BEGINNING METAL

HANDS-ON CHALLENGES

Beginning Metal

Caroline Begbie

Notice of Rights

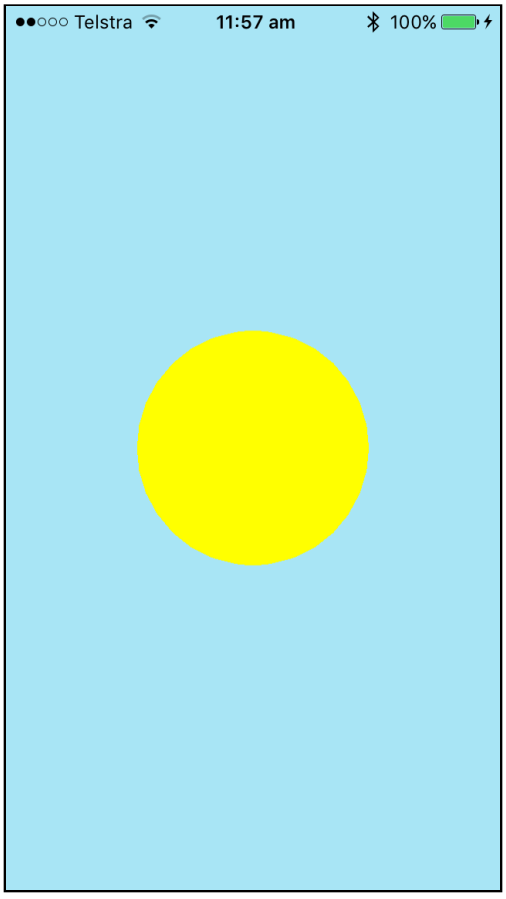
Notice of Liability

Trademarks

Challenge #9: Importing Models

By Caroline Begbie

Your challenge is to create a happy sun model in Blender (or your 3d app of choice), which will just be a sphere, and you'll color and load it into the scene.

