BEGINNING METAL

HANDS-ON CHALLENGES

Beginning Metal

Caroline Begbie

Notice of Rights

Notice of Liability

Trademarks

7 Challenge #7: Node Tree

By Caroline Begbie

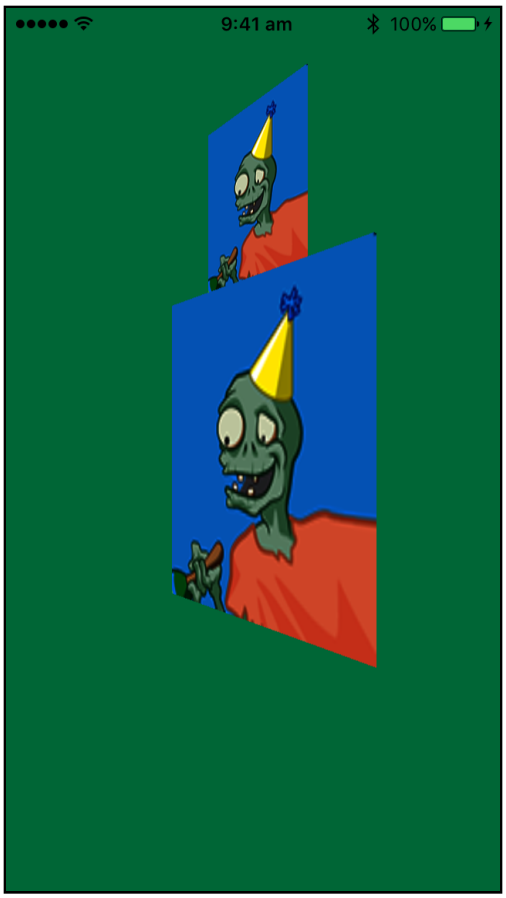
In the demo, I hard coded the matrix values in `Plane`'s render method. Obviously this is not very flexible, especially when we get other types of model.

Your challenge is to add a second quad as a smaller child of the first quad. But first you'll add a position, rotation and scale to the `Node` class. Then you can easily set the position of any node, and even animate them, whether it be a `Plane` or a `Scene` or a `Camera`.

There will be a number of things to refactor here.

1. Add properties for position, rotation and scale to `Node`.
2. Change `Renderable` so that all nodes that conform have a render method
3. Add the render method to `Plane`
4. Replace the current `Node`'s render method with one that multiplies the parent model matrix with the node's model matrix.
5. Start the rendering process from `Scene` with a view matrix.
6. For animation, create an update method that's performed on the `Scene` every frame.

At the end of the challenge, you'll be able to add Plane models easily to GameScene and animate them.



In Node.swift, add properties for position, rotation and scale to Node:

```
var position = float3(0)
var rotation = float3(0)
var scale = float3(1)
```

The node's model matrix is made up of the position, rotation and scale, so create a convenience property to return the model matrix:

```
var modelMatrix: matrix_float4x4 {
  var matrix = matrix_float4x4(translationX: position.x,
                               y: position.y, z: position.z)
  matrix = matrix.rotatedBy(rotationAngle: rotation.x,
                            x: 1, y: 0, z: 0)
  matrix = matrix.rotatedBy(rotationAngle: rotation.y,
                            x: 0, y: 1, z: 0)
  matrix = matrix.rotatedBy(rotationAngle: rotation.z,
                            x: 0, y: 0, z: 1)
  matrix = matrix.scaledBy(x: scale.x, y: scale.y, z: scale.z)
  return matrix
}
```

So now you can remove some code from the `Plane`'s render method.

Remove these lines from `render(commandEncoder:deltaTime:)`:

```
time += deltaTime
let animateBy = abs(sin(time)/2 + 0.5)

let rotationMatrix = matrix_float4x4(rotationAngle: animateBy,
                                     x: 0, y: 0, z: 1)
let viewMatrix = matrix_float4x4(translationX: 0, y: 0, z: -6)

let modelViewMatrix = matrix_multiply(rotationMatrix, viewMatrix)
modelConstants.modelViewMatrix = modelViewMatrix
```

You're now going to change the rendering process.

Currently `Renderer` calls `Scene`'s render method, which calls the child nodes' render methods recursively. This is the current rendering method in `Node`:

```
func render(commandEncoder:MTLRenderCommandEncoder, deltaTime: Float) {
  for child in children  {
    child.render(commandEncoder: commandEncoder, deltaTime: deltaTime)
  }
}
```

Each child node should have the parent's `modelViewMatrix` applied to it. So you'll need to send the parent's `modelViewMatrix` as a parameter in the rendering method.

