逻辑教育
Logic education

# Hello CC

Metal 主题【1】

## 视觉班—iOS 中 Metal 初探

课程研发:CC老师
课程授课:CC老师

# 什么是Metal？

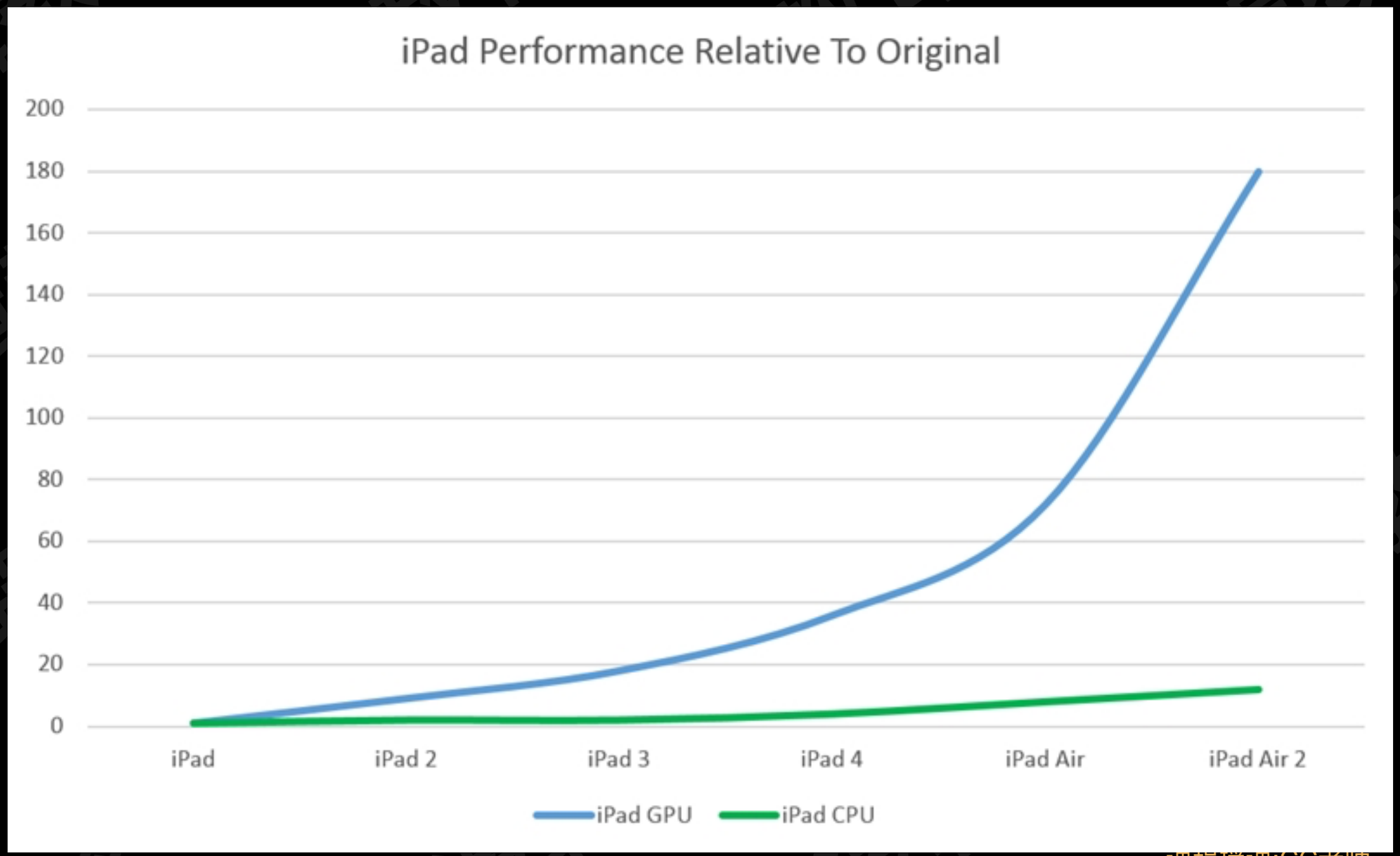
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# Metal 与 OpenGL ES 区别?

