材质

原文	Materials (http://learnopengl.com/#!Lighting/Materials)
作者	JoeyDeVries
翻译	Meow J
校对	暂未校对

在现实世界里，每个物体会对光产生不同的反应。比如说，钢看起来通常会比陶瓷花瓶更闪闪发光，木头箱子也不会像钢制箱子那样对光产生很强的反射。每个物体对镜面高光也有不同的反应。有些物体反射光的时候不会有太多的散射(Scatter)，因而产生一个较小的高光点，而有些物体则会散射很多，产生一个有着更大半径的高光点。如果我们想要在OpenGL中模拟多种类型的物体，我们必须为每个物体分别定义一个材质(Material)属性。

在上一节中，我们指定了一个物体和光的颜色，以及结合环境光和镜面强度分量，来定义物体的视觉输出。当描述一个物体的时候，我们可以用这三个分量来定义一个材质颜色(Material Color)：环境光照(Ambient Lighting)、漫反射光照(Diffuse Lighting)和镜面光照(Specular Lighting)。通过为每个分量指定一个颜色，我们就能够对物体的颜色输出有着精细的控制了。现在，我们再添加反光度(Shininess)这个分量到上述的三个颜色中，这就有我们需要的所有材质属性了：

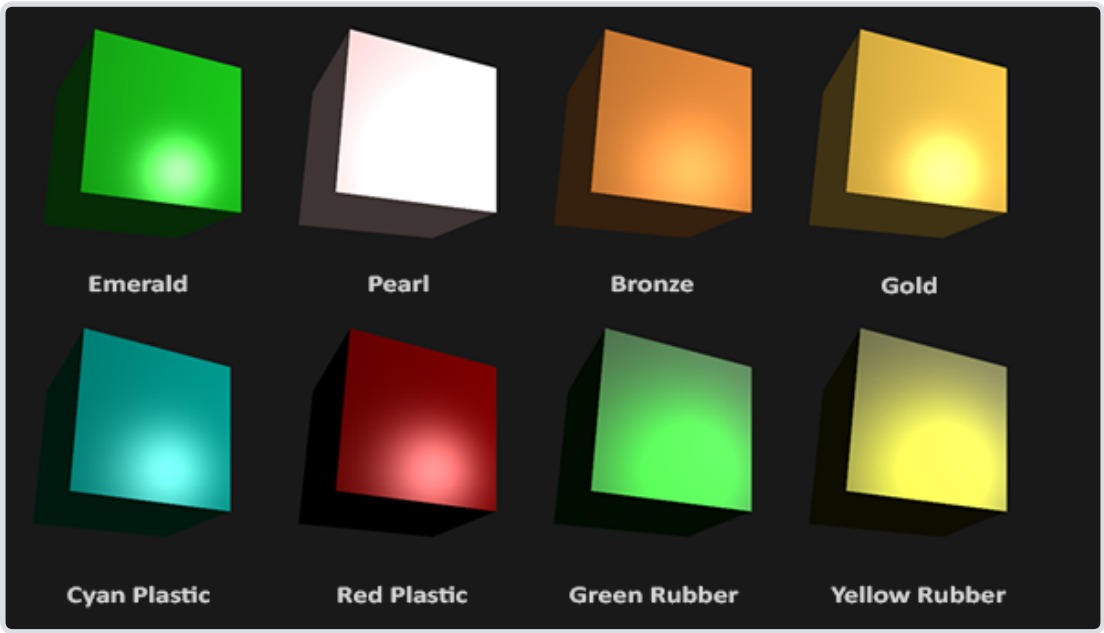
```
#version 330 core
struct Material {
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;
```

在片段着色器中，我们创建一个结构体(Struct)来储存物体的材质属性。我们也可以把它们储存为独立的uniform值，但是作为一个结构体来储存会更有条理一些。我们首先定义结构体的布局(Layout)，然后使用刚创建的结构体为类型，简单地声明一个uniform变量。

你可以看到，我们为每个冯氏光照模型的分量都定义一个颜色向量。`ambient`材质向量定义了在环境光照下这个物体反射得是什么颜色，通常这是和物体颜色相同的颜色。`diffuse`材质向量定义了在漫反射光照下物体的颜色。（和环境光照一样）漫反射颜色也要设置为我们需要的物体颜色。`specular`材质向量设置的是镜面光照对物体的颜色影响（或者甚至可能反射一个物体特定的镜面高光颜色）。最后，`shininess`影响镜面高光的散射/半径。

