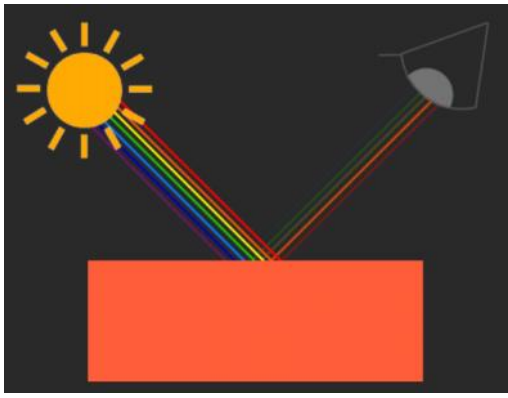
COLORS



In the real world, colors can take any color value within the visible spectrum, with each object having its own color(s).
However, in the digital world, we need to map the (infinite) real colors to (limited) digital values. Therefore, not all real-world colors can be represented digitally.

Colors are digitally represented using **RGB** components (red, green, blue).
- Using different combinations of these 3 color components is how we obtain different colors, digitally.
- Example: You can get a coral color using an RGB value of `(1.0, 0.5, 0.31)`.

The color of an object (in both the real and digital world) is directly influenced by the color of light that is being cast onto it.
- Example: If you shine a red light in a completely dark room, everything you see in the room will appear as some shade of red.
- By multiplying the color of a light with the color of a fragment, we can generate the reflected color of the object (its **perceived color**).

Calculate the resulting color of a white light shining on a coral-colored object. Then, do the same with a blue light. Finally, do the same with a light with color values between 0.2 and 0.8. Observe the results.
- You can just use a vec3 as a representation of an object.
- At this point, you can strip the color and texture coordinate data from your vertex data. If you do this, make sure to update your vertex attribute pointers!
- Color of Object In Different Lights

Create another cube that will act as a light source.
- You will need to create another VAO object for this new object.
- Light Source Cube

One thing we want to do is prevent the light cube's color from being updated by changing the lighting shaders later on.

Create a second set of shaders for drawing the light cube.
- Light Source Shaders

The main purpose of the light source cube is to show where the light is coming from.

Render the light cube at (1.2f, 1.0f, 2.0f) at 20% size.
- Rendering the Light Cube

Color of Object In Different Lights

Monday, March 28, 2022 1:57 PM

```
glm::vec3 whiteLightColor{ 1.0f, 1.0f, 1.0f };
glm::vec3 blueLightColor{ 0.0f, 1.0f, 0.0f };
glm::vec3 pinkLightColor{ 0.67f, 0.23f, 0.59f };

glm::vec3 objColor{ 1.0f, 0.5f, 0.31f }; // coral color

glm::vec3 wlResult = whiteLightColor * objColor; // wlResult == objColor because the light is white
glm::vec3 blResult = blueLightColor * objColor; // blResult = (0.0f, 0.5f, 0.0f) <- Only blue
glm::vec3 plResult = pinkLightColor * objColor; // plResult = (0.67f, 0.12f, 0.18f) <- A mix between both colors
```

NOTE: The object is *absorbing* a large potion of the light, but is reflecting several red, blue, and green values based on its own color value. This is (kin d of) how colors work in real life.

Light Source Cube

Monday, March 28, 2022 2:39 PM

default.frag

```
#version 330 core

out vec4 FragColor;

uniform vec3 objectColor;
uniform vec3 lightColor;

void main() {
   FragColor = vec4(objectColor * lightColor, 1.0);
}
```

main.cpp

```
// Creates the VAO for a light cube
unsigned int lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// The vbo already contains the buffer data for a cube
glBindBuffer(GL_ARRAY_BUFFER, vbo);

// Sets the aPos layout
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

// Unbinds the VAO and VBO
glBindVertexArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);

// Object and light colors
glm::vec3 objColor{ 1.0f, 0.5f, 0.31f }; // coral color
glm::vec3 lightColor{ 1.0f, 1.0f, 1.0f }; // white color
```

Render Loop

```
// Set the objectColor and lightColor uniforms
glUniform3fv(glGetUniformLocation(shaderProgram.id, "objectColor"), 1, glm::value_ptr(objColor));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "lightColor"), 1, glm::value_ptr(lightColor));
```

Light Source Shaders

Monday, March 28, 2022 2:59 PM

light.vert

This is the same as default.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
   gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

light.frag

```
#version 330 core

out vec4 FragColor;

void main() {
   FragColor = vec4(1.0); // sets all 4 vector values to 1.0
}
```

Rendering the Light Cube

Monday, March 28, 2022 4:18 PM

main.cpp

```
// Creates the lighting shader program
Shader lightShaderProgram{ "Shaders/light.vert", "Shaders/light.frag" };

// Object and light colors
glm::vec3 objColor{ 1.0f, 0.5f, 0.31f }; // coral color
glm::vec3 lightColor{ 1.0f, 1.0f, 1.0f }; // white color
// Light position
glm::vec3 lightPos = { 1.2f, 1.0f, 2.0f };
```

Render Loop

```
// Creates the model matrix for the light cube
glm::mat4 lightModel{ 1.0f };
lightModel = glm::translate(lightModel, lightPos);
lightModel = glm::scale(lightModel, glm::vec3(0.2f));

// Creates the model matrix for the object
glm::mat4 objModel = glm::mat4(1.0f);

// Creates the view matrix (camera)
glm::mat4 view = camera.GetViewMatrix();
// Creates the projection matrix using perspective projection
glm::mat4 projection = glm::perspective(glm::radians(camera.fov), (float)WIDTH / (float)HEIGHT, 0.1f, 100.0f);

// Uses the lighting shader program
lightShaderProgram.use();
// Sets the uniforms for the model, view and projection matrices in the lighting vertex shader
glUniformMatrix4fv(glGetUniformLocation(lightShaderProgram.id, "model"), 1, GL_FALSE, glm::value_ptr(lightModel));
glUniformMatrix4fv(glGetUniformLocation(lightShaderProgram.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(lightShaderProgram.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

// Binds the light cube object
glBindVertexArray(lightVAO);
// Renders the light cube object to the screen
glDrawArrays(GL_TRIANGLES, 0, 36);

// Uses the shader program
shaderProgram.use();
// Sets the uniforms for the model, view and projection matrices in the default vertex shader
glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "model"), 1, GL_FALSE, glm::value_ptr(objModel));
glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
// Sets the objectColor and lightColor uniforms in the default fragment shader
glUniform3fv(glGetUniformLocation(shaderProgram.id, "objectColor"), 1, glm::value_ptr(objColor));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "lightColor"), 1, glm::value_ptr(lightColor));

// Binds the object we want to render
glBindVertexArray(vao);
// Renders the object to the screen
glDrawArrays(GL_TRIANGLES, 0, 36);
```