1. Switch-on lighting in the given program using a point light source

a) Replace the cube with any parametric surface of your choice (refer to Lab 2). Generate normals for the object. Lighting computations (e.g. in the Phong model) require normals along with vertex data. Compute per-vertex normals (remember to normalize the normals - i.e. make sure normals are unit vectors in all your computations).

Parametric equation of a sphere:
x = radius *sin(phi) *cos(theta);
y = radius *sin(phi) *sin(theta);
z = radius *cos(phi);

Calculating normal of each face on the sphere:
delta_theta = vector(radius * sin(phi) * cos(theta), radius * glm::sin(phi) * glm::sin(theta),0.0f);
delta_phi = vector(radius * cos(phi) * cos(theta), radius * glm::cos(phi) * glm::sin(theta), -radius * glm::sin(phi));
Normal = normalize(cross_product(delta_theta, delta_phi));

Now we can generate vertices, normals and indices array of sphere whing these two:

```
for (int r = 0; r < longitudinal_lines; ++r)
{
    for (int s = 0; s < latitudinal_lines; ++s)
    {
        float phi = glm::pi<float>() * r * fR;
        float theta = 2 * glm::pi<float>() * s * fS;

        float x = radius * glm::sin(phi) * glm::cos(theta);
        float y = radius * glm::sin(phi) * glm::sin(theta);
        float z = radius * glm::cos(phi);

        verts[vertexIndex++] = x;
        verts[vertexIndex++] = y;
        verts[vertexIndex++] = z;

        glm::vec3 d_theta = glm::vec3(radius * glm::sin(phi) * glm::cos(theta),
            radius * glm::sin(phi) * glm::sin(theta),
            0.0f);
        glm::vec3 d_phi = glm::vec3(radius * glm::cos(phi) * glm::cos(theta),
            radius * glm::cos(phi) * glm::sin(theta),
            -radius * glm::sin(phi));

        glm::vec3 normal = glm::normalize(glm::cross(d_theta, d_phi));

        norms[normalIndex++] = normal.x;
        norms[normalIndex++] = normal.y;
        norms[normalIndex++] = normal.z;
    }
}
```

```
int index = 0;
for (int r = 0; r < longitudinal_lines - 1; ++r)
{
    for (int s = 0; s < latitudinal_lines - 1; ++s)
    {
        // First triangle (tracing counter-clockwise)
        indices[index++] = r * latitudinal_lines + s;
        indices[index++] = r * latitudinal_lines + (s + 1);
        indices[index++] = (r + 1) * latitudinal_lines + s;

        // Second triangle (tracing counter-clockwise)
        indices[index++] = (r + 1) * latitudinal_lines + s;
        indices[index++] = r * latitudinal_lines + (s + 1);
        indices[index++] = (r + 1) * latitudinal_lines + (s + 1);
    }
}
```

b)Add a fixed point light source
In programmable OpenGL, lighting computations are done entirely in the shaders. A point light source is identified by the location of the light and its color. Add these variables to your vertex shader as uniforms and pass-on certain values from your C++ program.

Introduce these in vshader:
uniform vec3 lightPosition; // Position of the light source
uniform vec3 lightColor; // Color of the light source

Update main code with these lines:

```
// Define light source position
float lightPositionX = 10.0f; // Change these values as needed
float lightPositionY = 10.0f;
float lightPositionZ = 10.0f;

// Define light source color
float lightColorR = 1.0f;
float lightColorG = 0.0f;
float lightColorB = 0.0f;
```

glUseProgram

```
glUniform3f(glGetUniformLocation(shaderProgram, "lightPosition"),
lightPositionX, lightPositionY, lightPositionZ);
glUniform3f(glGetUniformLocation(shaderProgram, "lightColor"),
lightColorR, lightColorG, lightColorB);
```

c)Use Phong lighting to render the object with specular highlights.
Modify your vertex shader to include all three lighting components: ambient, diffuse, and specular. Use Phong exponent = 32. Notice that the back side of the object will not be pitch dark since you have added an ambient component.

