# CIS 515: COMPUTER GRAPHICS
# LAB – 6
UNIVERSITY OF MICHIGAN – DEARBORN

FALL 2024

By,

VISHVENDRA REDDY BHOOMIDI

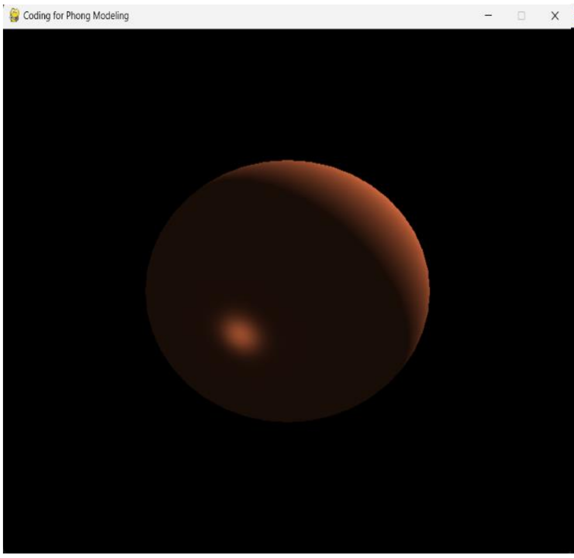
bhoomidi@umich.edu

## TASK 1: Coding for Phong Modeling

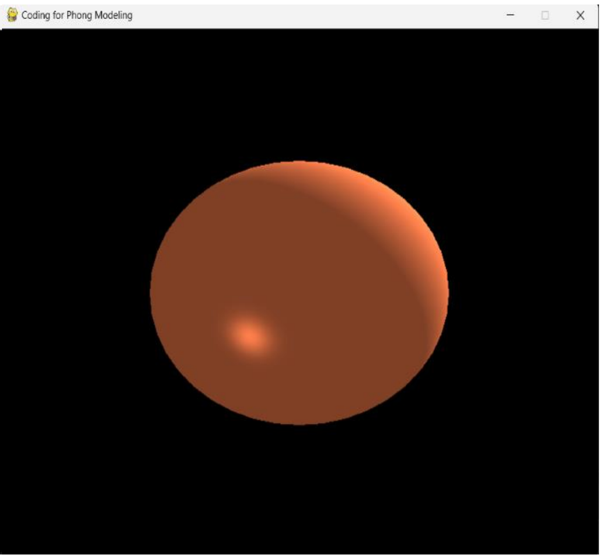
```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader
import numpy as np

# Shaders
vertex_shader = """
#version 330
in vec3 position;
in vec3 normal;
out vec3 fragPosition;
out vec3 fragNormal;

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

void main()
{
   fragPosition = vec3(model * vec4(position, 1.0));
   fragNormal = mat3(transpose(inverse(model))) * normal;
   gl_Position = projection * view * model * vec4(position, 1.0);
}
"""

fragment_shader = """
#version 330
```

```
in vec3 fragPosition;
in vec3 fragNormal;
out vec4 color;

uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform vec3 objectColor;
uniform float shininess;
uniform float ambientStrength;

void main()
{
    // Ambient lighting
    vec3 ambient = ambientStrength * lightColor;

    // Diffuse lighting
    vec3 norm = normalize(fragNormal);
    vec3 lightDir = normalize(lightPos - fragPosition);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // Specular lighting
    float specularStrength = 0.5;
    vec3 viewDir = normalize(viewPos - fragPosition);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;
    vec3 result = (ambient + diffuse + specular) * objectColor;
    color = vec4(result, 1.0);
}
"""

def create_shader_program():
    return compileProgram(
        compileShader(vertex_shader, GL_VERTEX_SHADER),
        compileShader(fragment_shader, GL_FRAGMENT_SHADER)
    )

def create_sphere(radius, slices, stacks):
    vertices = []
    normals = []
    for i in range(stacks):
        lat0 = np.pi * (-0.5 + float(i) / stacks)
        z0 = radius * np.sin(lat0)
        zr0 = radius * np.cos(lat0)

        lat1 = np.pi * (-0.5 + float(i + 1) / stacks)
        z1 = radius * np.sin(lat1)
        zr1 = radius * np.cos(lat1)

        for j in range(slices):
            lng = 2 * np.pi * float(j) / slices
            x = np.cos(lng)
            y = np.sin(lng)

            vertices.extend([x * zr0, y * zr0, z0])
            vertices.extend([x * zr1, y * zr1, z1])

            normals.extend([x * zr0, y * zr0, z0])
            normals.extend([x * zr1, y * zr1, z1])

    return np.array(vertices, dtype=np.float32), np.array(normals, dtype=np.float32)