这四个元素定义了一个物体的材质，通过它们我们能够模拟很多现实世界中的材质。devernay.free.fr (http://devernay.free.fr/cours/opengl/materials.html)上的一个表格展示了几种材质属性，它们模拟了现实世界中的真实材质。下面的图片展示了几种现实世界的材质对我们的立方体的影响：



可以看到，通过正确地指定一个物体的材质属性，我们对这个物体的感知也就不同了。效果非常明显，但是要想获得更真实的效果，我们最终需要更加复杂的形状，而不单单是一个立方体。在后面的教程 (../../03 Model Loading/01 Assimp/)中，我们会讨论更复杂的形状。

为一个物体赋予一款合适的材质是非常困难的，这需要大量实验和丰富的经验，所以由于不合适的材质而毁了物体的视觉质量是件经常发生的事。

让我们在着色器中实现这样的一个材质系统。

设置材质

我们在片段着色器中创建了一个材质结构体的uniform，所以下面我们希望修改一下光照的计算来顺应新的材质属性。由于所有材质变量都储存在结构体中，我们可以从uniform变量`material`中访问它们：

```
void main()
{
    // 环境光
    vec3 ambient = lightColor * material.ambient;

    // 漫反射
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = lightColor * (diff * material.diffuse);

    // 镜面光
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    vec3 specular = lightColor * (spec * material.specular);

    vec3 result = ambient + diffuse + specular;
    FragColor = vec4(result, 1.0);
}
```

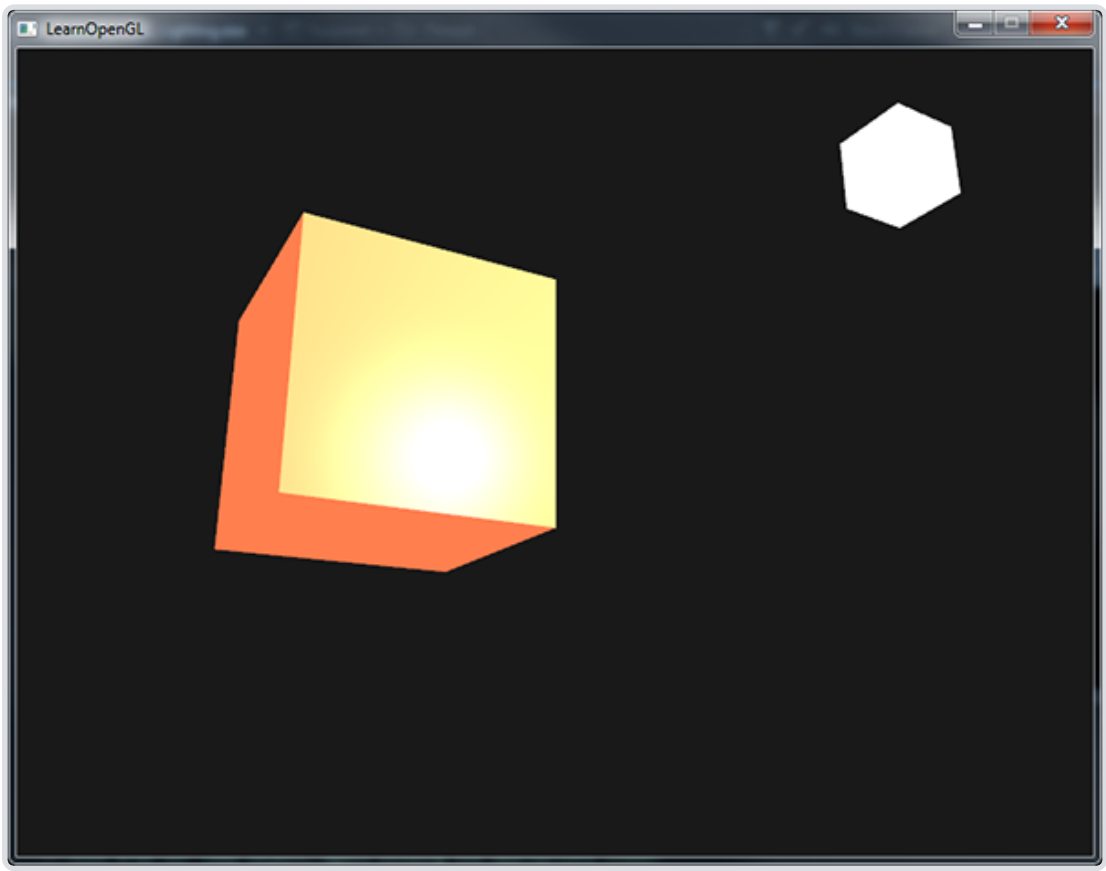
可以看到，我们现在在需要的地方访问了材质结构体中的所有属性，并且这次是根据材质的颜色来计算最终的输出颜色的。物体的每个材质属性都乘上了它们对应的光照分量。

