Lighting & Shading

COMP3015 Lab 3

Directional Light

Ambient, diffuse & specular lighting in the vertex shader are required for directional light. However, variables for light information that are sent to the shader should be directions as opposed to positions. The result of this is that you don't need to calculate the light direction within the shader. Look at the lecture slides in order to understand how to make distinctions between light directions & positions.

Multiple Lights

Overview

Ambient, diffuse & specular lighting, as in the previous lab, are required for directional light. The only difference is that the light calculations need to be applied to 3 lights as opposed to one. The fragment shader can be kept similar to the previous shader.

Vertex Shader

The vertex shader requires two vec3 input variables named VertexPosition & VertexNormal, indexed at locations 0 & 1 respectively. It also requires one output vec3 variable named Colour.

Two uniform structs are required. The first, as displayed below, contains the various lights:

```
uniform struct LightInfo {

        vec4 Position; // Light position in eye coords.

        vec3 La; // Ambient light intensity

        vec3 L; // Diffuse and specular light intensity

} lights[3];
```

The second required struct is for the material:

```
uniform struct MaterialInfo {
        vec3 Ka; // Ambient reflectivity
        vec3 Kd; // Diffuse reflectivity
        vec3 Ks; // Specular reflectivity
        float Shininess; // Specular shininess factor
} Material;
```

We need to create a function for calculating the ambient, diffuse & specular light. We will call the function phongModel. The function's return type should be vec3 & include the following parameters: int light, vec3 position, vec3 normal.

```
vec3 phongModel( int light, vec3 position, vec3 n )

{

        //calculate ambient here, to access each light La value use this: lights[light].La

        vec3 ambient = …



        //calculate diffuse here

        vec3 s = … //find out s vector

        float sDotN = … //calculate dot product between s & n

        vec3 diffuse = Material.Kd * sDotN;



        //calculate specular here

        vec3 spec = vec3(0.0);



        if( sDotN > 0.0 )

        {

                vec3 v = normalize(-position.xyz);

                vec3 r = reflect( -s, n );

                spec = Material.Ks * pow( max( dot(r,v), 0.0 ), Material.Shininess );

        }



        return ambient + lights[light].L * (diffuse + spec);

}
```

In the main() function, convert the vertex normal to eye coordinates. The result should be a vec3 (use NormalMatrix). Ensure the result is normalised.

```
vec3 n = normalize( NormalMatrix * VertexNormal);
```

Convert the vertex position to eye coordinates. The result should be a vec4 (use ModelViewMatrix). After this, evaluate the lighting equation for each light as shown below:

```
Colour = vec3(0.0);

for( int i = 0; i < 3; i++ )

        Colour += phongModel( i, camCoords, n );
```

Lastly, set gl_Position as shown below:

```
gl_Position = MVP * vec4(VertexPosition,1.0);
```

Setup

1. We need to import some files into the project. Unpack the folder on the DLE & add the following files to your project's helper folder: aabb.h, utils.h, objmesh.h, objmesh.cpp, plane.h & plane.cpp.

2. Once the required files are in the helper folder, ensure they are added to the project solution, as done in previous sessions.

3. Lastly, add the pig_triangulated.obj file to your project's media folder. Do not add it to your project solution. This is unnecessary as it is not a code file & in this case will cause errors.

Scenebasic Uniform

Header

Include the plane.h & objmesh.h files at the top of your header file & remove the torus.h include if still present. Add the two new variables as shown below & ensure they are private:

```
Plane plane;                        //plane surface

std::unique_ptr<ObjMesh> mesh;      //pig mesh
```

CPP

Includes

Include sstream at the top of the file:

```
#include <sstream>
```

Constructor

Within the constructor, replace the torus with the plane & load the pig_triangulated.obj file as a mesh, as displayed below:

```
SceneBasic_Uniform::SceneBasic_Uniform() : plane(10.0f, 10.0f, 100, 100)
{
        mesh = ObjMesh::load("../Project_Template/media/pig_triangulated.obj", true);
}
```

initScene() Function

Set the camera & the 3 lights' positions:

