Apple Metal

introduction

George Ostrobrod

1.

1. Рисуем кружочки

What is Metal

What is Metal

Drawing & Rendering Frameworks

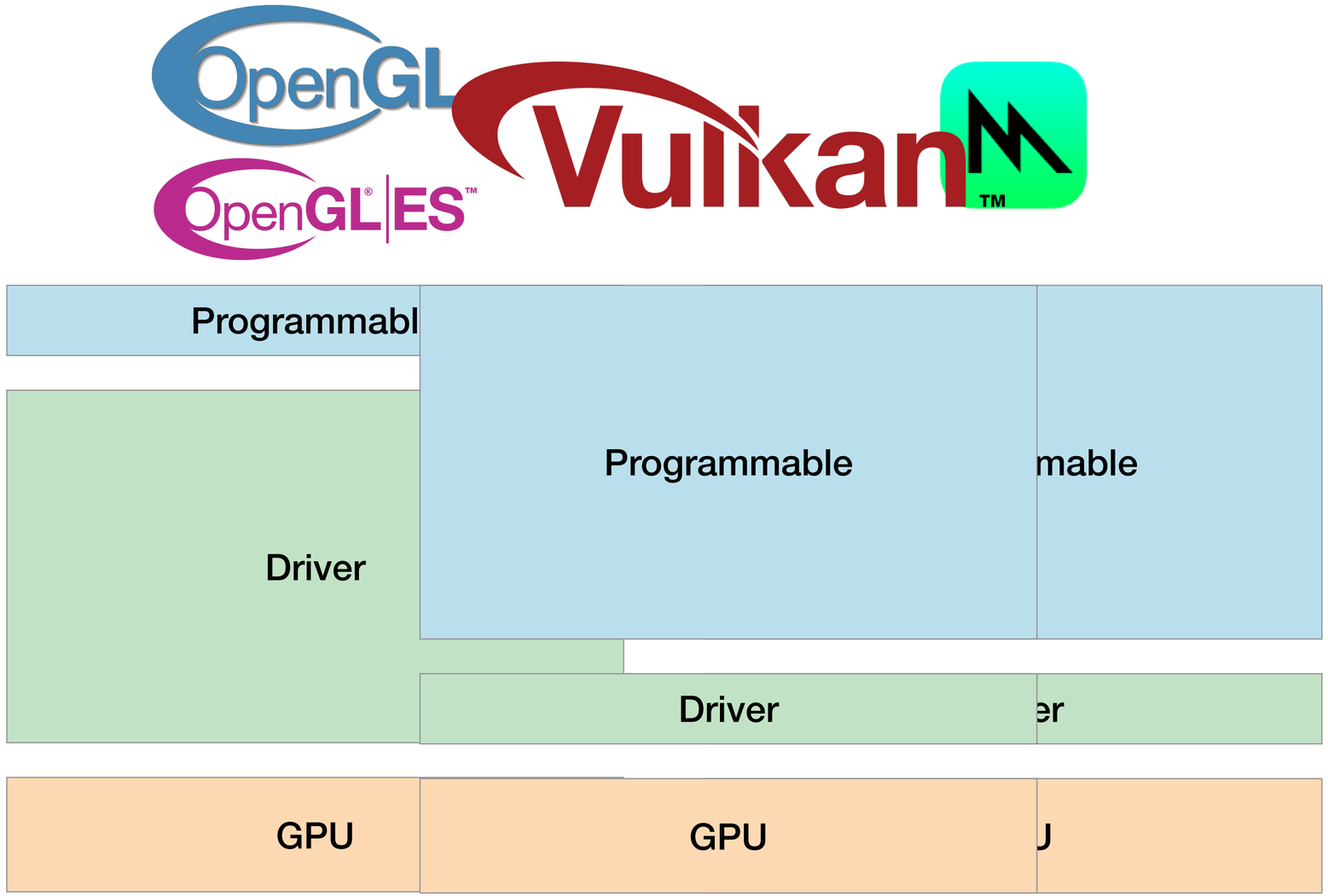
- CoreGraphics

  + vImage

- CoreImage (+CIKL)
- SpriteKit
- SceneKit

What is Metal

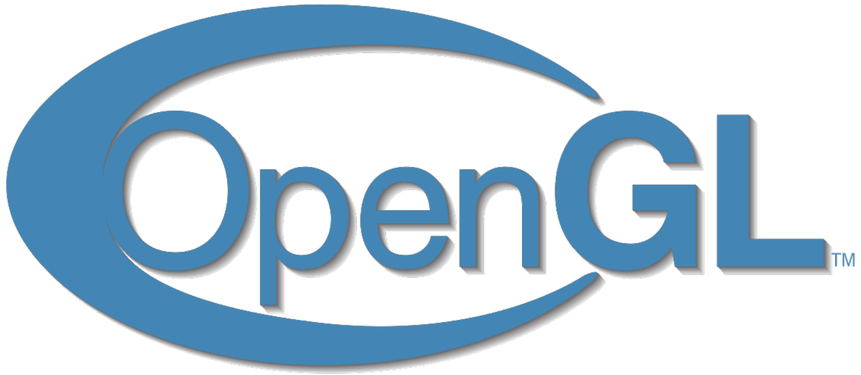
Metal vs OpenGL vs Vulkan



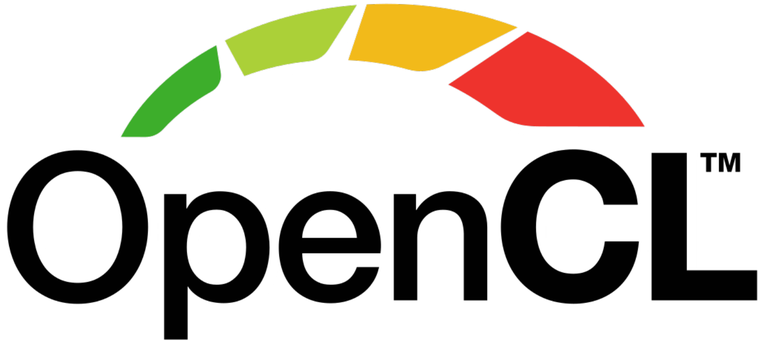
What is Metal

Metal vs OpenGL + OpenCL

≈



+



What is Metal

Typical Tasks

- Rendering (game engines, visualisation, etc)
- Image processing
- Parallel computing

MetalKit

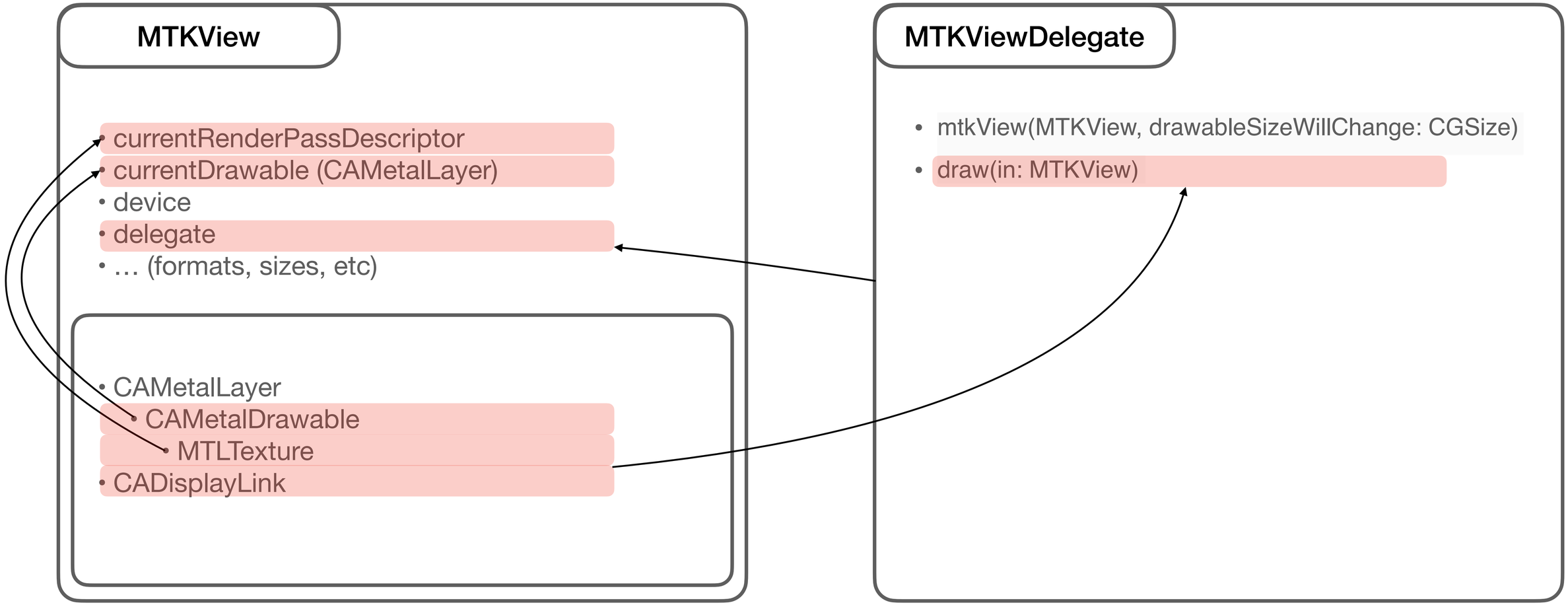
MetalKit

Tasks

- `MTKView + MTKViewDelegate`
- `MTKTextureLoader`
- `MTKMesh`

MetalKit

View

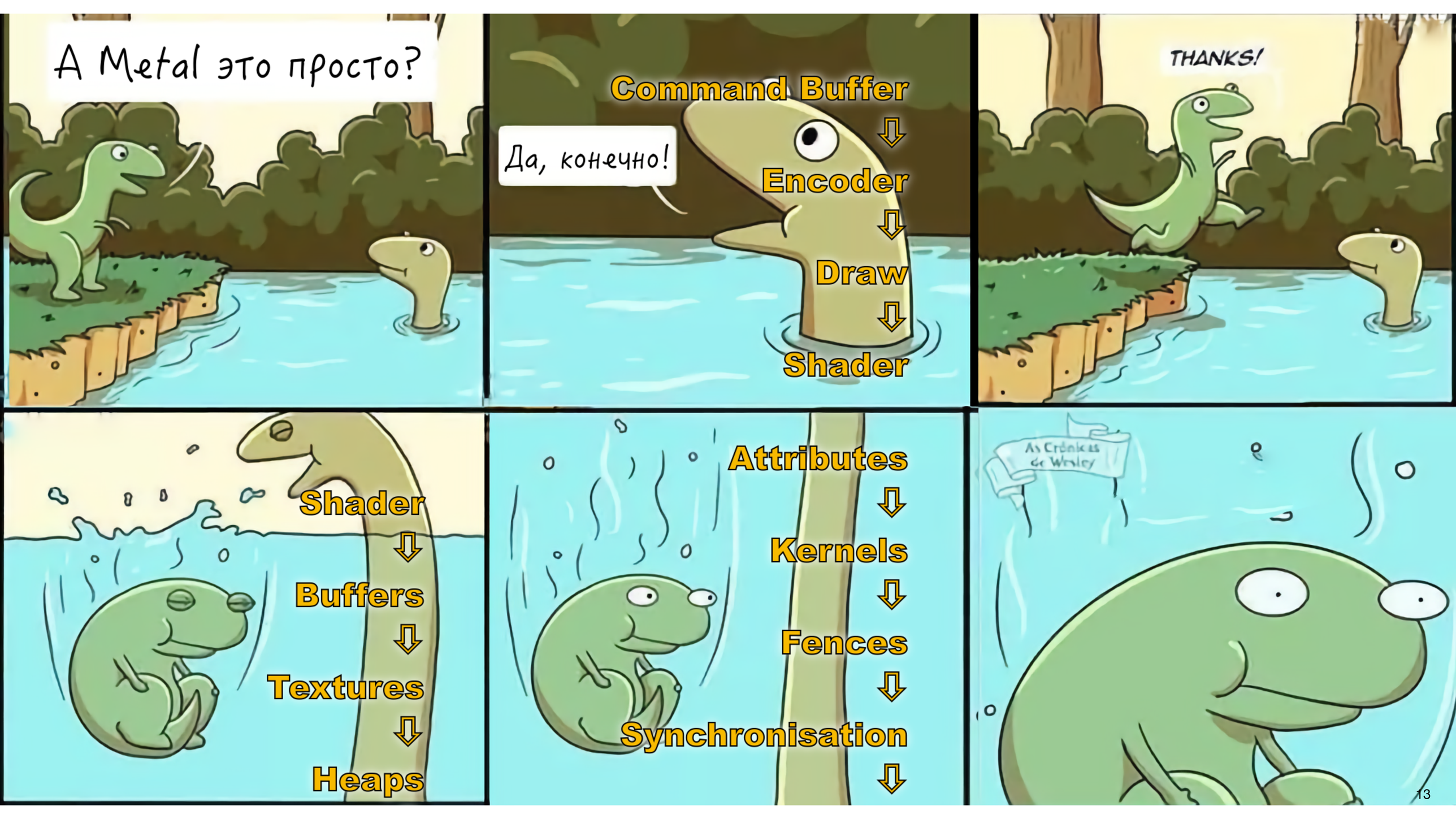


MetalKit

Texture loader

```
func loadTexture(device: MTLDevice, image: UIImage) -> MTLTexture? {
   let textureLoader = MTKTextureLoader(device: device)
   let textureLoaderOptions = [
    MTKTextureLoader.Option.textureUsage: NSNumber(value: MTLTextureUsage.shaderRead.rawValue),
    MTKTextureLoader.Option.textureStorageMode: NSNumber(value: MTLStorageMode.`private`.rawValue),
    MTKTextureLoader.Option.SRGB: false
   ]

   var texture: MTLTexture? = nil
   do {
      texture = try textureLoader.newTexture(cgImage: image.cgImage!, options: textureLoaderOptions)
   } catch {
      print("Unable to load texture. Error info: \(error)")
   }

   return texture
}
```

Metal Architecture


A Metal это просто?
Да, конечно!
Command Buffer
⇩
Encoder
⇩
Draw
⇩
Shader
THANKS!
Shader
⇩
Buffers
⇩
Textures
⇩
Heaps
Attributes
⇩
Kernels
⇩
Fences
⇩
Synchronisation
⇩

Metal Architecture

Queue

Command Buffer

Encoder

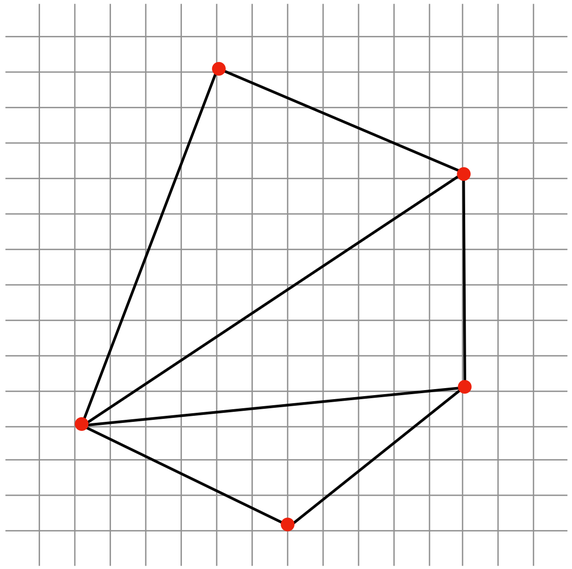
Metal Architecture

Draw calls

Shaders

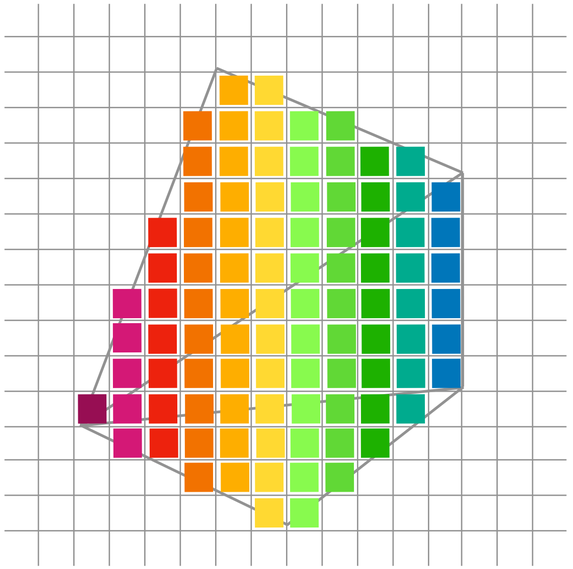
Shaders

Vertex



Rasterisation

Fragment



Metal Shading Language

Shaders

MSL: C++ Restrictions

- lambda expressions
- recursive function calls
- `dynamic_cast` operator
- type identification
- `new` and `delete` operators
- `noexcept` operator
- `goto` statement
- `register`, `thread_local` storage attributes
- derived classes
- exception handling

Shaders

Vertex

```
vertex ColorInOut vertexShader(
     unsigned int vid [[vertex_id]])
{
     ColorInOut out;
     constexpr float2 vertices[] = {
          float2(0, 0),
          float2(1, 0),
          float2(0, 1),
          float2(1, 1)
     };
     float2 pos = vertices[vid % 4];
     out.texCoord = pos;
     out.position = float4(pos * 2.0 - 1.0,
                           0.0, 1.0);
     return out;
}
```

```
typedef struct {
     float4 position [[position]];
     float2 texCoord;
} ColorInOut;
```

Fragment

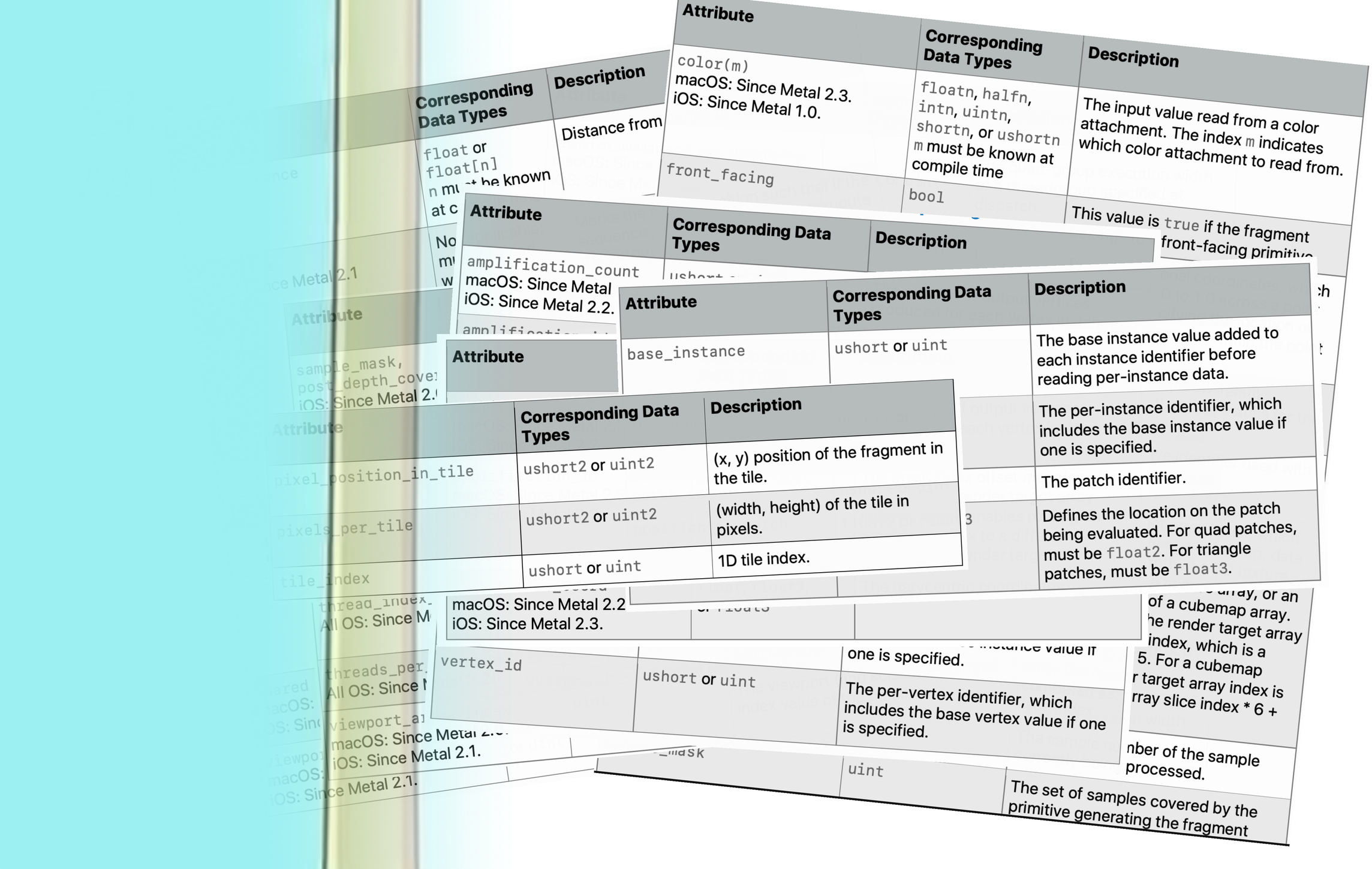
```
fragment float4 fragmentShader(
     ColorInOut in [[stage_in]])
{
     return float4(in.texCoord, 0, 1);
}
```

Shaders

Attributes

```
typedef struct {
    float4 position [[position]];
    float2 texCoord;
} TextureInOut;

typedef struct {
    float2 position [[attribute(0)]];
} Vertex;

vertex TextureInOut vshMeshBuffers(
    Vertex in [[stage_in]],
    constant float2 &image_size [[buffer(2)]])
{
    ColorInOut out;
    // ...
    return out;
}
```

```
fragment float4 fshMeshTexturePremultiply(
    TextureInOut in [[stage_in]],
    float4 back [[color(0)]],
    texture2d<float> image [[texture(0)]])
{
    constexpr sampler imageSampler(
        address::clamp_to_edge,
        filter::linear);

    float4 res = image.sample(imageSampler,
                              in.texCoord);
    res *= res;
    back *= back;
    res += (1.0 - res.a) * back;

    return sqrt(res);
}
```



Attribute	Corresponding Data Types	Description
color(m) macOS: Since Metal 2.3. iOS: Since Metal 1.0.	floatn, halfn, intn, uintn, shortn, or ushortn m must be known at compile time	The input value read from a color attachment. The index m indicates which color attachment to read from.
front_facing	bool	This value is true if the fragment front-facing primitive

Attribute	Corresponding Data Types	Description
amplification_count macOS: Since Metal iOS: Since Metal 2.2.		

Attribute	Corresponding Data Types	Description
base_instance	ushort or uint	The base instance value added to each instance identifier before reading per-instance data.
		The per-instance identifier, which includes the base instance value if one is specified.
		The patch identifier.
		Defines the location on the patch being evaluated. For quad patches, must be float2. For triangle patches, must be float3.

Attribute	Corresponding Data Types	Description
pixel_position_in_tile	ushort2 or uint2	(x, y) position of the fragment in the tile.
pixels_per_tile	ushort2 or uint2	(width, height) of the tile in pixels.
tile_index	ushort or uint	1D tile index.

