Lighting and Shading

Computer Graphics

Yu-Ting Wu

Recap.

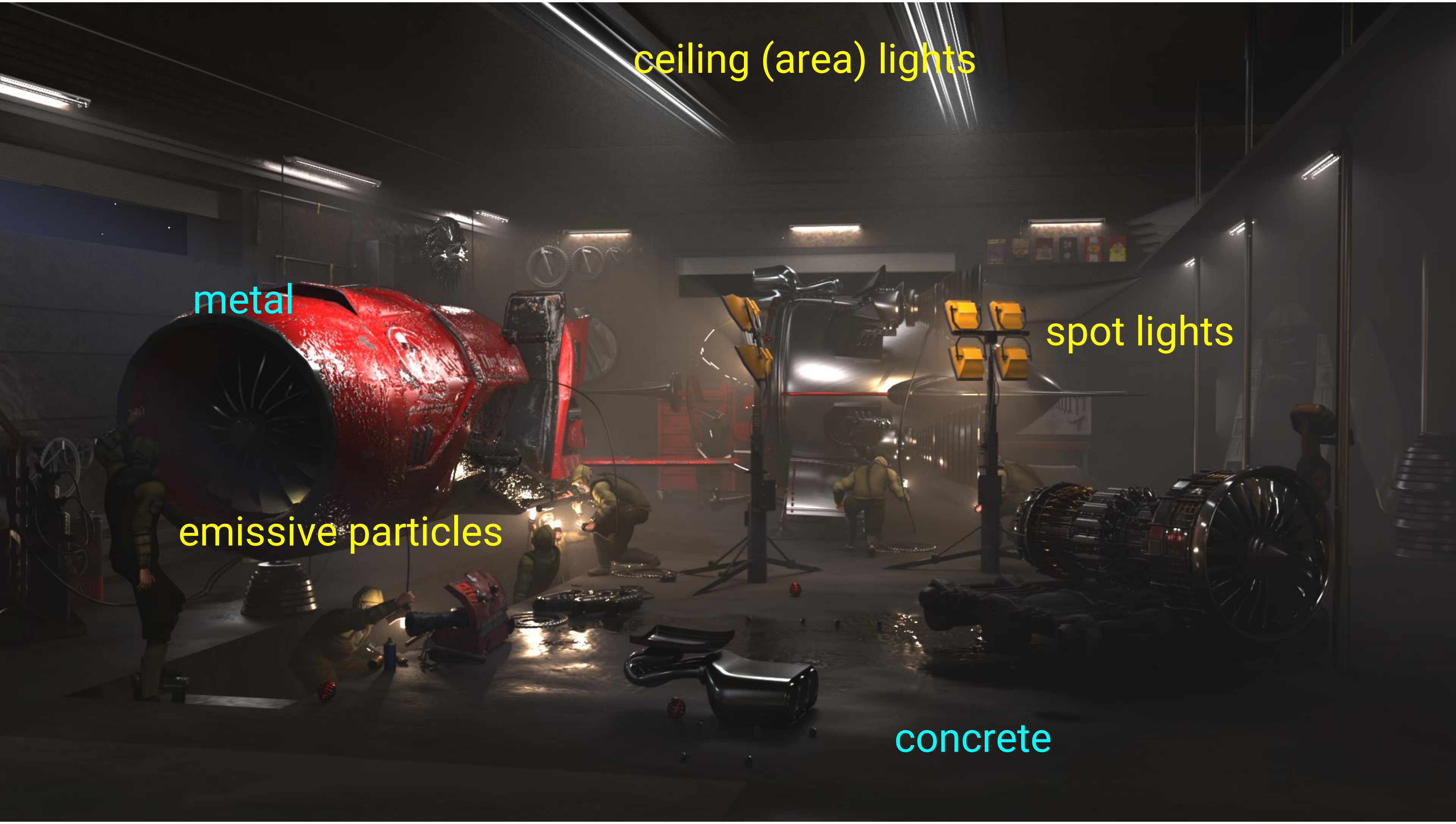
- Prior the midterm, we have introduced
  - How to represent a 3D scene
  - How the virtual camera works
  - How to bring triangles into pixels with the GPU graphics pipeline
- In the following weeks, we will talk about how to determine the fragment color
  - **Lighting and shading**
  - Texture mapping
  - Alpha blending for transparency objects

Outline

Outline

- **Overview**
- Lights
- Materials
- OpenGL implementation

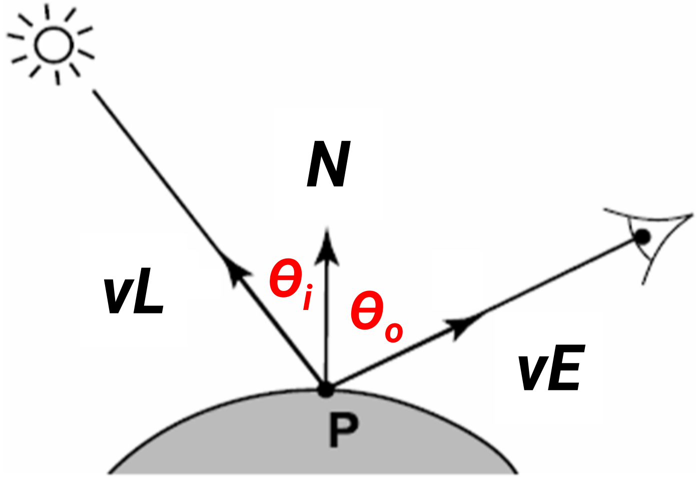
Shading: Materials and Lighting



Shading: Materials and Lighting (cont.)

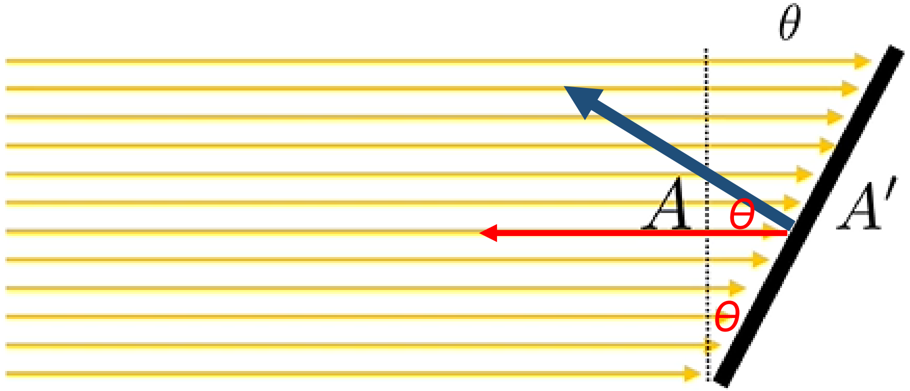
Shading

- Shading refers to the process of altering the color of an object/surface/polygon in the 3D scene
- In physically-based rendering, shading tries to approximate the **local behavior** of lights on the object's surface, based on things like
  - Surface orientation (normal) ***N***
  - Lighting direction ***vL*** (and θ_i)
  - Viewing direction ***vE*** (and θ_o)
  - Material properties
  - Participating media
  - etc.



Lambertian Cosine Law

- Illumination on an oblique surface is less than on a normal one
- Generally, illumination falls off as **cosθ**



$$A = A' \cos\theta$$

$$E = \frac{\Phi}{A'} = \frac{\Phi\cos\theta}{A}$$

Outline

- Overview
- **Lights**
- Materials
- OpenGL implementation

Lights in Computer Graphics

- Point light
- Spot light
- Area light

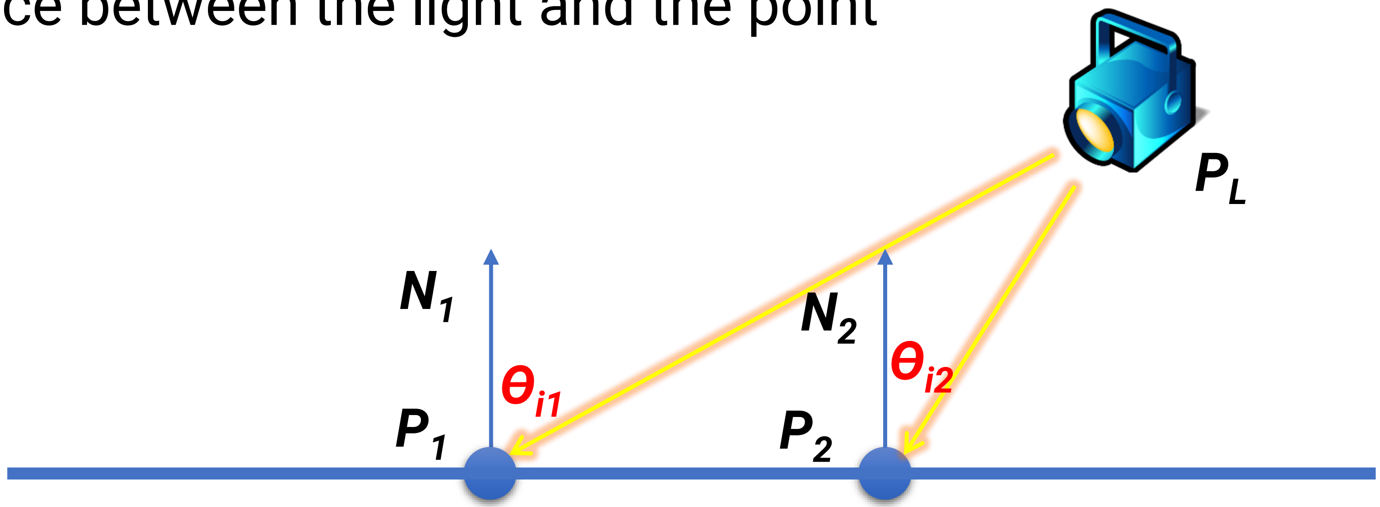
local lights

- Directional light
- Environment light

distant lights

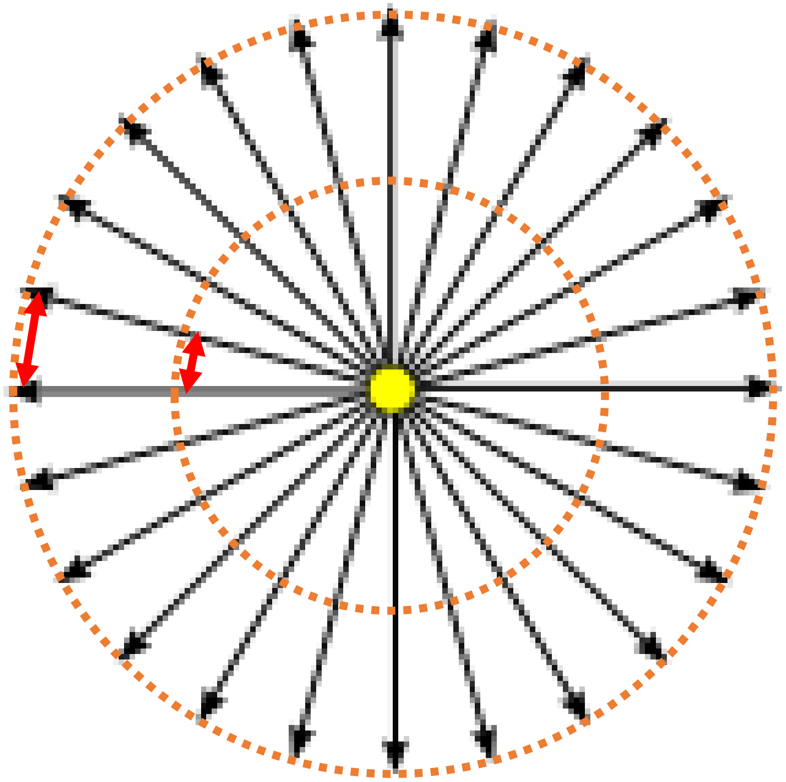
Local Light

- The distance between a light and a surface is **NOT** long enough compared to the scene scale
- The position of light needs to be considered during shading
  - **Lighting direction vL = $|P_L - P|$**
  - **Lighting attenuation** is proportional to the square of the distance between the light and the point