我们现在可以在程序中设置适当的uniform，对物体设置材质了。GLSL中的结构体在设置uniform时并没有什么特别之处。结构体只是作为uniform变量的一个封装，所以如果想填充这个结构体的话，我们仍需要对每个单独的uniform进行设置，但这次要带上结构体名的前缀：

```
lightingShader.setVec3("material.ambient",  1.0f, 0.5f, 0.31f);
lightingShader.setVec3("material.diffuse",  1.0f, 0.5f, 0.31f);
lightingShader.setVec3("material.specular", 0.5f, 0.5f, 0.5f);
lightingShader.setFloat("material.shininess", 32.0f);
```

我们将环境光和漫反射分量设置成我们想要让物体所拥有的颜色，而将镜面分量设置为一个中等亮度的颜色，我们不希望镜面分量在这个物体上过于强烈。我们将反光度保持为32。现在我们能够程序中非常容易地修改物体的材质了。

运行程序，你应该会得到下面这样的结果：



但它看起来很奇怪不是吗？

光的属性

这个物体太亮了。物体过亮的原因是环境光、漫反射和镜面光这三个颜色对任何一个光源都会去全力反射。光源对环境光、漫反射和镜面光分量也具有着不同的强度。前面的教程，我们通过使用一个强度值改变环境光和镜面光强度的方式解决了这个问题。我们想做一个类似的系统，但是这次是要为每个光照分量都指定一个强度向量。如果我们假设`lightColor`是`vec3(1.0)`，代码会看起来像这样：

```
vec3 ambient  = vec3(1.0) * material.ambient;
vec3 diffuse  = vec3(1.0) * (diff * material.diffuse);
vec3 specular = vec3(1.0) * (spec * material.specular);
```

所以物体的每个材质属性对每一个光照分量都返回了最大的强度。对单个光源来说，这些`vec3(1.0)`值同样可以分别改变，而这通常就是我们想要的。现在，物体的环境光分量完全地影响了立方体的颜色，可是环境光分量实际上不应该对最终的颜色有这么大的影响，所以我们会将光源的环境光强度设置为一个小一点的值，从而限制环境光颜色：

```
vec3 ambient = vec3(0.1) * material.ambient;
```

我们可以用同样的方式修改光源的漫反射和镜面光强度。这和我们在上一节 (../02 Basic Lighting/)中所做的极为相似，你可以说我们已经创建了一些光照属性来影响每个单独的光照分量。我们希望为光照属性创建一个与材质结构体类似的结构体：

```
struct Light {
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

uniform Light light;
```

一个光源对它的`ambient`、`diffuse`和`specular`光照有着不同的强度。环境光照通常会设置为一个比较低的强度，因为我们不希望环境光颜色太过显眼。光源的漫反射分量通常设置为光所具有的颜色，通常是一个比较明亮的白色。镜面光分量通常会保持为`vec3(1.0)`，以最大强度发光。注意我们也将光源的位置添加到了结构体中。

