

# Accessing the GPU & the GPUImage Library

Instructor - Simon Lucey

**16-623 - Advanced Computer Vision Apps**

# Today

---

- Motivation
- GPU
- OpenGL
- GPUImage Library

Algorithm

Software

Architecture

SOC Hardware

Algorithm

Software

Architecture

SOC Hardware

## Correlation Filters with Limited Boundaries

Hamed Kiani Galoogahi  
Istituto Italiano di Tecnologia  
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:  $\mathcal{O}(ND \log D)$  in the frequency domain, versus  $\mathcal{O}(D^3 + ND^2)$  spatially where  $D$  is signal length, and  $N$  is the number of signals. Recent extensions to correlation filters, such as MOSSE, have reignited interest of 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  $\frac{1}{D}$  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 redundancy 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. Impressive 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 ap-

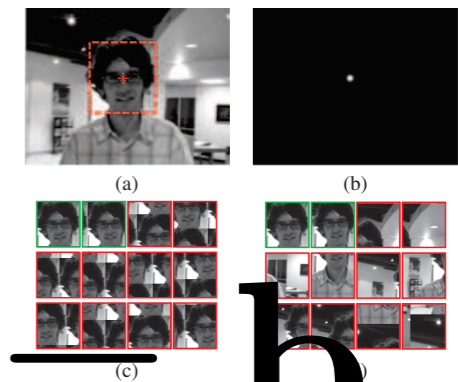


Figure 1. (a) Defines the example fixed spatial support within the image from which the peak correlation output should occur. (b) The desired output response, based on (a), of the correlation filter when applied to the entire image. (c) A subset of patch examples used in a canonical correlation filter where green denotes a non-zero correlation output, and red denotes a zero correlation output in direct accordance with (b). (d) A subset of patch examples used in our proposed correlation filter. Note that our proposed approach uses all possible patches stemming from different parts of the image, whereas the canonical correlation filter simply employs circular shifted versions of the same single patch. The central dilemma in this paper is how to perform (d) efficiently in the Fourier domain. The two last patches of (d) show that  $\frac{D-1}{T}$  patches near the image border are affected by circular shift in our method which can be greatly diminished by choosing  $D \ll T$ , where  $D$  and  $T$  indicate the length of the vectorized face patch in (a) and the whole image in (a), respectively.

proach 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

Algorithm

Software



```
// 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
```

Swift

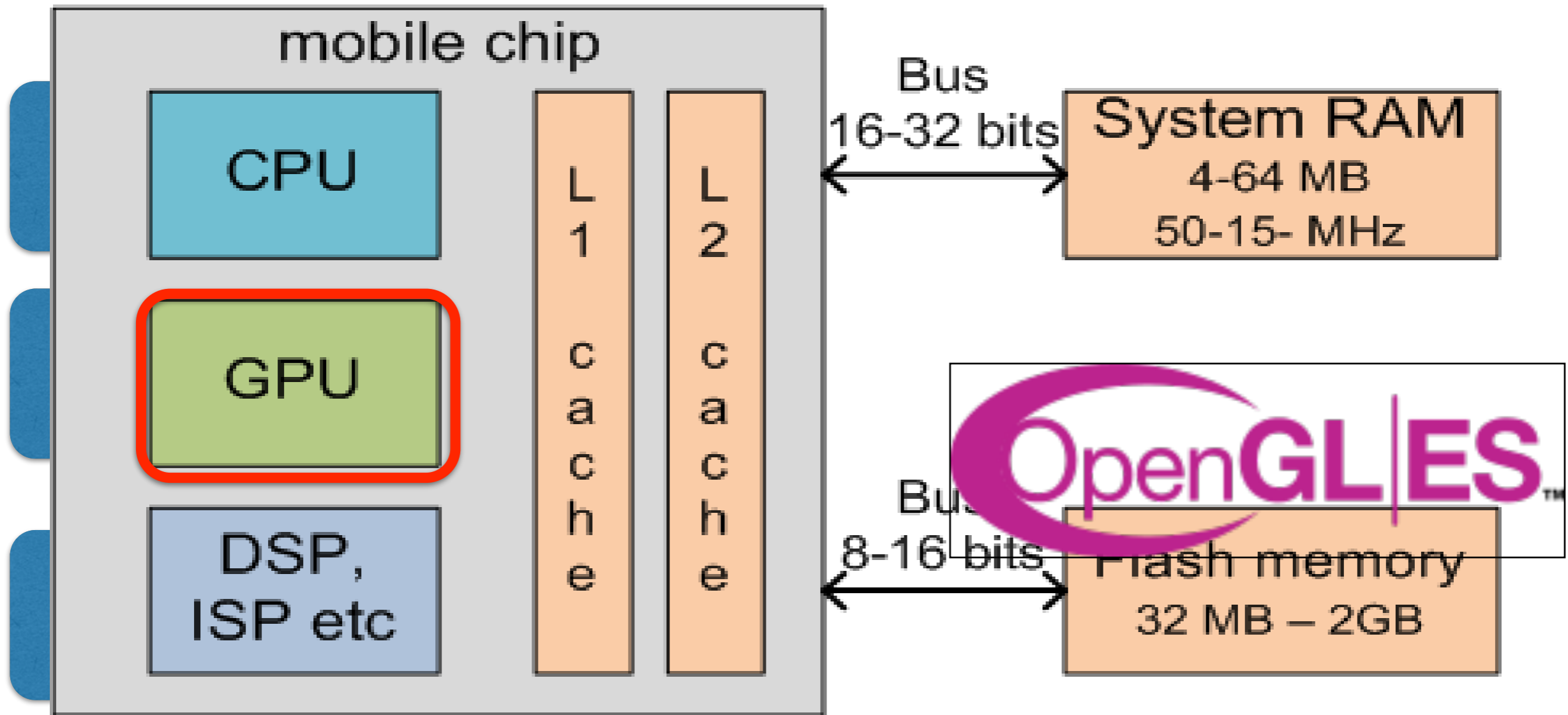
Algorithm



**SIMD (Single Instruction, Multiple Data)**

Architecture

SOC Hardware



SOC Hardware

# Reminder: Alternatives to OpenCV

**FastCV Computer Vision SDK**

A product of Qualcomm Technologies, Inc.