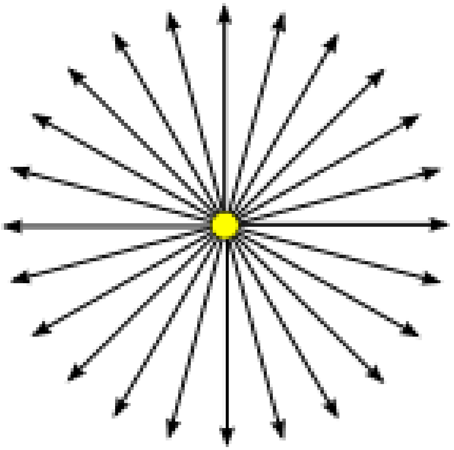
Local Light Attenuation

- The length of the side of a receiver patch is proportional to its distance from the light
- As a result, the average energy per unit area is proportional to the **square of the distance** from the light



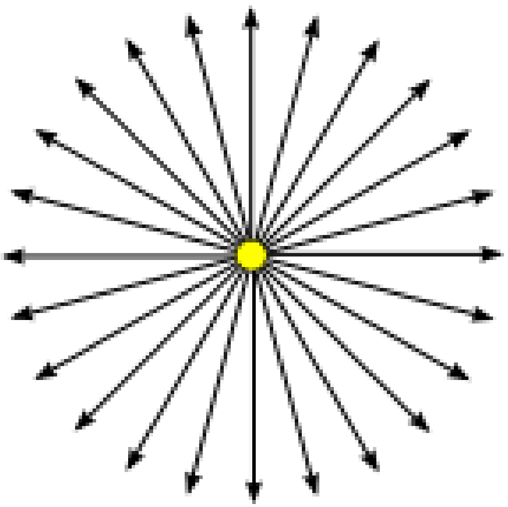
Point Light

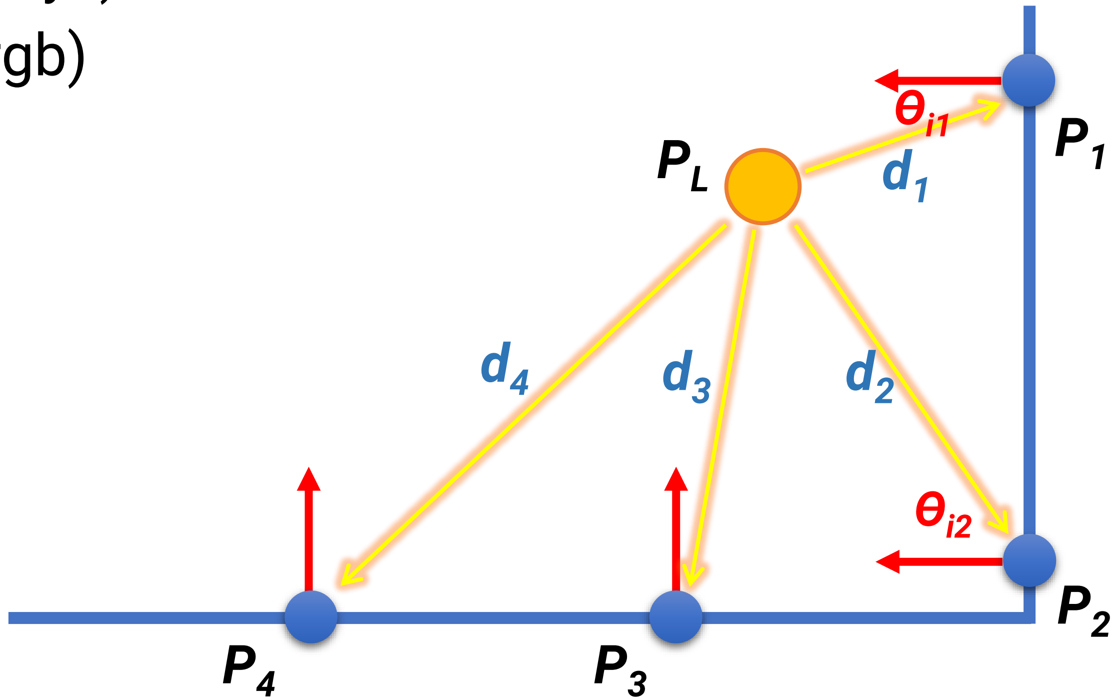
A scene illuminated by a point light



Point Light (cont.)

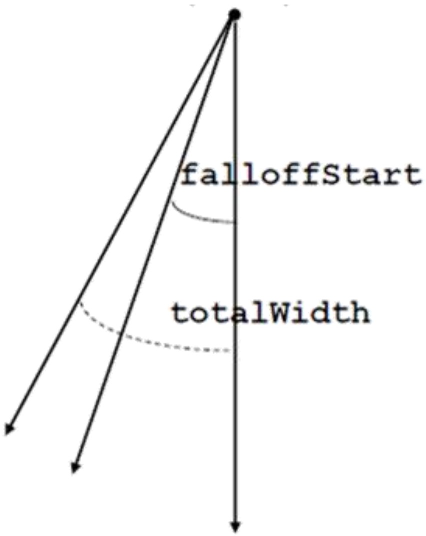
- An isotropic point light source that emits the same amount of light in all directions
- Described by
  - Light position (**P**$_L$, xyz)
  - Light intensity (**I**, rgb)



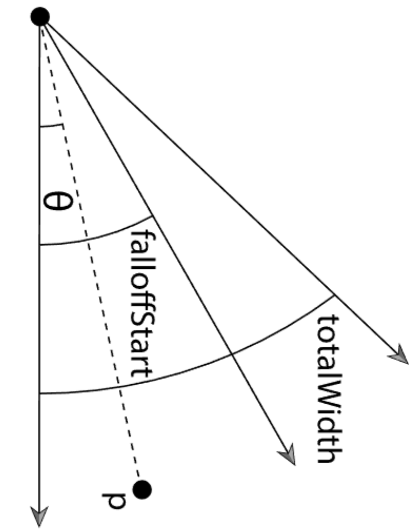


Spot Light

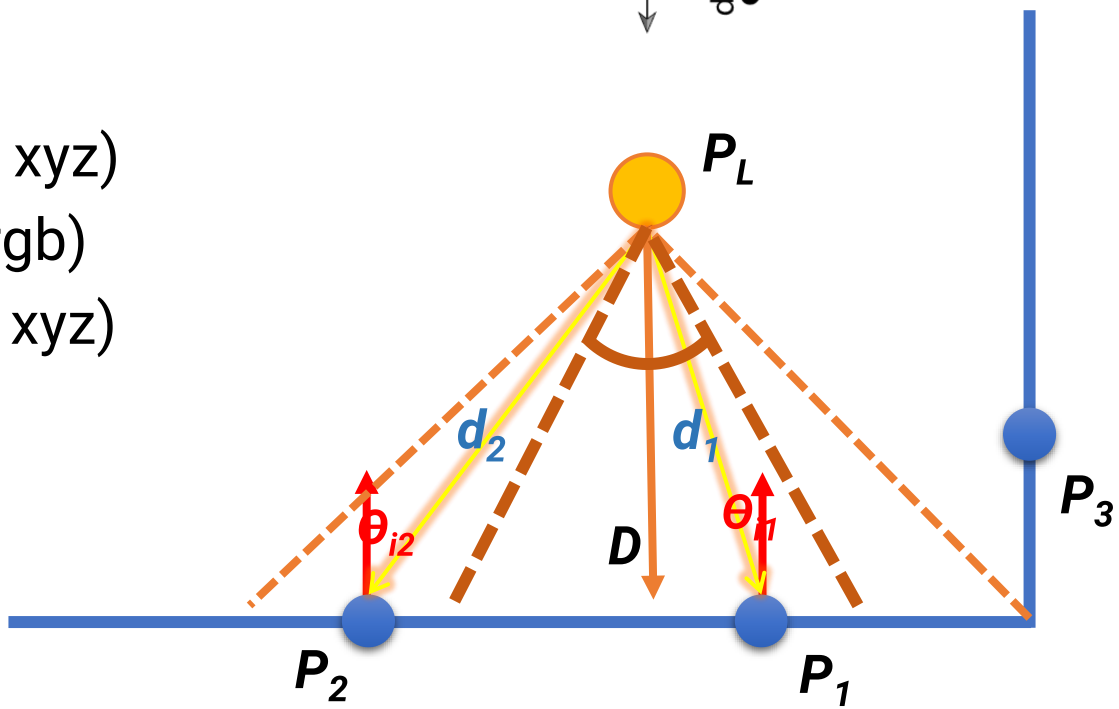
A scene illuminated by a spot light



Spot Light (cont.)

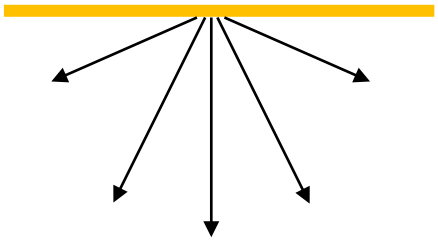


- A handy variation on point lights
- Rather than shining illumination in all directions, it emits light in a cone of directions from its position
- Described by
  - Light position (**P**$_L$, xyz)
  - Light intensity (**I**, rgb)
  - Light direction (**D**, xyz)
  - TotalWidth
  - FalloffStart



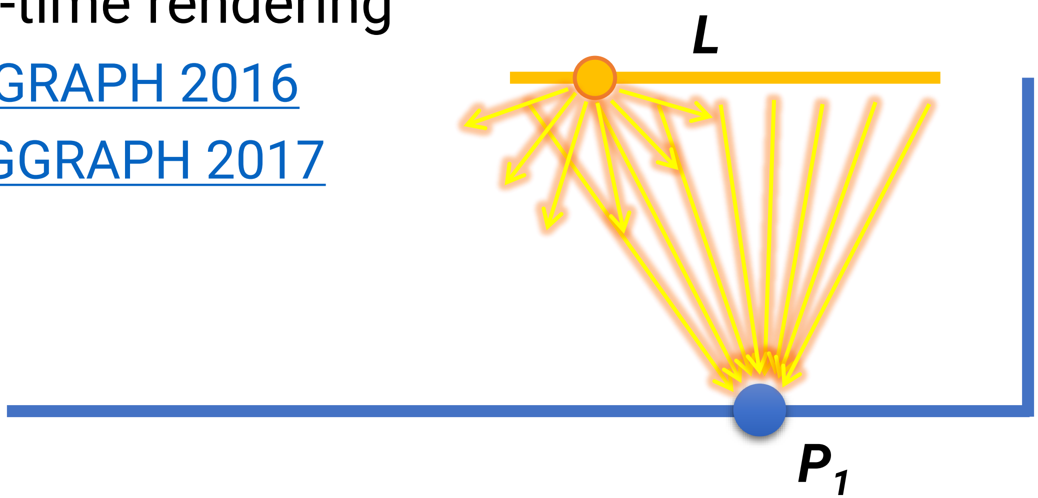
Area Light

A scene illuminated by an area light



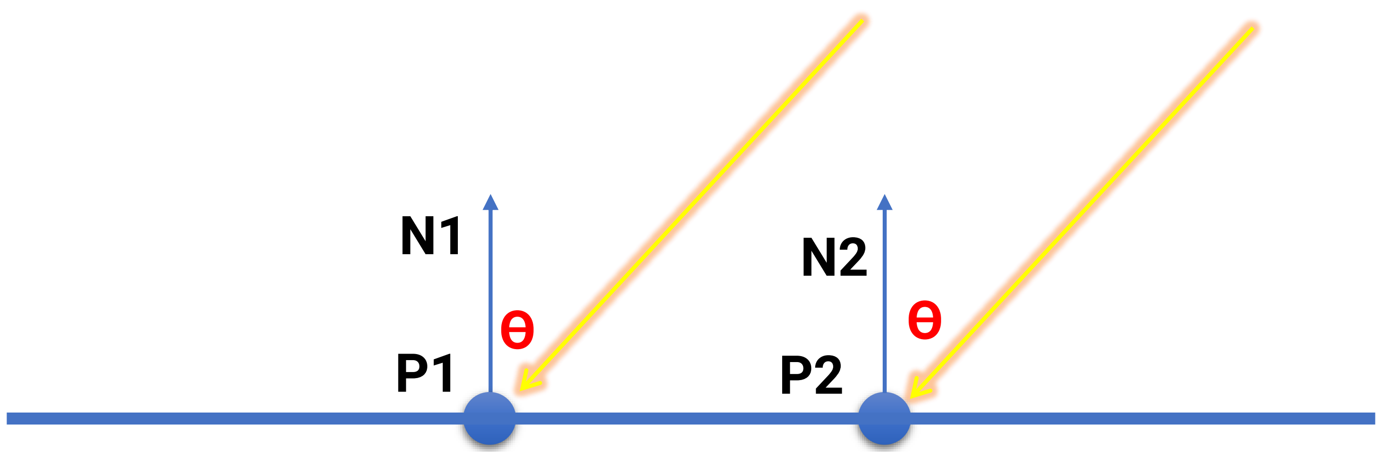
Area Light (cont.)

