

Trazendo Metal para sua Vida (View)

Ricardo Rachaus

Indice

- Metal?
- Onde usar?
- Animando com Metal

The Roots Of Metal

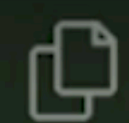
NSBrazil 2019

https://www.youtube.com/watch?v=Qlf_j_Zcpd4

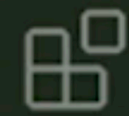
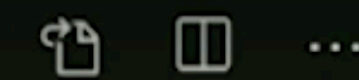


busted

NSBrazil19 - The Roots of Metal



Preview Metal.md ×



Antes de começar, um pequeno aviso:

Você nunca fará:

```
import Metal
```

▶ ⏮ 🔊 1:22 / 30:25





Metal

- API gráfica de baixo nível.
- É o “DirectX da Apple”.
- Metal "significa hardware" e não Heavy Metal.
- UIKit, SwiftUI e SceneKit usam Metal.
- É extremamente rápida.
- Usável com Swift, Objective-C e "C++".

Metal

- Buffers
- Shaders
 - Vertex
 - Fragment
 - Kernel (Compute)
- Comandos
- Renderizar

Onde usar?

Onde usar?

- Criar jogos
- Machine Learning
- Cálculos físicos e matemáticos
- Qualquer tela de devices da Apple
- Efeitos visuais complexos
- Entre outros

Onde usar?

- É fácil de integrar com:
 - SpriteKit
 - SceneKit
 - UIKit
 - SwiftUI
- Ou usar apenas o Metal para processamento

Animando com Metal

Animando com Metal

- Vamos criar uma transição customizada.
- O push da Navigation Controller é muito sem graça.
- Para deixar mais interessante, vamos usar Metal.

16:34 🌙







Doom Screen Melt



https://doom.fandom.com/wiki/Screen_melt

Animando com Metal

- Vamos fazer a transição do Doom.
- Ao fazer o push, a tela “derreterá”.
- O código servirá para qualquer app, não só telas coloridas.
- E agora você poderá fazer o “import Metal” no seu projeto.

Vai ser muito código

Mas não se assuste!


```
extension FirstViewController: UINavigationControllerDelegate {  
  
    func navigationController(  
        _ navigationController: UINavigationController,  
        animationControllerFor operation: UINavigationController.Operation,  
        from fromVC: UIViewController,  
        to toVC: UIViewController  
    ) -> UIViewControllerAnimatedTransitioning? {  
  
    }  
  
}
```

```
extension FirstViewController: UINavigationControllerDelegate {  
  
    func navigationController(  
        _ navigationController: UINavigationController,  
        animationControllerFor operation: UINavigationController.Operation,  
        from fromVC: UIViewController,  
        to toVC: UIViewController  
    ) -> UIViewControllerAnimatedTransitioning? {  
        return DoomTransition()  
    }  
  
}
```

```
class DoomTransition: NSObject, UIViewControllerAnimatedTransitioning {
```

```
}
```

```
class DoomTransition: NSObject, UIViewControllerAnimatedTransitioning {
```

```
let duration: TimeInterval = 2
let view = MetalView(device: device)
let queue = DispatchQueue.main
```

}


```
class DoomTransition: NSObject, UIViewControllerAnimatedTransitioning {  
  
    let duration: TimeInterval = 2  
    let view = MetalView(device: device)  
    let queue = DispatchQueue.main  
  
    func transitionDuration(  
        using transitionContext: UIViewControllerContextTransitioning?  
    ) -> TimeInterval {  
        duration  
    }  
  
}
```

```
class DoomTransition: NSObject, UIViewControllerAnimatedTransitioning {  
  
    let duration: TimeInterval = 2  
    let view = MetalView(device: device)  
    let queue = DispatchQueue.main  
  
    func transitionDuration(  
        using transitionContext: UIViewControllerContextTransitioning?  
    ) -> TimeInterval {  
        duration  
    }  
  
    func animateTransition(  
        using transitionContext: UIViewControllerContextTransitioning  
    ) {  
        /// Aqui inicia a animação  
    }  
  
}
```

animateTransition

animateTransition

```
guard
    let from = transitionContext.viewController(forKey: .from),
    let to = transitionContext.viewController(forKey: .to)
else { return }

let container = transitionContext.containerView
let frame = container.frame
```