课程研发:CC老师
课程授课:CC老师

# CPU/GPU 迭代



逻辑教育
Logic education

# Metal 的表现

# Metal 苹果官方介绍

## Metal 2

### Accelerating graphics and much more.

Metal 2 provides near-direct access to the graphics processing unit (GPU), enabling you to maximize the graphics and compute potential of your apps on iOS, macOS, and tvOS. Building on an efficient low-overhead architecture with precompiled shaders, fine-grained resource control, and multithreading support, Metal 2 evolves to give the GPU even greater control of its graphics pipeline, accelerate neural network training, and provide powerful new tools that give deep insight into your shader code.

# Metal 官方文档介绍





## Fundamental Concepts

> IMPORTANT
>
> This document is no longer being updated. For the latest information about Apple SDKs, visit the documentation website.

Metal provides the lowest-overhead access to the GPU, enabling you to maximize the graphics and compute potential of your app on iOS, macOS, and tvOS. Every millisecond and every bit is integral to a Metal app and the user experience–it's your responsibility to make sure your Metal app performs as efficiently as possible by following the best practices described in this guide. Unless otherwise stated, these best practices apply to all platforms that support Metal.

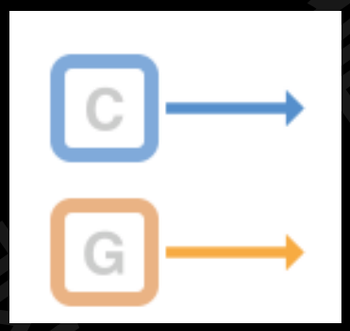
An efficient Metal app requires:


**Low CPU overhead.** Metal is designed to reduce or eliminate many CPU-side performance bottlenecks. Your app can benefit from this design only if you use the Metal API as recommended.

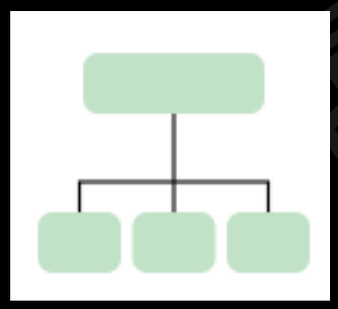
# Metal 优化


**Low CPU overhead.** Metal is designed to reduce or eliminate many CPU-side performance bottlenecks. Your app can benefit from this design only if you use the Metal API as recommended.


**Optimal GPU performance.** Metal allows you to create and submit commands to the GPU. To optimize GPU performance, your app should optimize the configuration and organization of these commands.




**Continuous processor parallelism.** Metal is designed to maximize CPU and GPU parallelism. Your app should keep these processors busy and working simultaneously.



