HW2: Lighting and Shading

Introduction to Computer Graphics

Yu-Ting Wu

HW Description

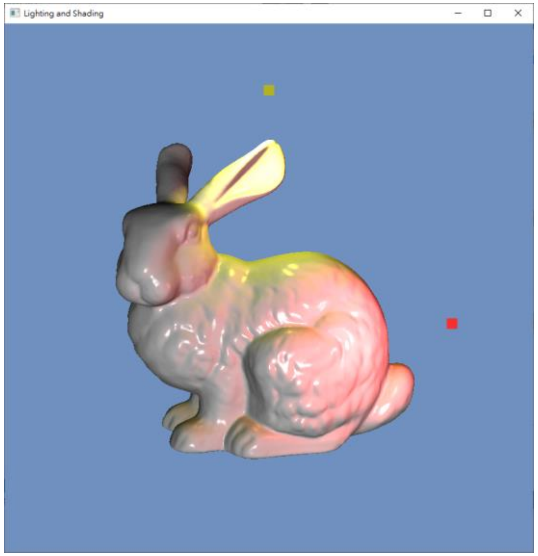
- **Web Link:**
  - https://kevincosner.github.io/courses/ICG2022/hw2.html

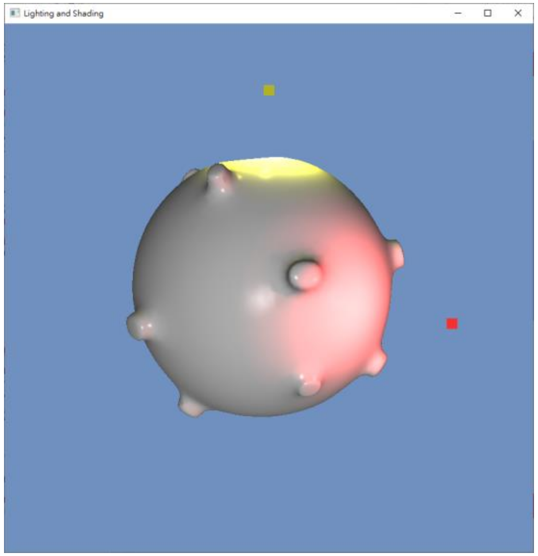
- **Major Task**
  - Implement **GPU vertex and fragment shaders** as well as a **CPU program** to perform ***Phong* shading** using the ***Phong* lighting model**
    - The material properties of the model should be loaded from a ***Material Template File (*.mtl)***
  - Implement the formulas of lighting computations of a **point light**, a **directional light**, and a **spot light**

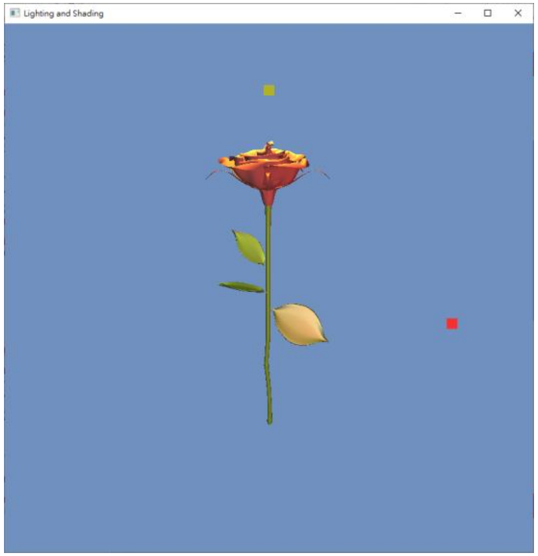
Grading Policy

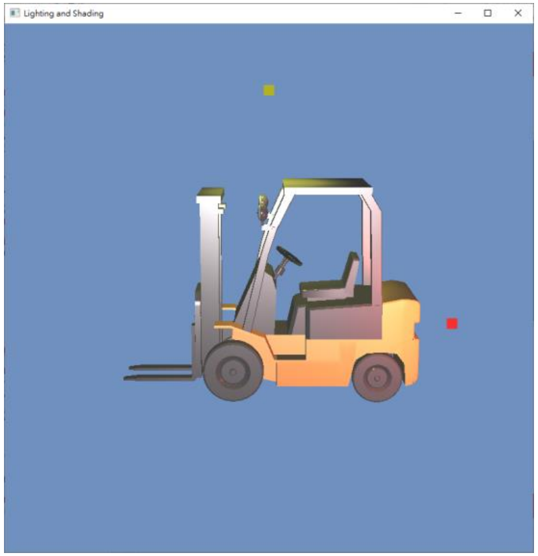
- Loading the material data correctly (35%) [Test Models]
- Implement ***Phong*** shading correctly with
  - Ambient light (5%)
  - Diffuse and specular shading with a point light (15%)
  - Diffuse and specular shading with a directional light (10%)
  - Diffuse and specular shading with a spot light (15%)
- Code organization (10%)
- Report (5%)
  - Introduce your implementation and put some screenshots
- Bonus (5%)
  - Propose a way to visualize directional light or
  - Implement a microfacet model