animateTransition

```
guard
    let from = transitionContext.viewController(forKey: .from),
    let to = transitionContext.viewController(forKey: .to)
else { return }

let container = transitionContext.containerView
let frame = container.frame

view.frame = CGRect(x: 0, y: 0, width: frame.width, height: frame.height)
container.addSubview(to.view)
container.addSubview(view)
```

animateTransition

```
guard
    let from = transitionContext.viewController(forKey: .from),
    let to = transitionContext.viewController(forKey: .to)
else { return }

let container = transitionContext.containerView
let frame = container.frame

view.frame = CGRect(x: 0, y: 0, width: frame.width, height: frame.height)
container.addSubview(to.view)
container.addSubview(view)

view.fromTexture = from.view.snapshot()
view.toTexture = to.view.snapshot()
```

animateTransition

```
guard
    let from = transitionContext.viewController(forKey: .from),
    let to = transitionContext.viewController(forKey: .to)
else { return }

let container = transitionContext.containerView
let frame = container.frame

view.frame = CGRect(x: 0, y: 0, width: frame.width, height: frame.height)
container.addSubview(to.view)
container.addSubview(view)

view.fromTexture = from.view.snapshot()
view.toTexture = to.view.snapshot()

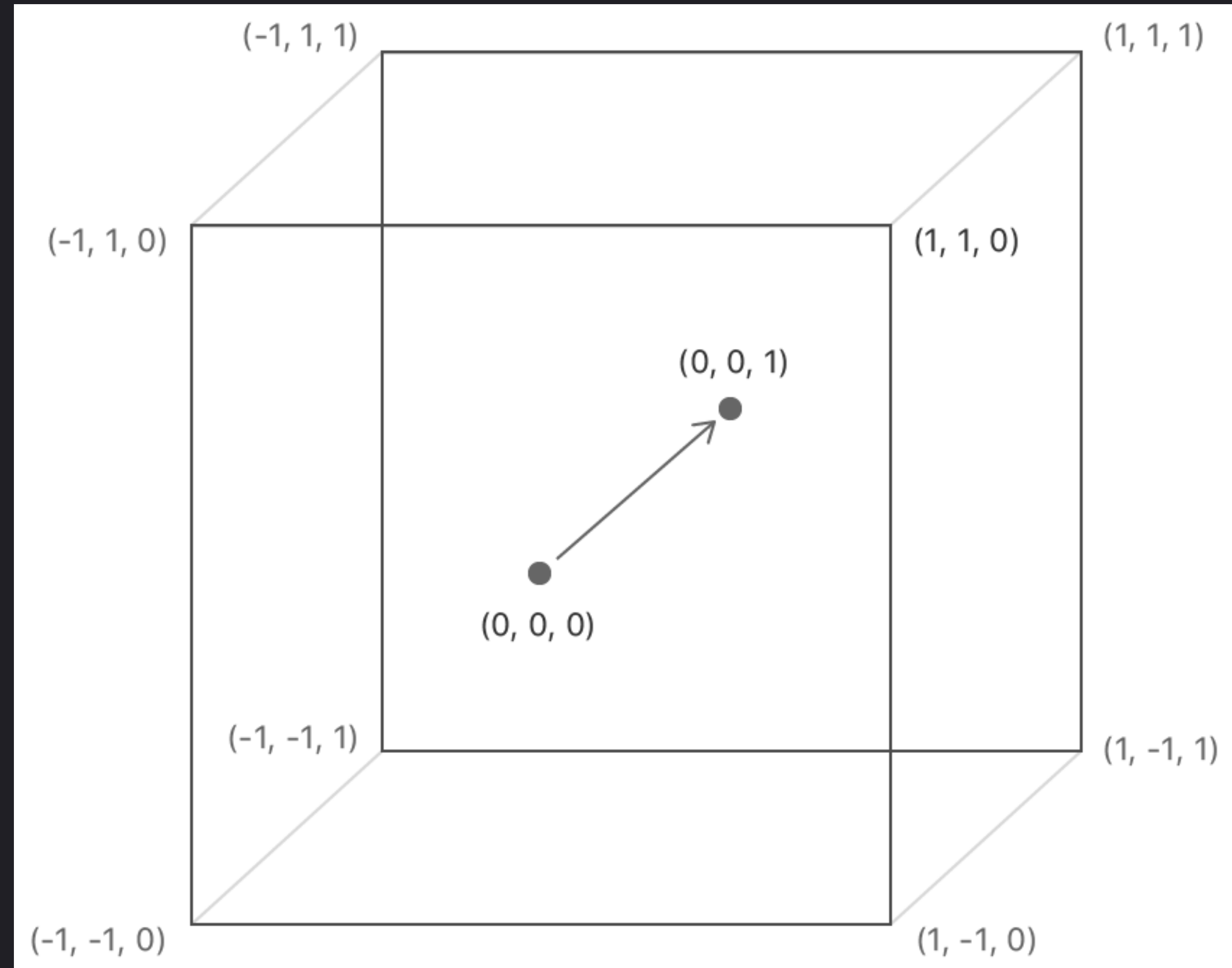
queue.asyncAfter(deadline: .now() + duration) {
    self.view.removeFromSuperview()
    transitionContext.completeTransition(
        !transitionContext.transitionWasCancelled
    )
}
```