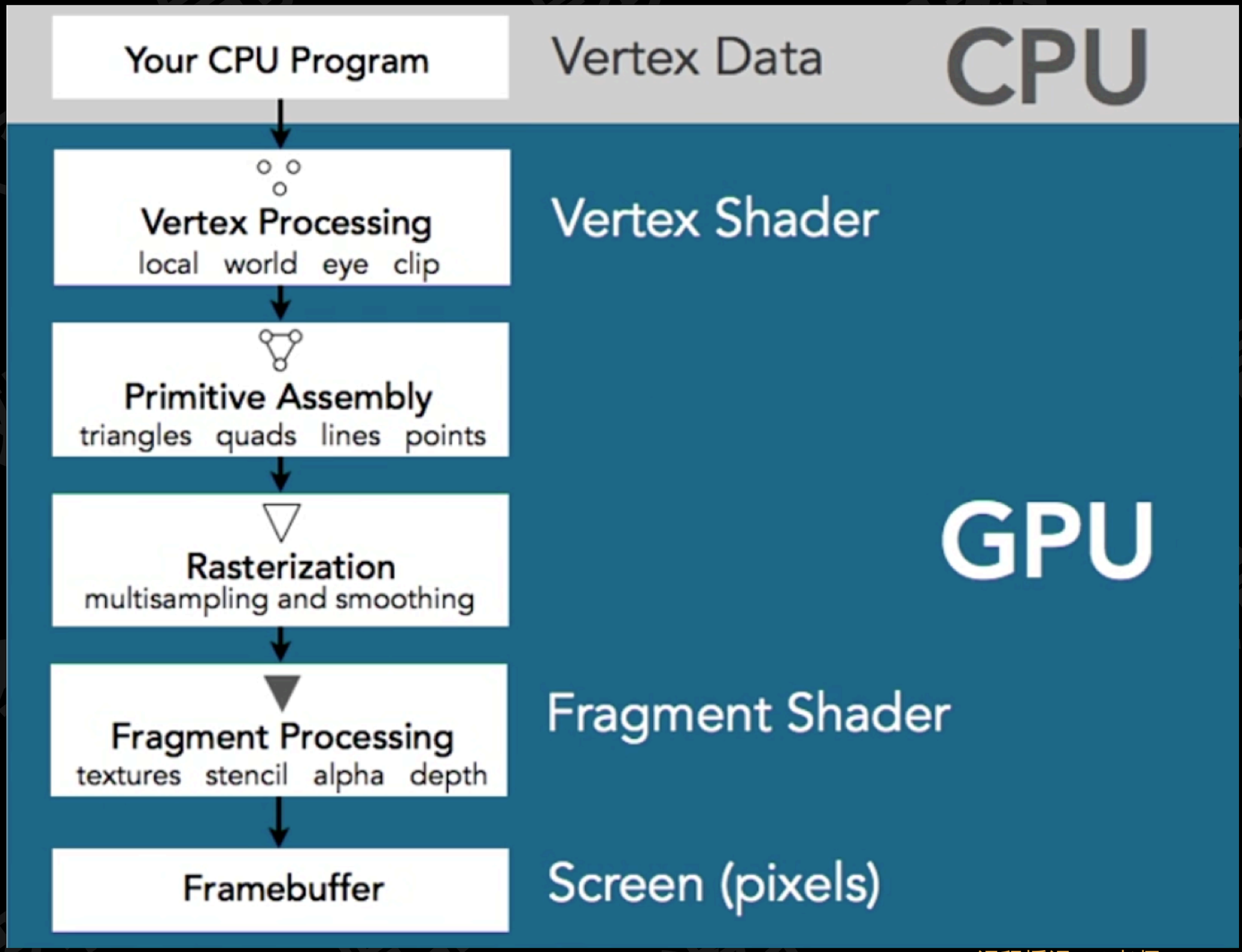
**Effective resource management.** Metal provides simple yet powerful interfaces to your resource objects. Your app should manage these resources effectively to reduce memory consumption and increase access speed.

# graphics pipeline

# 图形管道

课程研发:CC老师
课程授课:CC老师

# graphics pipeline



# Apple 建议: Separate Your Rendering Loop

```
_render =[[CCRenderer alloc]initWithMetalKitView:_view];
if (!_render) {
    NSLog(@"Renderer failed initialization");
    return;
}
_view.delegate = _render;
```

课程研发:CC老师
课程授课:CC老师

# Apple 建议: Respond to View Events

```
- (void)mtkView:(nonnull MTKView *)view drawableSizeWillChange:(CGSize)size;

- (void)drawInMTKView:(nonnull MTKView *)view;
```

课程研发:CC老师
课程授课:CC老师

# Apple 建议: Respond to View Events

```
- (void)mtkView:(nonnull MTKView *)view drawableSizeWillChange:(CGSize)size;

- (void)drawInMTKView:(nonnull MTKView *)view;
```