- Defined by one or more **shapes** that emit light from their surface, with some directional distribution of energy at each point on the surface
- Require **integration** of lighting contribution across the light surface
    - In offline rendering, usually estimated by sampling
    - Expensive for real-time rendering
        - [Heitz et al., SIGGRAPH 2016]()
        - [Dupuy et al., SIGGRAPH 2017]()



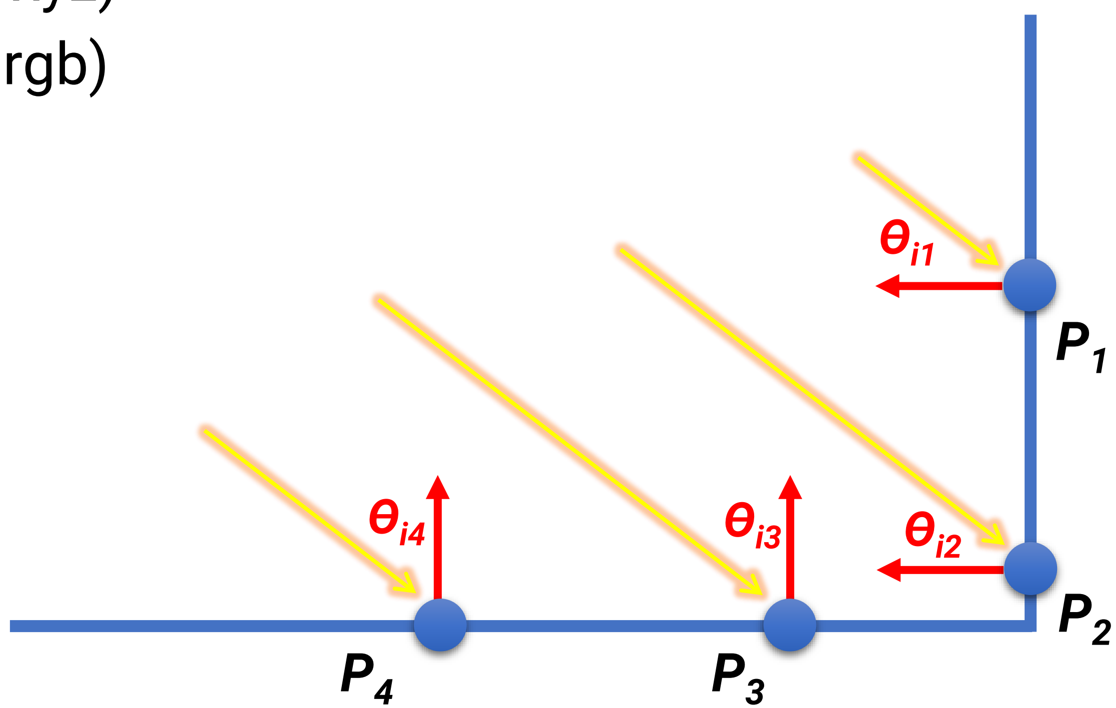
Distant Light

- The distance between a light and a surface is long enough compared to the scene scale and **can be ignored**
  - **Lighting direction** is **fixed**
  - **No lighting attenuation**
- **Directional light (sun)** is the most common distant light



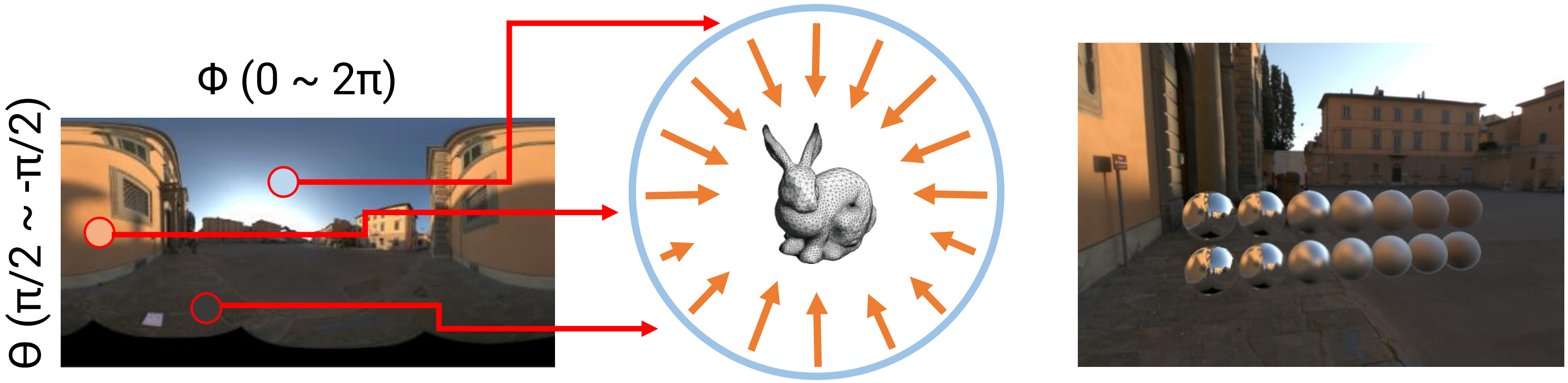
Directional Light

- Describes an emitter that deposits illumination from the **same direction** at every point in space
- Described by
  - Light direction (**D**, xyz)
  - Light radiance (**L**, rgb)



Environment Light

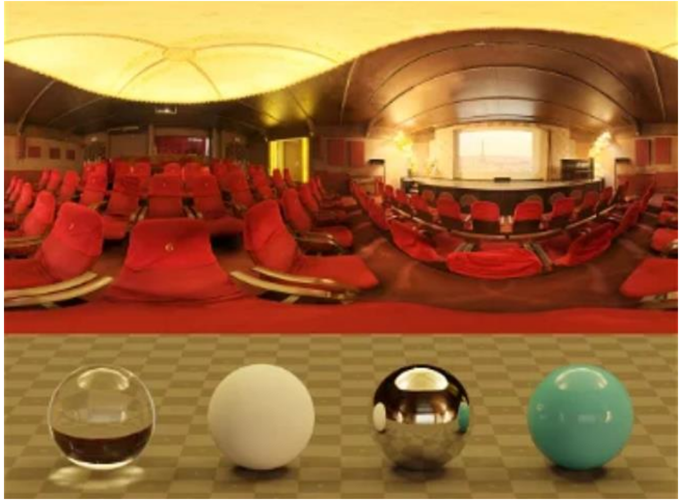
- Use a **texture** (cube map or longitude-latitude image) to represent a **spherical energy distribution**
  - Each texel maps to a spherical direction, considered as a directional light
  - The whole map illuminates the scene from a virtual sphere at an infinite distance
- Also called **image-based lighting (IBL)**

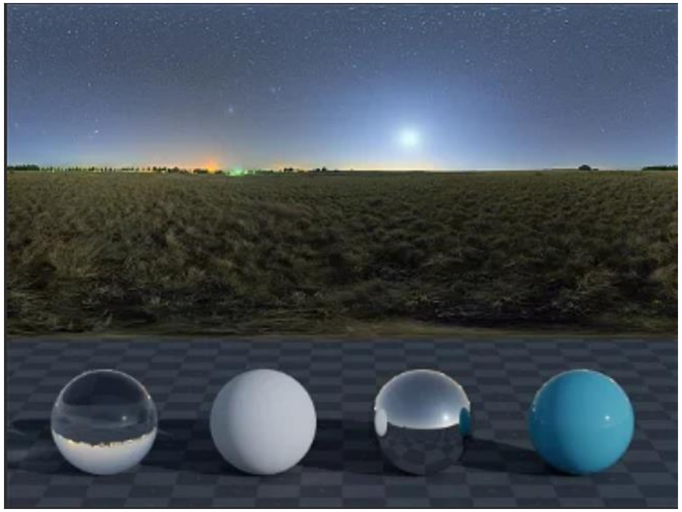


Environment Light (cont.)

- Widely used in digital visual effects and film production

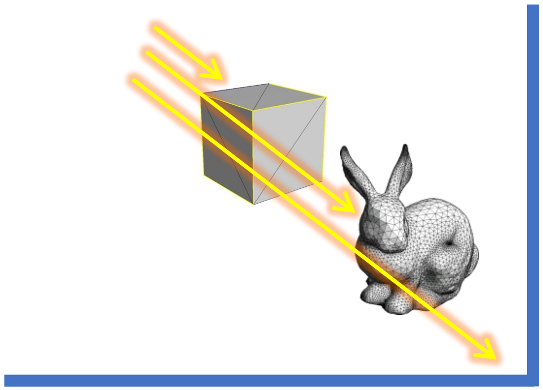
Environment Light (cont.)



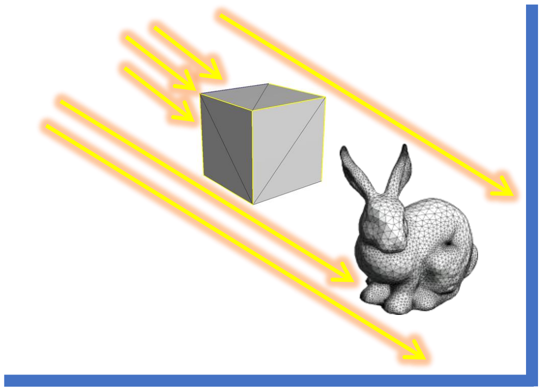


Local, Direct, and Global Illumination

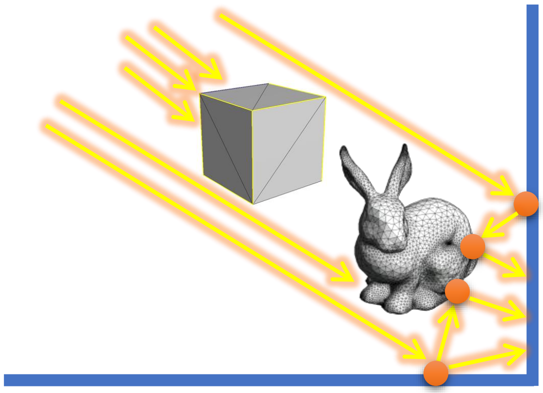
- Direct illumination considers only the **direct** contribution of lights
- Local illumination can be considered as direct lighting **without occlusion** (all lights are fully visible, no shadows)
- Global illumination includes **multi-bounce** illumination reflected from other surfaces (need **recursive** computation!)



local illumination

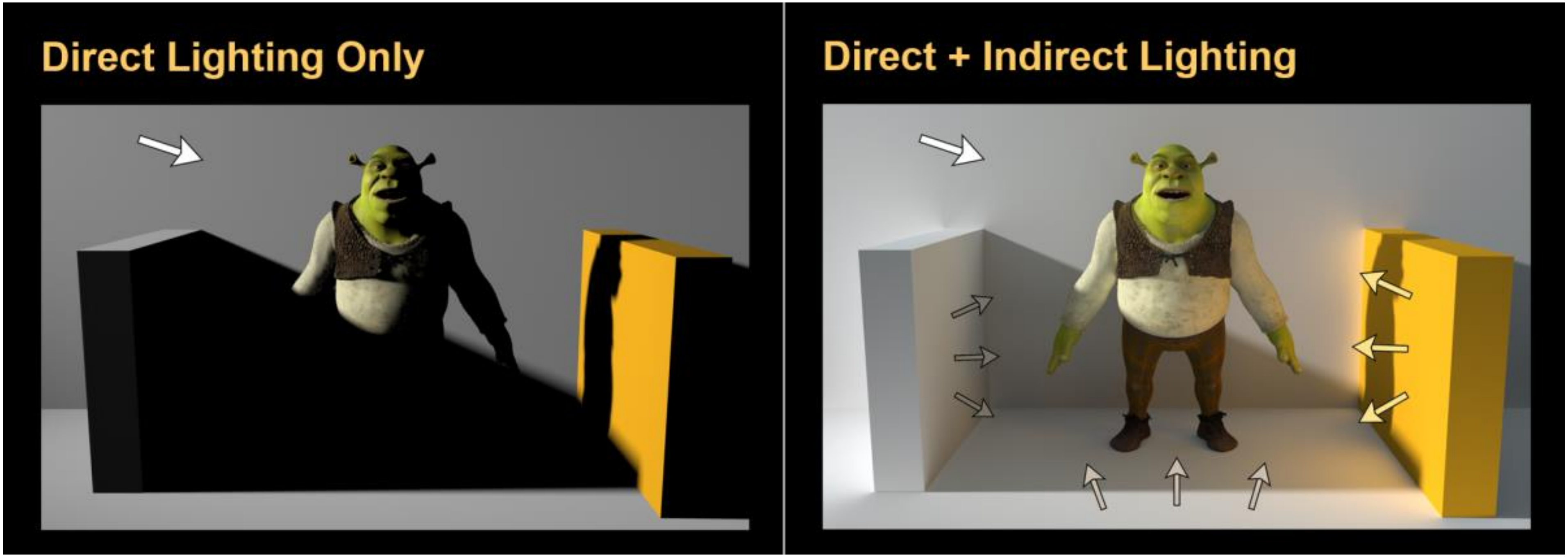


direct illumination



global illumination

Local, Direct, and Global Illumination (cont.)