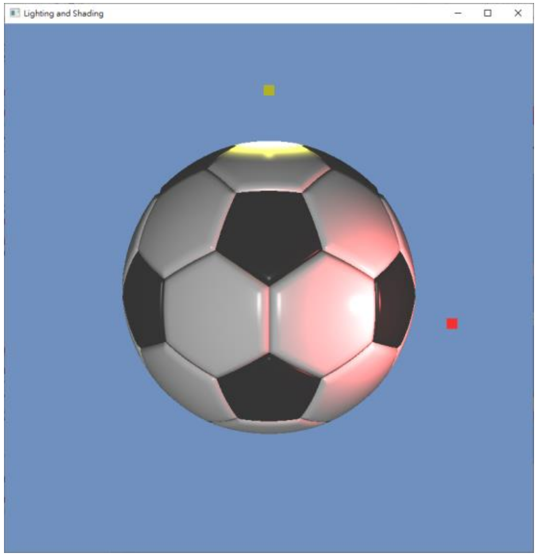
Reference Results

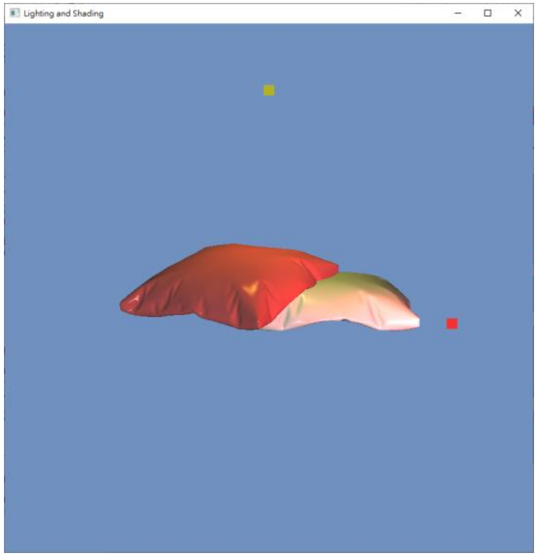












Submission

- **Deadline: Dec. 04, 2022 (PM 11:59)**

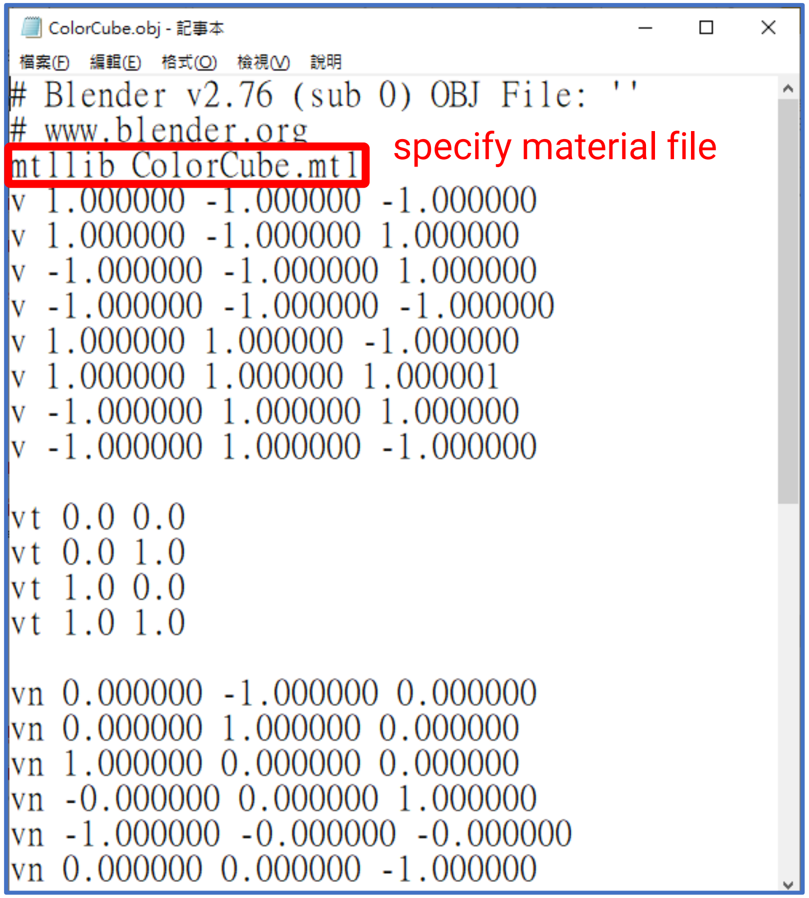
- **Submission rule**
  - The same as HW1

