

Apple Metal

introduction

George Ostrobrod

1.



1. Рисуем кружочки

What is Metal

What is Metal

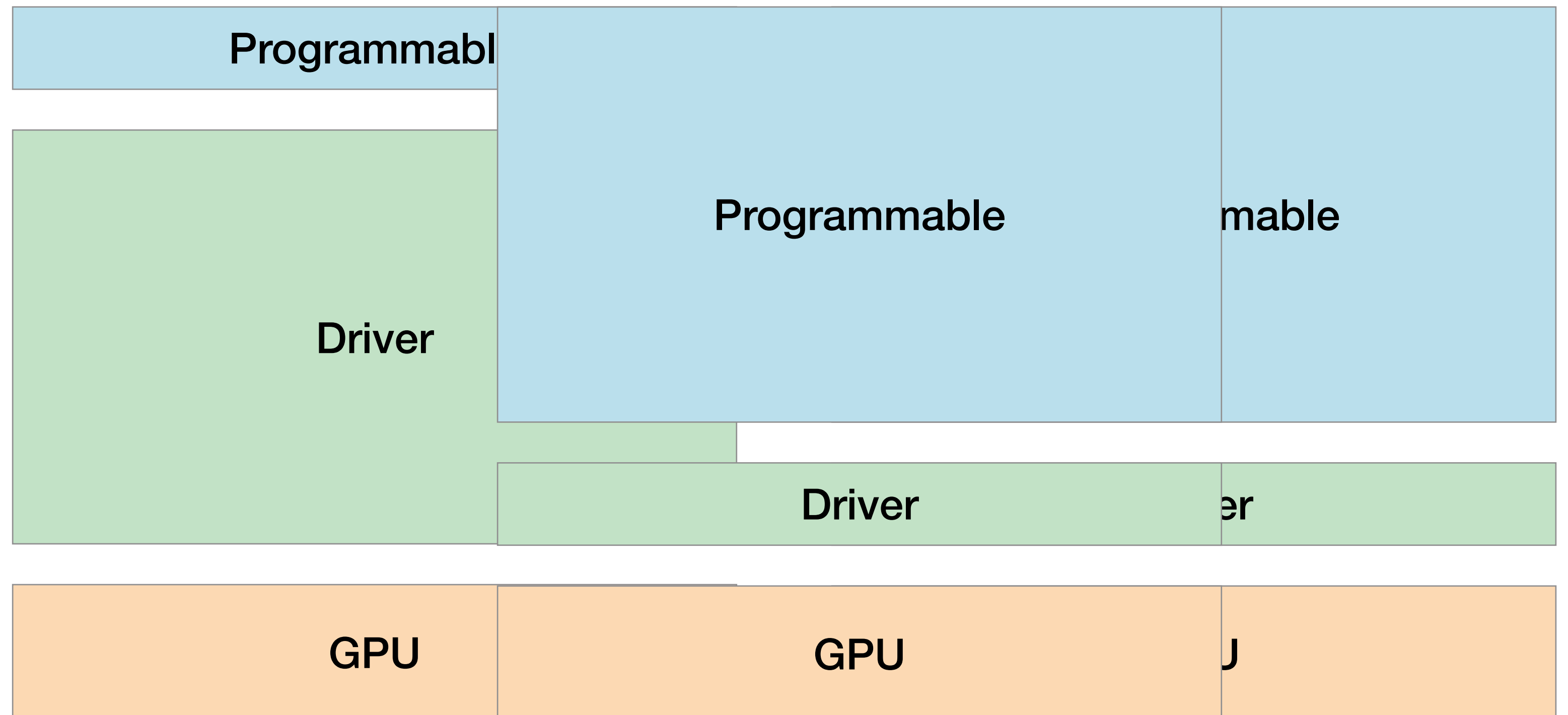
Drawing & Rendering Frameworks

- CoreGraphics
 - + vImage
- CoreImage (+CIKL)
- SpriteKit
- SceneKit



What is Metal

Metal vs OpenGL vs Vulkan

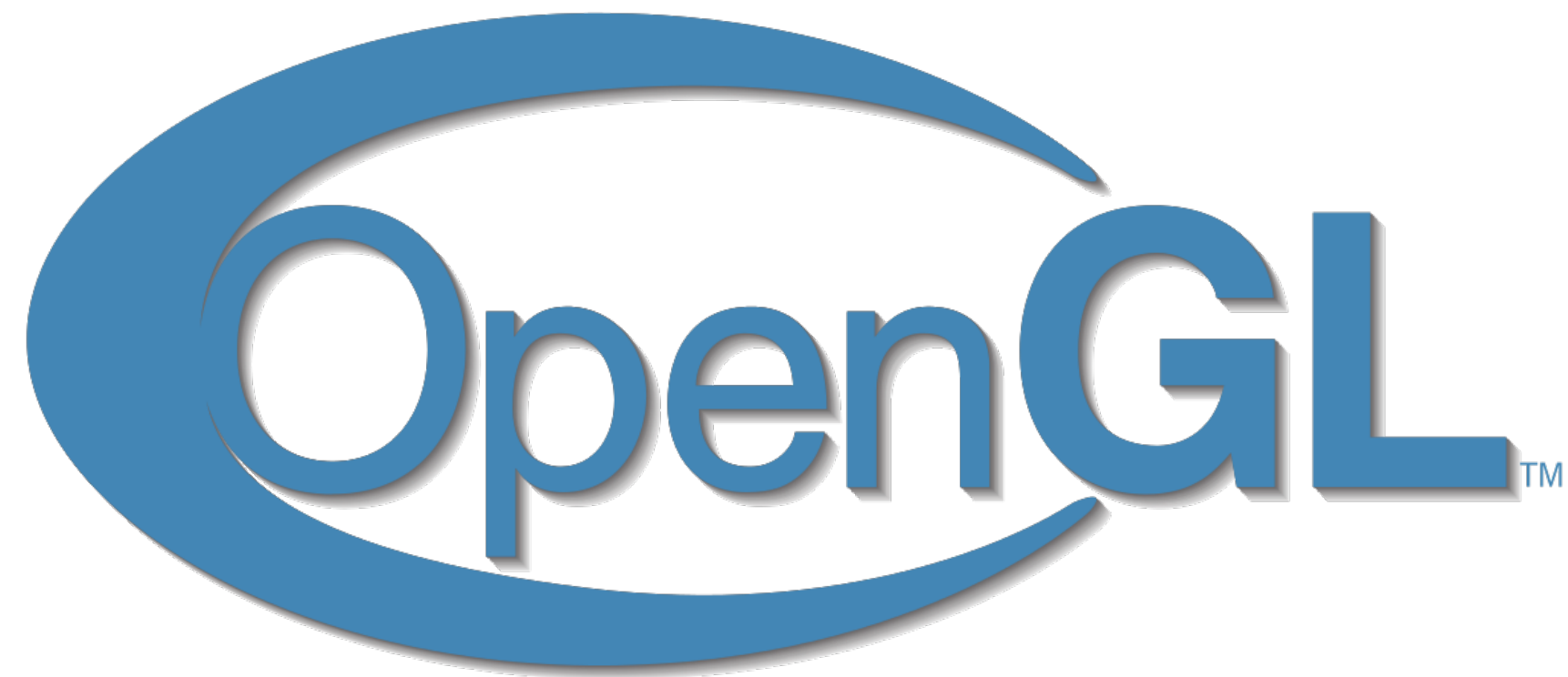


What is Metal

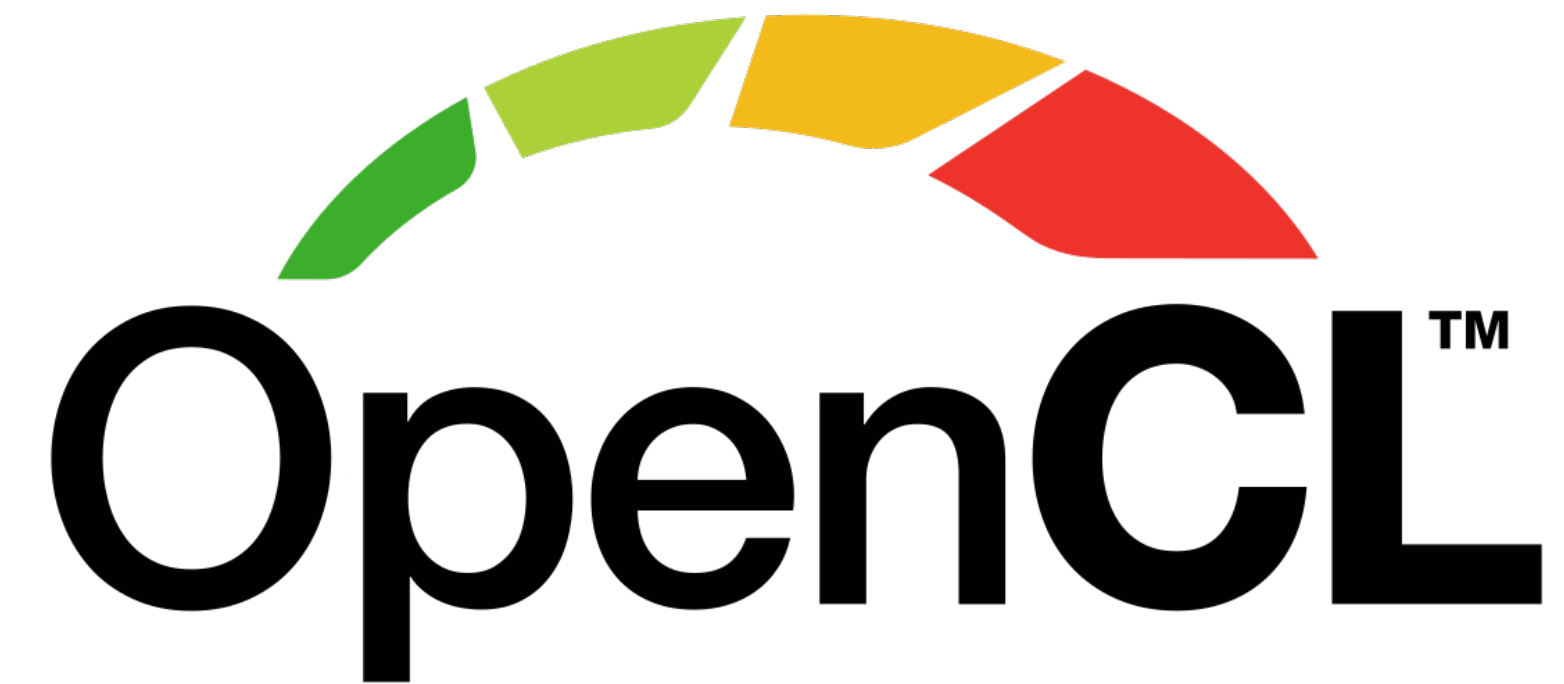
Metal vs OpenGL + OpenCL



≈



+



What is Metal

Typical Tasks

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

MetalKit

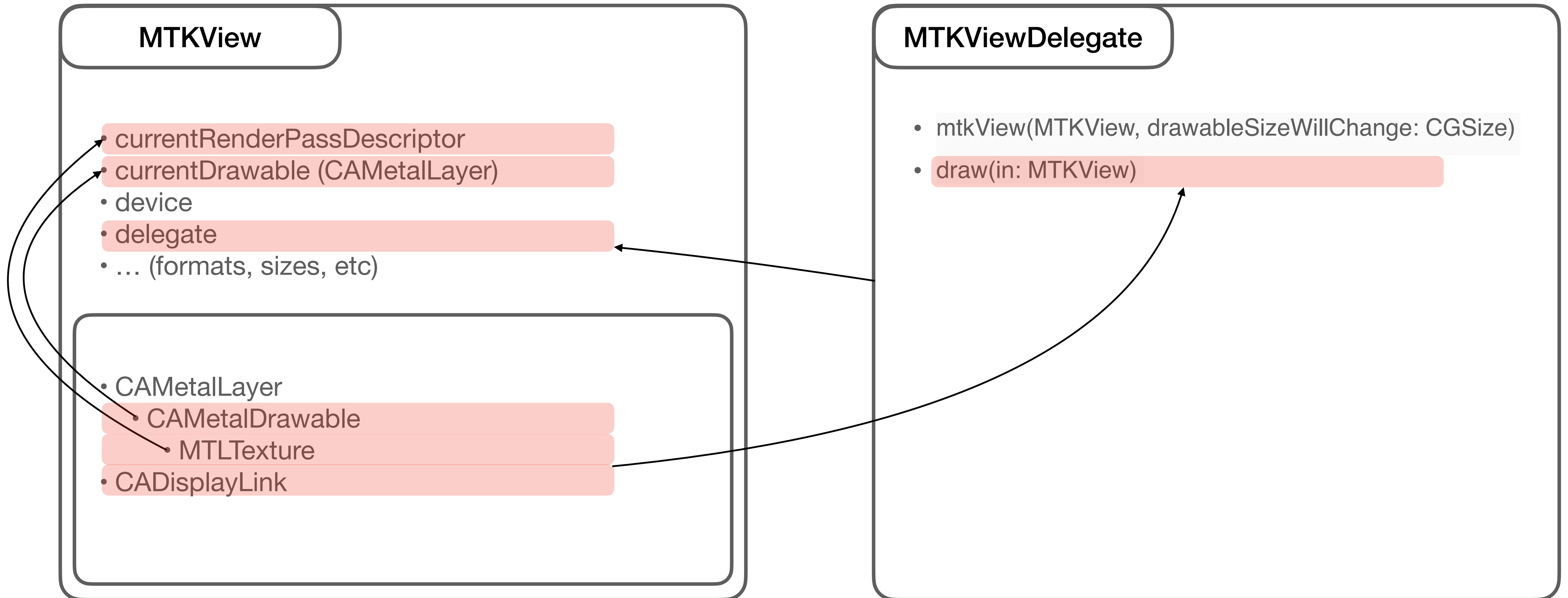
MetalKit

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

А Metal это просто?



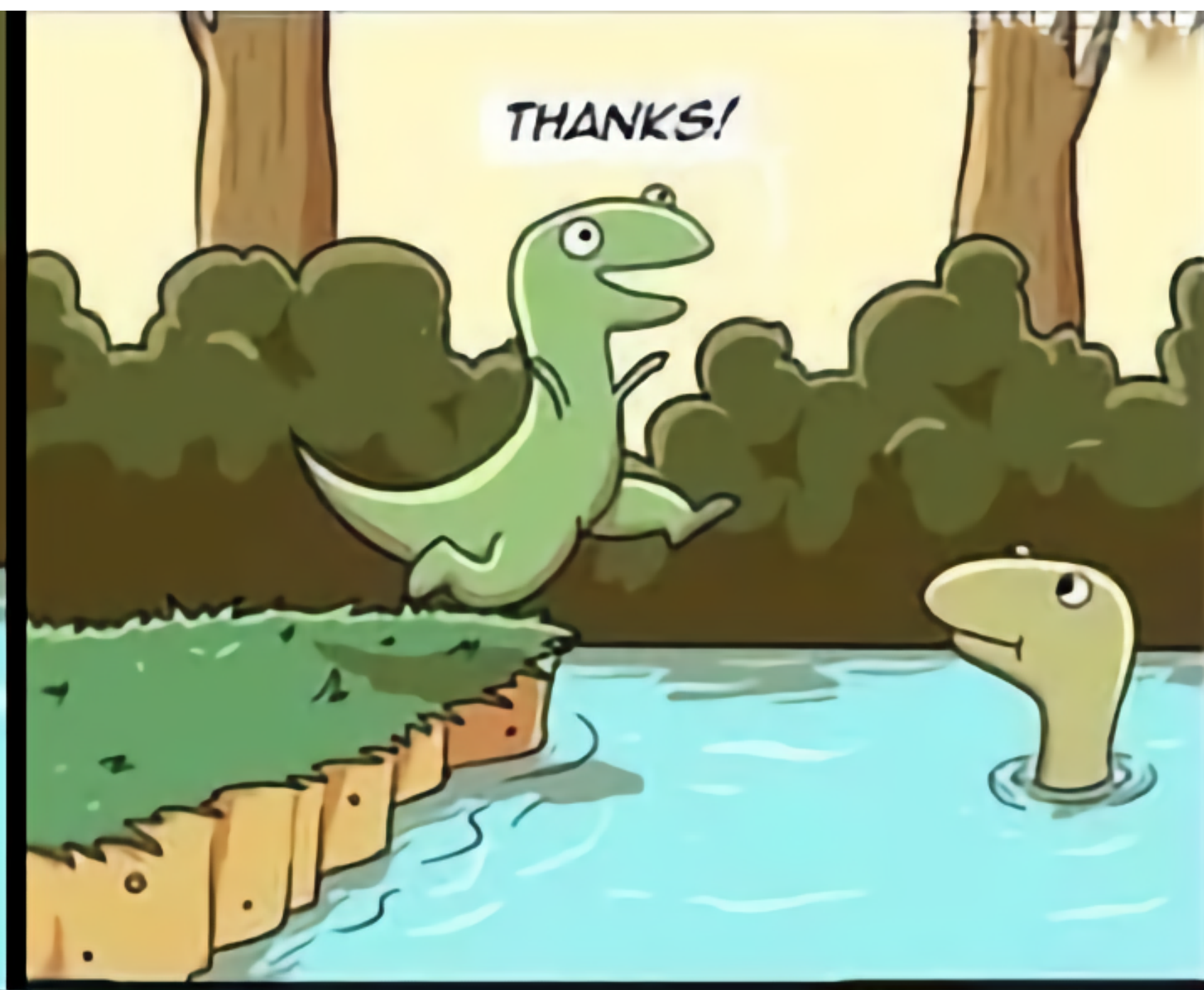
Да, конечно!

Command Buffer

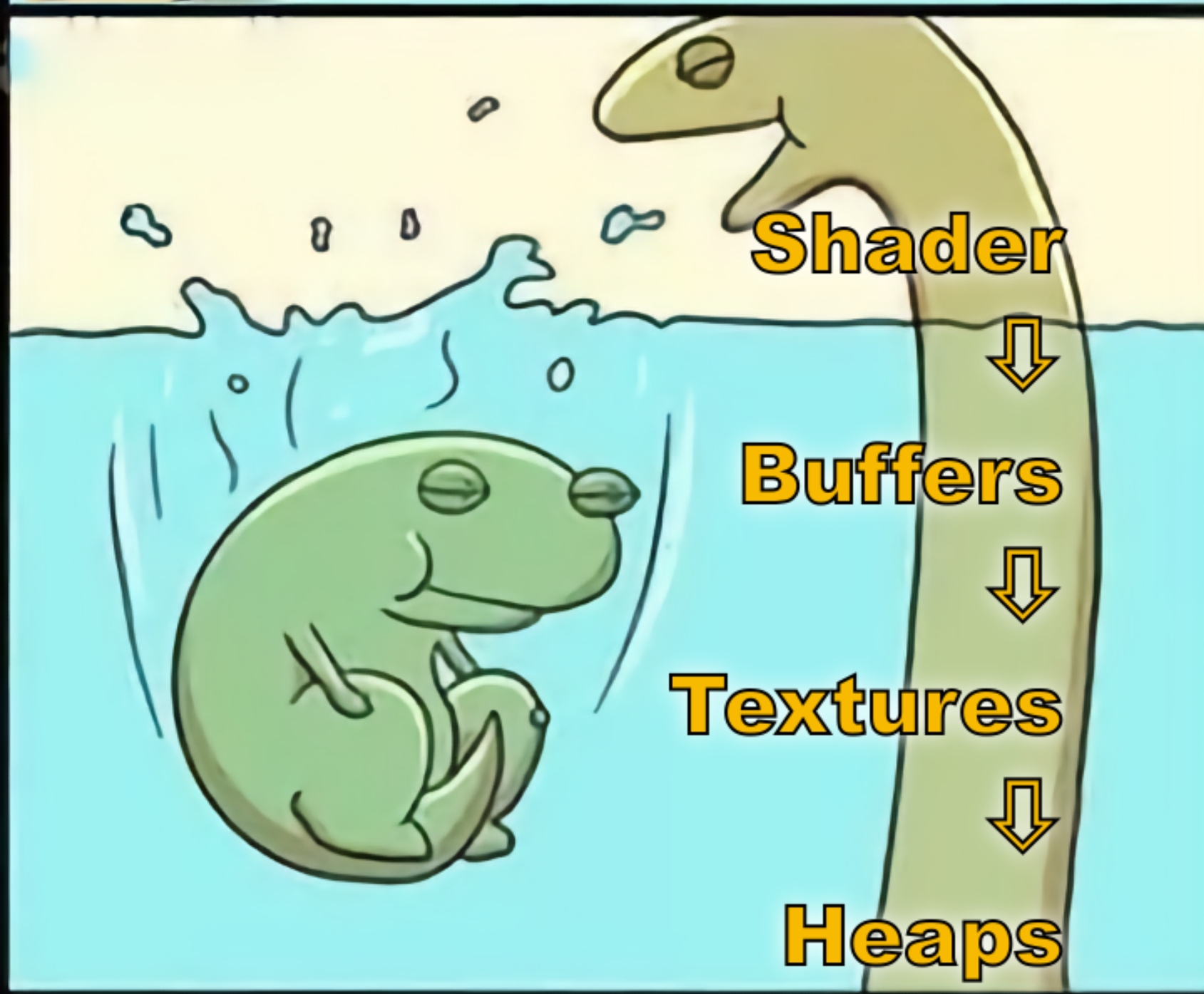
Encoder

Draw

Shader



THANKS!