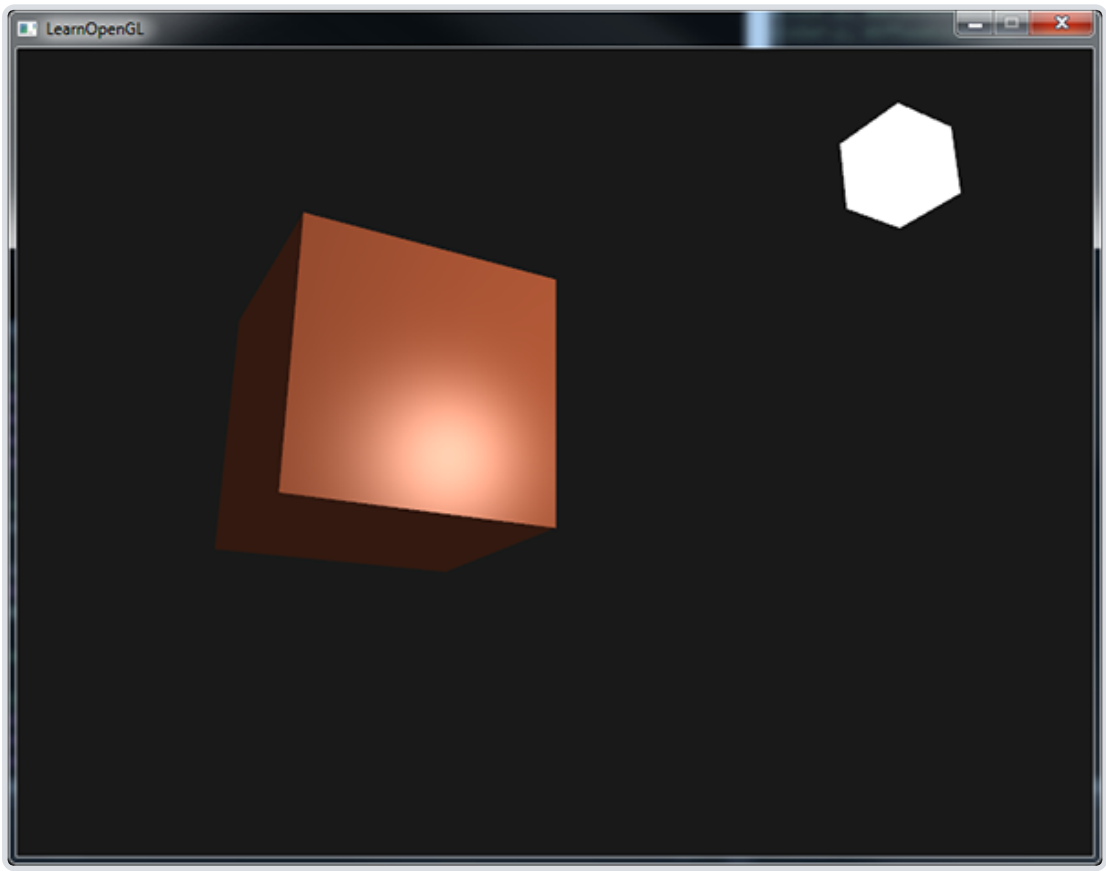
和材质uniform一样，我们需要更新片段着色器：

```
vec3 ambient  = light.ambient * material.ambient;
vec3 diffuse  = light.diffuse * (diff * material.diffuse);
vec3 specular = light.specular * (spec * material.specular);
```

我们接下来在程序中设置光照强度：

```
lightingShader.setVec3("light.ambient",  0.2f, 0.2f, 0.2f);
lightingShader.setVec3("light.diffuse",  0.5f, 0.5f, 0.5f); // 将光照调暗了一些以搭配场景
lightingShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);
```