(<https://developer.qualcomm.com/software/fastcv-sdk>)

The logo for OpenVX, featuring the word "OpenVX" in a large, red, stylized font. Below it, the word "KRONOS" is written in a smaller, black, sans-serif font, with "GROUP" underneath. A red circular graphic element is positioned to the left of the "OpenVX" text.

(<https://www.khronos.org/openvx/>)



**Accelerated CV**

(<http://opencv.org/itseez-announces-release-of-accelerated-cv-library.html>)



**GPUImage**

(<https://github.com/BradLarson/GPUImage>)



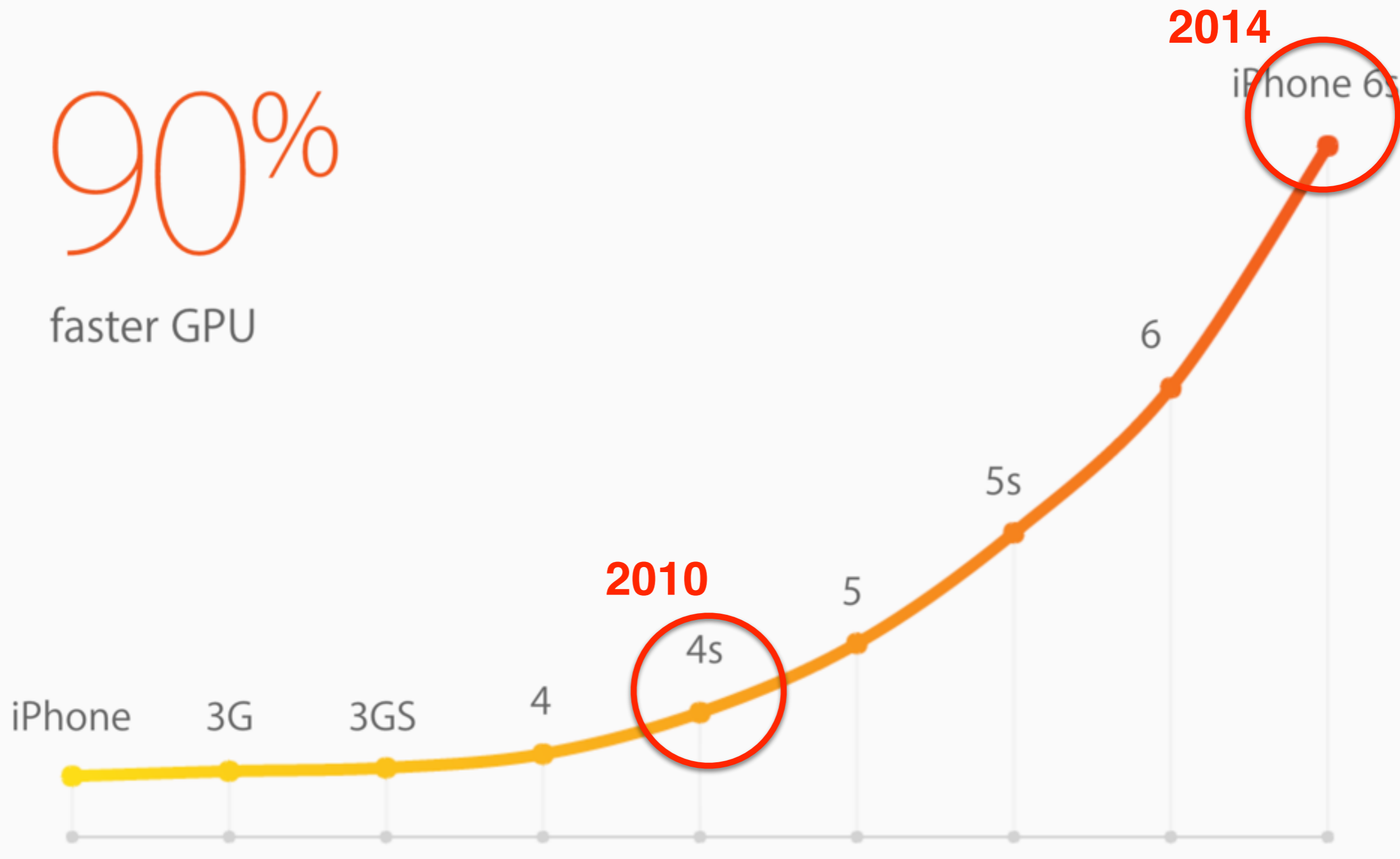
# Today

---

- Motivation
- GPU
- OpenGL
- GPUImage Library

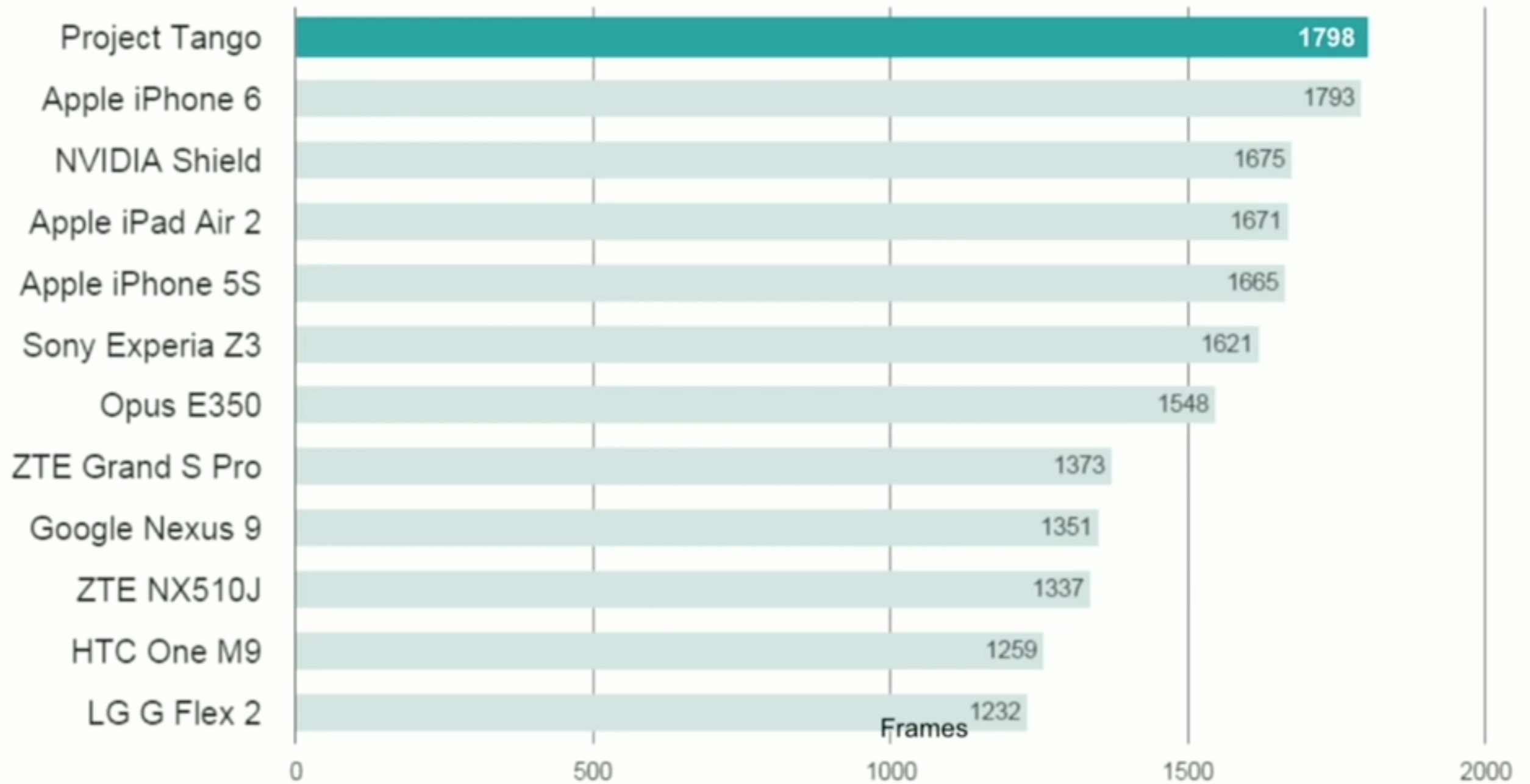
90%

faster GPU

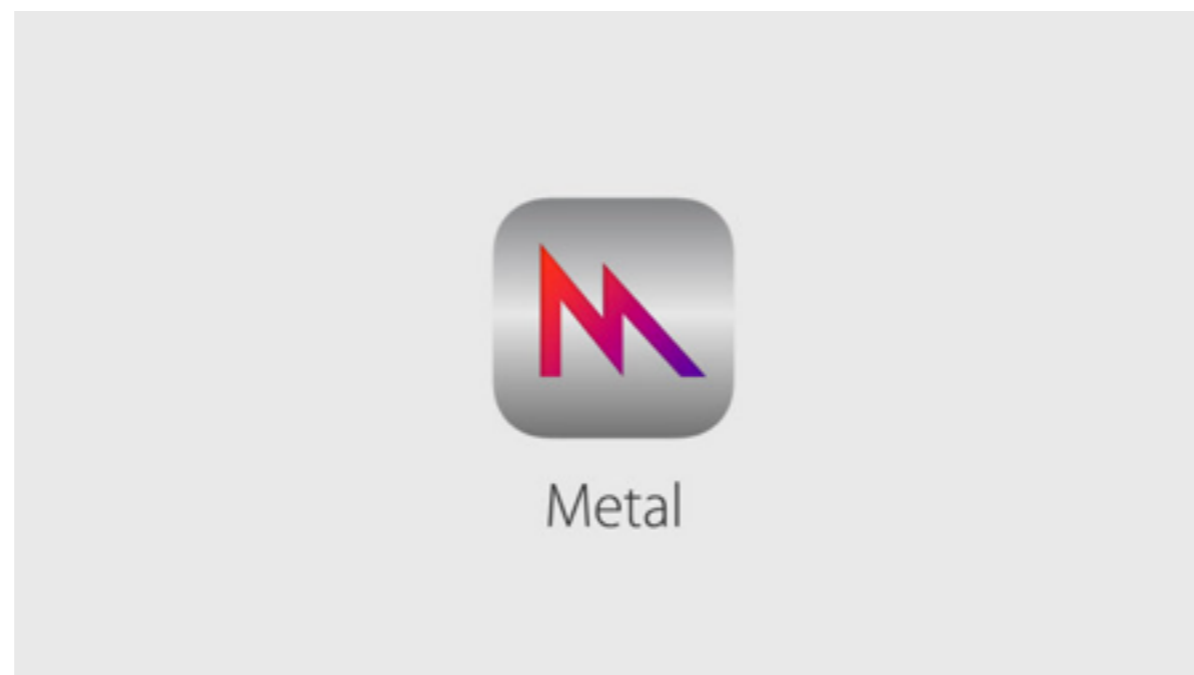
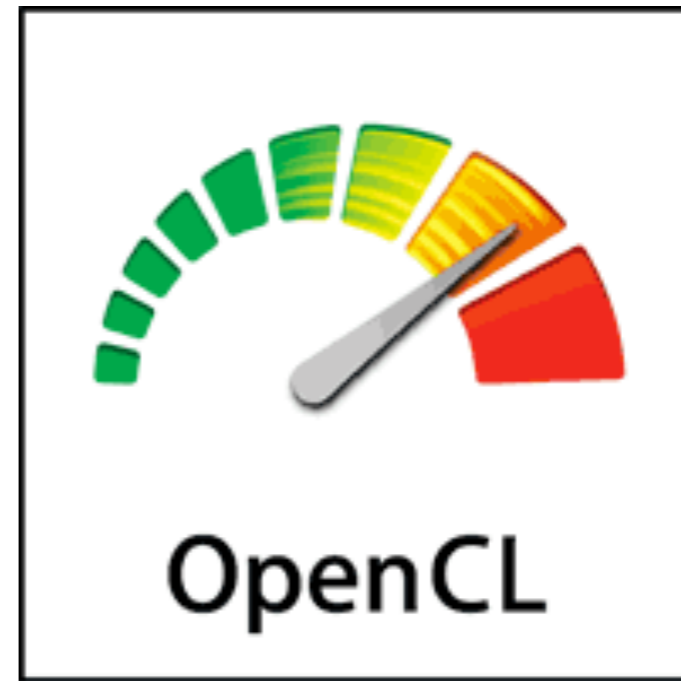


# GFX Bench - Manhattan 3.0

ARM processor - FHD (1920x1080) devices 2015/05/24 [\[link\]](#)



(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.





