

.....  
**BEGINNING  
METAL**  
.....



**HANDS-ON CHALLENGES**

## Beginning Metal

Caroline Begbie

Copyright ©2016 Razeware LLC.

## Notice of Rights

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written permission of the copyright owner.

## Notice of Liability

This challenge and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use of other dealing in the software.

## Trademarks

All trademarks and registered trademarks appearing in this book are the property of their own respective owners.

# Challenge #2: Renderer

By Caroline Begbie

Currently the rendering code is all in `ViewController`. Your challenge is to create a new `Renderer` class which will take care of all the rendering.

As you progress through this course, you'll often refactor the code that we write in the demo. This gives you an opportunity to review and experiment with the code.

Abstracting code is important as the code base gets more complex. When you come to create a game, you'll be glad of the rendering details being hidden away in a separate class. You'll be able to add models to the scene without thinking about pipelines and command buffers.

Firstly, create a new Swift file called **Renderer.swift**.

Import `MetalKit` at the top of the file:

```
import MetalKit
```

Create a class with `device` and `commandQueue` just as you had in `ViewController`:

```
class Renderer: NSObject {  
    let device: MTLDevice  
    let commandQueue: MTLCommandQueue  
}
```

`Renderer` is a `NSObject` because you'll be conforming to `MTKViewDelegate`, and this requires that all conforming objects have the base class `NSObject`.

`device` represents the GPU and `commandQueue` organizes the command buffers on the GPU.

Create an initializer which receives device.

```
init(device: MTLDevice) {  
    self.device = device  
    commandQueue = device.makeCommandQueue()  
    super.init()  
}
```

Move the ViewController's MTKViewDelegate extension into Renderer, and change the extension class to Renderer from ViewController:

```
extension Renderer: MTKViewDelegate {  
    func mtkView(_ view: MTKView, drawableSizeWillChange size: CGSize) { }  
  
    func draw(in view: MTKView) {  
        guard let drawable = view.currentDrawable,  
              let descriptor = view.currentRenderPassDescriptor else { return }  
  
        let commandBuffer = commandQueue.makeCommandBuffer()  
        let commandEncoder =  
            commandBuffer.makeRenderCommandEncoder(descriptor: descriptor)  
        commandEncoder.endEncoding()  
        commandBuffer.present(drawable)  
        commandBuffer.commit()  
    }  
}
```

In ViewController, add a property for Renderer:

```
var renderer: Renderer?
```

Remove the device and commandQueue properties.

Change viewDidLoad() to create renderer with the device, and set renderer to be the delegate.

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    metalView.device = MTLCreateSystemDefaultDevice()  
    guard let device = metalView.device else {  
        fatalError("Device not created. Run on a physical device.")  
    }  
    metalView.clearColor = Colors.wenderlichGreen  
  
    renderer = Renderer(device: device)  
    metalView.delegate = renderer  
}
```

Build and run, and you should get a Wenderlich colored screen just as you did before you added Renderer.