- **Late policy**

• One day	90%
• Two days	80%
• Three days	70%
• Four days	60%
• Five days+	50%

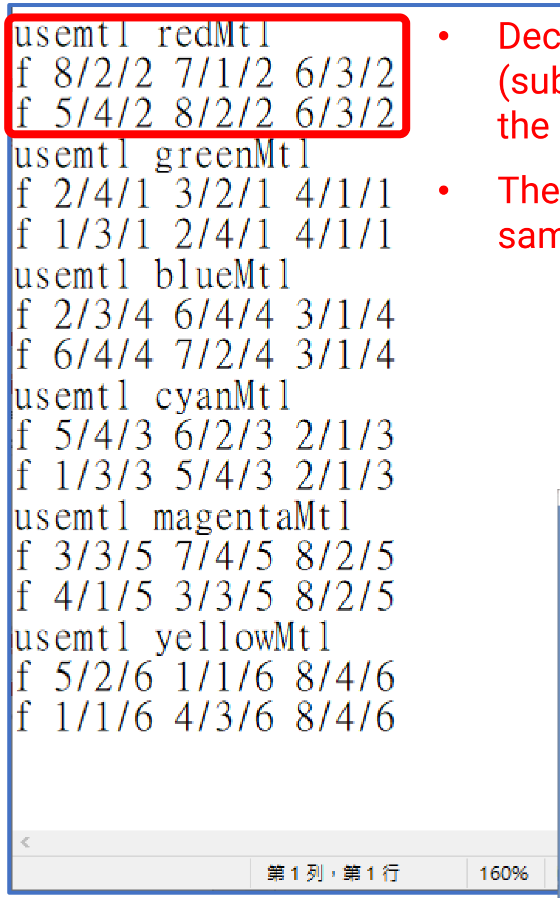
Recap: Material Template Format

- ColorCube.obj



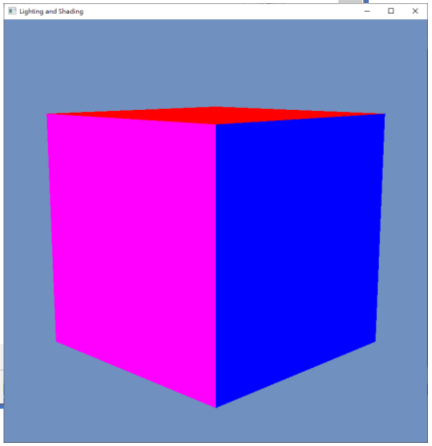
```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib ColorCube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000

vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0

vn 0.000000 -1.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -0.000000 0.000000 1.000000
vn -1.000000 -0.000000 -0.000000
vn 0.000000 0.000000 -1.000000
```

specify material file



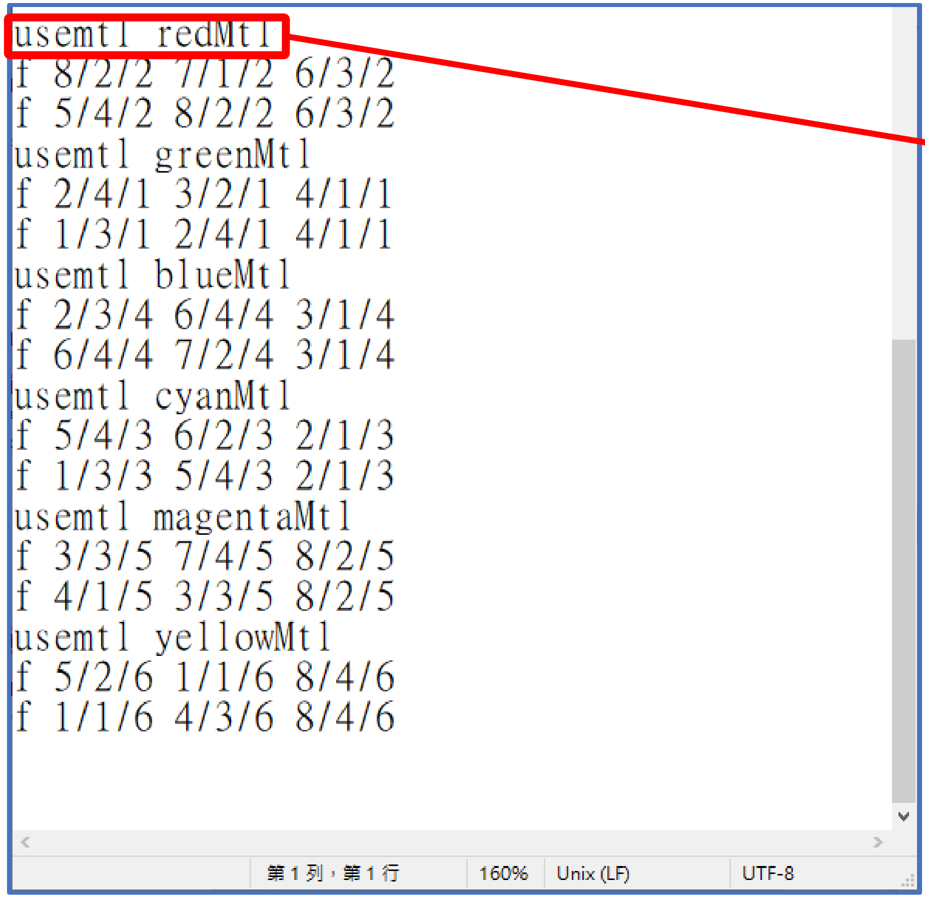
```
usemtl redMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
usemtl greenMtl
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
usemtl blueMtl
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
usemtl cyanMtl
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
usemtl magentaMtl
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
usemtl yellowMtl
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

- Declare a new group (submesh) that uses the “redMtl” material
- These faces use the same “redMtl” material

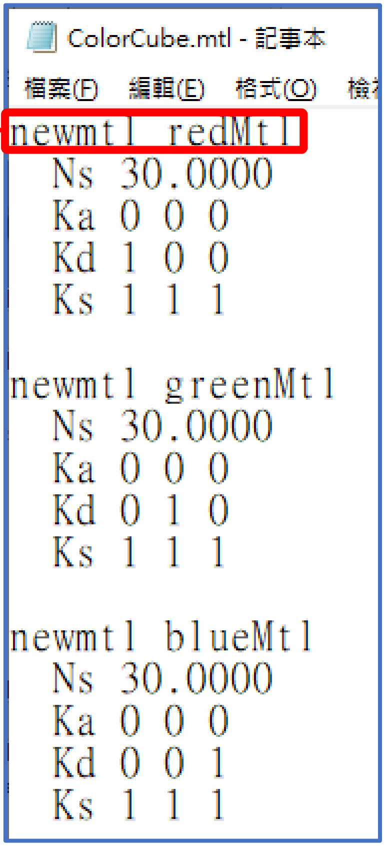


Recap: Material Template Format

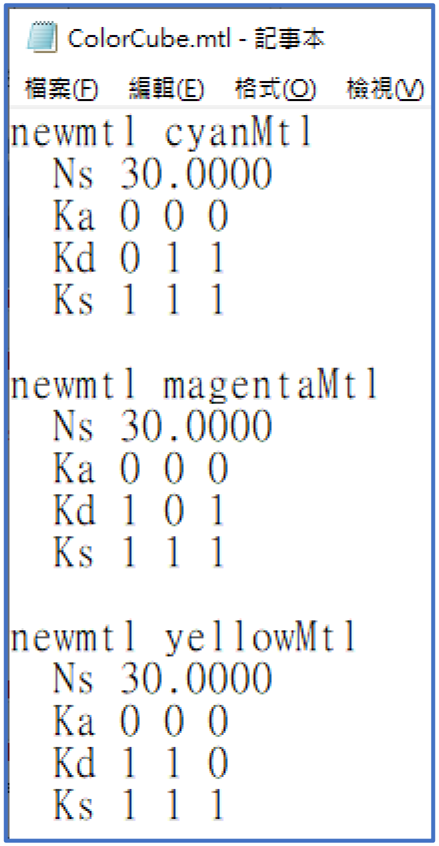
- ColorCube.mtl