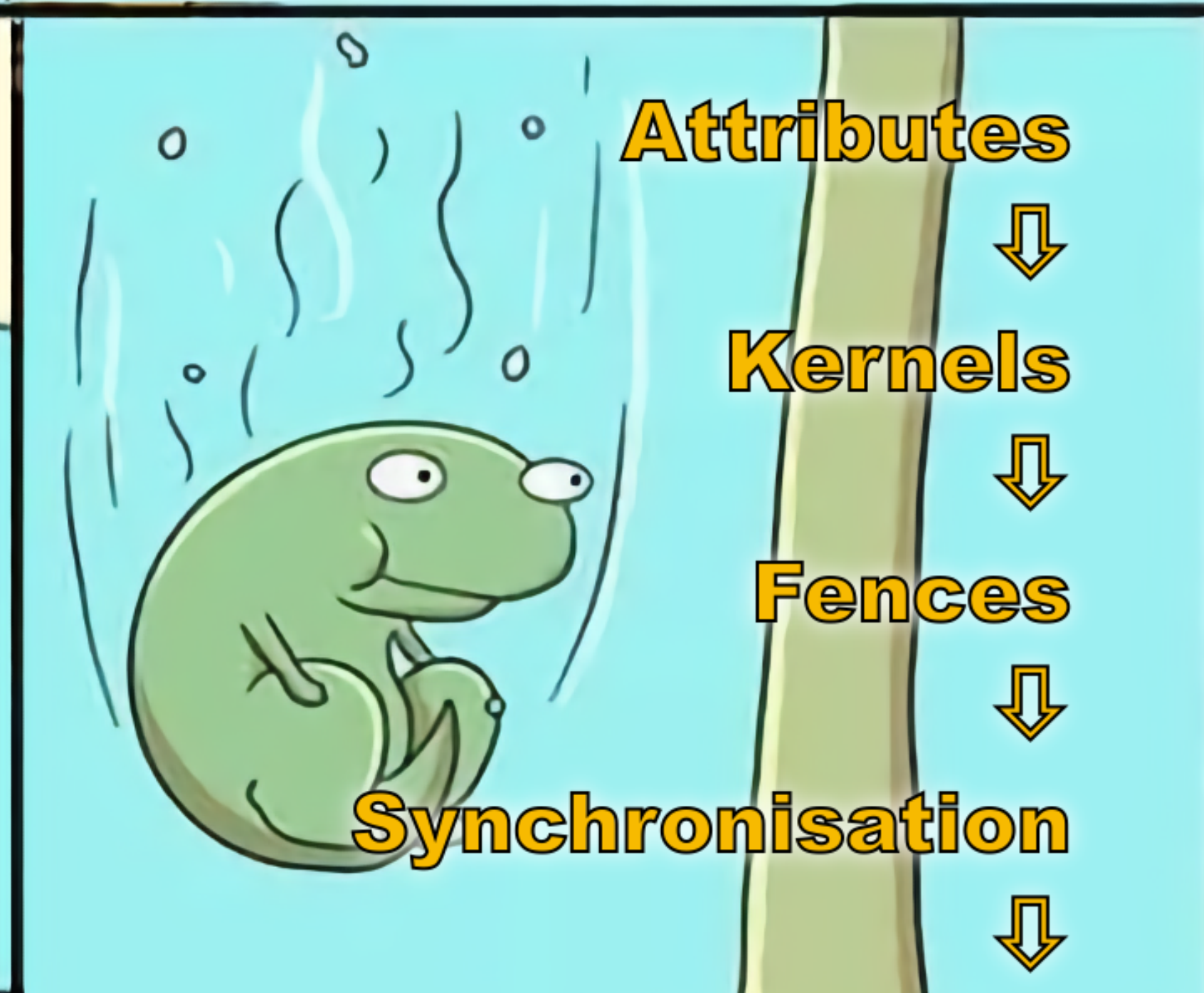


Shader

Buffers

Textures

Heaps



Attributes

Kernels

Fences

Synchronisation



As Crônicas de Wesley

Metal Architecture

Queue



Command Buffer



Encoder



Metal Architecture

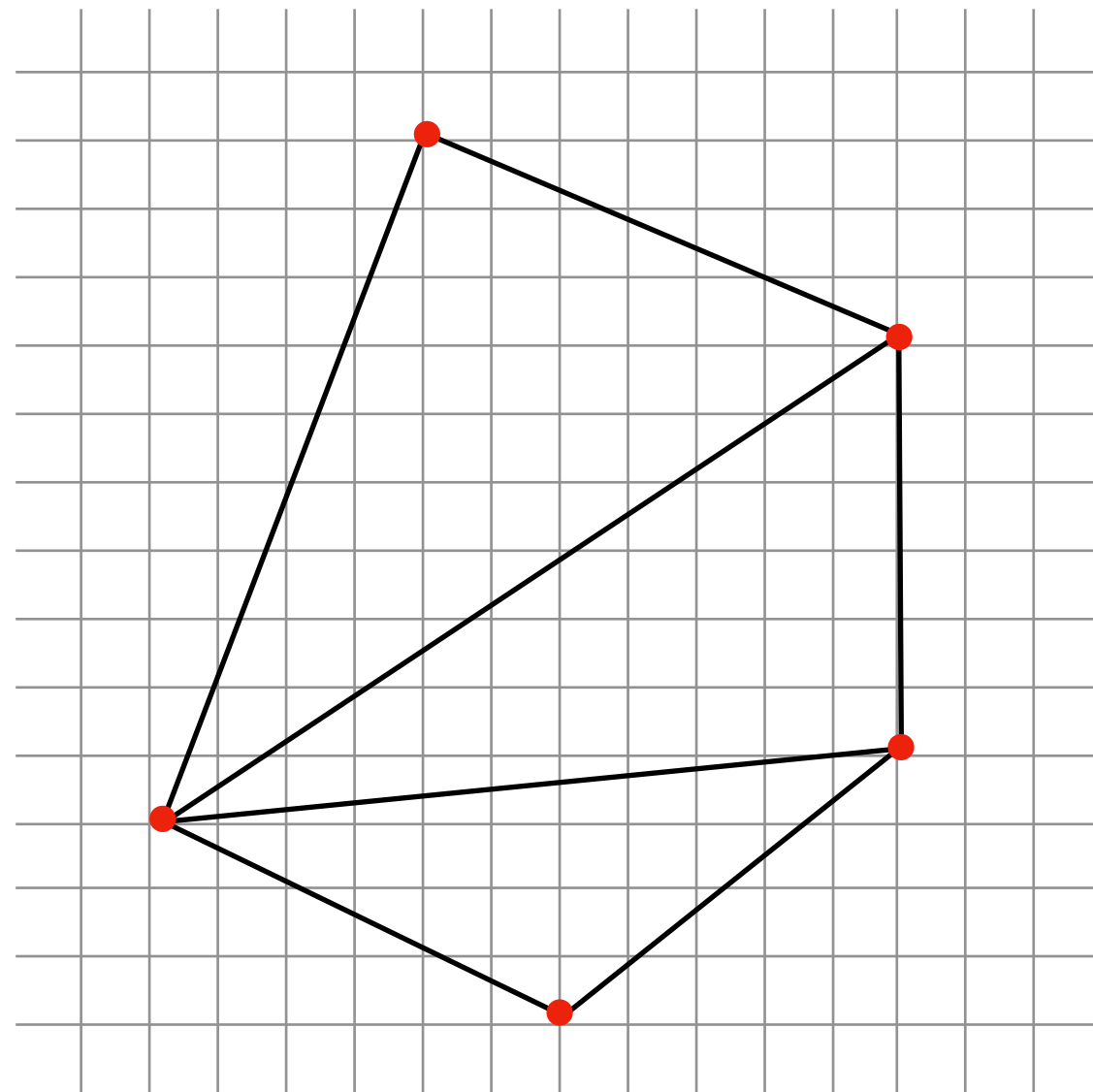
Draw calls



Shaders

Shaders

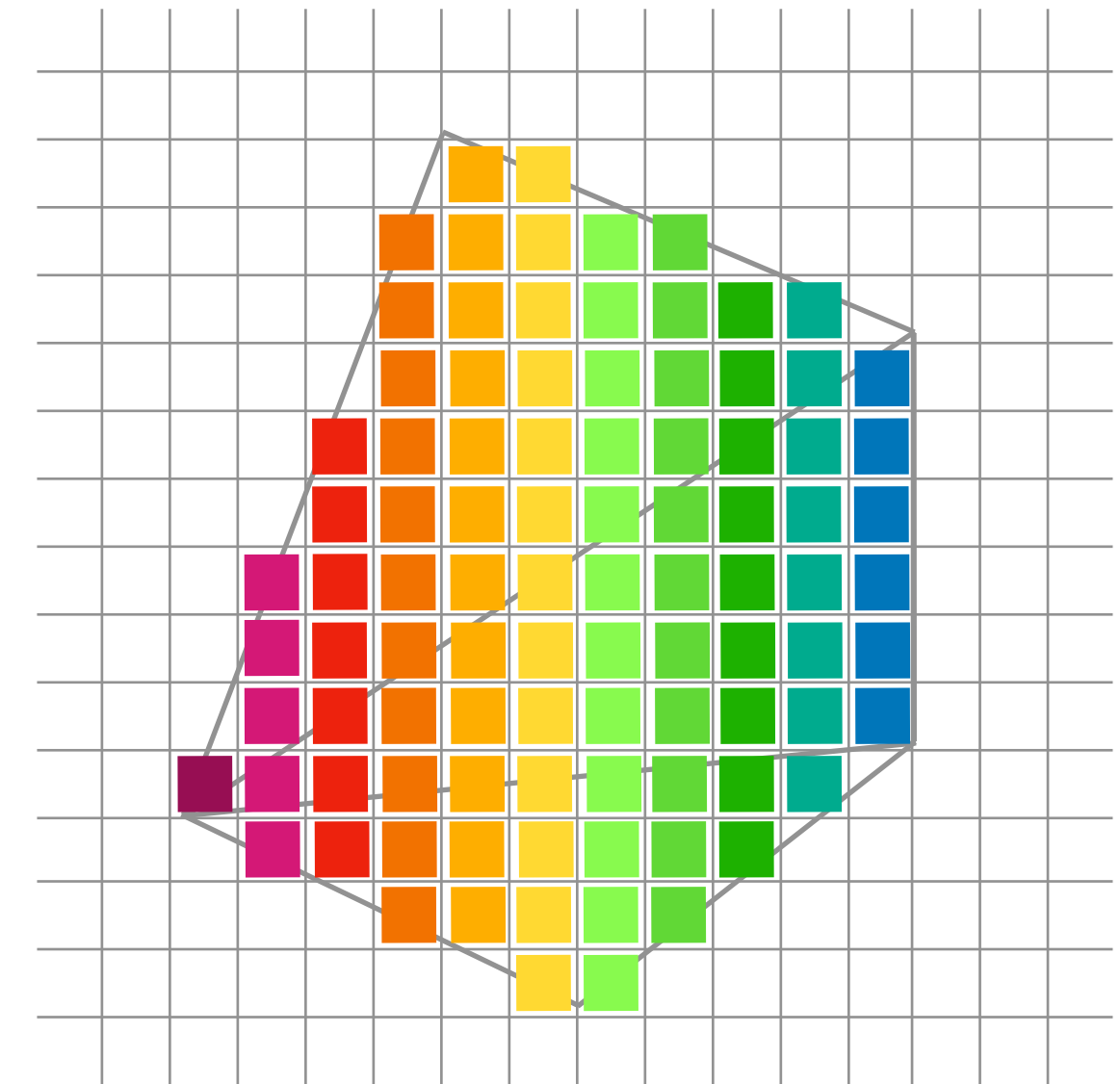
Vertex



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

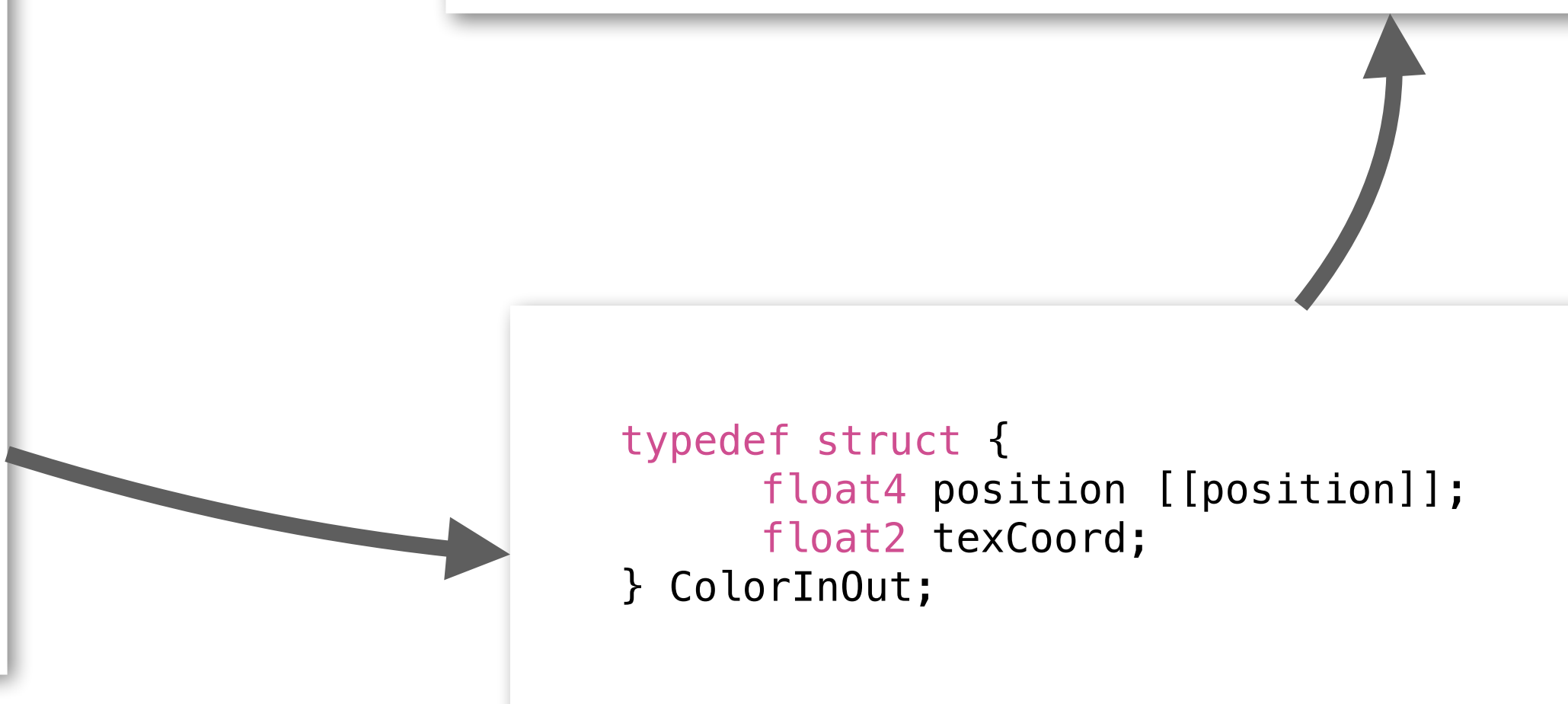
Vertex

```
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;  
}
```

Fragment

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

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



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)	float or float[n] n must be known at compile time	Distance from the origin of the primitive to the color attachment. The index <i>m</i> indicates which color attachment to read from.
front_facing	bool	This value is true if the fragment is front-facing.
amplification_count	ushort or uint	The base instance value added to each instance identifier before reading per-instance data.
base_instance	ushort or uint	The per-instance identifier, which includes the base instance value if one is specified.
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.
vertex_id	ushort or uint	The per-vertex identifier, which includes the base vertex value if one is specified.
viewport_array_mask	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

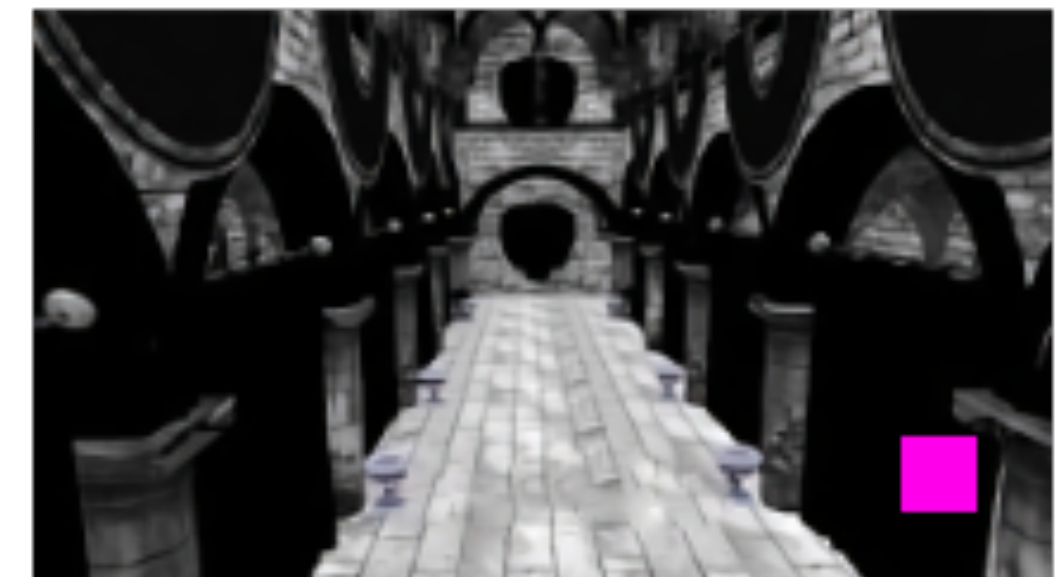
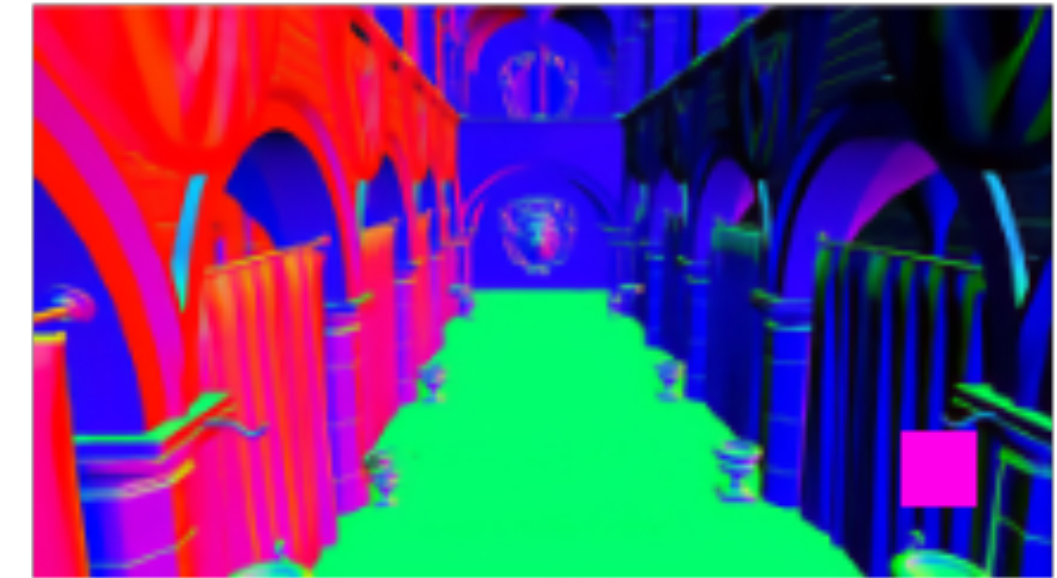
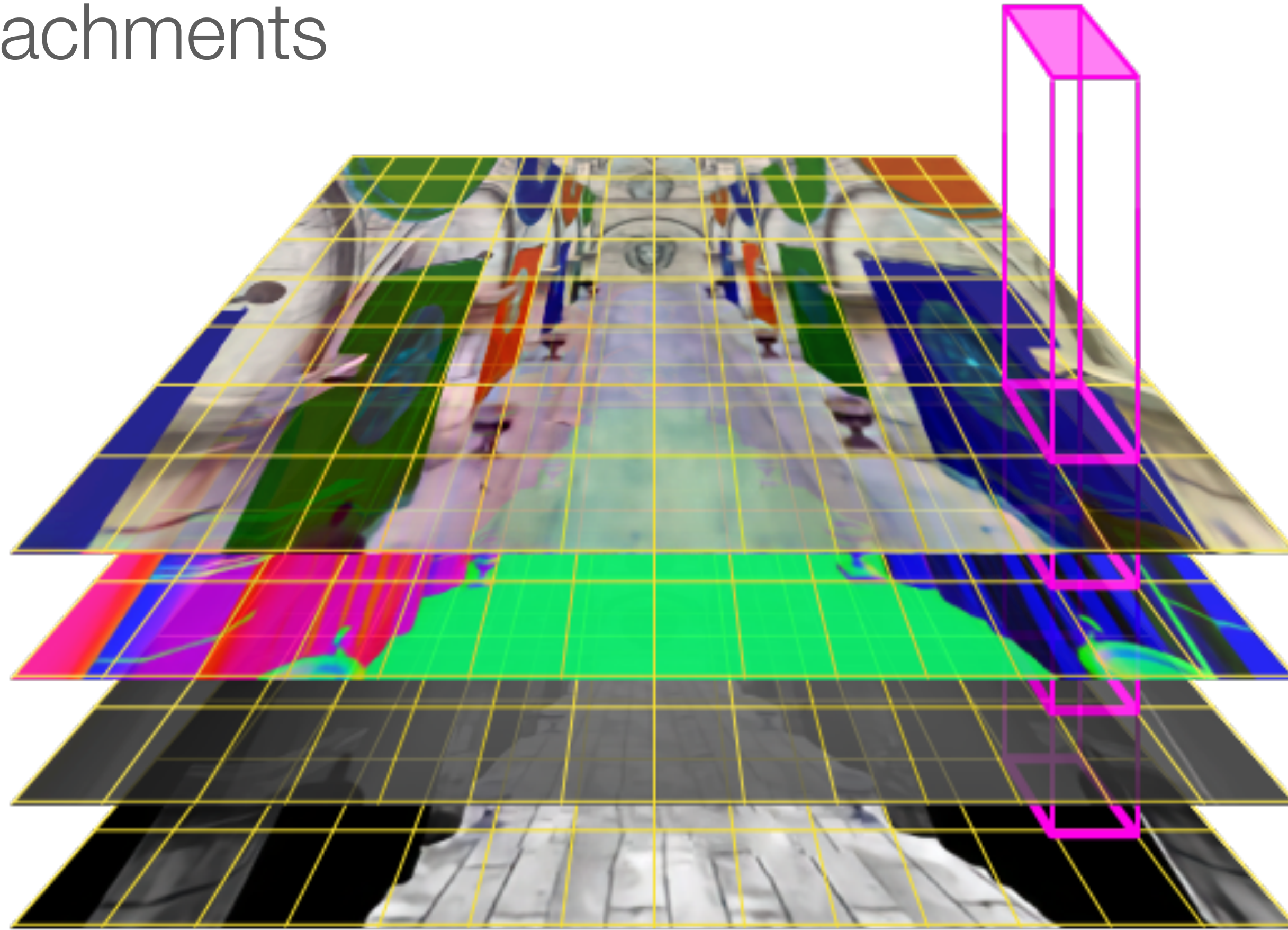
if let renderEncoder = commandBuffer.makeRenderCommandEncoder(descriptor: renderEncoderDescr) {
    renderEncoder.setRenderPipelineState(renderPipeline)

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

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

Render Encoder

Attachments



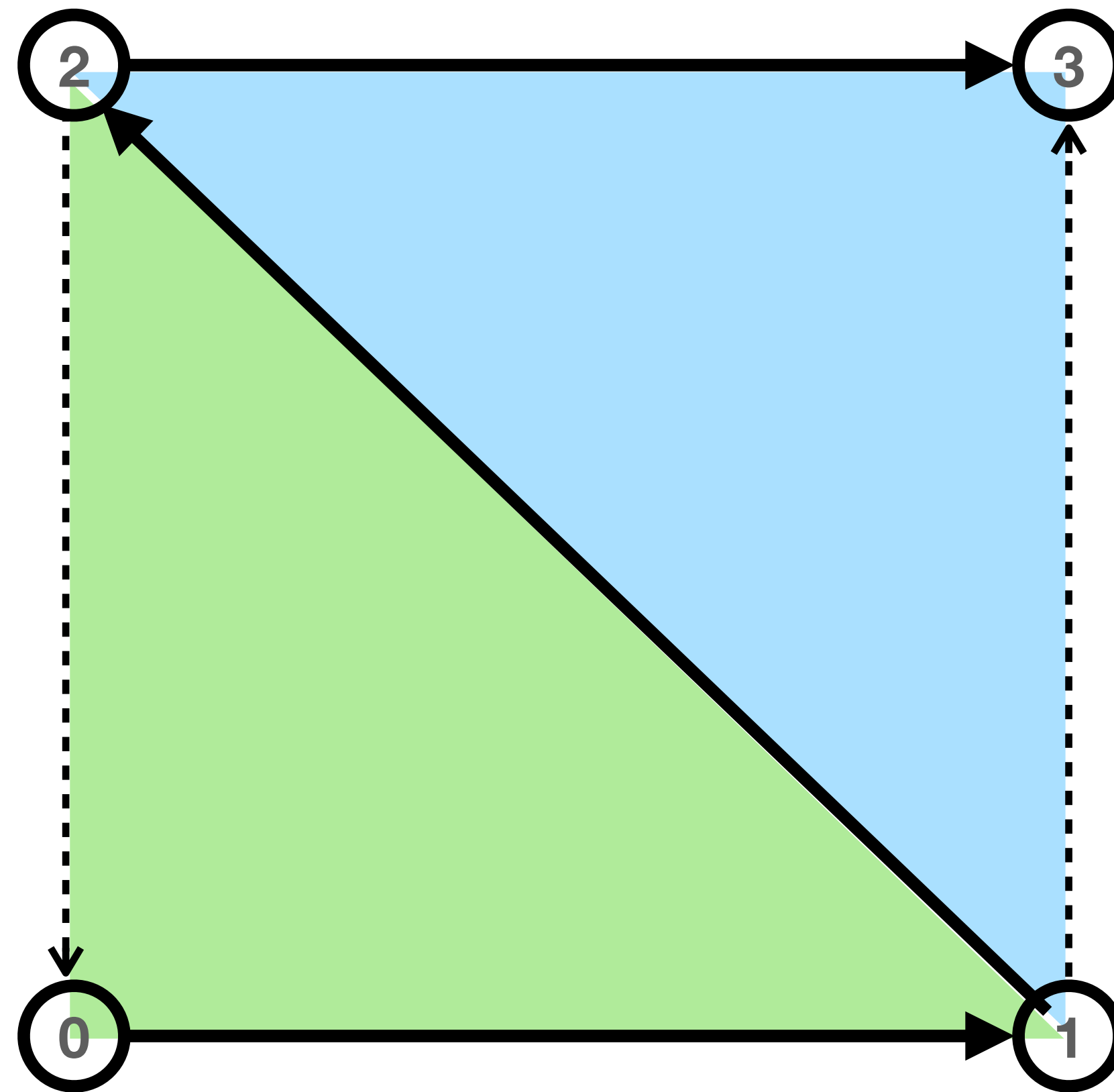
Render Encoder

Draw Call

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

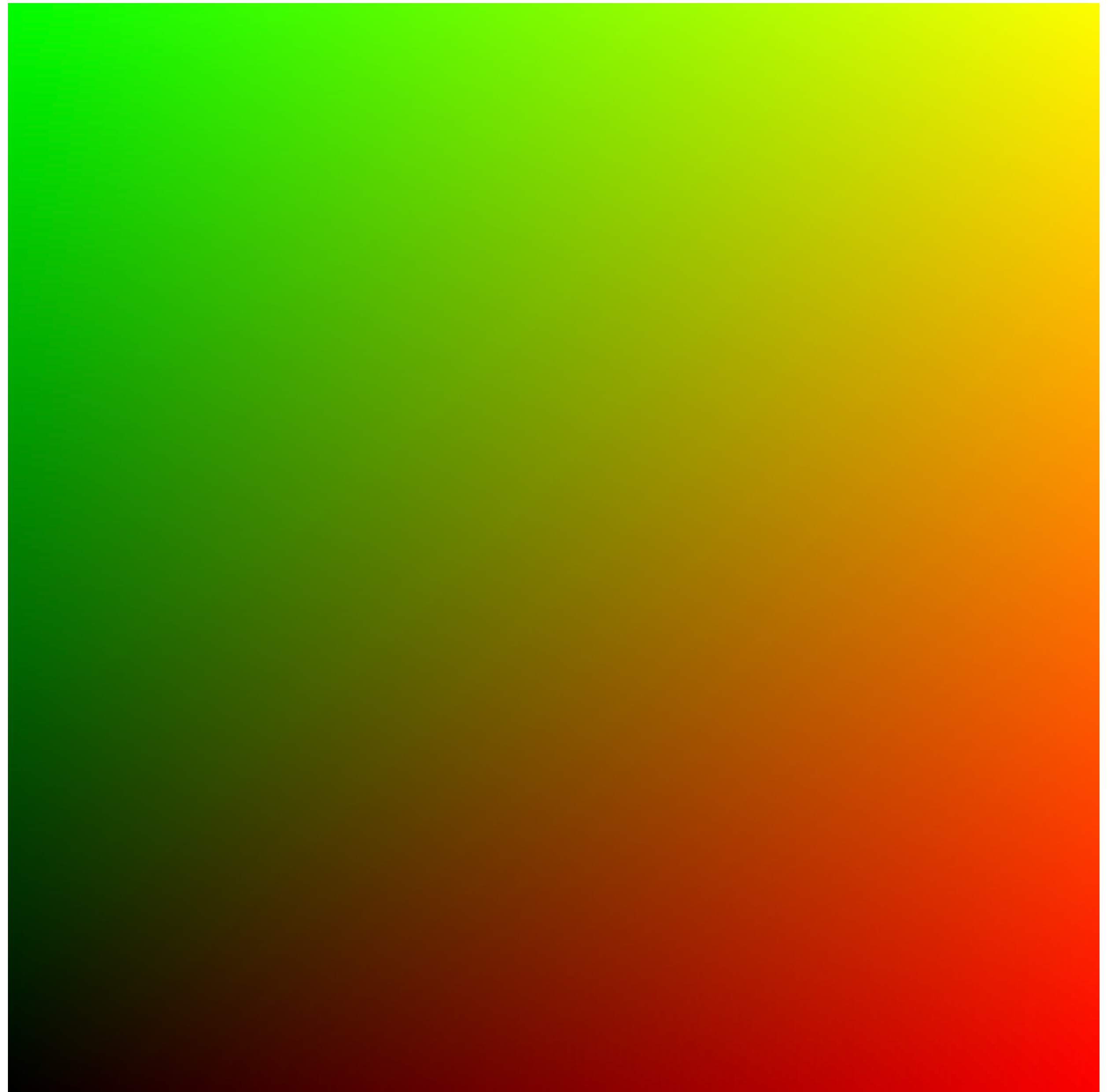
Render Encoder

(quad for triangle strip)



