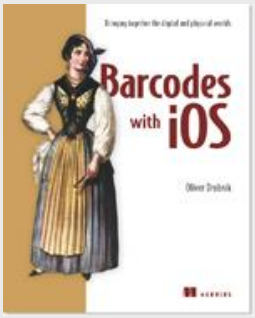


Barcodes with iOS: Introducing Core Image

By Oliver Drobnik

The Core Graphics framework is written in pure C, meaning that it's impossible to use CGImage instances directly with UIKit. Apple created UIImage as an Objective-C wrapper class around CGImage to bridge the gap. We explore Core Image in this article.

This article is excerpted from Barcodes with iOS. Save 39% on Barcodes with iOS *with code 15dzamia at manning.com.*

At their core, images consist of colored pixels. Depending on the color space, they might have different numbers of "channels," most commonly red, green, blue, and alpha. Usually one byte is used per channel. The size in bytes for such a bitmapped image is calculated as `width` x `height` x `bytes_per_channel` x `number_of_channels`.

Core Graphics—a.k.a. Quartz—represents such bitmapped images as `CGImage` instances. The Core Graphics framework is written in pure C, meaning that it's impossible to use `CGImage` instances directly with UIKit. Apple created `UIImage` as an Objective-C wrapper class around `CGImage` to bridge the gap. `UIImage` instances usually carry a `CGImage` in their belly that contains the actual image data.

Figure 1 Finished QR Code Builder app with printed output

When manipulating images in UIKit or Quartz, you never get the benefit of the GPU. That's why Apple created *Core Image* as a framework for manipulating images in real time with the full benefit of hardware acceleration by the graphics processor.

In Core Image you don't deal with individual pixels but rather with *manipulation steps*. Each such step, represented by a `CIFilter`, is a recipe for manipulating images represented by `CIImage` instances. If you chain multiple manipulation steps, Core Image compiles those down to a single GPU program, called a *shader*. When you request the final output of such a filter chain, the initial input is loaded on the GPU, the compiled shader is run, and you receive the resulting output. Figure 5.4 shows a `CGImage` being turned into a `CIImage`, the chained filters doing their work on that, and a new `CGImage` being created via a `CIContext`.

`CIImage` instances can be created from a wide variety of sources. Static images will usually come from `CGImage` instances. You can also pass `CVPixelBuffer` instances if you want to handle live video coming from an `AVCaptureDevice`.

Most Core Image filters have an `inputImage` parameter for supplying the source image. One category of Core Image filters—the so-called *generators*—don't, because they themselves are able to generate images. Generators can serve as input for other filters,