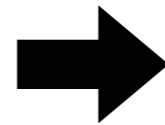
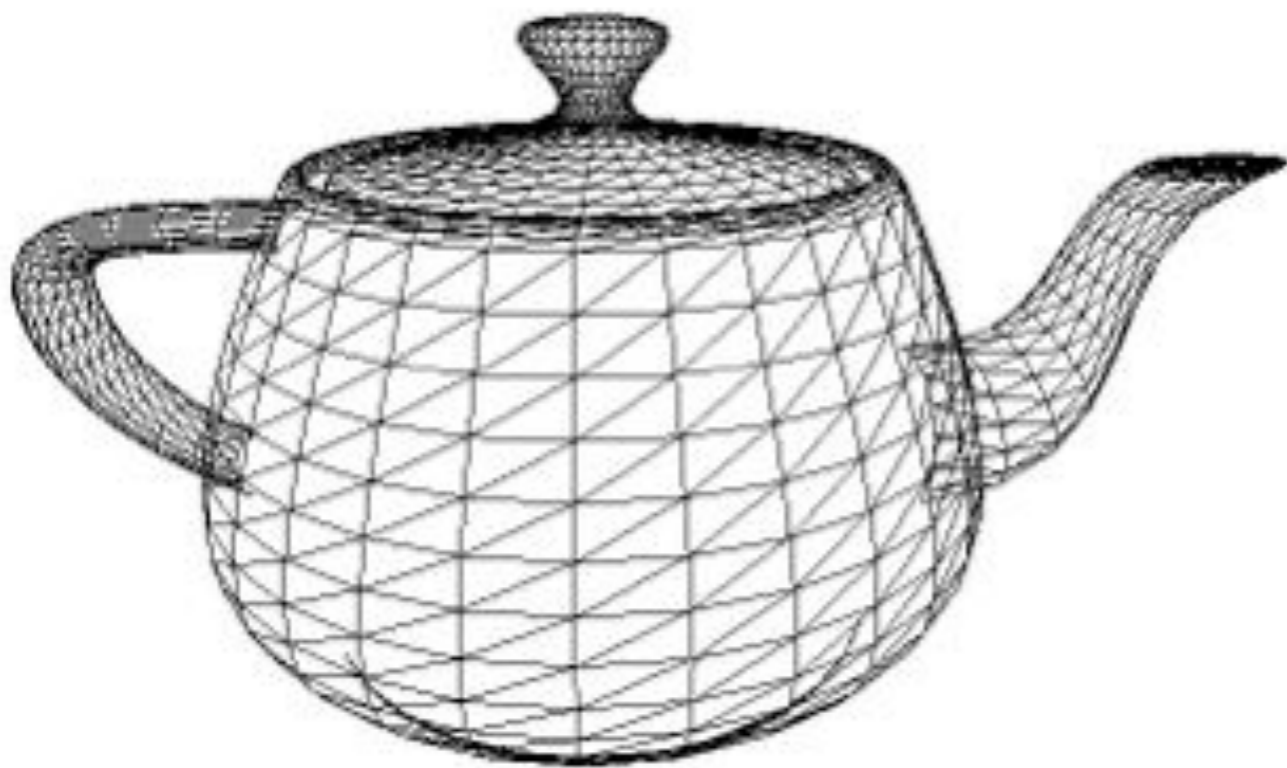
# OpenGL ES

