LIGHTING MAPS

In the previous chapter, we defined a material for an entire object as a whole. Objects in the real world, however, usually do not consist of a single material, but of multiple materials.

- Example: Think of a car; its interior consists of a shiny fabric, it has windows that partly reflect the surrounding environment, its tires are dull, and its wheels are shiny. A car also displays many different ambient and diffuse colors.

We will extend the material system from the previous chapter by introducing diffuse and specular maps.

- These allow us to influence the diffuse (and indirectly the ambient component because they should always be the same) and specular components of an object with much more precision.

Diffuse Maps

Diffuse maps are similar to textures, just in a lit scene.

- A diffuse map is just an image wrapped around an object that we can index for unique color values per fragment, same as a texture.

Configure the fragment shader to use a diffuse map.

- Use this new container texture: Container with Steel Border
- Diffuse Map

Specular Maps

You probably noticed that the wood is much shinier than it should be. We need to be able to control what parts of the object show a specular highlight with varying intensity.

- We can also use a texture map just for specular highlights.
  - Specular maps are textures that are usually black and white.
  - The intensity of the specular highlight comes from the brightness of each pixel in the specular map.

Configure the fragment shader to use a specular map.

- Use this specular map texture: Specular Map for Container w/ Border
- Specular Map

EXERCISES

1. Fool around with the light source's ambient, diffuse, and specular vectors and see how they affect the visual output of the container.
2. Try inverting the color of the specular map in the fragment shader so that the wood shows specular highlights and the steel borders do not (note that, due to the cracks in the steel border, the borders still show some specular highlight, although with less intensity).
3. Try creating a specular map from the diffuse texture that uses actual colors instead of black and white and see that the result doesn't look too realistic. You can use this colored specular map if you can't generate one yourself.
4. Add an emission map, which is a texture that stores emission values per fragment. Emission values are colors an object may emits as if it contains a light source itself; this way, an object can glow regardless of the light conditions. Emission maps are often what you see when objects in a game glow (like eyes of a robot, or light strips on a container).
   - Add this texture as an emission map onto the container as if the letters emit light: Emission Map Texture

Diffuse Map

Monday, April 4, 2022 9:44 PM

Don't need the ambient colors vector because the ambient color is based on the diffuse color anyways.
NOTE: `sampler2D` is an **opaque type**, meaning we can't instantiate it. We can only define uniforms of that type (e.g. can't be used as a function parameter).

default.frag

```
struct Material {
    sampler2D diffuse;
    vec3 specular;
    float shininess;
};
...
in vec2 TexCoords;
...
void main() {
   vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
   ...
   vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
   ...
   FragColor = vec4(ambient + diffuse + specular, 1.0);
}
```

default.vert

```
...
layout (location = 2) in vec2 aTexCoords;
...
out vec2 TexCoords;
...
void main() {
   ...
   TexCoords = aTexCoords;
}
```

main.cpp

```
Texture2D containerDiffuseMap{ "Textures/container2.png", 0 };
```

Render Loop

```
containerDiffuseMap.bind();

glUniform1i(glGetUniformLocation(shaderProgram.id, "material.diffuse"), containerDiffuseMap.getTexUnit());
```

Specular Map

Monday, April 4, 2022 10:28 PM

default.frag

```
#version 330 core

struct Material {
   sampler2D diffuse;
   sampler2D specular;
   float shininess; // scattering/radius of the specular highlight
};
...
void main() {
   ...
   vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));

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

main.cpp

```
Texture2D containerSpecularMap{ "Textures/container2_specular.png", 1 };
```

Render Loop

```
containerSpecularMap.bind();

glUniform1i(glGetUniformLocation(shaderProgram.id, "material.specular"), containerSpecularMap.getTexUnit());
```