课程研发:CC老师
课程授课:CC老师

# Apple 建议: Metal Command Objects

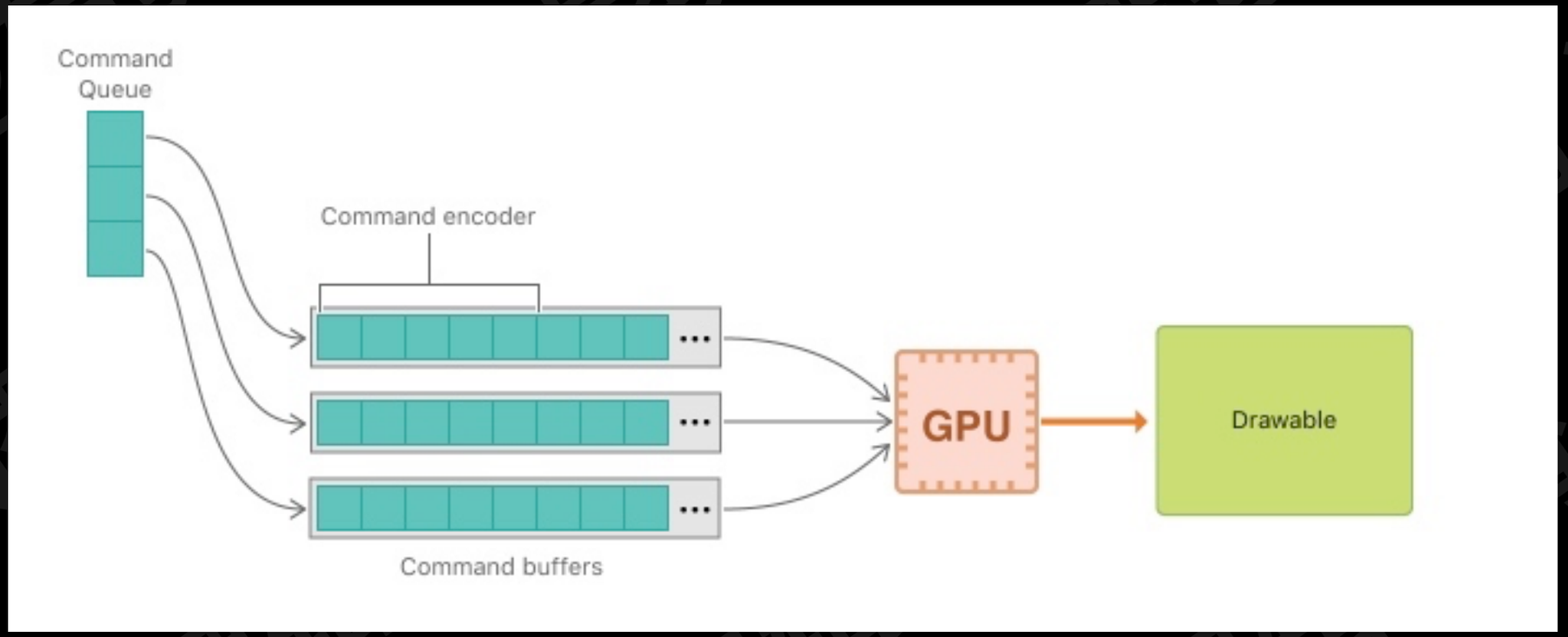
```
//MTLDevice 对象表示GPU.
 _view.device = MTLCreateSystemDefaultDevice();

 //所有应用程序需要与GPU交互的第一个对象是`MTLCommandQueue`对象
_commandQueue = [_device newCommandQueue];


//使用MTLCommandQueue 创建对象并且加入到MTCommandBuffer对象中去.
//为当前渲染的每个渲染传递创建一个新的命令缓冲区