---

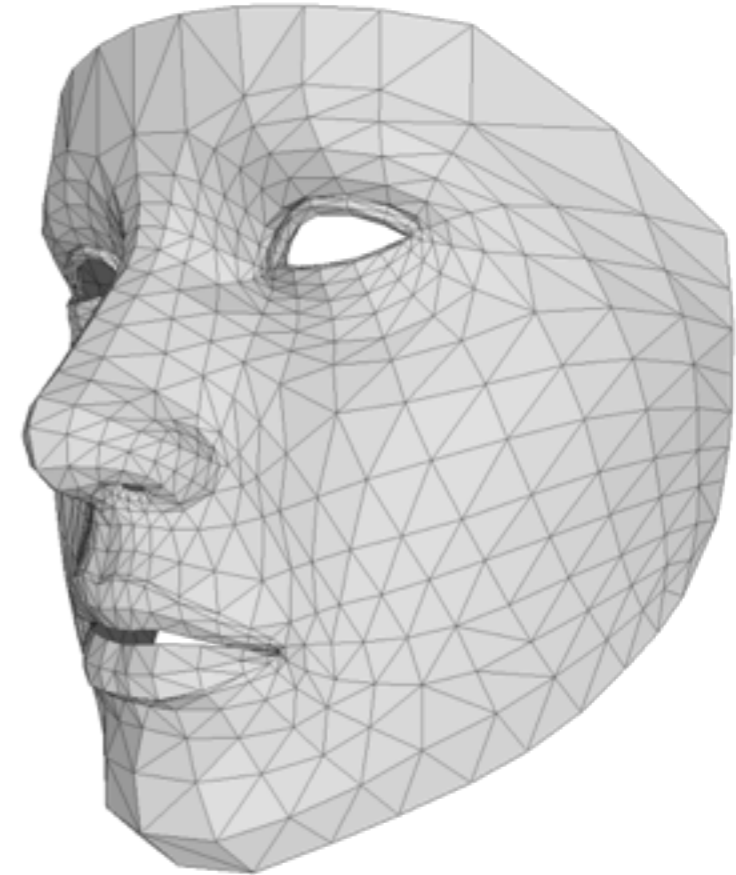
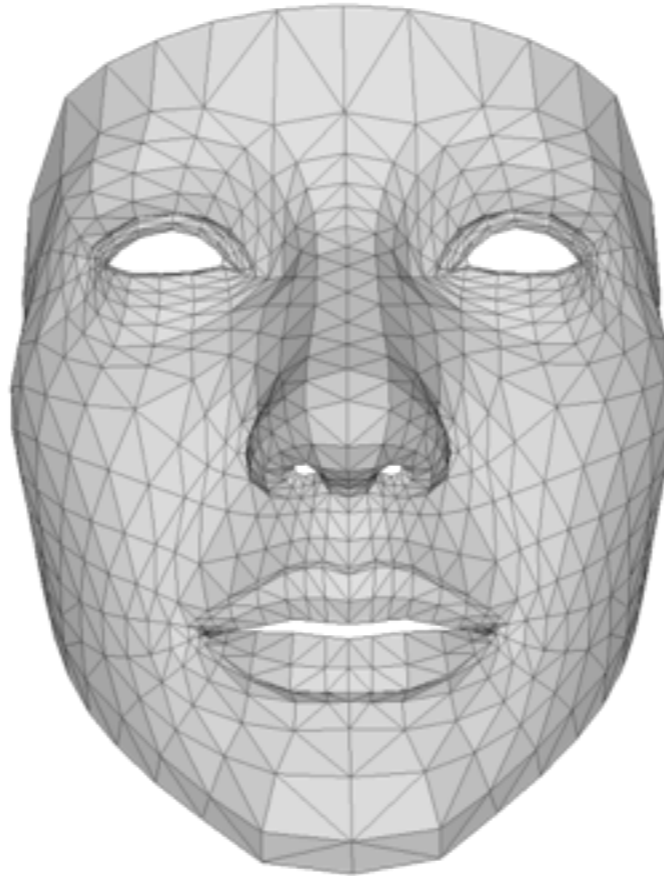
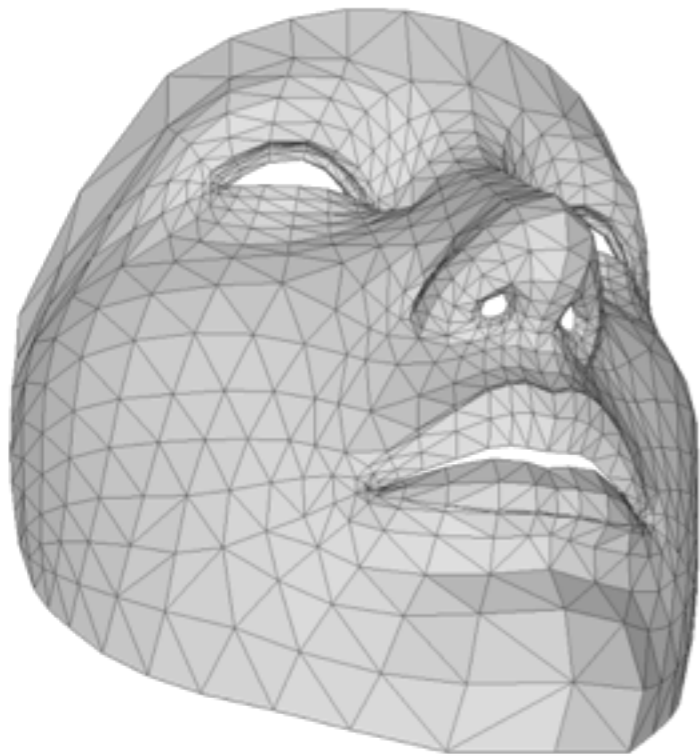
- ES stands for Embedded Systems (ES).
- Subset of OpenGL API
  - Libraries GLUT and GLU not available.
- Designed for embedded systems like smart devices.
- Released in 2003, also maintained by Khronos.



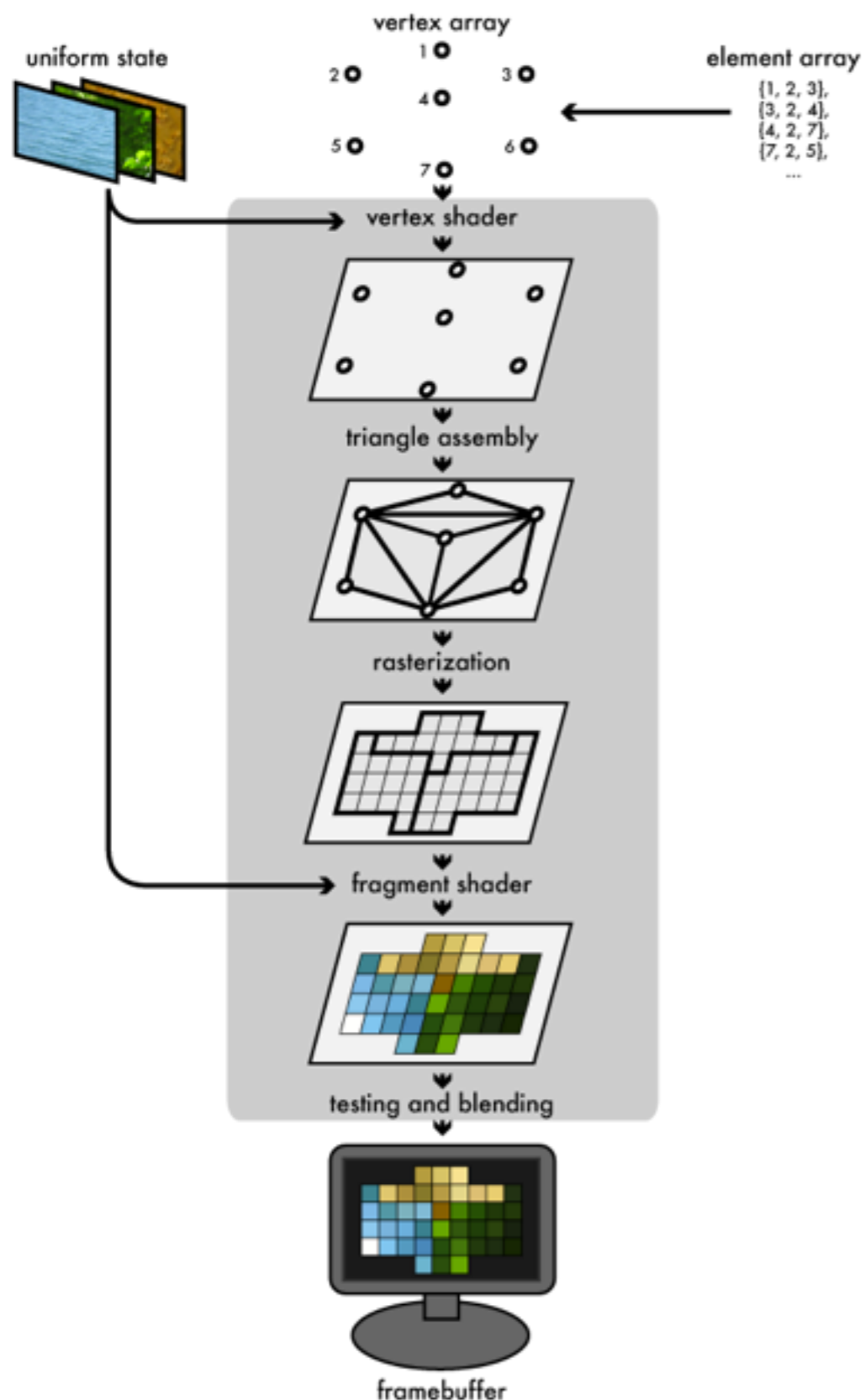
# The “World” is Triangular!!!