```
usemtl redMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
usemtl greenMtl
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
usemtl blueMtl
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
usemtl cyanMtl
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
usemtl magentaMtl
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
usemtl yellowMtl
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```



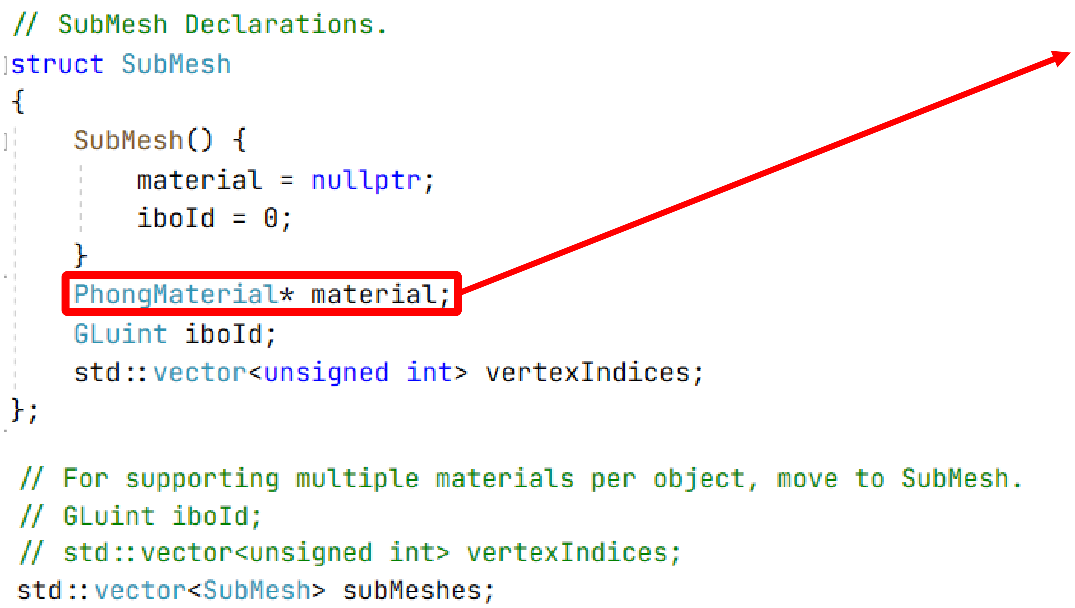
```
newmtl redMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 1 0 0
    Ks 1 1 1

newmtl greenMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 0 1 0
    Ks 1 1 1

newmtl blueMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 0 0 1
    Ks 1 1 1
```



```
newmtl cyanMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 0 1 1
    Ks 1 1 1

newmtl magentaMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 1 0 1
    Ks 1 1 1

newmtl yellowMtl
    Ns 30.0000
    Ka 0 0 0
    Kd 1 1 0
    Ks 1 1 1
```