Additionally, the model won't need a reference to `deltaTime`, because all the animation will be handled where it should be - in the scene.

Each `Node` will still recursively call it's child node's render method, but each `Node` subclass that conforms to `Renderable` will have a `doRender()` method where the draw call happens. This means that you'll be able to have `Node` classes that are not `Renderable` - for example, if you want to group `Renderable` objects together, you can have a `Node` group parent that doesn't render.

On to the code, then.

All `Renderable` objects should have a `ModelConstants` struct, so in **Renderable.swift**, add this to the `Renderable` protocol:

```
var modelConstants: ModelConstants { get set }
```

Add this method to `Renderable` to describe the actual rendering:

```
func doRender(commandEncoder: MTLRenderCommandEncoder,
              modelViewMatrix: matrix_float4x4)
```

When you add this method, you now get a complaint from **Plane.swift** that it doesn't conform to `Renderable`.

In **Plane.swift**'s `Renderable` extension, create the method `doRender(commandEncoder:modelViewMatrix:)` to conform to `Renderable`:

```
  func doRender(commandEncoder: MTLRenderCommandEncoder,
                modelViewMatrix: matrix_float4x4) {
  }
```

Find the method `render(commandEncoder:deltaTime:)` in `Plane`. Copy the code from that method (not including `super.render(commandEncoder: commandEncoder, deltaTime: deltaTime)` to the new method `doRender(commandEncoder:modelViewMatrix:)`.

Your new method should read:

```
  func doRender(commandEncoder: MTLRenderCommandEncoder,
                modelViewMatrix: matrix_float4x4) {
    guard let indexBuffer = indexBuffer else { return }

    let aspect = Float(750.0/1334.0)
    let projectionMatrix = matrix_float4x4(projectionFov:
radians(fromDegrees: 65),
                                           aspect: aspect, nearZ: 0.1,
farZ: 100)
    modelConstants.modelViewMatrix = matrix_multiply(projectionMatrix,
modelViewMatrix)

    commandEncoder.setRenderPipelineState(pipelineState)

    commandEncoder.setFragmentTexture(texture, at: 0)
    commandEncoder.setFragmentTexture(maskTexture, at: 1)

    commandEncoder.setVertexBuffer(vertexBuffer, offset: 0, at: 0)
    commandEncoder.setVertexBytes(&modelConstants,
                                  length:
MemoryLayout<ModelConstants>.stride,
                                  at: 1)
    commandEncoder.drawIndexedPrimitives(type: .triangle,
                                         indexCount: indices.count,
                                         indexType: .uint16,
                                         indexBuffer: indexBuffer,
                                         indexBufferOffset: 0)
  }
```

We're leaving the projection matrix transformation in there for the moment. In the next challenge you'll add a `Camera` class which will store the scene's view and projection matrices.

Remove the method `render(commandEncoder:deltaTime:)`.

You've now set up a render method that uses the current model view matrix. This model view matrix will be updated by the recursive call in `Node`.

In Node, replace render(commandEncoder:deltaTime:) with this method:

```
func render(commandEncoder: MTLRenderCommandEncoder,
            parentModelViewMatrix: matrix_float4x4) {
  let modelViewMatrix = matrix_multiply(parentModelViewMatrix,
                                        modelMatrix)
  for child in children {
    child.render(commandEncoder: commandEncoder,
                 parentModelViewMatrix: modelViewMatrix)
  }
}
```

Here's where you multiply each child node's matrix with its parent. Because you removed render(commandEncoder:deltaTime:) you currently have a compile error in Renderer complaining that the method has incorrect arguments. You'll fix this in a moment.

To render each Node add this to the end of render(commandEncoder:parentModelViewMatrix:):

```
if let renderable = self as? Renderable {
  commandEncoder.pushDebugGroup(name)
  renderable.doRender(commandEncoder: commandEncoder,
                      modelViewMatrix: modelViewMatrix)
  commandEncoder.popDebugGroup()
}
```

Here you call doRender(commandEncoder:modelViewMatrix:) if the Node conforms to Renderable.

The command encoder's pushDebugGroup(_:) and popDebugGroup() methods allow you to inspect each node's render commands when using the GPU debugger more easily by pushing the name of the node you're currently rendering. When you release your app, you can remove these lines.