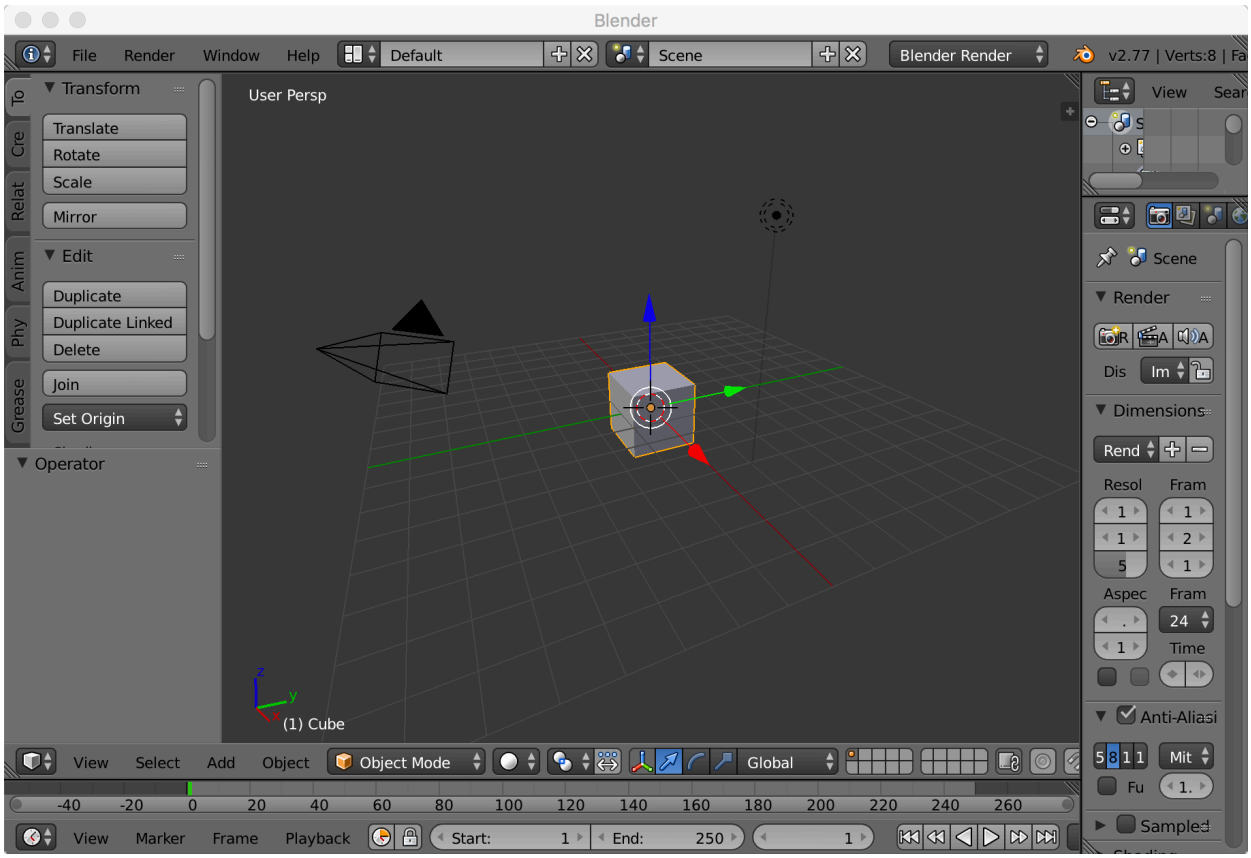


If you're interested in making models have a look at Vicki Wenderlich's tutorial on how to make a mushroom at https://www.raywenderlich.com/49955/blender-tutorial-for-beginners-how-to-make-a-mushroom.

If you don't have Blender, or want to skip the modelling part of this challenge, you can use the sun model in the Resources > Sun Model folder for this video. Skip down to the **Importing the sun model** section.

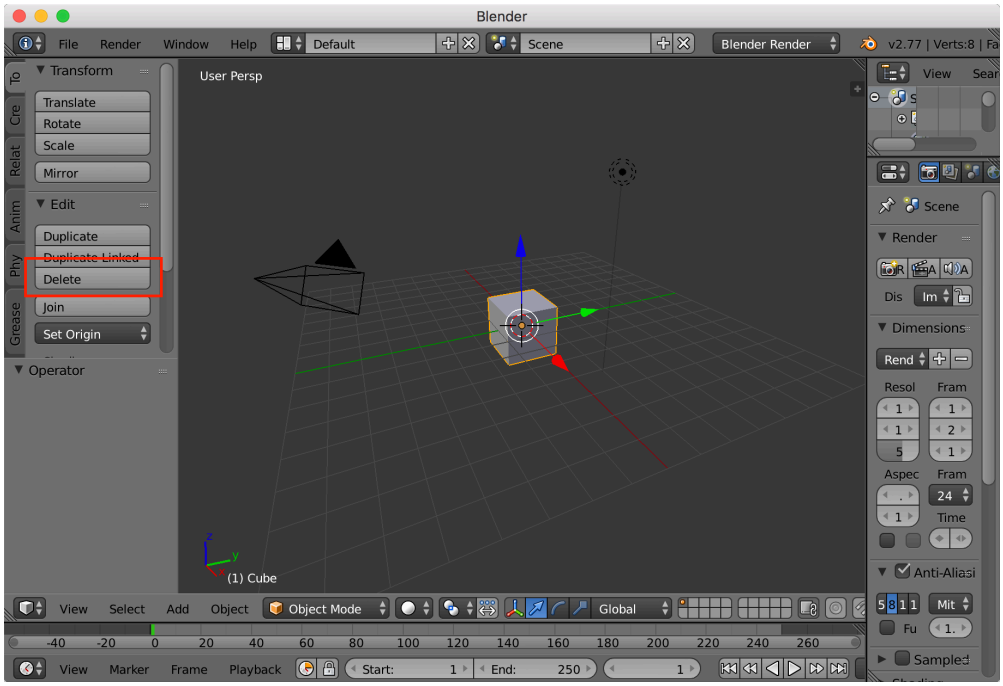
Blender

I'll be using all Blender defaults - In Blender, I've done **File > Reset** to Factory Settings. If you've changed your interface, it may look a little different.