Attribute	Corresponding Data Types	Description
vertex_id	ushort or uint	The per-vertex identifier, which includes the base vertex value if one is specified.
	uint	The set of samples covered by the primitive generating the fragment

Shaders

Attributes

- `[[  buffer(index)   ]]`
- `[[ texture(index)   ]]`
- `[[    stage_in      ]]`
- `[[    position      ]]`
- `[[ attribute(index) ]]`
- `[[   color(index)   ]]`

Command Buffer

Command Buffer

render somewhere

```
// ...
if let commandBuffer = commandQueue.makeCommandBuffer() {
    // encoding
    render(in: commandBuffer)

    commandBuffer.commit()
}
// ...
```

Command Buffer

render on screen

```
// ...
if let commandBuffer = commandQueue.makeCommandBuffer() {
    // encoding
    render(in: commandBuffer)

    // update view
    if let drawable = view.currentDrawable {
        commandBuffer.present(drawable)
    }

    commandBuffer.commit()
}
// ...
```

Command Buffer

render and wait until buffer started running

```
// ...
    commandBuffer.waitUntilScheduled()
// ...
```

render and wait until whole command buffer is completed

```
// ...
    commandBuffer.waitUntilCompleted()
// ...
```

Render Encoder

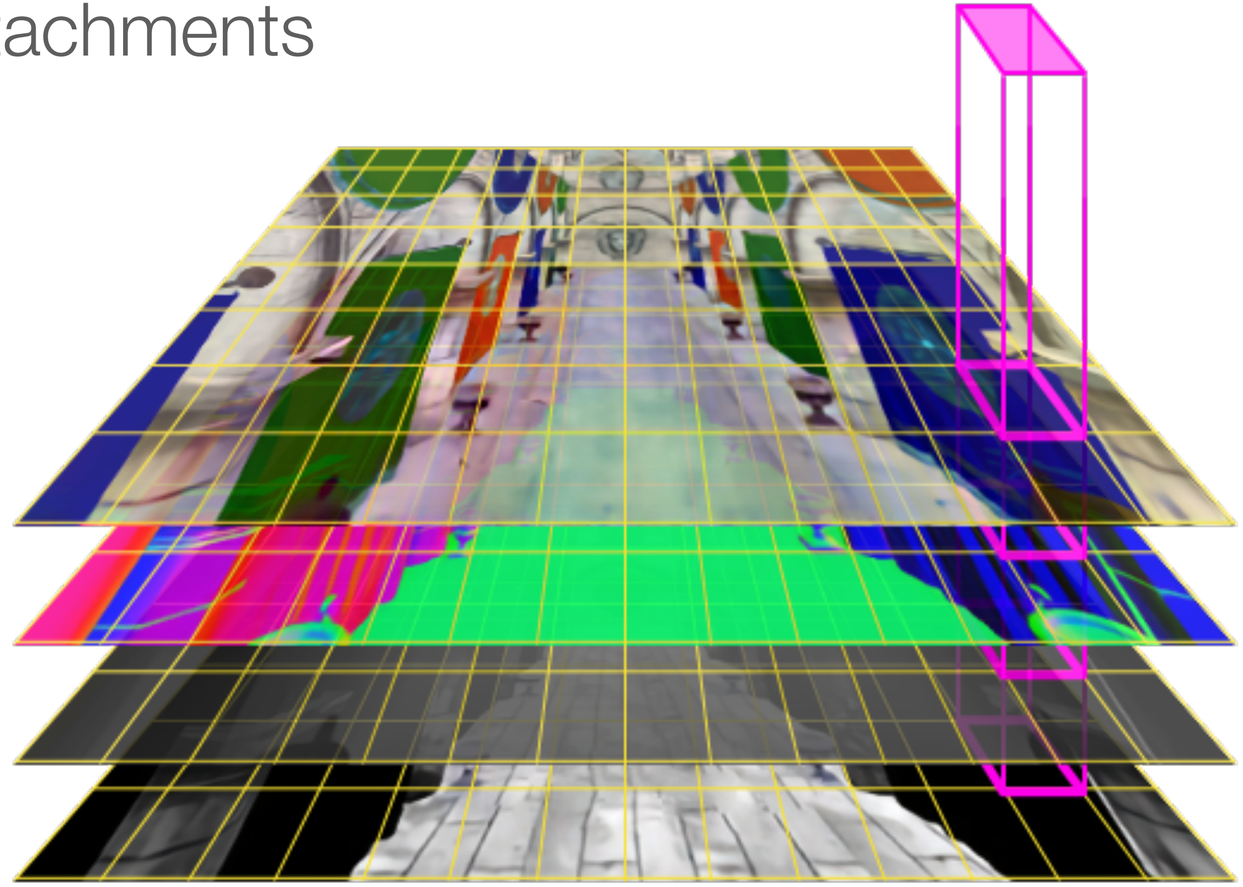
Render Encoder

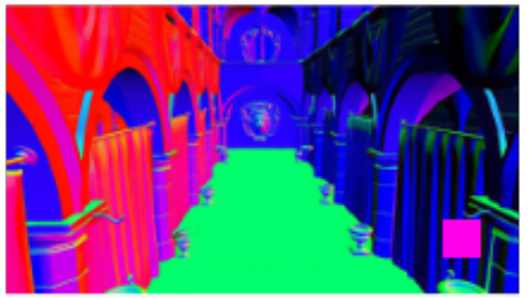
Creating and using

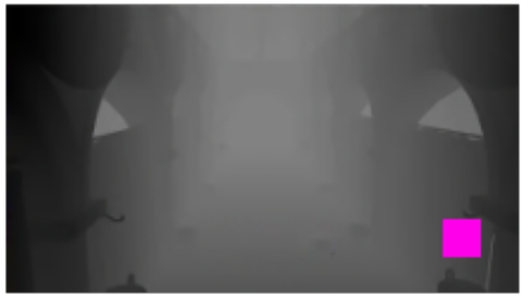
```
// ...
let renderEncoderDescr = MTLRenderPassDescriptor()
renderEncoderDescr.colorAttachments[0].texture = outTexture
renderEncoderDescr.colorAttachments[0].clearColor = MTLClearColorMake(0,0.2,0.2,1)
renderEncoderDescr.colorAttachments[0].loadAction = .clear

if let renderEncoder = commandBuffer.makeRenderCommandEncoder(descriptor: renderEncoderDescr) {

    renderEncoder.setRenderPipelineState(renderPipeline))

    renderEncoder.drawPrimitives(type: .triangleStrip, vertexStart: 0, vertexCount: 4)

    renderEncoder.endEncoding()
}
// ...
```

Render Encoder

Attachments







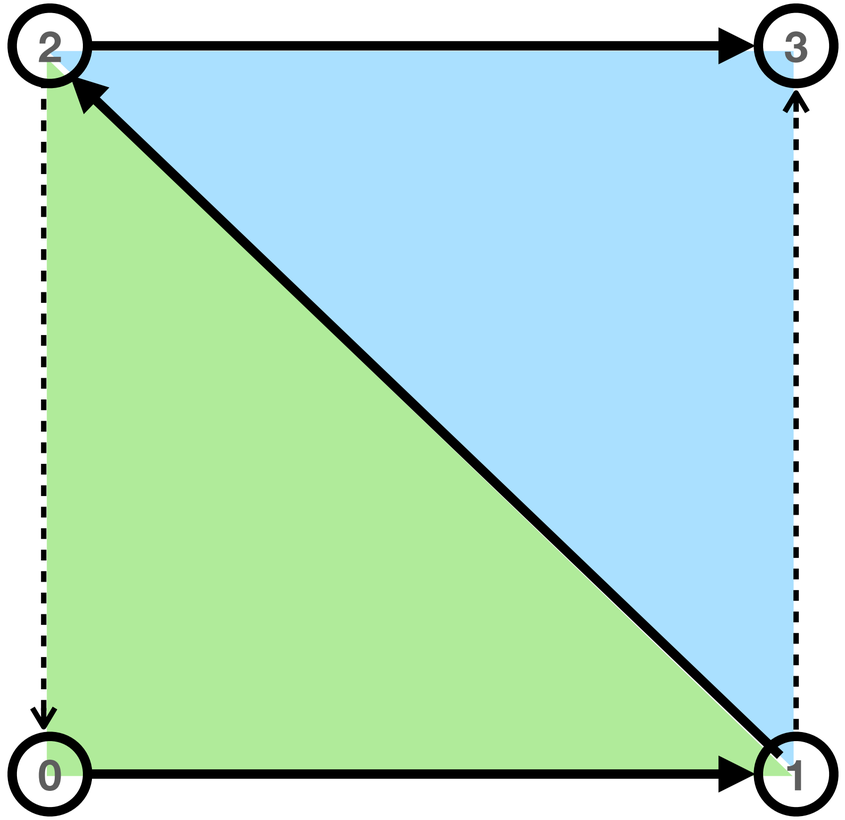
Render Encoder

Draw Call

```
// ...
renderEncoder.drawPrimitives(type: .triangleStrip, vertexStart: 0, vertexCount: 4)
// ...
```

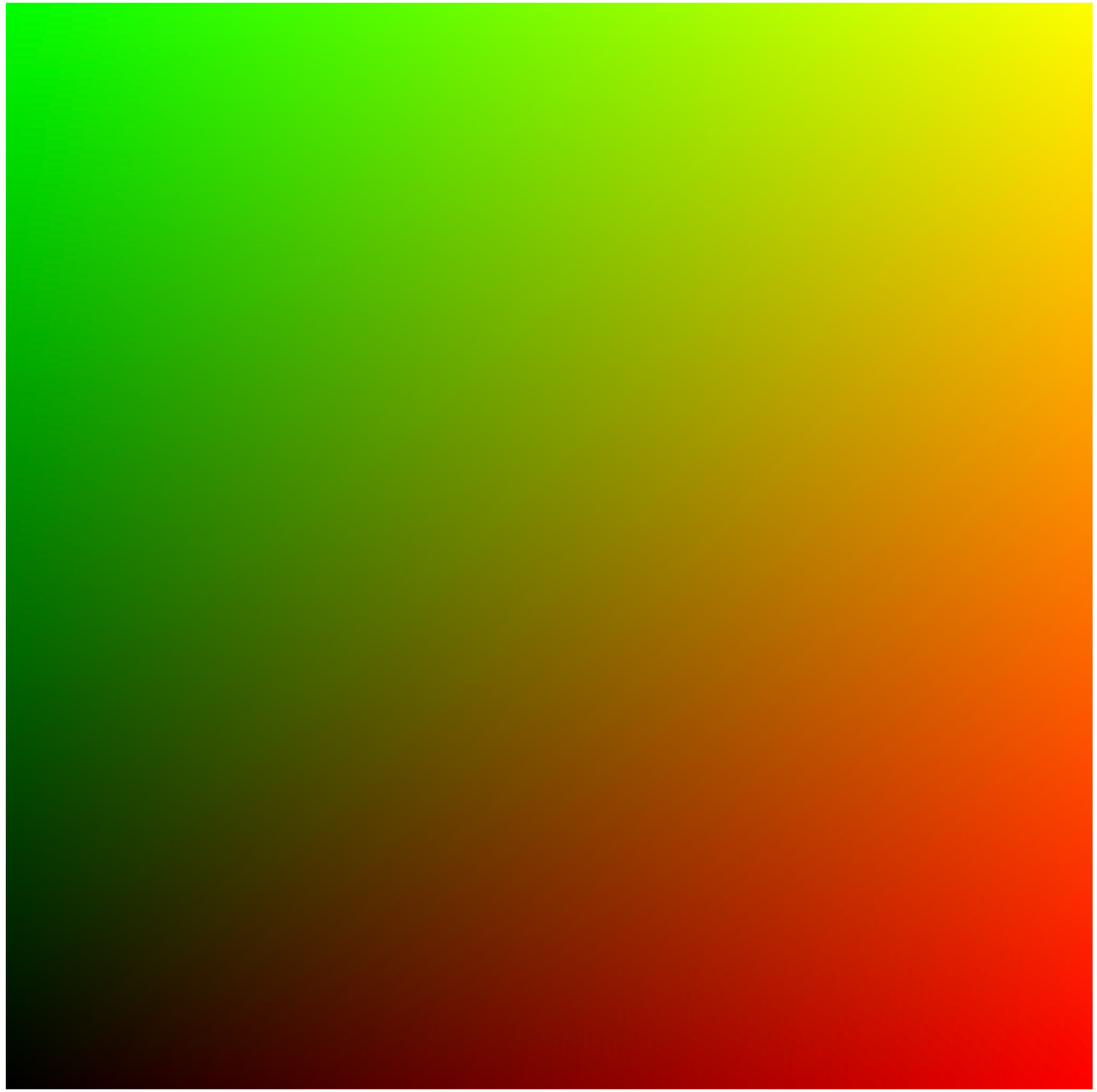
Render Encoder

(quad for triangle strip)



Render Encoder

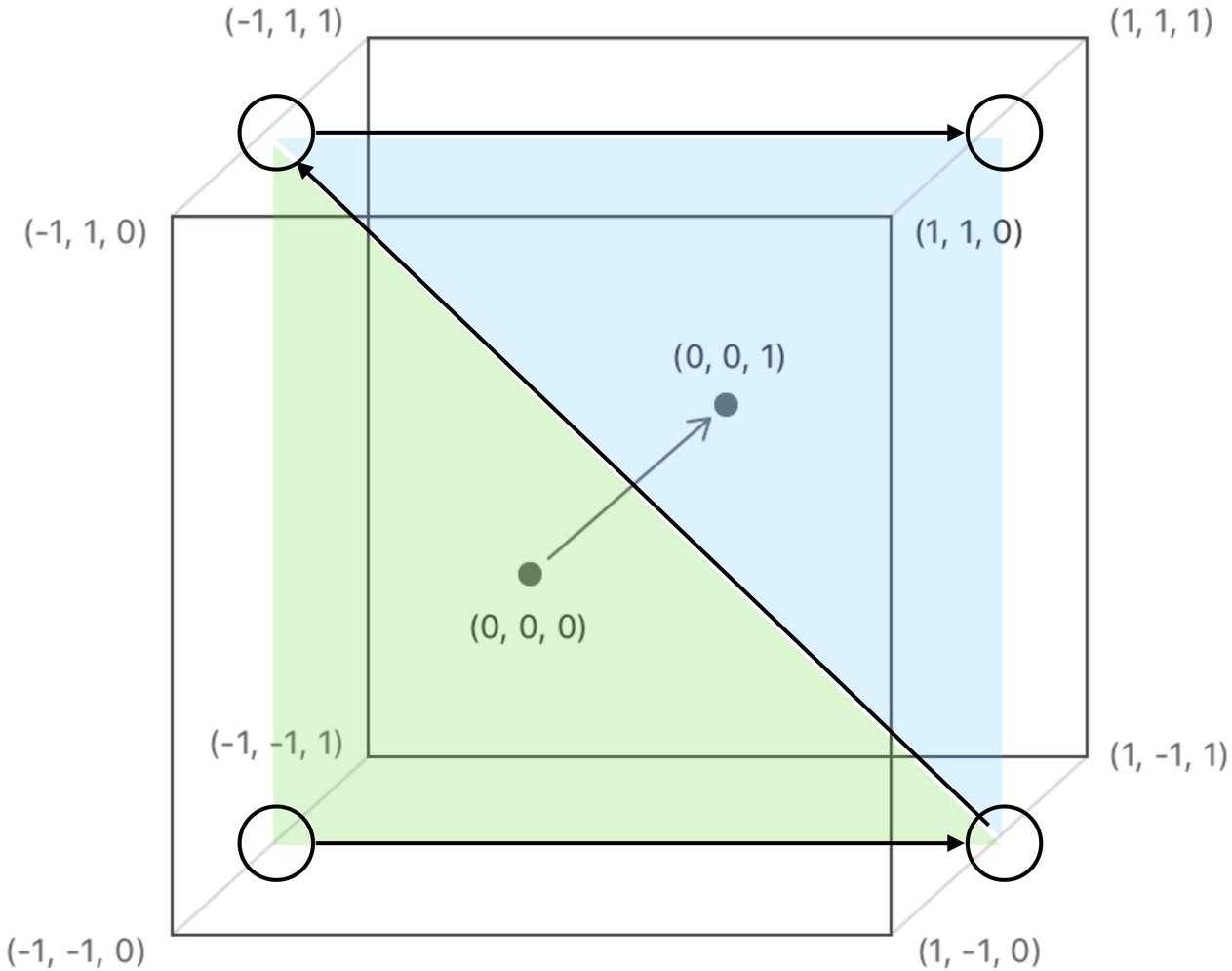
Simple full screen quad



Render Encoder

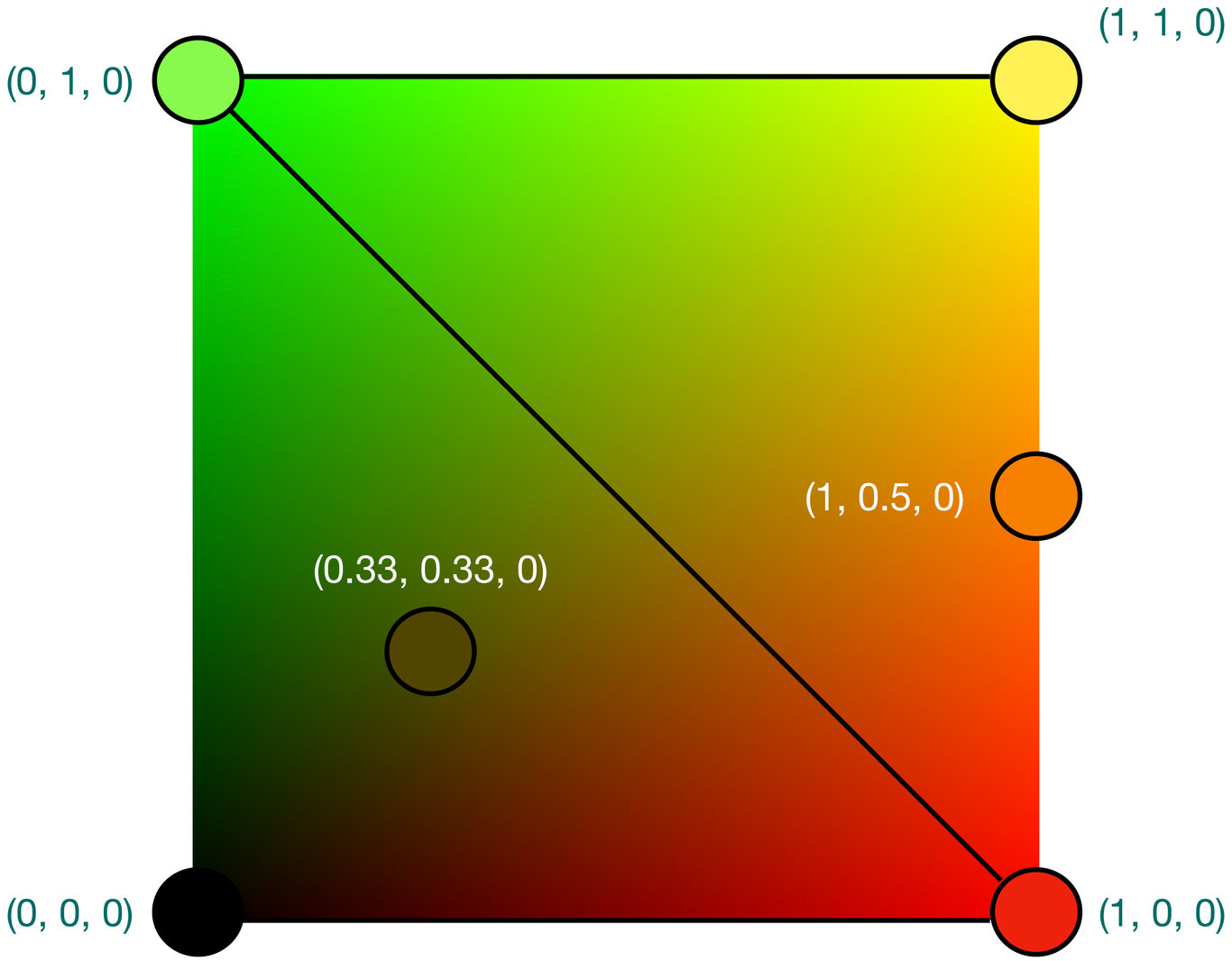
Viewport Coordinates

```
vertex ColorInOut vertexShader(
     unsigned int vid [[vertex_id]])
{
     ColorInOut out;
     constexpr float2 vertices[] = {
          float2(0, 0),
          float2(1, 0),
          float2(0, 1),
          float2(1, 1)
     };
     float2 pos = vertices[vid % 4];
     out.texCoord = pos;
     out.position = float4(pos * 2.0 - 1.0,
                           0.0, 1.0);
     return out;
}
```



Render Encoder

barycentric interpolation



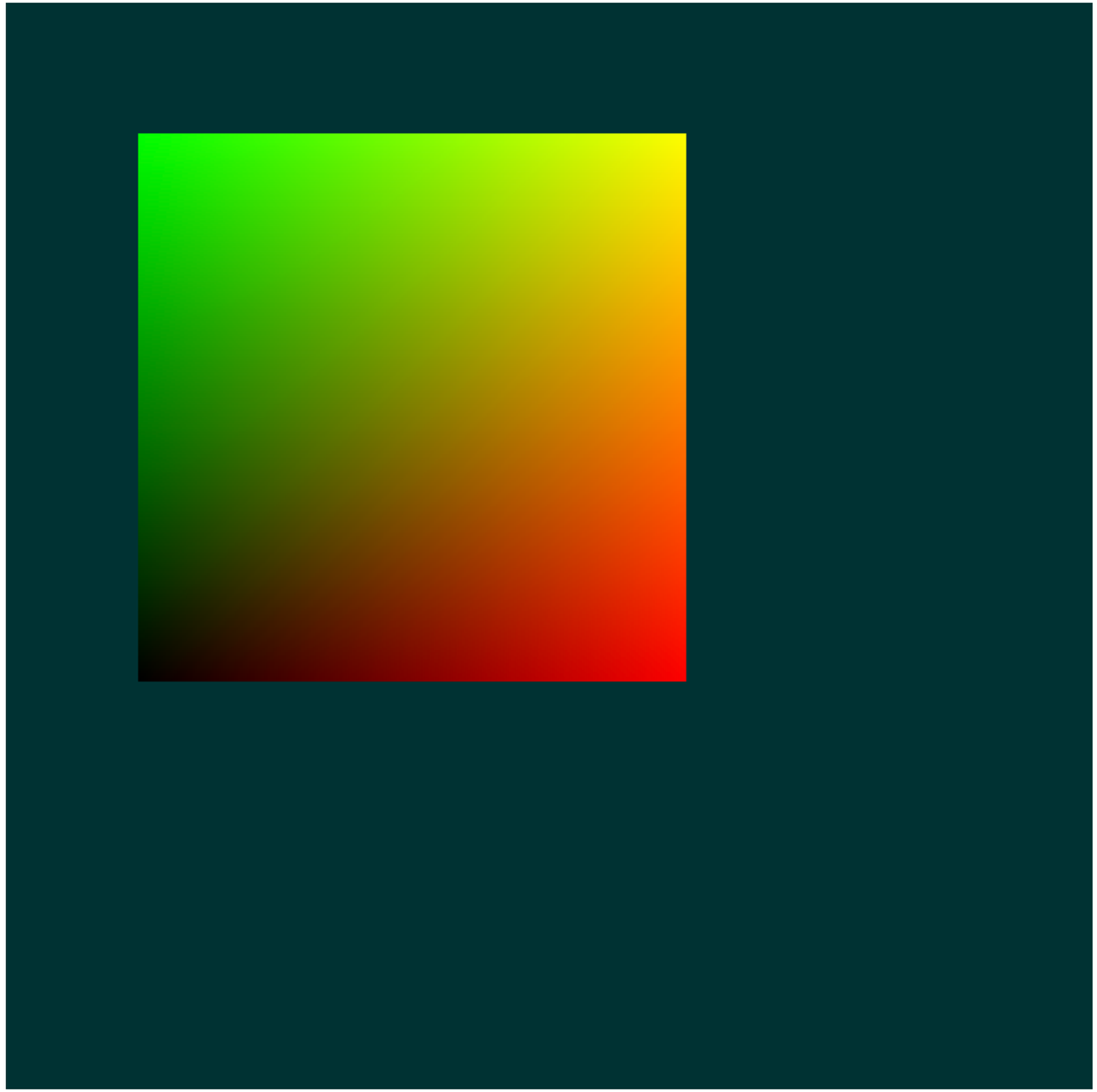
Render Encoder

Viewport

```
// ...

    renderEncoder.setViewport(MTLViewport(originX: Double(size.width) * 0.125,
                                          originY: Double(size.height) * 0.125,
                                          width: Double(size.width) * 0.5,
                                          height: Double(size.height) * 0.5,
                                          znear: -1, zfar: 1))

// ...
```