Update vshader:

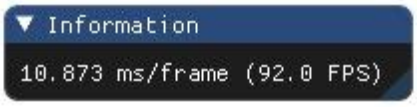
```
#version 330 core

in vec3 vVertex;
in vec3 vNormal;

uniform mat4 vModel;
uniform mat4 vView;
uniform mat4 vProjection;
uniform vec3 lpos_world;
uniform vec3 eye_normal;

out vec3 n;
out vec3 e;
out vec3 l;

void main() {
    gl_Position = vProjection * vView * vModel * vec4(vVertex, 1.0);
    n = normalize(vNormal);
    l = normalize(lpos_world - vVertex);
    e = eye_normal;
}
```

Update fshader:

```
in vec3 n;
in vec3 e;
```

```
in vec3 l;
out vec4 outColor;

vec3 Ls = vec3(1.0, 1.0, 1.0);
vec3 Ld = vec3(0.7, 0.7, 0.7);
vec3 La = vec3(1, 0.3, 0.4);

vec3 ks = vec3(1.0, 1.0, 1.0);
vec3 kd = vec3(0.5, 0.6, 0.4);
vec3 ka = vec3(0.0, 1.0, 0.0);

float spec_exp = 32;

// Calculate ambient light component (Ia)
vec3 Ia = La * ka;

// Calculate diffuse light component (Id)
vec3 L = normalize(l);
vec3 N = normalize(n);
float diffuseFactor = max(dot(N, L), 0.0);
vec3 Id = Ld * kd * diffuseFactor;

// Calculate specular light component (Is)
vec3 R = reflect(-L, N);
vec3 V = normalize(e);
vec3 H = normalize(L+V);

float specularFactor = pow(max(dot(n, H), 0.0), spec_exp);
vec3 Is = Ls * ks * specularFactor;

//ambient + diffuse + specular
vec3 fColor = Ia + Id + Is;

void main(void) {
    outColor = vec4(fColor, 1.0);
}
```

Also introduce lpos_world and eye_normal in main file.



2. Create a spot light source

(a) A spotlight is a light source that emits light in a cone of directions. As a result only object within the cone of illumination are lit. In OpenGL you can represent a spot light with a position, a direction, and a cutoff angle (half cone angle) that defines the cone of illumination. In your code, implement a spot light such that only part of the object falls inside the cone of illumination. Choose the cutoff angle appropriately (anything between 15°-30° is good). Perform all lighting calculations in the fragment Shader.

Spot light will also have lightColor and lightPosition like point light but it also have a direction so we introduce a lightDir variable in fragment shader class. We also make a float variable name cutOff which represent the cutOff angle.