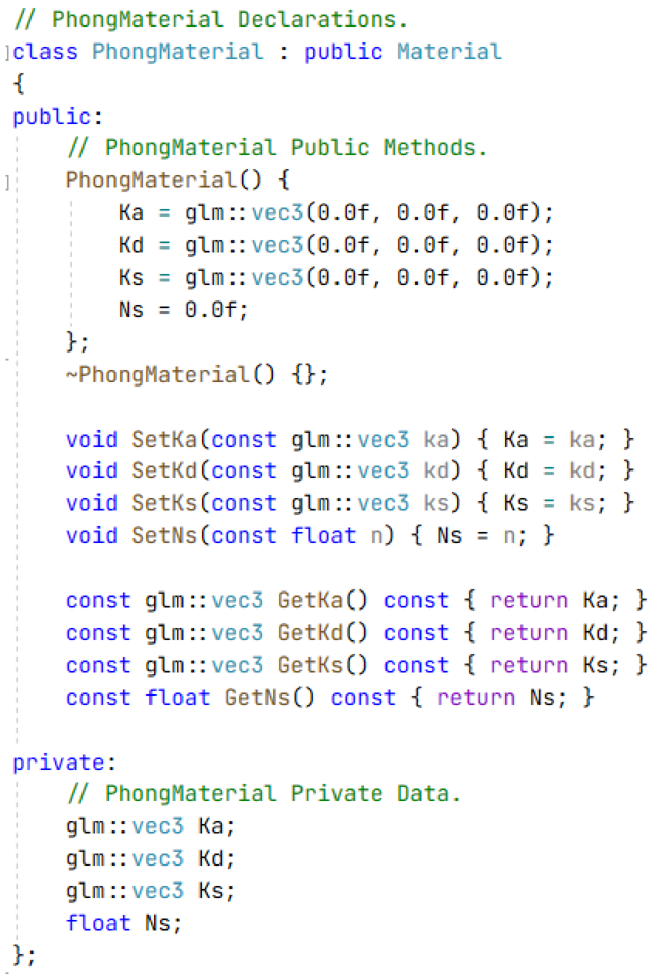
SubMesh Structure

in trianglemesh.h



```
// SubMesh Declarations.
struct SubMesh
{
    SubMesh() {
        material = nullptr;
        iboId = 0;
    }
    PhongMaterial* material;
    GLuint iboId;
    std::vector<unsigned int> vertexIndices;
};

// For supporting multiple materials per object, move to SubMesh.
// GLuint iboId;
// std::vector<unsigned int> vertexIndices;
std::vector<SubMesh> subMeshes;
```

in material.h



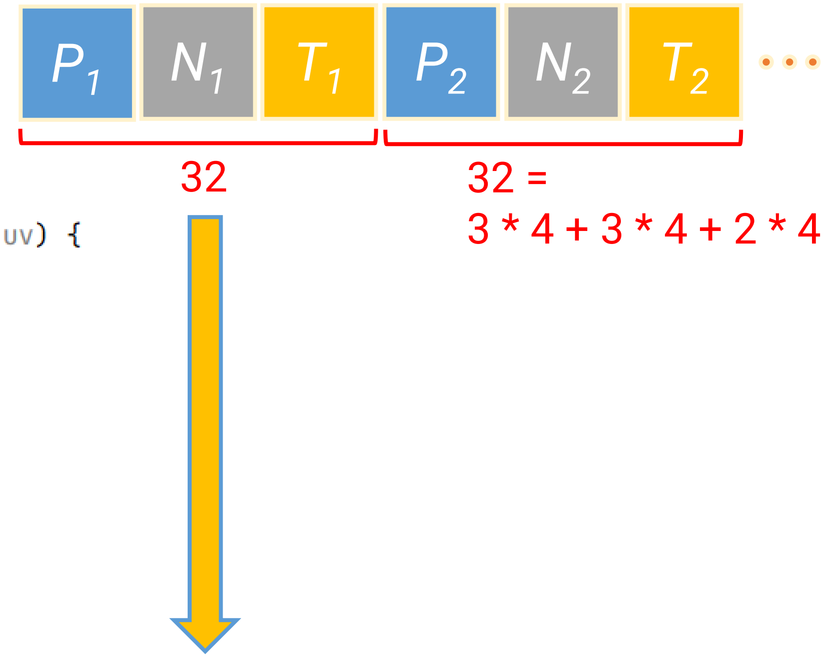
```
// PhongMaterial Declarations.
class PhongMaterial : public Material
{
public:
    // PhongMaterial Public Methods.
    PhongMaterial() {
        Ka = glm::vec3(0.0f, 0.0f, 0.0f);
        Kd = glm::vec3(0.0f, 0.0f, 0.0f);
        Ks = glm::vec3(0.0f, 0.0f, 0.0f);
        Ns = 0.0f;
    };
    ~PhongMaterial() {};

    void SetKa(const glm::vec3 ka) { Ka = ka; }
    void SetKd(const glm::vec3 kd) { Kd = kd; }
    void SetKs(const glm::vec3 ks) { Ks = ks; }
    void SetNs(const float n) { Ns = n; }

    const glm::vec3 GetKa() const { return Ka; }
    const glm::vec3 GetKd() const { return Kd; }
    const glm::vec3 GetKs() const { return Ks; }
    const float GetNs() const { return Ns; }

private:
    // PhongMaterial Private Data.
    glm::vec3 Ka;
    glm::vec3 Kd;
    glm::vec3 Ks;
    float Ns;
};
```

Recap: Multiple Vertex Attributes (cont.)

- Example: with position/normal/texcoord data

```
// VertexPTN Declarations.
struct VertexPTN
{
    VertexPTN() {
        position = glm::vec3(0.0f, 0.0f, 0.0f);
        normal = glm::vec3(0.0f, 1.0f, 0.0f);
        texcoord = glm::vec2(0.0f, 0.0f);
    }
    VertexPTN(glm::vec3 p, glm::vec3 n, glm::vec2 uv) {
        position = p;
        normal = n;
        texcoord = uv;
    }
    glm::vec3 position;
    glm::vec3 normal;
    glm::vec2 texcoord;
};
```