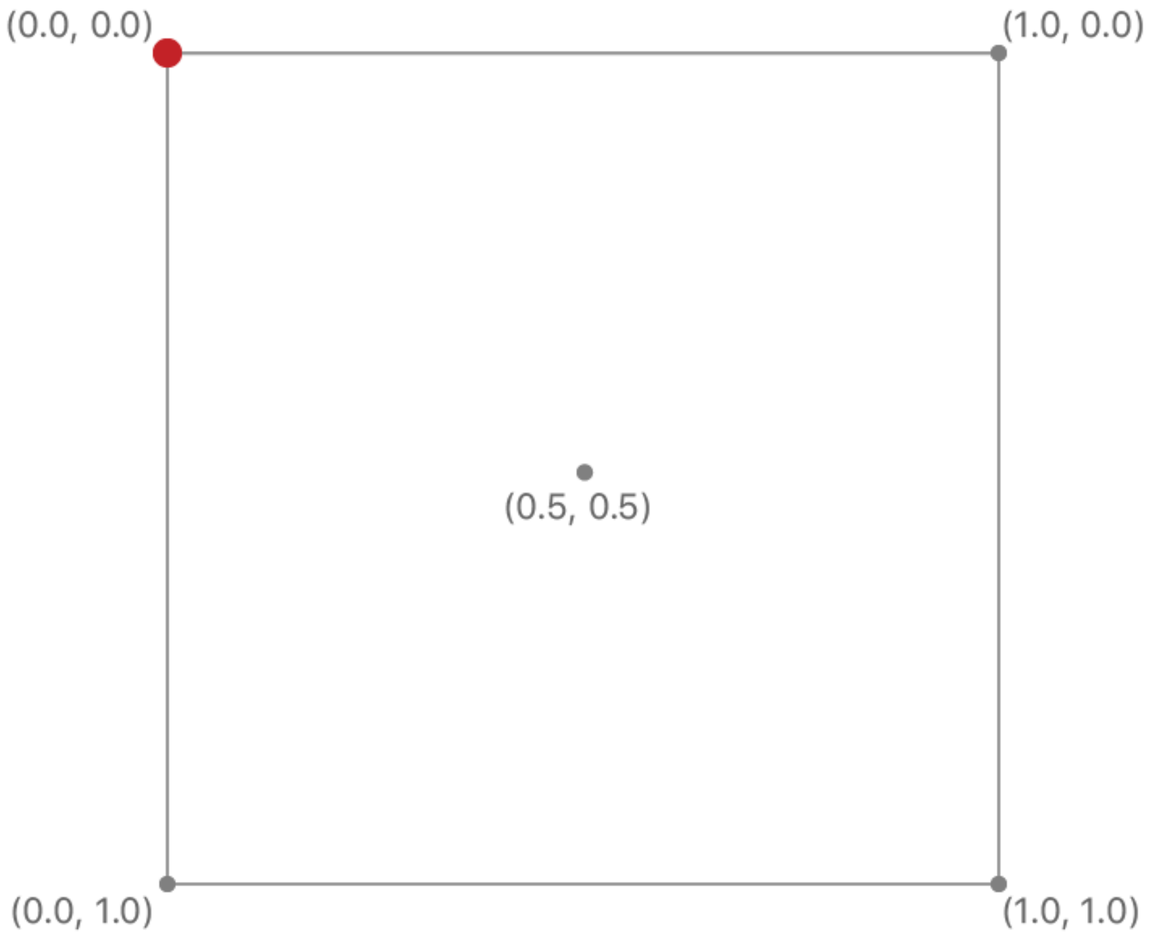
Render Encoder

Simple “full screen” quad into viewport



Render Encoder

Texture Coordinates



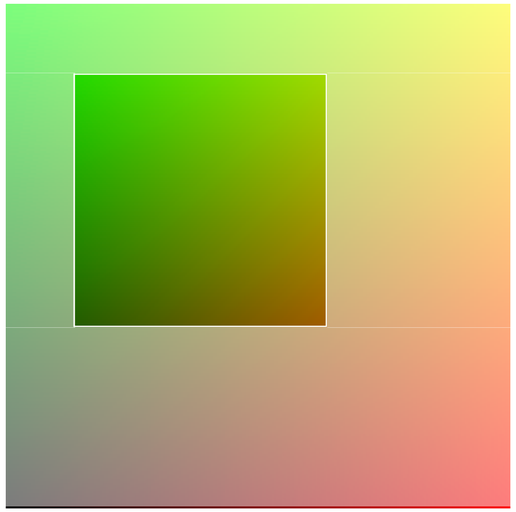
Render Encoder

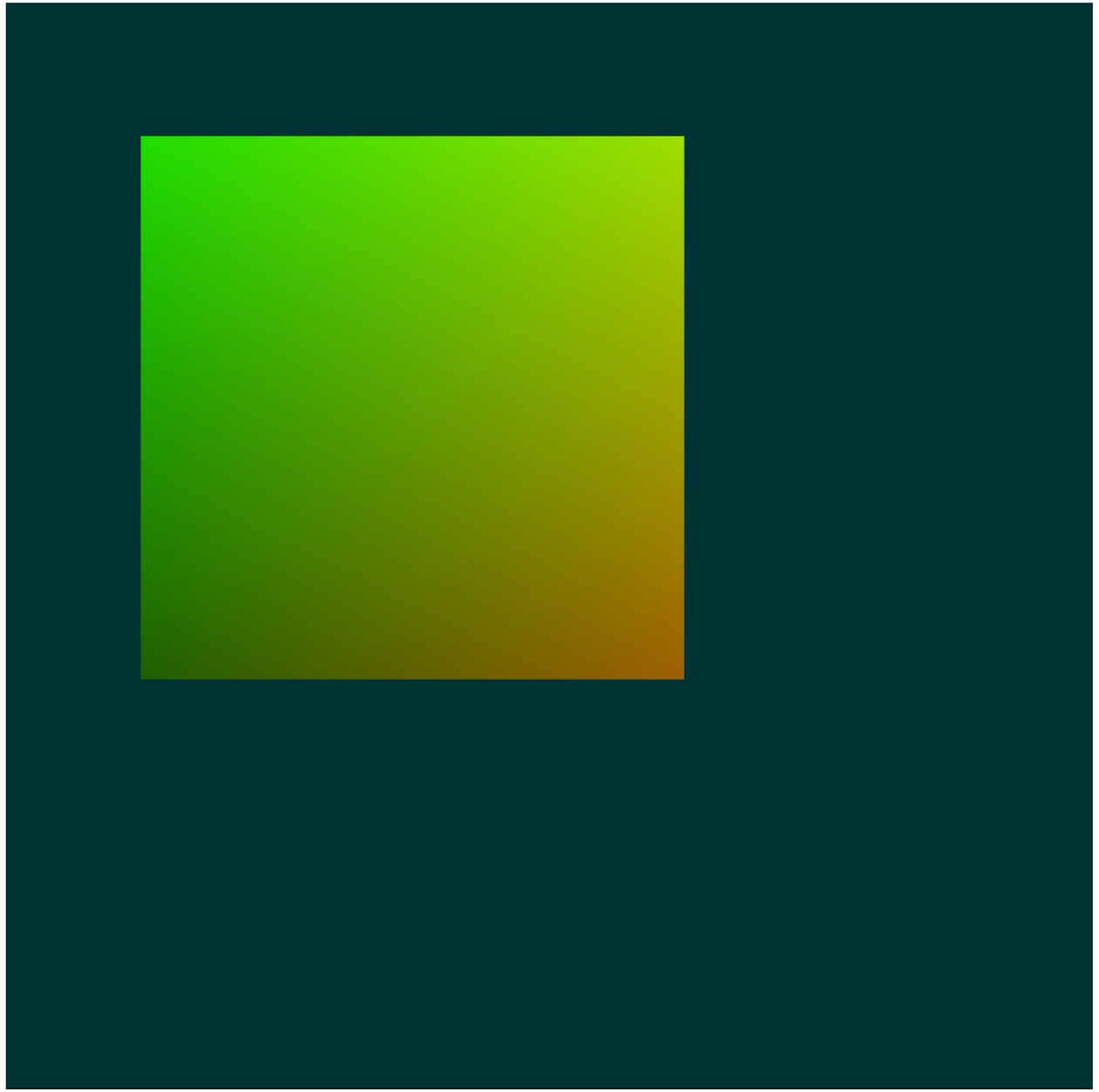
Scissor Test

```
// ...

    renderEncoder.setScissorRect(MTLScissorRect(x: Int(size.width * 0.125),
                                                y: Int(size.height * 0.125),
                                                width: Int(size.width * 0.5),
                                                height: Int(size.width * 0.5)))

// ...
```

Render Encoder

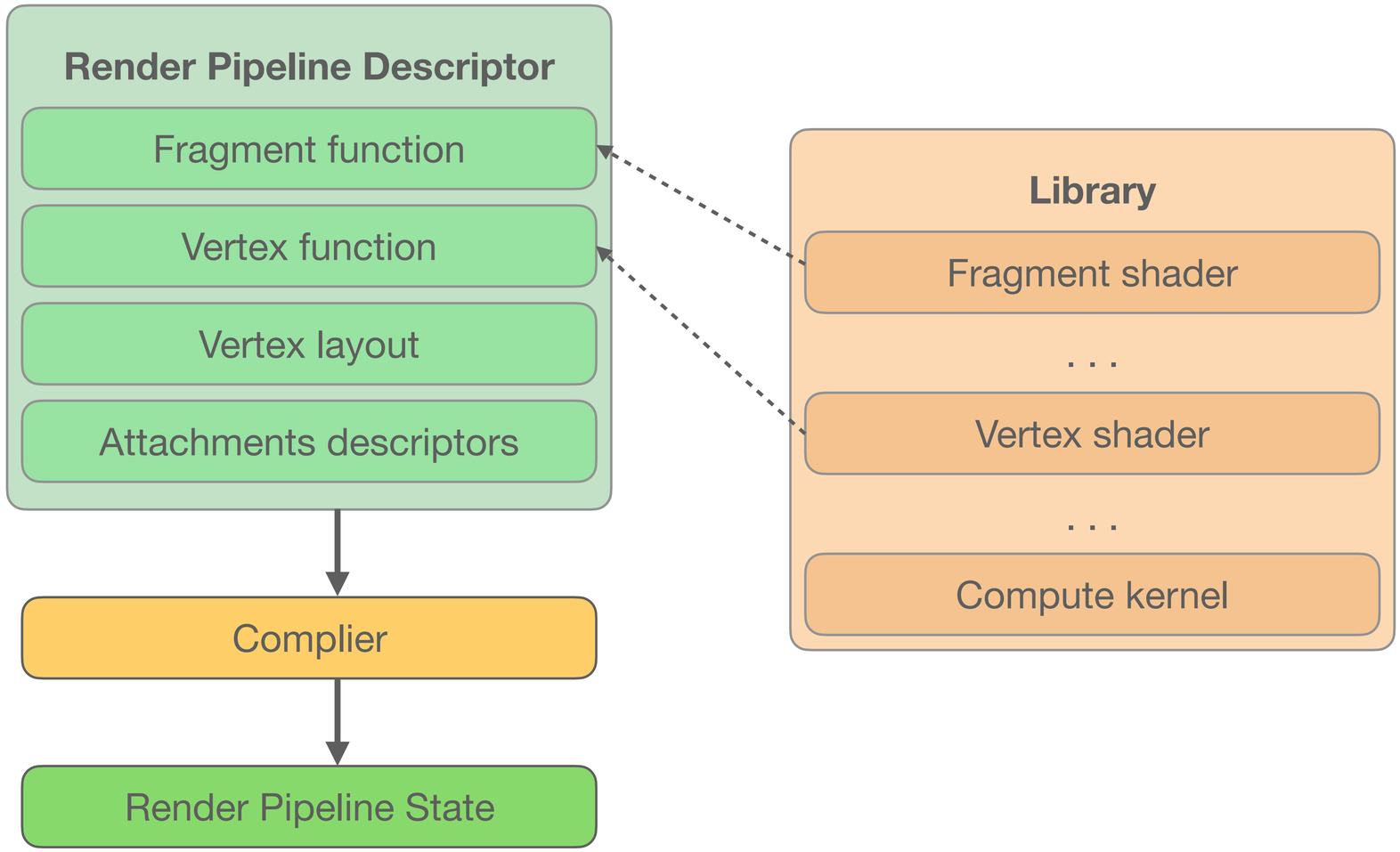
Simple full screen quad with scissor test





Pipeline

Render State



Pipeline

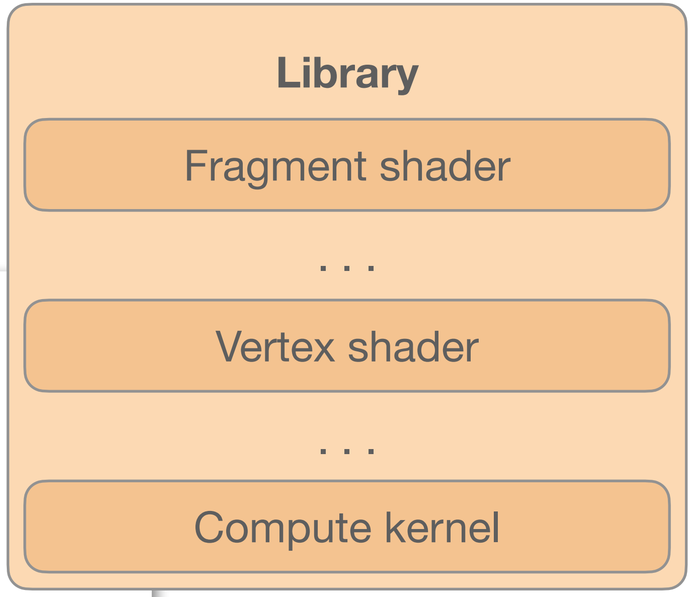
Creating

```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```

Pipeline

Creating

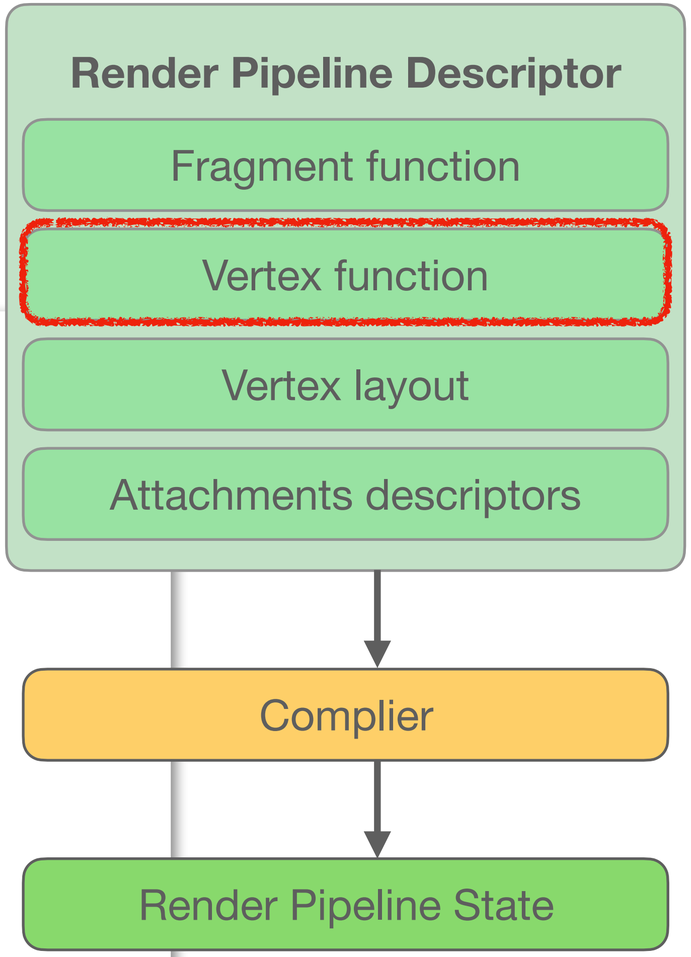
```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Pipeline

Creating

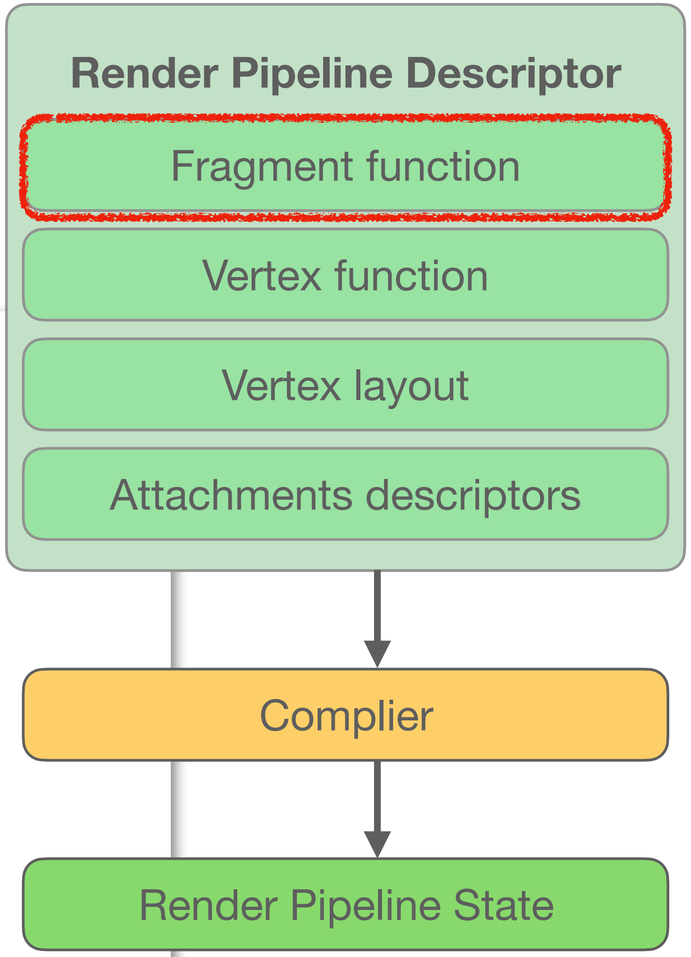
```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Pipeline

Creating

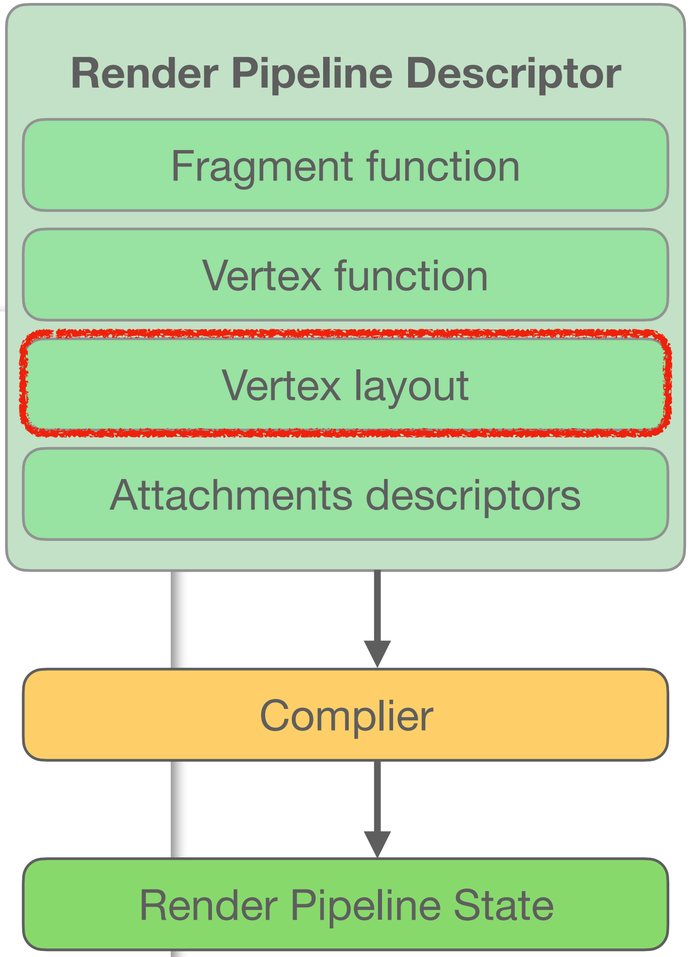
```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Pipeline

Creating

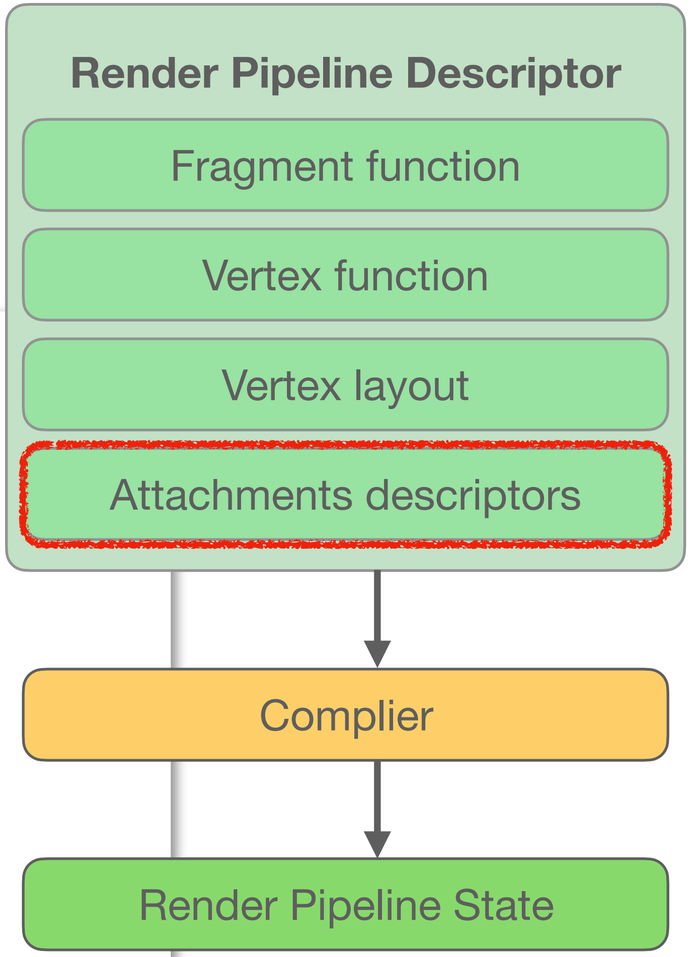
```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Pipeline

Creating

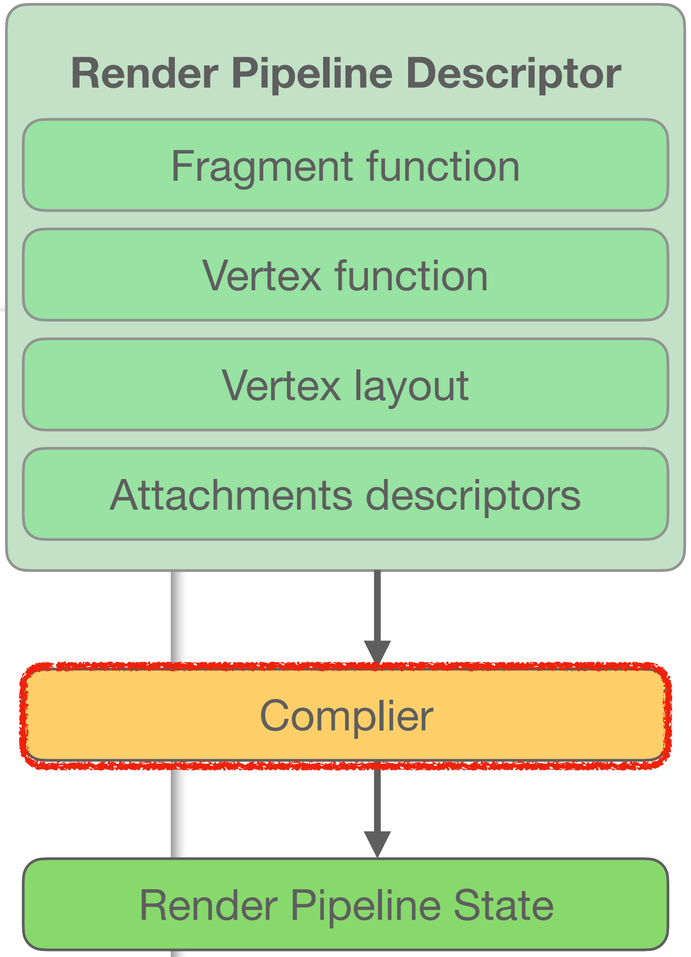
```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Pipeline