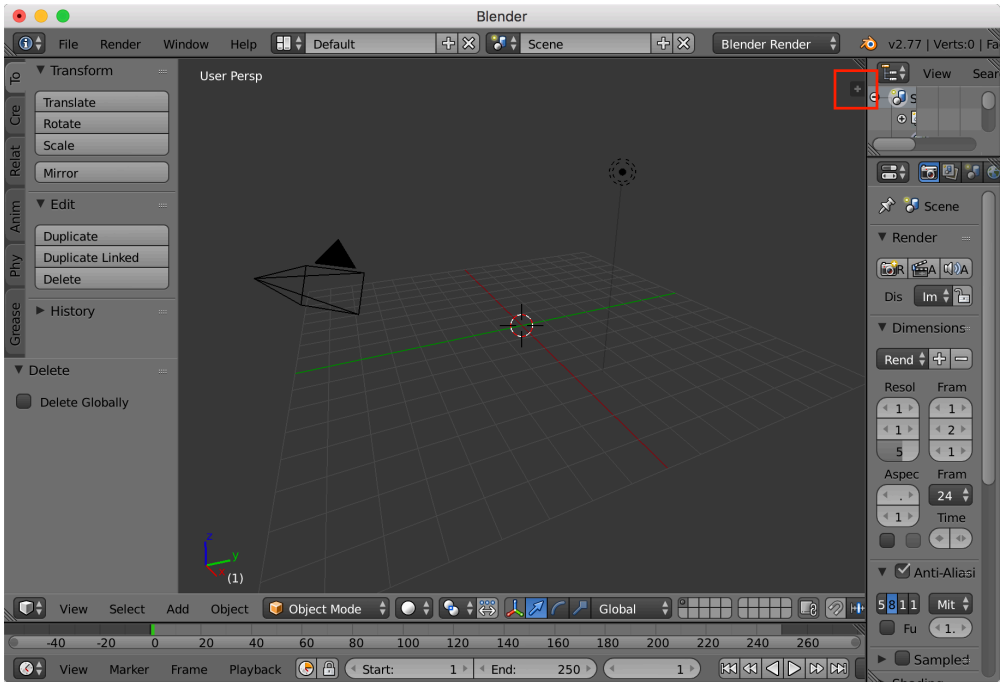
Select the cube that's in the center of the screen. Blender has a very unusual interface, and to select the cube, right click it. The cube will get an orange outline.

On the left pane, you should have the **Tools tab** open.Click **Delete**, and you'll get a pop up menu. Click **Delete** again to delete the cube.