# 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.

# 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 being 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.
- More to read:- D. Blythe “Rise of the Graphics Processor” Proceedings of the IEEE 2008.

# Today

---

- Motivation
- GPU
- OpenGL
- GPUImage Library

**GPUImage**

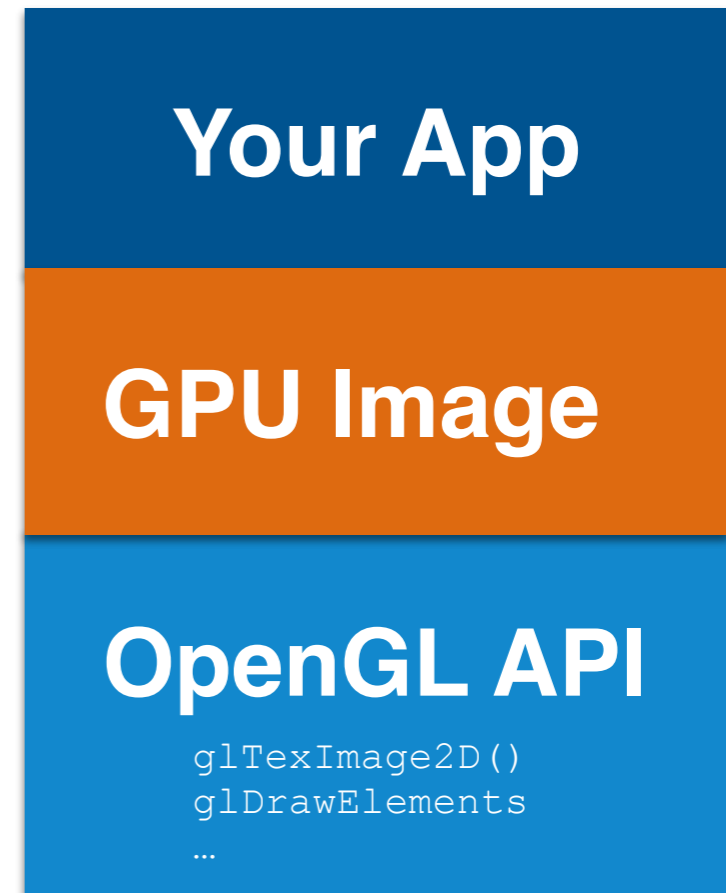
+





# 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 - Example Filters

- **GPUImageShiTomasiCornerDetectionFilter:** Runs the Shi-Tomasi feature detector. It behaves as described above for the Harris detector.
  - *blurRadiusInPixels:* The radius of the underlying Gaussian blur. The default is 2.0.
  - *sensitivity:* An internal scaling factor applied to adjust the dynamic range of the cornerness maps generated in the filter. The default is 1.5.
  - *threshold:* The threshold at which a point is detected as a corner. This can vary significantly based on the size, lighting conditions, and iOS device camera type, so it might take a little experimentation to get right for your cases. Default is 0.2.
- **GPUImageCannyEdgeDetectionFilter:** This uses the full Canny process to highlight one-pixel-wide edges
  - *texelWidth:*
  - *texelHeight:* These parameters affect the visibility of the detected edges
  - *blurRadiusInPixels:* The underlying blur radius for the Gaussian blur. Default is 2.0.
  - *blurTexelSpacingMultiplier:* The underlying blur texel spacing multiplier. Default is 1.0.
  - *upperThreshold:* Any edge with a gradient magnitude above this threshold will pass and show up in the final result. Default is 0.4.
  - *lowerThreshold:* Any edge with a gradient magnitude below this threshold will fail and be removed from the final result. Default is 0.1.
- **GPUImageHoughTransformLineDetector:** Detects lines in the image using a Hough transform into parallel coordinate space. This approach is based entirely on the PC lines process developed by the Graph@FIT research group at the Brno University of Technology and described in their publications: M. Dubská, J. Havel, and A. Herout. Real-Time Detection of Lines using Parallel Coordinates and OpenGL. Proceedings of SCCG 2011, Bratislava, SK, p. 7 (<http://medusa.fit.vutbr.cz/public/data/papers/2011-SCCG-Dubaska-Real-Time-Line-Detection-Using-PC-and-OpenGL.pdf>) and M. Dubská, J. Havel, and A. Herout. PClines — Line detection using