id<MTLCommandBuffer> commandBuffer = [_commandQueue commandBuffer];
commandBuffer.label = @"MyCommand";
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# Metal 命令对象之间的关系

1) 命令缓存区(command buffer) 是从命令队列(command queue) 创建的
2) 命令编码器(command encoders) 将命令编码到命令缓存区中
3) 提交命令缓存区并将其发送到GPU
4) GPU执行命令并将结果呈现为可绘制.




课程研发:CC老师
课程授课:CC老师

# 案例实战(一)

学习目的:

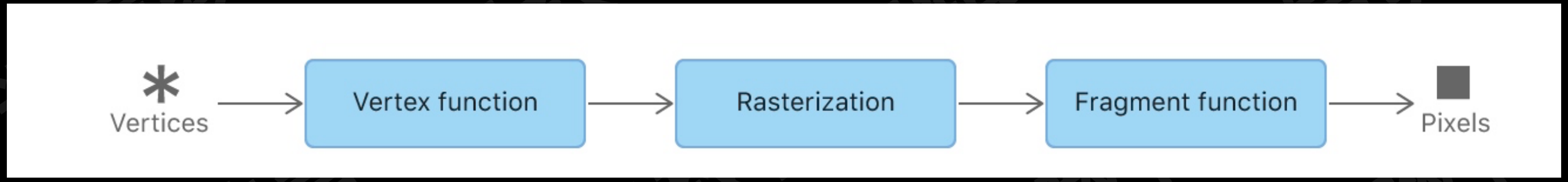
1.理解Metal 应用程序
2.如何向GPU发送基本的渲染命令
3.如何获取Metal 设备
4.如何配置MetalKit 视图
5.如何创建并执行GPU指令
6.显示渲染的内容

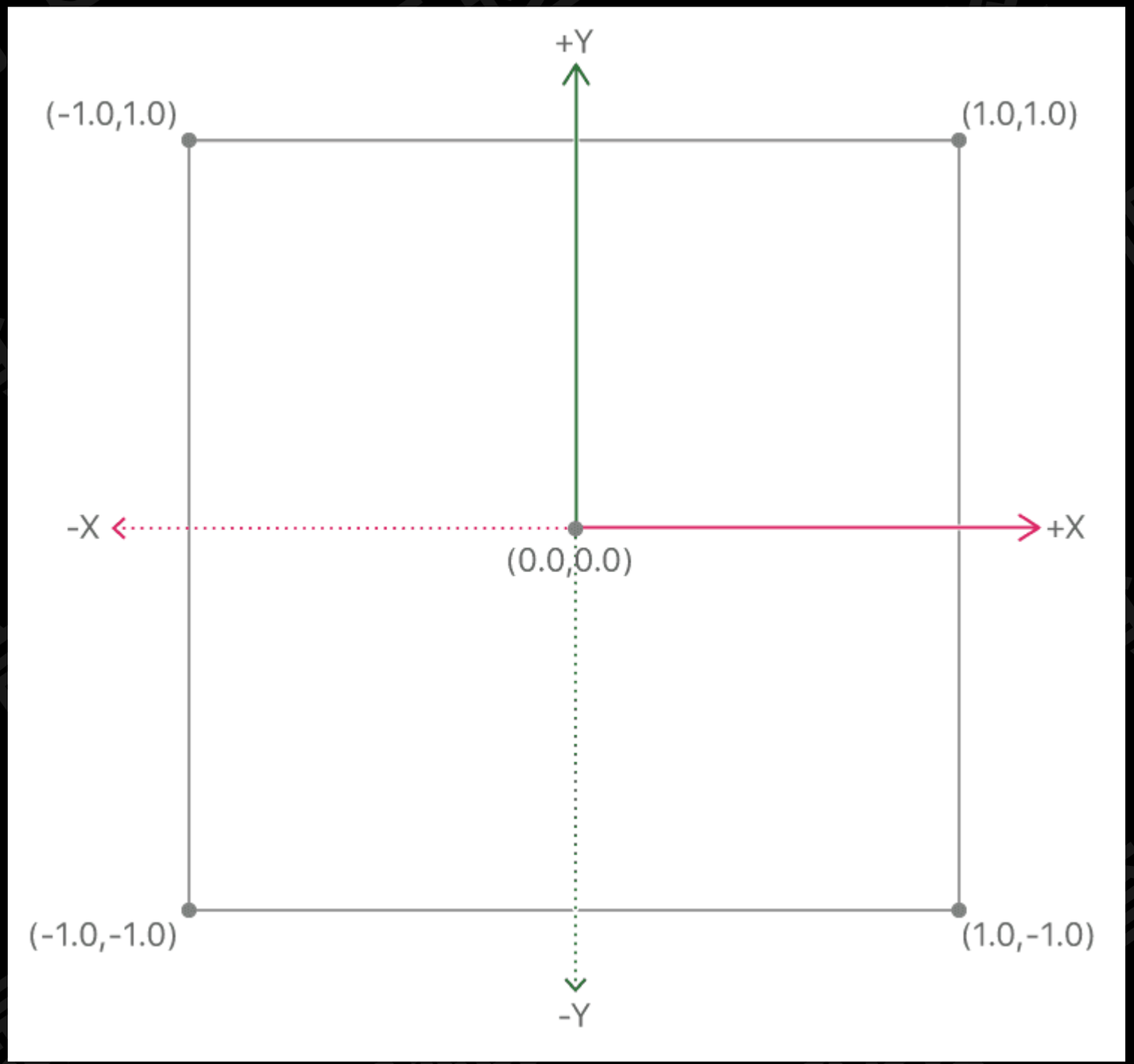
课程研发:CC老师
课程授课:CC老师

# Metal 与 Metal Kit 区别?

课程研发:CC老师
课程授课:CC老师

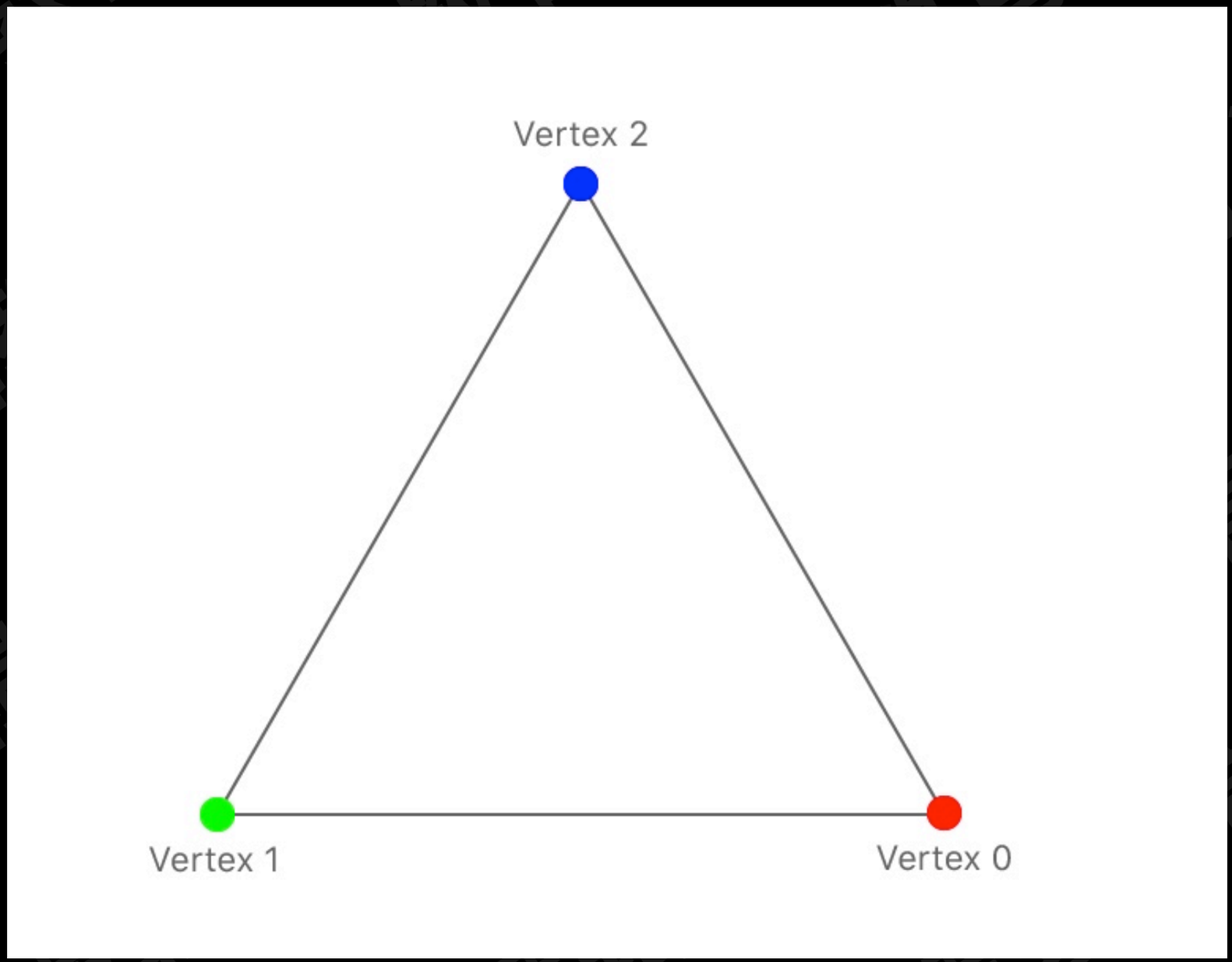
# 渲染管线的三大阶段






课程研发:CC老师
课程授课:CC老师

# 三角形顶点数据




课程研发:CC老师
课程授课:CC老师

# 顶点函数声明

