MULTIPLE LIGHTS

To use more than one light source in the scene, we are going to want to do the lighting calculations from GLSL functions to keep the code organized, since multiple light types can cause calculations to get messy.

Functions in GLSL are just like C-functions (function name, return type, prototype before being called, etc.)

When using multiple lights in a scene:

1. Designate a single color vector that represents the fragment's output color.
2. For each light, the light's contribution to the fragment is added to this output color vector.
   - Each light will calculate its individual impact and contribute that to the final output color.

Here's an example of what the above fragment coloring structure would look like: Example Fragment Color Calculations

Directional Light

Create a struct for storing directional light data and define a function that returns the light value of a fragment based on the directional light.

- Directional Light Function

Point Light

Create a struct for storing point light data and define a function that returns the light value of a fragment based on the point light.

- Point Light Function

Spotlight

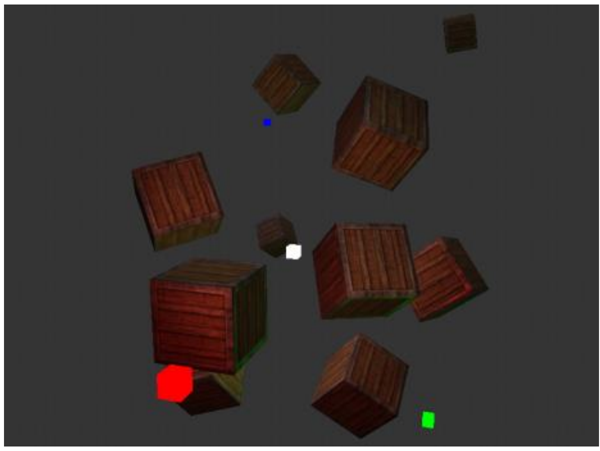
Create a struct for storing spotlight data and define a function that returns the light value of a fragment based on the spotlight.

- Spotlight Function

Putting It All Together

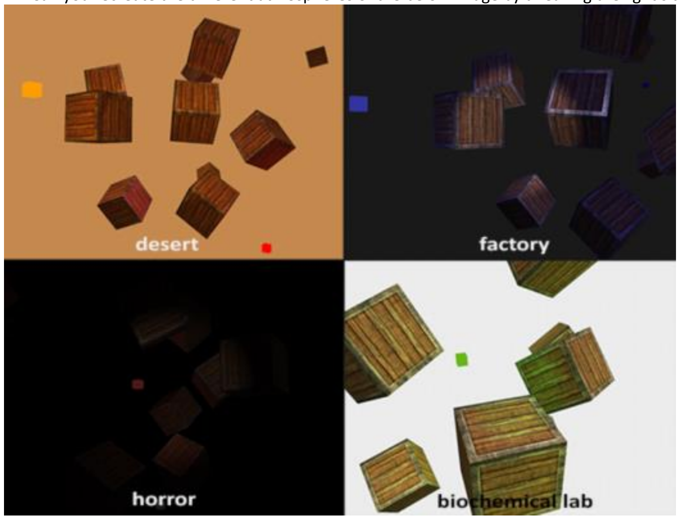
Implement the new main function that calculates the color of a fragment based on the light values generated from the functions.

- Use these point light positions and colors: Point Light Positions/Colors
- Multiple Lights Implementation



EXERCISES

1. Can you recreate the different atmospheres of the below image by tweaking the light's attribute values?



Example Fragment Color Calculations

Monday, April 11, 2022 3:16 PM

```
out vec4 FragColor;

void main() {
   // define an output color value
   vec3 output = vec3(0.0);
   // add the directional light's contribution to the output
   output += someFunctionToCalculateDirectionalLight();
   // do the same for all point lights
   for(int i = 0; i < nr_of_point_lights; i++)
      output += someFunctionToCalculatePointLight();
   // and add others lights as well (like spotlights)
   output += someFunctionToCalculateSpotLight();

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

Directional Light Function

Monday, April 11, 2022 3:33 PM

default.frag

```
...

struct DirLight {
   vec3 direction;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};
uniform DirLight dirLight;

...