Render Encoder

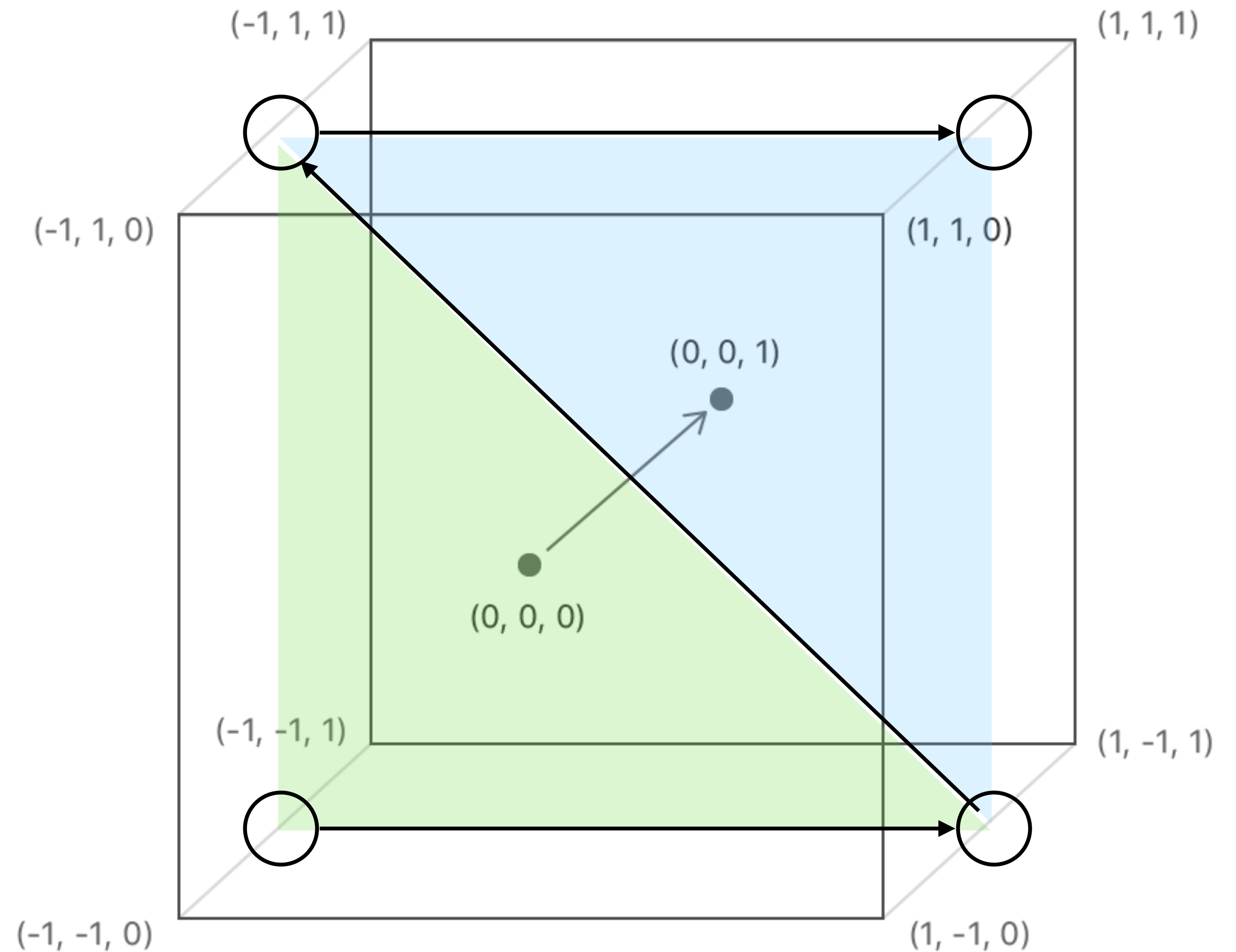
Simple full screen quad



Render Encoder

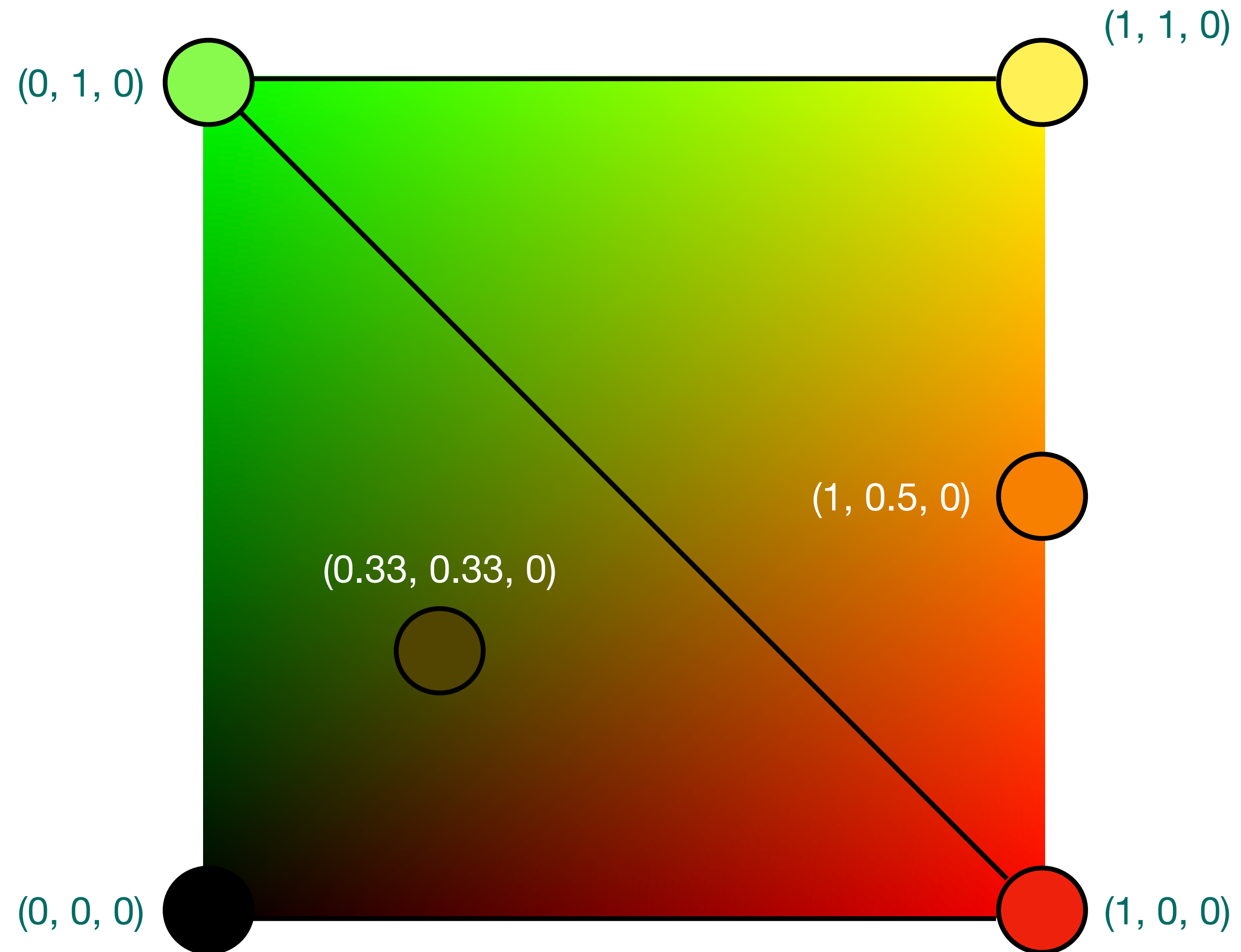
Viewport Coordinates

```
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



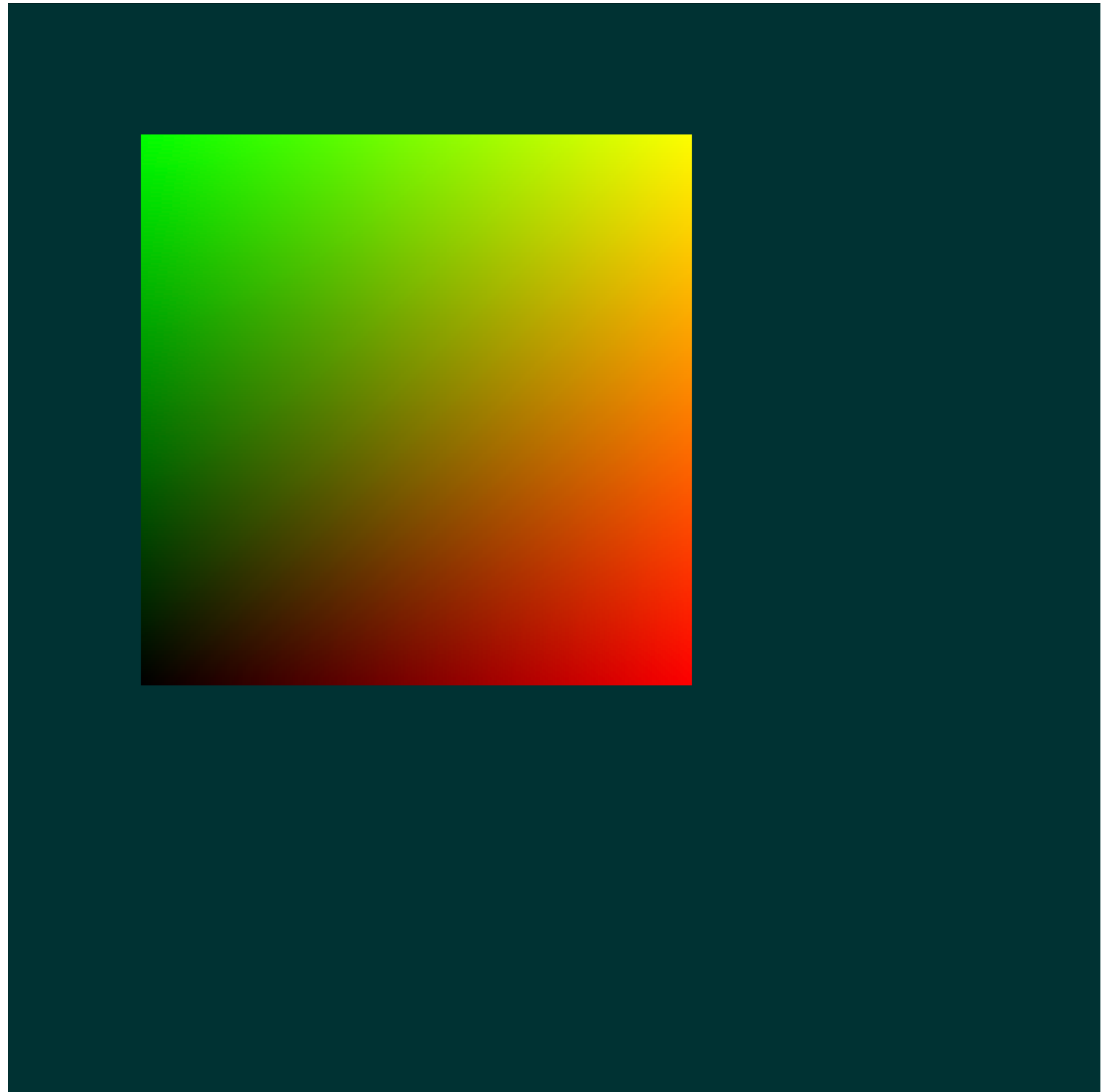
Render Encoder

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))  
  
// ...
```

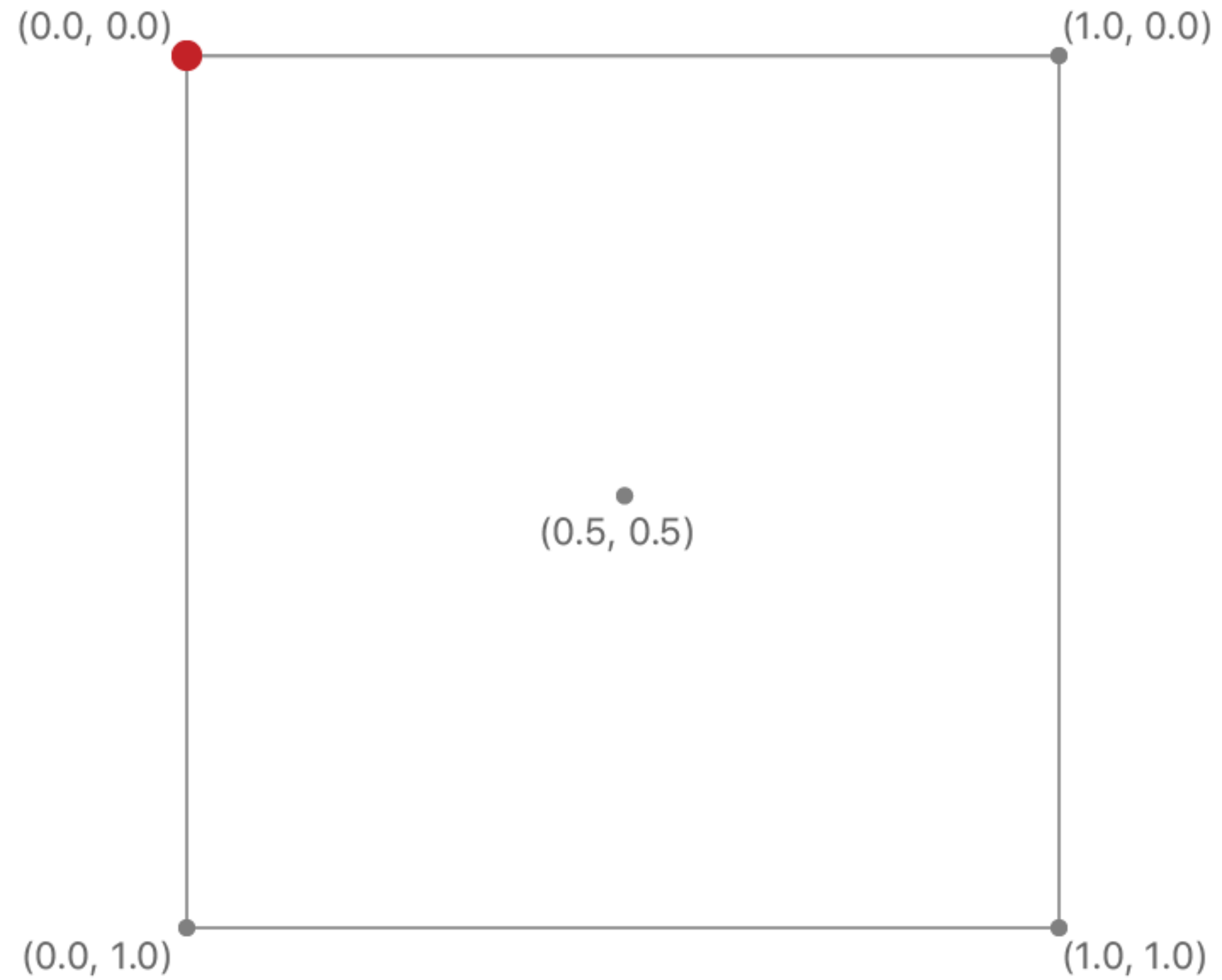
Render Encoder

Simple “full screen” quad into viewport



Render Encoder

Texture Coordinates



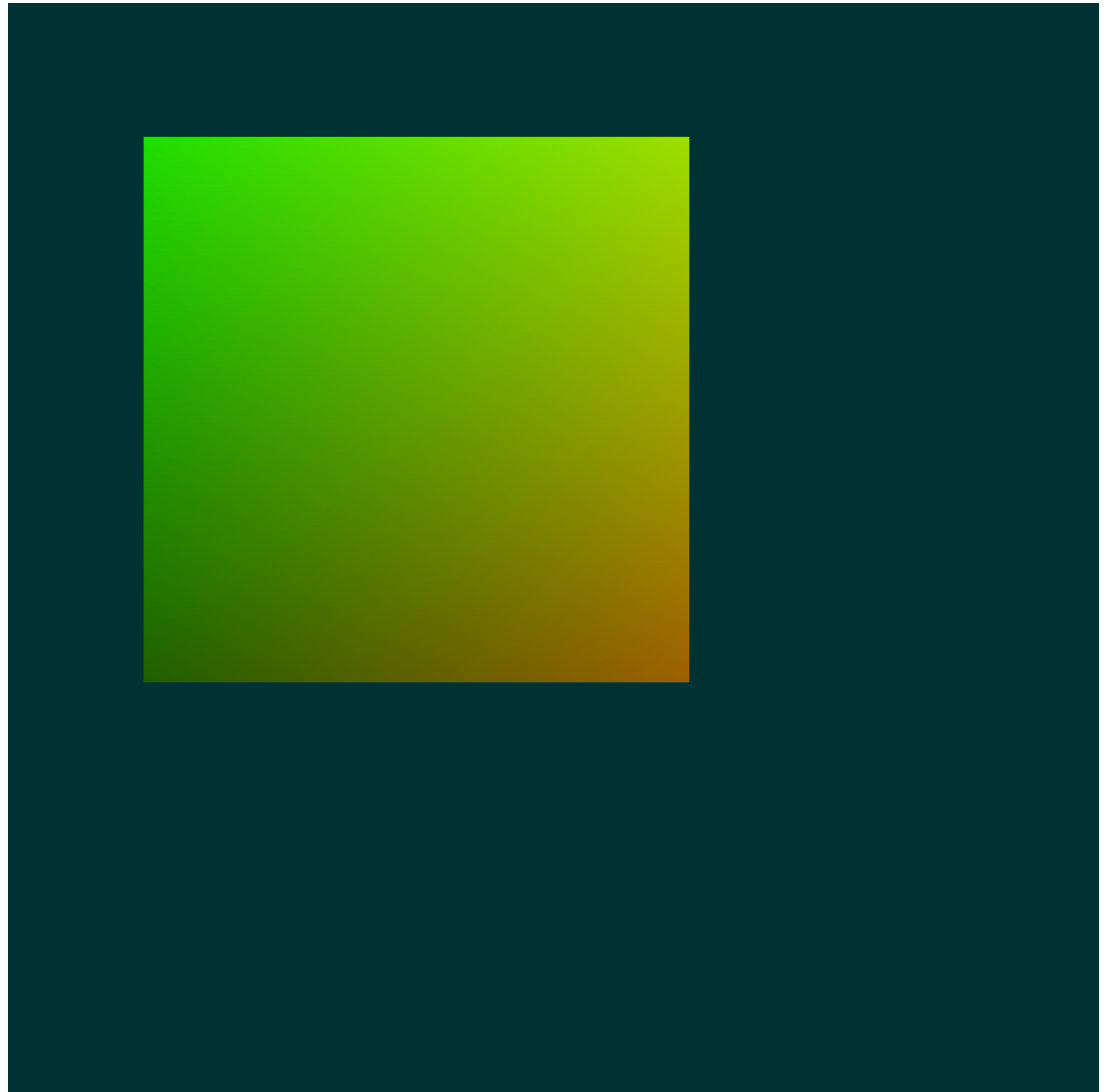
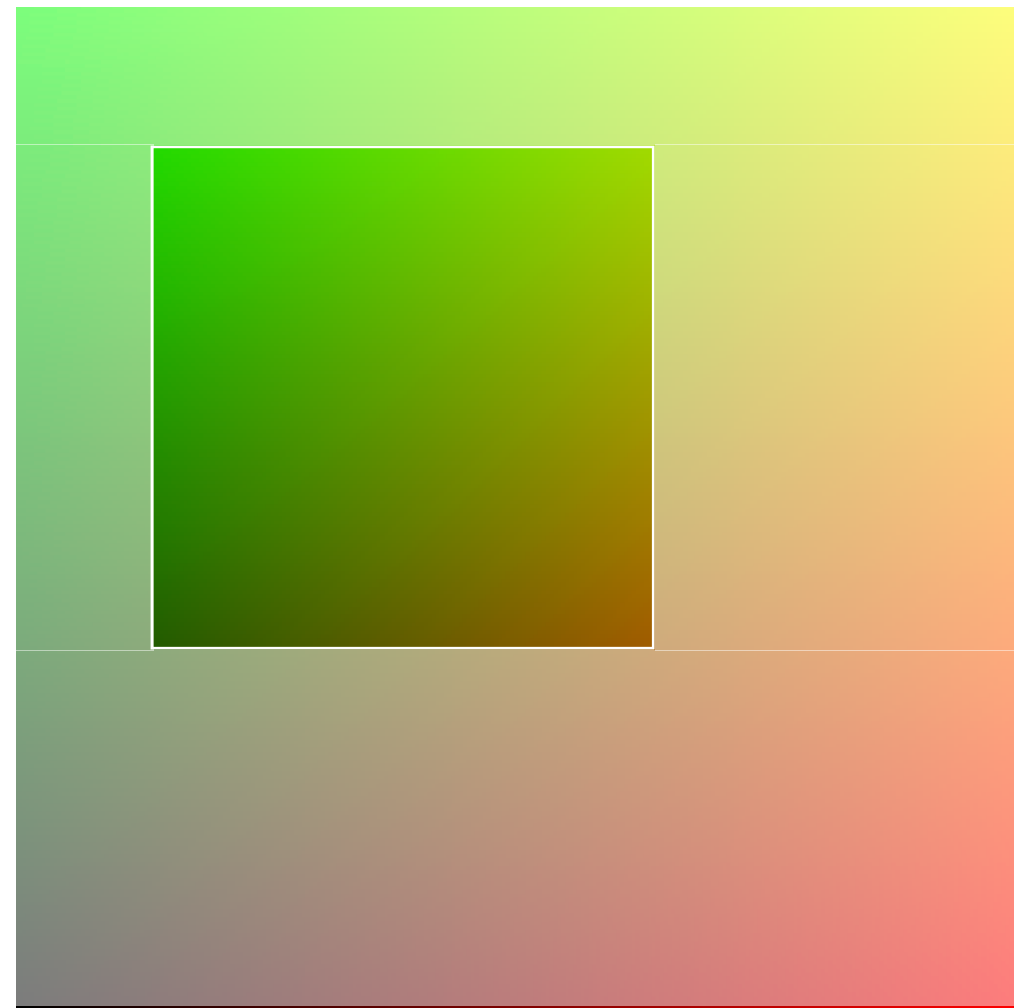
Render Encoder