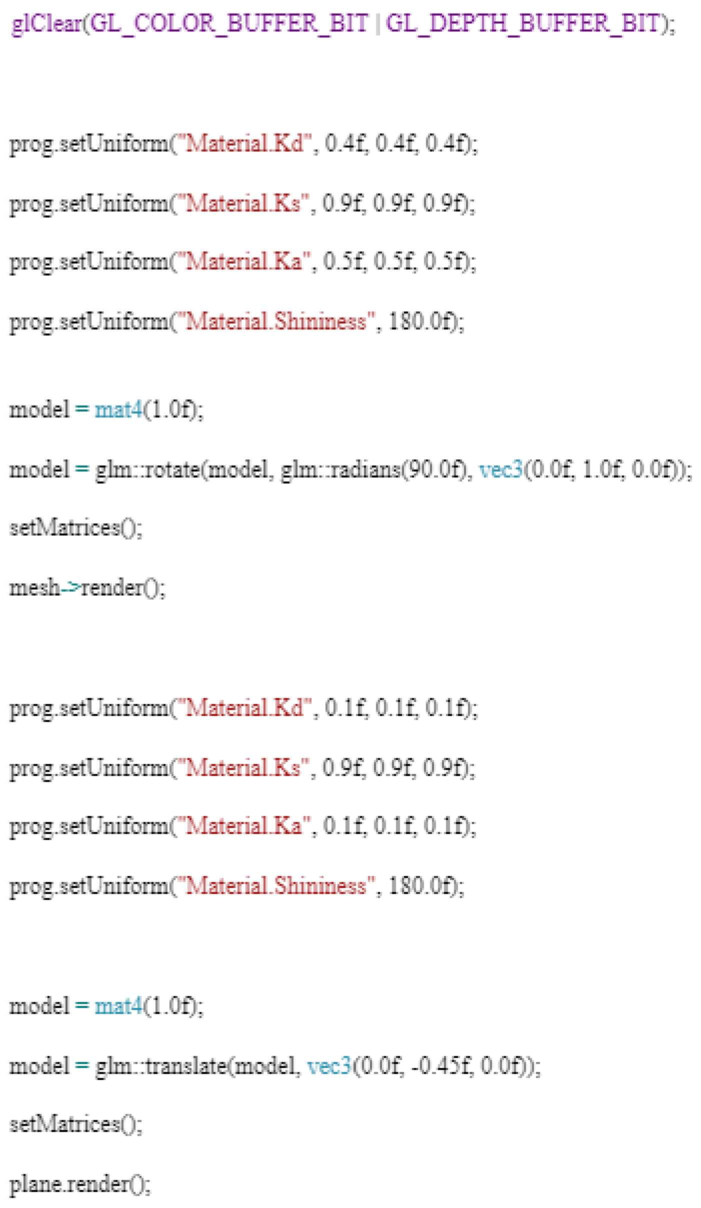
```
compile();
glEnable(GL_DEPTH_TEST);
view = glm::lookAt(vec3(0.5f, 0.75f, 0.75f), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
projection = mat4(1.0f);
float x, z;
for (int i = 0; i < 3; i++)
{
        std::stringstream name;
        name << "lights[" << i << "].Position";
        x = 2.0f * cosf((glm::two_pi<float>() / 3) * i);
        z = 2.0f * sinf((glm::two_pi<float>() / 3) * i);
        prog.setUniform(name.str().c_str(), view * glm::vec4(x, 1.2f, z + 1.0f, 1.0f));
}
```

Set the uniforms for the lights & put together the diffuse & the specular lights under a light called L as shown below. Also, ensure you set the uniforms for the ambient light.

```
prog.setUniform("lights[0].L", vec3(0.0f, 0.0f, 0.8f));
prog.setUniform("lights[1].L", vec3(0.0f, 0.8f, 0.0f));
prog.setUniform("lights[2].L", vec3(0.8f, 0.0f, 0.0f));
```

Render() Function

Set the Material struct uniforms for our plane & pig mesh:



```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


prog.setUniform("Material.Kd", 0.4f, 0.4f, 0.4f);

prog.setUniform("Material.Ks", 0.9f, 0.9f, 0.9f);

prog.setUniform("Material.Ka", 0.5f, 0.5f, 0.5f);

prog.setUniform("Material.Shininess", 180.0f);

model = mat4(1.0f);

model = glm::rotate(model, glm::radians(90.0f), vec3(0.0f, 1.0f, 0.0f));

setMatrices();

mesh->render();


prog.setUniform("Material.Kd", 0.1f, 0.1f, 0.1f);

prog.setUniform("Material.Ks", 0.9f, 0.9f, 0.9f);

prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);

prog.setUniform("Material.Shininess", 180.0f);


model = mat4(1.0f);

model = glm::translate(model, vec3(0.0f, -0.45f, 0.0f));

setMatrices();

plane.render();
```