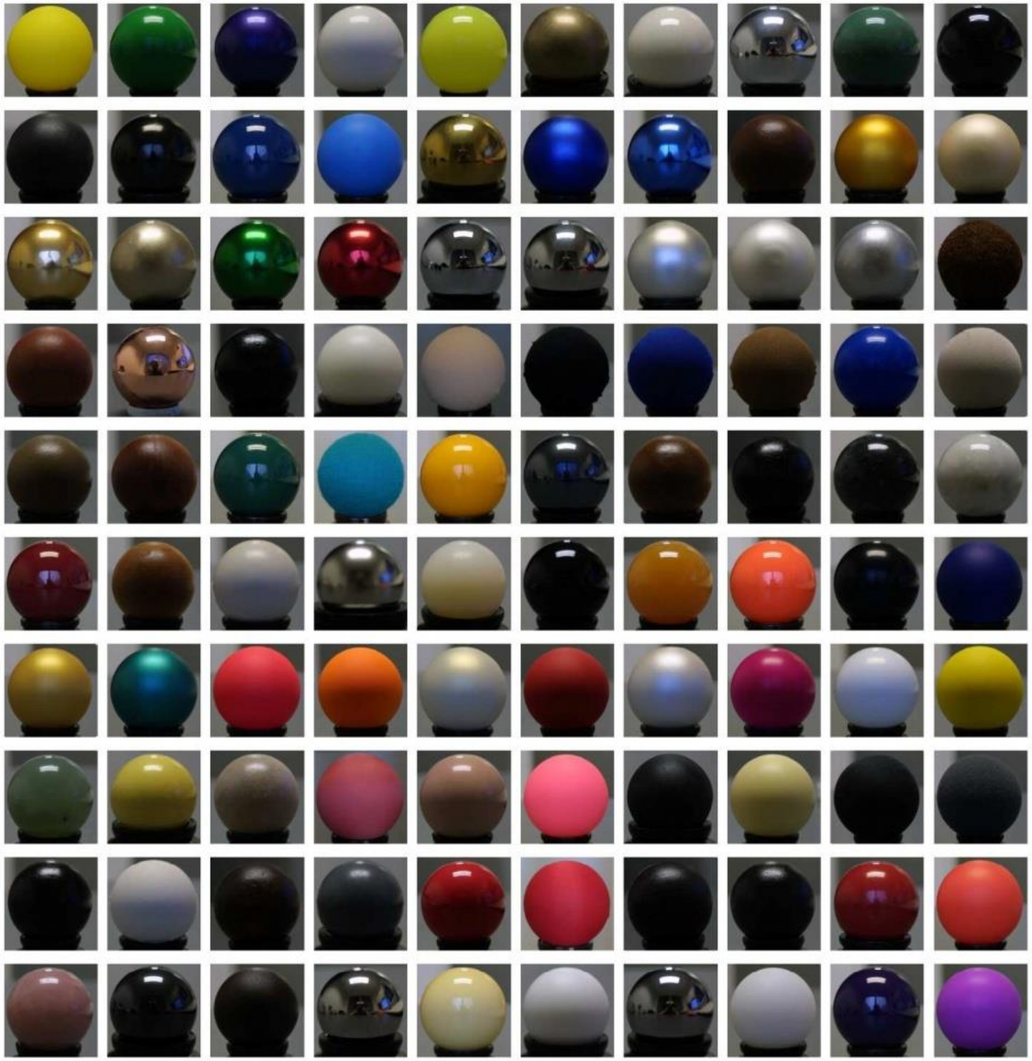


Comparison of direct and global illumination

Outline

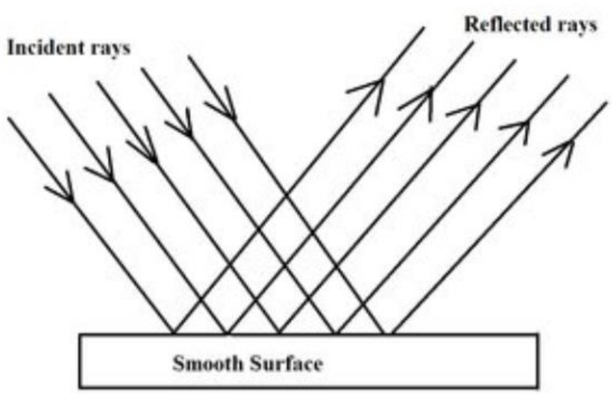
- Overview
- Lights
- **Materials**
- OpenGL implementation

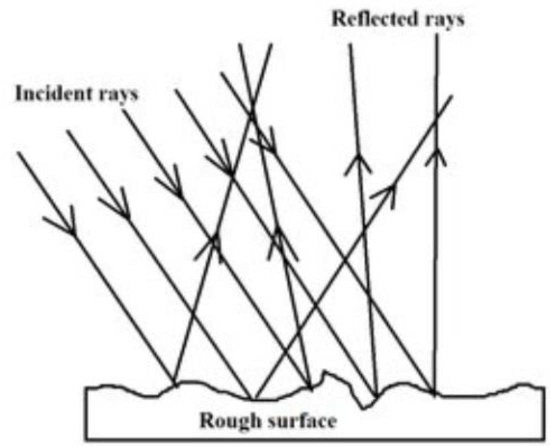
Materials



Materials (cont.)

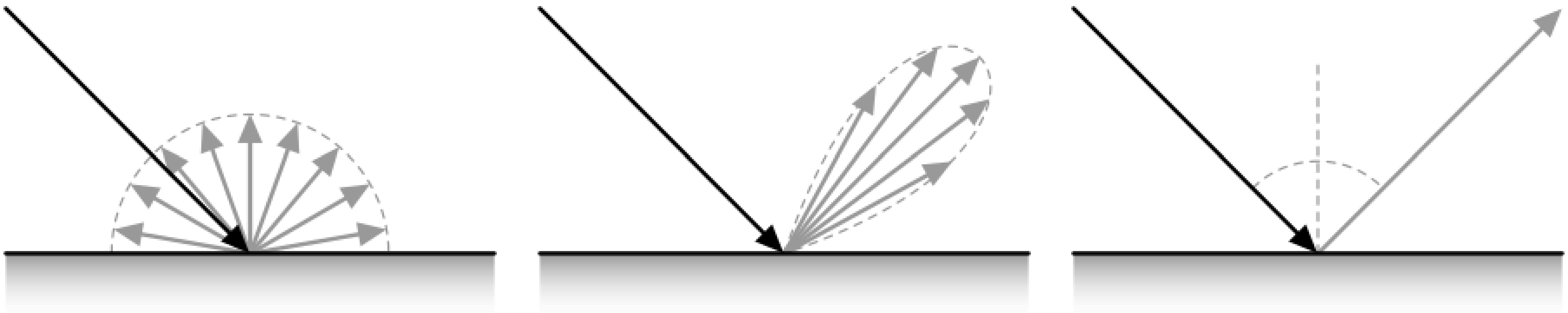
- Highly related to surface types
- The **smoother** a surface, the more reflected light is concentrated in the direction a **perfect mirror** would reflect the light





Materials (cont.)

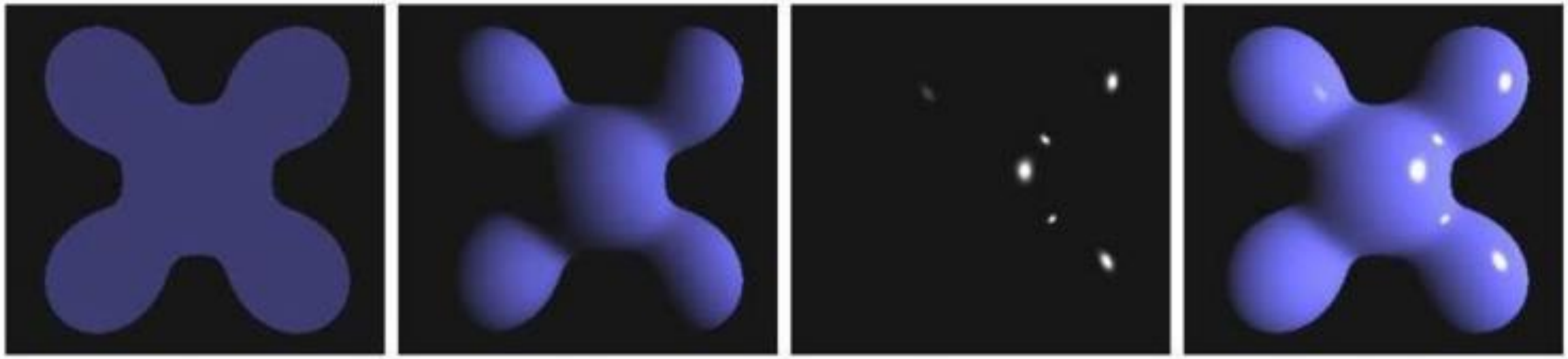
- Highly related to surface types
- The **smoother** a surface, the more reflected light is concentrated in the direction a **perfect mirror** would reflect the light



diffuse glossy specular

Phong Lighting Model

- **Diffuse reflection**
  - Light goes everywhere; colored by object color
- **Specular reflection**
  - Happens only near mirror configuration; usually white
- **Ambient reflection**
  - Constant accounted for global illumination (cheap hack)

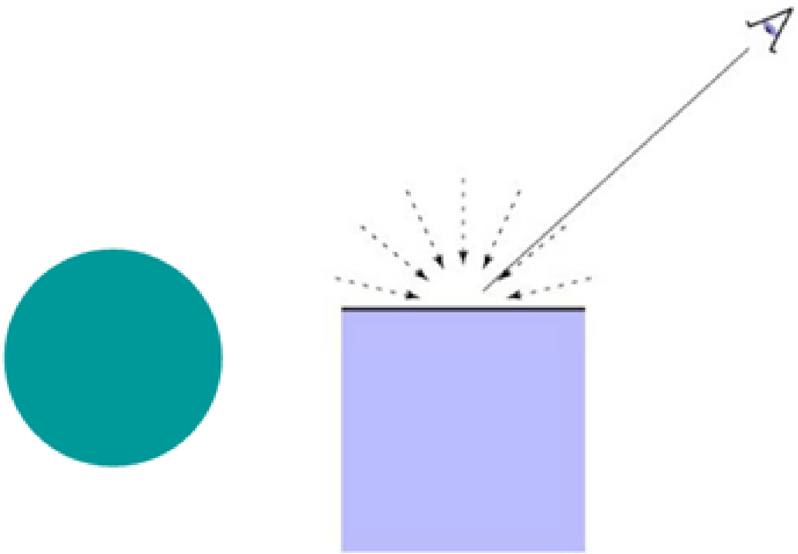


Ambient Shading

- Add constant color to account for disregarded illumination and fill black shadows

Ambient Shading (cont.)