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)))  
  
// ...
```

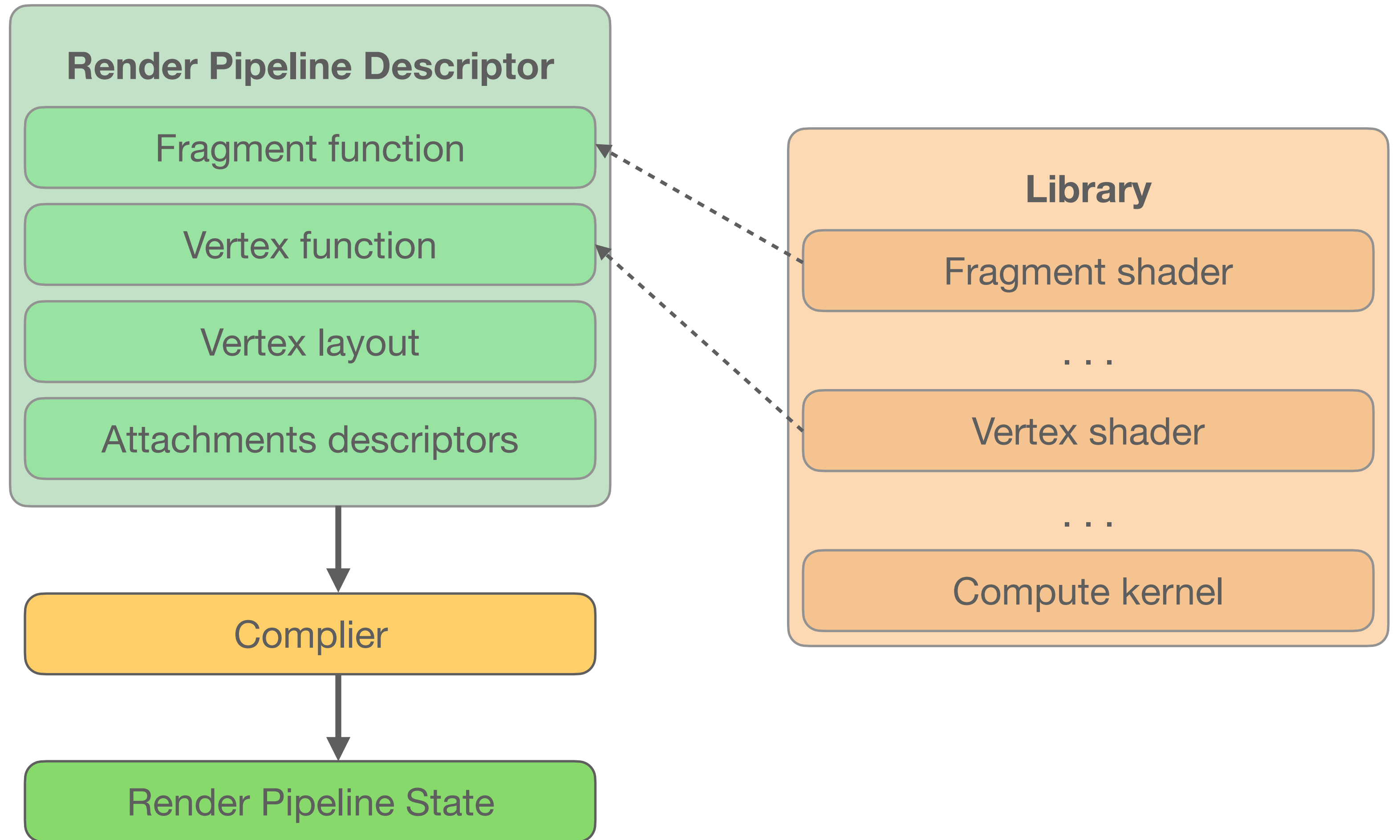
Render Encoder

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)
}
```

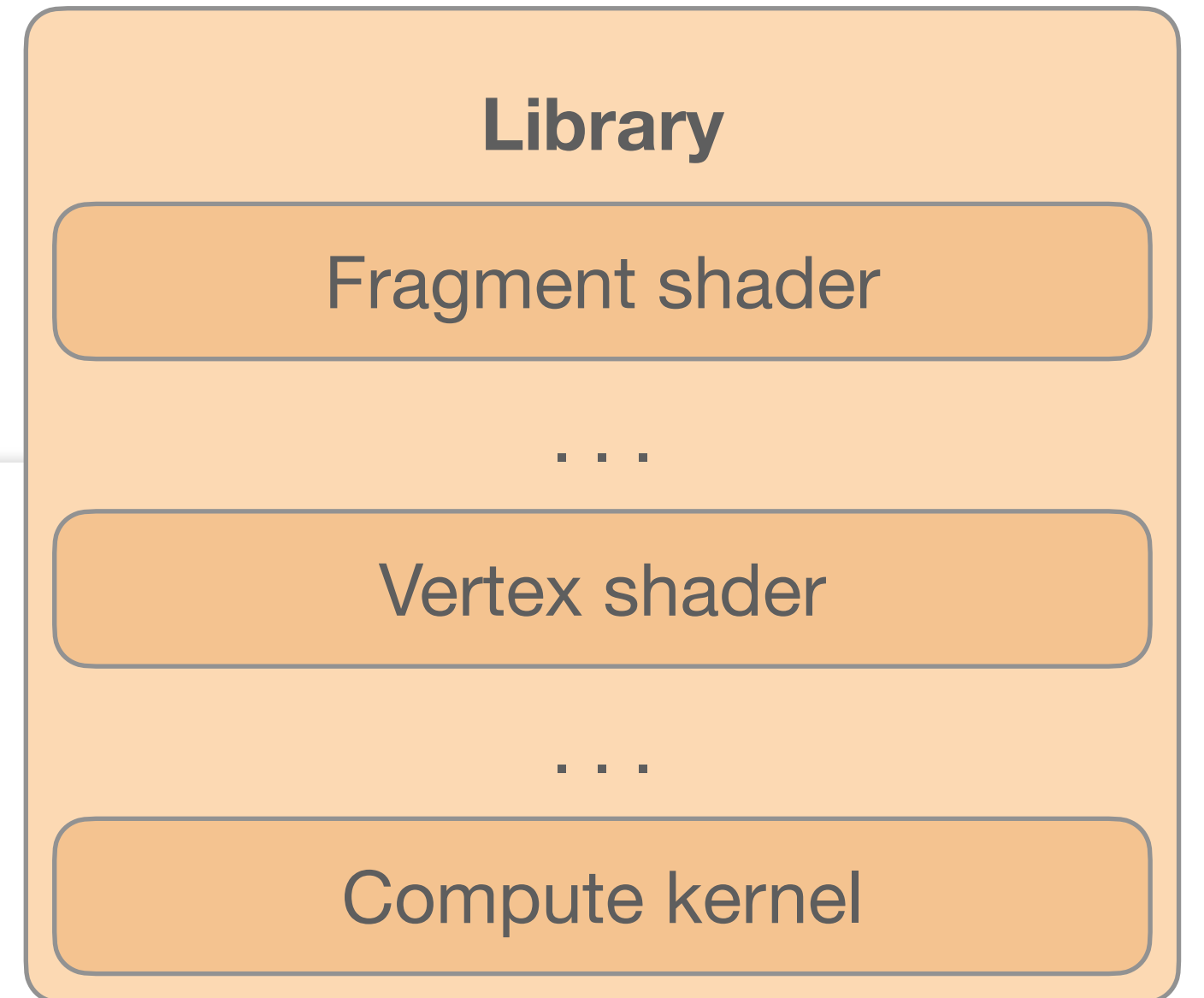
Pipeline

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

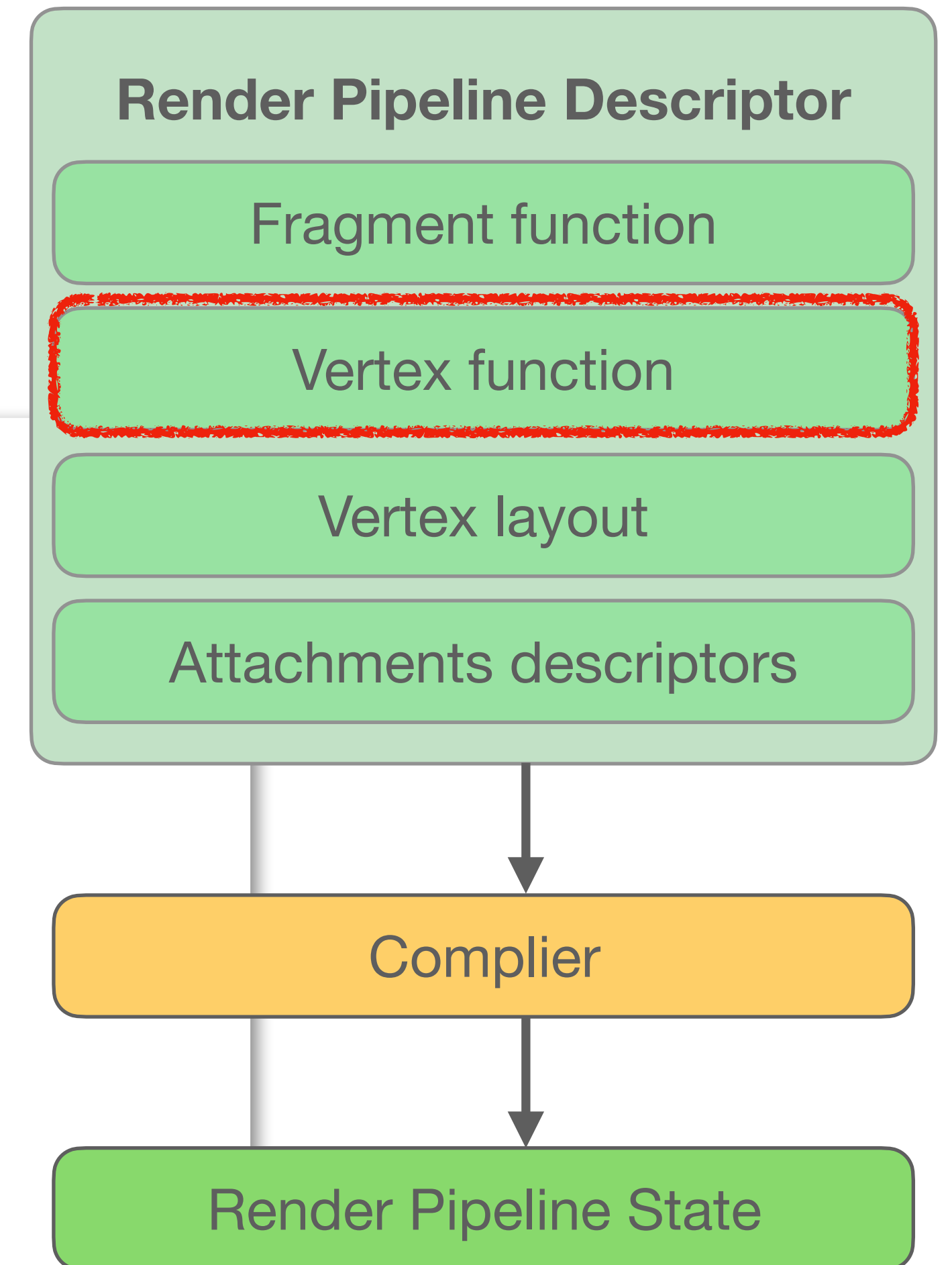
    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



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)  
}
```



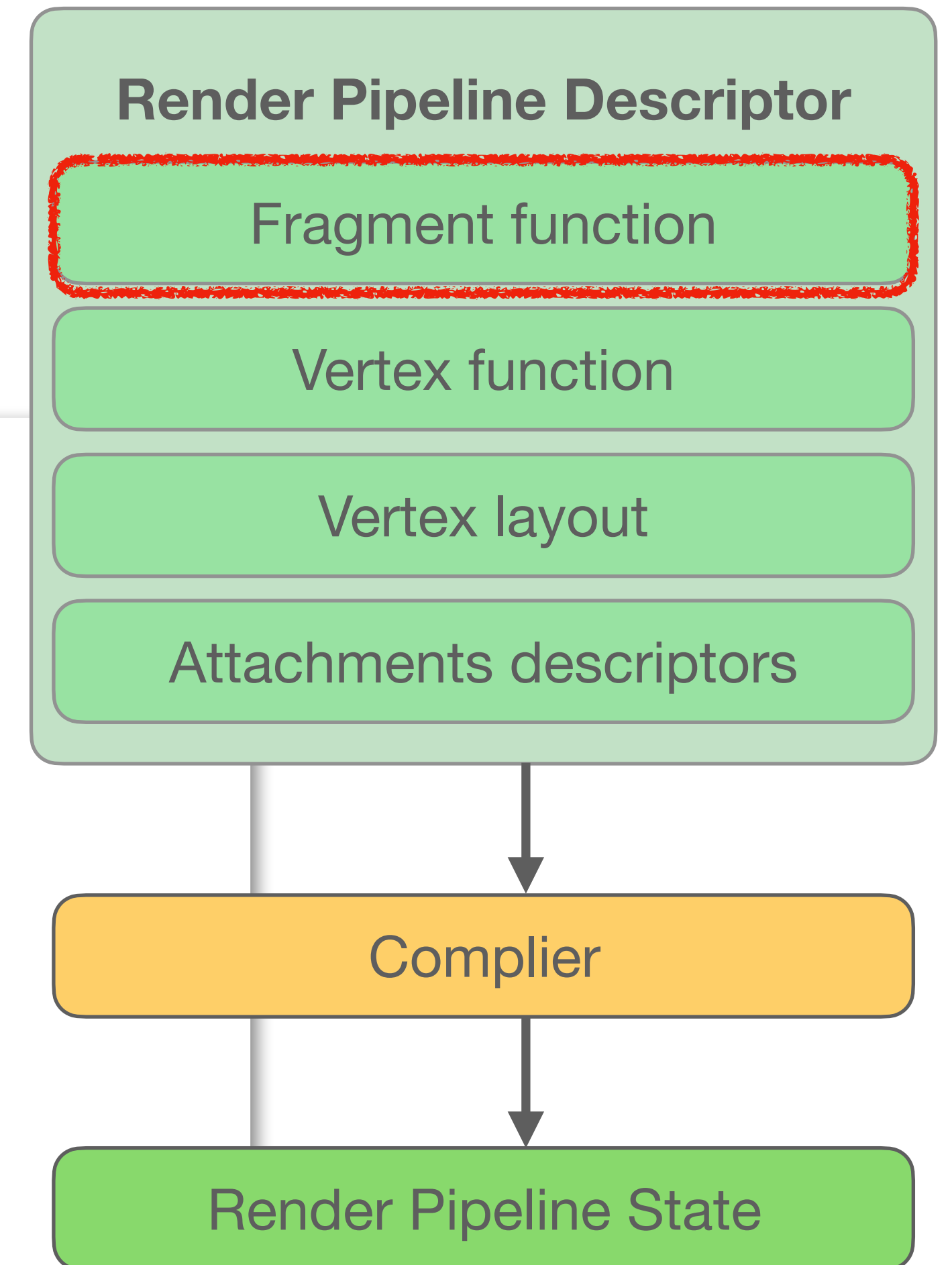
Pipeline

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)
}
```



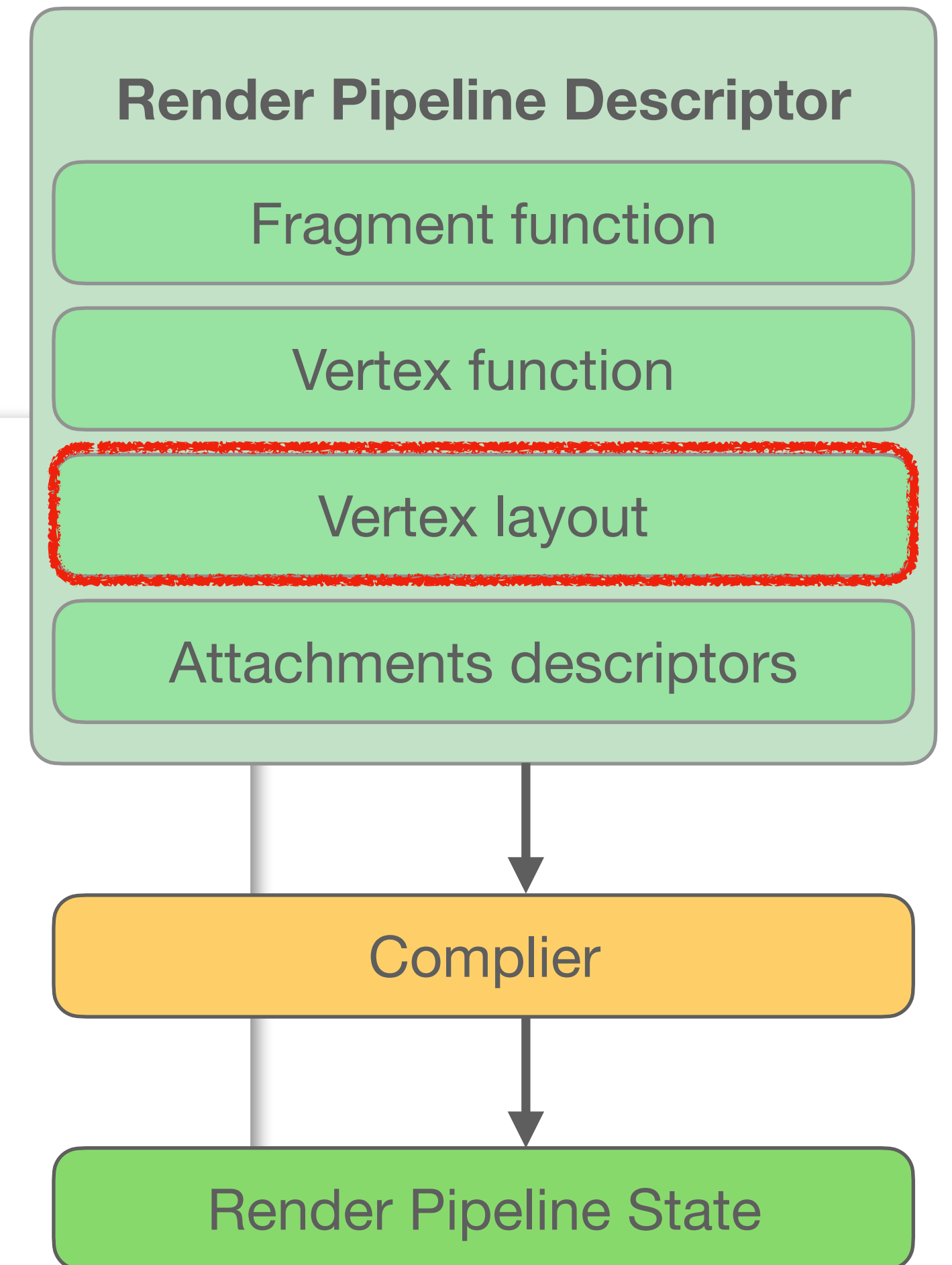
Pipeline

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)
}
```



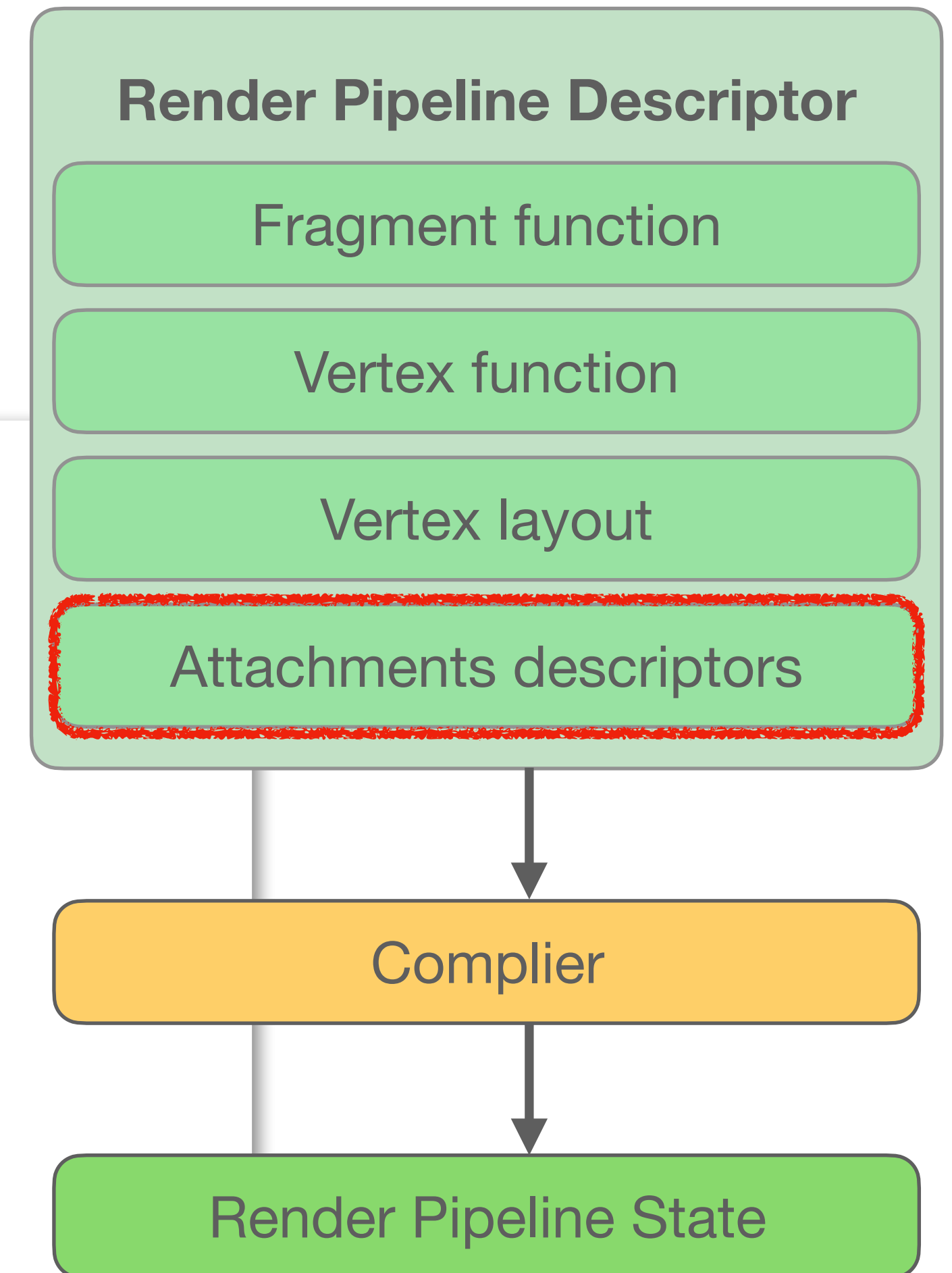
Pipeline

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

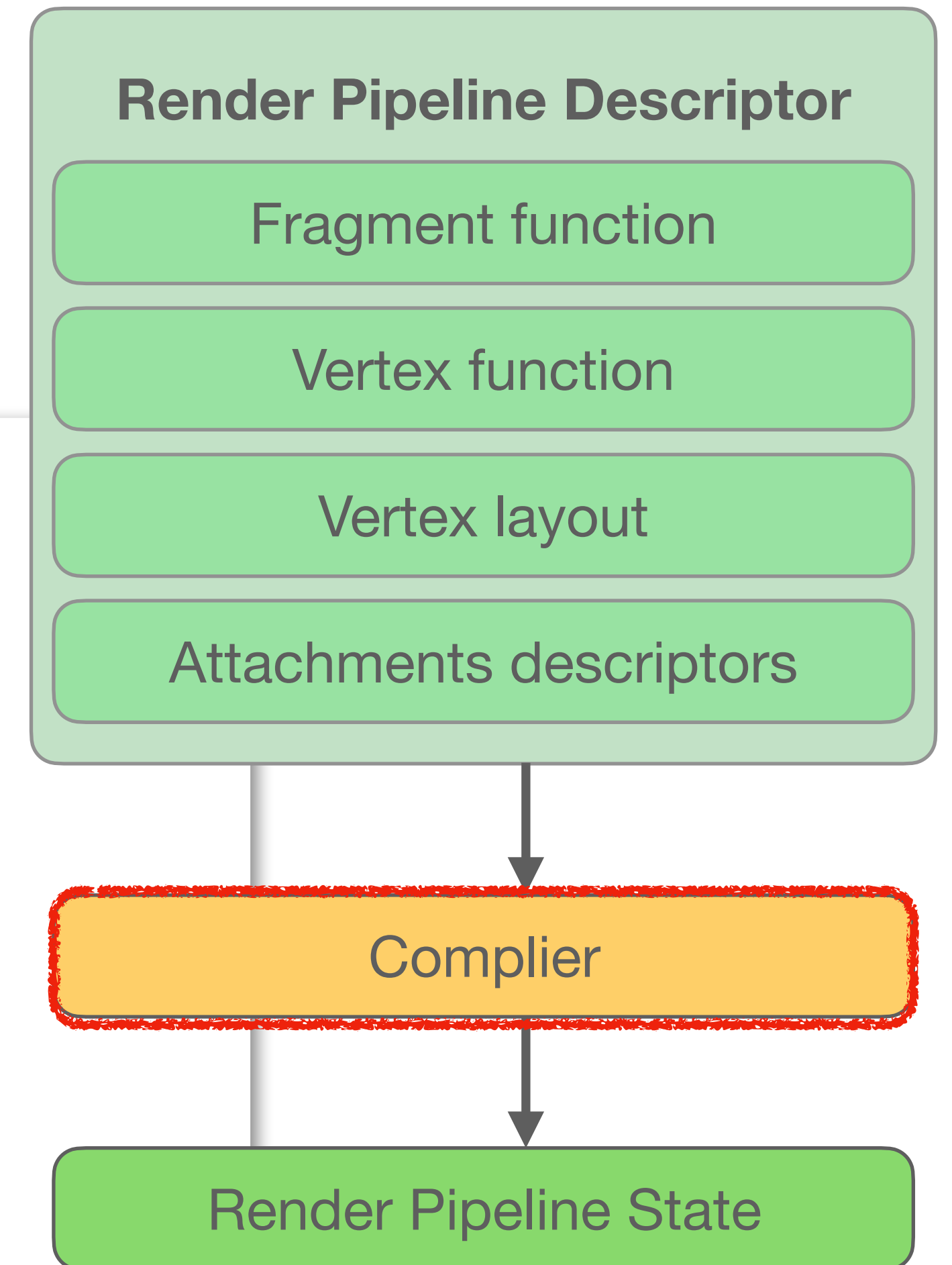
    return try device.makeRenderPipelineState(descriptor: pipelineDescriptor)
}
```



