImageSepiaFilter alloc] init];
    // Daisy chain the filters together (you can add as many filters as you like)
    [movieFile addTarget:sepiaFilter];
    [sepiaFilter addTarget:movieView_];

    // Rotates the video right so it displays properly
    // [sepiaFilter setOutputRotation:KGPUImageRotateRightAtIndex:0];
    // Process the movie
    // [movieFile startProcessing];

    - (void)didReceiveMemoryWarning {
        [super didReceiveMemoryWarning];
        // Dispose of any resources that can be recreated.
    }
}
@end
```
Movies with GPUImage

• On your browser please go to the address,

https://github.com/slucey-cs-cmu-edu/Movie_GPUImage

• Or better yet, if you have git installed you can type from the command line.

$ git clone https://github.com/slucey-cs-cmu-edu/Movie_GPUImage
What About?
For more information on MetalGL check out - [https://metalgl.com/](https://metalgl.com/)
Supposedly 3x Faster!!

For more information on MetalGL check out - [https://metalgl.com/](https://metalgl.com/)
For more information on MetalGL check out - https://metalgl.com/
More Examples to Play With…

- Download the complete GPUImage library from,
  - [https://github.com/BradLarson/GPUImage](https://github.com/BradLarson/GPUImage)

- In there you will find a fair amount of example code,
  - SimpleVideoFileFilter
  - FilterShowCase
  - MultiViewFilterExample
  - BenchmarkSuite
  - RawDataTest