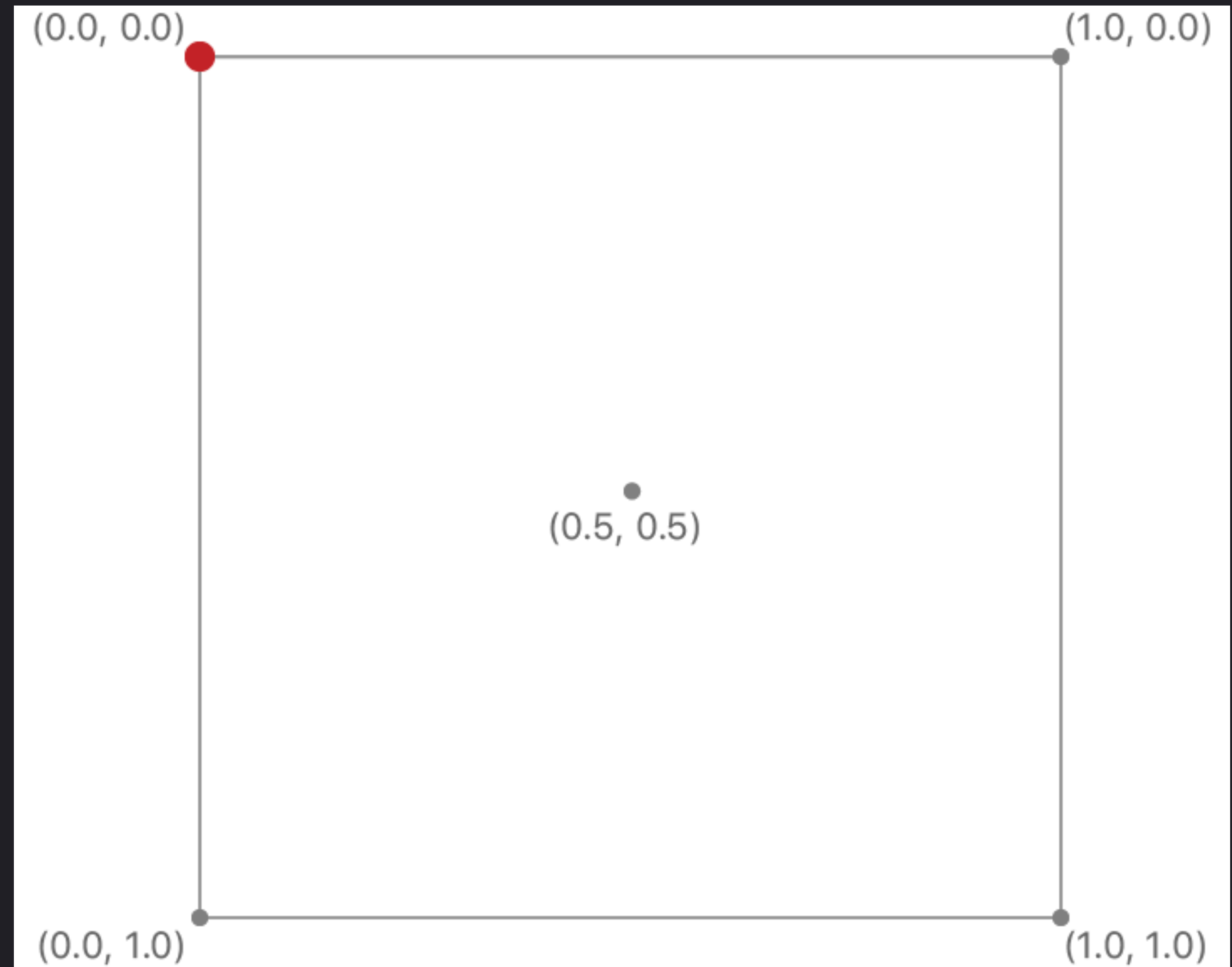
```
final class MetalView: MTKView {  
  
    let positions = [  
        // ...  
    ]  
  
    let textureCoordinate = [  
        // ...  
    ]  
  
    var vertexBuffer: MTLBuffer!  
    var renderPipelineState: MTLRenderPipelineState!  
    var commandQueue: MTLCommandQueue!  
    var sampler: MTLSamplerState!  
  
}
```

```
final class MetalView: MTKView {  
  
    let positions = [  
        // ...  
    ]  
  
    let textureCoordinate = [  
        // ...  
    ]  
  
    var vertexBuffer: MTLBuffer!  
    var renderPipelineState: MTLRenderPipelineState!  
    var commandQueue: MTLCommandQueue!  
    var sampler: MTLSamplerState!  
  
    var fromTexture: MTLTexture?  
    var toTexture: MTLTexture?  
    var deltaTime: Float = 0  
}
```

```
let positions = [  
  // Top Left  
  SIMD3<Float>(-1, 1, 0),  
  // Bottom Left  
  SIMD3<Float>(-1, -1, 0),  
  // Bottom Right  
  SIMD3<Float>( 1, -1, 0),  
  
  // Top Right  
  SIMD3<Float>( 1, 1, 0),  
  // Top Left  
  SIMD3<Float>(-1, 1, 0),  
  // Bottom Right  
  SIMD3<Float>( 1, -1, 0),  
]
```



```
let textureCoordinate = [  
  // Top Left  
  SIMD2<Float>(0, 0),  
  // Bottom Left  
  SIMD2<Float>(0, 1),  
  // Bottom Right  
  SIMD2<Float>(1, 1),  
  
  // Top Right  
  SIMD2<Float>(1, 0),  
  // Top Left  
  SIMD2<Float>(0, 0),  
  // Bottom Right  
  SIMD2<Float>(1, 1),  
]
```