vertex buffer layout



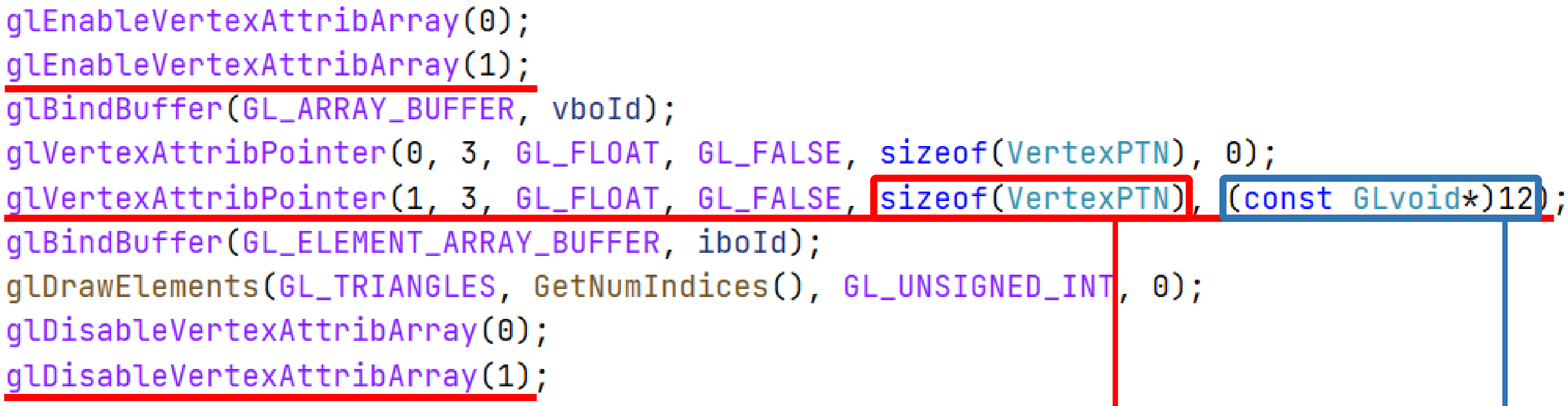
- During rendering:

stride = 32

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), 0);
```

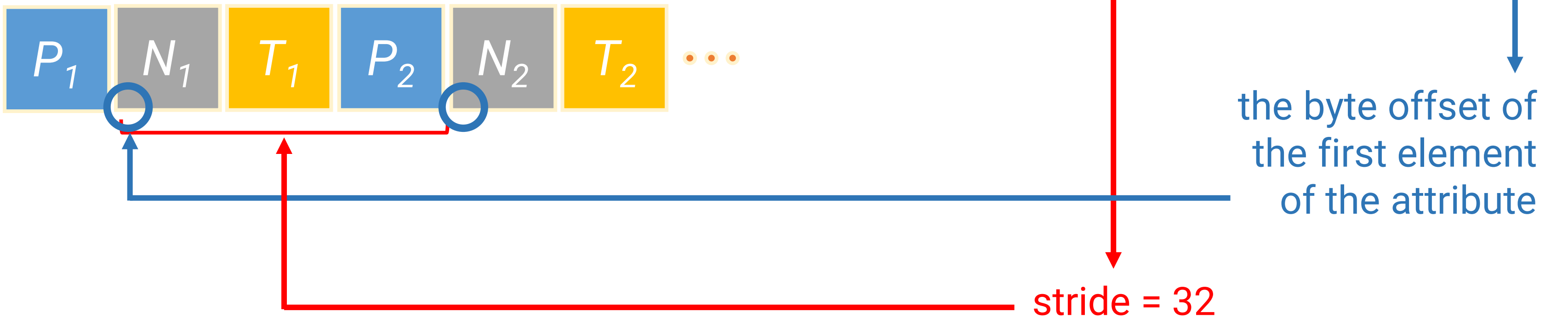
Recap: Multiple Vertex Attributes (cont.)

- Render with only the position and normal attributes



```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vboId);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), 0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPTN), (const GLvoid*)12);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboId);
glDrawElements(GL_TRIANGLES, GetNumIndices(), GL_UNSIGNED_INT, 0);
glDisableVertexAttribArray(0);
glDisableVertexAttribArray(1);
```

vertex buffer layout



the byte offset of the first element of the attribute

stride = 32

Recap: Multiple Vertex Attributes (cont.)

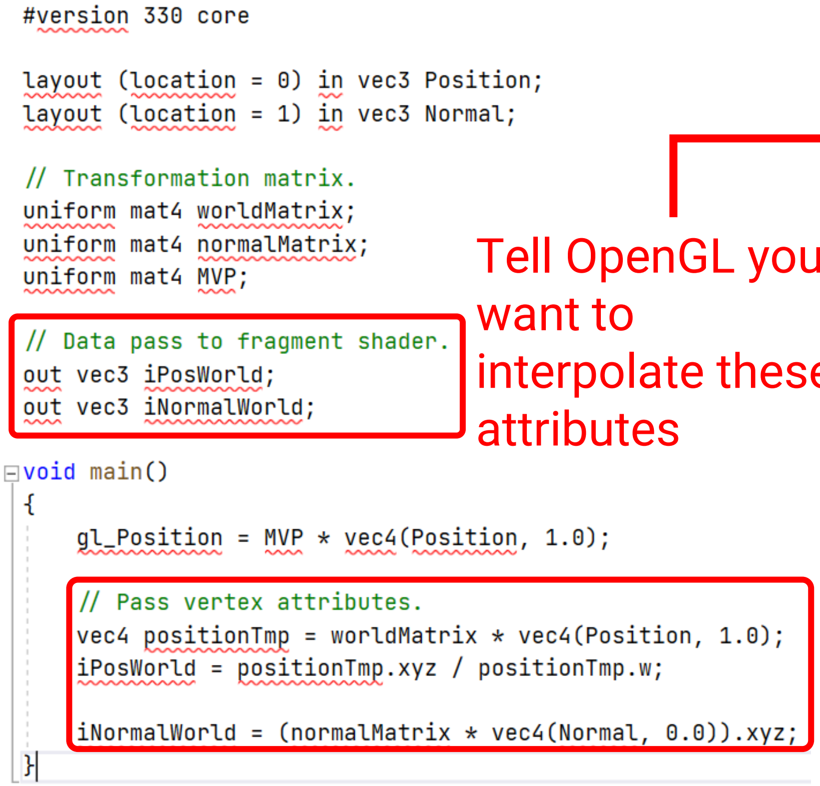
- Vertex shader