The Renderer currently calls the Scene's render method. This was the method that you just changed in Node. So now add that render method to **Scene.swift**:

```
func render(commandEncoder: MTLRenderCommandEncoder,
            deltaTime: Float) {
}
```

The project should now build without errors.

The Scene's render method will be the top method to call the Node's recursive render method. Add this to Scene's render(commmandEncoder:deltaTime:):

```
let viewMatrix = matrix_float4x4(translationX: 0, y: 0, z: -4)
```

This is the position of the camera. Later on you'll create a Camera class to do this more attractively.

Add this to `render(commmandEncoder:deltaTime:)`

```
for child in children  {
  child.render(commandEncoder: commandEncoder,
               parentModelViewMatrix: viewMatrix)
}
```

Here you're going to render all the child nodes of the scene recursively. You've set the view matrix's **z** position to -4. When you multiply all the child nodes by this view matrix, all the scene's child nodes will move backwards by 4 units.

All the render code is now in place, so build and run, and your zombie should be central stage, but back 4 units.

Phew! If your zombie is behaving correctly, congratulations! This was the hardest section of the entire course. It's all downhill from here :]. If you're struggling with this, I recommend you go over the video and challenge several times. Experiment with changing matrix values.

Animation!

You'll now create an update method that's called on `Scene` every frame. Add this new method to `Scene`:

```
func update(deltaTime: Float) {}
```

This is a stub method that `Scene` subclasses can override.

Add this to the top of `render(commmandEncoder:deltaTime:)`:

```
update(deltaTime: deltaTime)
```

Here you're calling the update method every frame.

In `GameScene`, override `update(deltaTime:)`:

```
override func update(deltaTime: Float) {
}
```

Any animation code for the scene can now go in this update method. Make the zombie rotate by adding this line to `update(deltaTime)`:

```
quad.rotation.y += deltaTime
```

Build and run, and your zombie should now be rotating in three dimensions. Because of the projection, the front vertices are spread out more than the back vertices.

Just for fun, see how easy it to add models to the scene as well as animate them. Add this to the end of `GameScene`'s `init(device:size:)`:

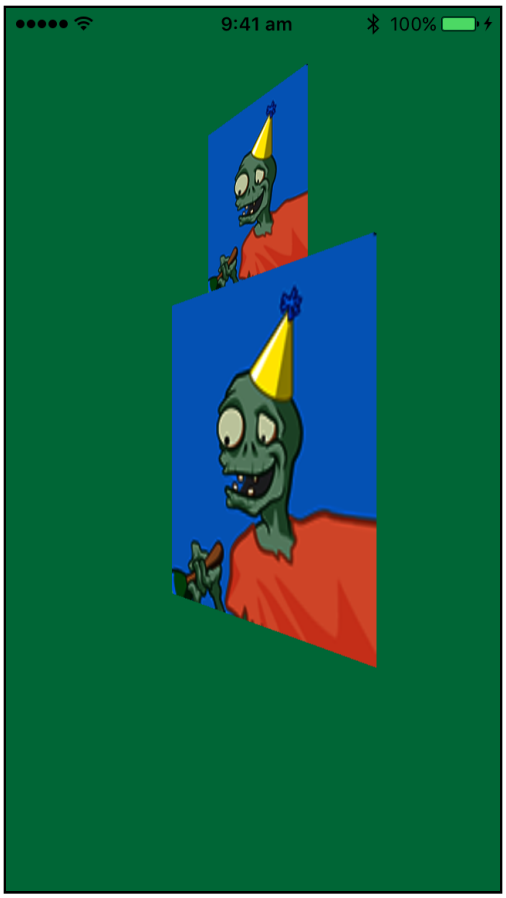
```
quad.position.x = -1
quad.position.y = 1

let quad2 = Plane(device: device, imageName: "picture.png")
quad2.position.x = 1
quad2.position.y = -1
add(childNode: quad2)
```

Here you've moved the original rotating zombie pic up into the top left of the screen, and added a second zombie pic and moved it down to the bottom right of the screen.

Now change that previous code to:

```
let quad2 = Plane(device: device, imageName: "picture.png")
quad2.scale = float3(0.5)
quad2.position.y = 1.5
quad.add(childNode: quad2)
```

Here you add the second zombie as a child of the first zombie, so the child rotates when the parent rotates.


9:41 am
100%