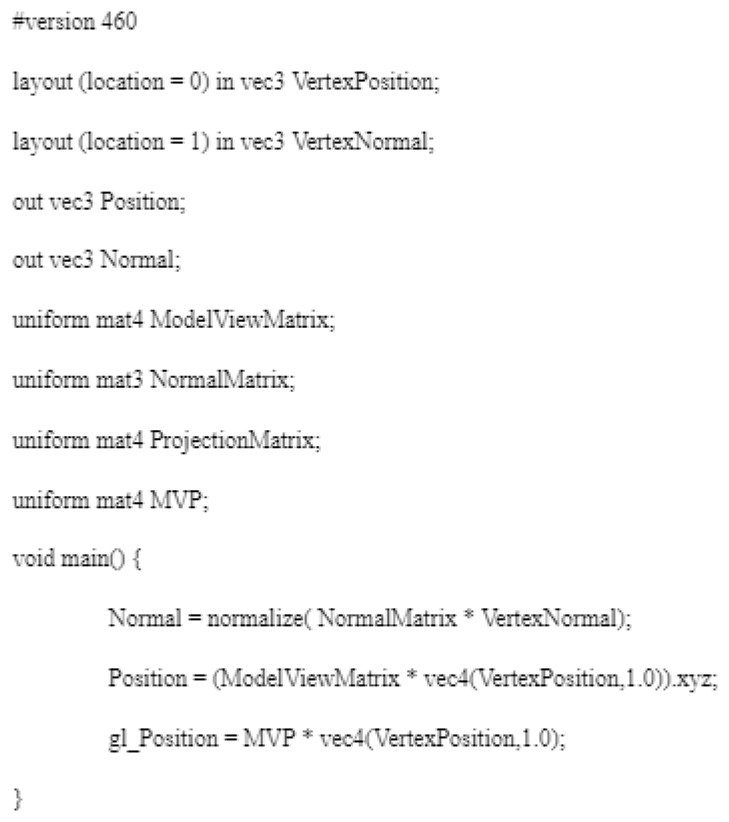
Per-Fragment Shading (Phong Model)

Overview

Use the multiple lights example, however use one light instead of 3. Also, ensure all calculations are done in the fragment shader as opposed to the vertex shader.

Vertex Shader

Use the following code as your vertex shader for your Phong model implementation. This has been covered prior, however if anything is unclear do ask for help.



```
#version 460

layout (location = 0) in vec3 VertexPosition;

layout (location = 1) in vec3 VertexNormal;

out vec3 Position;

out vec3 Normal;

uniform mat4 ModelViewMatrix;

uniform mat3 NormalMatrix;

uniform mat4 ProjectionMatrix;

uniform mat4 MVP;

void main() {

        Normal = normalize( NormalMatrix * VertexNormal);

        Position = (ModelViewMatrix * vec4(VertexPosition,1.0)).xyz;

        gl_Position = MVP * vec4(VertexPosition,1.0);

}
```

Fragment Shader

Create a function of type vec3 called phongModel with the following parameters: vec3 position & vec3 normal. When called, it will be used to return a vec3 composed of Ambient + Diffuse + Specular. Pass the vec3 vector into FragColour. See the lecture slides for elaboration / extra details.

Blinn-Phong Model

This model is similar to the Phong model, however the reflection vector isn't present & you use the half vector. See the lecture slides for details regarding its implementation.

Spotlight Model

Shaders

Use the same vertex shader as used for the Blinn-Phong model. Regarding the fragment shader however, see the lecture slides for its implementation.

Scenebasic_Uniform CPP

There are a few settings that make implementation easier. Within the constructor, we need to initialise our 3 objects: The plane, teapot & torus. Use the parameters as specified in the code below:

```
plane(50.0f, 50.0f, 1, 1),

teapot(14, glm::mat4(1.0f)),

torus(1.75f * 0.75f, 0.75f * 0.75f, 50, 50)
```

The code below displays how the initScene() function should look:

```
void SceneBasic_Uniform::initScene()

{

        compile();

        glEnable(GL_DEPTH_TEST);

        view = glm::lookAt(vec3(5.0f, 5.0f, 7.5f), vec3(0.0f, 0.75f, 0.0f), vec3(0.0f, 1.0f, 0.0f));

        projection = mat4(1.0f);

        prog.setUniform("Spot.L", vec3(0.9f));

        prog.setUniform("Spot.La", vec3(0.5f));

        prog.setUniform("Spot.Exponent", 50.0f);

        prog.setUniform("Spot.Cutoff", glm::radians(15.0f));

}
```

