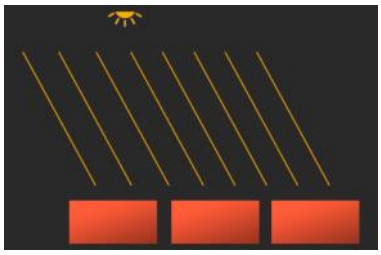
LIGHT CASTERS

A light source that casts light upon objects is called a light caster.

Common light casters are directional lights, point lights, and spotlights.

Directional Light



When a light source is far away, the light rays coming from the light source are close to parallel to each other; it looks li ke all the light rays are coming from the same direction, regardless of where the object and/or the viewer is (e.g. sunlight).

A directional light is a light caster that has all its light rays pointing in the same direction (independent of the location of the light source), and is modeled to be infinitely far away.

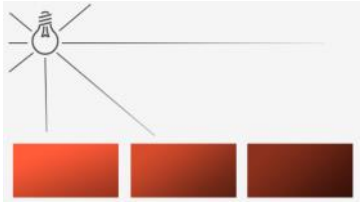
- Because all the light rays are parallel, it does not matter how each object relates to the light source's position, since the light direction remains the same for each object in the scene. Therefore, the lighting calculations will be similar for each object in the scene.

Configure the scene to use directional lighting.

- Use these cube positions to generate multiple cubes for viewing the effect of directional light: Cube Positions
- Directional Light

The light will no longer appear to come from the light source cube, since the light is now coming from a fixed direction rath er than a position in space.

Point Lights



A point light is a light caster with a given position somewhere in the world that illuminates in all directions, where the light rays fade over distance (e.g. light bulbs or torches).

- **NOTE:** We've basically already implemented a point light with the light we've been using; we just haven't made the light fade over distance.

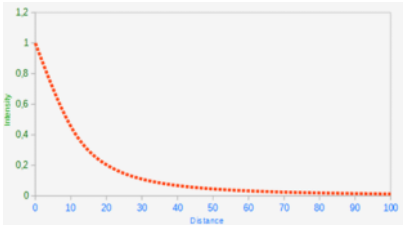
Attenuation

To reduce the intensity of light over the distance a light ray travels is called attenuation.

- You might think that you could just use a linear equation to reduce the light intensity over distance, but that looks a bit fake.
- Instead, most people use this forumla for attenuation based on a fragment's distance to the light source, which is later multiplied with the light's intensity vector:

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

  - d - distrance from the fragment to the light source.
  - K_c (constant) - generally kept at 1.0 to prevent the denominator from being less than 1 and accidentally boost the intensity at certain distances.
  - K_l (linear) - multiplied with the distance value that reduces the intensity in a linear fashion.
  - K_q (quadratic) - multiplied with the quadrant of the distance and sets a quadratic decrease of intensity for the light source.
    - The quadratic term is less significant when the distance is small, but much larger as the distance grows.



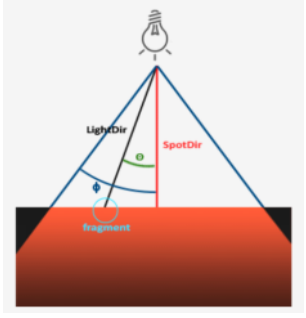
Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

The table to the right shows possible values you can use simulate a point light, using the above attenuation equation, that spans a certain distance.

Configure the scene to use point lighting with attenuation.

- Point Light

Spotlight



A spotlight is a light caster that is located somewhere in the environment that, instead of shooting light rays in all directions, only shoots them in a specific direction.

- Only the objects within a certain radius of the spotlight's direction are lit and everything else stays dark (e.g. flashlight).
- For each fragment, we calculate if the fragment is between the spotlight's cutoff directions. If so, we light the fragment accordingly.

Look at the image to the right:

- LightDir - The vector pointing from the fragment to the light source.
- SpotDir - The direction the spotlight is aiming at.
- Phi ϕ - The cutoff angle that specifies the spotlight's radius. Everything outside this angle is not lit by the spotlight.
- Theta Θ - The angle between the LightDir vector and the SpotDir vector. Θ should be smaller than ϕ to be inside the spotlight.

Configure a spotlight coming from the camera (like a flashlight).

- Flashlight

Smooth/Soft Edges

To create the effect of a smoothly-edged spotlight, we want to simulate a spotlight having an inner and outer cone. The light we have set up works as the inner cone, but the outer cone needs to gradually dim the light from the inner cone to the edges of the outer cone.

- If a fragment is between the inner cone and outer cone, it should calculate an intensity value between 0.0 and 1.0.
- We can use this equation to calculate the light intensity value of a fragment in between the inner and outer cone:

$$I = \frac{(\Theta - \gamma)}{\epsilon}$$

  - ϵ - The cosine difference between the inner (ϕ) and the outer cone (γ) ($\epsilon = \phi - \gamma$).
  - The resulting I value is then the intensity of the spotlight at the current fragment.

θ	θ in degrees	ϕ (inner cutoff)	ϕ in degrees	γ (outer cutoff)	γ in degrees	ϵ	I
0.87	30	0.91	25	0.82	35	0.91 - 0.82 = 0.09	0.87 - 0.82 / 0.09 = 0.56
0.9	26	0.91	25	0.82	35	0.91 - 0.82 = 0.09	0.9 - 0.82 / 0.09 = 0.89
0.97	14	0.91	25	0.82	35	0.91 - 0.82 = 0.09	0.97 - 0.82 / 0.09 = 1.67
0.83	34	0.91	25	0.82	35	0.91 - 0.82 = 0.09	0.83 - 0.82 / 0.09 = 0.11
0.64	50	0.91	25	0.82	35	0.91 - 0.82 = 0.09	0.64 - 0.82 / 0.09 = -2.0
0.966	15	0.9978	12.5	0.953	17.5	0.9978 - 0.953 = 0.0448	0.966 - 0.953 / 0.0448 = 0.29

The table to the right shows possible values you can use simulate a spotlight with smooth edges, using the above equation.