```
#version 330 core
layout (location = 0) in vec3 Position;
layout (location = 1) in vec3 Normal;

// Transformation matrices.
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 normalMatrix;
uniform mat4 MVP;
…
```

Recap: Vertex Attribute Interpolation

- Example: interpolate **world-space vertex position** and **world-space vertex normal**

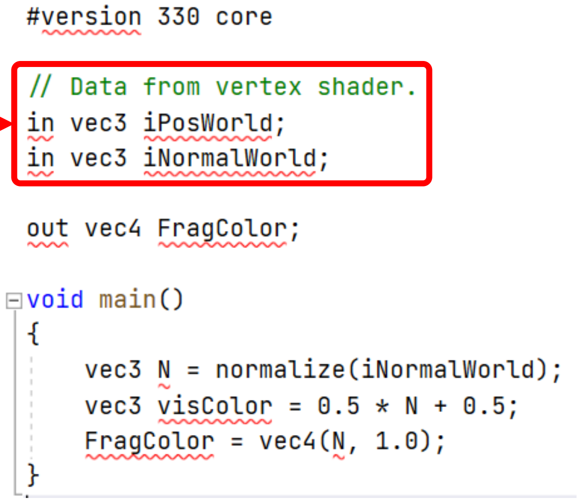
Vertex Shader



```
#version 330 core

layout (location = 0) in vec3 Position;
layout (location = 1) in vec3 Normal;

// Transformation matrix.
uniform mat4 worldMatrix;
uniform mat4 normalMatrix;
uniform mat4 MVP;

// Data pass to fragment shader.
out vec3 iPosWorld;
out vec3 iNormalWorld;

void main()
{
    gl_Position = MVP * vec4(Position, 1.0);

    // Pass vertex attributes.
    vec4 positionTmp = worldMatrix * vec4(Position, 1.0);
    iPosWorld = positionTmp.xyz / positionTmp.w;

    iNormalWorld = (normalMatrix * vec4(Normal, 0.0)).xyz;
}
```

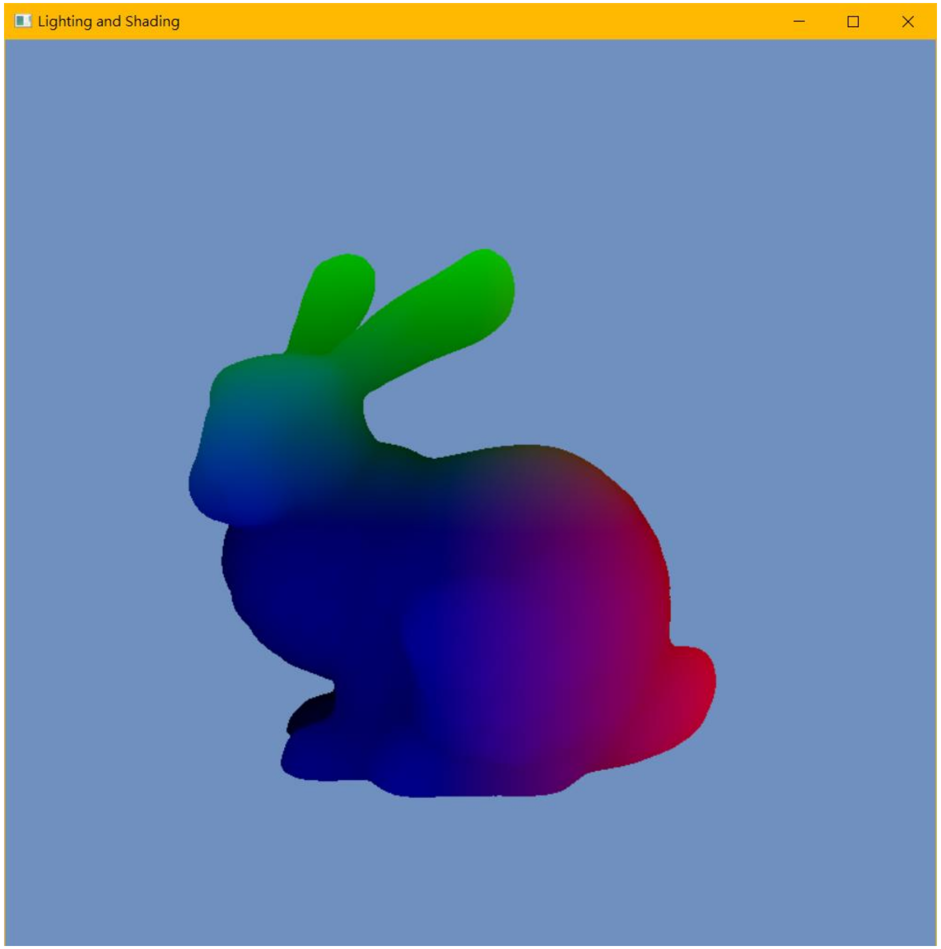
Tell OpenGL you want to interpolate these attributes

Fragment Shader

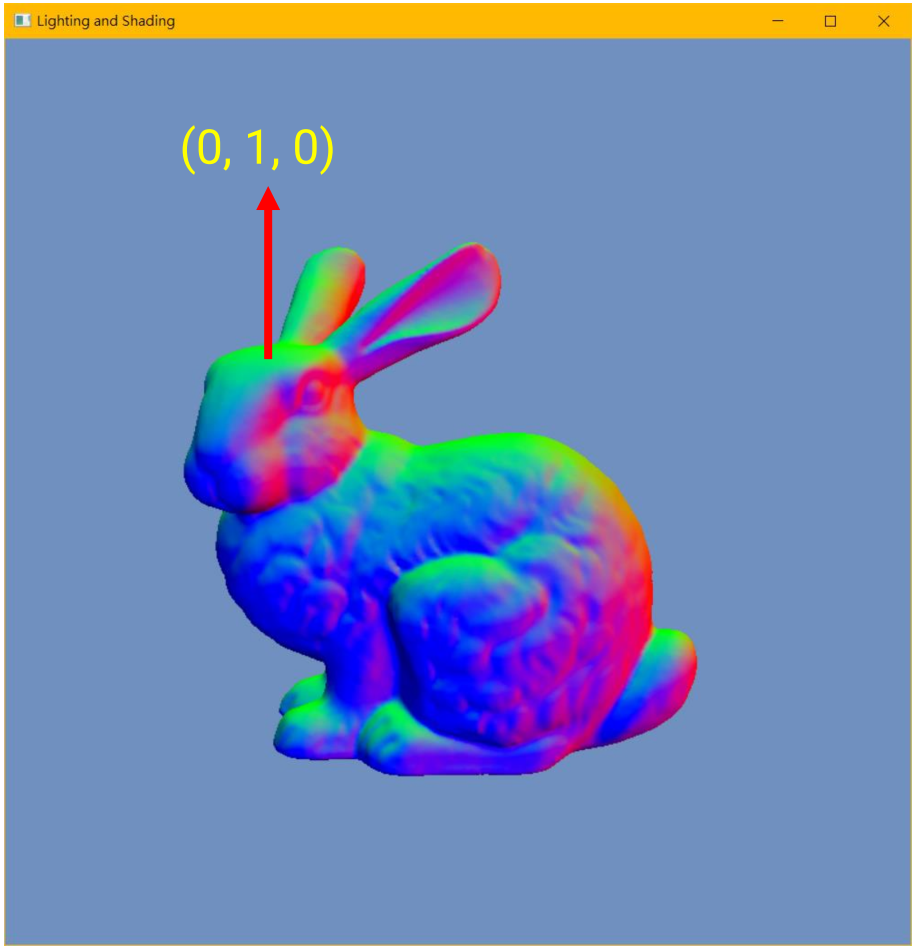


