3D Graphics with Metal

A very rapid introduction

https://github.com/cochrane/MetalDemo

Torsten Kammer - @zcochrane

What is Metal?

- New 3D Graphics API for iOS, Mac OS X and related
- An interface to write code and let it run on your GPU
  - With 3D graphics as a highly optimised special use case
- Low-level

Alternatives

Metal is difficult (but fun)

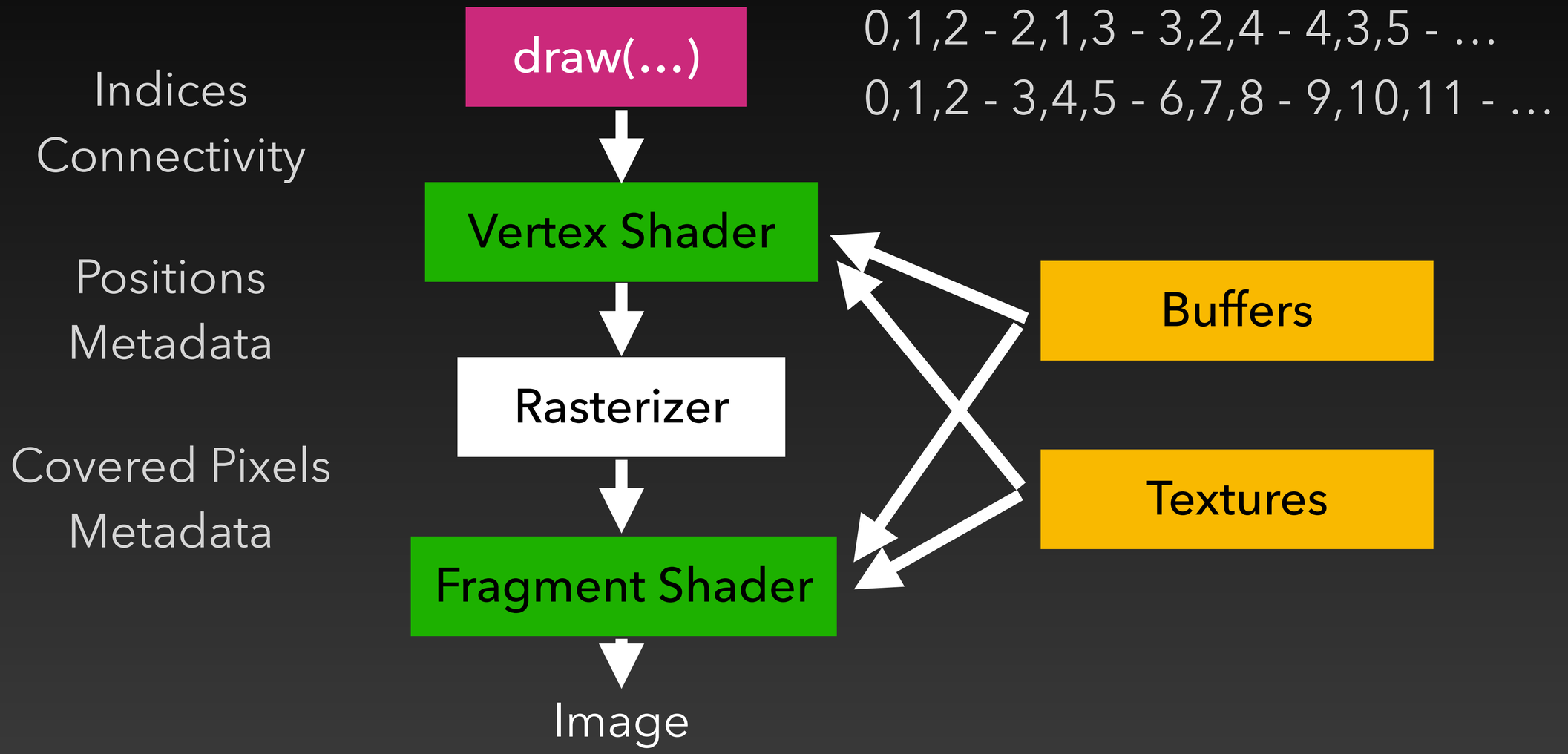
- ~~OpenGL~~
- SceneKit
- Unity or Unreal

- Vulkan via MoltenVK

MetalKit

- MTKView
  - Handles setup, boiler plate, render loop
  - Requires MTKViewDelegate to actually draw
  - AppKit and UIKit
- MTKTextureLoader
  - Loads textures from all formats