Smooth the edges of the spotlight using the equation above.

- Smoothed Spotlight

EXERCISES

Try experimenting with all the different light types and their fragment shaders. Try inverting some vectors and/or use < inst ead of >. Try to explain the different visual outcomes.

Cube Positions

Tuesday, April 5, 2022 3:16 PM

```
// Positions of cube objects in world space
glm::vec3 cubePositions[] = {
   glm::vec3(0.0f,  0.0f,  0.0f),
   glm::vec3(2.0f,  5.0f, -15.0f),
   glm::vec3(-1.5f, -2.2f, -2.5f),
   glm::vec3(-3.8f, -2.0f, -12.3f),
   glm::vec3(2.4f, -0.4f, -3.5f),
   glm::vec3(-1.7f,  3.0f, -7.5f),
   glm::vec3(1.3f, -2.0f, -2.5f),
   glm::vec3(1.5f,  2.0f, -2.5f),
   glm::vec3(1.5f,  0.2f, -1.5f),
   glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

Directional Light

Tuesday, April 5, 2022 3:14 PM

default.frag

```
...
struct Light {
   vec3 direction; // specifies the direction of the light

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};
...
void main() {
   ...
   vec3 lightDir = normalize(-light.direction);
   ...
}
```

NOTE: The lighting calculations that we used so far expect the light direction to be a direction from the fragment towards the light source, but people generally specify a directional light as a global direction point from the light source (as seen below by negating `lightPos`), so we negate the `light.direction` above.

Render Loop

```
// Specifies the direction of the light based on the light's position
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.direction"), 1, glm::value_ptr( -lightPos));
...
// 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));

   glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

Point Light

Tuesday, April 5, 2022 4:02 PM

default.frag

```
...
struct Light {
   vec3 position;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;

   float constant;
   float linear;
   float quadratic;
};
...
void main() {
   // Calculating Attenuation
   float distance = length(light.position - FragPos);
   float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic * (distance * distance));
   ...
   vec3 lightDir = normalize(light.position - FragPos);
   ...
   FragColor = vec4((ambient + diffuse + specular) * attenuation, 1.0);
}
```

Render Loop

```
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.position"), 1, glm::value_ptr(lightPos));

// Light with distance of 50
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.constant"), 1.0f);
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.linear"), 0.09f);
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.quadratic"), 0.032f);
```

Flashlight

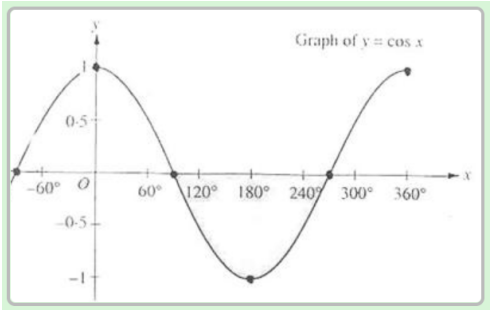
Tuesday, April 5, 2022 4:32 PM

default.frag

```
...
struct Light {
   vec3 position;
   vec3 direction;
   float cutoff;
   ...
};
...
void main() {
   // Ambient Lighting
   vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));

   vec3 lightDir = normalize(light.position - FragPos);
   float theta = dot(lightDir, normalize(-light.direction));


   if (theta > light.cutoff) { // The fragment is within the cutoff radius of the flashlight
      // do lighting calculations
      ...
      FragColor = vec4(ambient + diffuse + specular, 1.0);
   }
   else { // The fragment is outside the cutoff radius of the flashlight
      FragColor = vec4(ambient, 1.0);
   }
}
```



Notice how the check for if a fragment is within the cutoff angle is `theta > light.cutoff` and not `theta < light.cutoff`. This is because these are cosine values, so a value of 1.0 represents an angle of 0 degrees (the light is aimed directly at the fragment) and 0.0 represents an angle of 90 degrees.

Render Loop

```
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.position"), 1, glm::value_ptr(camera.position));
glUniform3fv(glGetUniformLocation(shaderProgram.id, "light.direction"), 1, glm::value_ptr(camera.front));
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.cutoff"), glm::cos(glm::radians(12.5f)));
```

Notice how the `light.cutoff` value is passed as a consine value rather than an angle. This is because the dot product of `lightDir` and `-light.direction` returns a cosine value and not an angle. If we wanted to do the comparison based on angles, we'd have to take the inverse cosine of theta, which is an expensive operation.

Smoothed Spotlight

Tuesday, April 5, 2022 5:30 PM

default.frag

```
...
struct Light {
   vec3 position;
   vec3 direction;
   float innerCutoff;
   float outerCutoff;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;

   float constant;
   float linear;
   float quadratic;
};
uniform Light light;
...
void main() {
   vec3 lightDir = normalize(light.position - FragPos);

   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

   vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords)); // ambient lighting should be unaffected by spotlight intensity
   ...
   vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords)) * intensity;
   ...
   vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords)) * intensity;

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

We no longer need the if-else statement for whether the fragment is inside the inner cone because:

- If the fragment is inside the inner cone, its light intensity value is greater than 1.0 (hence the clamp)
- If the fragment is outside the inner cone but inside the outer cone, its light intensity value is between 0.0 and 1.0
- If the fragment is outside the outer cone, its light intensity value is less than 0.0 (hence the clamp)

Render Loop

```
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.innerCutoff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(shaderProgram.id, "light.outerCutoff"), glm::cos(glm::radians(17.5f)));
```