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

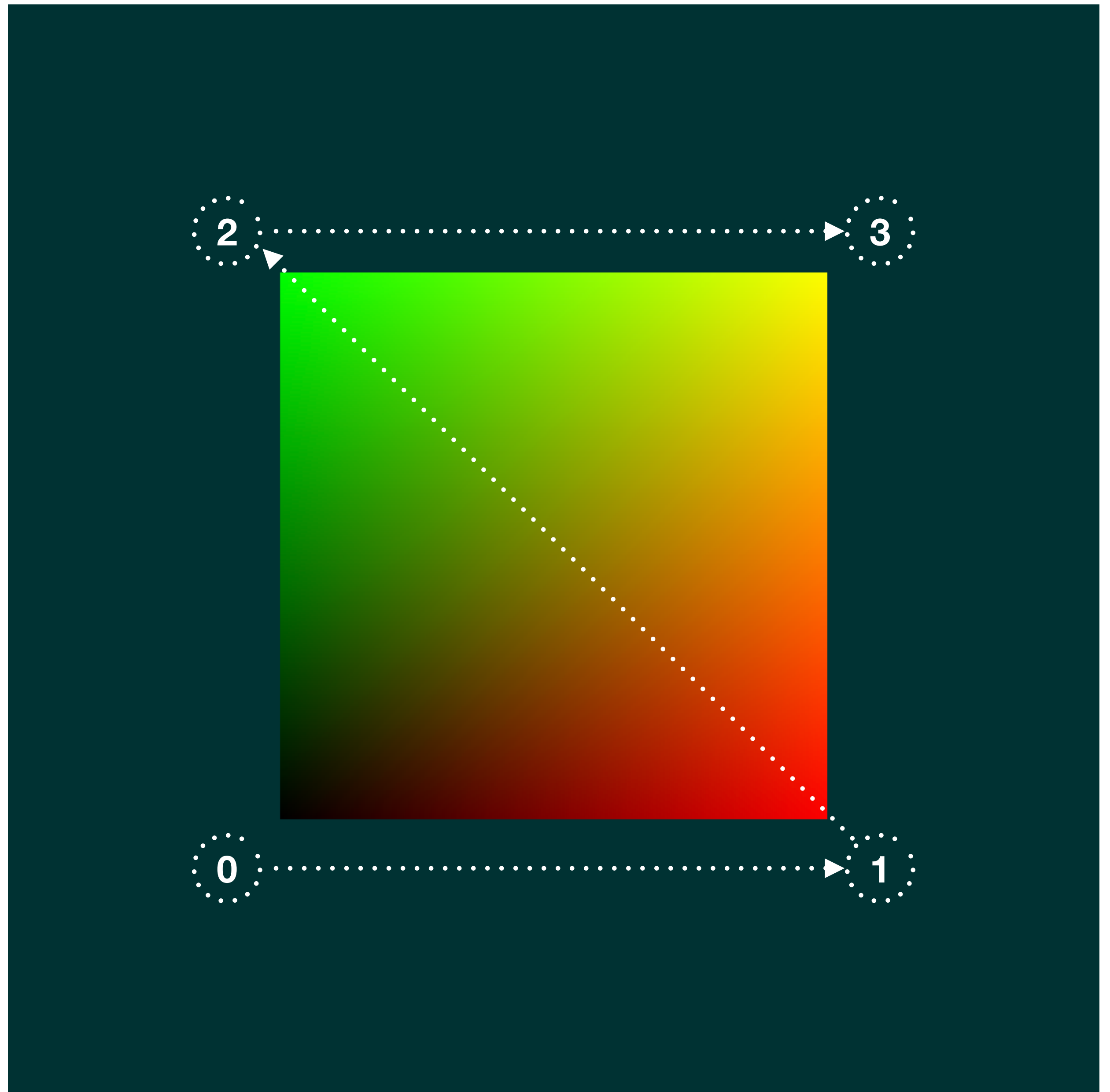
Draw Call

Primitives (MTLPrimitiveType)

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

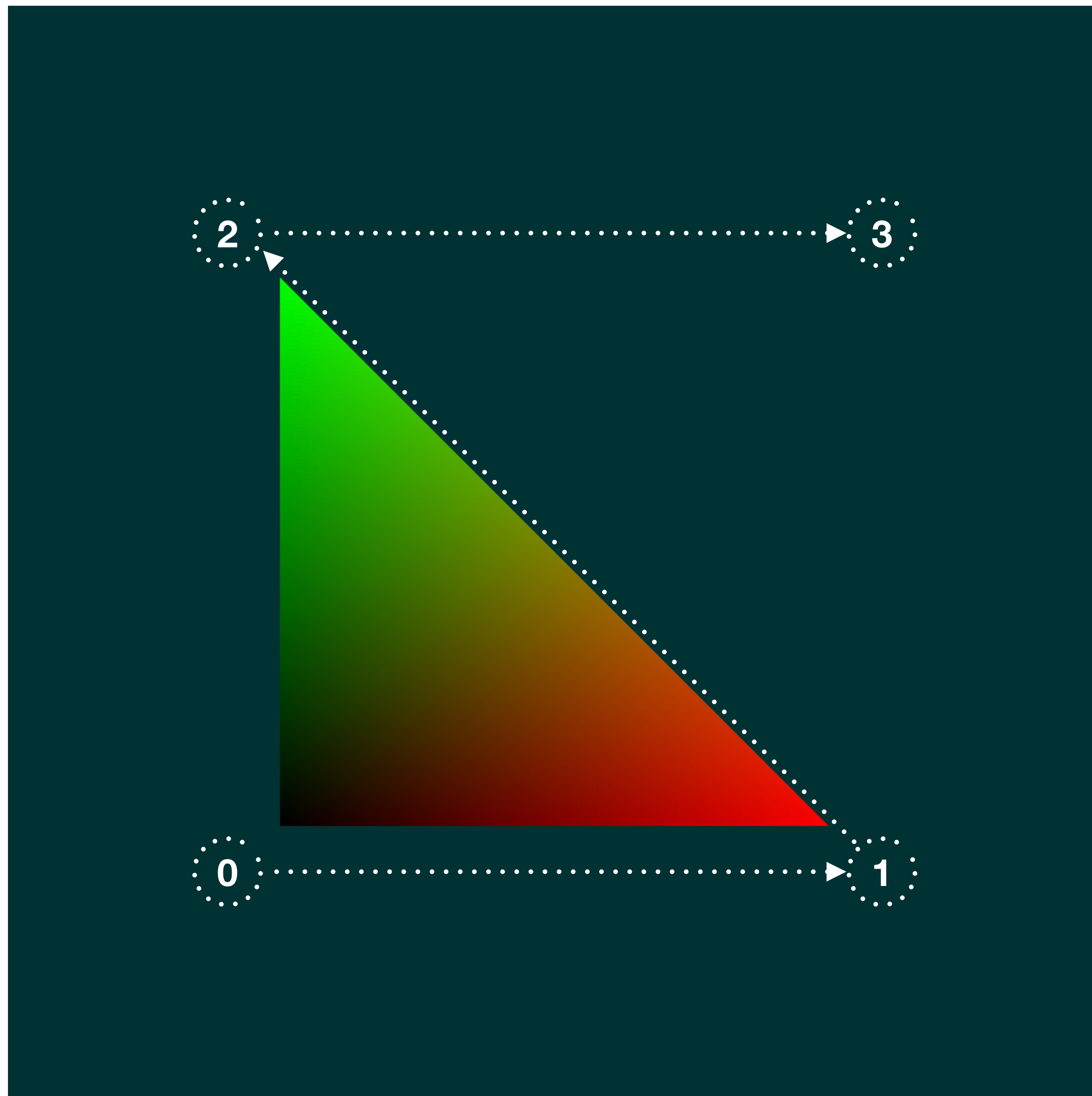
Draw Call Primitive

triangleStrip



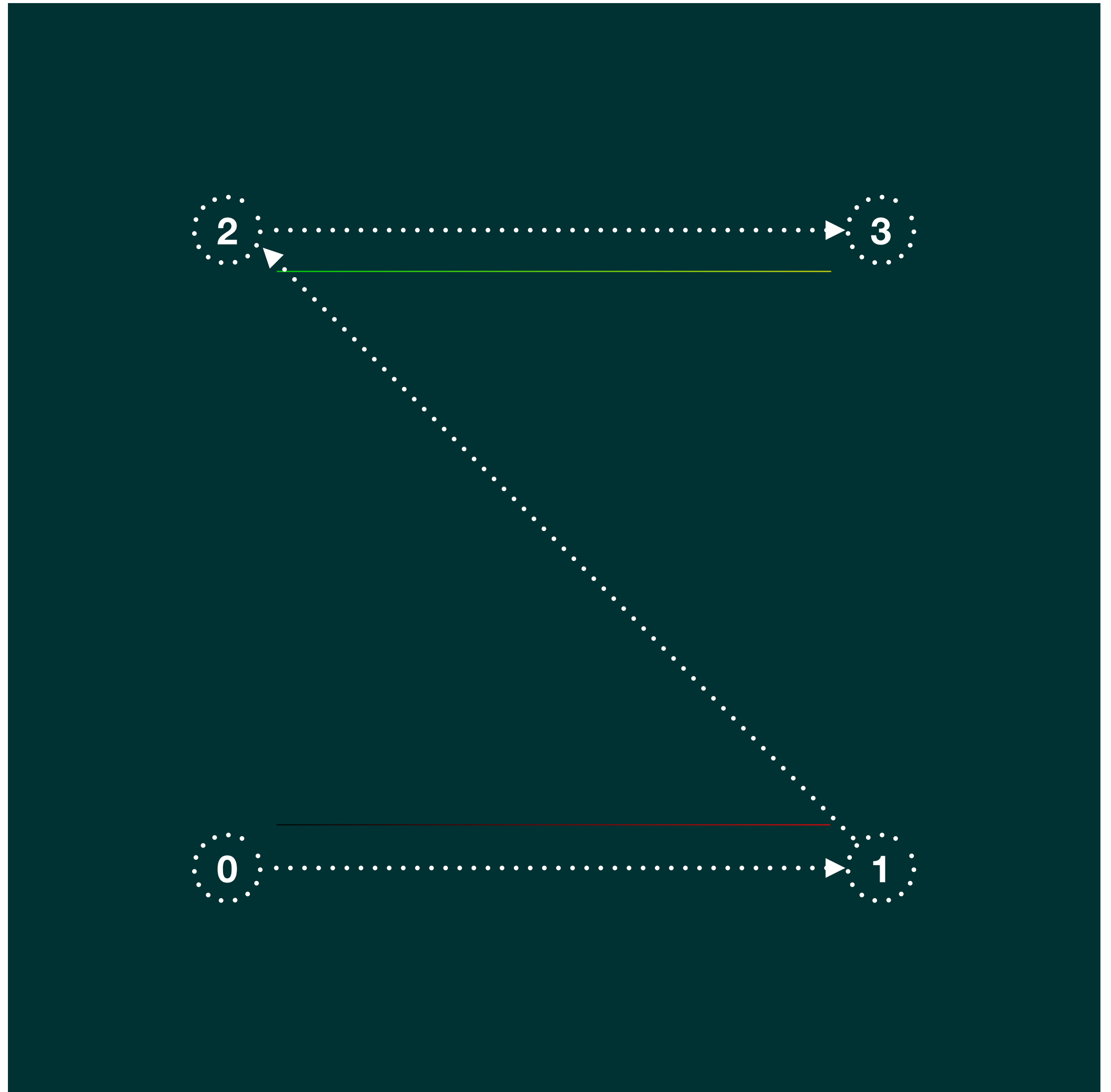
Draw Call Primitive

triangle



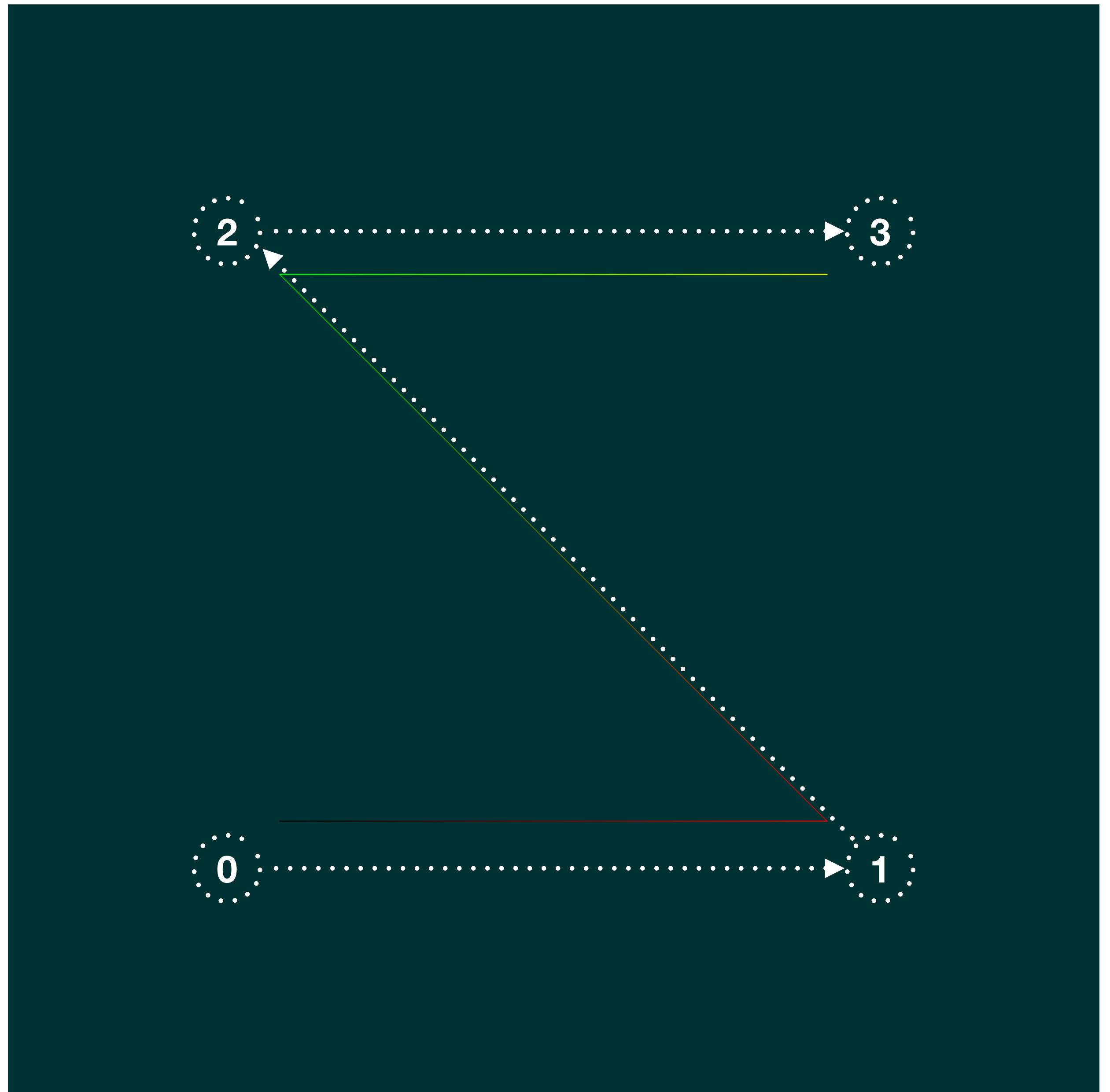
Draw Call Primitive

line



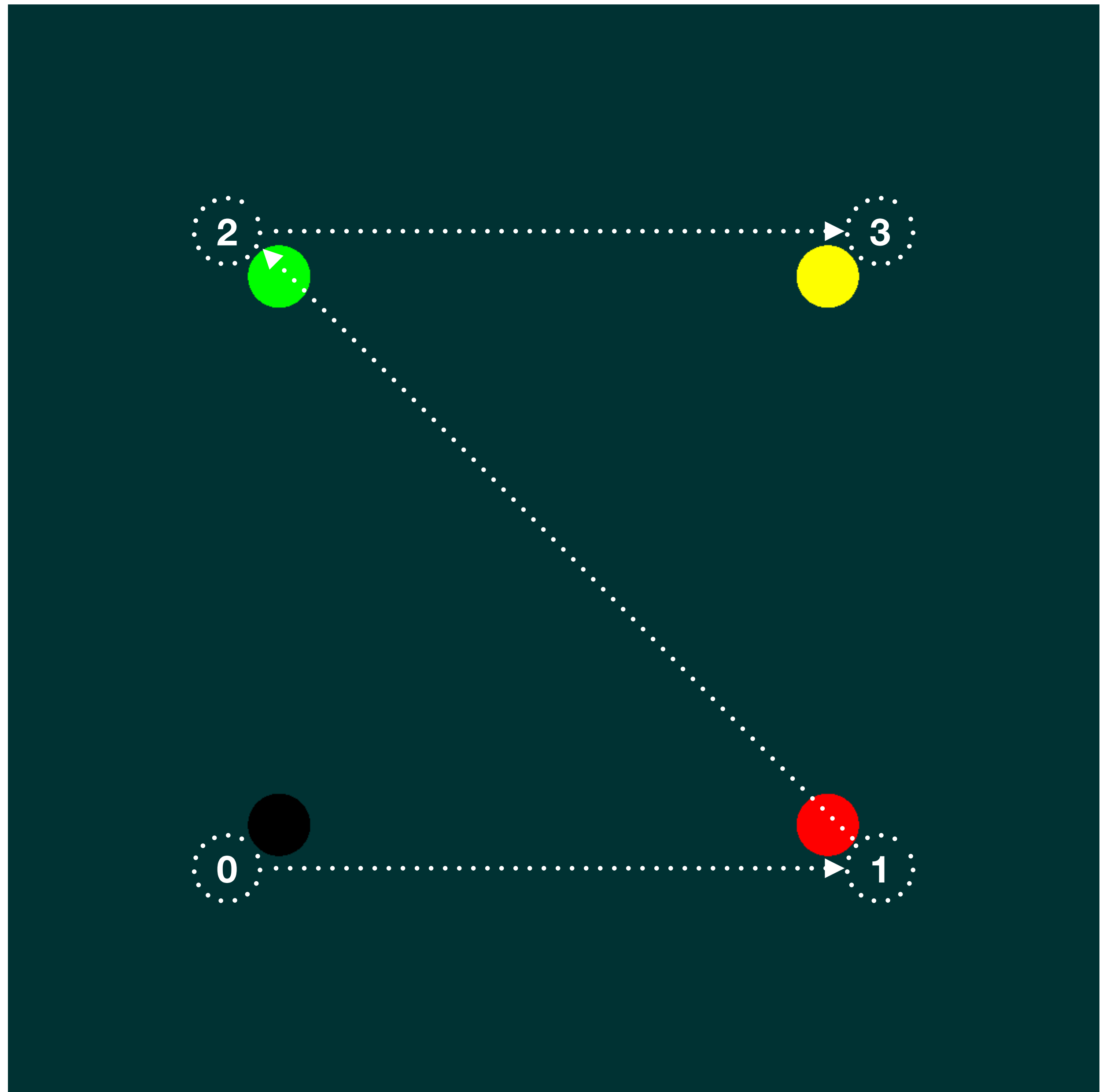
Draw Call Primitive

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;
}
```

```
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);
}
```

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


Draw Call

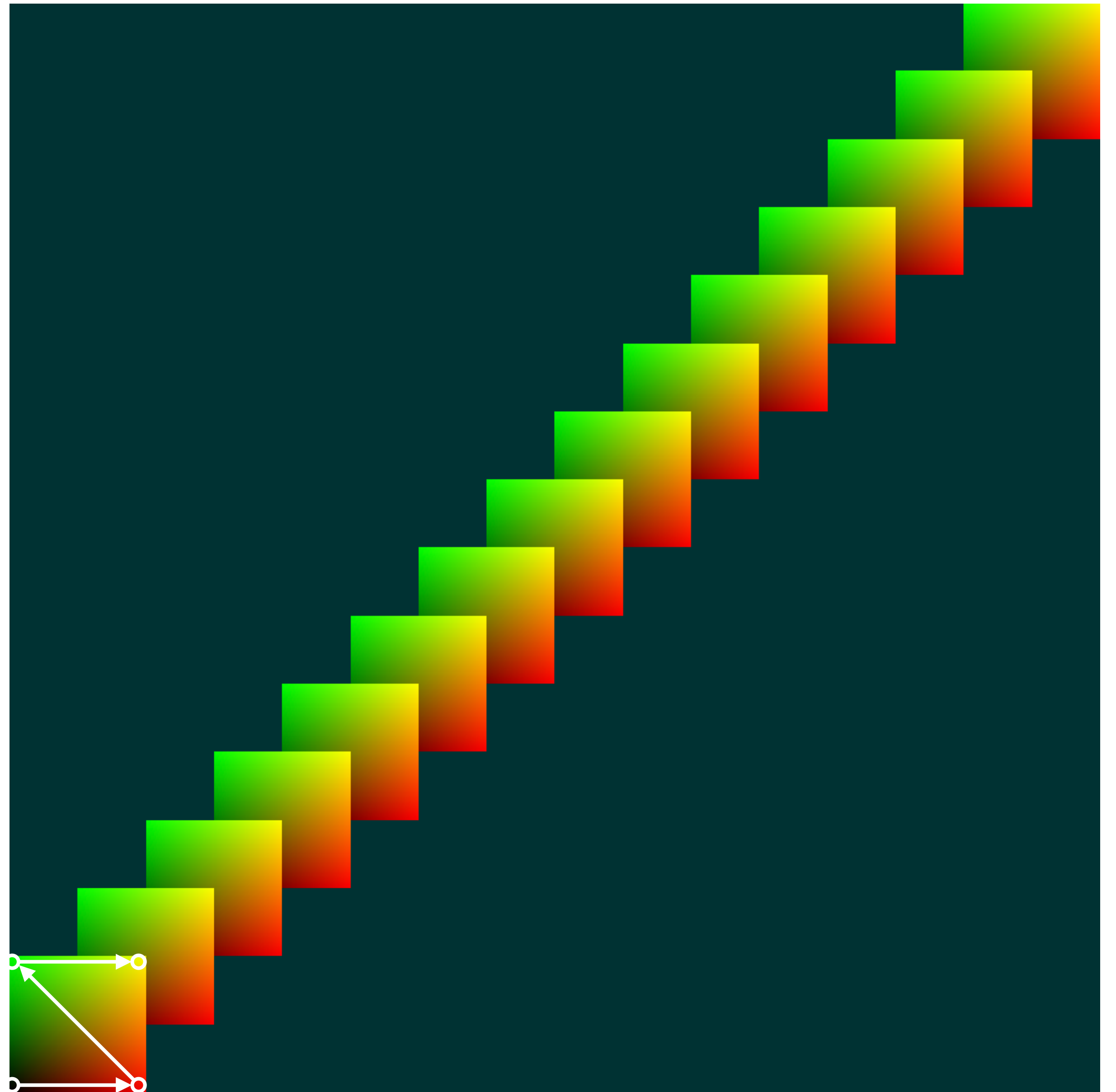
Vertices and Instances

