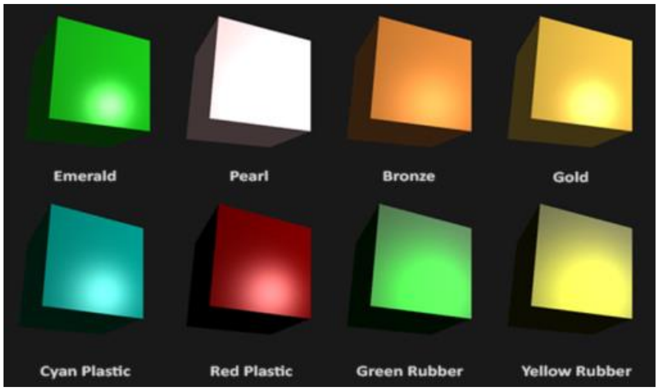
MATERIALS

If you want to simulate several different types of objects in OpenGL, you have to define **material** properties specific to each surface.



When describing a surface, we can define a material color for each of the 3 lighting components: ambient, diffuse, and specular lighting.

- By specifying a color for each of the components, we have fine-grained control over the color output of the surface.
- Add a shininess to those 3 colors and we have all the material properties we need.

Create a Material struct in the fragment shader for storing material properties and declare a material uniform variable.

- Material Struct

Setting Materials

Implement the material properties in the fragment shader main function.

- Material Struct Usage

The fragment shader is now much more modular than before, allowing for setting up many different types of materials.

Test out different materials by setting the uniforms.

- Setting Material Properties

If you followed the example code, the light is now having too much of an effect on the cube. To solve this:

1. *Create a Light struct for all the properties of a light and implment those properties into the fragment shader main function.*
   - Adjusting Light Intensities

Changing the visual aspects of objects is relatively easy now.

Different Light Colors

As it stands, if you use colored lights, the object's reflected color changes drastically. This is simply how light works.

- Remember, an object's color is dependent on what light it reflects. Therefore, if there is only red light in the scene, the object can only be a shade of red (or black, if the object absorbs *all* of the light).

Change the color of the light over time.

- Testing Colored Lights

EXERCISES

Can you make it so that changing the light color changes the color of the light cube object?
Can you simulate some of the real-world objects by defining their respective materials like we've seen at the start of the this chapter?

- Table of Material Properties Values
- **NOTE:** The table's ambient values are not the same as the diffuse values; they didn't take light intensities into account. To correctly set their values, you'd have to set all the light intensities to `vec(1.0)` to get the same output.

Material Struct

Thursday, March 31, 2022 10:46 PM

```
#version 330 core
struct Material {
    vec3 ambient; // color the surface reflects under ambient lighting
    vec3 diffuse; // color of the surface under diffuse lighting
    vec3 specular; // color of the specular highlight on the surface
    float shininess; // scattering/radius of the specular highlight
};

uniform Material material;
```

Material Struct Usage

Thursday, March 31, 2022 11:07 PM

Here's the before and after of using the material struct in the default.frag shader:

Before

```
#version 330 core

in vec3 Normal;
in vec3 FragPos;

out vec4 FragColor;

uniform vec3 objectColor;
uniform vec3 lightColor;
uniform vec3 lightPos;

uniform vec3 viewPos;

void main() {
   // Ambient Lighting
   float ambientStrength = 0.5;
   vec3 ambient = ambientStrength * lightColor;

   // Diffuse Lighting
   vec3 norm = normalize(Normal);
   vec3 lightDir = normalize(lightPos - FragPos);
   float diff = max(dot(norm, lightDir), 0.0);
   vec3 diffuse = diff * lightColor;

   // Specular Lighting
   float specularStrength = 0.5;
   vec3 viewDir = normalize(viewPos - FragPos);
   vec3 reflectDir = reflect(-lightDir, norm);
   float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
   vec3 specular = specularStrength * spec * lightColor;

   vec3 finalColor = (specular + diffuse + ambient) * objectColor;

   FragColor = vec4(finalColor, 1.0);
}
```

After

```
#version 330 core

```

```
struct Material {
   vec3 ambient; // color the surface reflects under ambient lighting
   vec3 diffuse; // color of the surface under diffuse lighting
   vec3 specular; // color of the specular highlight on the surface
   float shininess; // scattering/radius of the specular highlight
};

uniform Material material;

in vec3 Normal;
in vec3 FragPos;

out vec4 FragColor;

uniform vec3 objectColor;
uniform vec3 lightColor;
uniform vec3 lightPos;

uniform vec3 viewPos;

void main() {
   // Ambient Lighting
   vec3 ambient = lightColor * material.ambient;

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

   // Specular Lighting
   float specularStrength = 0.5;
   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 finalColor = (specular + diffuse + ambient) * objectColor;

   FragColor = vec4(finalColor, 1.0);
}
```

Setting Material Properties

Thursday, March 31, 2022 11:15 PM

```
// Sets the material properties
glUniform3fv(glGetUniformLocation(shaderProgram.id, "material.ambient"), 1, glm::value_ptr(glm::vec3(1.0f, 0.5f, 0.31f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "material.diffuse"), 1, glm::value_ptr(glm::vec3(1.0f, 0.5f, 0.31f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "material.specular"), 1, glm::value_ptr(glm::vec3(0.5f, 0.5f, 0.5f)));
glUniform1f(glGetUniformLocation(shaderProgram.id, "material.shininess"), 32.0f);
```

Notice any strange about the result?

- The object is very bright. Initially, you might think this is the fault of the values that were set, but it's actually because we *removed* the intensity values (ambientStrength, specularStrength) that controlled the brightness of the reflected color.

Adjusting Light Intensities

Thursday, March 31, 2022 11:41 PM

<u>default.frag</u>

```
struct Light {
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

uniform Light light;
```

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

<u>Render Loop</u>

```
// Sets the light properties
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.position"), 1, glm::value_ptr(lightPos));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.ambient"), 1, glm::value_ptr(glm::vec3(0.2f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.diffuse"), 1, glm::value_ptr(glm::vec3(0.5f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.specular"), 1, glm::value_ptr(glm::vec3(1.0f)));
```

Testing Colored Lights

Thursday, March 31, 2022 11:52 PM

```
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);
glm::vec3 ambientColor = diffuseColor * glm::vec3(0.2f);

glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.ambient"), 1, glm::value_ptr(ambientColor));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.diffuse"), 1, glm::value_ptr(diffuseColor));
```