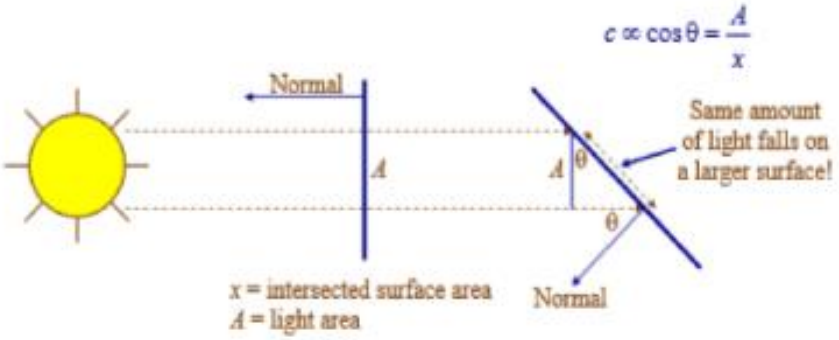
Objectives

1. Learn how to code the Lambertian diffuse light.
2. Learn how to code the specular light and combine everything into the Phong lighting model.

Diffuse light

Diffuse light or diffuse reflection refers to light that is reflected on non-metallic surface. There are many diffuse models but one model that is popular is the Lambertian reflectance model. The Lambertian obeys Lambert's cosine law.



First, in fragment shader, we can separate the ambient light into a function for future convenience.

Shader.frag

```
#version 330

in vec4 vCol;
in vec2 TexCoord;
out vec4 colour;

uniform vec3 lightColour;

uniform sampler2D texture2D;

vec3 ambientLight()
{
    float ambientStrength = 0.2f;
    vec3 ambient = lightColour * ambientStrength;

    return ambient;
}

void main()
{
    colour = texture(texture2D, TexCoord) * vec4(ambientLight(), 1.0f);
}
```

The Lambertian reflection model requires normal vectors of the model. Therefore, we need to acquire them in vertex shader and pass the values to fragment shader.

Shader.vert

```
#version 330

layout (location = 0) in vec3 pos;
layout (location = 1) in vec2 aTexCoord;
layout (location = 2) in vec3 aNormal;

out vec4 vCol;
out vec2 TexCoord;
out vec3 Normal;

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

void main()
{
    gl_Position = projection * view * model * vec4(pos, 1.0);
    vCol = vec4(clamp(pos, 0.0f, 1.0f), 1.0f);
    TexCoord = aTexCoord;
    Normal = aNormal;
}
```

Shader.frag

```
#version 330

in vec4 vCol;
in vec2 TexCoord;
in vec3 Normal;
out vec4 colour;

uniform vec3 lightColour;

uniform sampler2D texture2D;

vec3 ambientLight()
{
    float ambientStrength = 0.2f;
    vec3 ambient = lightColour * ambientStrength;

    return ambient;
}

void main()
{
    colour = texture(texture2D, TexCoord) * vec4(ambientLight(), 1.0f);
}
```

Next, we need the light direction which can be calculated with light position and fragment position. Therefore, we will pass the fragment position from the vertex shader to the fragment shader first.

Shader.vert

```
#version 330

...
out vec3 FragPos;
...
void main()
{
    gl_Position = projection * view * model * vec4(pos, 1.0);
    FragPos = vec3(model * vec4(pos, 1.0));
    vCol = vec4(clamp(pos, 0.0f, 1.0f), 1.0f);
    TexCoord = aTexCoord;
    Normal = aNormal;
}
```

The fragment position will be received in the fragment shader. After that, the light position will also be received using uniform variable, which can be passed from the main program.