- Add constant color to account for disregarded illumination and fill black shadows



the **intensity** of ambient light

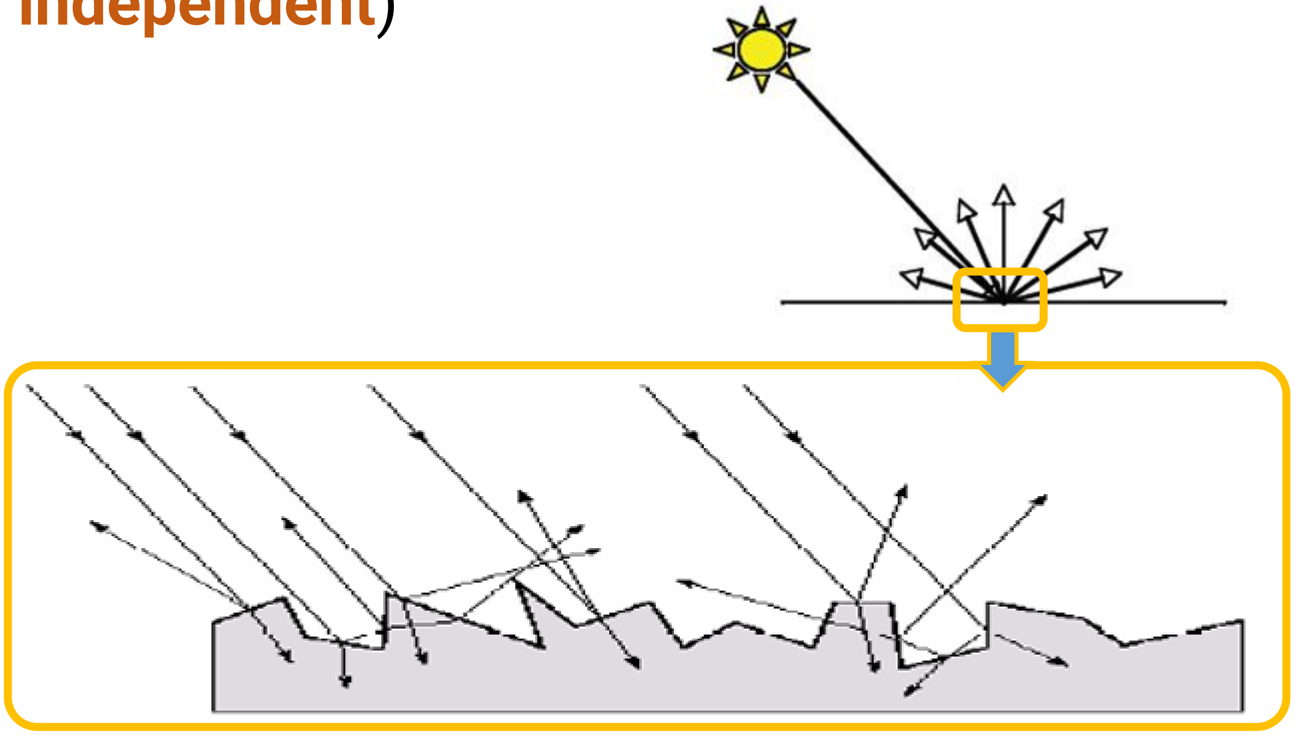
$$L_a = k_a \cdot I_a$$

ambient coefficient

reflected ambient light

Diffuse Shading

- Assume light reflects **equally in all directions**
  - The surface is rough with lots of tiny microfacets
- Therefore, the surface looks the same color from all views (**view independent**)

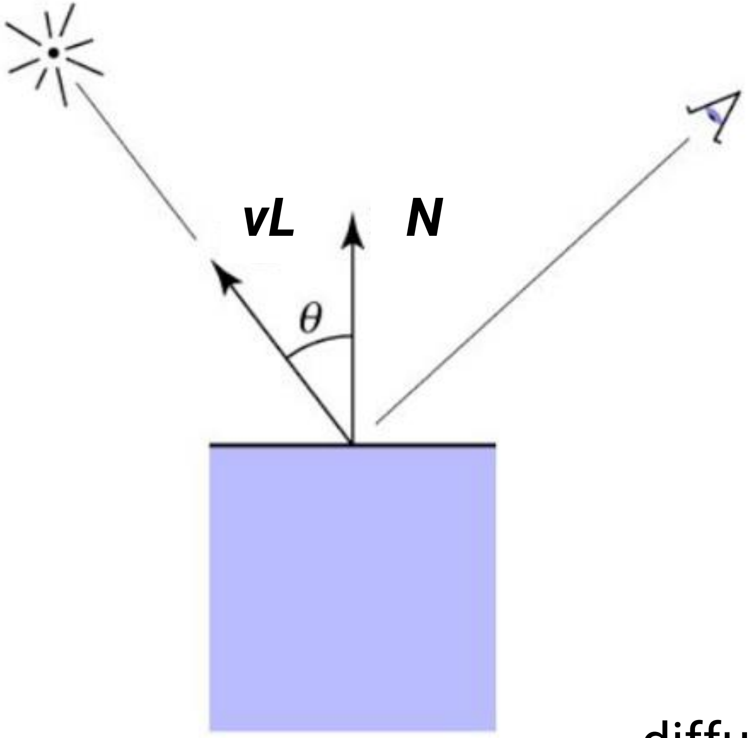


Diffuse Shading (cont.)

- Assume light reflects **equally in all directions**
  - The surface is rough with lots of tiny microfacets
- Therefore, the surface looks the same color from all views (**view independent**)

Diffuse Shading (cont.)

- Applies to diffuse or matte surface



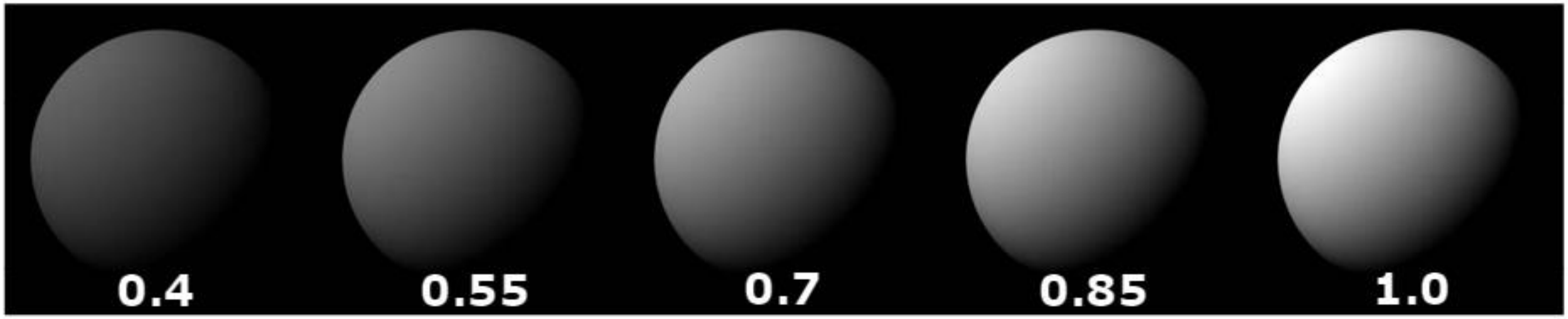
illumination from source

$$L_d = k_d \cdot I \cdot \underbrace{\max(0, N \cdot vL)}_{\text{Lambertian law}}$$

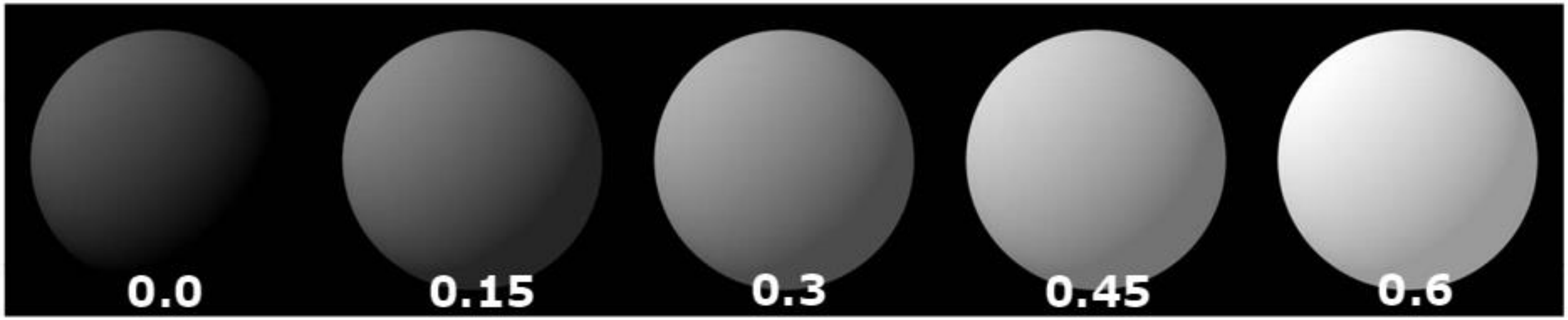
diffuse coefficient

diffusely reflected light

Diffuse Shading (cont.)



diffuse-reflection model with different k_d



ambient and diffuse-reflection model with different k_a

$$I_a = 1.0 \qquad k_d = 0.4$$

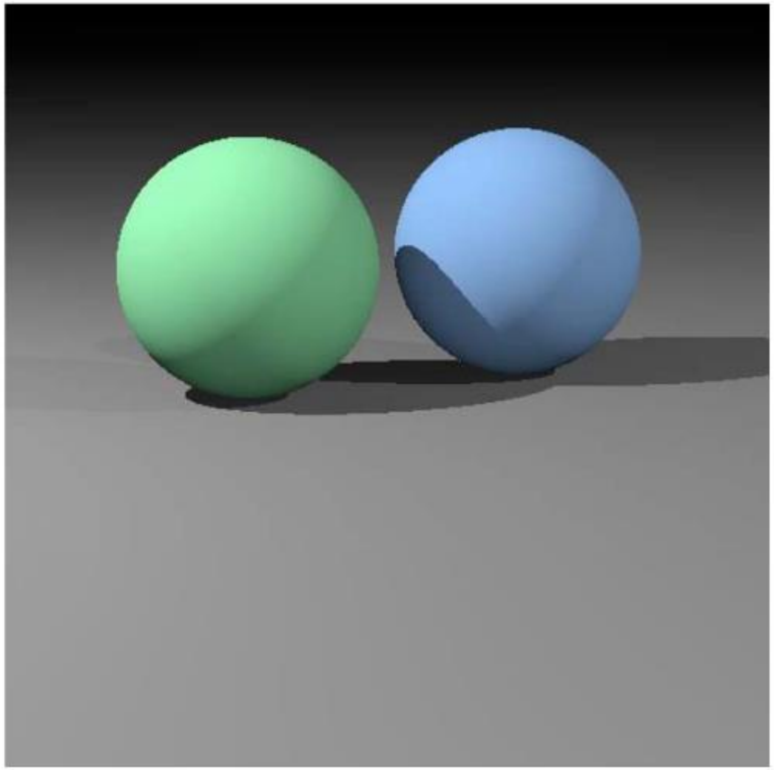
Diffuse Shading (cont.)

- For color objects, apply the formula for each color channel separately
- Light can also be non-white

Example:
white light: (0.9, 0.9, 0.9)
yellow light: (0.8, 0.8, 0.2)

$$L_d = \boxed{k_d} \cdot \boxed{I} \cdot \max(0, N \cdot vL)$$

Example:
green ball: (0.2, 0.7, 0.2)
blue ball: (0.2, 0.2, 0.7)

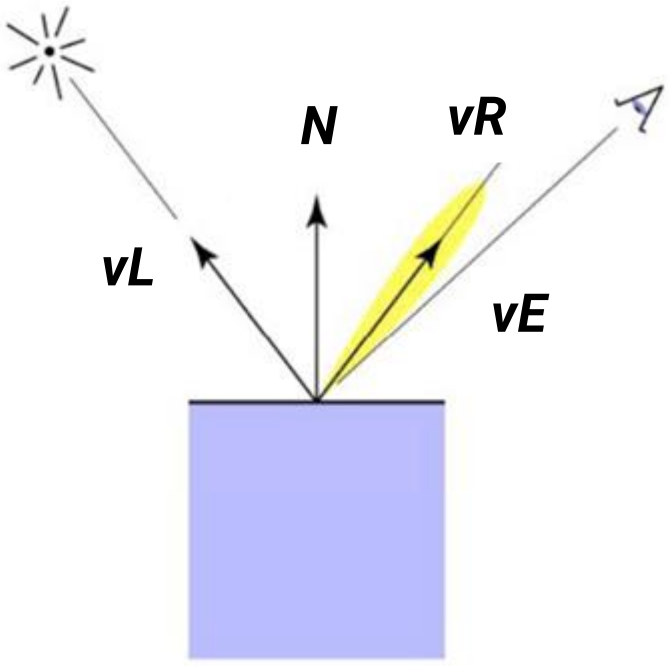


Specular Shading

- Some surfaces have highlights, mirror-like reflection
- **View direction dependent**
- Especially obvious for smooth shiny surfaces

Specular Shading (cont.)