def create_vertex_objects(vertices, normals):
    vao = glGenVertexArrays(1)
    vbo = glGenBuffers(2)

    glBindVertexArray(vao)

    glBindBuffer(GL_ARRAY_BUFFER, vbo[0])
    glBufferData(GL_ARRAY_BUFFER, vertices.nbytes, vertices, GL_STATIC_DRAW)
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, None)
    glEnableVertexAttribArray(0)

    glBindBuffer(GL_ARRAY_BUFFER, vbo[1])
    glBufferData(GL_ARRAY_BUFFER, normals.nbytes, normals, GL_STATIC_DRAW)
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, None)
    glEnableVertexAttribArray(1)

    glBindBuffer(GL_ARRAY_BUFFER, 0)
    glBindVertexArray(0)

    return vao, vbo

def main():
    pygame.init()
    screen = pygame.display.set_mode((800, 600), pygame.DOUBLEBUF | pygame.OPENGL)
    pygame.display.set_caption("Coding for Phong Modeling")

    glEnable(GL_DEPTH_TEST)

    shader = create_shader_program()
    glUseProgram(shader)

    vertices, normals = create_sphere(0.5, 40, 40)
    vao, vbo = create_vertex_objects(vertices, normals)

    model = np.identity(4, dtype=np.float32)
    view = np.identity(4, dtype=np.float32)
    projection = np.identity(4, dtype=np.float32)
    light_pos = np.array([5.0, 5.0, 5.0], dtype=np.float32)
    view_pos = np.array([0.0, 0.0, 5.0], dtype=np.float32)

    glUniformMatrix4fv(glGetUniformLocation(shader, "model"), 1, GL_FALSE, model)
    glUniformMatrix4fv(glGetUniformLocation(shader, "view"), 1, GL_FALSE, view)
    glUniformMatrix4fv(glGetUniformLocation(shader, "projection"), 1, GL_FALSE, projection)
    glUniform3fv(glGetUniformLocation(shader, "lightPos"), 1, light_pos)
    glUniform3fv(glGetUniformLocation(shader, "viewPos"), 1, view_pos)
    glUniform3f(glGetUniformLocation(shader, "lightColor"), 1.0, 1.0, 1.0)
    glUniform3f(glGetUniformLocation(shader, "objectColor"), 1.0, 0.5, 0.3)

    shininess = 32
    ambient_strength = 0.1
    glUniform1f(glGetUniformLocation(shader, "shininess"), shininess)
    glUniform1f(glGetUniformLocation(shader, "ambientStrength"), ambient_strength)

    clock = pygame.time.Clock()

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        glBindVertexArray(vao)
        glDrawArrays(GL_TRIANGLE_STRIP, 0, len(vertices) // 3)
        pygame.display.flip()

        clock.tick(60)

    glDeleteBuffers(2, vbo)
    glDeleteProgram(shader)
    pygame.quit()

if __name__ == "__main__":
    main()
```