Creating

```
func buildRenderPipeline(device: MTLDevice, format: MTLPixelFormat,
                         fragment: String, vertex: String,
                         vertexDescriptor: MTLVertexDescriptor? = nil)
throws -> MTLRenderPipelineState
{
    let library = device.makeDefaultLibrary()
    let vertexFunction = library?.makeFunction(name: vertex)
    let fragmentFunction = library?.makeFunction(name: fragment)

    let pipelineDescriptor = MTLRenderPipelineDescriptor()
    pipelineDescriptor.vertexFunction = vertexFunction
    pipelineDescriptor.fragmentFunction = fragmentFunction
    if let vertexDescriptor = vertexDescriptor {
        pipelineDescriptor.vertexDescriptor = vertexDescriptor
    }
    pipelineDescriptor.colorAttachments[0].pixelFormat = format

    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



Draw Call

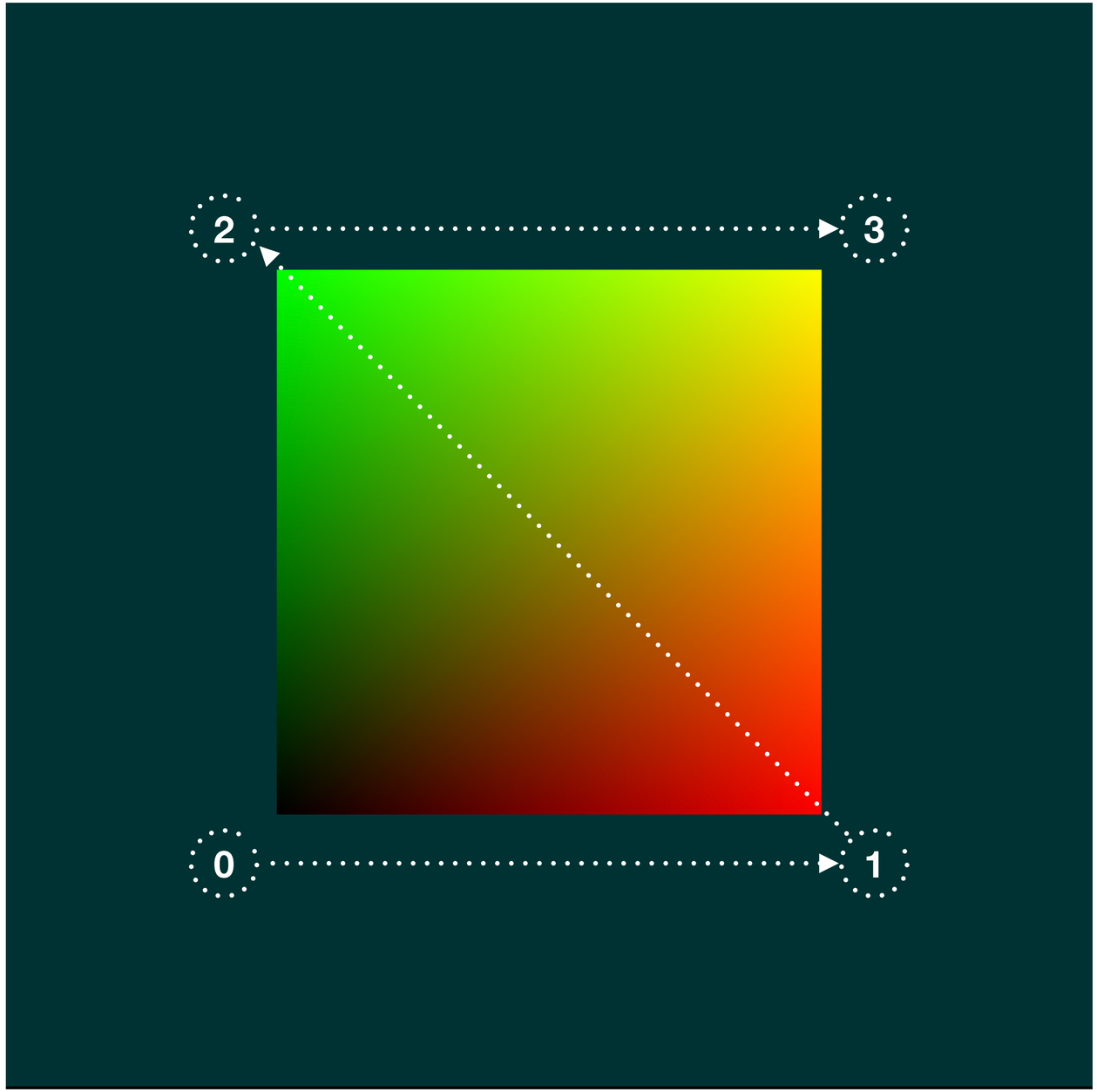
Draw Call

Primitives (MTLPrimitiveType)

- `triangleStrip`
- `triangle`
- `line`
- `lineStrip`
- `point`

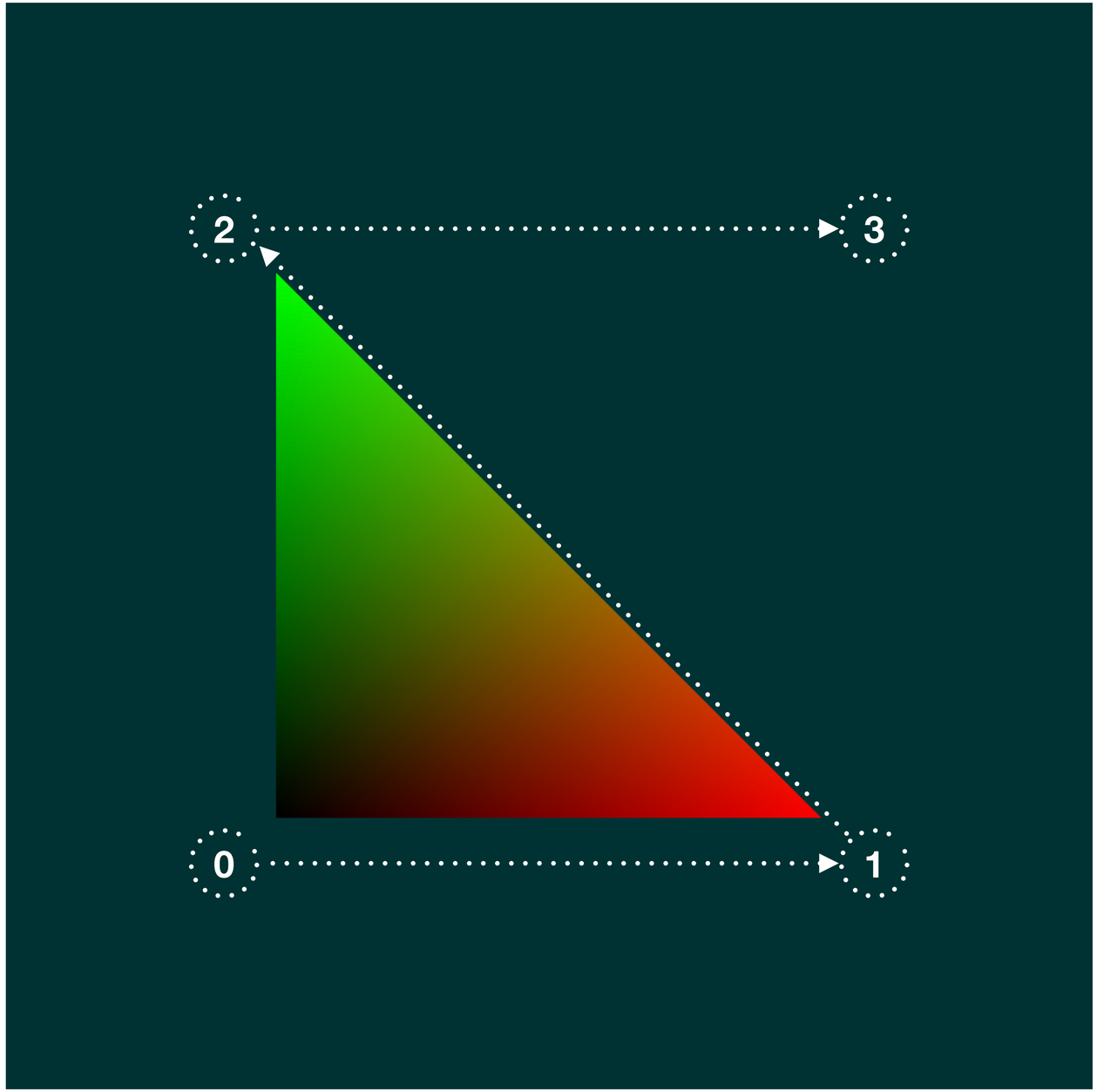
Draw Call Primitive

triangleStrip



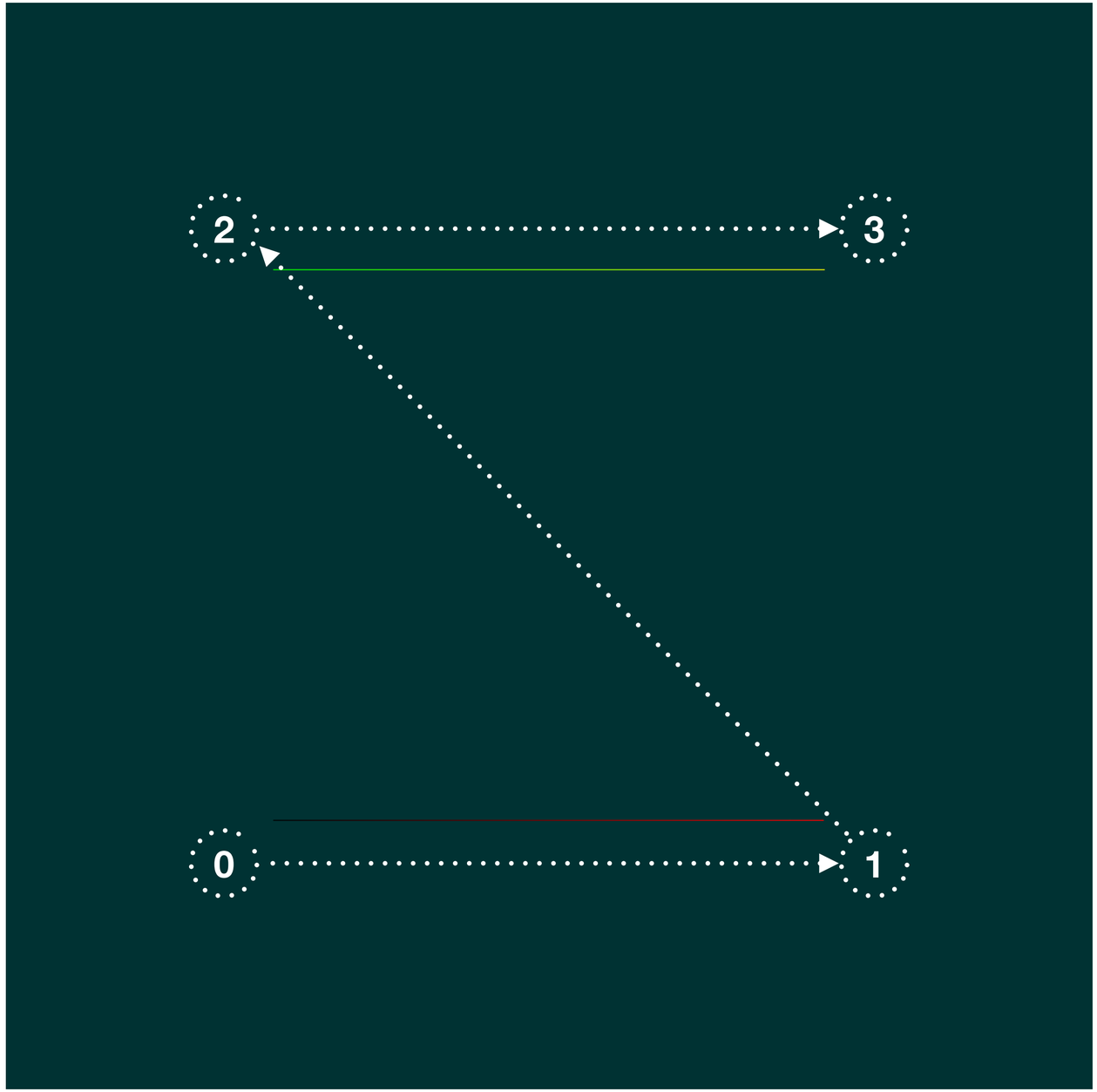
Draw Call Primitive

triangle



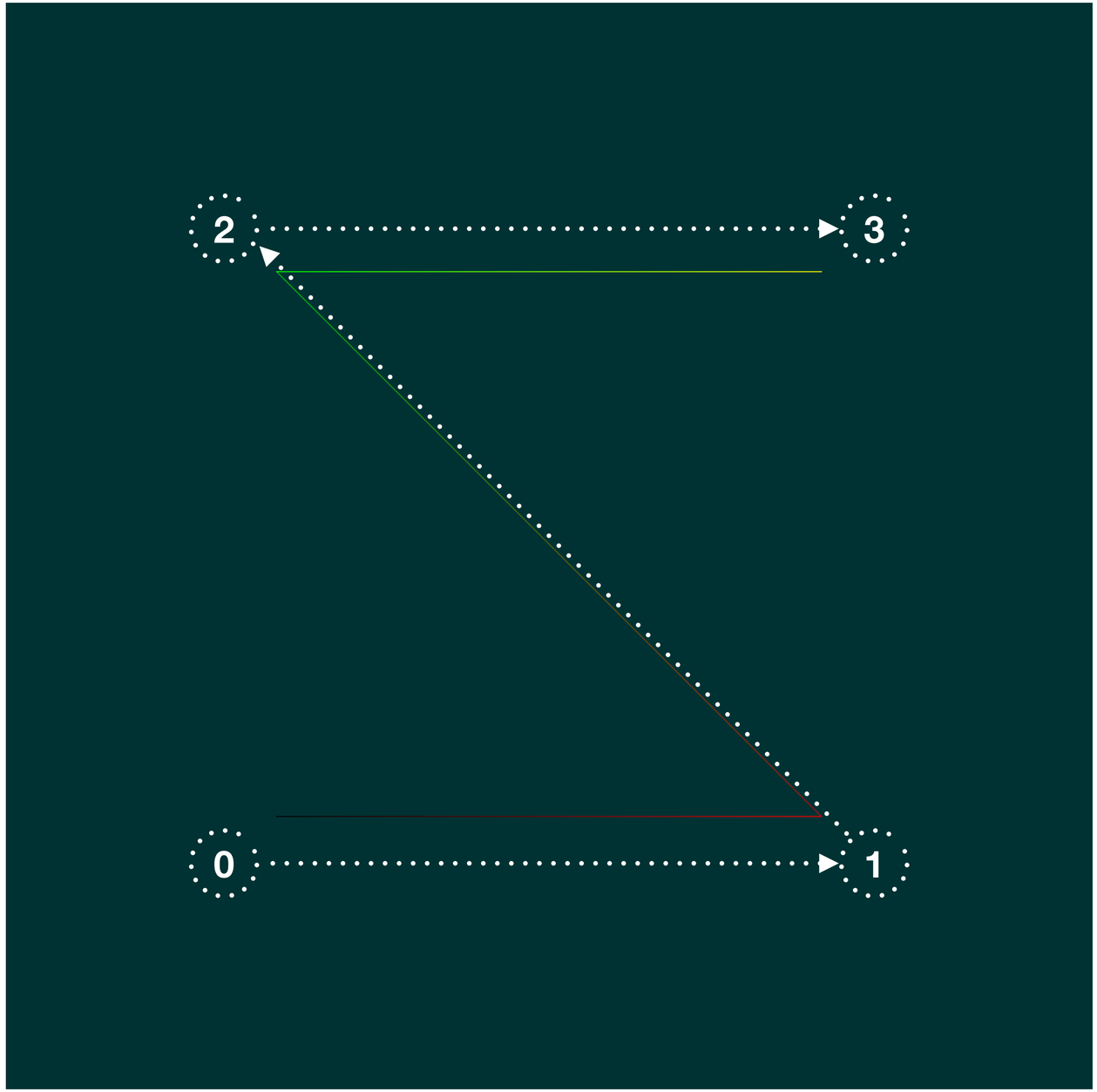
Draw Call Primitive

line



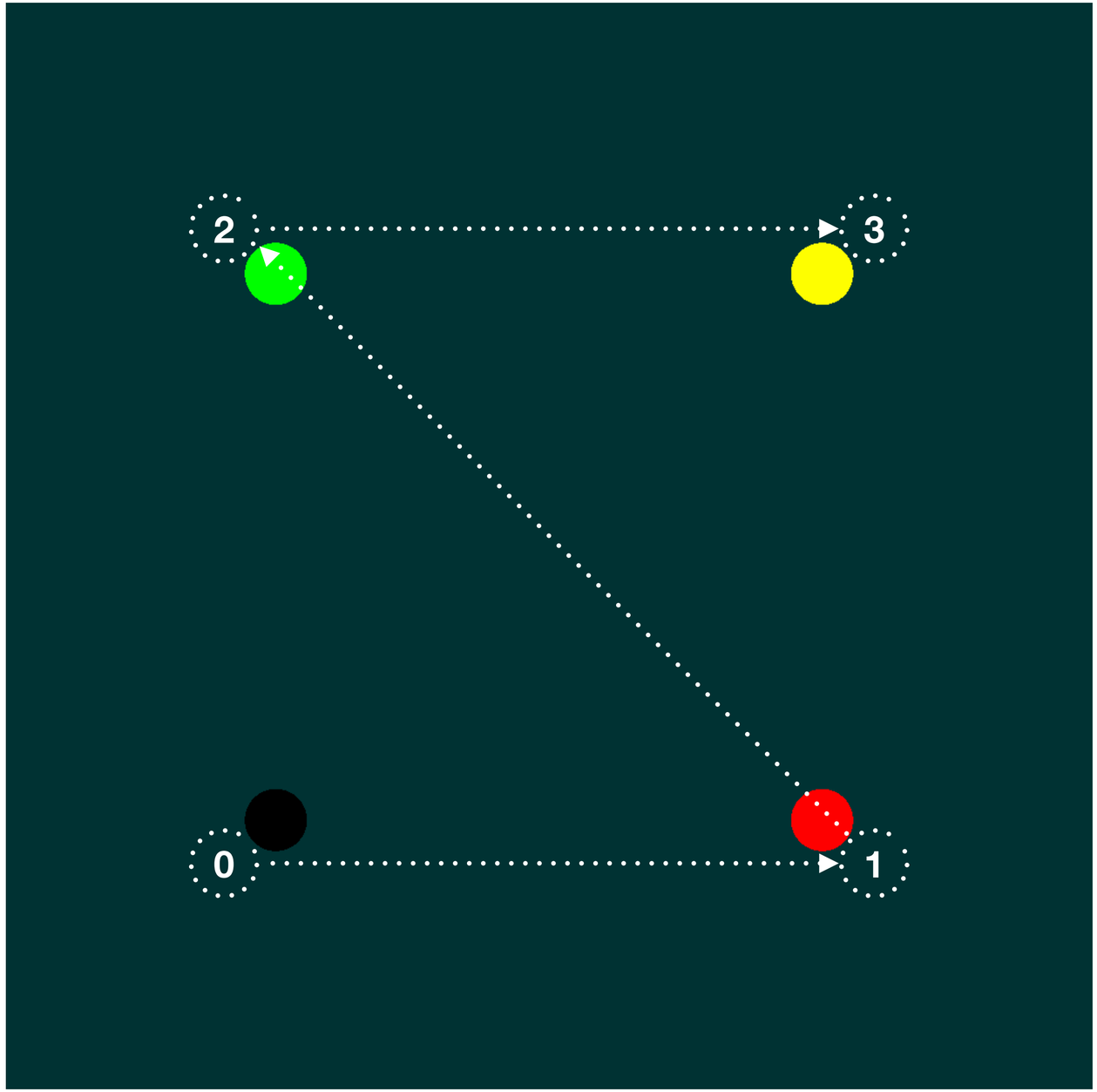
Draw Call Primitive

lineStrip



Draw Call Primitive

point



Draw Call Primitive

point

```
vertex ColorInOut vshPoint(
  unsigned int vid [[vertex_id]])
{
    ColorInOut out;
    constexpr float2 vertices[] = {
        float2(0, 0),
        float2(1, 0),
        float2(0, 1),
        float2(1, 1)
    };
    float2 position = vertices[vid % 4];
    out.pointSize = 64;
    out.texCoord = position;
    out.position =
      float4(position - 0.5, 0.0, 1.0);

    return out;
}
```

```
typedef struct {
    float4 position [[position]];
    float pointSize [[point_size]];
    float2 texCoord;
} ColorInOut;
```