MetalView

```
func createRenderPipelineState() {
```

```
}
```

MetalView

```
func createRenderPipelineState() {  
    let vertexDescriptor = MTLVertexDescriptor()  
    vertexDescriptor.attributes[0].format = .float3  
    vertexDescriptor.attributes[0].bufferIndex = 0  
    vertexDescriptor.attributes[0].offset = 0  
  
}
```

MetalView

```
func createRenderPipelineState() {  
    let vertexDescriptor = MTLVertexDescriptor()  
    vertexDescriptor.attributes[0].format = .float3  
    vertexDescriptor.attributes[0].bufferIndex = 0  
    vertexDescriptor.attributes[0].offset = 0  
    vertexDescriptor.attributes[1].format = .float2  
    vertexDescriptor.attributes[1].bufferIndex = 0  
    vertexDescriptor.attributes[1].offset = MemoryLayout<SIMD3<Float>>.size
```

```
}
```


MetalView

```
func createRenderPipelineState() {  
    let vertexDescriptor = MTLVertexDescriptor()  
    vertexDescriptor.attributes[0].format = .float3  
    vertexDescriptor.attributes[0].bufferIndex = 0  
    vertexDescriptor.attributes[0].offset = 0  
    vertexDescriptor.attributes[1].format = .float2  
    vertexDescriptor.attributes[1].bufferIndex = 0  
    vertexDescriptor.attributes[1].offset = MemoryLayout<SIMD3<Float>>.size  
    vertexDescriptor.layouts[0].stride = MemoryLayout<SIMD3<Float>>.stride  
    + MemoryLayout<SIMD2<Float>>.stride  
  
}
```

MetalView

```
func createRenderPipelineState() {  
    let vertexDescriptor = MTLVertexDescriptor()  
    vertexDescriptor.attributes[0].format = .float3  
    vertexDescriptor.attributes[0].bufferIndex = 0  
    vertexDescriptor.attributes[0].offset = 0  
    vertexDescriptor.attributes[1].format = .float2  
    vertexDescriptor.attributes[1].bufferIndex = 0  
    vertexDescriptor.attributes[1].offset = MemoryLayout<SIMD3<Float>>.size  
    vertexDescriptor.layouts[0].stride = MemoryLayout<SIMD3<Float>>.stride  
    + MemoryLayout<SIMD2<Float>>.stride  
  
    let renderPipelineDescriptor = MTLRenderPipelineDescriptor()  
    renderPipelineDescriptor.colorAttachments[0].pixelFormat = .bgra8Unorm  
    renderPipelineDescriptor.vertexFunction = vertexFunction  
    renderPipelineDescriptor.fragmentFunction = fragmentFunction  
    renderPipelineDescriptor.vertexDescriptor = vertexDescriptor  
  
}
```