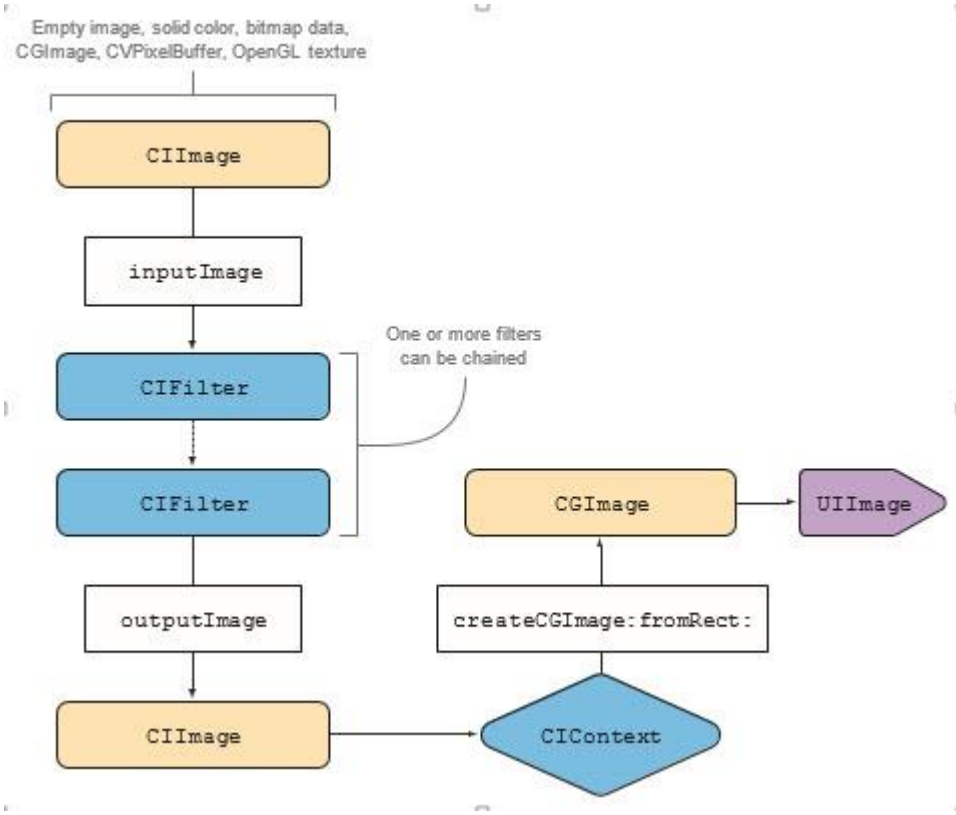
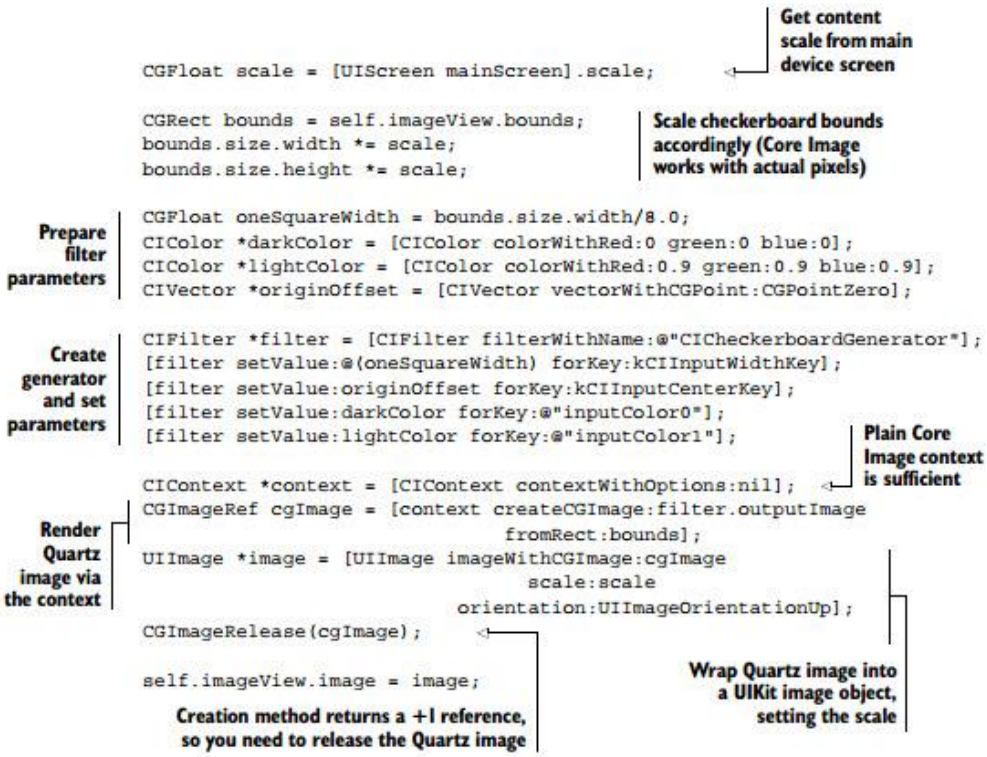


Figure 2 Core Image filter chain

or you can simply poll their output. For example, you can use `CIConstantColorGenerator` for creating images consisting of a single solid color or `CICheckerboardGenerator` for creating an image with a checkerboard pattern.

Let's try out a simple Core Image generator by creating an 8 x 8 checkerboard suitable for display with a `UIImageView`. Note the use of `CIColor` for specifying colors and `CIVector`

for specifying an x-y offset. Those are the typical immutable parameter objects used by Core Image. Values are specified as `NSNumber` objects:




```
CGFloat scale = [UIScreen mainScreen].scale;

CGRect bounds = self.imageView.bounds;
bounds.size.width *= scale;
bounds.size.height *= scale;

CGFloat oneSquareWidth = bounds.size.width/8.0;
CIColor *darkColor = [CIColor colorWithRed:0 green:0 blue:0];
CIColor *lightColor = [CIColor colorWithRed:0.9 green:0.9 blue:0.9];
CIVector *originOffset = [CIVector vectorWithCGPoint:CGPointZero];

CIFilter *filter = [CIFilter filterWithName:@"CICheckerboardGenerator"];
[filter setValue:@(oneSquareWidth) forKey:kCIInputWidthKey];
[filter setValue:originOffset forKey:kCIInputCenterKey];
[filter setValue:darkColor forKey:@"inputColor0"];
[filter setValue:lightColor forKey:@"inputColor1"];

CIContext *context = [CIContext contextWithOptions:nil];
CGImageRef cgImage = [context createCGImage:filter.outputImage
                                   fromRect:bounds];
UIImage *image = [UIImage imageWithCGImage:cgImage
                                     scale:scale
                               orientation:UIImageOrientationUp];
CGImageRelease(cgImage);

self.imageView.image = image;
```

Generators output `CIImage` instances via their `outputImage` method. To use one with UIKit, you need to render it into a `CGImage` by means of a `CIContext`. As you can see in the preceding example, Core Image doesn't have any knowledge of the device's content scale, which would be 2 for Retina displays. Because of this, you need to double the size of the generated image and then specify this scale in the method that makes a `UIImage` out of the `CGImage`.

This should give you enough information about the general workings of Core Image generators.