# 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).

```
varying highp vec2 textureCoordinate;

uniform sampler2D inputImageTexture;

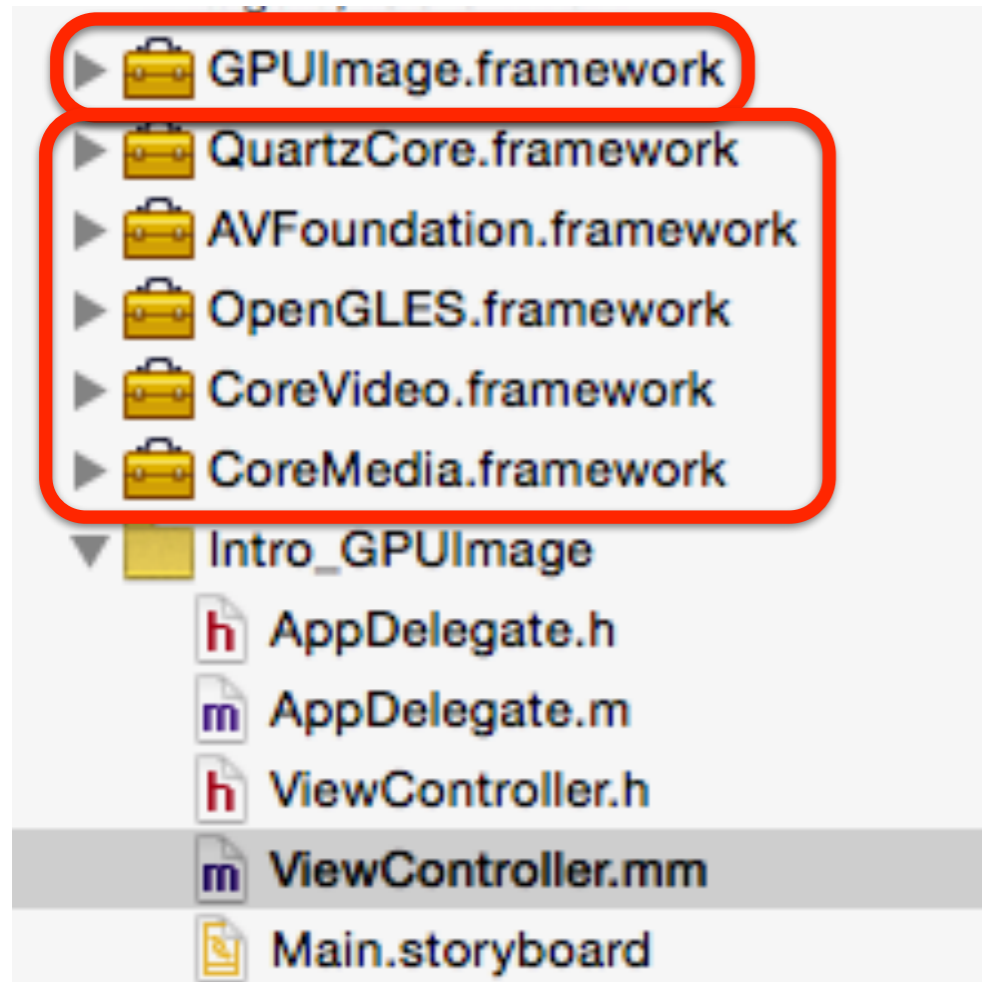
void main()
{
    lowp vec4 textureColor = texture2D(inputImageTexture, textureCoordinate);
    lowp vec4 outputColor;
    outputColor.r = (textureColor.r * 0.393) + (textureColor.g * 0.769) + (textureColor.b * 0.188);
    outputColor.g = (textureColor.r * 0.349) + (textureColor.g * 0.686) + (textureColor.b * 0.169);
    outputColor.b = (textureColor.r * 0.272) + (textureColor.g * 0.534) + (textureColor.b * 0.491);
    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

@implementation ViewController
```

# Playing with GPUImage

---

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

[https://github.com/slucy-cs-cmu-edu/Intro\\_GPUImage](https://github.com/slucy-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/slucy-cs-cmu-edu/Intro\_GPUImage.git
```

# Playing with GPUImage

```
8
9  #import "ViewController.h"
10 #import <GPUImage/GPUImage.h>
11
12 @interface ViewController () {
13     // Setup the view (this time using GPUImageView)
14     GPUImageView *imageView_;
15 }
16
17 @end
18
19 @implementation ViewController
20
21 - (void)viewDidLoad {
22     [super viewDidLoad];
23     // Do any additional setup after loading the view, typically from a nib.
24
25     // Setup GPUImageView (not we are not using UIImageView here).....
26     imageView_ = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self
27
28     // Important: add as a subview
29     [self.view addSubview:imageView_];
30
```

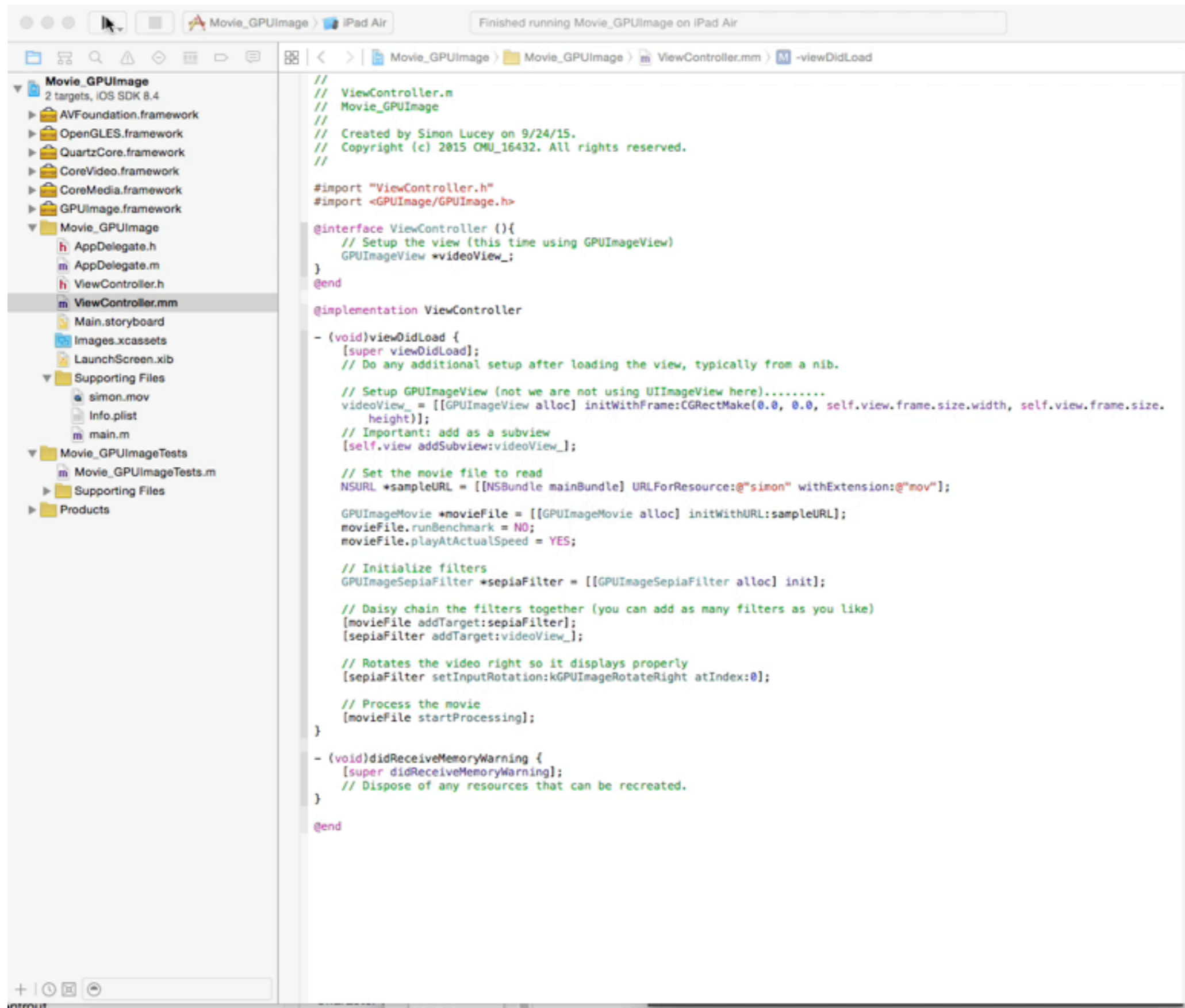


# Playing with GPUImage