The code below displays the render() function. If the nature of the code is confusing, do ask:

```
void SceneBasic_Uniform::render()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        vec4 lightPos = vec4(0.0f, 10.0f, 0.0f, 1.0f);
        prog.setUniform("Spot.Position", vec3(view * lightPos));
        mat3 normalMatrix = mat3(vec3(view[0]), vec3(view[1]), vec3(view[2]));
        prog.setUniform("Spot.Direction", normalMatrix * vec3(-lightPos));

        prog.setUniform("Material.Kd", 0.2f, 0.55f, 0.9f);
        prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
        prog.setUniform("Material.Ka", 0.2f * 0.3f, 0.55f * 0.3f, 0.9f * 0.3f);
        prog.setUniform("Material.Shininess", 100.0f);

        model = mat4(1.0f);
        model = glm::translate(model, vec3(0.0f, 0.0f, -2.0f));
        model = glm::rotate(model, glm::radians(45.0f), vec3(0.0f, 1.0f, 0.0f));
        model = glm::rotate(model, glm::radians(-90.0f), vec3(1.0f, 0.0f, 0.0f));
        setMatrices();
        teapot.render();

        prog.setUniform("Material.Kd", 0.2f, 0.55f, 0.9f);
        prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
        prog.setUniform("Material.Ka", 0.2f * 0.3f, 0.55f * 0.3f, 0.9f * 0.3f);
        prog.setUniform("Material.Shininess", 100.0f);

        model = mat4(1.0f);
        model = glm::translate(model, vec3(-1.0f, 0.75f, 3.0f));
        model = glm::rotate(model, glm::radians(-90.0f), vec3(1.0f, 0.0f, 0.0f));
        setMatrices();
        torus.render();

        prog.setUniform("Material.Kd", 0.7f, 0.7f, 0.7f);
        prog.setUniform("Material.Ks", 0.9f, 0.9f, 0.9f);
        prog.setUniform("Material.Ka", 0.2f, 0.2f, 0.2f);
        prog.setUniform("Material.Shininess", 180.0f);

        model = mat4(1.0f);
        setMatrices();
        plane.render();
}
```

Toon Shading

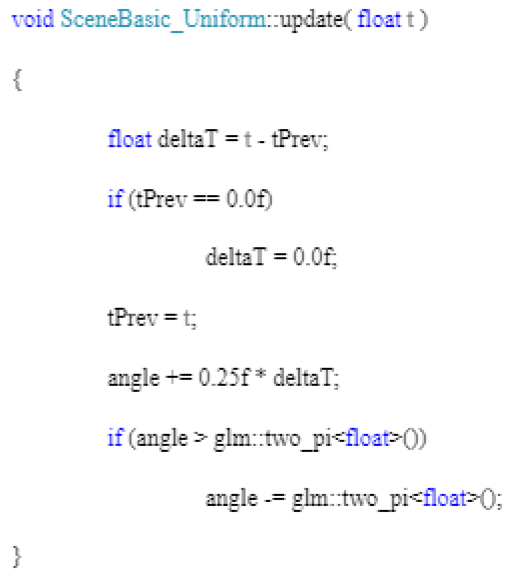
Overview

Use the same vertex shader as used for the Blinn-Phong model. Regarding the fragment shader however, see the lecture slides for its implementation.

Scenebasic Uniform CPP

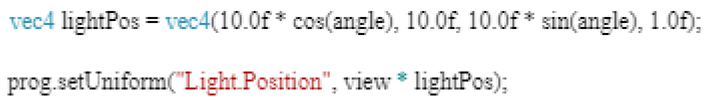
You can use the same format as used in the spotlight implementation. If you desire, the specular implementation can be skipped in order to result in a more natural look. If you want to animate the light, use two variables named “tPrev” & “angle,” both of type float & set both of their values to 0 in initScene().

Update the angle value in the update() function:



```
void SceneBasic_Uniform::update( float t )
{
        float deltaT = t - tPrev;
        if (tPrev == 0.0f)
                deltaT = 0.0f;
        tPrev = t;
        angle += 0.25f * deltaT;
        if (angle > glm::two_pi<float>())
                angle -= glm::two_pi<float>();
}
```

Update the light position value with your angle variable in the render() function:



```
vec4 lightPos = vec4(10.0f * cos(angle), 10.0f, 10.0f * sin(angle), 1.0f);
prog.setUniform("Light.Position", view * lightPos);
```

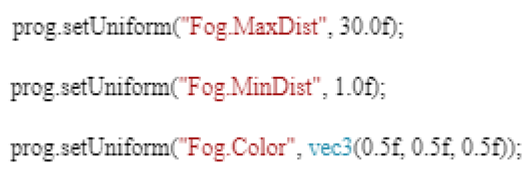
Fog Simulation

Overview

Use the same vertex shader as used for the Blinn-Phong model. Regarding the fragment shader however, see the lecture slides for its implementation.

Scenebasic Uniform CPP

You can use the same format as used in the spotlight implementation. In the initScene() function, set all the non-changing uniforms, including at least the following:



```
prog.setUniform("Fog.MaxDist", 30.0f);

prog.setUniform("Fog.MinDist", 1.0f);

prog.setUniform("Fog.Color", vec3(0.5f, 0.5f, 0.5f));
```

Note that the fog is being applied to objects as opposed to the background. Because of this, the fog & background need to use the same colour in order to look realistic.

Experiment with the distance (z value) of the objects (teapot, torus, etc) in order to manifest different fog values around your objects. The further away an object is, the more fog that is applied to it. If anything is unclear / you are stuck, do ask for help.