```
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;  
}
```

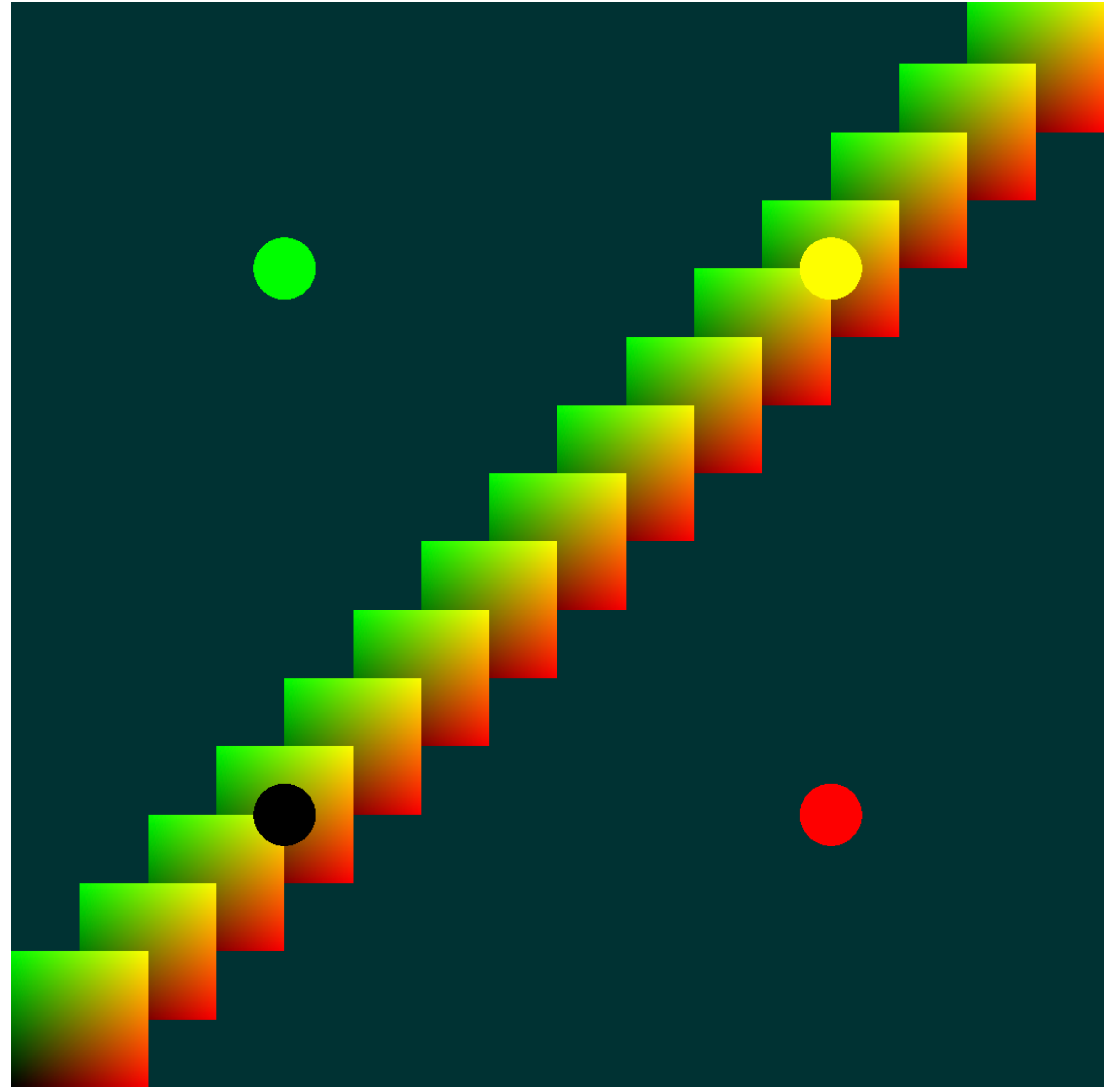
Draw Call

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)

Buffer 1: *Coordinates*

0:	(120, 130)
1:	(135, 114)
2:	(117, 143)
3:	(214, 31)
	...
<i>n</i> :	(830, 105)

Buffer 2: *Indices*

0:	1
1:	3
2:	5
	...
<i>m</i> :	42

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;
}
```

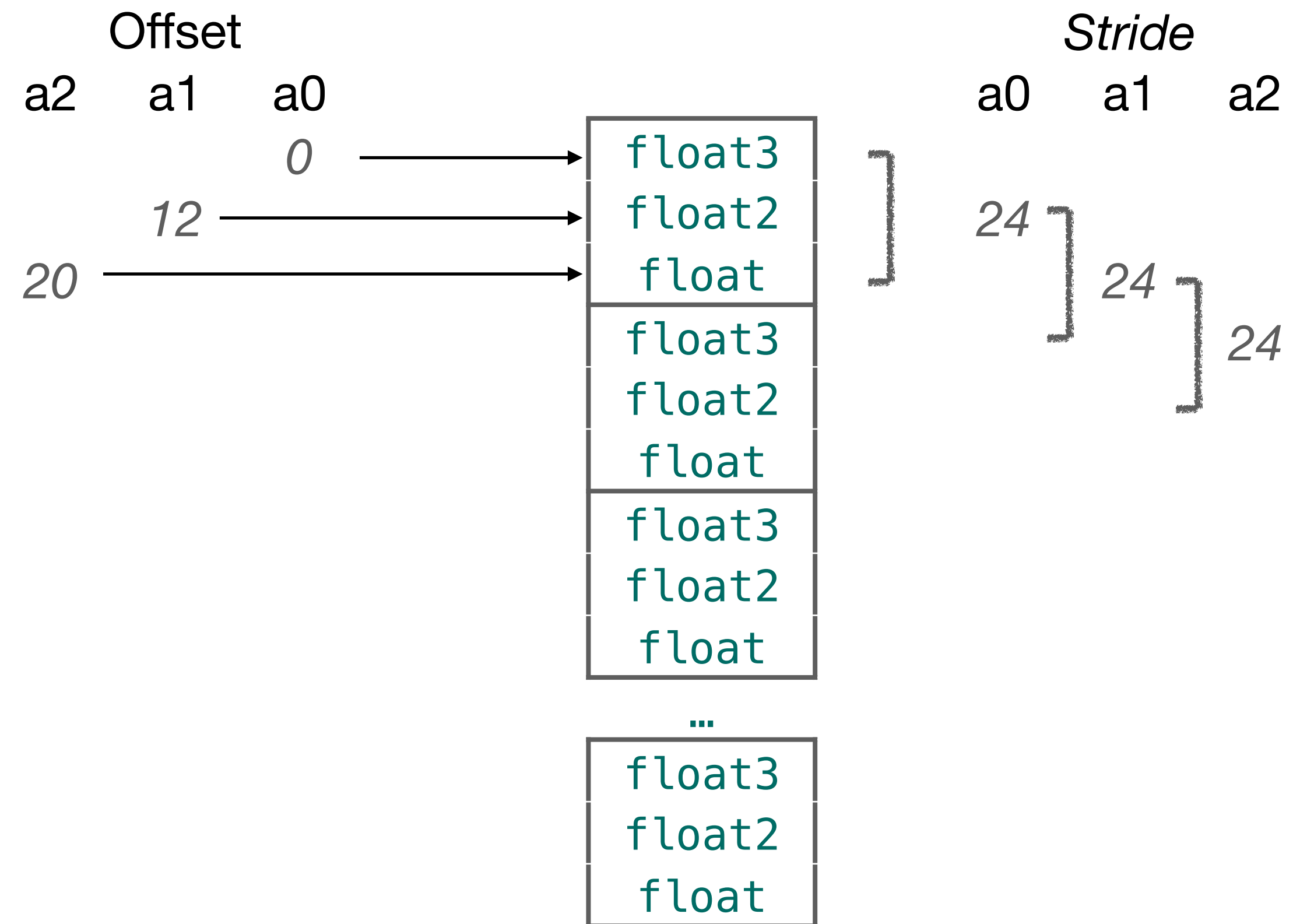
Buffers

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

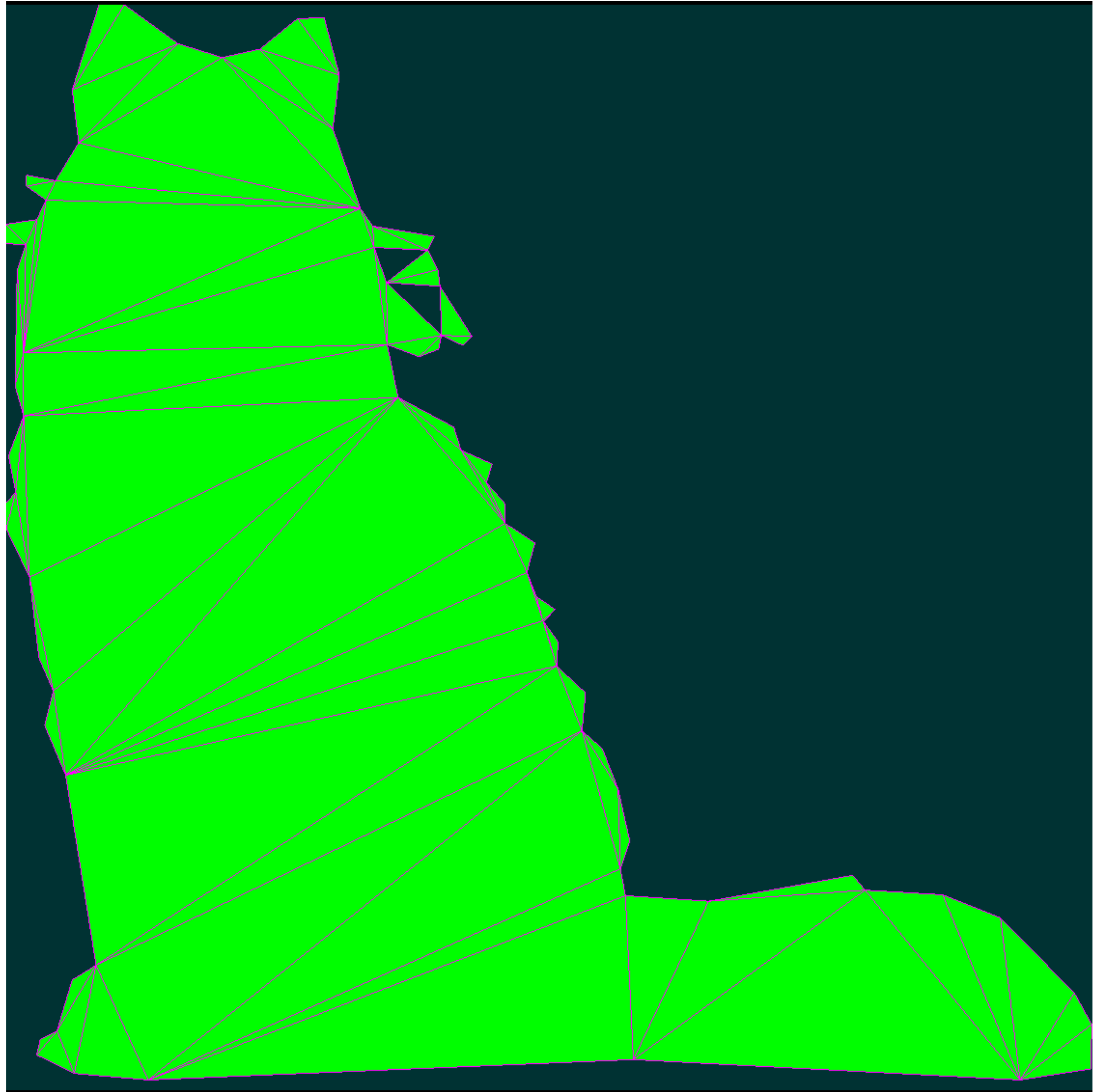
```
renderEncoder.setVertexBuffer(vertices, offset: 0, index: 0)
renderEncoder.drawPrimitives(type: .triangle,
                             vertexStart: 0,
                             vertexCount: mesh_vertices_count)
```

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



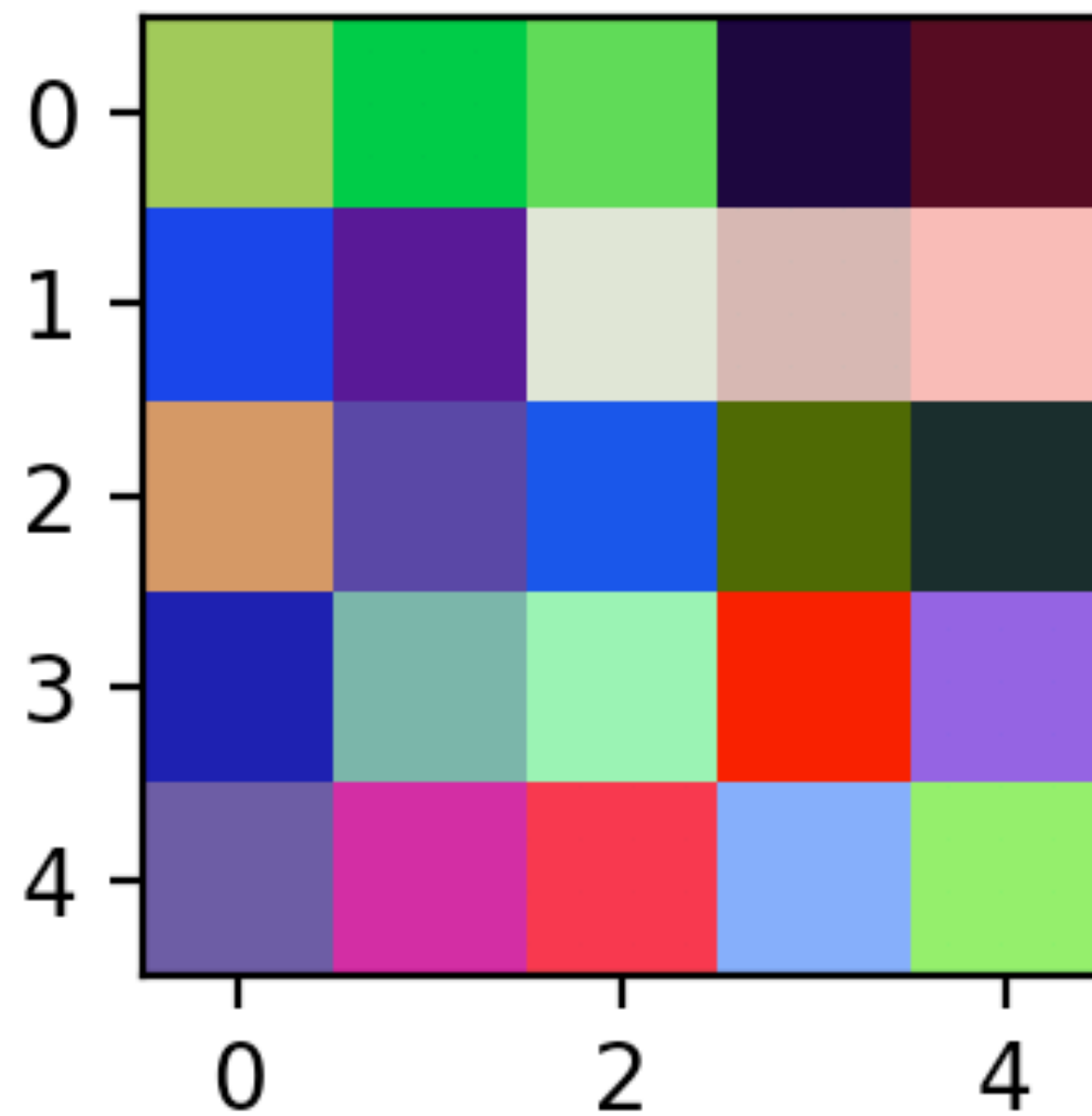
Sampling
clamp_to_zero



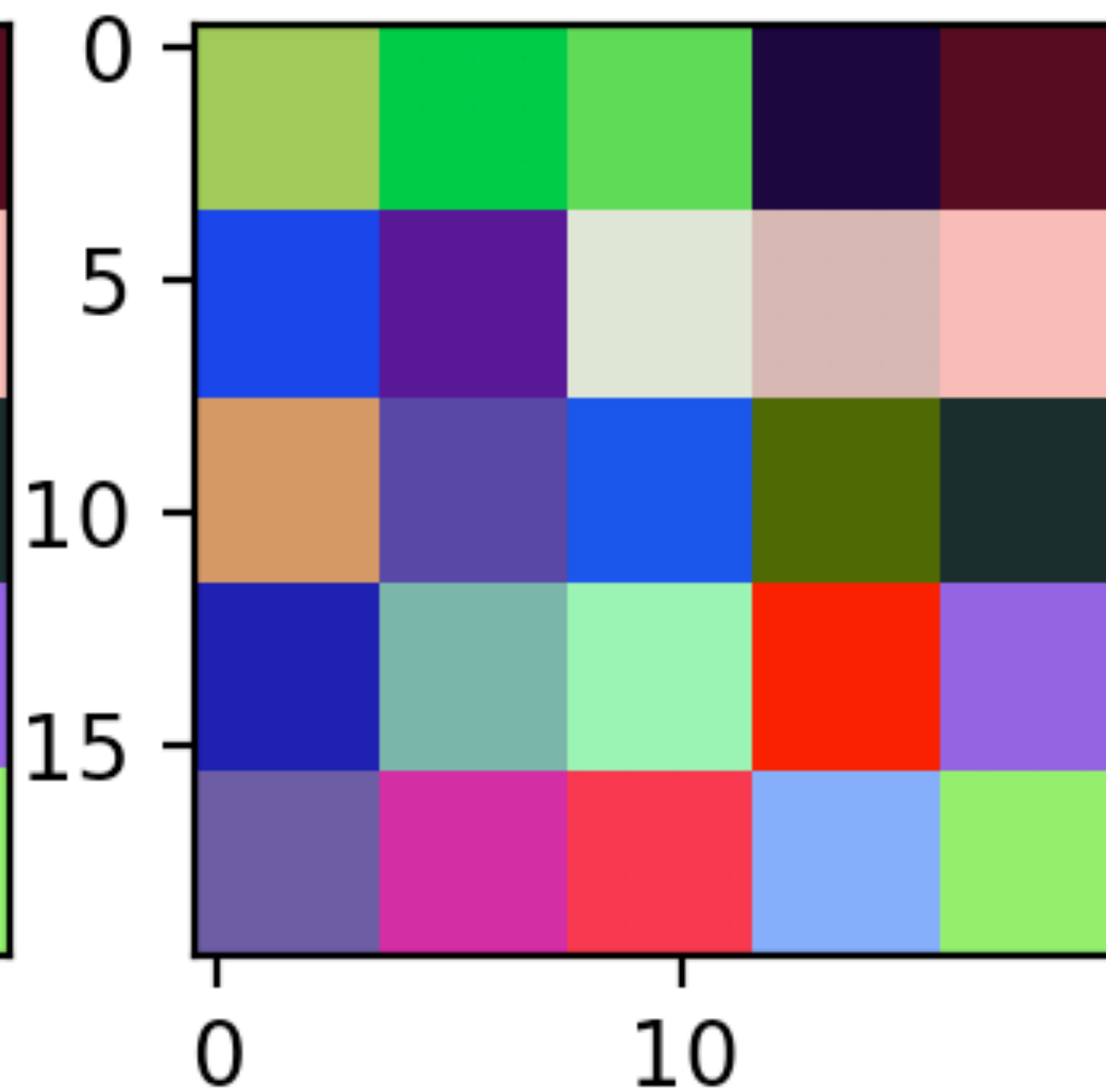
Sampling

nearest vs linear

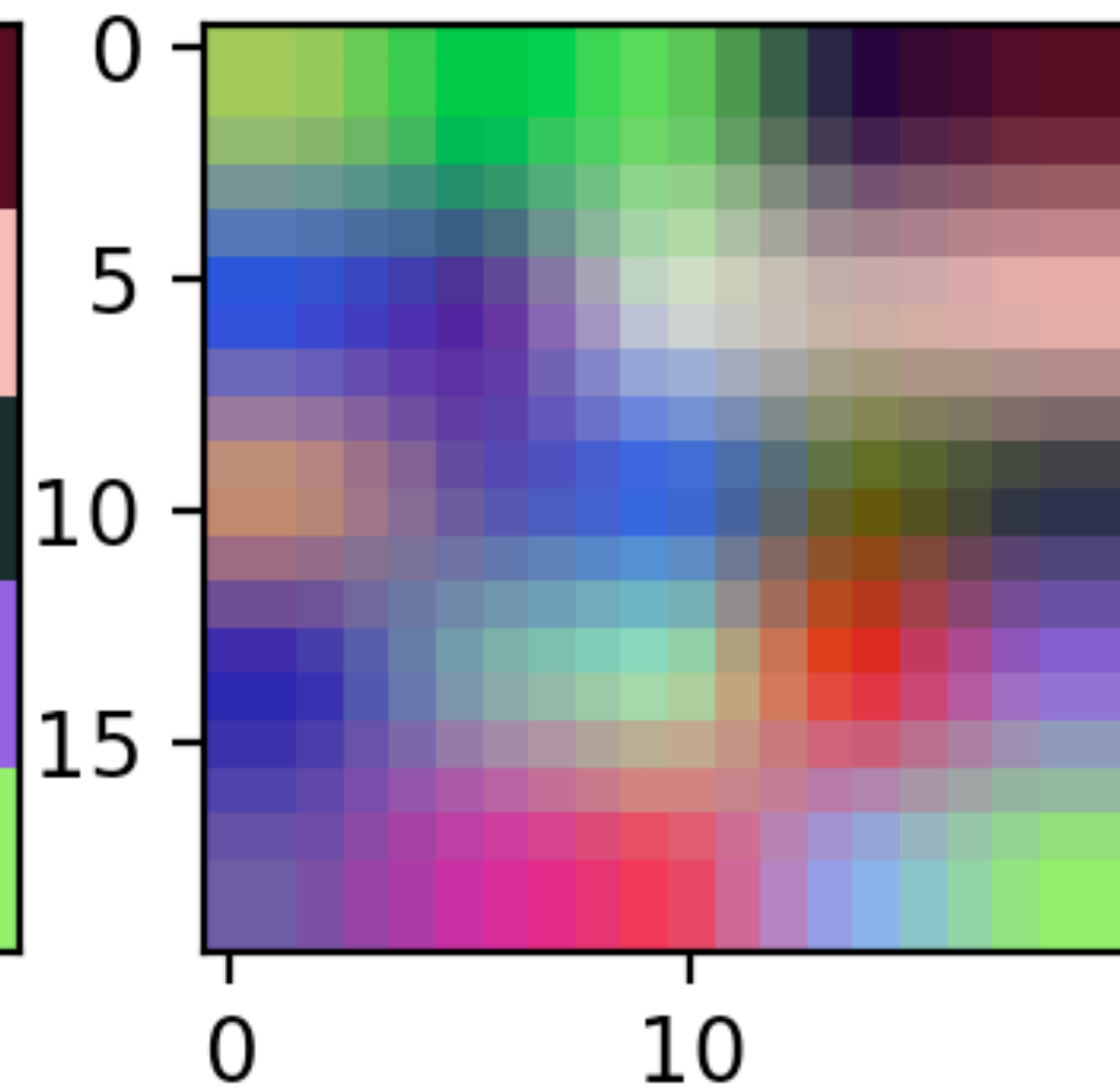
Source image (5x5)