Shader.frag

```
#version 330

in vec4 vCol;
in vec2 TexCoord;
in vec3 FragPos;
in vec3 Normal;
out vec4 colour;

uniform vec3 lightColour;
uniform vec3 lightPos;

uniform sampler2D texture2D;

...
void main()
{
    colour = texture(texture2D, TexCoord) * vec4(ambientLight(), 1.0f);
}
```

main.cpp

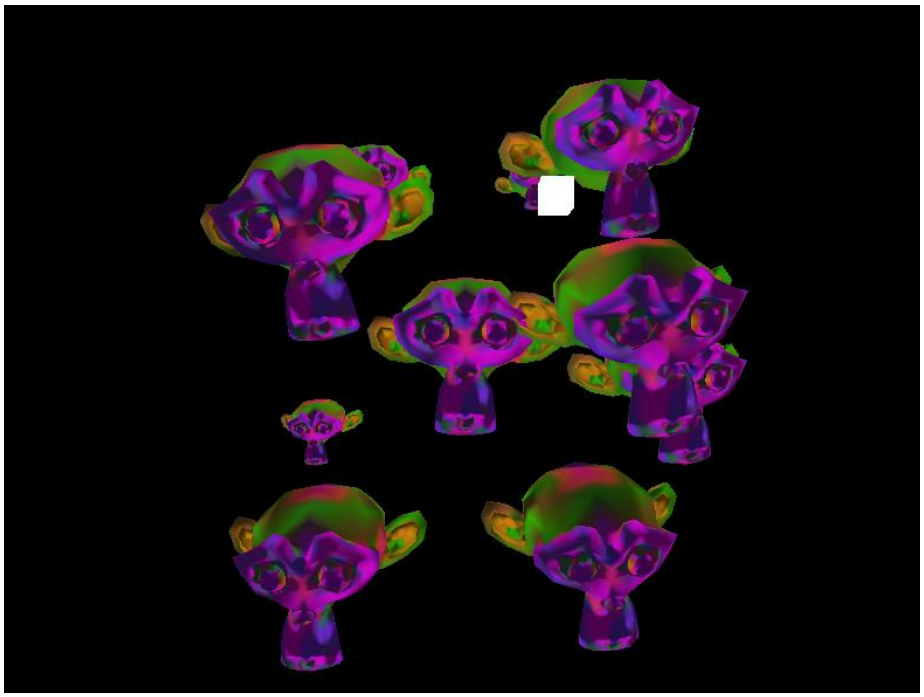
```
...
glm::vec3 lightPos = glm::vec3(5.0f, 5.0f, 0.0f);
...
int main()
{
    ...
    //Object

    ...


    //light
    glUniform3fv(shaderList[0]->GetUniformLocation("lightPos"), 1, (GLfloat *)&lightPos);

    ...
    meshList[i]->RenderMesh();
    ...
}
```

Create another function to calculate the diffuse light as follows.

Shader.frag

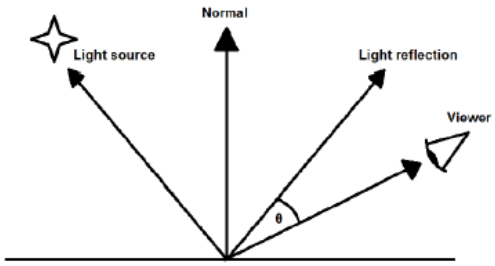
```
#version 330
...

vec3 diffuseLight()
{
    float diffuseStrength = 0.5f;

    vec3 lightDir = normalize(lightPos - FragPos);
    vec3 norm = normalize(Normal);

    float diff = max(dot(norm, lightDir), 0.0f);
    vec3 diffuse = lightColour * diff * diffuseStrength;
    return diffuse;
}
...

void main()
{
    colour = texture(texture2D, TexCoord) * vec4(ambientLight() + diffuseLight(), 1.0f);
}
```

This creates a scene as below.



Specular light

Specular reflections refer to light that is reflected on smooth reflective surface such as mirror, metal, etc. One of the popular specular light models is Phong reflectance model. This model requires the light reflection direction and viewing direction, plus the shininess of the surface.



The viewing direction can be calculated by the current position of the camera and the fragment position, the current position of the camera (we use this already in the previous lab) can be obtained via a uniform variable as follows.

