

# Initialization & Configuration

MTLDevice.CreateSystemDefaultDevice() -> MTLDevice // once in App

## MTLDevice

makeCommandQueue() -> MTLCommandQueue // once in App

## MTLCommandQueue

makeCommandBuffer()  
-> MTLCommandBuffer // per draw

## MTLCommandBuffer

```
addCompletionHandler { _ in semaphore.signal() }  
semaphore.wait()  
// in case of async rendering
```

makeRenderCommandEncoder( descriptor: )  
-> MTLRenderCommandEncoder // per draw

## MTLRenderCommandEncoder

```
setDepthStencilState( _ )  
setRenderPipelineState( _ )  
setCullMode( _ : MTLCullMode (.none, .front, .back) )  
setTriangleFillMode( _ : MTLTriangleFillMode .fill, .lines )  
// .lines is useful for debugging.  
setViewport( _ : MTLViewport )
```

**// All the other encoding functions go here.**

```
present( _ : MTLDrawable ) // in case rendering to a screen  
endEncoding()  
commit()
```

waitUntilCompleted() // in case of synchronized rendering

## MTLViewport

originX: Double **0.0**  
originY: Double **0.0**  
width: Double  
height: Double  
znear: Double **0.0**  
zfar: Double **1.0**

## MTLVertexDescriptor

layouts[i].stride: Int <num bytes for the corresponding attributes>  
layouts[i].stepRate: Int **1**  
layouts[i].stepFunction: MTLVertexStepFunction

**.perVertex** (default)  
**.perPatchControlPoint** (for tessellation)

attributes[j].format: MTLVertexFormat (.float3 for normal etc)  
attributes[j].offset: Int <byte offset in the layout i>  
attributes[j].bufferIndex: Int <i, the index of the layout[i]>

## MTLDepthStencilDescriptor

depthCompareFunction: MTLCompareFunction **.less**, **.always**, ...  
isDepthWriteEnabled: Bool

## MTLDevice

makeDepthStencilState( descriptor: )  
-> MTLDepthStencilState // once in App per render target

## MTLRenderPassDescriptor

colorAttachment[i].texture: MTLTexture  
colorAttachment[i].loadAction: MTLLoadAction **.dontCare**, **.load**, **.clear**  
colorAttachment[i].storeAction: MTLStoreAction **.dontCare**, **.store**, ...  
depthAttachment.texture: MTLTexture.  
depthAttachment.loadAction: MTLLoadAction **.dontCare**, **.load**, **.clear**  
depthAttachment.storeAction: MTLStoreAction **.dontCare**, **.store**, ...

## MTKView

currentRenderPassDescriptor: MTLRenderPassDescriptor  
colorPixelFormat: MTLPixelFormat **.bgra8Unorm**  
depthStencilPixelFormat: MTLPixelFormat **.depth32Float**  
sampleCount: Int **1**  
frame.size: CGSize  
contentScaleFactor: CGFloat  
currentDrawable: CAMetalDrawable? (protocol MTLDrawable)

## MTLRenderPipelineDescriptor

rasterSampleCount: Int  
colorAttachments[i].pixelFormat: MTLPixelFormat  
depthAttachmentPixelFormat: MTLPixelFormat  
stencilAttachmentPixelFormat: MTLPixelFormat **.invalid**  
vertexFunction: MTLFunction?  
fragmentFunction: MTLFunction?  
vertexDescriptor: MTLVertexDescriptor?

tessellationPartitionMode: MTLTessellationPartitionMode **.pow2**  
tessellationFactorStepFunction: MTLTessellationFactorStepFunction **.constant** (default), **.perPatch** (for tessellation)  
tessellationOutputWindingOrder: MTLWinding **.clockwise** (default)  
maxTessellationFactor: Int 16 for iOS, 64 for MacOS. See [Metal-Shading-Language-Specification.pdf](#).  
tessellationControlPointIndexType: MTLTessellationControlPointIndexType **.uint32**

colorAttachments[i].isBlendingEnabled: Bool True for alpha blending  
colorAttachments[i].rgbBlendOperation: MTLBlendOperation **.add**  
colorAttachments[i].sourceRGBBlendFactor: MTLBlendFactor **.sourceAlpha**  
colorAttachments[i].destinationRGBBlendFactor: MTLBlendFactor **.oneMinusSourceAlpha**