```
fragment float4 fshPoint(
  ColorInOut in [[stage_in]],
  float2 pointCoord  [[point_coord]])
{
    if (length(pointCoord - 0.5) > 0.5) {
        discard_fragment();
    }
    return float4(in.texCoord, 0, 1);
}
```

Draw Call

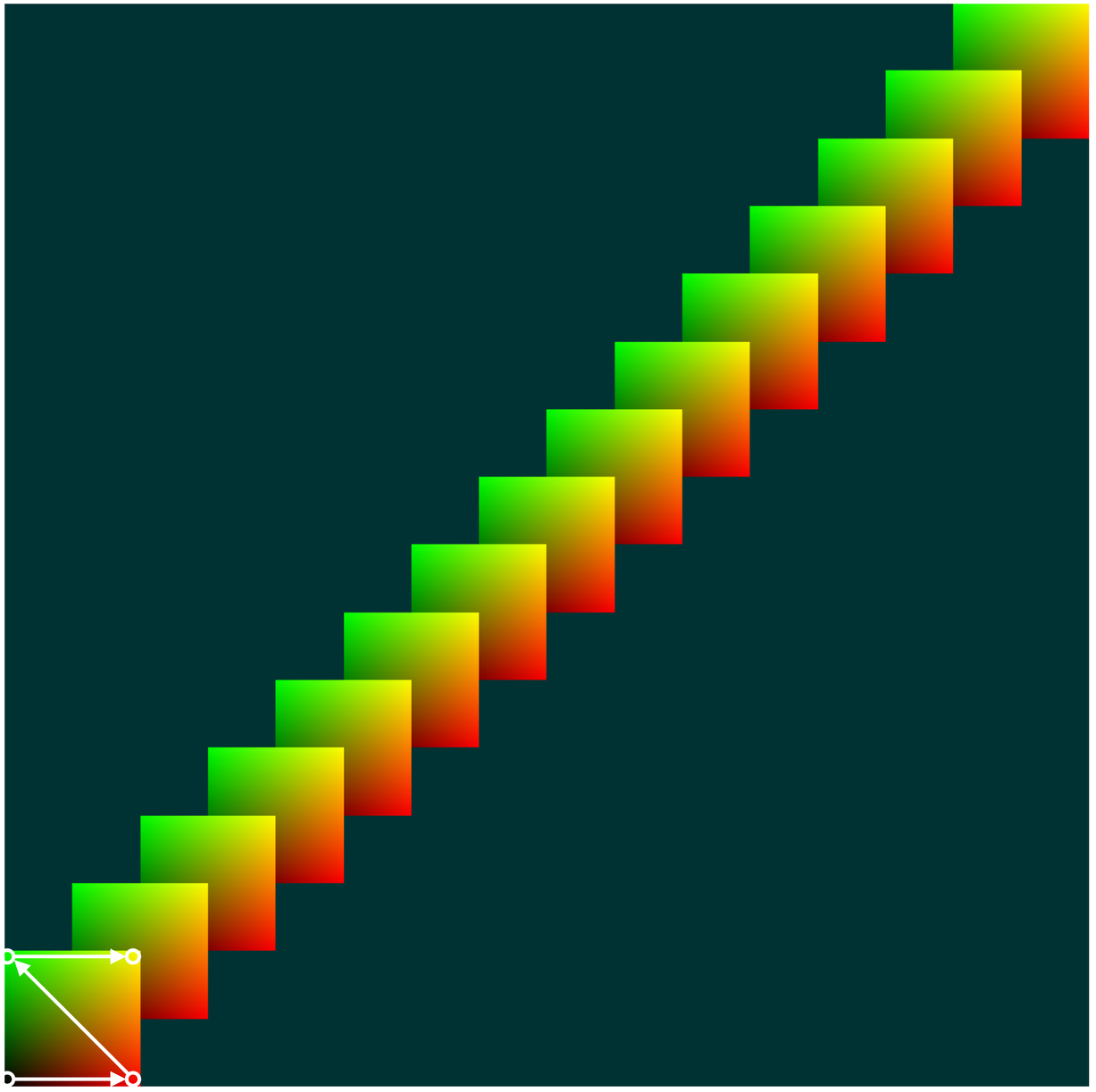
Vertices and Instances

```
renderEncoder.drawPrimitives(
    type: .triangleStrip,
    vertexStart: 0,
    vertexCount: 4,
    instanceCount: 15)
```

```
vertex ColorInOut vshInstances(
  unsigned int vid [[vertex_id]],
  unsigned int iid [[instance_id]])
{
    ColorInOut out;
    constexpr float2 vertices[] = {
        float2(0, 0), float2(1, 0),
        float2(0, 1), float2(1, 1)
    };
    float2 position = vertices[vid % 4];
    out.texCoord = position;
    out.position = float4(
      (1 - 0.125 - 0.125 * iid) +
      (position * 2.0 - 1.0) * 0.125,
      0.0, 1.0);
    return out;
}
```

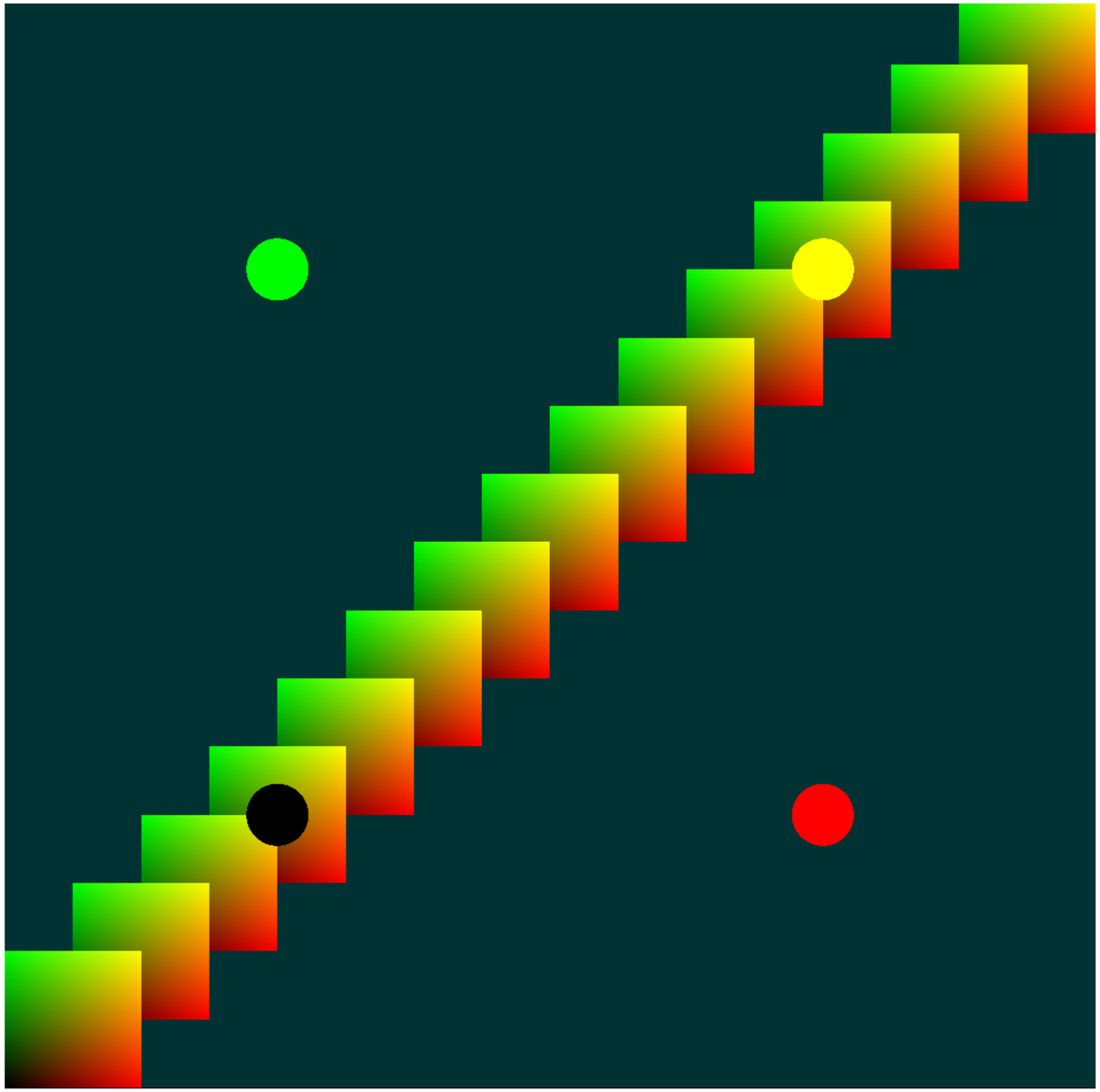
Draw Call

Vertices and Instances



Draw Call

Several draw calls



Buffers

Buffers

explicit buffer

```
var vertices: MTLBuffer?
var mesh_vertices: [SIMD2<Float32>]

// ...put vertex data into `mesh_vertices`
vertices = device.makeBuffer(
    bytes: &mesh_vertices,
    length: MemoryLayout<SIMD2<Float>>.size * mesh_vertices.count,
    options: .storageModeShared)

// ...
renderEncoder.setVertexBuffer(vertices, offset: 0, index: 0)
renderEncoder.drawPrimitives(type: .triangle,
                             vertexStart: 0,
                             vertexCount: mesh_vertices.count)
```

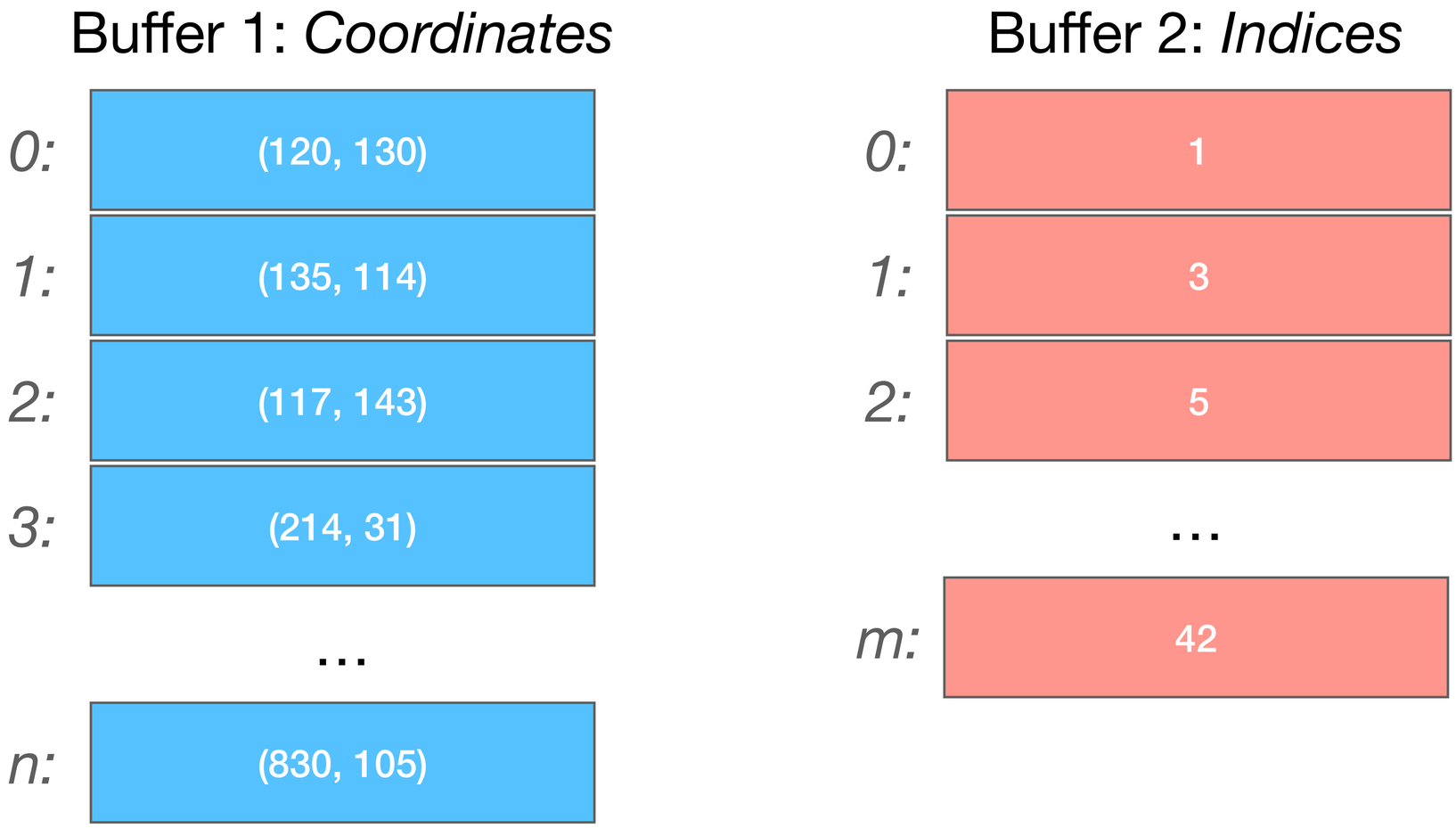
Buffers

implicit buffer

```
var imageSize: SIMD2<Float32>

// ...
renderEncoder.setVertexBytes(
    &mesh_image_size,
    length: MemoryLayout<SIMD2<Float32>>.size,
    index: 2)
```

Buffers

(mesh representation example)



Buffers

direct access from buffer with index

```
vertex ColorInOut vshMeshBuffers(unsigned int vid [[vertex_id]],
                                 device float2 *coords [[buffer(0)]],
                                 device ushort *indices [[buffer(1)]],
                                 constant float2 &image_size [[buffer(2)]]
                                 device float2 *norm_pos [[buffer(3)]])
{
    ColorInOut out;

    float2 position = coords[indices[vid]] / image_size;
    out.position = float4(position * 2.0 - 1.0, 0.0, 1.0);
    out.position.y = -out.position.y;

    norm_pos[vid] = position;

    return out;
}
```

Buffers

access with vertex layout (GPU side)

```
typedef struct {
    float2 position [[attribute(0)]];
} Vertex;

vertex ColorInOut vshMeshAttributes(Vertex in [[stage_in]],
                                    constant float2 &image_size [[buffer(2)]])
{
    ColorInOut out;

    float2 position = in.position / image_size;
    out.position = float4(position * 2.0 - 1.0, 0.0, 1.0);
    out.position.y = -out.position.y;

    return out;
}
```

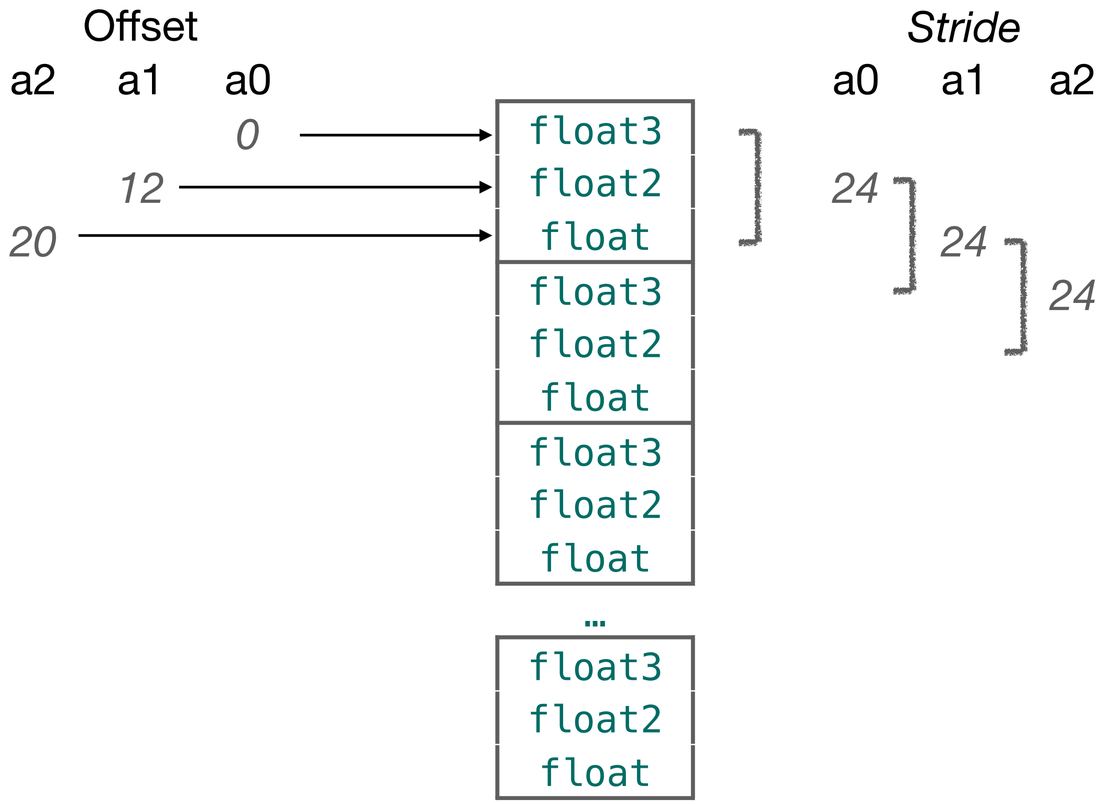
Buffers

Vertex descriptor of pipeline state

```
let vertexDescriptor = MTLVertexDescriptor()
vertexDescriptor.attributes[0].format = MTLVertexFormat.float2
vertexDescriptor.attributes[0].offset = 0
vertexDescriptor.attributes[0].bufferIndex = 0

vertexDescriptor.layouts[0].stride = MemoryLayout<SIMD2<Float32>>.size
vertexDescriptor.layouts[0].stepRate = 1
vertexDescriptor.layouts[0].stepFunction = MTLVertexStepFunction.perVertex
```

Buffers

Attributes layouts explanation



* use MemoryLayout.offset(of: \A.x) to be sure in proper offset
* use MemoryLayout<A>.size to be sure in proper stride

Buffers

Render

Vertices directly

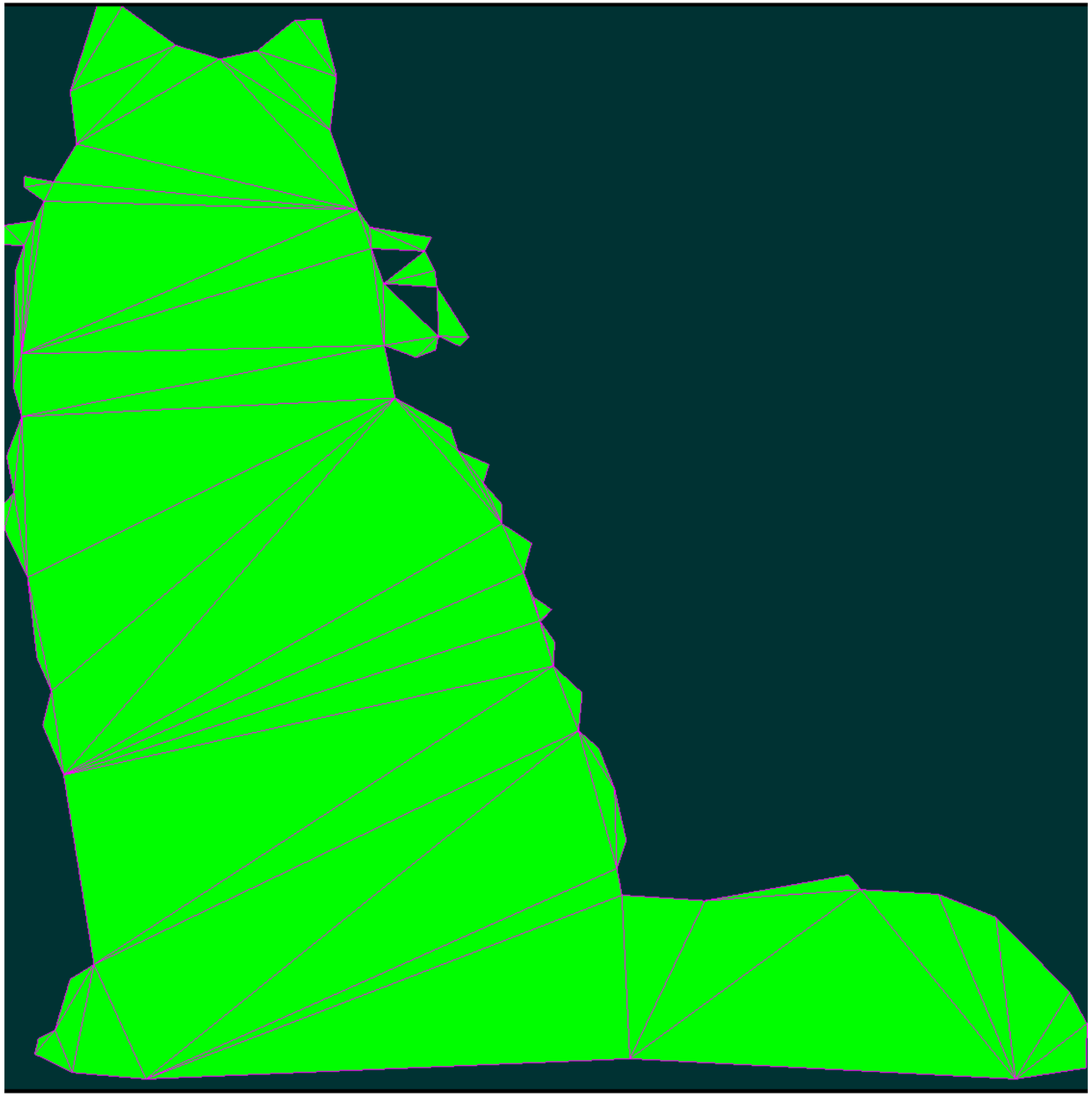
```
renderEncoder.setVertexBuffer(vertices, offset: 0, index: 0)

renderEncoder.drawPrimitives(type: .triangle,
                             vertexStart: 0,
                             vertexCount: mesh_vertices_count)
```

Vertices indexed

```
renderEncoder.setVertexBuffer(vertices, offset: 0, index: 0)

renderEncoder.drawIndexedPrimitives(type: .triangle,
                                    indexCount: mesh_indices_count,
                                    indexType: .uint16,
                                    indexBuffer: indices!,
                                    indexBufferOffset: 0)
```

Buffers

Render mesh from buffer



Textures

Textures

Creating

```
func createTexture(device: MTLDevice,
                   width: Int,
                   height: Int,
                   format: MTLPixelFormat)
-> MTLTexture
{
    let mtlTextureDescriptor = MTLTextureDescriptor();

    mtlTextureDescriptor.pixelFormat = format;
    mtlTextureDescriptor.width = width;
    mtlTextureDescriptor.height = height;
    mtlTextureDescriptor.storageMode = MTLStorageMode.private;
    mtlTextureDescriptor.usage = [
        MTLTextureUsage.shaderRead,
        MTLTextureUsage.shaderWrite,
        MTLTextureUsage.renderTarget]

    return device.makeTexture(descriptor: mtlTextureDescriptor)!
}
```

Textures

Binding

CPU: binding texture in encoder

```
renderEncoder.setFragmentTexture(image, index: 0)
```

GPU: sampling texture in a fragment shader

