```
Render {
                                RenderPipeline(vertexShader: vertexShader, fragmentShader: fragmentShader) {
                                  Draw { encoder in
                                     encoder.setVertexBytes(<...>)
                                     encoder.drawPrimitives(type: .triangle, vertexStart: 0, vertexCount: 3)
                                   .parameter("color", SIMD4<Float>([1, 0, 0, 1]))
                                      let commandQueue = device.makeCommandQueue()!
                                      let commandBuffer = commandQueue.makeCommandBuffer()!
   (Host)
                                      let renderPassDescriptor = MTLRenderPassDescriptor()
                                      renderPassDescriptor.colorAttachments[0].texture = texture
                                      renderPassDescriptor.colorAttachments[0].loadAction = .clear
 Render (*)
                                      renderPassDescriptor.colorAttachments[0].clearColor = MTLClearColor(red: 0, green: 0, blue: 0, alpha: 0)
                                     renderPassDescriptor.colorAttachments[0].storeAction = .store
                                      let renderEncoder = commandBuffer.makeRenderCommandEncoder(descriptor: renderPassDescriptor)!
RenderPipeline
                                     let pipelineDescriptor = MTLRenderPipelineDescriptor()
                                     pipelineDescriptor.vertexFunction = vertexFunction
                                     pipelineDescriptor.fragmentFunction = fragmentFunction
                                      pipelineDescriptor.vertexDescriptor = vertexDescriptor
                                     pipelineDescriptor.colorAttachments[0].pixelFormat = pixelFormat
                                     let (pipelineState, reflection) = try device.makeRenderPipelineState(descriptor: pipelineDescriptor,
                                     options: .bindingInfo)
                  VertexShader
                                      let library = try device.makeLibrary(source: source, options: nil)
                                      let vertexFunction = library.makeFunction(name: "vertex_main")!
                                      let fragmentFunction = library.makeFunction(name: "fragment main")!
                 FragmentShader
                                      let colorIndex = reflection!.fragmentBindings.first { $0.name == "color" }!.index
                                      renderEncoder.setFragmentBytes(&color, length: MemoryLayout<SIMD4<Float>>.stride, index: colorIndex)
                 User Parameters
   Draw
                                   ▶ // Currently same for vanilla Metal vs Ultraviolence – uv can improve in future
                   Draw Calls
```