- Phong specular model [1975]



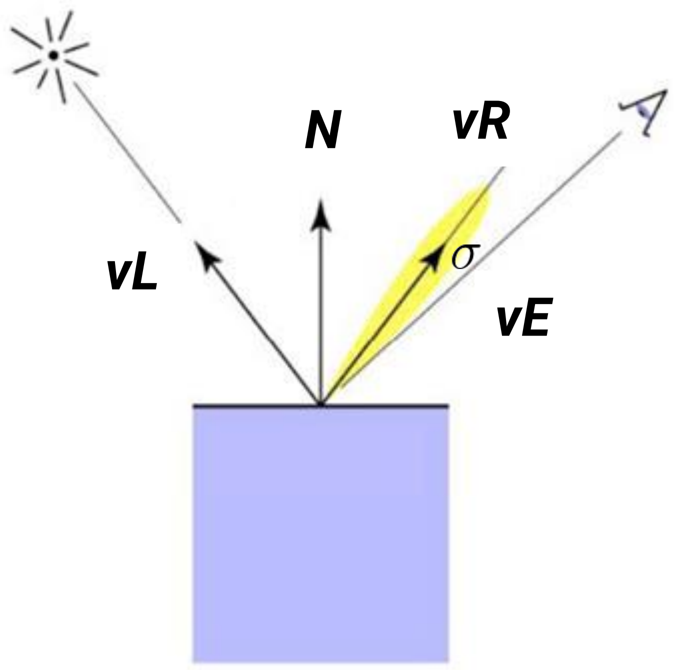
$$vR = vL + 2((N \cdot vL)N - vL)$$
$$= 2(N \cdot vL)N - vL$$

perfectly reflected direction

(you can find the proof here)

Specular Shading (cont.)

- **Phong specular model [1975]**
  - Fall off gradually from the perfect reflection direction



$$\begin{aligned} vR &= vL + 2((N \cdot vL)N - vL) \\ &= 2(N \cdot vL)N - vL \end{aligned}$$

specular exponent

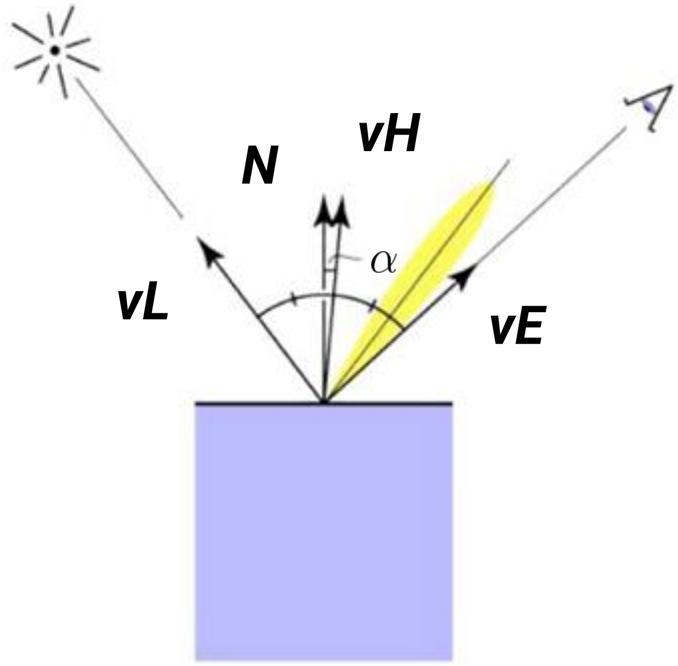
$$\begin{aligned} L_s &= k_s \cdot I \cdot \max(0, \cos\sigma)^{\boxed{n}} \\ &= k_s \cdot I \cdot \max(0, vE \cdot vR)^n \end{aligned}$$

specular coefficient

specularly reflected light

Phong specular Variant: Blinn-Phong

- Rather than computing reflection directly, just compare to normal bisection property
- One can prove $\cos^n(\sigma) = \cos^{4n}(\alpha)$



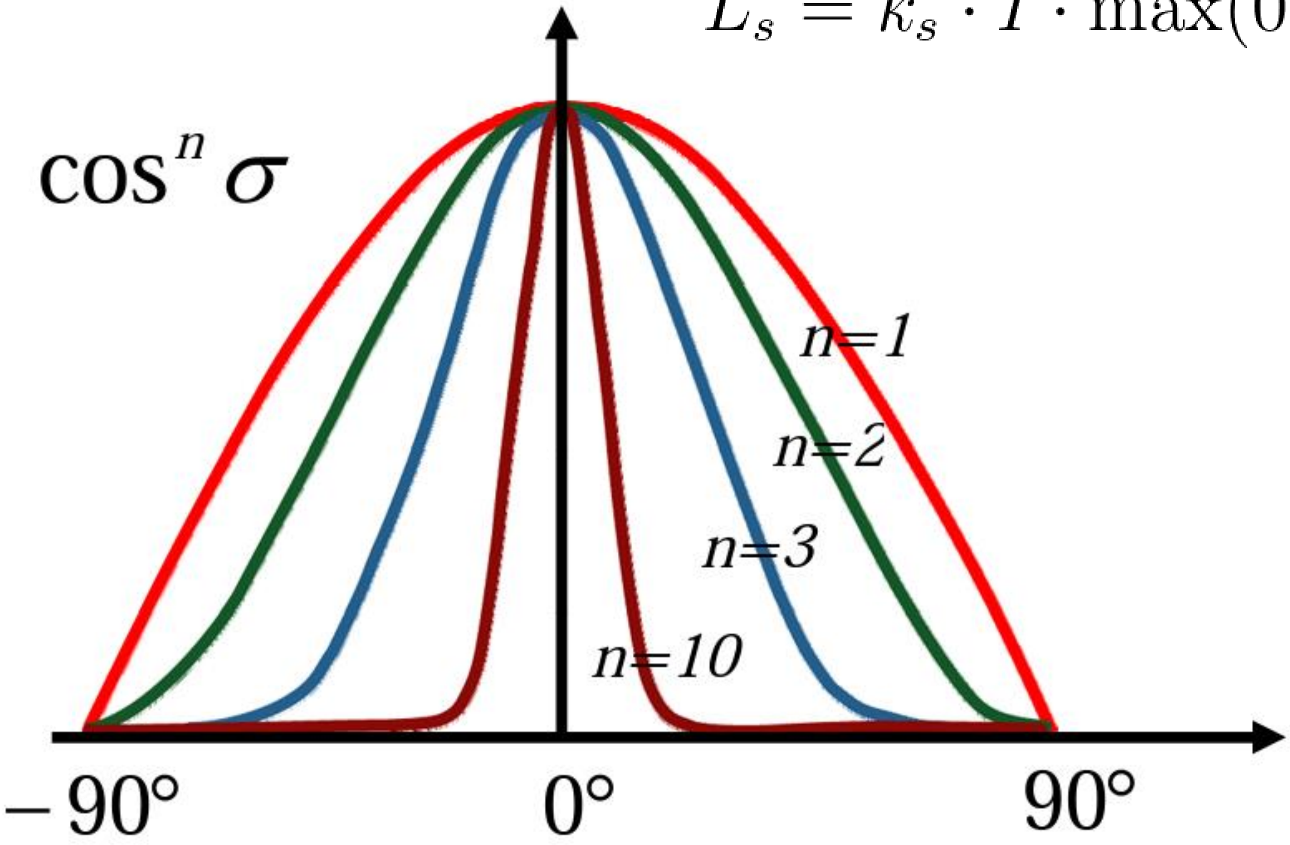
$$vH = \text{bisector}(vL, vE) = \frac{(vL + vE)}{\|vL + vE\|}$$

$$L_s = k_s \cdot I \cdot \max(0, \cos\alpha)^n = k_s \cdot I \cdot \max(0, N \cdot vH)^n$$

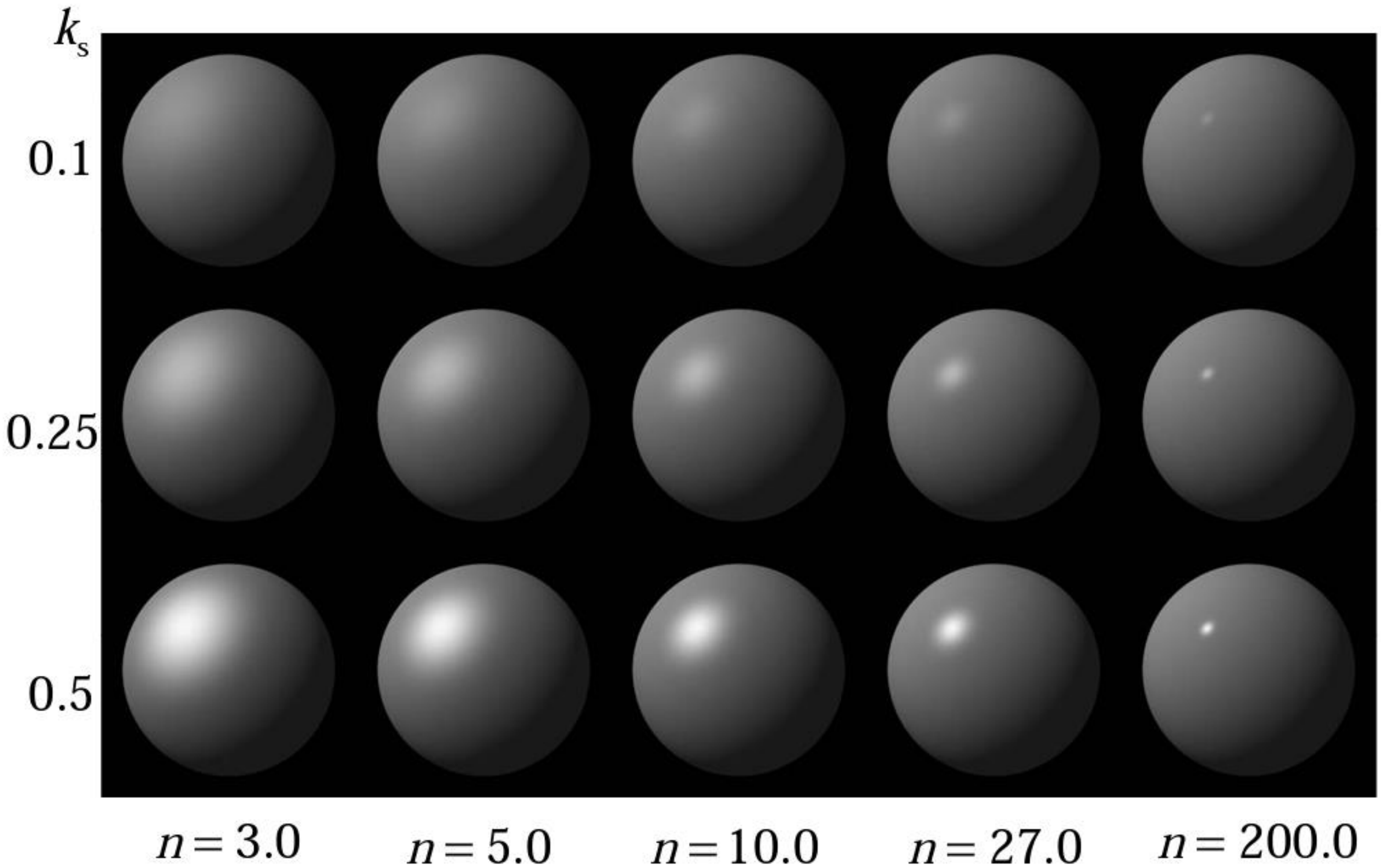
Specular Shading (cont.)

- Increase *n* narrows the lobe

$$L_s = k_s \cdot I \cdot \max(0, \cos\sigma)^{\boxed{n}}$$



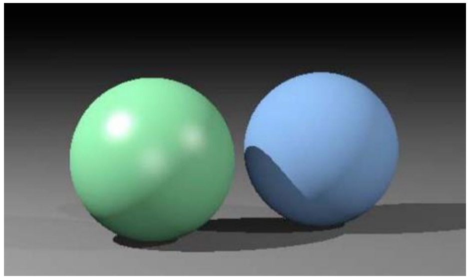
Specular Shading (cont.)