## example01.metal

```
...  
constant bool var01 [[ function_constant(</>) ]];  
...  
vertex VOut vertex_func(...) { ... }  
fragment float4 fragment_func(...) { ... }  
...
```

## MTLFunctionConstantValues

```
setConstantValue(  
    _ : <UnsafeRawPointer>  
    type: MTLDataType .bool, ...  
    index: Int <i>  
)
```

name of the kernel function  
in \*.metal file

## MTLDevice

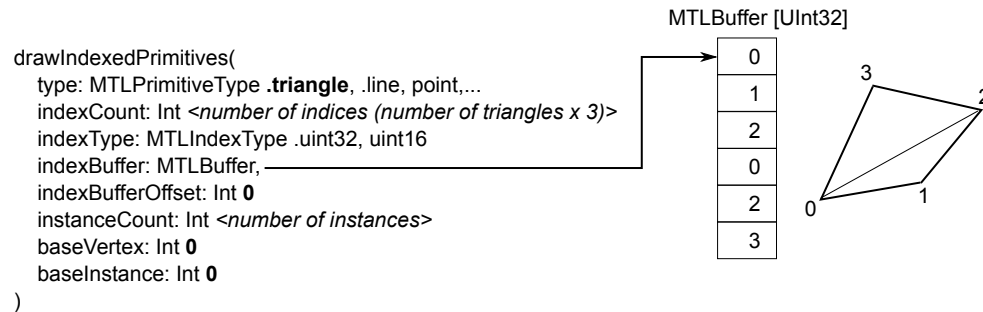
makeDefaultLibrary() -> MTLLibrary

## MTLLibrary

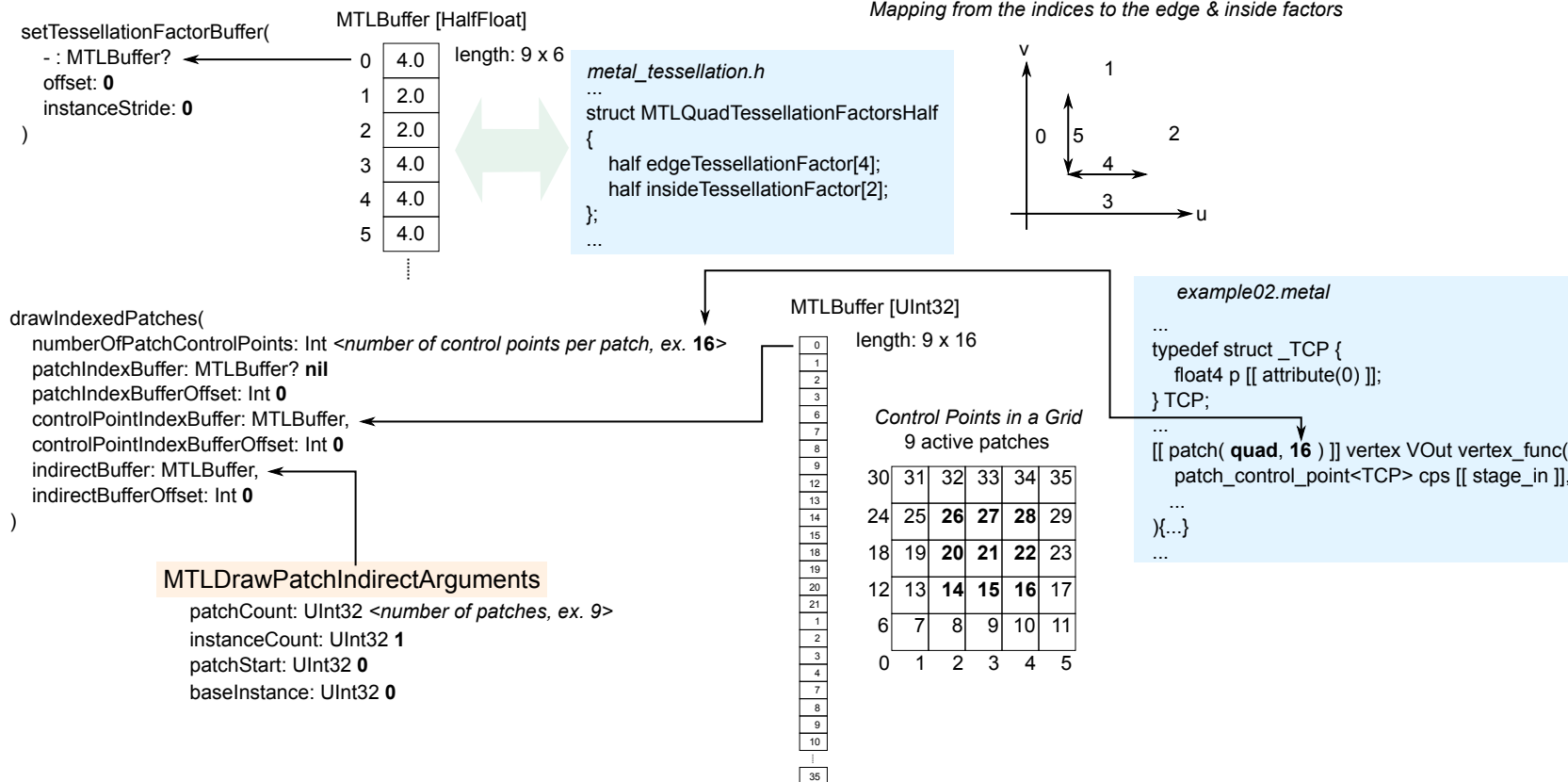
makeFunction( name: constantValues: )  
-> MTLFunction

# Drawing Triangles & Patches, and the Indices

## Drawing Triangles



## Drawing Patches



# Assignment of the Parameters to the Vertex & Fragment Shaders

## MTLDevice

```
makeBuffer(
  [ bytes: UnsafeRawPointer, ]
  length: Int <number of bytes>,
  options : MTLResourceOptions .storageModeShared, .storageModePrivate
) -> MTLBuffer
```

## MTLSamplerDescriptor

```
sAddressMode: MTLSamplerAddressMode .repeat
tAddressMode: MTLSamplerAddressMode .repeat
mipFilter: MTLSamplerMipFilter .linear
maxAnisotropy: Int 8
```

## MTLTexture

## MTLBuffer

```
setVertexBytes( _ UnsafeRawPointer, length: Int <in bytes, less than 4 KB>, index: Int )
```

```
setVertexBuffer( _ MTLBuffer?, offset: Int 0, index: Int )
```

```
setVertexTexture( _ MTLTexture?, index: Int )
```

```
setVertexSamplerState( _ MTLSamplerState?, index: Int )
```

```
setFragmentBytes( _ UnsafeRawPointer, length: Int <in bytes, less than 4 KB>, index: Int )
```

```
setFragmentBuffer( _ MTLBuffer?, offset: Int 0, index: Int )
```

```
setFragmentTexture( _ MTLTexture?, index: Int )
```

```
setFragmentSamplerState( _ MTLSamplerState?, index: Int )
```

```
setFragmentTextures( _ [ MTLTexture? ], range: Range<Int> ex. N ..< N + M )
```