Result image (20x20)
nearest interpolation



Result image (20x20)
linear interpolation



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;

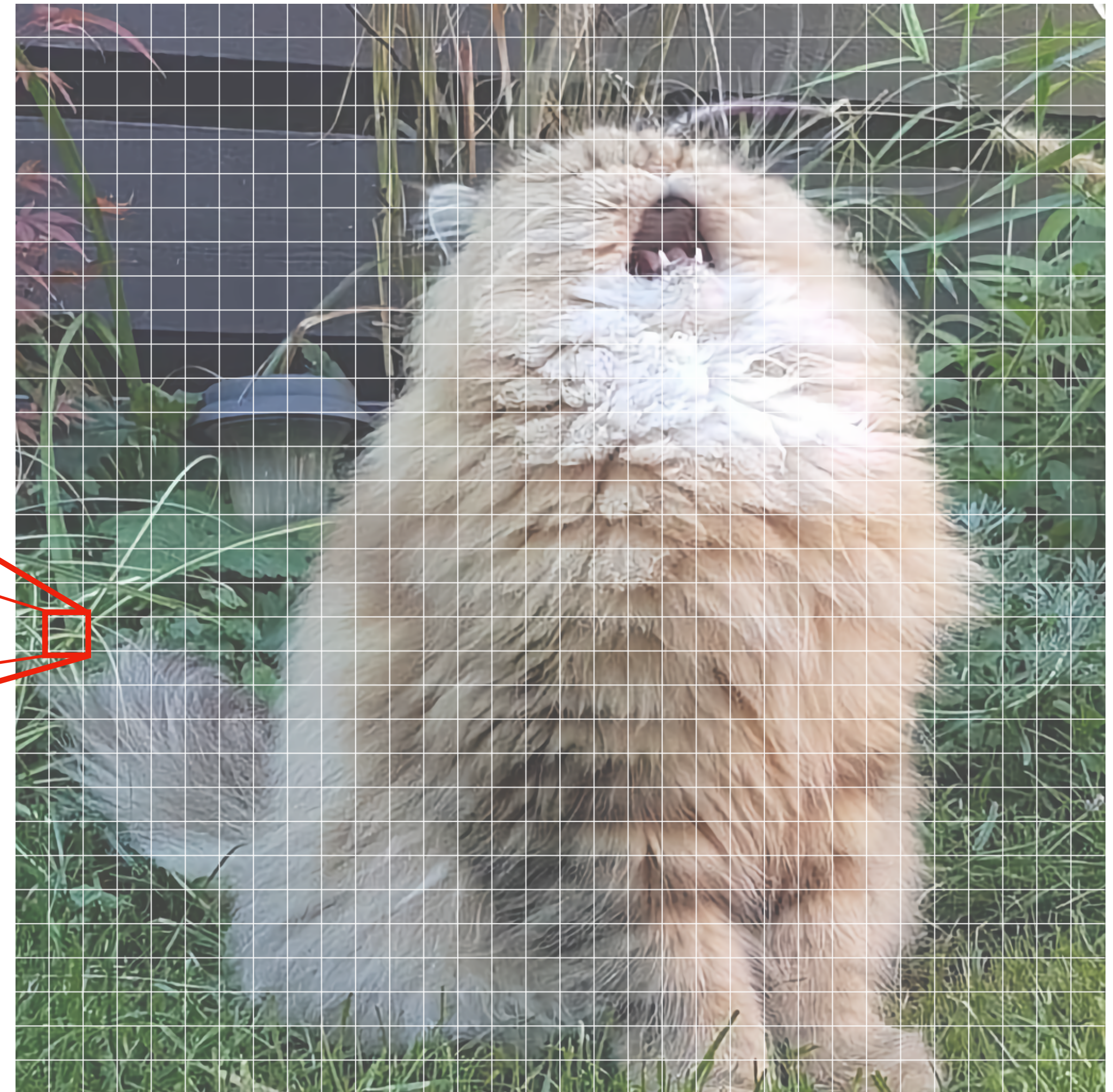
    pos_val[gid] = particle;
}
```

Compute Encoder

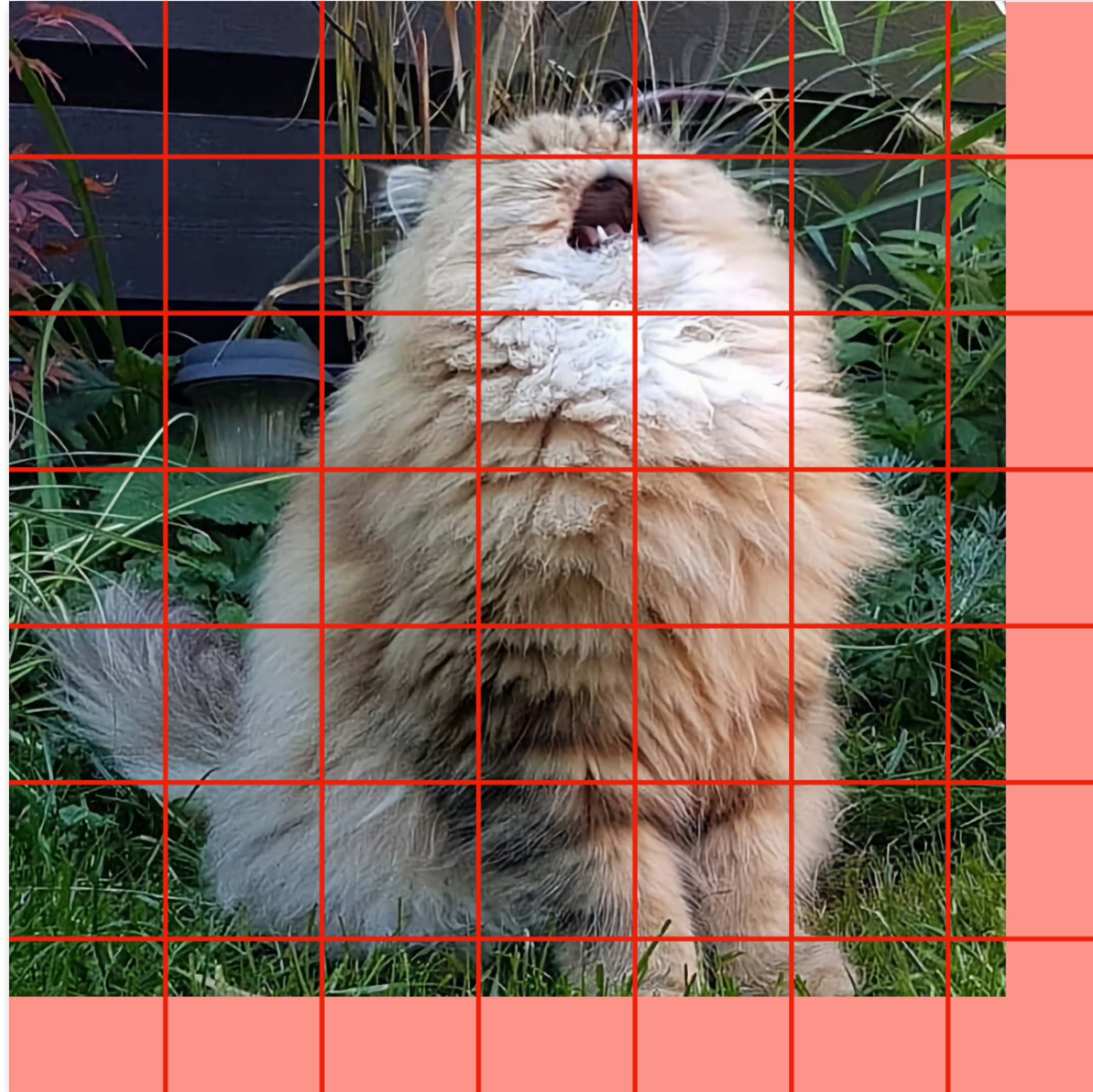
Threadgroups

32x32 thread groups
1024x1024 px

Thread group:
32x32 threads



dispatchThreadgroups

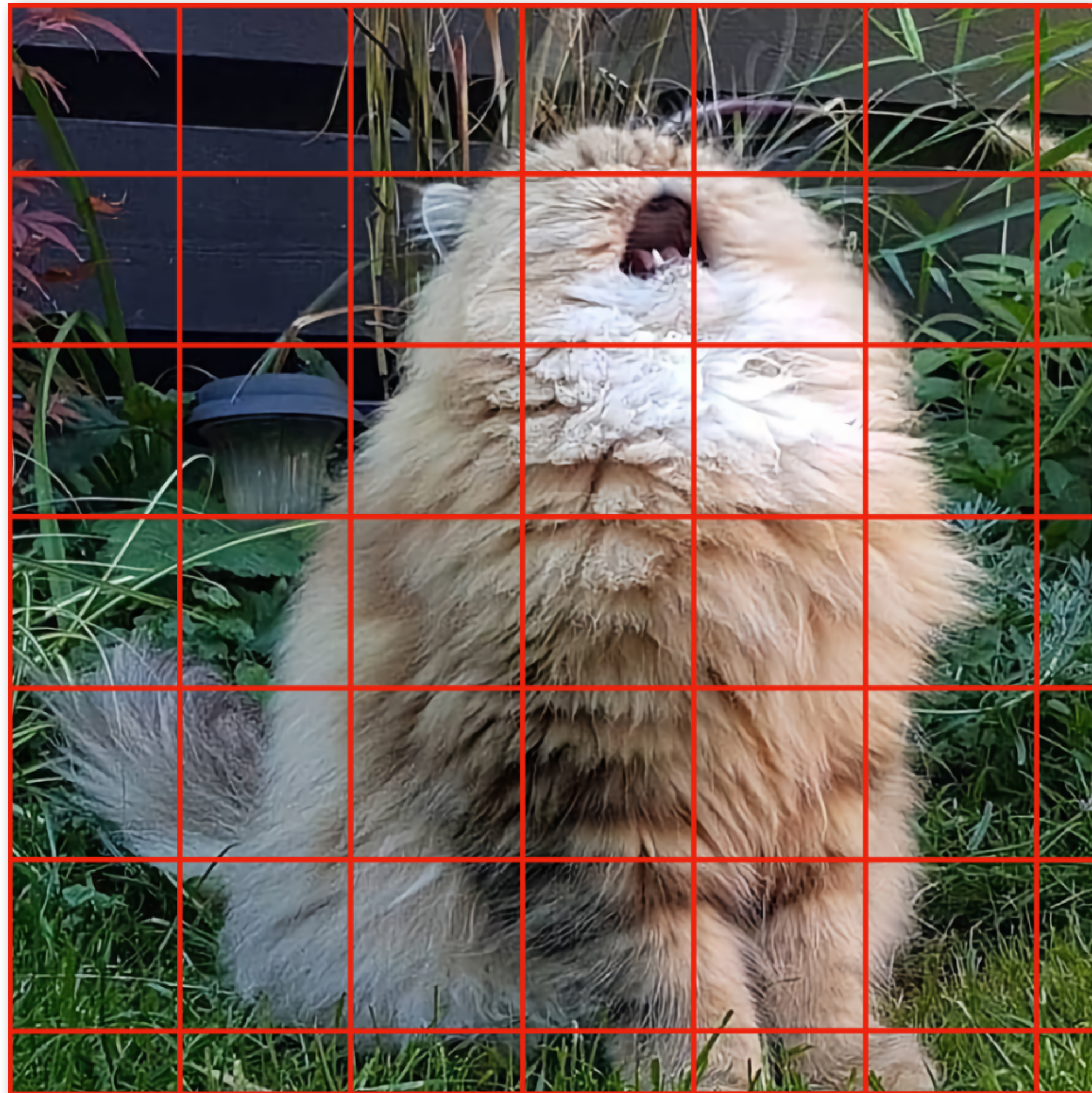


dispatchThreadgroups

```
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;
    }

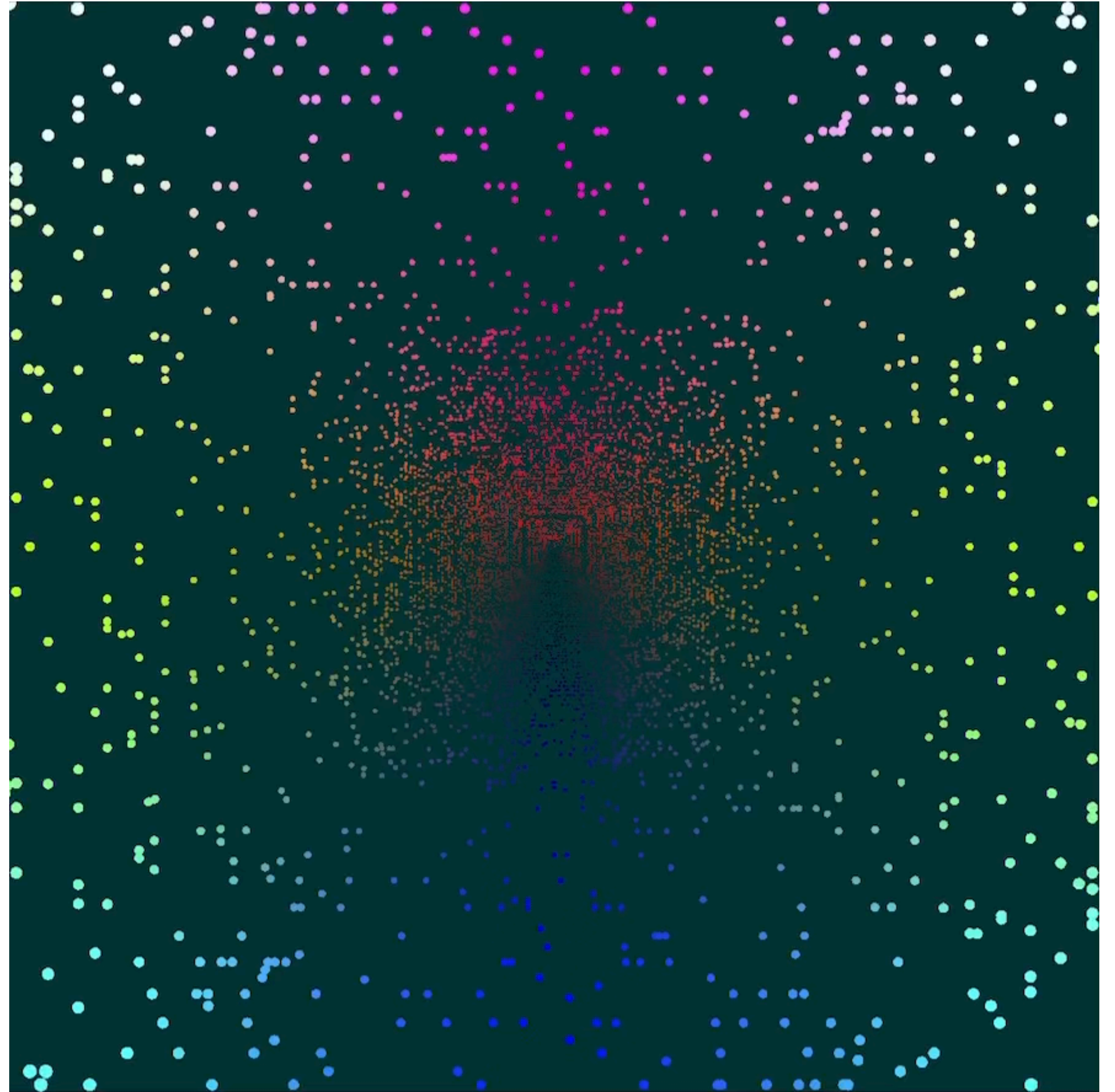
    float4 src = in.read(gid);
    out.write(src * src, gid);
}
```

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

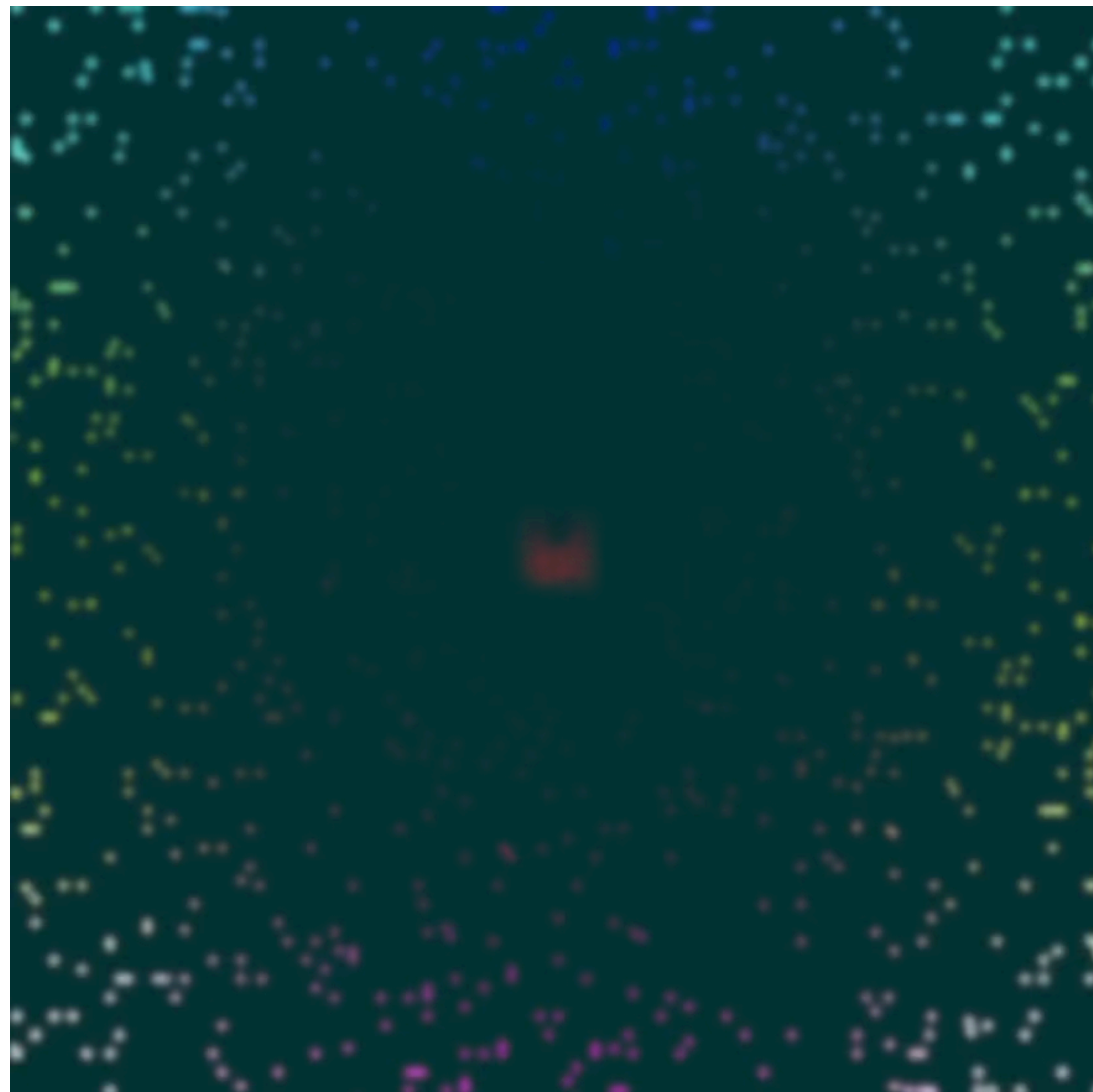
```
import MetalPerformanceShaders

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

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

MPS

Blur on particles



Metal Performance Shaders

Under Hood

Compute 2 0x2838d13b0

- > 41 [dispatchThreadgroups:{36, 18, 1} threadsPerThreadgroup:{32, 32, 1}]
- > 47 [dispatchThreadgroups:{18, 18, 1} threadsPerThreadgroup:{32, 32, 1}]
- > 53 [dispatchThreadgroups:{18, 18, 1} threadsPerThreadgroup:{32, 32, 1}]
- > 59 [dispatchThreadgroups:{18, 18, 1} threadsPerThreadgroup:{32, 32, 1}]
- > 65 [dispatchThreadgroups:{36, 18, 1} threadsPerThreadgroup:{32, 32, 1}]
- > 71 [dispatchThreadgroups:{36, 36, 1} threadsPerThreadgroup:{32, 32, 1}]