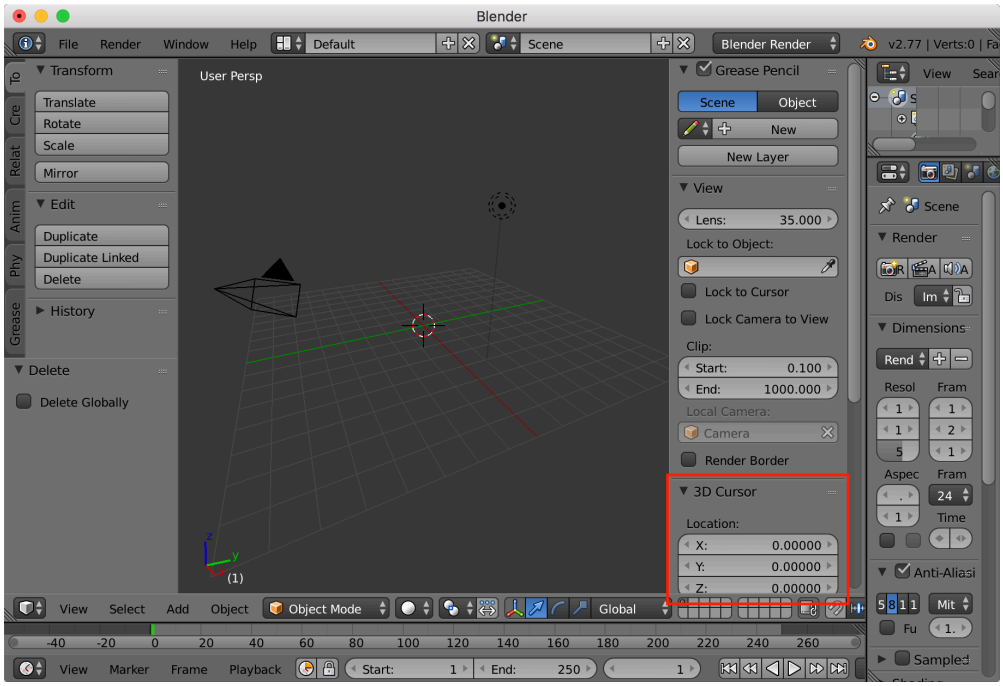


You're going to add a sphere for the sun.

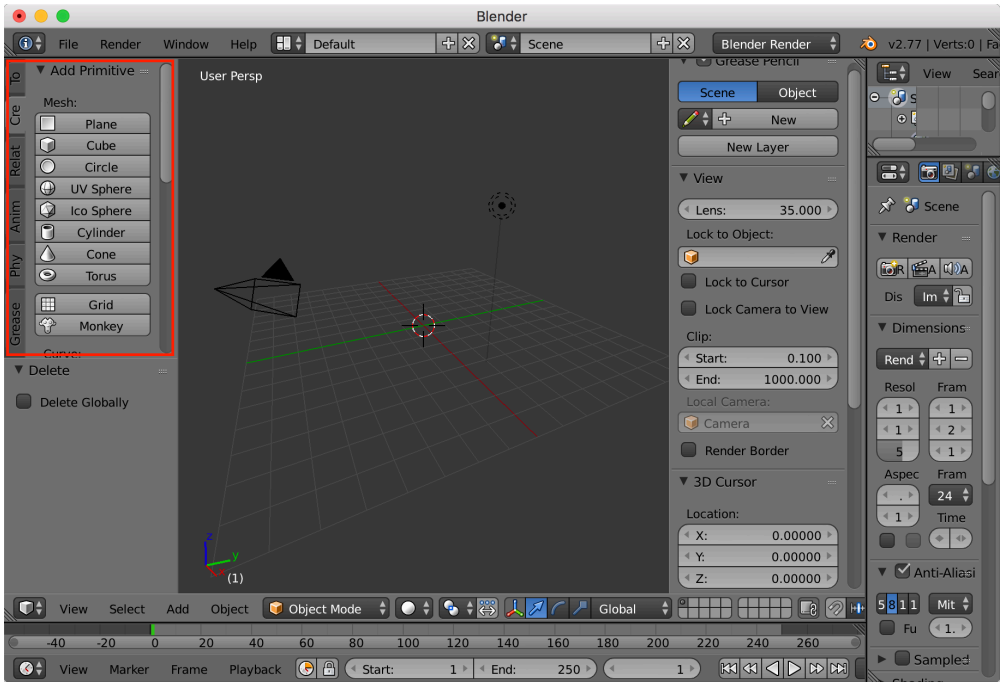
Left click at the origin (the center of the scene) and place the 3d cursor to position the next object. Click the **+** at the top right of the view pane to open the **Properties** panel.



