

.....
**BEGINNING
METAL**
.....



HANDS-ON CHALLENGES

Beginning Metal

Caroline Begbie

Copyright ©2016 Razeware LLC.

Notice of Rights

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written permission of the copyright owner.

Notice of Liability

This challenge and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connection with the software or the use of other dealing in the software.

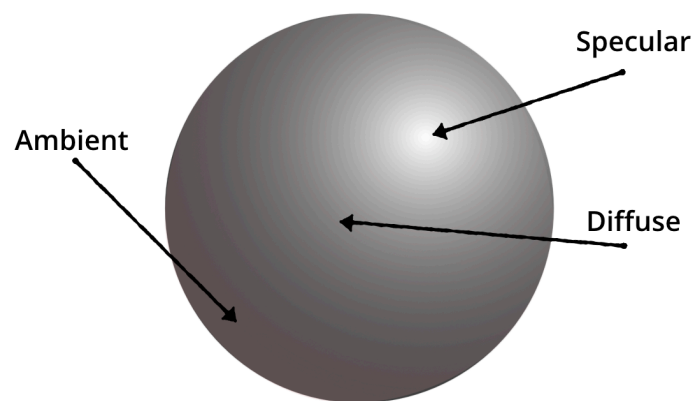
Trademarks

All trademarks and registered trademarks appearing in this book are the property of their own respective owners.

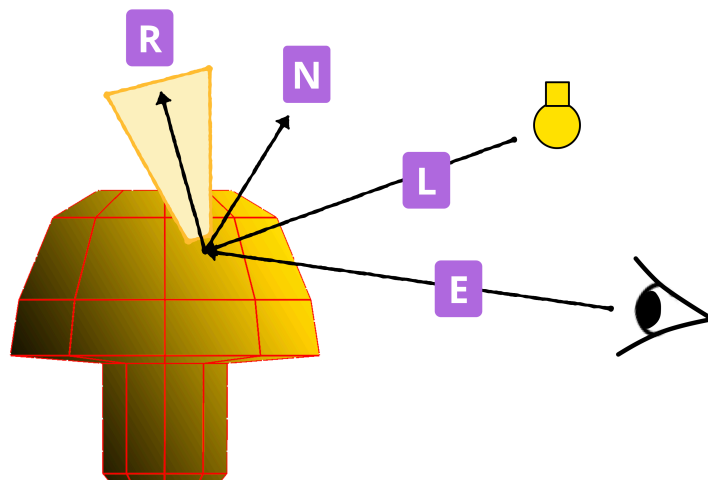
Challenge #12: Specular Lighting

By Caroline Begbie

In this challenge, you're going to add specular lighting. This is the lighting that shows how shiny an object is.



Specular lighting is similar to diffuse lighting in that you use the normal for calculation. However, there are extra properties required for calculation.



You'll need to record the eye position, specular intensity and shininess. In our case, the eye position will be the position of the camera. The specular intensity and shininess are part of the model's materials attributes. For example the specular highlight of skin and metal would be completely different.

In Node, add the new material properties:

```
var specularIntensity: Float = 1
var shininess: Float = 1
```

You'll need to send these values with the model constants to the GPU, so in **Types.swift**, add the new properties to ModelConstants:

```
var specularIntensity: Float = 1
var shininess: Float = 1
```

In Model's doRender(commandEncoder:modelViewMatrix:) move the node material properties to the model constants struct just under the other model constants properties:

```
modelConstants.shininess = shininess
modelConstants.specularIntensity = specularIntensity
```

On the GPU side, in **Shader.metal**, add these properties to ModelConstants:

```
float specularIntensity;
float shininess;
```

You'll pass on these values to the fragment function, so add these properties to VertexOut:

```
float specularIntensity;
float shininess;
```

In the vertex function, move these values to VertexOut:

```
vertexOut.shininess = modelConstants.shininess;
vertexOut.specularIntensity = modelConstants.specularIntensity;
```

So that's set up the material's specular properties. Now for the eye position.

Add a property to VertexOut:

```
float3 eyePosition;
```

In the vertex function, the eye position should be in camera coordinates without the projection, so we multiply the vertex position by the model view matrix:

```
vertexOut.eyePosition = (modelConstants.modelViewMatrix  
                        * vertexIn.position).xyz;
```

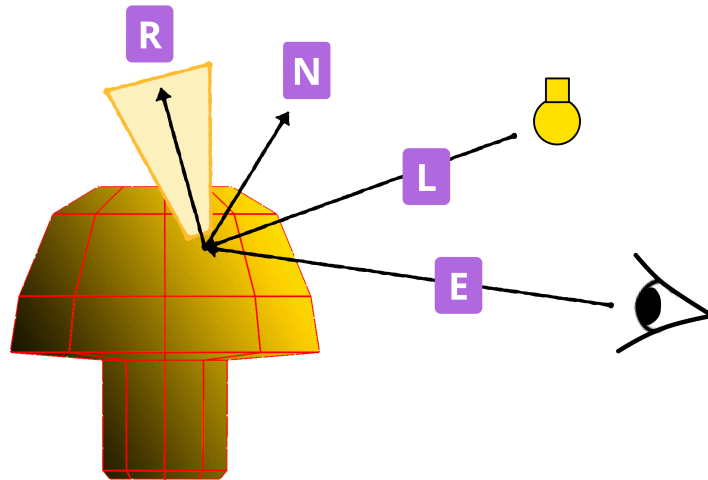
The eye position is just the x,y,z position.

In `lit_textured_fragment` we can now calculate the lighting.

Add this after the diffuse calculation:

```
// Specular  
float3 eye = normalize(vertexIn.eyePosition);
```

First normalize the eye position so that it's between 0 and 1. Here's that illustration again so you can visualize what's going on.



Reflect the light direction about the fragment's normal:

```
float3 reflection = reflect(light.direction, normal);
```

Metal Shading Language has a built-in `reflect` function.

Get the negative dot product of the reflection and the eye and clamp it between 0 and 1 using the built-in `saturate` function. Raise the result to the shininess power to get the specular factor:

```
float specularFactor = pow(saturate(-dot(reflection, eye)),  
                          vertexIn.shininess);
```

Now you can calculate the specular color from the light color, specular intensity and specular factor:

```
float3 specularColor = light.color * vertexIn.specularIntensity  
                    * specularFactor;
```

Change the final color assignment from:

```
color = color * float4((ambientColor + diffuseColor), 1);
```

to:

```
color = color * float4(ambientColor + diffuseColor + specularColor, 1);
```

That's all the code that's necessary for lighting :].

Now in `LightingScene`, set the specular properties for the mushroom in `init(device:size:)` before adding the mushroom to the scene:

```
mushroom.specularIntensity = 1.0  
mushroom.shininess = 8.0
```

And build and run:



The mushroom looks a bit plastic.

Try changing the settings to:

```
mushroom.specularIntensity = 0.2  
mushroom.shininess = 2.0
```

Build and run:



That's closer to a mushroom's real properties.

Hooray! You've now implemented a scene graph with a camera and lighting. You're ready to go to the next level and create a real game :].