## example03.metal

```
...
vertex VOut vertex_func(
  ...
  device T1& var01 [[ buffer(1) ]],
  device T2* arr01 [[ buffer(2) ]],
  ...
  texture2d<float> tex01 [[ texture(0) ]],
  depth2d<float> tex02 [[ texture(1) ]],
  ...
  sampler smp [[sampler(0)]],
) { ... }
...
fragment float4 fragment_func(
  ...
  constant T3& var01 [[ buffer(1) ]],
  constant T4* arr01 [[ buffer(2) ]],
  ...
  texture2d<float> tex01 [[ texture(0) ]],
  depth2d<float> tex02 [[ texture(1) ]],
  ...
  sampler smp [[sampler(0)]],
  array< texture2d<float>, M > texes [[ texture(N) ]],
) { ... }
...
```

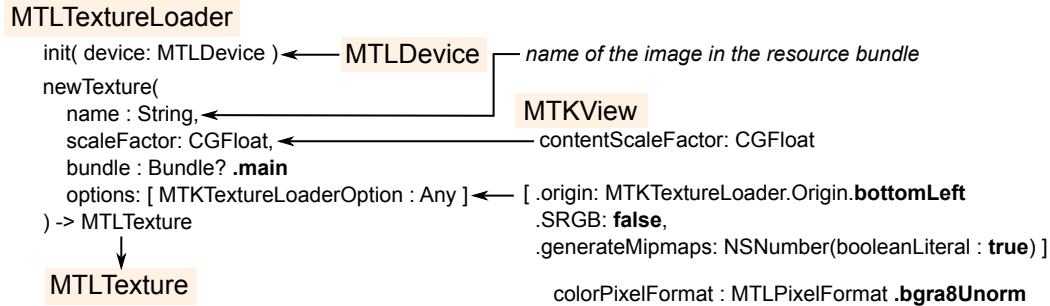
**NOTE:** Multiple textures of variable numbers M with the starting index N can be specified with the following pair of APIs.

- setFragmentTextures() in Swift.
- array< texture2d<float>, M> texes [[ ... ]] in the formal parameters of the fragment shader.

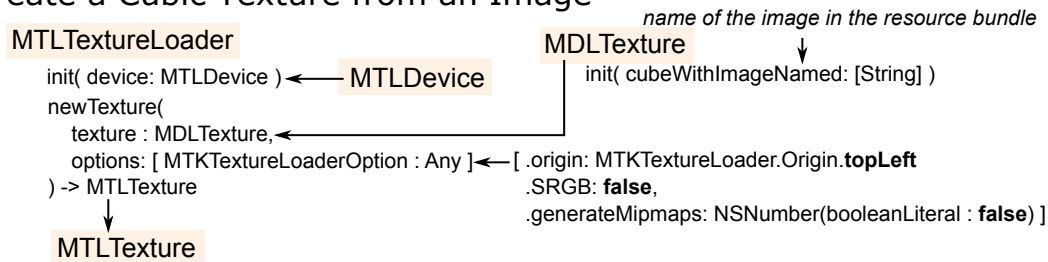
M must be determined at the compilation time. However, **at runtime, not all the M elements must be specified in [MTLTexture?]**. In the fragment shader each texture can be accessed by array indexing, i.e., texes[N+i].