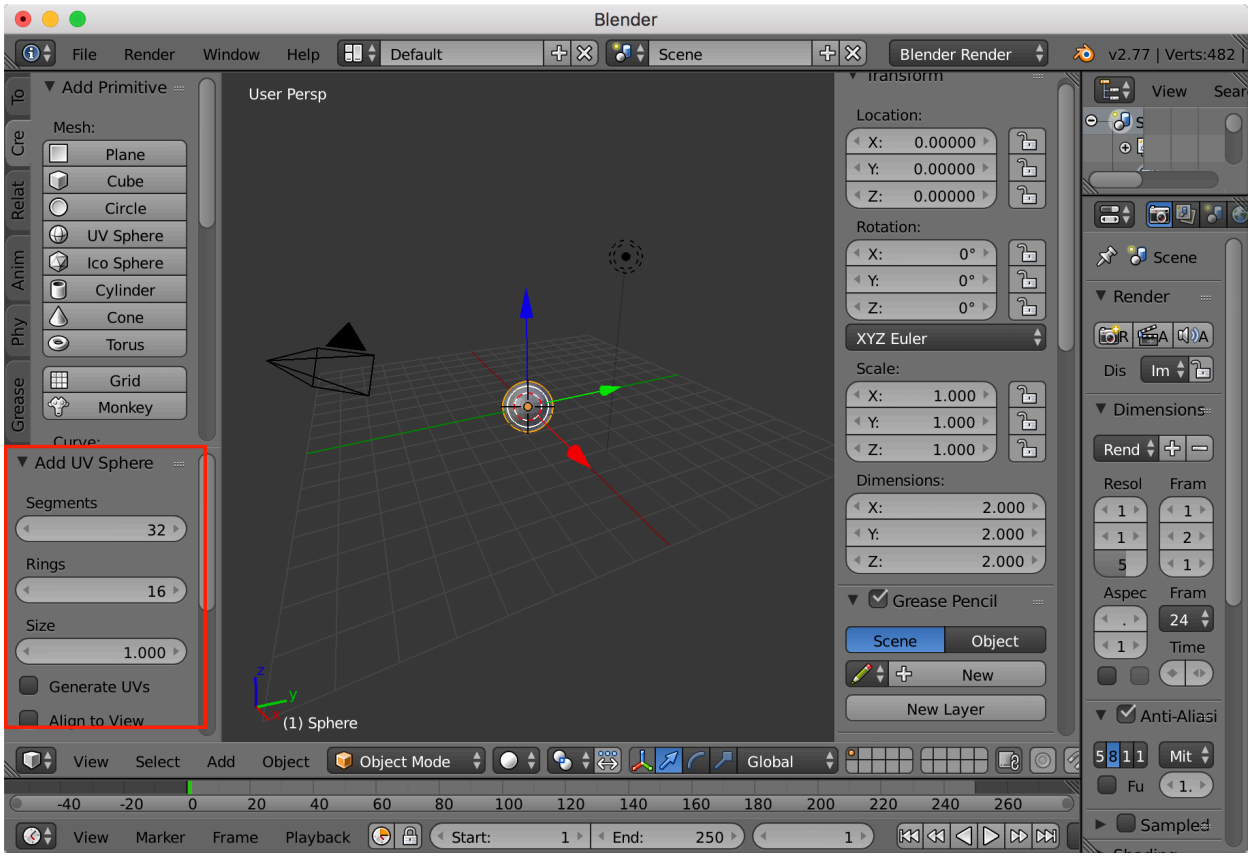
In the Properties panel, locate the 3D cursor section and change X, Y and Z to 0. The next object you add to the scene will be placed at this 3d cursor.



Click the **Create** tab and choose UV sphere.



Leave the segments at 32, the Rings at 16 and the size as 1.0



If you want to use mouse controls:

To **Zoom**, if you have a magic mouse, hold down **Ctrl** and scroll in. With a normal mouse you can use the scroll wheel.

To **Pan**, if you have a magic mouse, hold **Shift** and gesture to move the scene around. With a normal mouse, hold **Shift** and middle-click and drag.

To **Orbit**, just use gesture on the magic mouse or middle-click and drag with a normal mouse.

To export the .obj file, choose **File > Export > Wavefront (.obj)**, name the file **sun.obj** and save it to your chosen directory.

The .obj file is now ready for loading into your project. Blender creates a .mtl file as well. This holds material information about the object, such as color and surface effects, but we won't be using that in this tutorial.

Importing the sun model

Import **sun.obj** into your Xcode project by dragging the file from Finder to the Project Navigator.

