

# 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](mailto:guanshan.liu@gmail.com)

# What is Core Image

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

- It uses GPU to process image or CPU  
(`kCIContextUseSoftwareRenderer: 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.
- **CImage** An image abstraction.
- **CIFilter** A filter takes one or more images as input, produces a CImage object as output based on key-value pairs of input parameters.

# CIContext

In iOS 7, the CPU renderer was used when

- GPU texture limits were exceeded
- The application needed to render briefly in the background
- The application wanted to render in a low priority thread

**Copied from Session 514, WWDC 2014**

# CIContext

Full support for images greater than the GPU limits in iOS 8

- Input images can be > 4K
- Output renders can be > 4K

~~GPU texture limits were exceeded~~

- No longer a limit in iOS 8 Core Image

**Copied from Session 514, WWDC 2014**

# CIContext

In iOS 8

Renders within a short time of switching to background

- Use faster GPU renderer
- Serviced with a lower priority
- Will not disturb foreground GPU usage

**Copied from Session 514, WWDC 2014**

# CIContext

- ~~The application needed to render briefly in the background~~
- ~~The application wanted to render in a low priority thread~~
  - Can now request `kCIContextPriorityRequestLow` in iOS 8 Core Image

**Copied from Session 514, WWDC 2014**



# 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: (
    CIColorEffect,
    CIColorVideo,
    CIColorInterlaced,
    CIColorNonSquarePixels,
    CIColorStillImage,
    CIColorBuiltIn,
    CIColorXMPSerializable
), 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 CUIImage from UIImage
let image = UIImage(named: "Image")!
let input = CUIImage(image: image)

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

// Get output CUIImage 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 pipeline. Just put the output image of a filter as input image of the next filter.

# Auto-Enhancement

UIImage 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

```
typealias Filter = CIImage -> CIImage

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