Complete Phong Lighting Model

- Compute the contribution from a light to a point by including **ambient**, **diffuse**, and **specular** components

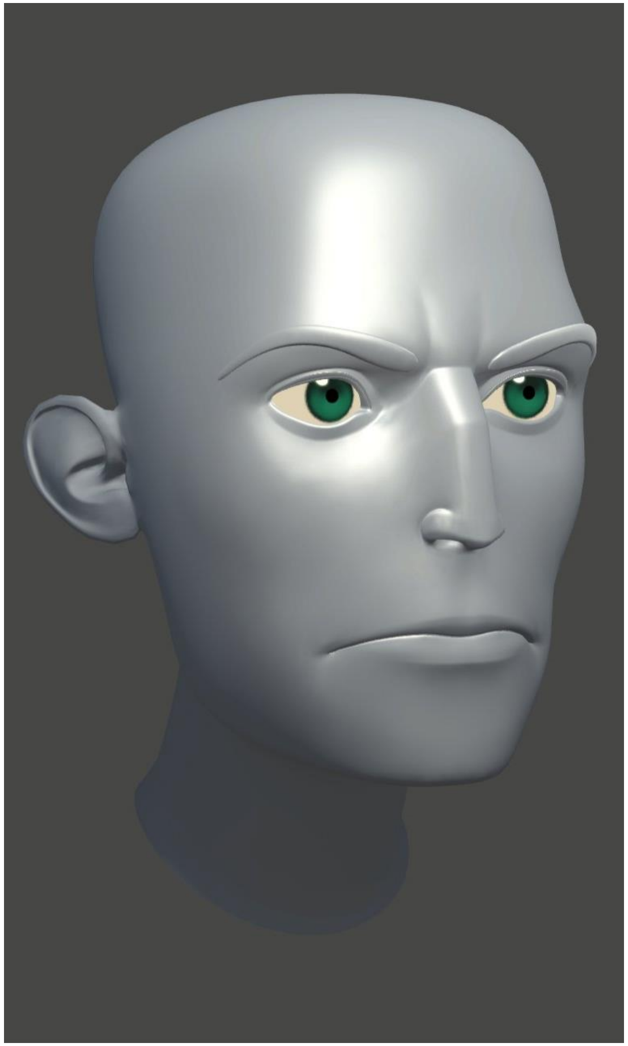
$$
\begin{aligned}
L &= L_a + L_d + L_s \\
&= k_a \cdot I_a + I(k_d \cdot \max(0, N \cdot vL) + k_s \cdot \max(0, N \cdot vH)^n)
\end{aligned}
$$



- If there are ***s*** lights, just sum over all the lights because the lighting is **linear**

$$L = k_a \cdot I_a + \sum_i^s (I_i(k_d \cdot \max(0, N \cdot vL_i) + k_s \cdot \max(0, N \cdot vH_i)^n))$$

Some Results with Phong Lighting Model



Material File Format

Material Template Library

- A MTL file defines the materials of a *.obj model

TexCube.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

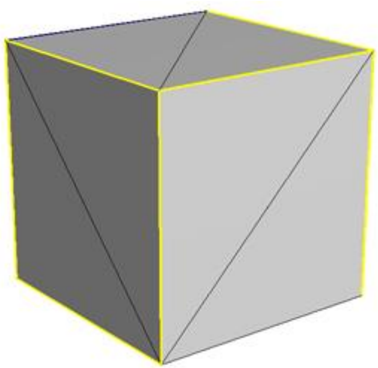
```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib TexCube.mtl
v 1.0 -1.0 -1.0
v 1.0 -1.0 1.0
v -1.0 -1.0 1.0
v -1.0 -1.0 -1.0
v 1.0 1.0 -1.0
v 1.0 1.0 1.0
v -1.0 1.0 1.0
v -1.0 1.0 -1.0

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

vn 0.0 -1.0 0.0
vn 0.0 1.0 0.0
vn 1.0 0.0 0.0
vn -0.0 0.0 1.0
vn -1.0 -0.0 -0.0
vn 0.0 0.0 -1.0
```

specify material file

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

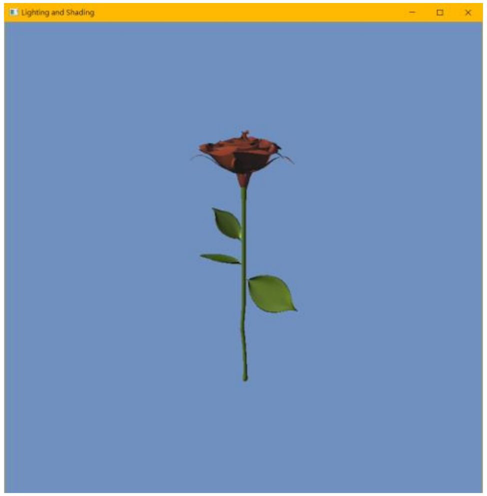
declare a new group (submesh) that uses the “cubeMtl” material

these faces use the “cubeMtl” material



Material Template Library (cont.)

- A model can have multiple groups (sub-meshes)
- The faces in the same group have the same material properties



Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
vn 0.0164 -0.9999 0.0000
usemtl phongE1
f 1/1/1 29/2/2 32/3/3 2/4/4
f 2/4/4 32/3/3 33/5/5 3/6/6
f 3/6/6 33/5/5 34/7/7 4/8/8
f 4/8/8 34/7/7 3344/9/9 3345/
f 29/2/2 30/11/11 35/12/12 32
```

第 253798 列，第 34 行 100% Unix (L

Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
vn 0.7047 0.0907 0.7036
vn 0.5859 0.0935 0.8050
vn 0.4528 0.0964 0.8864
usemtl phong1
f 79857/93559/80376 80519/935
f 80519/93560/80377 79858/935
f 80839/93561/80378 80520/935
```

第 337781 列，第 24 行 100% Unix (L

Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
usemtl phong2
f 81179/95085/81578 81529/95086/
f 81529/95086/81579 81180/95089/
f 81703/95087/81580 81530/95090/
f 81532/95088/81581 81703/95087/
f 81180/95089/81582 81533/95094/
f 81533/95094/81587 81181/95096/
```

第 341462 列，第 1 行 100% Unix (LF)

Material Template Library (cont.)

- The material template library (*.mtl) used by a Wavefront OBJ (*.obj) file describes material properties using
  - Phong lighting model (Ka, Kd, Ks, Ns)
  - Texture maps (mapKa, mapKd, mapKs, mapNs …)
  - Transparency (d, Tr, Ni)
  - … etc.

- You can refer to the wiki page for more information https://en.wikipedia.org/wiki/Wavefront_.obj_file

Material Template Library (cont.)

Rose.obj

Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
vn 0.7047 0.0907 0.7036
vn 0.5859 0.0935 0.8050
vn 0.4528 0.0964 0.8864
usemtl phong1
f 79857/93559/80376 80519/935
f 80519/93560/80377 79858/935
f 80839/93561/80378 80520/935
```

第 337781 列，第 24 行 100% Unix (L

Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
usemtl phong2
f 81179/95085/81578 81529/95086/
f 81529/95086/81579 81180/95089/
f 81703/95087/81580 81530/95090/
f 81532/95088/81581 81703/95087/
f 81180/95089/81582 81533/95094/
f 81533/95094/81587 81181/95096/
```

第 341462 列，第 1 行 100% Unix (LF)

Rose.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
vn 0.0164 -0.9999 0.0000
usemtl phongE1
f 1/1/1 29/2/2 32/3/3 2/4/4
f 2/4/4 32/3/3 33/5/5 3/6/6
f 3/6/6 33/5/5 34/7/7 4/8/8
f 4/8/8 34/7/7 3344/9/9 3345/
f 29/2/2 30/11/11 35/12/12 32
```

第 253798 列，第 34 行 100% Unix (L

Rose.mtl

Rose.mtl - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
# Blender MTL File: 'None'
# Material Count: 3

newmtl phong1
Ns 179.999996
Ka 0.500000 0.500000 0.500000
Kd 0.420000 0.620000 0.058000
Ks 0.500000 0.500000 0.500000

newmtl phong2
Ns 18.000005
Ka 0.149351 0.149351 0.149351
Kd 0.478000 0.651000 0.318000
Ks 0.500000 0.500000 0.500000

newmtl phongE1
Ns 179.999996
Ka 0.500000 0.500000 0.500000
Kd 0.700000 0.240000 0.240000
Ks 0.300000 0.300000 0.300000
```

第 1 列，第 1 行 100% Unix (LF)

Outline

- Overview
- Lights
- Materials
- **OpenGL implementation**

Overview

- The sample program ***Shading*** implements **phong lighting model** with a point light and a directional light in the **Vertex Shader**
- Introduce how to calculate **ambient** and **diffuse** lighting
  - Specular term is part of your HW #2

Files

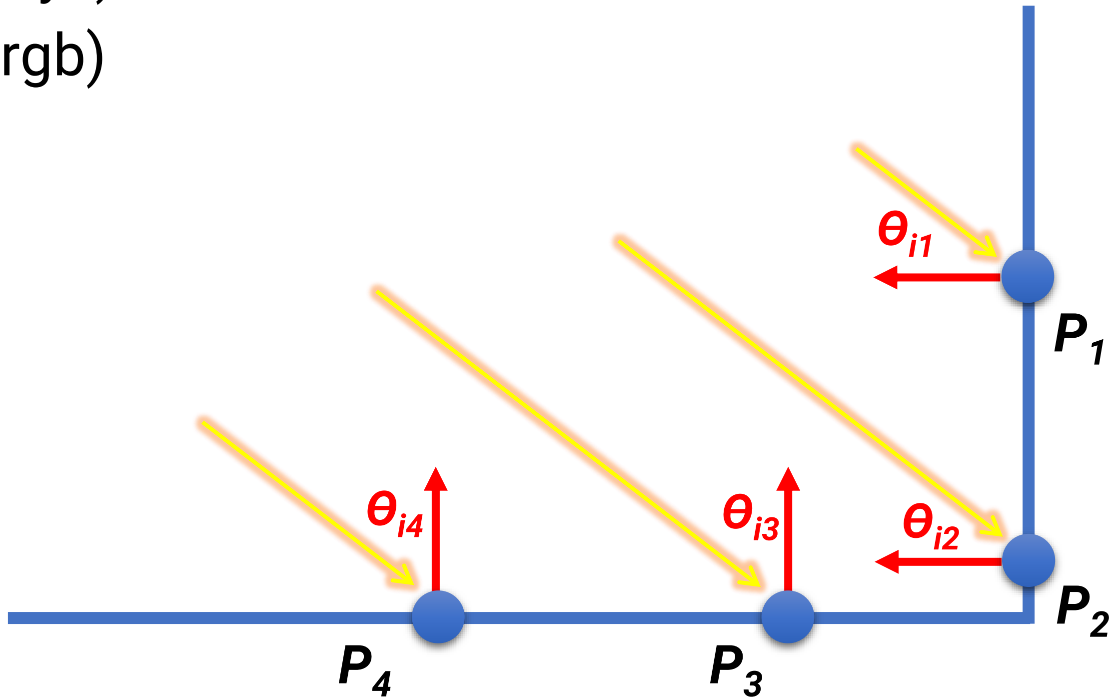
- **C/C++ files**
  - **Shading.cpp** main program (entry point)
  - header.h
  - sphere.h / sphere.cpp class for creating / rendering a sphere
  - camera.h / camera.cpp class for creating a virtual camera
  - **light.h** class for creating a point / directional light
  - **shaderprog.h / shaderprog.cpp** class for creating a shader
- **Shader files**
  - fixed_color.vs / fixed_color.fs
  - **gouraud_shading_demo.vs / gouraud_shading_demo.fs**

Data Structure: Lights

- Defined in ***light.h***
- Two types of lights implemented
    - Directional light
    - Point light

Recap: Directional Light

- Describes an emitter that deposits illumination from the **same direction** at every point in space
- Described by
  - Light direction (***D***, xyz)
  - Light radiance (***L***, rgb)



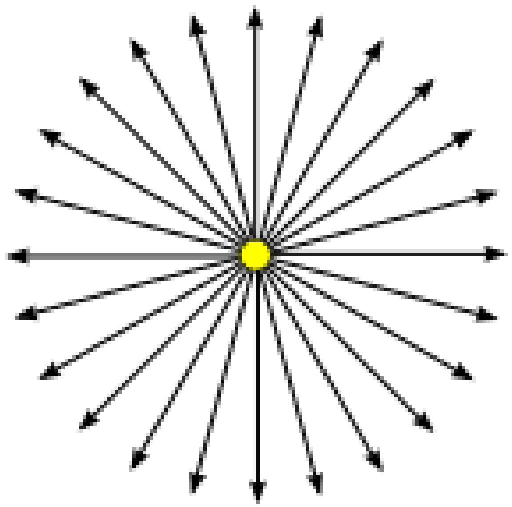
Data Structure: Directional Light

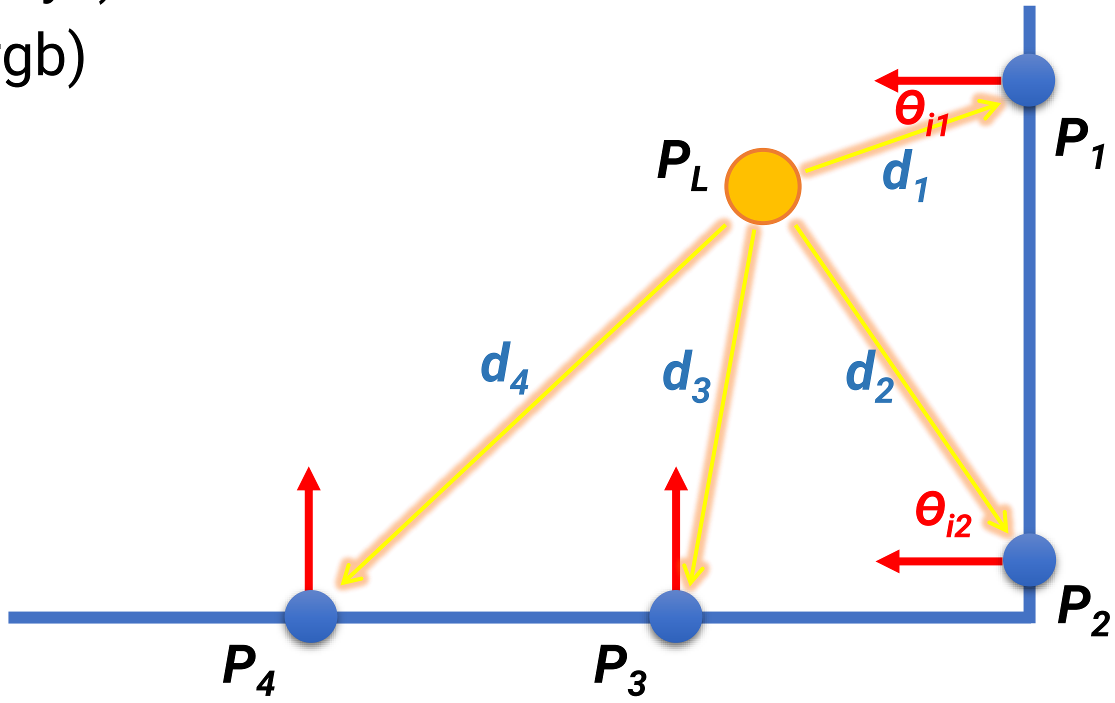
(world space)