In the Scenes folder, create a new Swift file called **LandscapeScene.swift**. This is going to be a completely new scene, so add the code to run an empty scene:

```
import MetalKit

class LandscapeScene: Scene {
  override init(device: MTLDevice, size: CGSize) {
    super.init(device: device, size: size)
  }

  override func update(deltaTime: Float) {
  }

}
```

In `ViewController`, in `viewDidLoad()`, change the scene that's rendered.

Change:

```
renderer?.scene = GameScene(device: device, size: view.bounds.size)
```

to:

```
renderer?.scene = LandscapeScene(device: device, size: view.bounds.size)
```

Add this to the `Colors` enum at the top of **ViewController.swift**

```
static let skyBlue = MTLClearColor(red: 0.66,
                                   green: 0.9,
                                   blue: 0.96,
                                   alpha: 1.0)
```

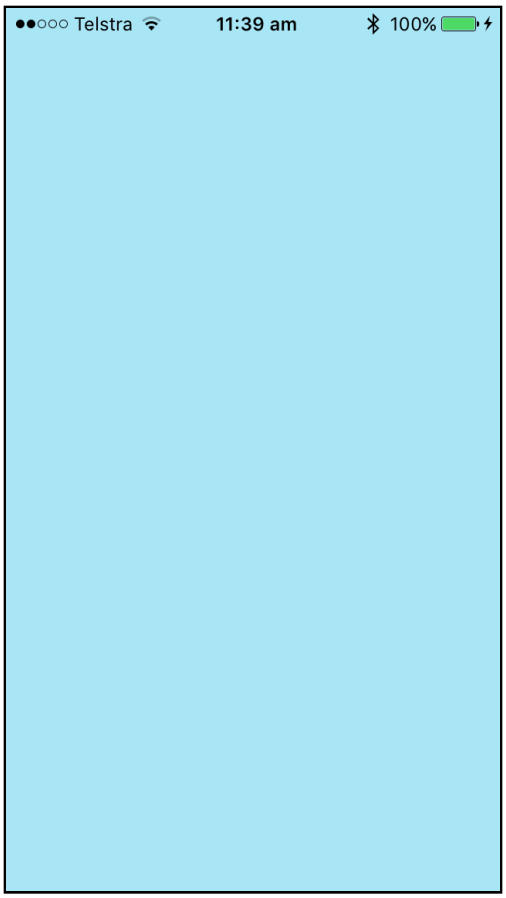
and in `viewDidLoad()`, change:

```
metalView.clearColor = Colors.wenderlichGreen
```

to

```
metalView.clearColor = Colors.skyBlue
```

Build and run and you should have a blank screen with a blue background.



Add the sun to `LandscapeScene`. First add the property:

```
let sun: Model
```

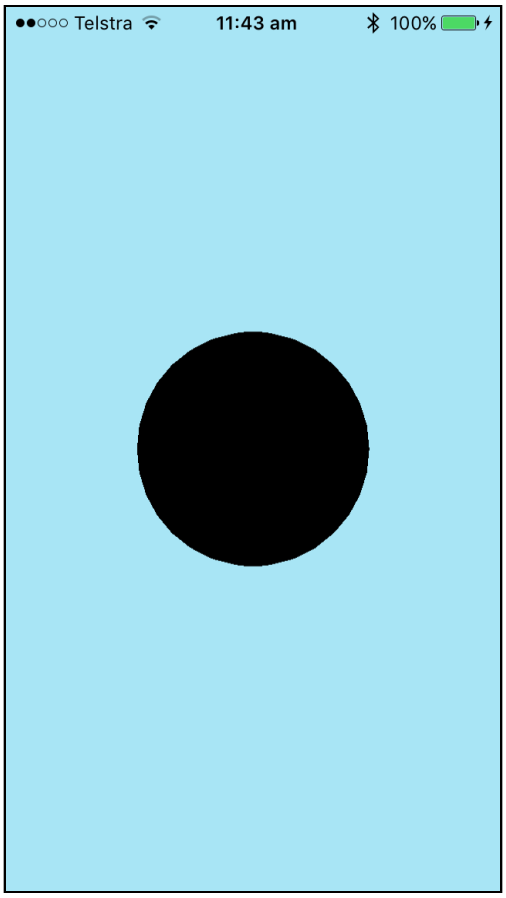
And at the top of `init(device:size)`, instantiate this:

```
sun = Model(device: device, modelName: "sun")
```

At the bottom of `init(device:size)`, add it to the scene:

```
add(childNode: sun)
```

Build and run, and you have a black sun in your scene:



To be able to color a model, instead of adding color to each vertex as we did before, we'll add a property to `Node`.