MetalView

```
func createRenderPipelineState() {
    let vertexDescriptor = MTLVertexDescriptor()
    vertexDescriptor.attributes[0].format = .float3
    vertexDescriptor.attributes[0].bufferIndex = 0
    vertexDescriptor.attributes[0].offset = 0
    vertexDescriptor.attributes[1].format = .float2
    vertexDescriptor.attributes[1].bufferIndex = 0
    vertexDescriptor.attributes[1].offset = MemoryLayout<SIMD3<Float>>.size
    vertexDescriptor.layouts[0].stride = MemoryLayout<SIMD3<Float>>.stride
    + MemoryLayout<SIMD2<Float>>.stride

    let renderPipelineDescriptor = MTLRenderPipelineDescriptor()
    renderPipelineDescriptor.colorAttachments[0].pixelFormat = .bgra8Unorm
    renderPipelineDescriptor.vertexFunction = vertexFunction
    renderPipelineDescriptor.fragmentFunction = fragmentFunction
    renderPipelineDescriptor.vertexDescriptor = vertexDescriptor

    renderPipelineState = try! device!.makeRenderPipelineState(
        descriptor: renderPipelineDescriptor
    )
}
```

MetalView

```
override fun draw(_ rect: CGRect) {  
    // Antes de chamar drawPrimitives
```

MetalView

```
override fun draw(_ rect: CGRect) {  
    // Antes de chamar drawPrimitives  
  
    deltaTime += 1 / Float(preferredFramesPerSecond)
```

MetalView

```
override fun draw(_ rect: CGRect) {  
    // Antes de chamar drawPrimitives  
  
    deltaTime += 1 / Float(preferredFramesPerSecond)  
  
    commandEncoder?.setVertexBytes(  
        textureCoordinates,  
        length: MemoryLayout<SIMD2<Float>>.stride * textureCoordinates.count,  
        index: 1  
    )  
}
```

MetaView

```
override fun draw(_ rect: CGRect) {  
    // Antes de chamar drawPrimitives  
  
    deltaTime += 1 / Float(preferredFramesPerSecond)  
  
    commandEncoder?.setVertexBytes(  
        textureCoordinates,  
        length: MemoryLayout<SIMD2<Float>>.stride * textureCoordinates.count,  
        index: 1  
    )  
  
    if let fromTexture, let toTexture {  
        commandEncoder?.setFragmentTexture(fromTexture, index: 0)  
        commandEncoder?.setFragmentTexture(toTexture, index: 1)  
        commandEncoder?.setFragmentSamplerState(sampler, index: 0)  
        commandEncoder?.setFragmentBytes(  
            &deltaTime,  
            length: MemoryLayout<Float>.stride,  
            index: 0  
        )  
    }  
}
```

MetalView

```
override fun draw(_ rect: CGRect) {  
    // Antes de chamar drawPrimitives  
  
    deltaTime += 1 / Float(preferredFramesPerSecond)  
  
    commandEncoder?.setVertexBytes(  
        textureCoordinates,  
        length: MemoryLayout<SIMD2<Float>>.stride * textureCoordinates.count,  
        index: 1  
    )  
  
    if let fromTexture, let toTexture {  
        commandEncoder?.setFragmentTexture(fromTexture, index: 0)  
        commandEncoder?.setFragmentTexture(toTexture, index: 1)  
        commandEncoder?.setFragmentSamplerState(sampler, index: 0)  
        commandEncoder?.setFragmentBytes(  
            &deltaTime,  
            length: MemoryLayout<Float>.stride,  
            index: 0  
        )  
    }  
  
    // Renderiza  
}
```


Vertex Shader

Vertex Shader

```
struct VertexOut {
    float4 position [[ position ]];
    float2 textureCoordinate;
};
```

Vertex Shader

```
struct VertexOut {  
    float4 position [[ position ]];  
    float2 textureCoordinate;  
};
```

```
vertex VertexOut main_vertex(  
    in float4 position [[ position ]],  
    in float2 textureCoordinate [[ textureCoordinate ]]) {  
  
}
```