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);

void main() {
...
}

// Returns the new fragment color based on a directional light.
vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir) {
   // Ambient Lighting
   vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));

   // Diffuse Lighting
   vec3 lightDir = normalize(-light.direction);
   float diff = max(dot(normal, lightDir), 0.0);
   vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));

   // Specular Lighting
   vec3 reflectDir = reflect(-lightDir, normal);
   float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
   vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));

   return (ambient + diffuse + specular);
}
```

Point Light Function

Monday, April 11, 2022 3:48 PM

default.frag

```
...

struct PointLight {
   vec3 position;

   float constant;
   float linear;
   float quadratic;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};
#define NUM_POINT_LIGHTS 4
uniform PointLight pointLight[NUM_POINT_LIGHTS];

...

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir);

void main() {
...
}

// Returns the new fragment color based on a point light.
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{
   // Attenuation
   float distance = length(light.position - fragPos);
   float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic * (distance * distance));

   // Ambient Lighting
   vec3 ambient  = light.ambient * vec3(texture(material.diffuse, TexCoords));

   // Diffuse Lighting
   vec3 lightDir = normalize(light.position - fragPos);
   float diff = max(dot(normal, lightDir), 0.0);
   vec3 diffuse  = light.diffuse  * diff * vec3(texture(material.diffuse, TexCoords)) * attenuation;

   // Specular Lighting
   vec3 reflectDir = reflect(-lightDir, normal);
   float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
   vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords)) * attenuation;

   return (ambient + diffuse + specular);
}
```

NOTE: We used a pre-processor directive (`#define`) to declare that we want to use 4 point lights (`NUM_POINT_LIGHTS 4`). We then created an array of `PointLight` structs to store data for 4 different point lights.

Spotlight Function

Monday, April 11, 2022 5:00 PM

default.frag

```
...

struct SpotLight {
   vec3 position;
   vec3 direction;

   float innerCutoff;
   float outerCutoff;

   float constant;
   float linear;
   float quadratic;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};
uniform SpotLight spotLight;

...

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir);
vec3 CalcSpotLight(SpotLight light, vec3 normal, vec3 fragPos, vec3 viewDir);

void main() {
...
}

...

// Returns the new fragment color based on a spotlight.
vec3 CalcSpotLight(SpotLight light, vec3 normal, vec3 fragPos, vec3 viewDir) {
   vec3 lightDir = normalize(light.position - fragPos);

   // Calculating Attenuation
   float distance = length(light.position - fragPos);
   float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic * (distance * distance));

   // Calculating Spotlight Intensity
   float theta = dot(lightDir, normalize(-light.direction));
   float epsilon = light.innerCutoff - light.outerCutoff;
   float intensity = clamp((theta - light.outerCutoff) / epsilon, 0.0, 1.0); // ensures intensity is between 0.0 and 1.0

   // Ambient Lighting
   vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));

   // Diffuse Lighting
   float diff = max(dot(normal, lightDir), 0.0);
   vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords)) * attenuation * intensity;

   // Specular Lighting
   vec3 reflectDir = reflect(-lightDir, normal);
   float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
   vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords)) * attenuation * intensity;

   return (ambient + diffuse + specular);
}
```

Point Light Positions/Colors

Monday, April 11, 2022 5:05 PM

main.cpp

```
glm::vec3 pointLightPositions[] = {
   glm::vec3(0.7f,  0.2f,  2.0f),
   glm::vec3(2.3f, -3.3f, -4.0f),
   glm::vec3(-4.0f,  2.0f, -12.0f),
   glm::vec3(0.0f,  0.0f, -3.0f)
};

glm::vec3 pointLightColors[] = {
   glm::vec3(1.0f, 0.0f, 0.0f),
   glm::vec3(0.0f, 1.0f, 0.0f),
   glm::vec3(0.0f, 0.0f, 1.0f),
   glm::vec3(1.0f, 1.0f, 1.0f)
};
```

Multiple Lights Implementation

Monday, April 11, 2022 5:07 PM

<u>default.frag</u>

```
void main() {
   vec3 norm = normalize(Normal);
   vec3 viewDir = normalize(viewPos - FragPos);

   vec3 color = CalcDirLight(dirLight, norm, viewDir);
   for (int i = 0; i < NUM_POINT_LIGHTS; ++i) {
      color += CalcPointLight(pointLights[i], norm, FragPos, viewDir);
   }
   color += CalcSpotLight(spotLight, norm, FragPos, viewDir);