```
30
31 // Read in the image (of the famous Lena)
32 UIImage *inputImage = [UIImage imageNamed:@"lena.png"];
33
34 // Initialize filters
35 GPUImagePicture *stillImageSource = [[GPUImagePicture alloc] initWithImage:inputImage];
36 GPUImageSepiaFilter *stillImageFilter = [[GPUImageSepiaFilter alloc] init];
37
38 // Daisy chain the filters together (you can add as many filters as you like)
39 [stillImageSource addTarget:stillImageFilter];
40 [stillImageFilter addTarget:imageView_];
41
42 // Process the image
43 [stillImageSource processImage];
44
45 }
46
47 - (void)didReceiveMemoryWarning {
48     [super didReceiveMemoryWarning];
49     // Dispose of any resources that can be recreated.
50 }
51
52 @end
```



# GPUImage for Movies



```
//
// ViewController.m
// Movie_GPUImage
//
// Created by Simon Lucey on 9/24/15.
// Copyright (c) 2015 CMU_16432. All rights reserved.
//

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

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

@implementation ViewController

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

    // Setup GPUImageView (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:videoView_];

    // Rotates the video right so it displays properly
    [sepiaFilter setInputRotation:kGPUImageRotateRight atIndex: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/slucy-cs-cmu-edu/Movie\\_GPUImage](https://github.com/slucy-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/slucy-cs-cmu-edu/Movie\_GPUImage
```

# GPUImage2

---

- GPUImage 2 was released in 2016...
- Second generation of GPUImage framework for SWIFT.



- Check out on <https://github.com/BradLarson/GPUImage2>



Metal

**What About?**

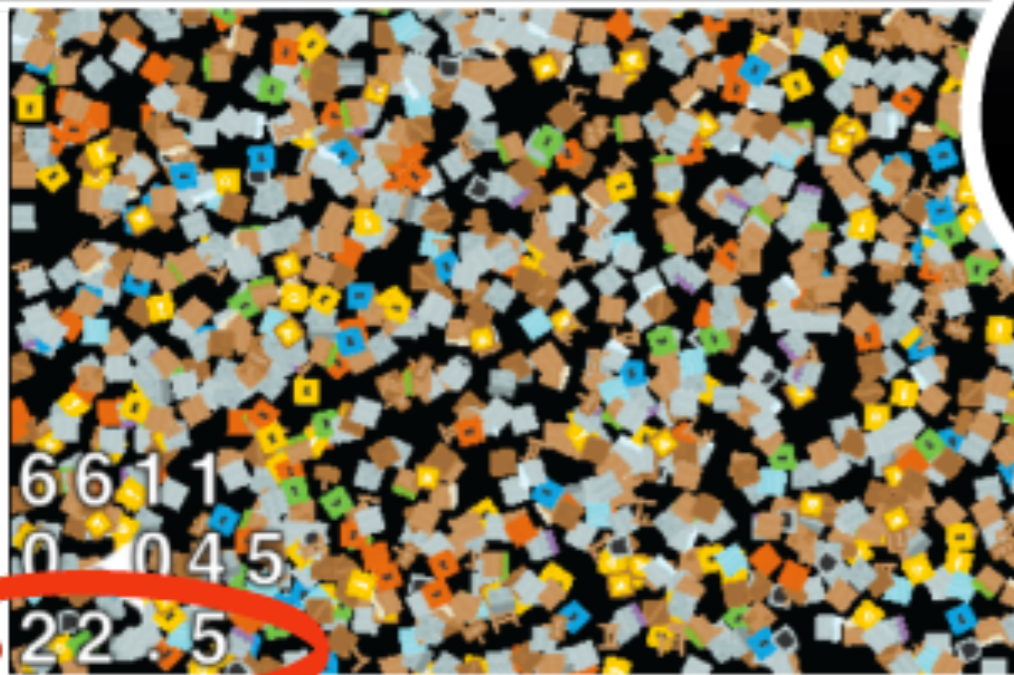
**GPUImage**

+



**MetalGL**

OpenGL



MetalGL



**Supposedly 3x Faster!!**



# OpenGL

Your App

OpenGL API  
glTexImage2D()  
glDrawElements  
...

OpenGL Engine

# MetalGL

Your App

OpenGL API  
glTexImage2D()  
glDrawElements  
...

MetalGL

Metal



# More Examples to Play With...

---

- Download the complete GPUImage library from,
  - <https://github.com/BradLarson/GPUImage>
- In there you will find a fair amount of example code,
  - SimpleVideoFileFilter
  - FilterShowCase
  - MultiViewFilterExample
  - BenchmarkSuite
  - RawDataTest