```
#version 330 core

// Data from vertex shader.
in vec3 iPosWorld;
in vec3 iNormalWorld;

out vec4 FragColor;

void main()
{
    vec3 N = normalize(iNormalWorld);
    vec3 visColor = 0.5 * N + 0.5;
    FragColor = vec4(N, 1.0);
}
```

Recap: Vertex Attribute Interpolation (cont.)



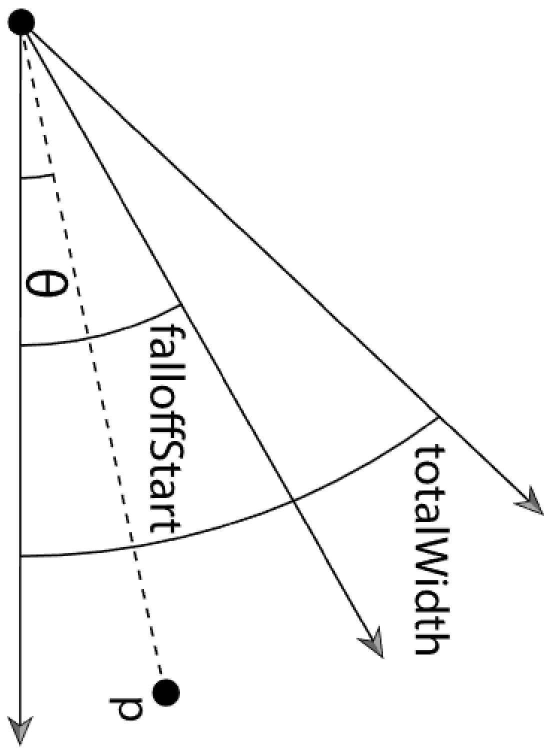
visualize world-space position as color



visualize world-space normal as color

Recap: Spot Light

- Surface inside falloffStart can get **full** contribution from the spot light
- Surface outside totalWidth gets **zero** contribution from the spot light
- Surface between falloffStart and totalWidth receives a **linearly falloff (w.r.t the cosine value of angle)** contribution from the spot light



- You also need to compute the Lambertian term and the attenuation

Task List

- Revise your **vertex buffer** to support **multiple vertex attributes** (position and **normal**)
- Load and create the materials defined in a **material file**
- Revise your OBJ loader (implemented in HW1) with the **SubMesh** structure for supporting multiple materials
- Revise the rendering function for the vertex buffer with multiple vertex attributes and the the SubMesh structure
- Implement the **SpotLight** class
- Incorporate shaders and implement **per-fragment** ambient, diffuse, and specular shading
- Implement a bonus if you want

Task List (cont.)

- Please download the skeleton code from 數位學苑 3.0
- You might need to retouch (but not limited to):
  - ICG2022_HW2.cpp

    ➔ Mostly in **RenderSceneCB()**

    ➔ Other minor parts if you would like to change the flow of program (please refer to the **comment!**)
  - trianglemesh.h / trianglemesh.cpp

    ➔ Add your HW1 code & **materials loading**

(cont.)

Task List (cont.)

- You might need to retouch (but not limited to):
  - light.h

    ➔ Add your implementation of the **SpotLight** class
  - shaderprog.h / shaderprog.cpp

    ➔ Add some data/methods for the **SpotLight** object
  - phong_shading_demo.vs / phong_shading_demo.fs

    ➔ The entire shaders!

Any Questions?