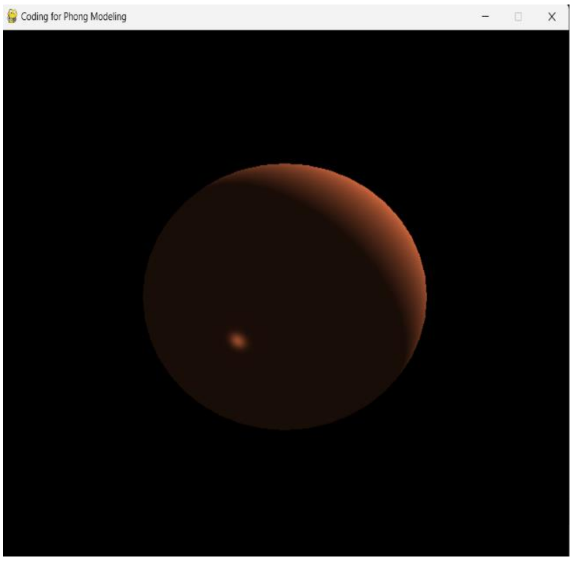
Shininess: 32, Ambient Strength: 0.1



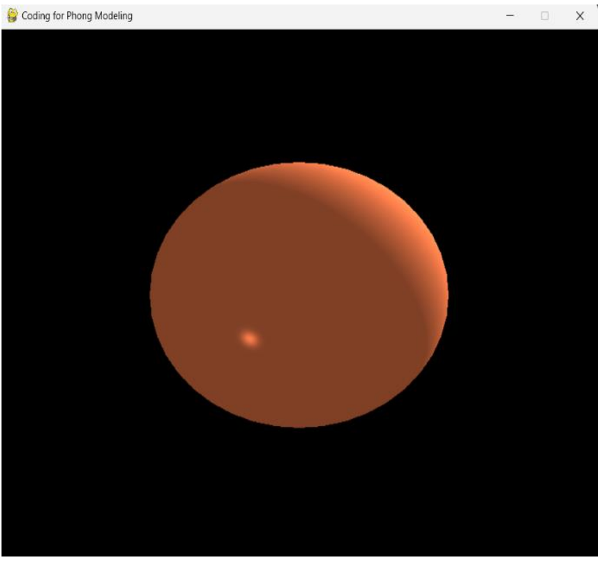
Shininess: 32, Ambient Strength: 0.5



Shininess: 200, Ambient Strength: 0.1



Shininess: 200, Ambient Strength: 0.5



## TASK 2: Calculation of Phong Model

We have to calculate the Ambient Component

Ambient color = Surface color * Ambient light intensity

= (0.7, 0.7, 0.7) * 0.2 = **(0.14, 0.14, 0.14)**

The Diffuse Component is given as,

Light Vector (L): Normalize (Light Position - Surface Position)

= Normalize ((5.0, 5.0, 5.0) - (1.5, 1.5, 0.0))

= Normalize ((3.5, 3.5, 5.0)) = **(0.498, 0.498, 0.711)**

To calculate the Dot Product (L · N):

= (0.498, 0.498, 0.711) · (0.0, 0.0, 1.0) = **0.711**

Now for the Diffuse color,

Diffuse color = Surface color * Light color * (L · N)

= (0.7, 0.7, 0.7) * (1.0, 1.0, 1.0) * 0.711 = **(0.4977, 0.4977, 0.4977)**

To get the Specular Component,

Reflection Vector (R): 2 * (L · N) * N - L

= 2 * 0.711 * (0.0, 0.0, 1.0) - (0.498, 0.498, 0.711)

= (0.0, 0.0, 1.422) - (0.498, 0.498, 0.711) = **(-0.498, -0.498, 0.711)**

We should calculate View Vector (V): Normalize(Viewer Position - Surface Position)

= Normalize((0.0, 0.0, 0.0) - (1.5, 1.5, 0.0))

= Normalize((-1.5, -1.5, 0.0)) = **(-0.707, -0.707, 0.0)**

To Calculate the Dot Product (R · V): (-0.498, -0.498, 0.711) · (-0.707, -0.707, 0.0) = **0.704**

To calculate the Specular color,

Specular color = Light color * Specular intensity * $(R \cdot V)^{\text{Shininess}}$

= (1.0, 1.0, 1.0) * 0.3 * $(0.704)^{32}$ ≈ **(0.0, 0.0, 0.0)**

The Final Phong Model Output can be given as,

Phong output = Ambient color + Diffuse color + Specular color

= (0.14, 0.14, 0.14) + (0.4977, 0.4977, 0.4977) + (0.0, 0.0, 0.0)

**= (0.6377, 0.6377, 0.6377)**