```
fragment float4 fshMeshTexture(TextureInOut in [[stage_in]],
                               texture2d<float> image [[ texture(0) ]])
{
    constexpr sampler imageSampler(address::clamp_to_edge, filter::linear);
    float4 res = image.sample(imageSampler, in.texCoord);
    return res;
}
```

Sampling

`repeat`

Sampling

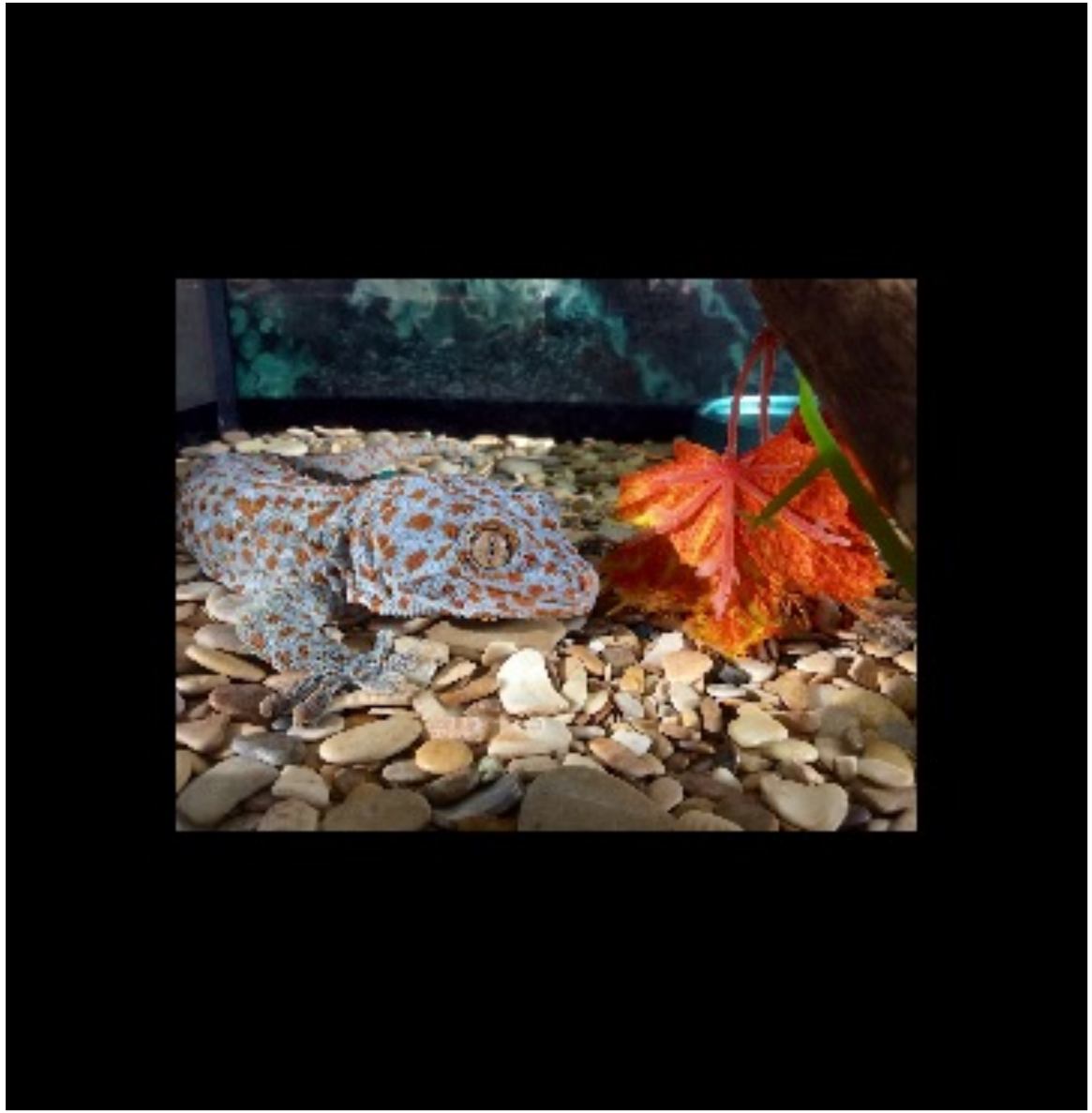
`mirrored_repeat`

Sampling

`clamp_to_edge`

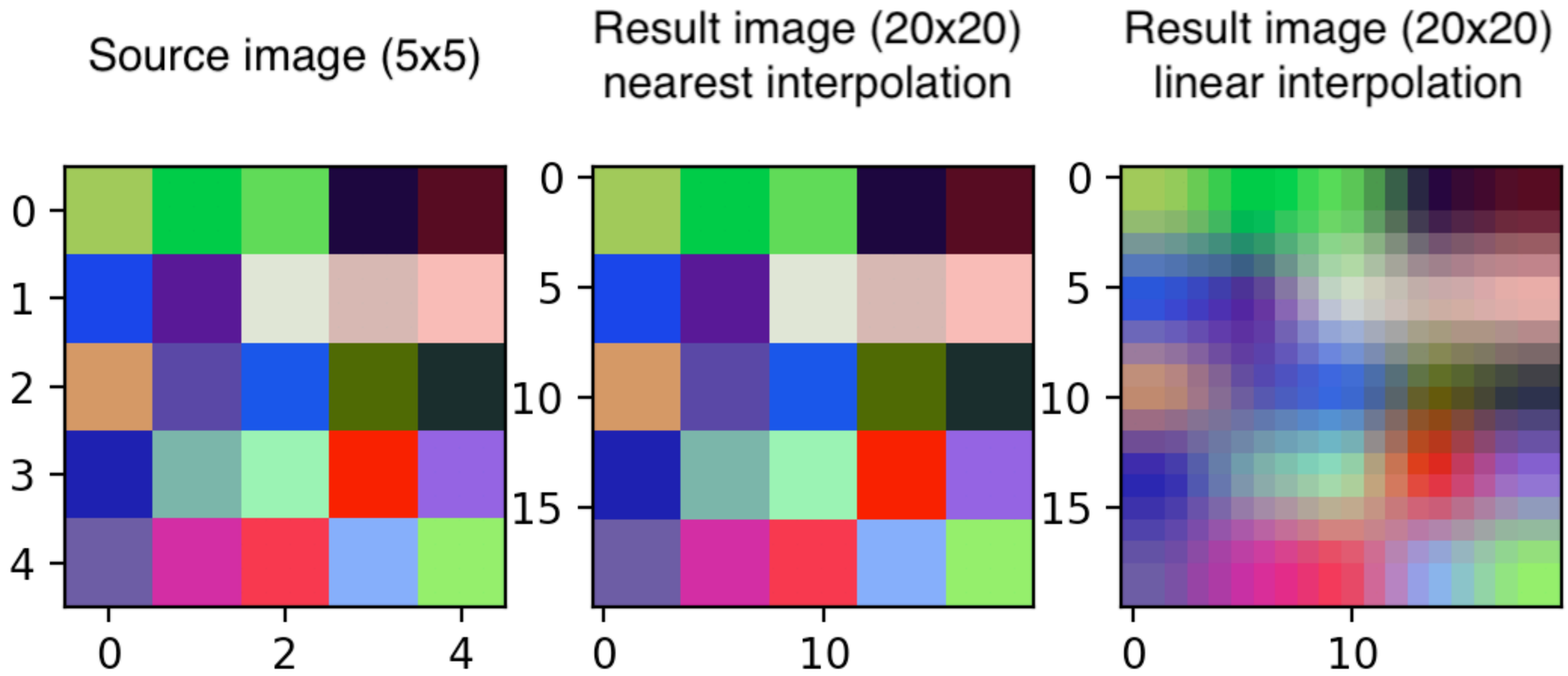
Sampling

`clamp_to_zero`



Sampling

nearest vs linear



Buffers

Render textured mesh

Blit Encoder

Compute Encoder

Compute Encoder

Compute state pipeline

```
func buildComputePipeline(device: MTLDevice, kernel: String) throws ->
MTLComputePipelineState {
    let library = device.makeDefaultLibrary()
    let kernelFunction = library?.makeFunction(name: kernel)
    return try device.makeComputePipelineState(function: kernelFunction!)
}
```

Compute Encoder

Run!

```
if let computeEncoder = commandBuffer.makeComputeCommandEncoder() {
    computeEncoder.setComputePipelineState(computePipeline)
    computeEncoder.setBuffer(vertices, offset: 0, index: 0)

    let threadgroupSize = MTLSizeMake(instances / 16, 1, 1);
    let threadgroupCount = MTLSizeMake((instances + threadgroupSize.width -
1) / threadgroupSize.width, 1, 1);

    computeEncoder.dispatchThreadgroups(
        threadgroupCount,
        threadsPerThreadgroup: threadgroupSize)

    computeEncoder.endEncoding()
}
```

Compute Encoder

Kernel

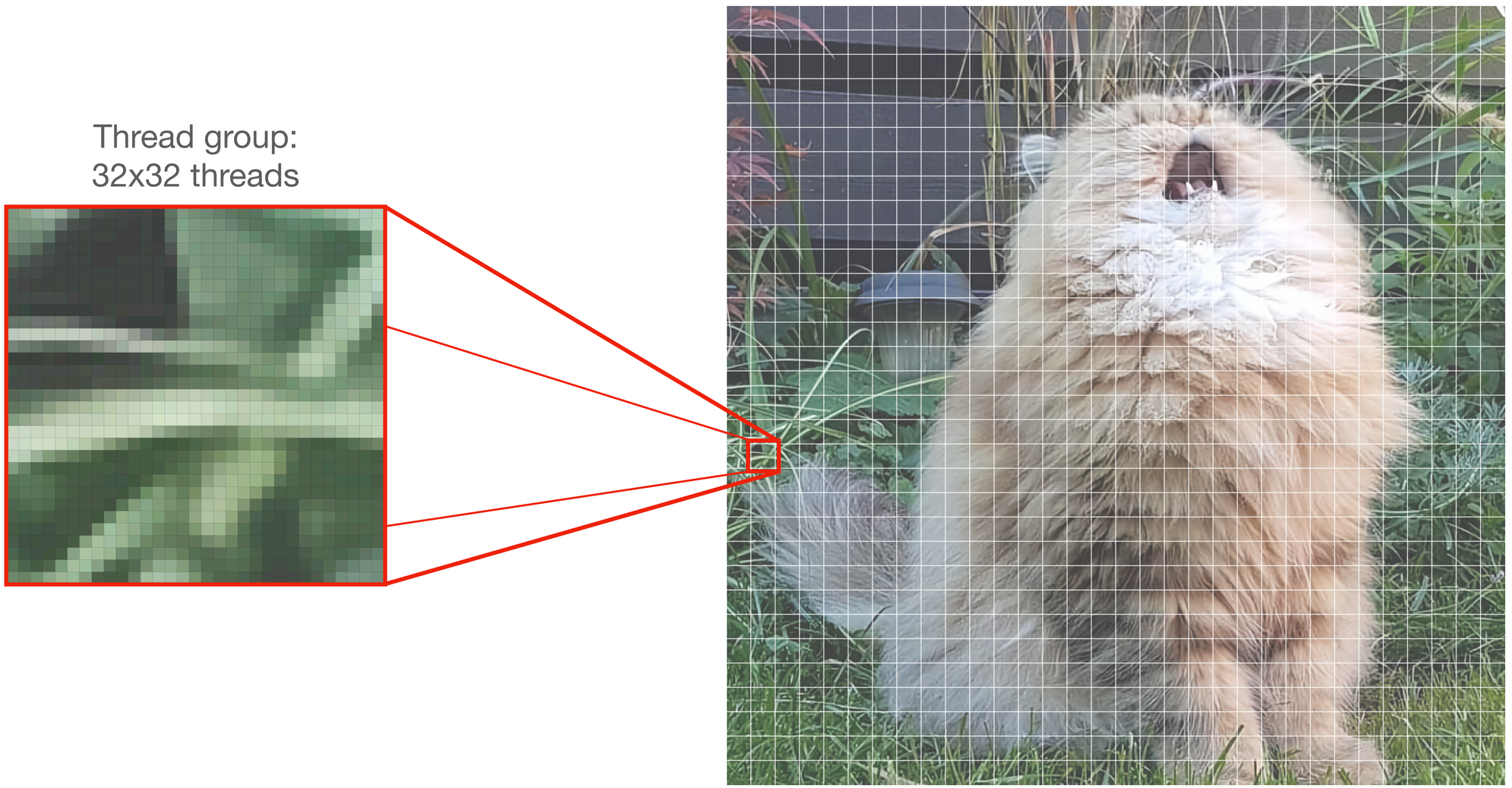
```
kernel void krnCompute(device float4 *pos_val [[buffer(0)]],
                       uint gid [[thread_position_in_grid]])
{
    float4 particle = pos_val[gid];

    if (any(abs(particle.xy) > 1)) {
        particle.xy = 0;
    }
    if (length(particle.xy) < 1e-6) {
        particle.zw = float2(gid % 128, gid / 128) / 128 - 0.5;
        particle.zw += rand2(particle.xy + particle.zw);
        particle.zw *= 0.0001;
    }
    particle.xy += particle.zw;
    particle.zw *= 1.01;

    pos_val[gid] = particle;
}
```

Compute Encoder

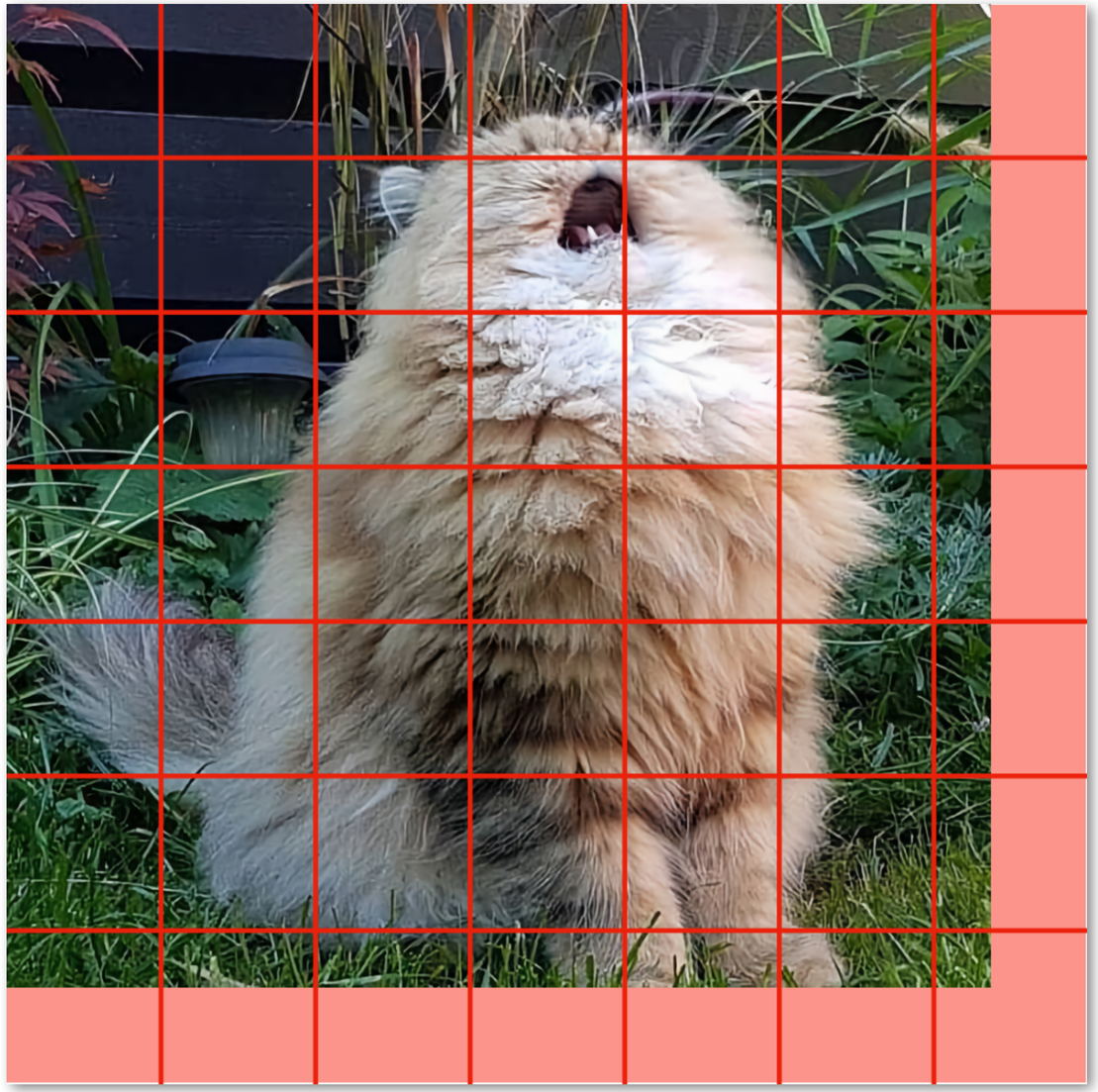
Threadgroups

32x32 thread groups
1024x1024 px

Thread group:
32x32 threads



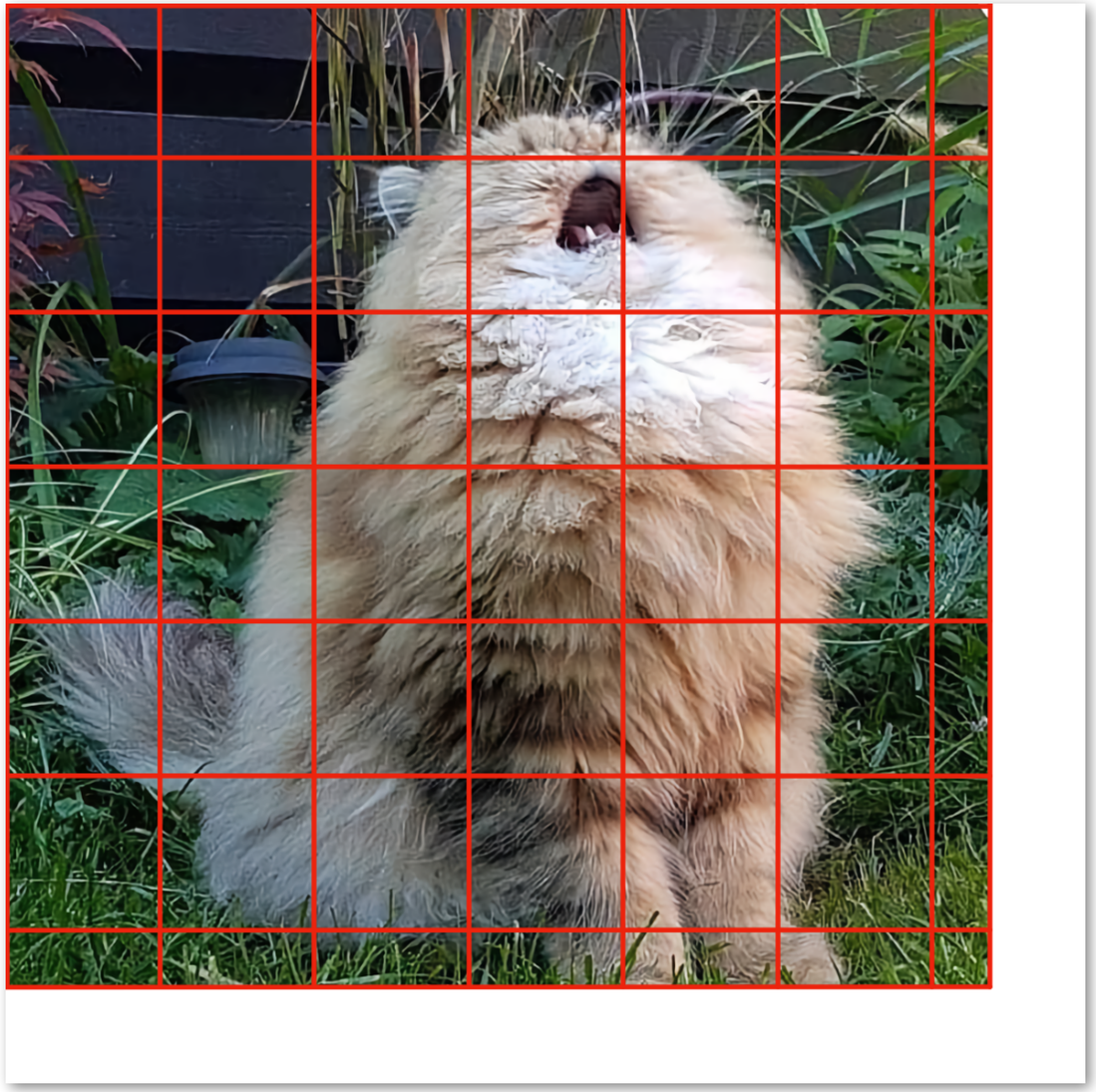
dispatchThreadgroups



dispatchThreadgroups

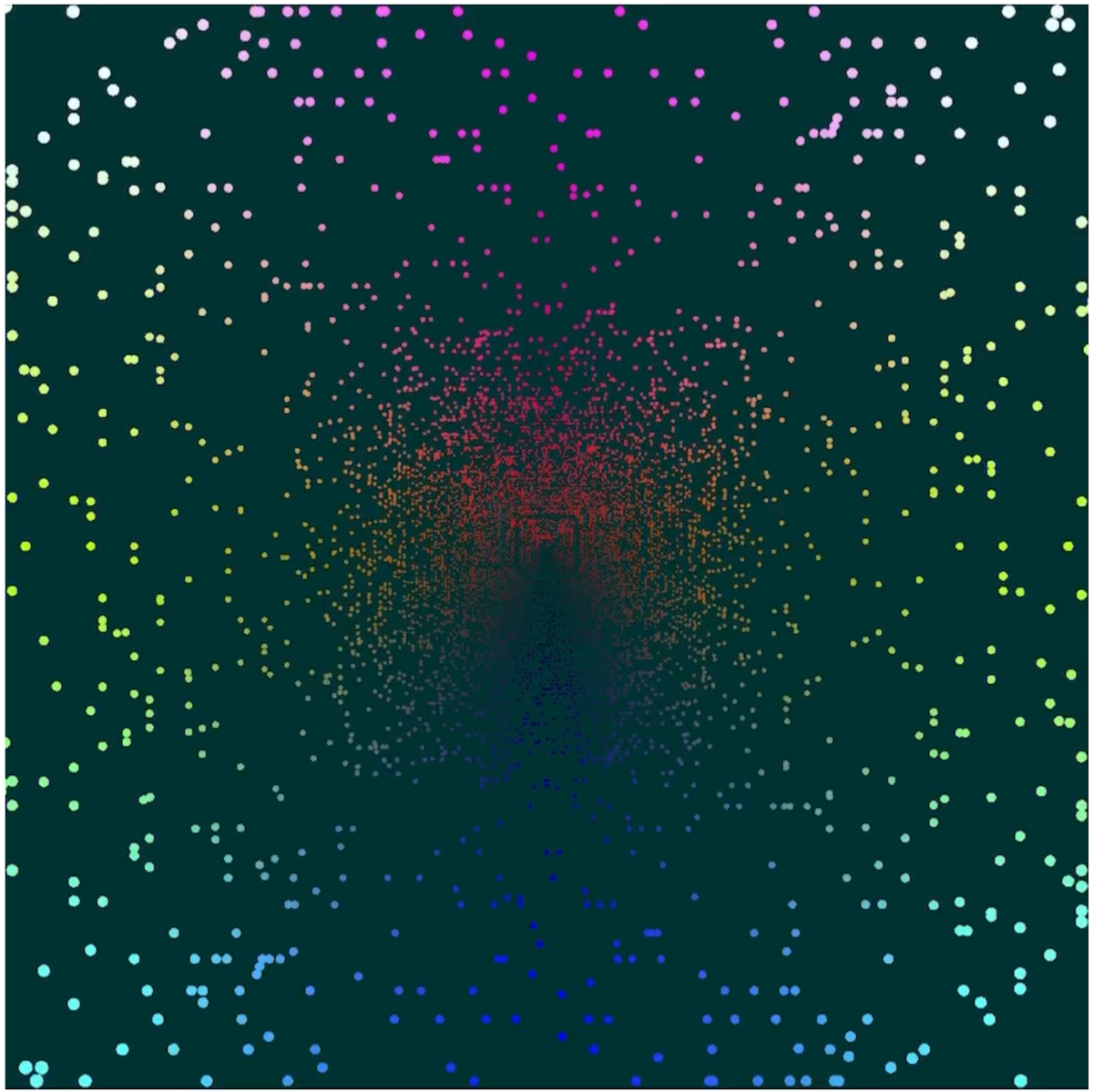
```
kernel void krnFunction(texture2d<float, access::read> in [[ texture(0) ]],
                        texture2d<float, access::write> out [[ texture(1) ]],
                       uint2 gid [[thread_position_in_grid]])
{
   uint2 buf_size = uint2(in.get_width(), in.get_height());
   if (any(gid >= buf_size)) {
      return;
   }

   float4 src = in.read(gid);
   out.write(src * src, gid);
}
```

dispatchThreads



Compute encoder

Particle system update



Metal Performance Shaders

Metal Performance Shaders

Tasks

- Image processing (filters)
- Neural networks
  - Layers
  - CNN
  - RNN
- Matrices and vectors
- Ray tracing

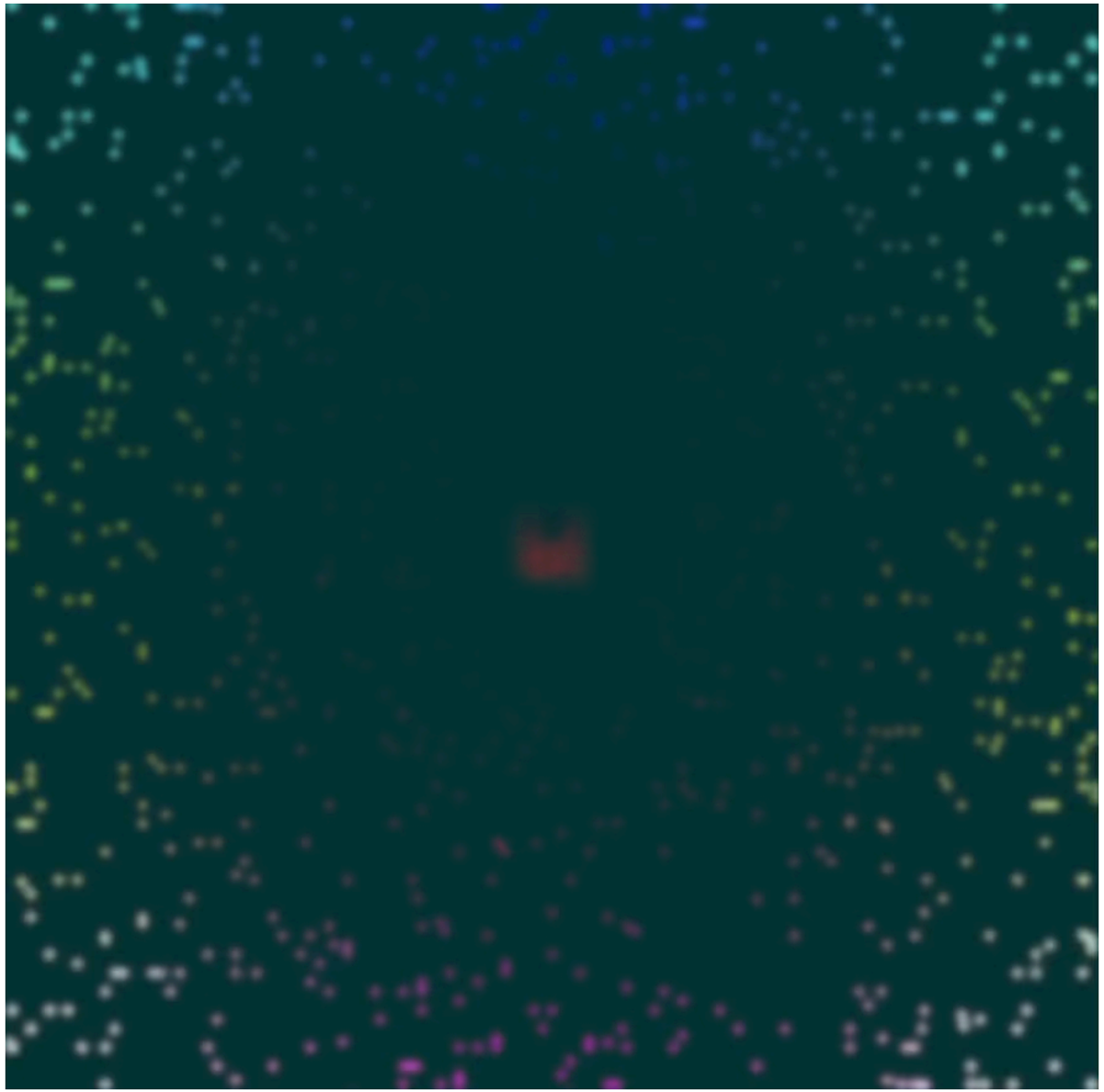
Metal Performance Shaders

Blur example