Shader.frag

```
#version 330

...

uniform vec3 viewPos;
...
```

main.cpp

```
...
int main()
{
    ...
    //Object

    ...


    //light
    glUniform3fv(shaderList[0]->GetUniformLocation("viewPos"), 1, (GLfloat *)&cameraPos);

    ...
    meshList[i]->RenderMesh();
    ...
}
```

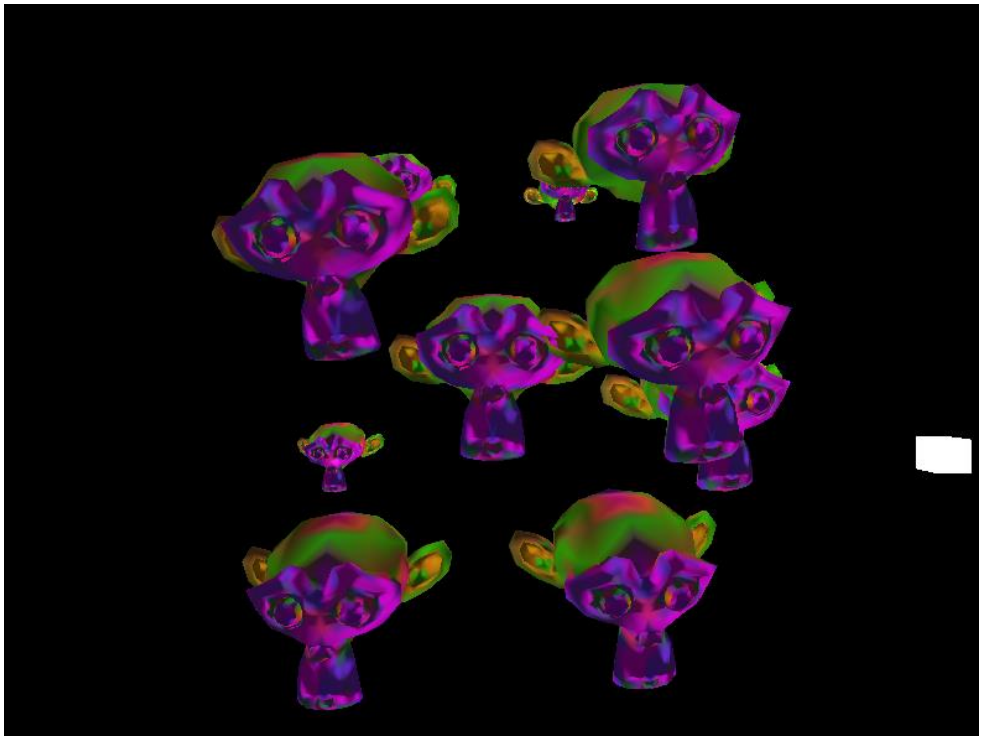
The reflection direction can be calculated by the light direction and the normal vector. Therefore, the specular light can be calculated in a function as follows.

Shader.frag
```
#version 330
...

vec3 specularLight()
{
    float specularStrength = 0.8f;
    float shininess = 64.0f;

    vec3 lightDir = normalize(lightPos - FragPos);
    vec3 norm = normalize(Normal);
    vec3 reflectDir = reflect(-lightDir, norm);
    vec3 viewDir = normalize(viewPos - FragPos);

    float spec = pow(max(dot(viewDir, reflectDir), 0.0f), shininess);

    vec3 specular = lightColour * spec * specularStrength;

    return specular;
}
...
```

When we combine ambient light, diffuse light, and specular light, we will have a lighting model called Phong lighting model.

Shader.frag
```
#version 330
...

void main()
{
    //Phong Shading
    colour = texture(texture2D, TexCoord) * vec4(ambientLight() + diffuseLight() + specular-
Light(), 1.0f);
}
```

This creates a scene as follows.



Tasks (Submit through Google Classroom): -

1. Complete the diffuse and specular lights in this lab.
2. Submit through Google Classroom before Monday 23.59.