Basics: How drawing works



Shaders

- Vertex or Fragment shaders
- (Or others, but no Geometry)
- Written in C++ with extra rules
- .metal file
- Converted to binary, compiled at runtime

```
constant int texCoordSetEmission [[ function_constant(100 + GLLFragmentArgumentIndexTextureEmission) ]];
constant int texCoordSetBump [[ function_constant(100 + GLLFragmentArgumentIndexTextureBump) ]];
constant int texCoordSetBump1 [[ function_constant(100 + GLLFragmentArgumentIndexTextureBump1) ]];
constant int texCoordSetBump2 [[ function_constant(100 + GLLFragmentArgumentIndexTextureBump2) ]];
constant int texCoordSetMask [[ function_constant(100 + GLLFragmentArgumentIndexTextureMask) ]];
constant int texCoordSetLightmap [[ function_constant(100 + GLLFragmentArgumentIndexTextureLightmap) ]];
// reflection tex coord is calculated dynamically that's the whole point

struct XnaLaraFragmentArguments {
    texture2d<float> diffuseTexture [[ id(GLLFragmentArgumentIndexTextureDiffuse) ]];
    texture2d<float> specularTexture [[ id(GLLFragmentArgumentIndexTextureSpecular) ]];
    texture2d<float> emissionTexture [[ id(GLLFragmentArgumentIndexTextureEmission) ]];
    texture2d<float> bumpTexture [[ id(GLLFragmentArgumentIndexTextureBump) ]];
    texture2d<float> bump1Texture [[ id(GLLFragmentArgumentIndexTextureBump1) ]];
    texture2d<float> bump2Texture [[ id(GLLFragmentArgumentIndexTextureBump2) ]];
    texture2d<float> maskTexture [[ id(GLLFragmentArgumentIndexTextureMask) ]];
    texture2d<float> lightmapTexture [[ id(GLLFragmentArgumentIndexTextureLightmap) ]];
    texture2d<float> reflectionTexture [[ id(GLLFragmentArgumentIndexTextureReflection) ]];

    // Color for ambient lighting
    float4 ambientColor [[ id(GLLFragmentArgumentIndexAmbientColor) ]];
    // Color for diffuse lighting
    float4 diffuseColor [[ id(GLLFragmentArgumentIndexDiffuseColor) ]];
    // Color for specular lighting
    float4 specularColor [[ id(GLLFragmentArgumentIndexSpecularColor) ]];

    // Exponent for specular lighting
    float specularExponent [[ id(GLLFragmentArgumentIndexSpecularExponent) ]];
    // Scale for first detail bump map, used by some XNALara shaders
    float bump1UVScale [[ id(GLLFragmentArgumentIndexBump1UVScale) ]];
    // Scale for second detail bump map, used by some XNALara shaders
    float bump2UVScale [[ id(GLLFragmentArgumentIndexBump2UVScale) ]];
    // Scale for specular texture
    float specularTextureScale [[ id(GLLFragmentArgumentIndexSpecularTextureScale) ]];
    // Degree to which reflection (e.g. from environment map) gets blended in
    float reflectionAmount [[ id(GLLFragmentArgumentIndexReflectionAmount) ]];
};

fragment float4 xnaLaraFragment(XnaLaraRasterizerData in [[ stage_in ]],
        device XnaLaraFragmentArguments & arguments [[ buffer(GLLFragmentBufferIndexArguments) ]],
        device GLLLightsBuffer & lights [[ buffer(GLLFragmentBufferIndexLights) ]],
        sampler textureSampler [[ sampler(0) ]],
        depth2d<float> depthPeelFrontBuffer [[ texture(GLLFragmentArgumentIndexTextureDepthPeelFront), function_constant(hasDepthPeelFrontB
        uint currentSample [[ sample_id ]]) {

    if (hasDepthPeelFrontBuffer) {
        float depth = in.position.z;
        uint2 coords = uint2(in.position.xy);
        float frontDepth = depthPeelFrontBuffer.read(coords);
        if (depth <= frontDepth) {
            discard_fragment();
            return float4(1, 0, 0.5, 1);
        }
    }

    // Calculate diffuse color
    float4 diffuseTextureColor = float4(1);
    if (hasDiffuseTexture && hasTexCoord0) {
        diffuseTextureColor = arguments.diffuseTexture.sample(textureSampler, in.texCoordFor(texCoordSetDiffuse));
    }
    float4 usedDiffuseColor = diffuseTextureColor;
    if (hasVertexColor) {
        usedDiffuseColor *= in.color;
    }

    // If Shadeless then return just this diffuse color
    if (isShadeless) {
        return usedDiffuseColor;
    }

    // Calculate normal
    float3 normal;
    if (hasNormal) {
        if (calculateTangentToWorld) {
            float3 normalMapColor = arguments.bumpTexture.sample(textureSampler, in.texCoordFor(texCoordSetBump)).xyz;
            if (hasNormalDetailMap) {
                float2 maskColor = arguments.maskTexture.sample(textureSampler, in.texCoordFor(texCoordSetMask)).xy;
```