```
import MetalPerformanceShaders

// ...
var mpsGaussianBlur: MPSImageGaussianBlur
mpsGaussianBlur = MPSImageGaussianBlur(device: device, sigma: 5)

// ...
mpsGaussianBlur.encode(commandBuffer: commandBuffer,
                       inPlaceTexture: &outTexture!)
```

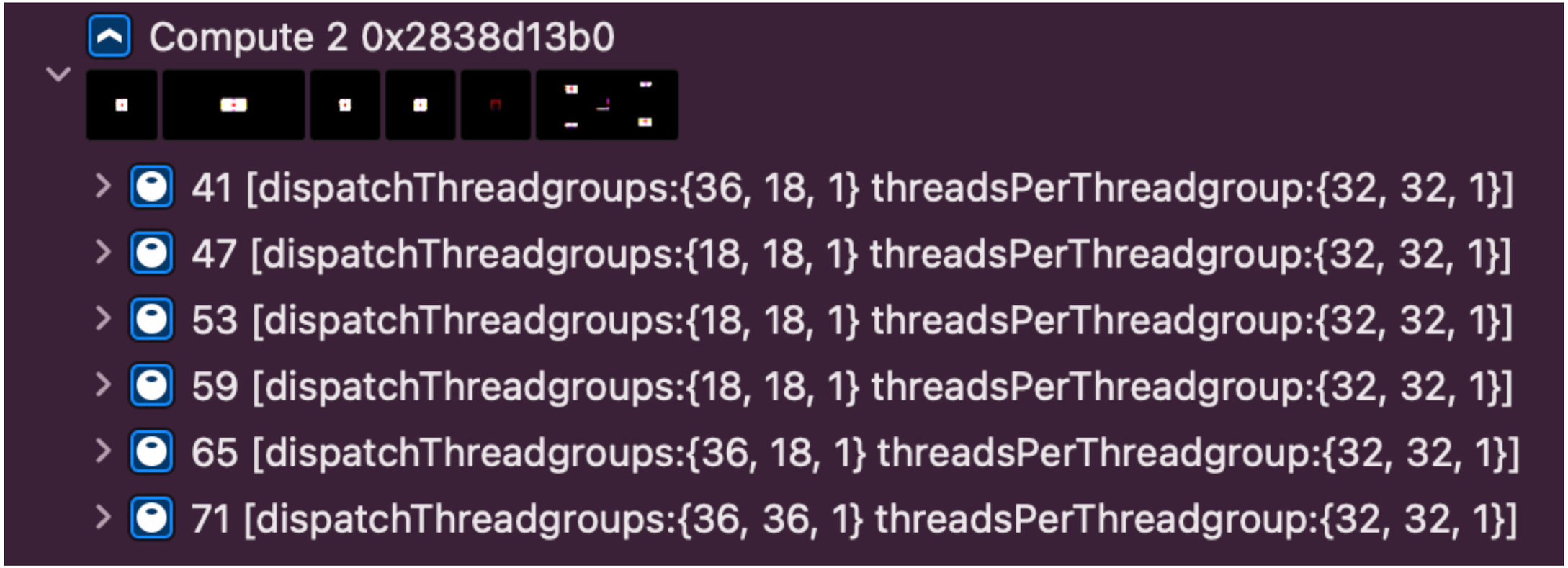
MPS

Blur on particles



Metal Performance Shaders

Under Hood



D2F6V

D2F6H

F13V

F13H

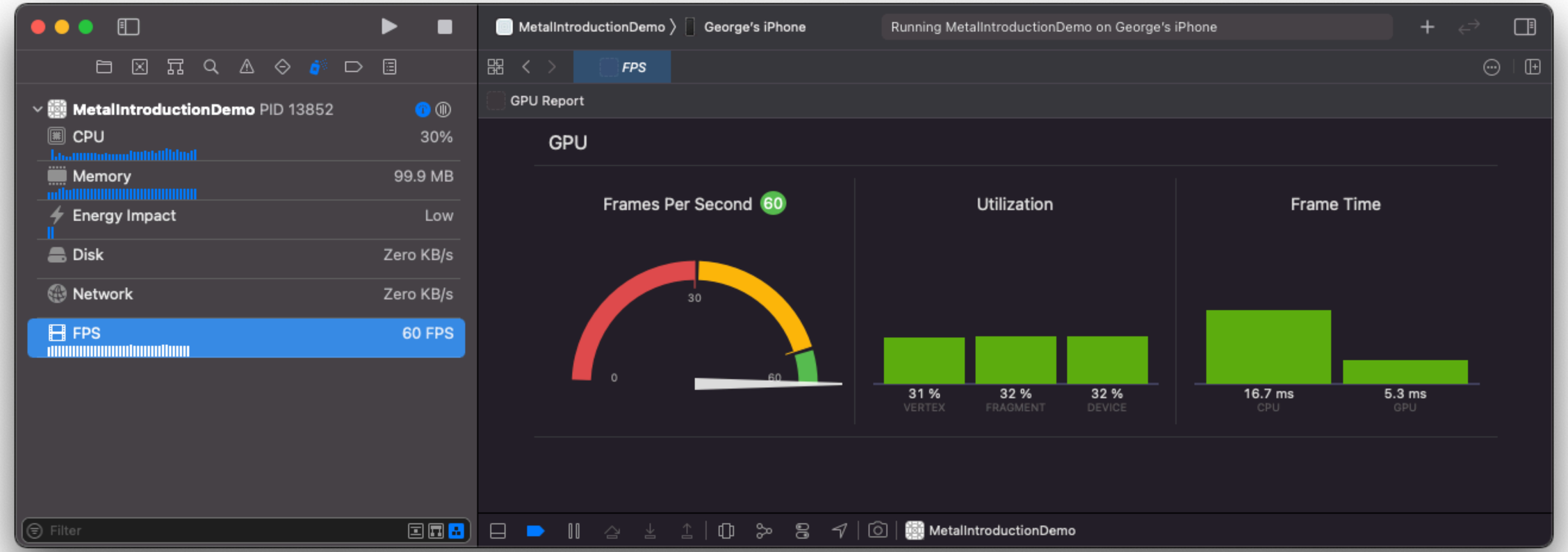
U2P3F3H

U2P3F3V

Tools

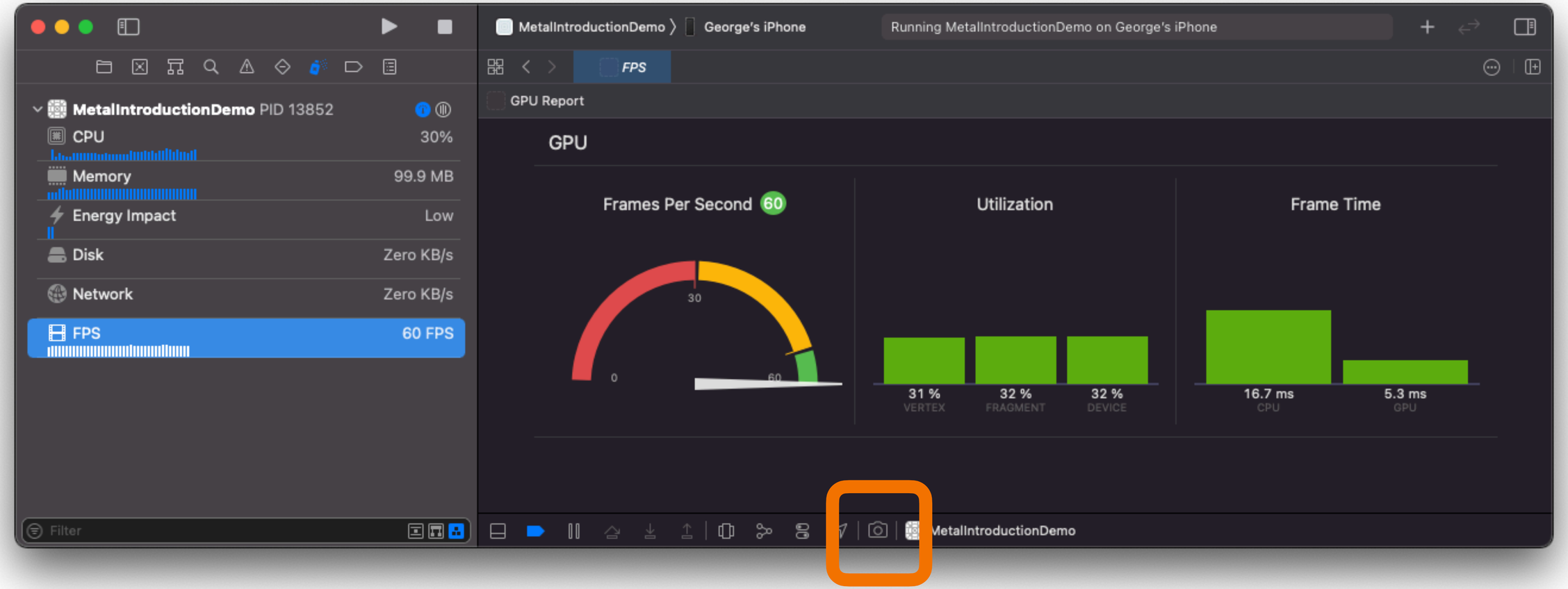
Tools

Xcode


MetalIntroductionDemo
PID 13852
CPU
30%
Memory
99.9 MB
Energy Impact
Low
Disk
Zero KB/s
Network
Zero KB/s
FPS
60 FPS
Filter
MetalIntroductionDemo
George's iPhone
Running MetalIntroductionDemo on George's iPhone
FPS
GPU Report
GPU
Frames Per Second
60
30
0
60
Utilization
31 %
VERTEX
32 %
FRAGMENT
32 %
DEVICE
Frame Time
16.7 ms
CPU
5.3 ms
GPU
MetalIntroductionDemo