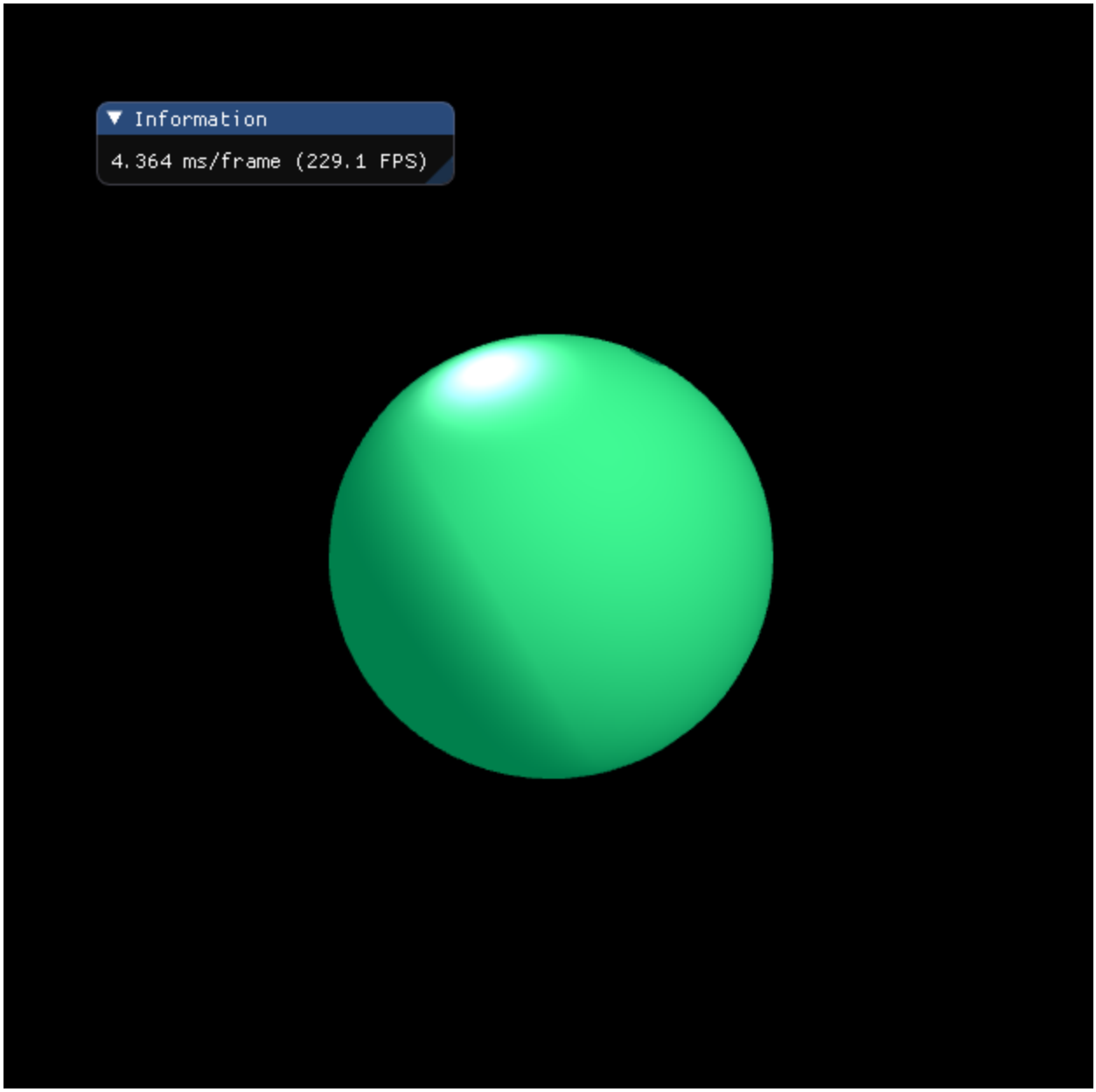
Update fshader:
vec3 position;
vec3 direction;
float cutOff;

```
 // Spotlight
    float theta = dot(lightDir, normalize(-light.direction));
    float epsilon = (light.cutOff);
    float intensity = clamp((theta) / epsilon, 0.0, 1.0);
    diffuse  *= intensity;
    specular *= intensity;

// Light attributes
glm::vec3 lightPos( 1.2f, 1.0f, 2.0f );
```

(b)You will observe that the above spot light has hard edges on the boundary of the illumination cone (where the light cone intersects with the object). Implement smooth soft edges for your spot light by creating an inner cone (with cutoff angle) and an outer cone (with cutoff angle). The position and directions of these two cones are same. Keep the same as cutoff angle in part (a), and to be slightly more (perhaps 5°-10°) than . The intensity of a fragment falling in between these two cones should drop down from 1 to 0 in a smooth fashion. For an angle at a fragment, linearly interpolate the intensity by multiplying with the fraction .

Update spotlight

```
    // Spotlight (soft edges)
    float theta = dot(lightDir, normalize(-light.direction));
    float epsilon = (light.cutOff - light.outerCutOff);
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);
    diffuse  *= intensity;
    specular *= intensity;
```


Information
4.364 ms/frame (229.1 FPS)