In `Node`, add this property:

```
var materialColor = float4(1)
```

This will set a default of white. `float4(1)` is equivalent to `float4(1, 1, 1, 1)`.

To be able to render this color, we'll need to add it to the `ModelConstants` struct that's sent to the GPU. We'll also create a new fragment function to render this color.

In **Types.swift**, add this to `ModelConstants`:

```
var materialColor = float4(1)
```

In **Shader.metal**, do the same. Add this to `ModelConstants`:

```
float4 materialColor;
```

The size of the two `ModelConstants` structs must exactly match.

Add the property also to `VertexOut` so that the fragment function can access it:

```
float4 materialColor;
```

In the vertex function `vertex_shader`, add this:

```
vertexOut.materialColor = modelConstants.materialColor;
```

Here you send the material color to the struct that the fragment function will use.

Create a new fragment function at the end of the file:

```
fragment half4 fragment_color(VertexOut vertexIn [[ stage_in ]]) {
  return half4(vertexIn.materialColor);
}
```

Whereas the original fragment shader took the color from each vertex, you're just returning the material color of the whole model.

While you're here, change the textured fragment function so that it will take into account the material color.

In `texturedFragment(...)`, after instantiating the `color`, add this line:

```
color = color * vertexIn.materialColor;
```

Here you multiply the texture color by the model's material color. When the material color is white, which has the value 1 and is the default, there will be no difference.

Now that you have a fragment function that renders a color, you can use this in the pipeline.

In `Model`, change:

```
var fragmentFunctionName: String = "fragment_shader"
```

to:

```
var fragmentFunctionName: String = "fragment_color"
```

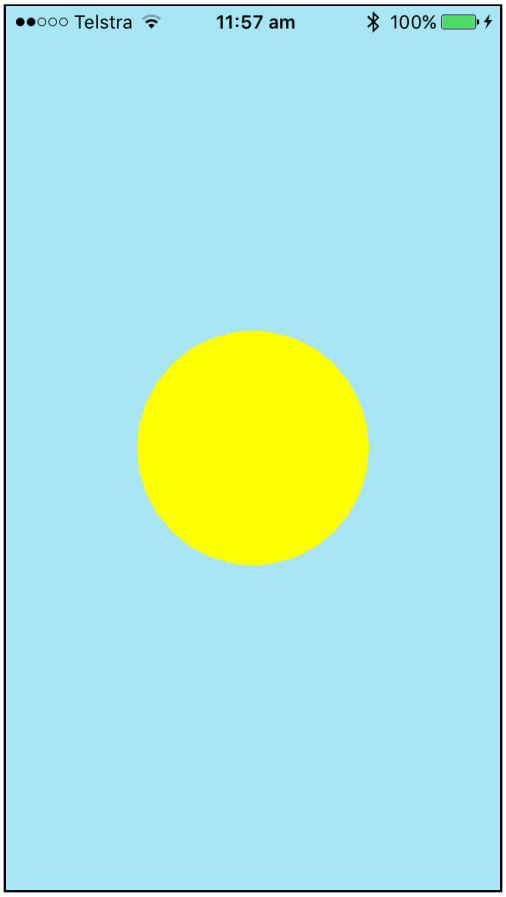
In `doRender(commandEncoder:modelViewMatrix:)`, just after setting the model constant's model view matrix, send the material color to the GPU:

```
modelConstants.materialColor = materialColor
```

In `LandscapeScene`'s `init(device:size:)`, set the color of the sun:

```
sun.materialColor = float4(1, 1, 0, 1)
```

Build and run and your sun should be yellow.



Later on, this material color will be very useful. We'll be able to create objects with a grey mottled texture and tint them using this material color.