```
vertex RasterizerData
vertexShader(uint vertexID [[vertex_id]],
             constant CCVertex *vertices [[buffer(CCVertexInputIndexVertices)]],
             constant vector_uint2 *viewportSizePointer [[buffer(CCVertexInputIndexViewportSize)]])
```

```
//vertexID：顶点索引                    [[vertex_id]] 属性修饰符
//vertices：顶点数组                    [[buffer(CCVertexInputIndexVertices)]] 属性修饰符
//viewportSizePointer：视口大小         [[buffer(CCVertexInputIndexViewportSize)]] 属性修饰符
```

课程研发:CC老师
课程授课:CC老师

头文件包含了 Metal shaders 与C/OBJC 源之间共享的类型和枚举常数

```
typedef enum CCVertexInputIndex
{
    CCVertexInputIndexVertices     = 0,
    CCVertexInputIndexViewportSize = 1,
} CCVertexInputIndex;


typedef struct
{
    vector_float2 position;
    vector_float4 color;

} CCVertex;
```

课程研发:CC老师
课程授课:CC老师

# 顶点函数返回值

```
vertex RasterizerData
vertexShader(uint vertexID [[vertex_id]],
             constant CCVertex *vertices [[buffer(CCVertexInputIndexVertices)]],
             constant vector_uint2 *viewportSizePointer [[buffer(CCVertexInputIndexViewportSize)]])
```

```
//vertexID：顶点索引                    [[vertex_id]] 属性修饰符
//vertices：顶点数组                    [[buffer(CCVertexInputIndexVertices)]] 属性修饰符
//viewportSizePointer：视口大小         [[buffer(CCVertexInputIndexViewportSize)]] 属性修饰符
```

课程研发:CC老师
课程授课:CC老师

# 顶点函数返回值 结构体定义

```
typedef struct
{
    float4 clipSpacePosition [[position]];

    float4 color;

} RasterizerData;
```

课程研发:CC老师
课程授课:CC老师

# 顶点函数实现

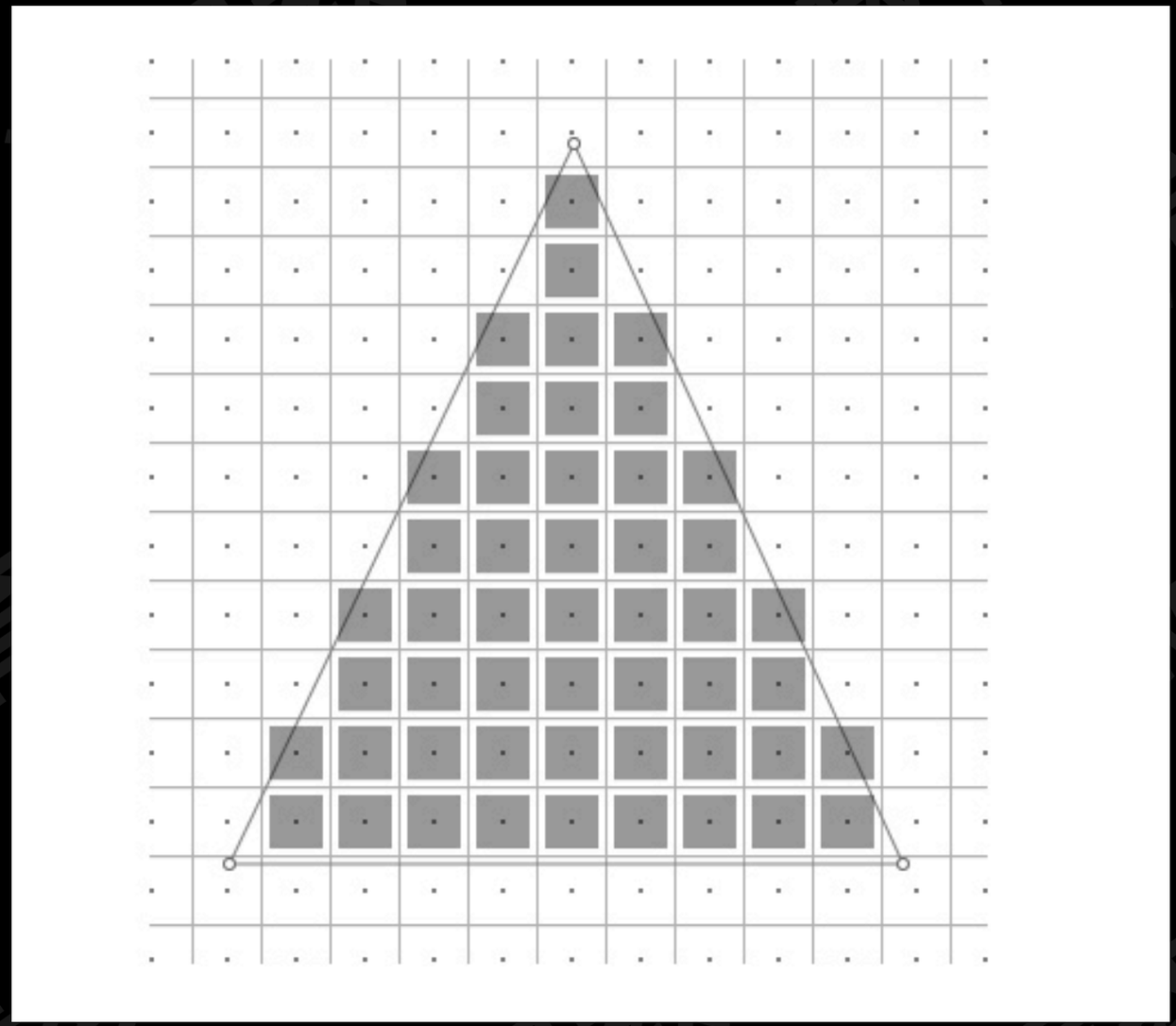
1. 执行坐标系转换，将生成的顶点剪辑空间位置写入`out.clipSpacePosition`返回值。
2. 将顶点颜色传递给`out.color`返回值。