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

<u>Render Loop</u>

```
// Binds the light cube object
glBindVertexArray(lightVAO);

for (int i = 0; i < 4; ++i) {
   glm::mat4 lightModel{ 1.0f };
   lightModel = glm::translate(lightModel, pointLightPositions[i]);
   lightModel = glm::scale(lightModel, glm::vec3(0.2f));
   glUniformMatrix4fv(glGetUniformLocation(lightShaderProgram.id, "model"), 1, GL_FALSE, glm::value_ptr(lightModel));
   glUniform3fv(glGetUniformLocation(lightShaderProgram.id, "lightColor"), 1, glm::value_ptr(pointLightColors[i]));
   // Renders the light cube object to the screen
   glDrawArrays(GL_TRIANGLES, 0, 36);
}

...

// Sets the directional light properties
glUniform3fv(glGetUniformLocation(shaderProgram.id, "dirLight.direction"), 1, glm::value_ptr(-glm::vec3(0.0, 1.0, 0.0)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "dirLight.ambient"), 1, glm::value_ptr(glm::vec3(0.05f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "dirLight.diffuse"), 1, glm::value_ptr(glm::vec3(0.4f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "dirLight.specular"), 1, glm::value_ptr(glm::vec3(0.5f)));

// Sets the 4 point lights' properties
for (int i = 0; i < 4; ++i) {
   std::string index = std::to_string(i);
   glUniform3fv(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].position").c_str()), 1,
      glm::value_ptr(pointLightPositions[i]));
   glUniform3fv(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].ambient").c_str()), 1,
      glm::value_ptr(glm::vec3(0.05f) * pointLightColors[i]));
   glUniform3fv(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].diffuse").c_str()), 1,
      glm::value_ptr(glm::vec3(0.8f) * pointLightColors[i]));
   glUniform3fv(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].specular").c_str()), 1,
      glm::value_ptr(glm::vec3(1.0f) * pointLightColors[i]));
   glUniform1f(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].constant").c_str()), 1.0f);
   glUniform1f(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].linear").c_str()), 0.09f);
   glUniform1f(glGetUniformLocation(shaderProgram.id, ("pointLights[" + index + "].quadratic").c_str()), 0.032f);
}

// Sets the spotlight properties
glUniform3fv(glGetUniformLocation(shaderProgram.id, "spotLight.position"), 1, glm::value_ptr(camera.position));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "spotLight.direction"), 1, glm::value_ptr(camera.front));
glUniform1f(glGetUniformLocation(shaderProgram.id, "spotLight.innerCutoff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(shaderProgram.id, "spotLight.outerCutoff"), glm::cos(glm::radians(17.5f)));
glUniform1f(glGetUniformLocation(shaderProgram.id, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(shaderProgram.id, "spotLight.linear"), 0.09f);
glUniform1f(glGetUniformLocation(shaderProgram.id, "spotLight.quadratic"), 0.032f);
glUniform3fv(glGetUniformLocation(shaderProgram.id, "spotLight.ambient"), 1, glm::value_ptr(glm::vec3(0.0f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "spotLight.diffuse"), 1, glm::value_ptr(glm::vec3(1.0f)));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "spotLight.specular"), 1, glm::value_ptr(glm::vec3(1.0f)));

// Binds the object we want to render
glBindVertexArray(vao);

// Renders 10 objects to the screen
for (int i = 0; i < 10; i++) {
   glm::mat4 objModel = glm::mat4(1.0f);
   objModel = glm::translate(objModel, cubePositions[i]);
   float angle = 20.0f * i;
   objModel = glm::rotate(objModel, glm::radians(angle), glm::vec3(1.0f, 0.3f, 0.5f));
   glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "model"), 1, GL_FALSE, glm::value_ptr(objModel));
   // Renders the object to the screen
   glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

NOTE: Some of the ambient values are being passed to the shader are low or even 0.0. This is because, since the main function of the fragment shader adds all of these ambient lights together, passing higher values for each could cause the whole scene to be fully lit, which is not good for testing other lights.