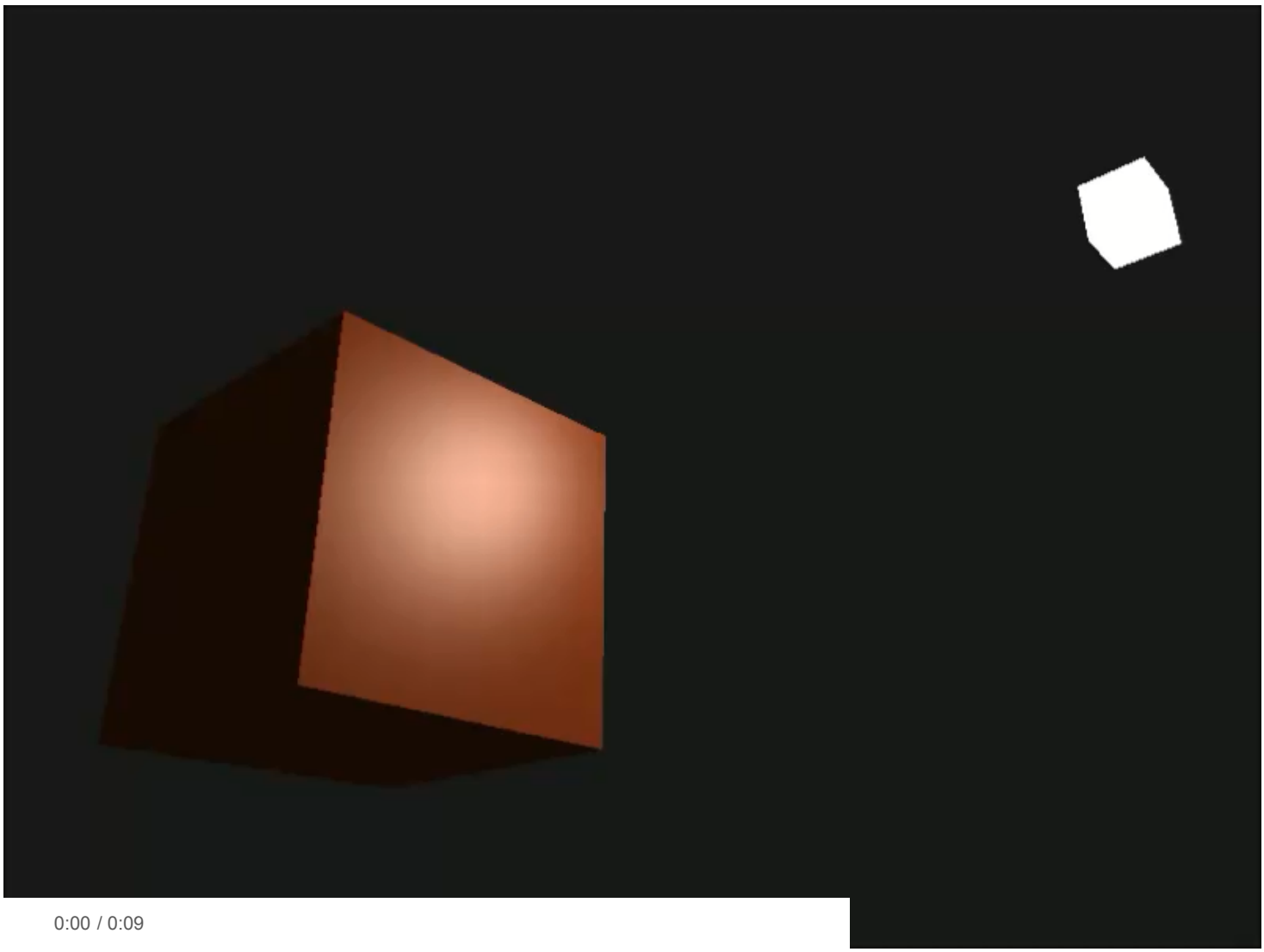
现在我们调整了光照对物体材质的影响，我们应该能得到一个更类似于上一节的视觉效果。但这次我们有了对光照和物体材质的完全掌控：



改变物体的视觉效果现在变得相对容易了。让我们做点更有趣的事！

不同的光源颜色

到目前为止，我们都只对光源设置了从白到灰到黑范围内的颜色，这样只会改变物体各个分量的强度，而不是它的真正颜色。由于现在能够非常容易地访问光照的属性了，我们可以随着时间改变它们的颜色，从而获得一些非常有意思的效果。由于所有的东西都在片段着色器中配置好了，修改光源的颜色非常简单，我们能够立刻创造一些很有趣的效果：



0:00 / 0:09

你可以看到，不同的光照颜色能够极大地影响物体的最终颜色输出。由于光照颜色能够直接影响物体能够反射的颜色（回想颜色 (../01 Colors/)这一节），这对视觉输出有着显著的影响。

我们可以利用`sin`和`glfwGetTime`函数改变光源的环境光和漫反射颜色，从而很容易地让光源的颜色随着时间变化：

```
glm::vec3 lightColor;
lightColor.x = sin(glfwGetTime() * 2.0f);
lightColor.y = sin(glfwGetTime() * 0.7f);
lightColor.z = sin(glfwGetTime() * 1.3f);

glm::vec3 diffuseColor = lightColor   * glm::vec3(0.5f); // 降低影响
glm::vec3 ambientColor = diffuseColor * glm::vec3(0.2f); // 很低的影响

lightingShader.setVec3("light.ambient", ambientColor);
lightingShader.setVec3("light.diffuse", diffuseColor);
```

尝试并实验一些光照和材质值，看看它们是怎样影响视觉输出的。你可以在这里(https://learnopengl.com/code_viewer_gh.php?code=src/2.lighting/3.1.materials/materials.cpp)找到程序的源码。

练习

- 你能像教程一开始那样，定义相应的材质来模拟现实世界的物体吗？注意材质表格(http://devernay.free.fr/cours/opengl/materials.html)中的环境光值可能与漫反射值不一样，它们没有考虑光照的强度。要想纠正这一问题，你需要将所有的光照强度都设置为`vec3(1.0)`，这样才能得到正确的输出：参考解答 (https://learnopengl.com/code_viewer_gh.php?code=src/2.lighting/3.2.materials_exercise1/materials_exercise1.cpp)，我做的是青色塑料(Cyan Plastic)的箱子。