Resources

Buffers and Textures

- Buffer: Holds arbitrary bytes
- Texture: Images to read from and draw to
  - Highly optimised
- Storage modes: Shared, managed, private…

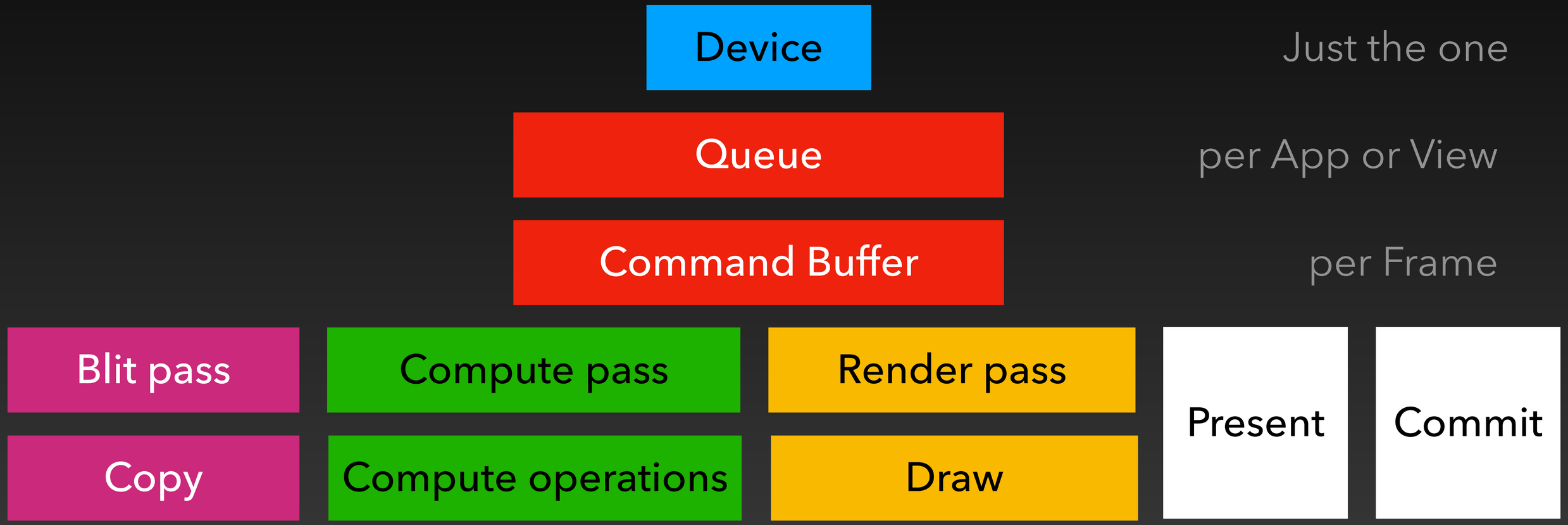
Descriptors

- Problem: NSBitmapImageRep init(bitmapDataPlanes:pixelsWide:pixelsHigh:bitsP
- Solution:

```
let descriptor = MTKThingDescriptor()
descriptor.size = 12
descriptor.format = .rgb
descriptor.label = "thing\(number)"
let actualThing = queue.makeThing(descriptor: descriptor)
```

High-Level Operations

Simple case



Drawing a frame

- draw(in:) on MTKViewDelegate
- Create Command Buffer
- Create (final) render pass with descriptor from view
- Draw in it
- Present Image in Command Buffer
- Commit Command Buffer

Example

https://github.com/cochrane/MetalDemo

I was promised 3D

- Matrix maths!
- Typically three matrices:
  - Model
  - View (Camera)
  - Projection
- Requires custom maths
- Apple provides types

$$\begin{pmatrix} \text{near/xmax} & 0 & 0 & 0 \\ 0 & \text{near/ymax} & 0 & 0 \\ 0 & 0 & \text{-(far+near)/(far-near)} & \text{-(2*far*near)/(far-near)} \\ 0 & 0 & \text{-1} & 0 \end{pmatrix}$$

Things in front of other things

- Depth Buffer
- Can do via MTKView
- This time: Do by hand just to show how it’s done

Example

https://github.com/cochrane/MetalDemo

Textures

- Image to put on things
- Load via MTLTextureLoader
  - Or from previous render pass…
- Automatic smooth sampling
- Use color values directly or as input

Example

https://github.com/cochrane/MetalDemo