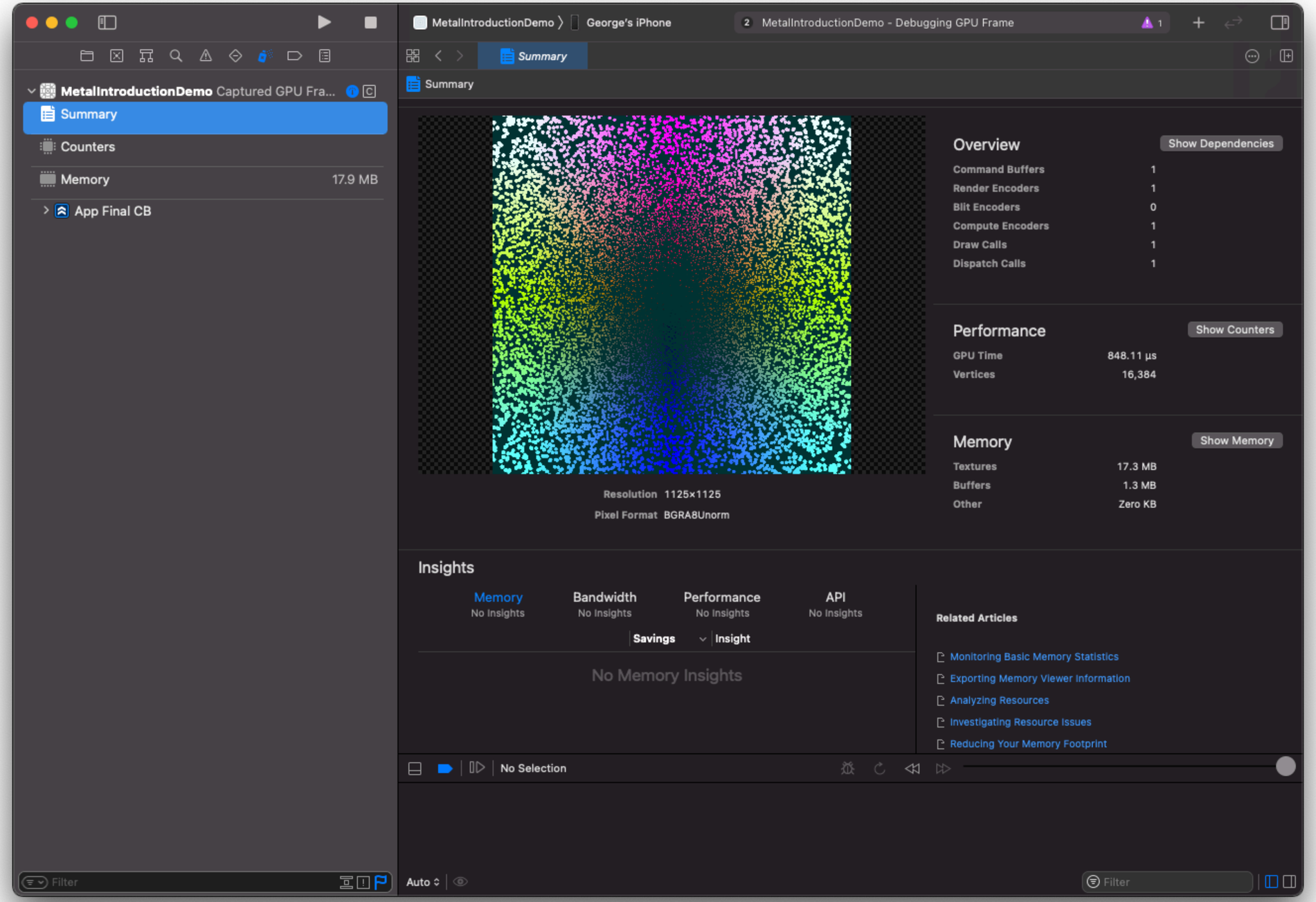
Tools

Xcode Frame Capture



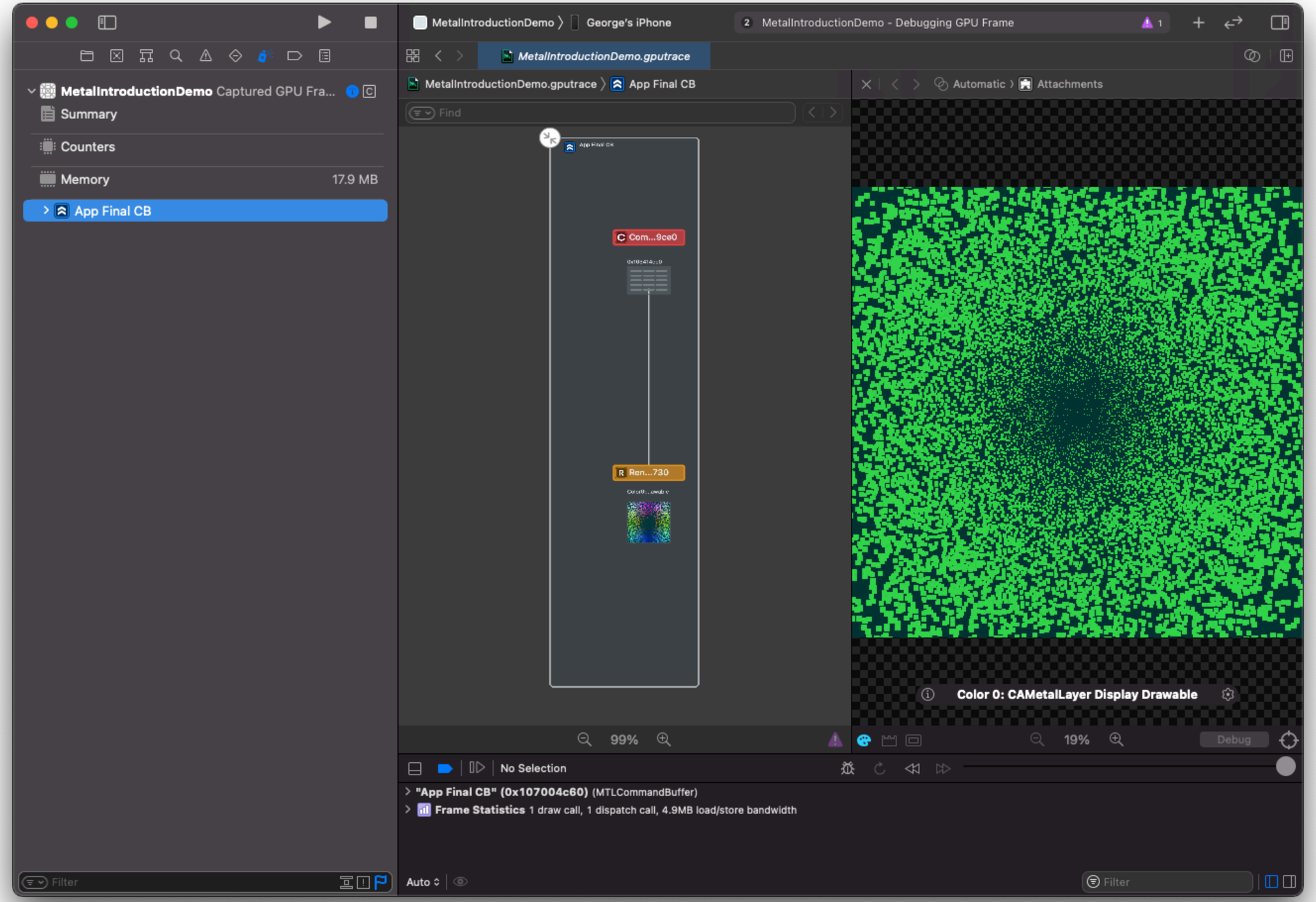
Tools

Xcode
Frame
Capture



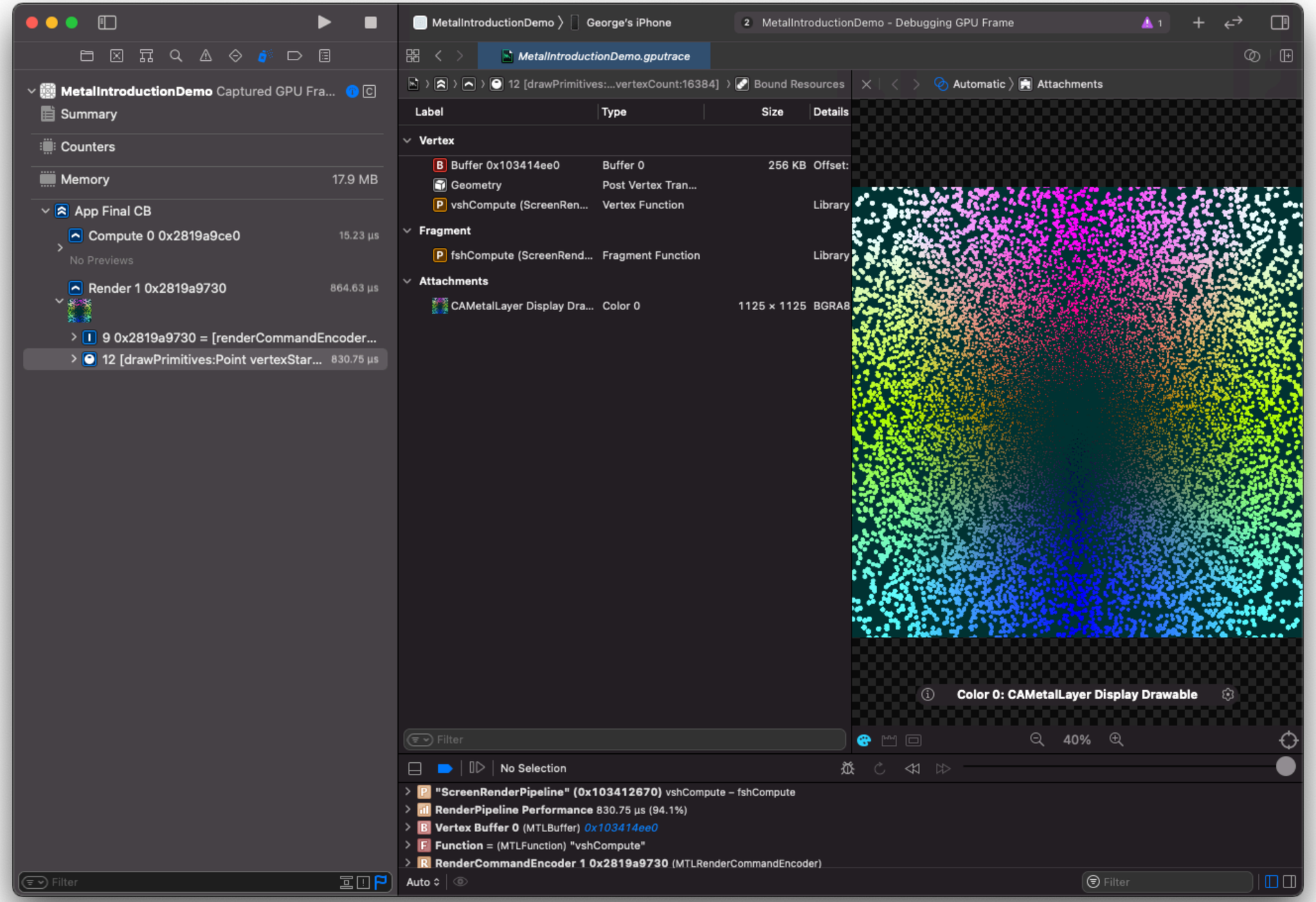
Tools

Xcode
Frame
Capture



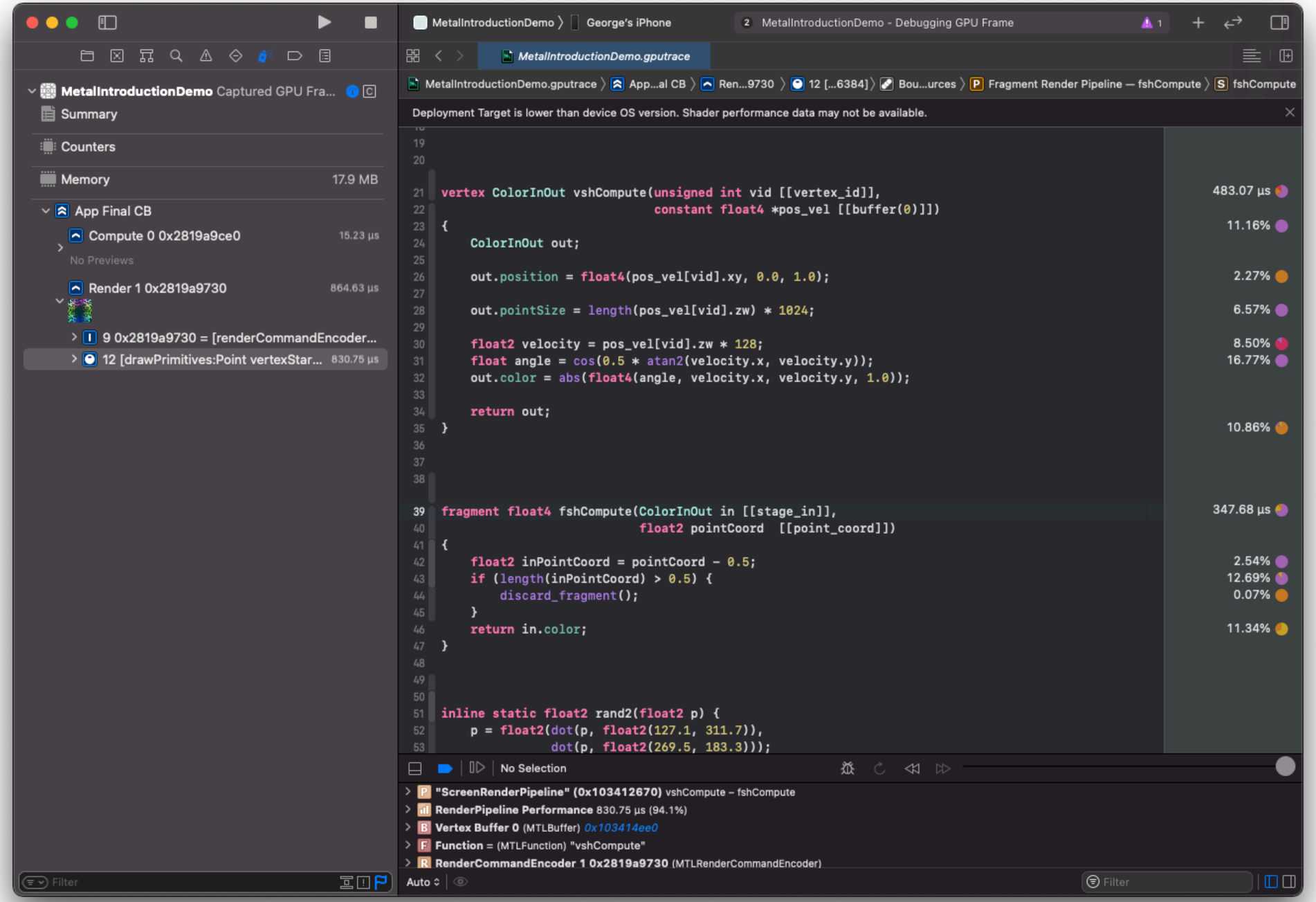
Tools

Xcode
Frame
Capture



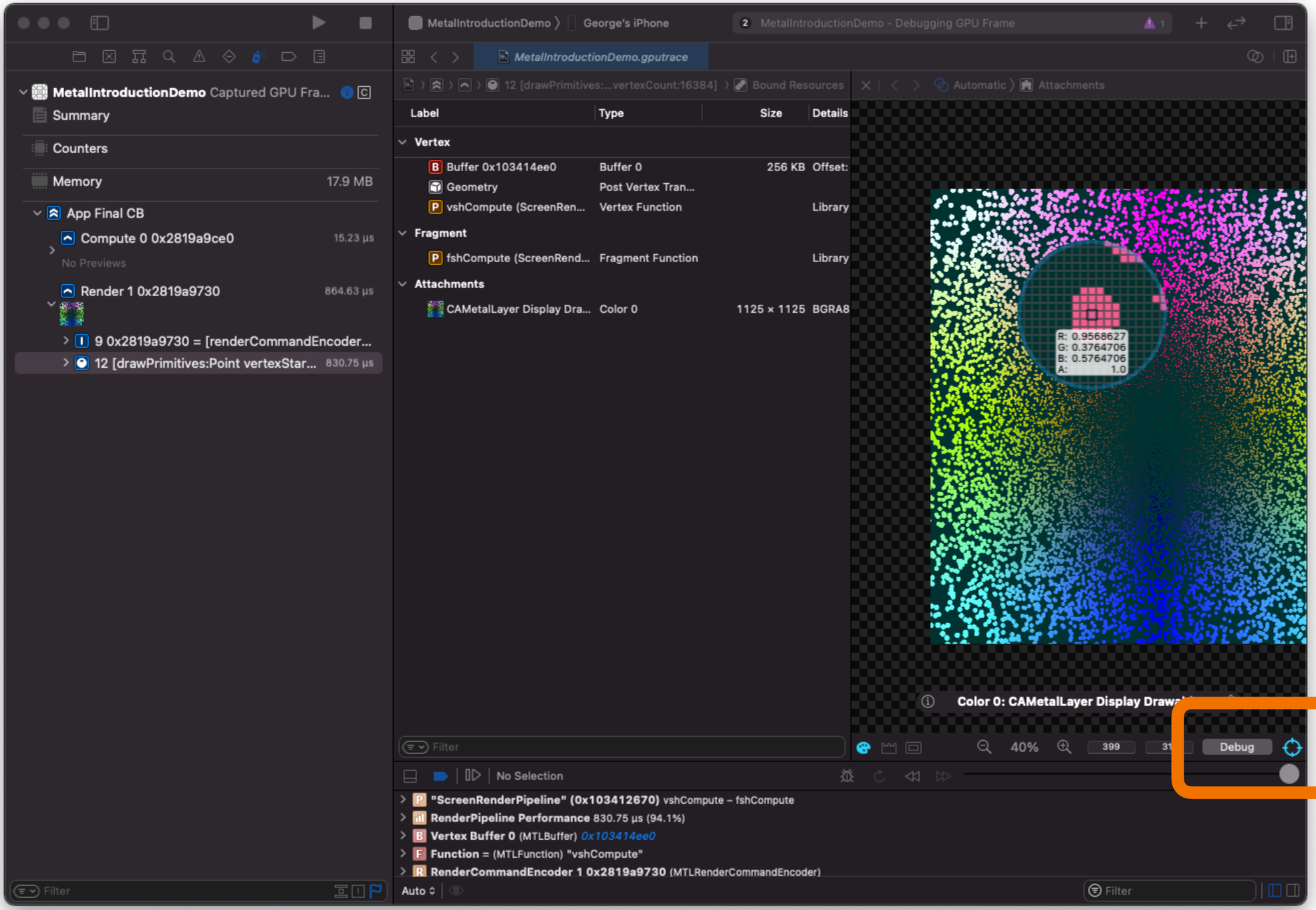
Tools

Xcode
Frame
Capture



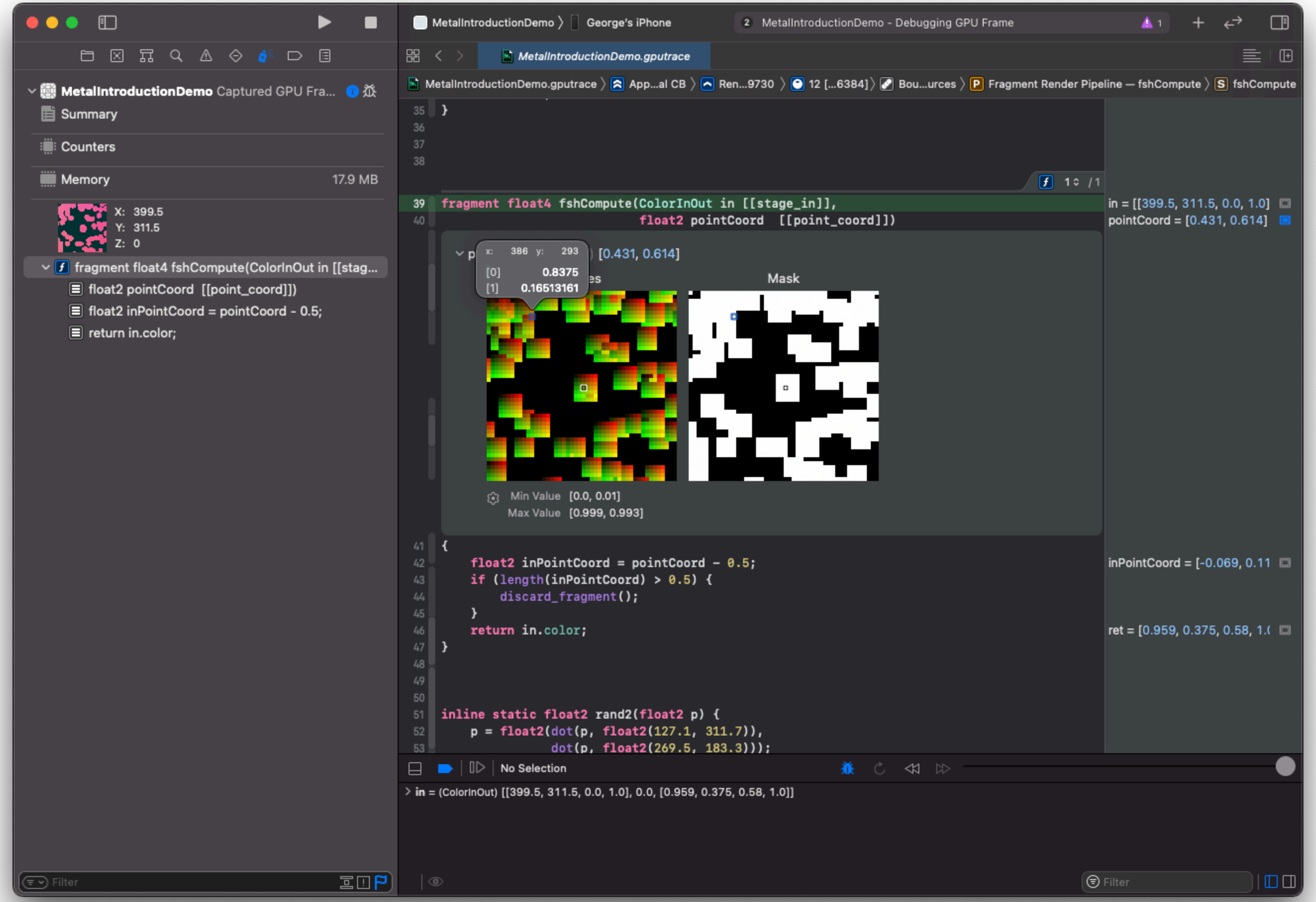
Tools

Xcode
Frame
Capture



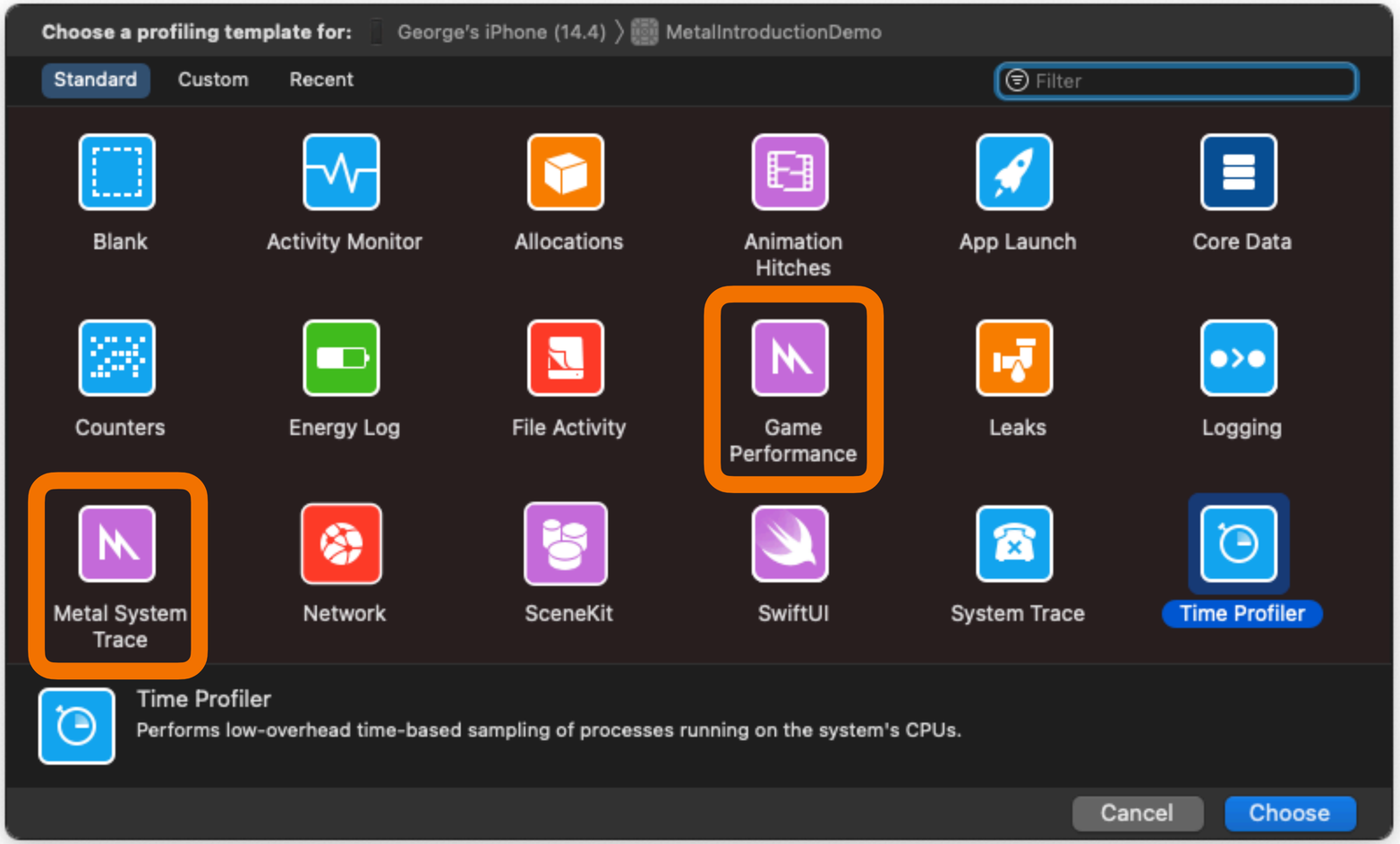
Tools

Xcode
Frame
Capture



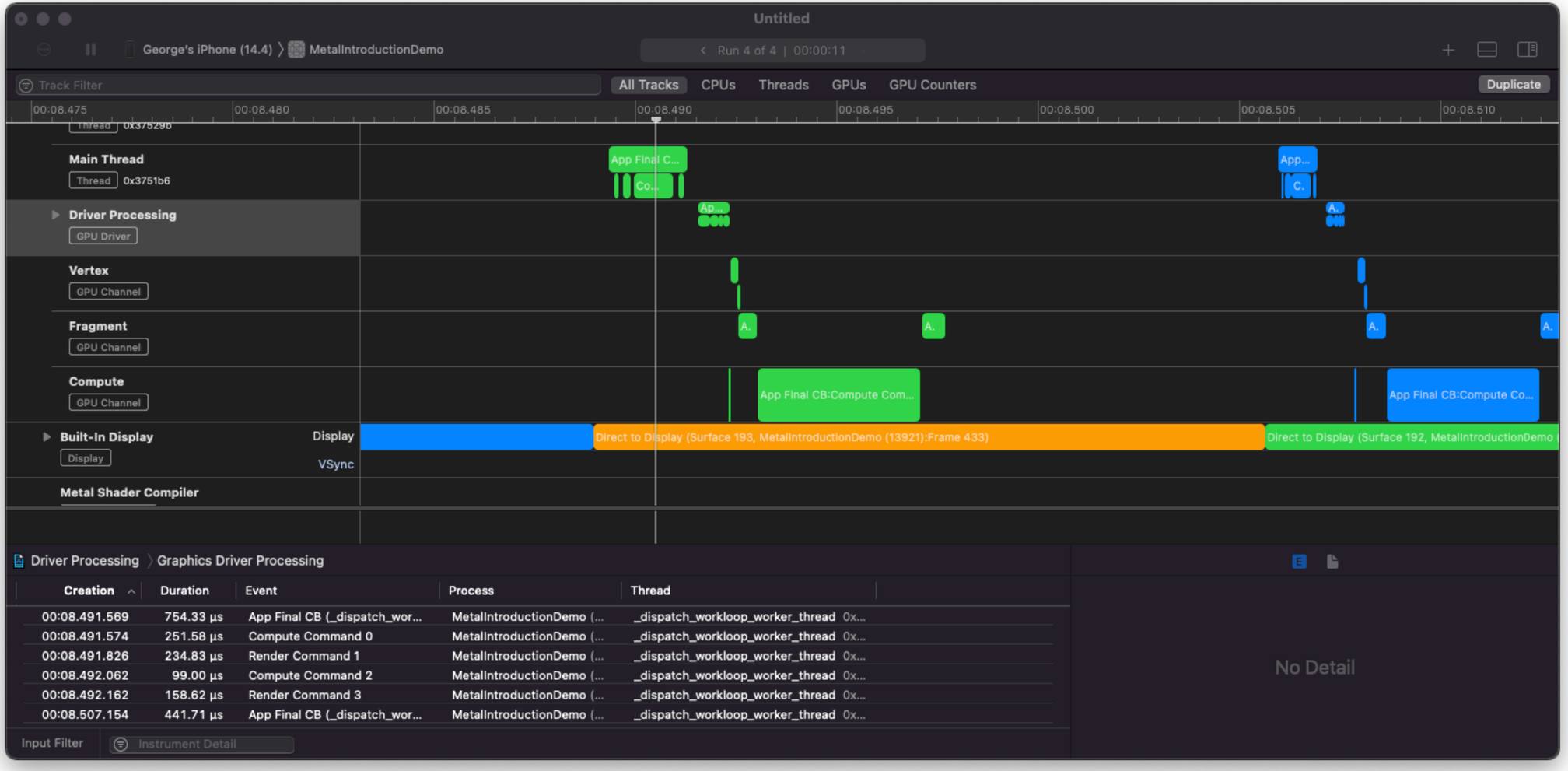
Tools

Xcode Profiling Tools



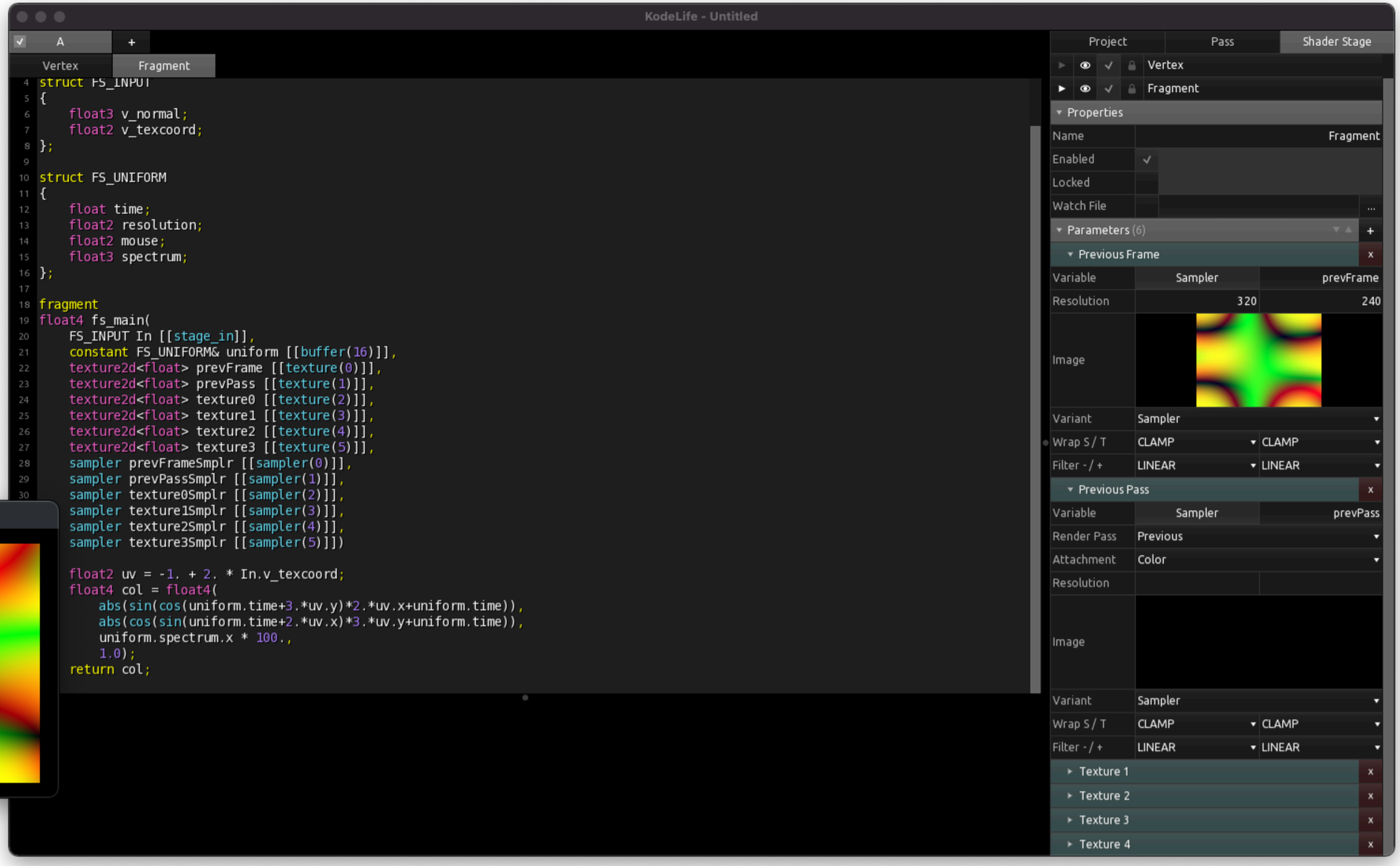
Tools

Xcode Profiling Tools



Tools

KodeLife



Links

- **Metal Shading Language Specification**
- **Metal Feature Set Tables**
- Metal Programming Guide
- Metal Best Practices Guide
- Debug Metal

- Demo app

Contacts

wdfeffects@gmail.com

@GOstrobrod

@GOstrobrod

https://wdfteam.gitlab.io