D2F6V

D2F6H

F13V

F13H

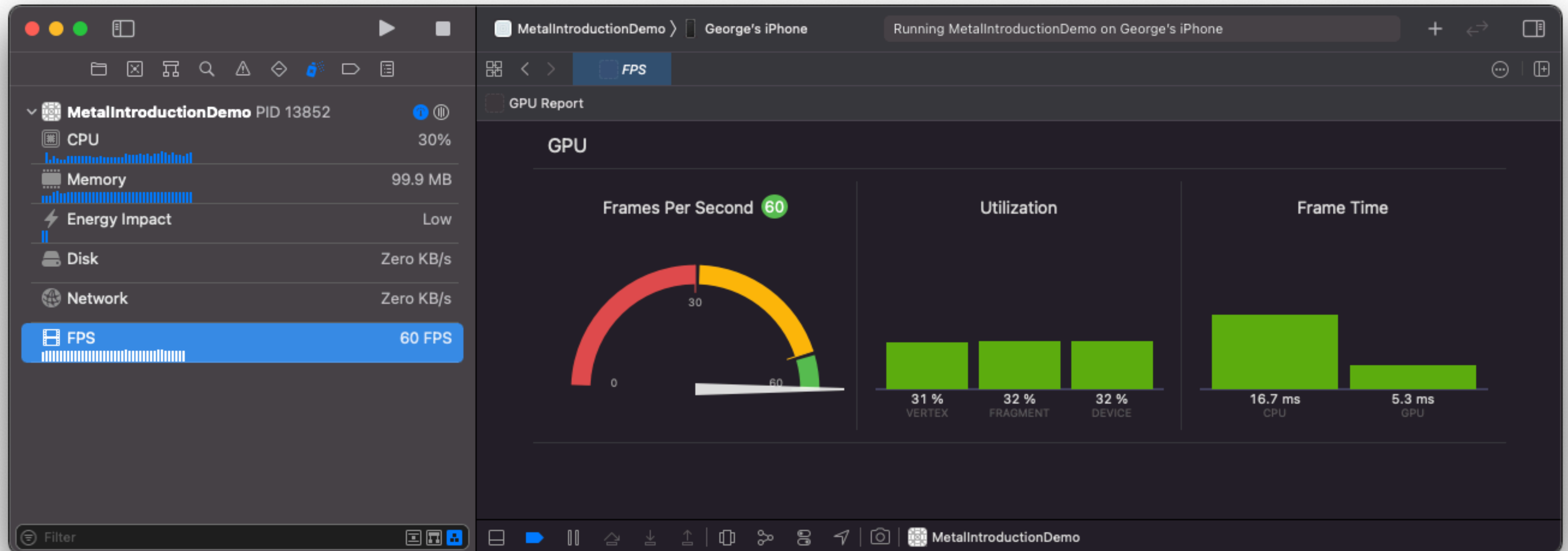
U2P3F3H

U2P3F3V

Tools

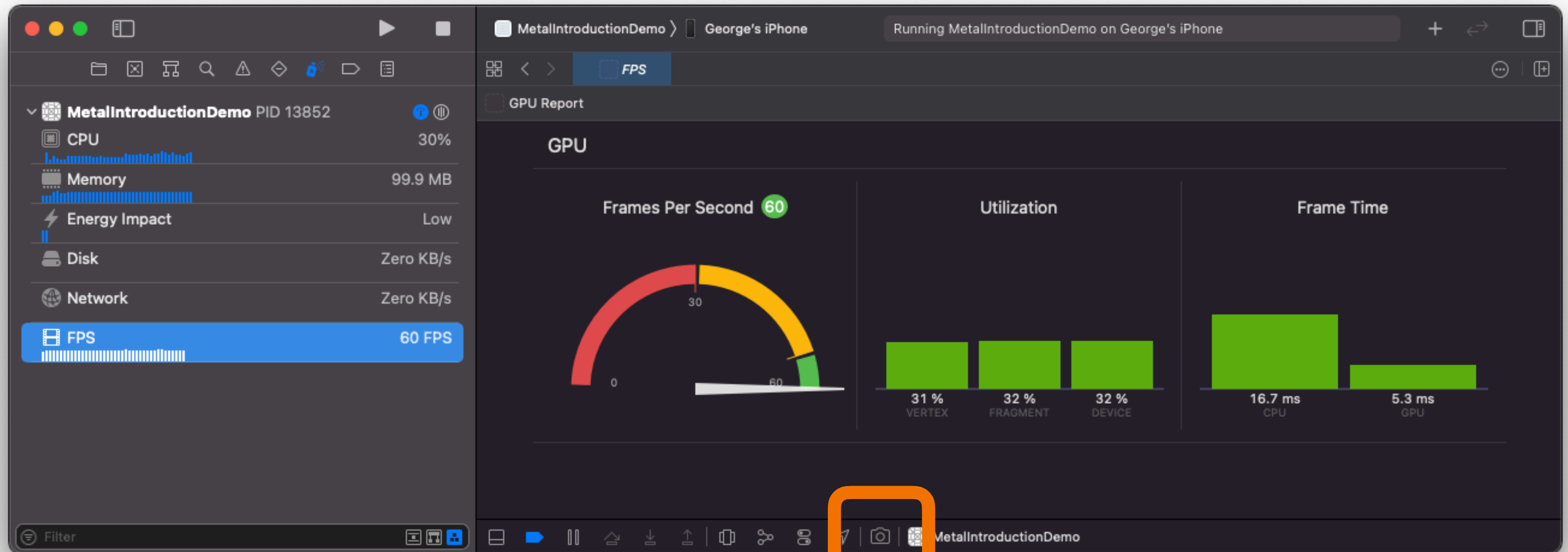
Tools

Xcode



Tools

Xcode Frame Capture



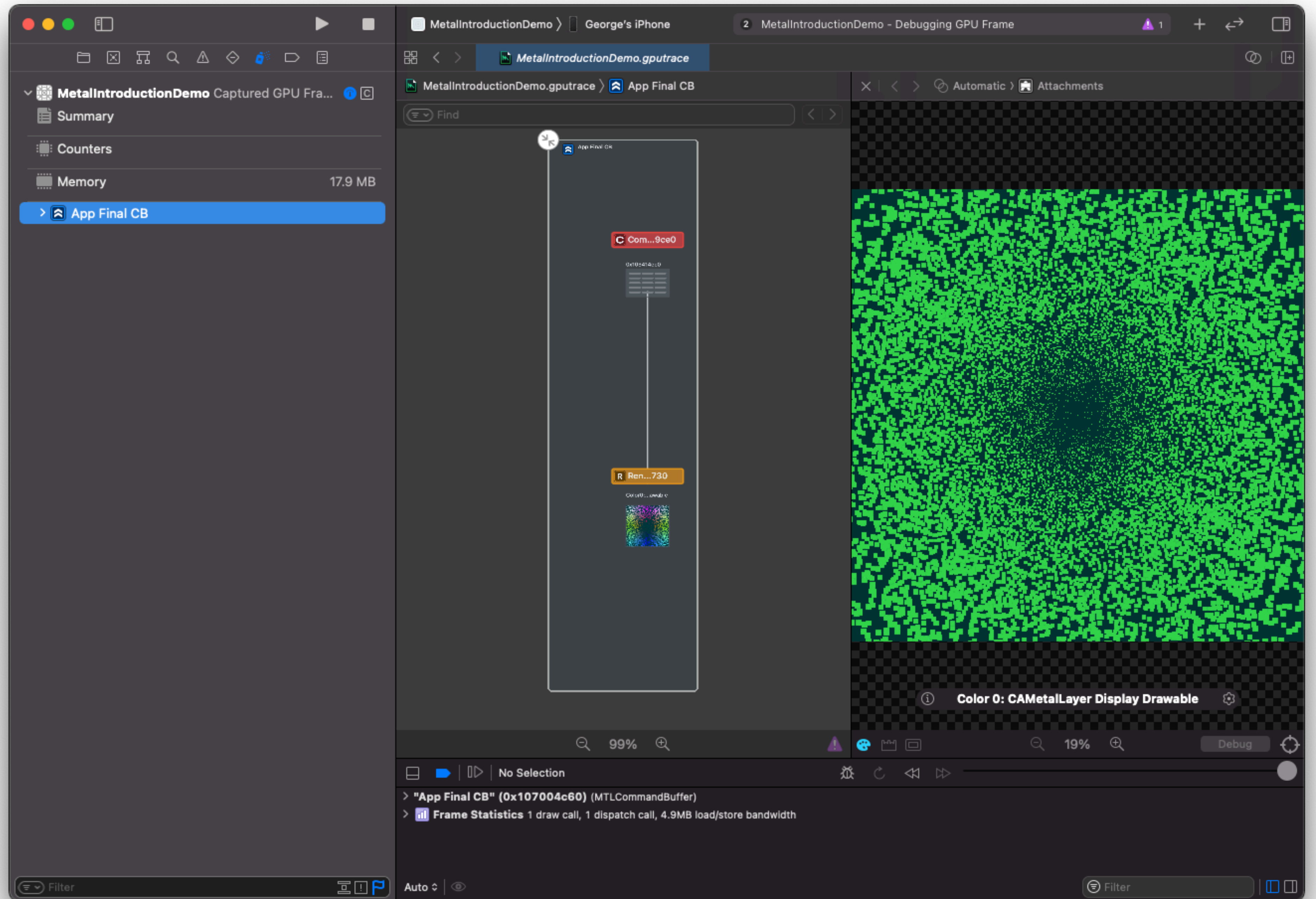
Tools

Xcode
Frame
Capture

The screenshot displays the Xcode GPU Frame Capture tool interface. The top-left sidebar shows the project name 'MetalIntroductionDemo' and the captured GPU frame. The main area is titled 'Summary' and features a large, colorful, abstract visualization of the GPU frame. Below this visualization, the resolution is listed as 1125x1125 and the pixel format as BGRA8Unorm. The right-hand side of the interface contains three summary panels: 'Overview' showing counts for Command Buffers (1), Render Encoders (1), Blit Encoders (0), Compute Encoders (1), Draw Calls (1), and Dispatch Calls (1); 'Performance' showing GPU Time (848.11 μs) and Vertices (16,384); and 'Memory' showing Textures (17.3 MB), Buffers (1.3 MB), and Other (Zero KB). At the bottom, there is an 'Insights' section with tabs for Memory, Bandwidth, Performance, and API, all currently showing 'No Insights'. A 'Related Articles' section is also visible on the right, listing several articles related to memory monitoring and optimization. The bottom of the interface includes a playback control bar with a play button, a 'No Selection' indicator, and a filter bar.

Tools

Xcode
Frame
Capture



Tools

Xcode

Frame

Capture

The screenshot displays the Xcode GPU Frame Capture tool for a Metal introduction demo. The interface is organized into several key sections:

- Left Sidebar:** A tree view showing the capture hierarchy. The selected frame is 12 [drawPrimitives:Point vertexStar...], which is part of a Render 1 command (0x2819a9730) within the App Final Command Buffer (CB).
- Central Table:** A table listing resources for the selected frame. It includes a Vertex section with a Buffer (256 KB), Geometry, and a Vertex Function (vshCompute). It also includes a Fragment section with a Fragment Function (fshCompute) and an Attachments section with a CAMetalLayer Display Drawable.
- Right Preview:** A large window showing the rendered scene, which is a colorful, abstract point cloud or particle system.
- Bottom Timeline:** A horizontal timeline showing the rendering pipeline steps: "ScreenRenderPipeline" (0x103412670), RenderPipeline Performance (830.75 μs, 94.1%), Vertex Buffer 0 (0x103414ee0), Function = (MTLFunction) "vshCompute", and RenderCommandEncoder 1 (0x2819a9730).

Tools

Xcode

Frame

Capture

The screenshot displays the Xcode GPU Frame Capture tool. The left sidebar shows a hierarchy of capture events for a Metal introduction demo. The central pane shows the shader source code for a vertex shader and a fragment shader. The right sidebar provides performance metrics for each line of code, including execution time in microseconds and percentage of total time.

Left Sidebar Hierarchy:

- MetallIntroductionDemo Captured GPU Fra...
 - Summary
 - Counters
 - Memory 17.9 MB
 - App Final CB
 - Compute 0 0x2819a9ce0 15.23 μs
 - No Previews
 - Render 1 0x2819a9730 864.63 μs
 - 9 0x2819a9730 = [renderCommandEncoder...]
 - 12 [drawPrimitives:Point vertexStar... 830.75 μs]

Shader Source Code:

```
19
20
21 vertex ColorInOut vshCompute(unsigned int vid [[vertex_id]],
22                             constant float4 *pos_vel [[buffer(0)]])
23 {
24     ColorInOut out;
25
26     out.position = float4(pos_vel[vid].xy, 0.0, 1.0);
27
28     out.pointSize = length(pos_vel[vid].zw) * 1024;
29
30     float2 velocity = pos_vel[vid].zw * 128;
31     float angle = cos(0.5 * atan2(velocity.x, velocity.y));
32     out.color = abs(float4(angle, velocity.x, velocity.y, 1.0));
33
34     return out;
35 }
36
37
38
39 fragment float4 fshCompute(ColorInOut in [[stage_in]],
40                             float2 pointCoord [[point_coord]])
41 {
42     float2 inPointCoord = pointCoord - 0.5;
43     if (length(inPointCoord) > 0.5) {
44         discard_fragment();
45     }
46     return in.color;
47 }
48
49
50
51 inline static float2 rand2(float2 p) {
52     p = float2(dot(p, float2(127.1, 311.7)),
53               dot(p, float2(269.5, 183.3)));
53 }
```

Performance Metrics:

Line	Time (μs)	Percentage
21-22	483.07	11.16%
26	2.27	2.27%
28	6.57	6.57%
30	8.50	8.50%
31	16.77	16.77%
34	10.86	10.86%
39-40	347.68	347.68 μs
42	2.54	2.54%
43	12.69	12.69%
44	0.07	0.07%
46	11.34	11.34%

Bottom Panel:

- "ScreenRenderPipeline" (0x103412670) vshCompute - fshCompute
- RenderPipeline Performance 830.75 μs (94.1%)
- Vertex Buffer 0 (MTLBuffer) 0x103414ee0
- Function = (MTLFunction) "vshCompute"
- RenderCommandEncoder 1 0x2819a9730 (MTLRenderCommandEncoder)

Tools

Xcode

Frame

Capture

Summary

- App Final CB
 - Compute 0 0x2819a9ce0 15.23 μ s
 - Render 1 0x2819a9730 864.63 μ s
 - 9 0x2819a9730 = [renderCommandEncoder...]
 - 12 [drawPrimitives:Point vertexStar... 830.75 μ s]

Label	Type	Size	Details
Vertex			
Buffer 0x103414ee0	Buffer 0	256 KB	Offset:
Geometry	Post Vertex Tran...		
vshCompute (ScreenRen...	Vertex Function		Library
Fragment			
fshCompute (ScreenRend...	Fragment Function		Library
Attachments			
CAMetalLayer Display Dra...	Color 0	1125 x 1125	BGRA8

Color 0: CAMetalLayer Display Draw...

R: 0.9568627
G: 0.3764706
B: 0.5764706
A: 1.0

Debug

"ScreenRenderPipeline" (0x103412670) vshCompute – fshCompute

RenderPipeline Performance 830.75 μ s (94.1%)

Vertex Buffer 0 (MTLBuffer) 0x103414ee0

Function = (MTLFunction) "vshCompute"

RenderCommandEncoder 1 0x2819a9730 (MTLRenderCommandEncoder)

Tools

Xcode

Frame

Capture

The screenshot displays the Xcode GPU Frame Capture tool interface. On the left, a sidebar shows the 'MetalIntroductionDemo' frame capture with a 'Summary' tab, 'Counters', and 'Memory' (17.9 MB). Below this, a small preview shows a grid of pink and green squares. The main editor area is titled 'MetalIntroductionDemo.gputrace' and shows a Metal shader function 'fshCompute' with the following code:

```
39 fragment float4 fshCompute(ColorInOut in [[stage_in]],
40                        float2 pointCoord [[point_coord]])
41 {
42     float2 inPointCoord = pointCoord - 0.5;
43     if (length(inPointCoord) > 0.5) {
44         discard_fragment();
45     }
46     return in.color;
47 }
48
49
50
51 inline static float2 rand2(float2 p) {
52     p = float2(dot(p, float2(127.1, 311.7)),
53              dot(p, float2(269.5, 183.3)));
```

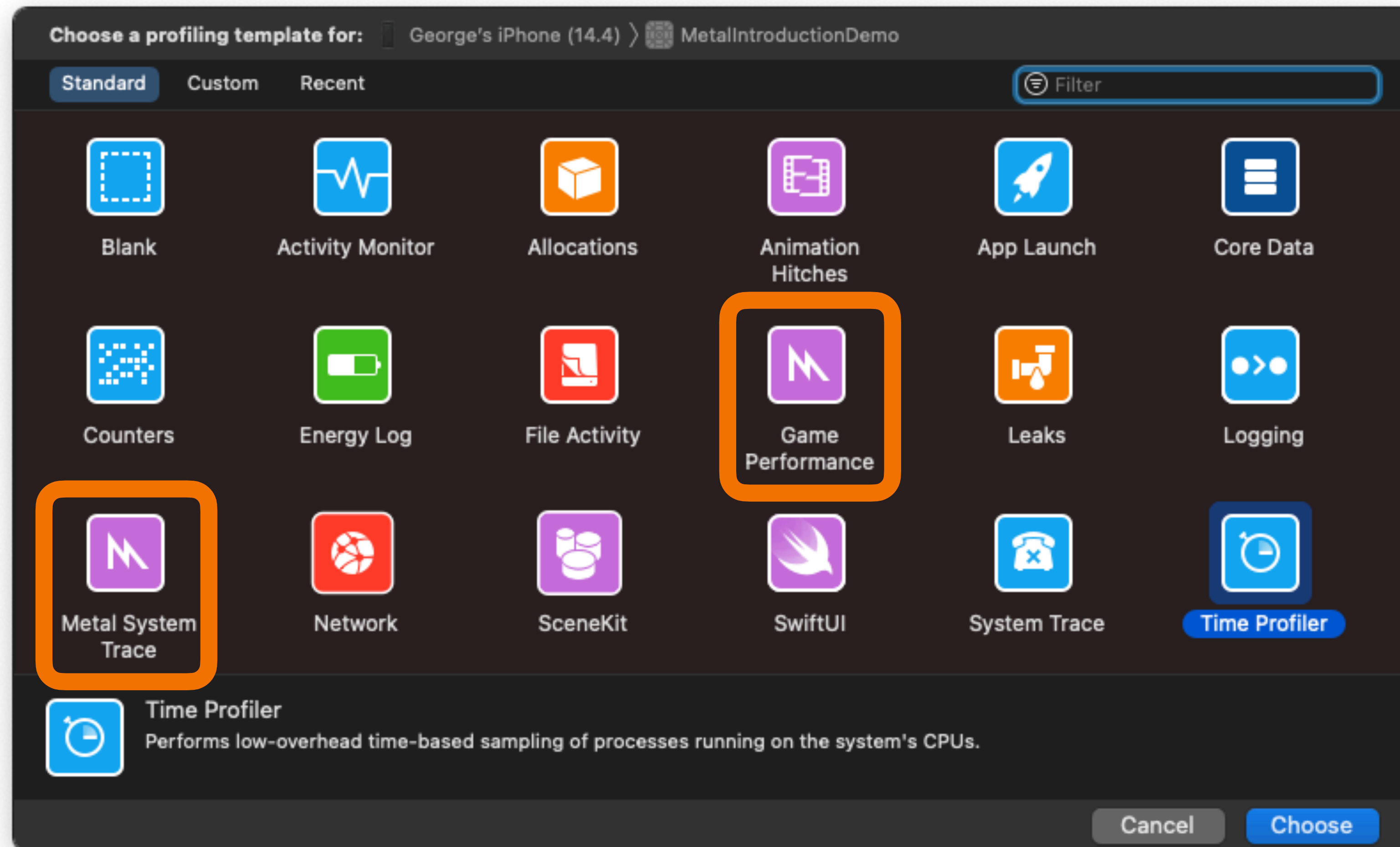
The right side of the editor shows a variable inspector with the following values:

- in = [[399.5, 311.5, 0.0, 1.0]]
- pointCoord = [0.431, 0.614]
- inPointCoord = [-0.069, 0.11]
- ret = [0.959, 0.375, 0.58, 1.0]

Below the code, a visualization shows a heatmap of the fragment's color output and a corresponding 'Mask' image. A tooltip over the heatmap indicates a pixel at (x: 386, y: 293) with values [0] 0.8375 and [1] 0.16513161. The visualization also shows 'Min Value [0.0, 0.01]' and 'Max Value [0.999, 0.993]'. At the bottom, a console shows the current frame's input: '> in = (ColorInOut) [[399.5, 311.5, 0.0, 1.0], 0.0, [0.959, 0.375, 0.58, 1.0]]'.

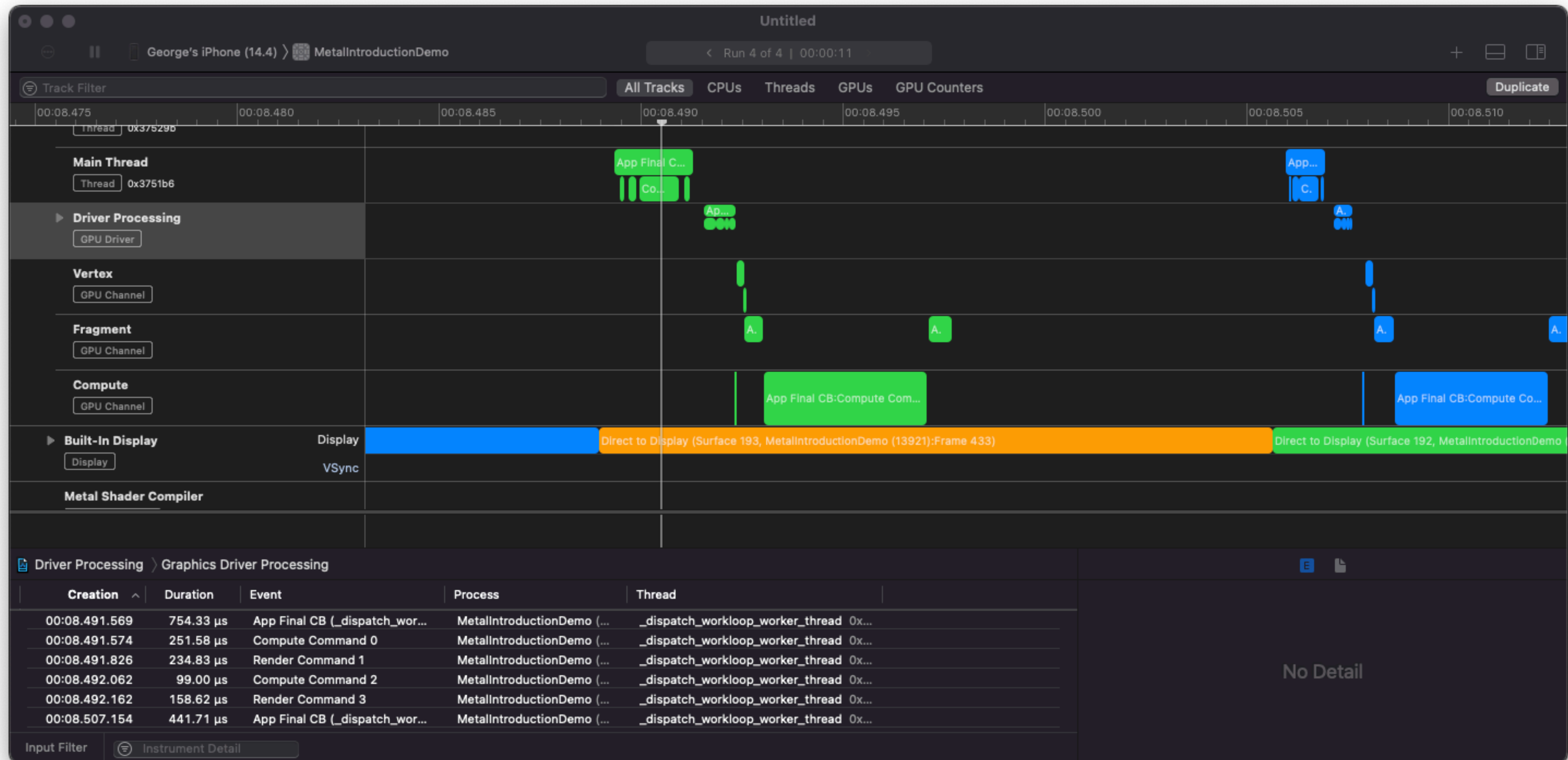
Tools

Xcode Profiling Tools



Tools

Xcode Profiling Tools



Tools

KodeLife

The image shows the KodeLife IDE interface. The main window displays a GLSL fragment shader with the following code:

```
4 struct FS_INPUT
5 {
6     float3 v_normal;
7     float2 v_texcoord;
8 };
9
10 struct FS_UNIFORM
11 {
12     float time;
13     float2 resolution;
14     float2 mouse;
15     float3 spectrum;
16 };
17
18 fragment
19 float4 fs_main(
20     FS_INPUT In [[stage_in]],
21     constant FS_UNIFORM& uniform [[buffer(16)]],
22     texture2d<float> prevFrame [[texture(0)]],
23     texture2d<float> prevPass [[texture(1)]],
24     texture2d<float> texture0 [[texture(2)]],
25     texture2d<float> texture1 [[texture(3)]],
26     texture2d<float> texture2 [[texture(4)]],
27     texture2d<float> texture3 [[texture(5)]],
28     sampler prevFrameSmplr [[sampler(0)]],
29     sampler prevPassSmplr [[sampler(1)]],
30     sampler texture0Smplr [[sampler(2)]],
31     sampler texture1Smplr [[sampler(3)]],
32     sampler texture2Smplr [[sampler(4)]],
33     sampler texture3Smplr [[sampler(5)]]
34 )
35 {
36     float2 uv = -1. + 2. * In.v_texcoord;
37     float4 col = float4(
38         abs(sin(cos(uniform.time+3.*uv.y)*2.*uv.x+uniform.time)),
39         abs(cos(sin(uniform.time+2.*uv.x)*3.*uv.y+uniform.time)),
40         uniform.spectrum.x * 100.,
41         1.0);
42     return col;
43 }
```

The right sidebar shows the Shader Stage configuration for the 'Fragment' stage. It includes a 'Properties' section with 'Name' set to 'Fragment' and 'Enabled' checked. Below that, there are two 'Parameters' sections: 'Previous Frame' and 'Previous Pass'. Each section has a table of variables and samplers. The 'Previous Frame' section shows 'Resolution' with values 320 and 240, and an 'Image' preview showing a colorful, abstract pattern. The 'Previous Pass' section shows 'Render Pass' set to 'Previous' and 'Attachment' set to 'Color'. At the bottom, there are four 'Texture' entries (Texture 1 to Texture 4) with expandable arrows and close buttons.

The bottom-left window, titled 'Output', shows a large preview of the rendered image, which is a colorful, abstract pattern with a central green and yellow area, surrounded by red and blue regions, resembling a stylized, distorted image.

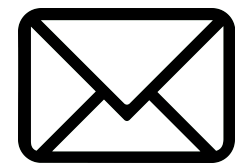
Links

- [**Metal Shading Language Specification**](#)
- [**Metal Feature Set Tables**](#)
- [Metal Programming Guide](#)
- [Metal Best Practices Guide](#)
- [Debug Metal](#)

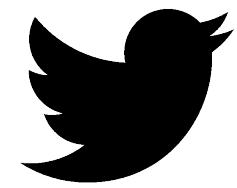
- [Demo app](#)



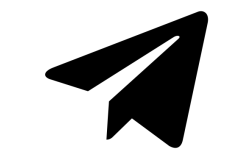
Contacts



wdfeffects@gmail.com



[@GOstrobrod](https://twitter.com/GOstrobrod)



[@GOstrobrod](https://t.me/GOstrobrod)



<https://wdfteam.gitlab.io>