Vertex Shader

[illegible]

Vertex Shader

```
struct VertexOut {  
    float4 position [[ position ]];  
    float2 textureCoordinate;  
};  
  
vertex VertexOut main_vertex(constant float3 *positions [[ buffer(0) ]],  
                             constant float2 *textureCoordinates [[ buffer(1) ]],  
                             uint vertexID [[ vertex_id ]]) {  
    VertexOut out {  
        .position = float4(positions[vertexID], 1),  
        .textureCoordinate = textureCoordinates[vertexID],  
    };  
    return out;  
}
```

Fragment Shader

```
constant float START_SPEED = 2.7;
constant float MELT_SPEED = 1;
```

```
fragment float4 doom_melt(
```

) {

}

Fragment Shader

```
constant float START_SPEED = 2.7;  
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],  
                           texture2d<float> to [[ texture(1) ]],
```

```
) {
```

```
}
```

Fragment Shader

```
constant float START_SPEED = 2.7;  
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],  
                           texture2d<float> to [[ texture(1) ]],  
                           sampler sampler [[ sampler(0) ]],  
                           VertexOut vertexIn [[ stage_in ]],  
                           ) {
```

```
}
```


Fragment Shader

```
constant float START_SPEED = 2.7;  
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],  
                           texture2d<float> to [[ texture(1) ]],  
                           sampler sampler [[ sampler(0) ]],  
                           VertexOut vertexIn [[ stage_in ]],  
                           constant float &deltaTime [[ buffer(0) ]]) {
```

```
}
```

Fragment Shader

```
constant float START_SPEED = 2.7;
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],
                          texture2d<float> to [[ texture(1) ]],
                          sampler sampler [[ sampler(0) ]],
                          VertexOut vertexIn [[ stage_in ]],
                          constant float &deltaTime [[ buffer(0) ]]) {
    float2 uv = vertexIn.textureCoordinate;
    float velocity = START_SPEED * deltaTime;
    if (velocity > 1) velocity = 1;

    uv.y -= velocity * 0.35 * fract(sin(dot(float2(uv.x, 0), float2(12.9898, 78.233))) * 43758.545);

}
```

Fragment Shader

```
constant float START_SPEED = 2.7;
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],
                           texture2d<float> to [[ texture(1) ]],
                           sampler sampler [[ sampler(0) ]],
                           VertexOut vertexIn [[ stage_in ]],
                           constant float &deltaTime [[ buffer(0) ]]) {
    float2 uv = vertexIn.textureCoordinate;
    float velocity = START_SPEED * deltaTime;
    if (velocity > 1) velocity = 1;

    uv.y -= velocity * 0.35 * fract(sin(dot(float2(uv.x, 0), float2(12.9898, 78.233))) * 43758.545);

    if (velocity == 1) uv.y -= MELT_SPEED * (deltaTime - velocity / START_SPEED);

}
```

Fragment Shader

```
constant float START_SPEED = 2.7;
constant float MELT_SPEED = 1;
```

```
/// Adaptation of the Doom Effect shader from: https://www.shadertoy.com/view/XtlyDn
```

```
fragment float4 doom_melt(texture2d<float> from [[ texture(0) ]],
                          texture2d<float> to [[ texture(1) ]],
                          sampler sampler [[ sampler(0) ]],
                          VertexOut vertexIn [[ stage_in ]],
                          constant float &deltaTime [[ buffer(0) ]]) {
    float2 uv = vertexIn.textureCoordinate;
    float velocity = START_SPEED * deltaTime;
    if (velocity > 1) velocity = 1;

    uv.y -= velocity * 0.35 * fract(sin(dot(float2(uv.x, 0), float2(12.9898, 78.233))) * 43758.545);

    if (velocity == 1) uv.y -= MELT_SPEED * (deltaTime - velocity / START_SPEED);

    if (uv.y < 0) {
        return to.sample(sampler, vertexIn.textureCoordinate);
    }

    return from.sample(sampler, uv);
}
```

16:49 🌙



Agora vocês podem:

Graphics Engineer

Apple
(Watch Faces)

[https://jobs.apple.com/en-us/details/
200440795/graphics-engineer](https://jobs.apple.com/en-us/details/200440795/graphics-engineer)



Repositório

<https://github.com/ricardorachaus/metaltransition>



Perguntas?