BEGINNING METAL

HANDS-ON CHALLENGES

Beginning Metal

Caroline Begbie

Notice of Rights

Notice of Liability

Trademarks

8 Challenge #8: Camera

By Caroline Begbie

Our code currently has the projection matrix hard coded in the `Primitive` render method. Your challenge is to create a `Camera` class, which will be a `Node` subclass. The `Camera` class will have all the projection data, so that in the scene, you'll be able to move it around and experiment with settings.

At the end of this challenge your app will look like this:

There's only one camera per scene, so there's only one view matrix and one projection matrix per scene. Just like each model has a set of constants, each scene will have a set of constants, and you'll create a scene constants struct to send to the GPU.

First, create a new file in the **Nodes** group called **Camera.swift**.

Replace the existing code with:

```
import MetalKit

class Camera: Node {

}
```

You've now created a `Camera` that's a subclass of `Node`, so you'll be able to position cameras easily in the scene.

Create a property for the view matrix. For the moment, just return the model matrix. As view matrices can be calculated from the model matrix in several different ways, it's a good idea to have a separate view matrix property, in case you decide to change the view matrix calculation at a future time.

```
var viewMatrix: matrix_float4x4 {
  return modelMatrix
}
```

Now for the projection matrix.

Currently you have this code in `Primitive`'s `doRender(commandEncoder:modelViewMatrix:)`:

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

You're going to remove these hard coded values. Create them as properties in `Camera`:

```
var fovDegrees: Float = 65
var fovRadians: Float {
  return radians(fromDegrees: fovDegrees)
}
var aspect: Float = 1
var nearZ: Float = 0.1
var farZ: Float = 100
```

You've set the field of view to be 65 degrees. Although we generally find it easier to work in degrees, our code works in radians, so it's useful to have a conversion calculated variable.

Create a property to return the projection matrix:

```
var projectionMatrix: matrix_float4x4 {
  return matrix_float4x4(projectionFov: fovRadians,
                         aspect: aspect,
                         nearZ: nearZ,
                         farZ: farZ)
}
```

This is the same code as you had above, but stored as a variable. That's everything you need for the camera.

In `Primitive`'s `doRender(commandEncoder:modelViewMatrix:)`, remove the projection matrix code:

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

Still in `doRender(commandEncoder:modelViewMatrix:)`, change this line:

```
modelConstants.modelViewMatrix =
    matrix_multiply(projectionMatrix, modelViewMatrix)
```

to:

```
modelConstants.modelViewMatrix = modelViewMatrix
```

You'll shortly change the vertex function to multiply each vertex against the camera's projection matrix instead of doing it here.

All `Scene`s will have a camera, so in **Scene.swift**, add this property to `Scene`:

```
var camera = Camera()
```

At the end of `init(device:size:)`, position the camera and add it to the scene:

```
camera.aspect = Float(size.width / size.height)
camera.position.z = -6
add(childNode: camera)
```

In `Scene`'s `render(commandEncoder:deltaTime:)`, change:

```
let viewMatrix = matrix_float4x4(translationX: 0, y: 0, z: -6)
for child in children  {
  child.render(commandEncoder: commandEncoder,
               parentModelViewMatrix: viewMatrix)
}
```

to:

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

Subclasses have the ability to reposition the camera, so you're now sending the camera's view matrix property instead of hard coding it.

The last thing to add is the projection matrix. If you build and run now, the projection matrix is not being taken into account, so your models will have disappeared.

In **Types.swift**, create a new struct for the scene constants:

```
struct SceneConstants {
  var projectionMatrix = matrix_identity_float4x4
}
```

Back in `Scene`, add a new property to hold the scene constants:

```
var sceneConstants = SceneConstants()
```

In `render(commandEncoder:deltaTime:)` after calling `update(deltaTime:)`, add this code:

```
sceneConstants.projectionMatrix = camera.projectionMatrix
commandEncoder.setVertexBytes(&sceneConstants,
                length: MemoryLayout<SceneConstants>.stride,
                at: 2)
```

Here you store the camera's projection matrix in the scene constants struct and tell the command encoder that the scene constants are stored at buffer index 2.

On the GPU side, you'll change the vertex function to receive the scene constants.

In **Shader.metal**, add the scene constants struct:

```
struct SceneConstants {
  float4x4 projectionMatrix;
};
```

Change the vertex function's header to:

```
vertex VertexOut vertex_shader(const VertexIn vertexIn [[ stage_in ]],
       constant ModelConstants &modelConstants [[ buffer(1) ]],
       constant SceneConstants &sceneConstants [[ buffer(2) ]]) {
```

Now you're able to access the scene constants in the vertex function.

In the vertex function change this code:

```
vertexOut.position = modelConstants.modelViewMatrix * vertexIn.position;
```

to:

```
float4x4 matrix = sceneConstants.projectionMatrix
                    * modelConstants.modelViewMatrix;
vertexOut.position = matrix * vertexIn.position;
```

You're now multiplying the model constants's model view matrix by the projection matrix just as you had earlier been doing in the `Primitive`'s render method.

Build and run and your rotating cube and zombie should be as before. Your code is nicely refactored now so you can experiment with moving the camera around the scene.

In `GameScene`, add this to the end of `init(device:size:)`:

```
camera.position.y = -1
camera.position.x = 1
camera.position.z = -6
camera.rotation.x = radians(fromDegrees: -45)
camera.rotation.y = radians(fromDegrees: -45)
```

Build and run, and by just repositioning the camera, you get an entirely different angle on the scene.