Learning Core Image

Guanshan Liu

@guanshanliu

Who Am I

— Working on TTPod for iOS at Alibaba Inc.

— I love design and make apps

— Twitter: @guanshanliu

— Email: guanshan.liu@gmail.com

What is Core Image

Core Image is a powerful image processing framework that allow you to easily add awesome effects to still images and live video. It is built on top of OpenGL, and uses shaders to do image processing.

What is Core Image

- It uses GPU to process image data by default
- You can choose to use CPU by setting key kCIContextUseSoftwareRenderer to YES
- Introduced in OS X 10.4, iOS 5
- You can create custom image kernels in iOS 8

Overview

- **CIContext** It's where all image processing happens. Similar to CoreGraphics or OpenGL context.
- **CIImage** An image abstraction.
- **CIFilter** A filter takes one or more images as input, produces a CIImage object as output based on key-value pairs of input parameters.

CIContext

— **GPU-based** Faster, but is limited to the hardware texture size of the GPU. Also it cannot continue to run if your app is in background.

— **CPU-based** Slower, but can handle any size and can continue to run in the background.

CIImage

It can be created in many ways:

— Raw pixel data: NSData, CVPixelBufferRef, etc.

— Image data classes: UIImage, CGImageRef, etc.

— OpenGL textures

CIFilter

Builtin Filters

— In Objective-C

```
[CIFilter filterNamesInCategory:kCICategoryBuiltIn]
```

— In Swift

```
CIFilter.filterNamesInCategory(kCICategoryBuiltIn)
```

CIFilter

Builtin Filters

— 169 filters on OS X 10.10

— 127 filters on iOS 8

CIFilter

Each filter has a dictionary containing filter's name, the kinds of input parameters the filters takes, the default and acceptable values, and its category.

CIFilter

In Objective-C

```
NSArray *filters = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];
for (NSString *filterName in filters) {
    CIFilter *filter = [CIFilter filterWithName:filterName];
    NSLog(@"%@", [filter attributes]);
}
```

In Swift

```
let filterNames = CIFilter.filterNamesInCategory(kCICategoryBuiltIn) as [String]
for filterName in filterNames {
    let filter = CIFilter(name: filterName)
    println(filter.attributes())
}
```

Example - CISepiaTone

```
[CIAttributeFilterDisplayName: Sepia Tone, CIAttributeFilterName: CISepiaTone,
inputImage: {
    CIAttributeClass = CIImage;
    CIAttributeType = CIAttributeTypeImage;
}, CIAttributeFilterCategories: (
    CICategoryColorEffect,
    CICategoryVideo,
    CICategoryInterlaced,
    CICategoryNonSquarePixels,
    CICategoryStillImage,
    CICategoryBuiltIn,
    CICategoryXMPSerializable
), inputIntensity: {
    CIAttributeClass = NSNumber;
    CIAttributeDefault = 1;
    CIAttributeIdentity = 0;
    CIAttributeMax = 1;
    CIAttributeMin = 0;
    CIAttributeSliderMax = 1;
    CIAttributeSliderMin = 0;
    CIAttributeType = CIAttributeTypeScalar;
}]
```

Example - CISepiaTone

```
// Create a CIContext
let context = CIContext()

// Get CIImage from UIImage
let image = UIImage(named: "Image")!
let input = CIImage(image: image)

// Create a fitler
let filter = CIFilter(name: "CISepiaTone")
filter.setValue(input, forKey: kCIInputImageKey)
filter.setValue(1.0, forKey: kCIInputIntensityKey)

// Get output CIImage from the filter
let output = filter.outputImage
let extent = output.extent()

// Get UIImage from CIContext
let imageRef = context.createCGImage(output, fromRect: extent)
let outputImage = UIImage(CGImage: imageRef, scale: image.scale, orientation: image.imageOrientation)!
```

Demo

Sepia Tone Filter

Example - Filter Chain

Filters can be chained together. It's like a pineline. Just put the output image of a filter as input image of the next filter.

Auto-Enhancement

CIImage has a method *autoAdjustmentFilters* that returns an array of filters including red eye reduction, flesh tone, etc.

You can use the array to apply a filter chain to an image.

Example - Filter Chain

```
func autoAdjustment(image: CIImage) -> CIImage {
    let filters = image.autoAdjustmentFilters() as [CIFilter]
    let output = filters.reduce(image, combine: { (input, filter) -> CIImage in
        filter.setValue(input, forKey: kCIInputImageKey)
        return filter.outputImage
    })
    return output
}
```

Demo

Filter Chain

Two More Demos

Example - Custom Image Kernel

1. Subclass CIFilter
2. let kernel = CIKernel(string: kernelSource)
3. override var outputImage: CIImage { get } method using kernel.applyWithExtent

Demo

Custom Image Kernel

Example - Live Video Filter

```
glContext = EAGLContext(API: .OpenGLES3)
glView.context = glContext
coreImageContext = CIContext(EAGLContext: glContext)

let videoOutput = AVCaptureVideoDataOutput()
videoOutput.videoSettings = [kCVPixelBufferPixelFormatTypeKey: kCVPixelFormatType_32BGRA]
videoOutput.setSampleBufferDelegate(self, queue: sessionQueue)
session.addOutput(videoOutput)

func captureOutput(captureOutput: AVCaptureOutput!,
            didOutputSampleBuffer sampleBuffer: CMSampleBuffer!,
            fromConnection connection: AVCaptureConnection!) {
    let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
    var image = CIImage(CVPixelBuffer: pixelBuffer)
    image = sepiaTone(image)
    coreImageContext.drawImage(image, inRect: bounds, fromRect: bounds)
    glContext.presentRenderbuffer(Int(GL_RENDERBUFFER))
}
```

Demo

Live Video Filter

Core Image with Functional Programming

```
extension CIFilter {
    convenience init(name: String, parameters: Parameters) {
        self.init(name: name)
        setDefaults()
        for (key, value) in parameters {
            setValue(value, forKey: key)
        }
    }

    var outputImage: CIImage {
        return self.valueForKey(kCIOutputImageKey) as CIImage
    }
}
```

More in Functional Programming in Swift

Core Image with Functional Programming

```
typealias Filter = CIImage -> CIImage

func blur(radius: Double) -> Filter {
    return { image in
        let parameters: Parameters = [
            kCIInputRadiusKey: radius,
            kCIInputImageKey: image
        ]
        let filter = CIFilter(name: "CIGaussianBlur", parameters: parameters)
        return filter.outputImage
    }
}
```

More in Functional Programming in Swift

Core Image with Functional Programming

```
func sepiaTone(intensity: Double) -> Filter {
    return { image in
        let parameters: Parameters = [
            kCIInputImageKey: image,
            kCIInputIntensityKey: intensity
        ]
        let filter = CIFilter(name: "CISepiaTone", withInputParameters: parameters)
        return filter.outputImage
    }
}
```

More in Functional Programming in Swift

Core Image with Functional Programming

```
infix operator · { associativity left }

public func · <T, U, V> (g: U -> V, f: T -> U) -> T -> V {
    return { x in g(f(x)) }
}

let myFilter = sepiaTone(0.8) · blur(5)
```

More in Functional Programming in Swift

Resources

Core Image

— WWDC sessions

1. 2011: 129, 422
2. 2012: 510, 511
3. 2013: 509
4. 2014: 514, 515

— Beginning Core Image in iOS 6

Custom Image Kernel

— GPUImage by Brad Larson

Slides and sample codes of this talk

— Available on GitHub

Thank you!