# Texture Generation

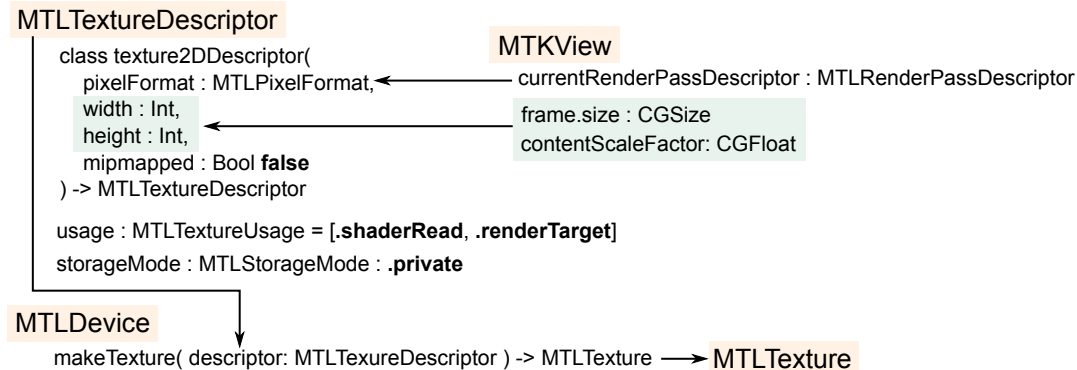
## Create a Texture from an Image



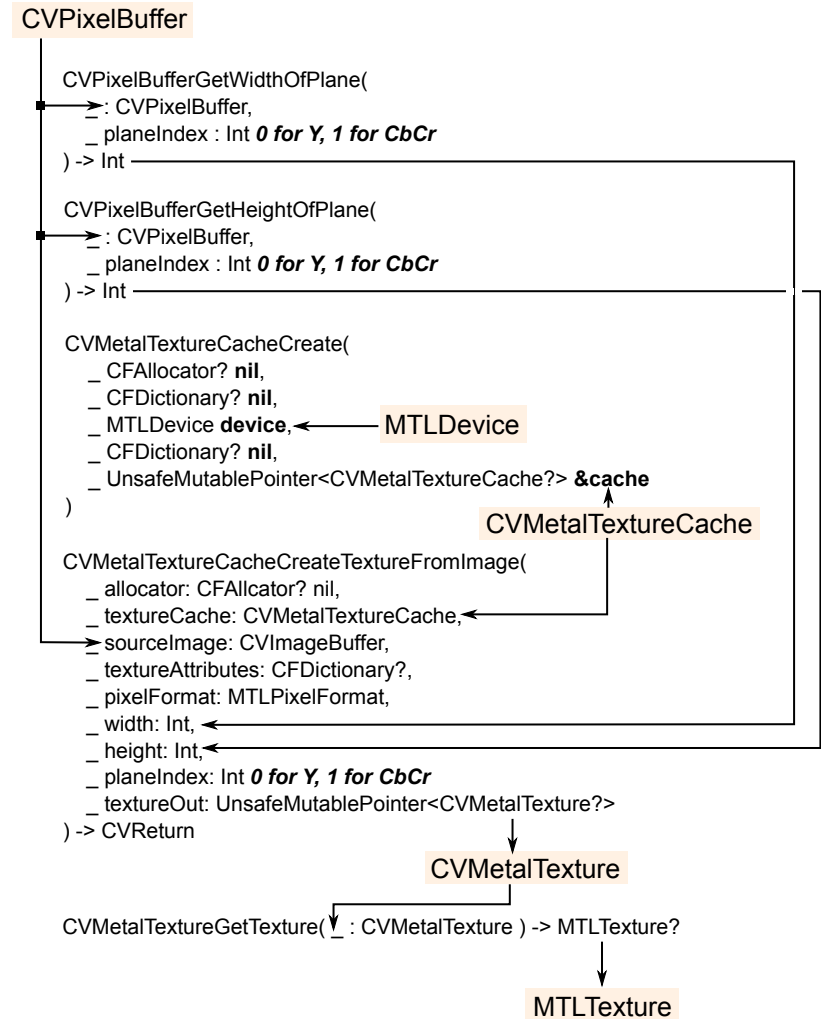
## Create a Cubic Texture from an Image



## Create an Empty Texture

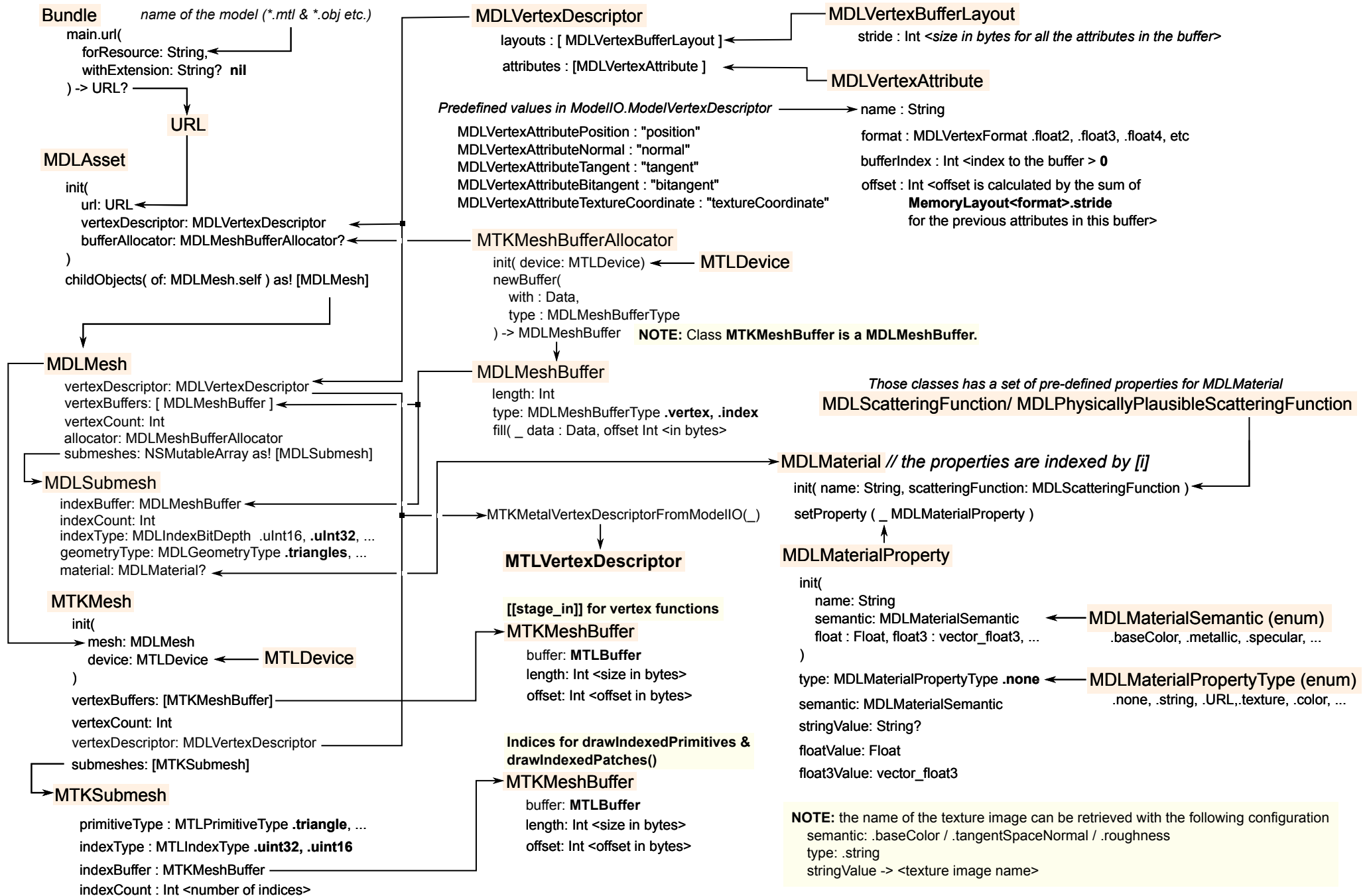


## Create a Texture from Core Video Images

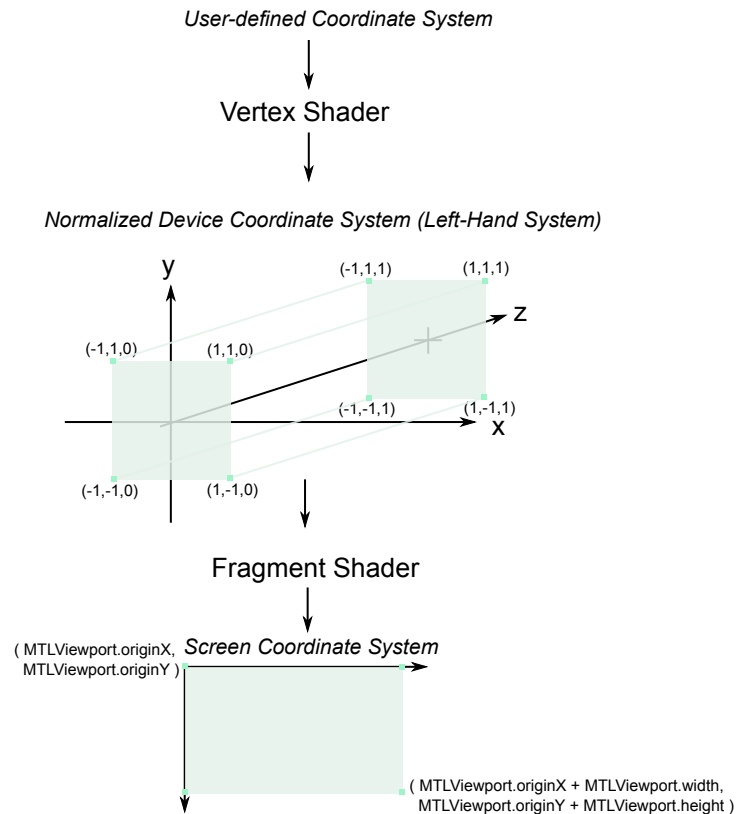


**NOTE:** CVImageBuffer == CVPixelBuffer == CVBuffer

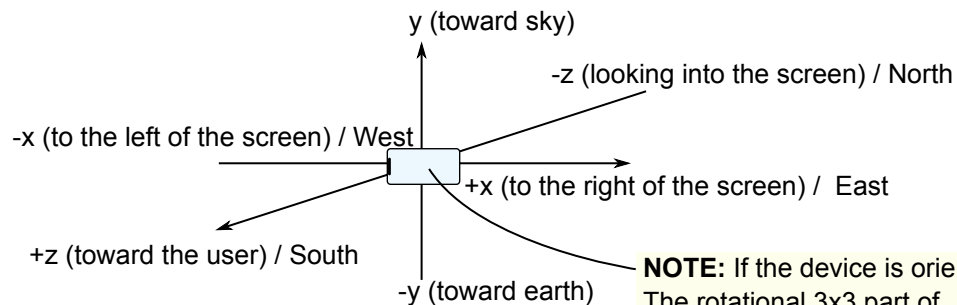
# MDLMesh and MTKMesh



# Coordinate Systems and Others



ARKit Coordinate System



**NOTE:** If the device is oriented this way, The rotational 3x3 part of `ARSession.currentFrame.camera.transform` is the identity matrix.

## How to discard a vertex in the vertex shader

```

...
struct VOClip {
    float4 p [[ position ]];
    ...
    float c [[ clip_distance ]] [1];
};
// The only difference from COVclip
// is the absence of float c.
struct VO {
    float4 p [[ position ]];
    ...
}
vertex VOClip vertex_func(...) {
    ...
    VOClip out {
        .p = position,
        ...
        .c = clip_distance // if negative, the vertex is discarded.
    }
    return out;
}
...
fragment float4 fragment_func(
    VO in [[ stage_in ]], // This is not VOClip, but VO.
    ... ) { ... }
  
```

## How to discard a vertex in the fragment shader

```

...
// Just call discard_fragment(); in the fragment shader.
fragment float4 fragment_func(...) {
    ...
    if (discard) {
        discard_fragment();
    }
}
  
```