```
RasterizerData out;

out.clipSpacePosition = vector_float4(0.0, 0.0, 0.0, 1.0);
float2 pixelSpacePosition = vertices[vertexID].position.xy;
vector_float2 viewportSize = vector_float2(*viewportSizePointer);
out.clipSpacePosition.xy = pixelSpacePosition / (viewportSize / 2.0);
out.color = vertices[vertexID].color;

return out;
```

课程研发:CC老师
课程授课:CC老师

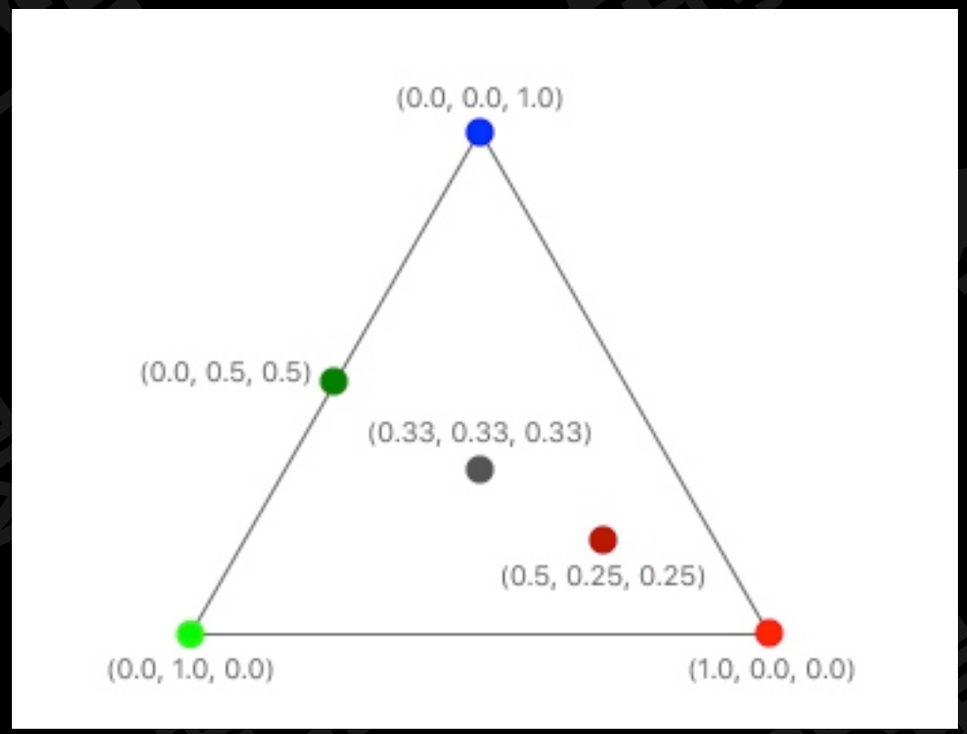
逻辑教育
Logic education

# 光栅化处理



课程研发:CC老师
课程授课:CC老师

# 颜色处理




课程研发:CC老师
课程授课:CC老师

# 片元函数实现

主要任务是处理传入的片段数据并计算可绘制像素的颜色值。

```
fragment float4 fragmentShader(RasterizerData in [[stage_in]])
{
    return in.color;
}
```

课程研发:CC老师
课程授课:CC老师

# 获取函数库并创建管道

```
id<MTLLibrary> defaultLibrary = [_device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];

id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 案例实战(二) Metal 与 OpenGL ES 对比学习

## 使用Metal框架 和 OpenGL ES 框架 完成同样的渲染动作对比?

课程研发:CC老师
课程授课:CC老师