```cpp
// DirectionalLight Declarations.
class DirectionalLight
{
public:
    // DirectionalLight Public Methods.
    DirectionalLight() {
        direction = glm::normalize(glm::vec3(0.0f, -1.0f, 0.0f));   // Default direction: coming from upward.
        radiance = glm::vec3(1.0f, 1.0f, 1.0f);                     // Default light color: white.
    };
    DirectionalLight(const glm::vec3 dir, const glm::vec3 L) {
        direction = glm::normalize(dir);
        radiance = L;
    }

    glm::vec3 GetDirection() const { return direction; }
    glm::vec3 GetRadiance()  const { return radiance;  }

private:
    // DirectionalLight Private Data.
    glm::vec3 direction;
    glm::vec3 radiance;
};
```

Recap: Point Light

- An isotropic point light source that emits the same amount of light in all directions
- Described by
  - Light position ($\boldsymbol{P_L}$, xyz)
  - Light intensity ($\boldsymbol{I}$, rgb)





Data Structure: Point Light

```
// PointLight Declarations.
class PointLight
{
public:
    // PointLight Public Methods.
    PointLight() {
        position = glm::vec3(0.0f, 0.0f, 0.0f);     // Default location.
        intensity = glm::vec3(1.0f, 1.0f, 1.0f);    // Default light color: white.
        CreateVisGeometry();
    }
    PointLight(const glm::vec3 p, const glm::vec3 I) {
        position = p;
        intensity = I;
        CreateVisGeometry();
    }

    glm::vec3 GetPosition() const  { return position;  }
    glm::vec3 GetIntensity() const { return intensity; }

    void Draw() {
        glPointSize(16.0f);
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, vboId);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexP), 0);
        glDrawArrays(GL_POINTS, 0, 1);
        glDisableVertexAttribArray(0);
        glPointSize(1.0f);
    }
```

(world space)

```
// VertexP Declarations.
struct VertexP
{
    VertexP() { position = glm::vec3(0.0f, 0.0f, 0.0f); }
    VertexP(glm::vec3 p) { position = p; }
    glm::vec3 position;
};
```

Data Structure: Point Light (cont.)

```
    void MoveLeft (const float moveSpeed) { position += moveSpeed * glm::vec3(-0.1f,  0.0f, 0.0f); }
    void MoveRight(const float moveSpeed) { position += moveSpeed * glm::vec3( 0.1f,  0.0f, 0.0f); }
    void MoveUp   (const float moveSpeed) { position += moveSpeed * glm::vec3( 0.0f,  0.1f, 0.0f); }
    void MoveDown (const float moveSpeed) { position += moveSpeed * glm::vec3( 0.0f, -0.1f, 0.0f); }

private:
    // PointLight Private Methods.
    void CreateVisGeometry() {
        VertexP lightVtx = glm::vec3(0, 0, 0);
        const int numVertex = 1;
        glGenBuffers(1, &vboId);
        glBindBuffer(GL_ARRAY_BUFFER, vboId);
        glBufferData(GL_ARRAY_BUFFER, sizeof(VertexP) * numVertex, &lightVtx, GL_STATIC_DRAW);
    }

    // PointLight Private Data.
    GLuint vboId;
    glm::vec3 position;
    glm::vec3 intensity;
};
```

create vertices in object space
(we will later transform it into world space)

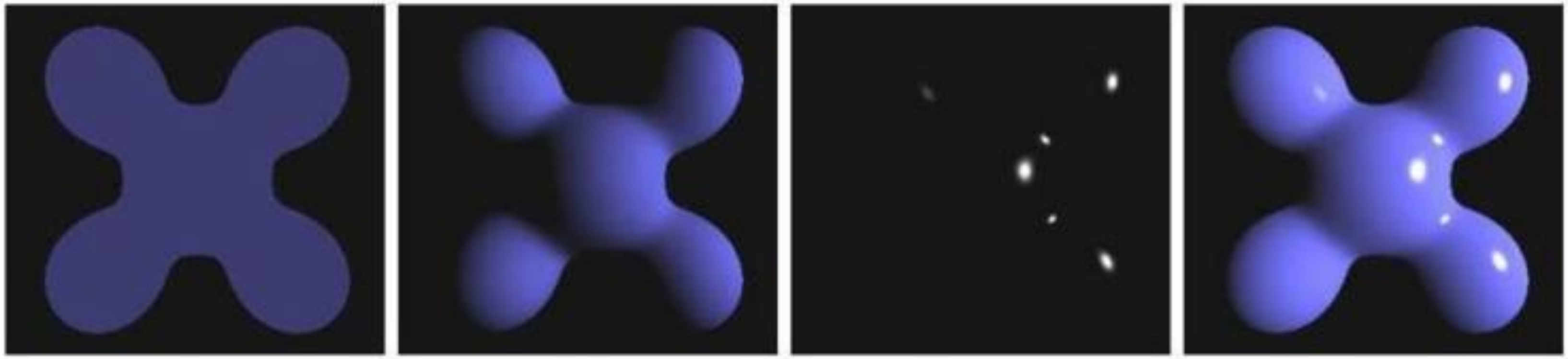
Data Structure: Scene Object

```cpp
// ScenePointLight (for visualization of a point light).
struct ScenePointLight
{
    ScenePointLight() {
        light = nullptr;
        worldMatrix = glm::mat4x4(1.0f);
        visColor = glm::vec3(1.0f, 1.0f, 1.0f);
    }
    PointLight* light;
    glm::mat4x4 worldMatrix;
    glm::vec3 visColor;
};
```

```cpp
// SceneObject.
struct SceneObject
{
    SceneObject() {
        mesh = nullptr;
        worldMatrix = glm::mat4x4(1.0f);
        Ka = glm::vec3(0.5f, 0.5f, 0.5f);
        Kd = glm::vec3(0.8f, 0.8f, 0.8f);
        Ks = glm::vec3(0.6f, 0.6f, 0.6f);
        Ns = 50.0f;
    }
    Sphere* mesh; simple sphere object, you can change to your triangle mesh
    glm::mat4x4 worldMatrix;
    // Material properties.
    glm::vec3 Ka;ambient coefficient
    glm::vec3 Kd;diffuse coefficient
    glm::vec3 Ks;specular coefficient
    float Ns; specular exponent (roughness)
};
SceneObject sceneObj;
```

Recap: Phong Lighting Model

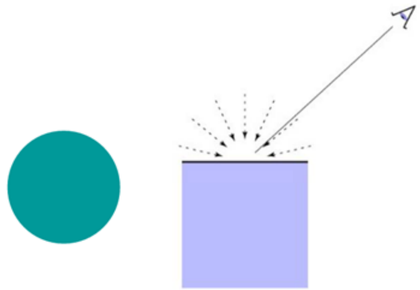
- **Diffuse reflection**
  - Light goes everywhere; colored by object color
- **Specular reflection**
  - Happens only near mirror configuration; usually white
- **Ambient reflection**
  - Constant accounted for global illumination (cheap hack)



ambient diffuse specular

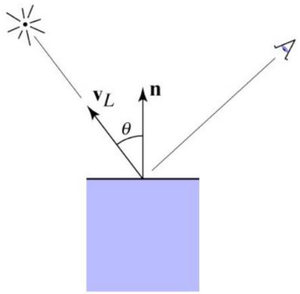
Recap: Phong Lighting Model

ambient



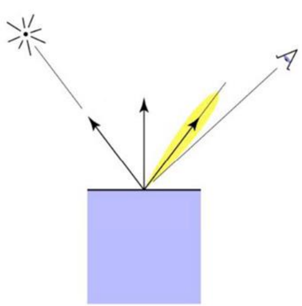
$$L_a = \boxed{k_a} \cdot I_a$$

diffuse



$$L_d = \boxed{k_d} \cdot I \cdot \max(0, N \cdot vL)$$

specular



$$\boxed{k_s} \cdot I \cdot \max(0, vE \cdot vR)^{\boxed{n}}$$

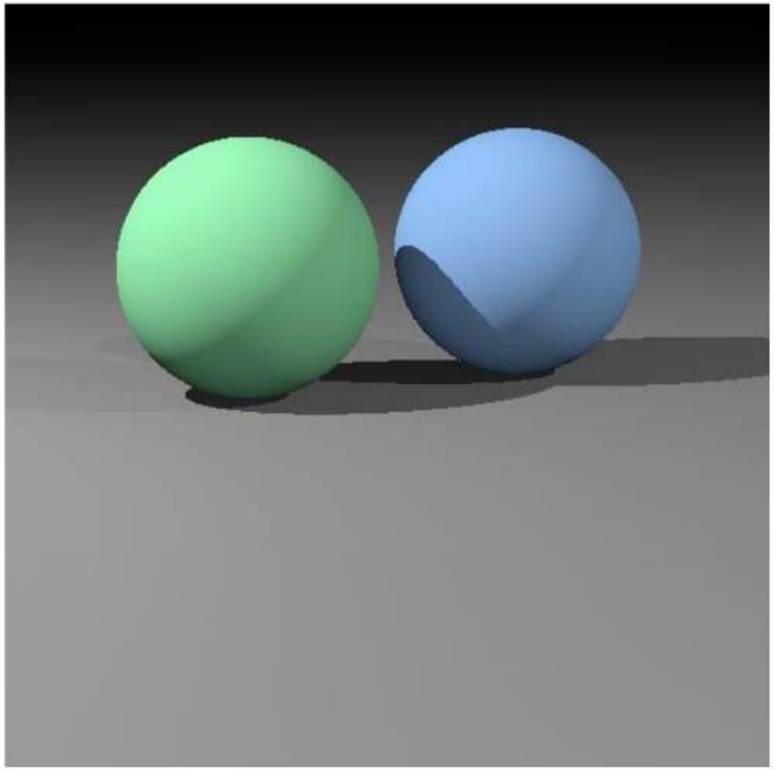
Recap: Lighting and Material Colors

- For color objects, apply the formula for each color channel separately
- Light can also be non-white

Example:
white light: (0.9, 0.9, 0.9)
yellow light: (0.8, 0.8, 0.2)

$$L_d = \boxed{k_d} \cdot \boxed{I} \cdot \max(0, N \cdot vL)$$

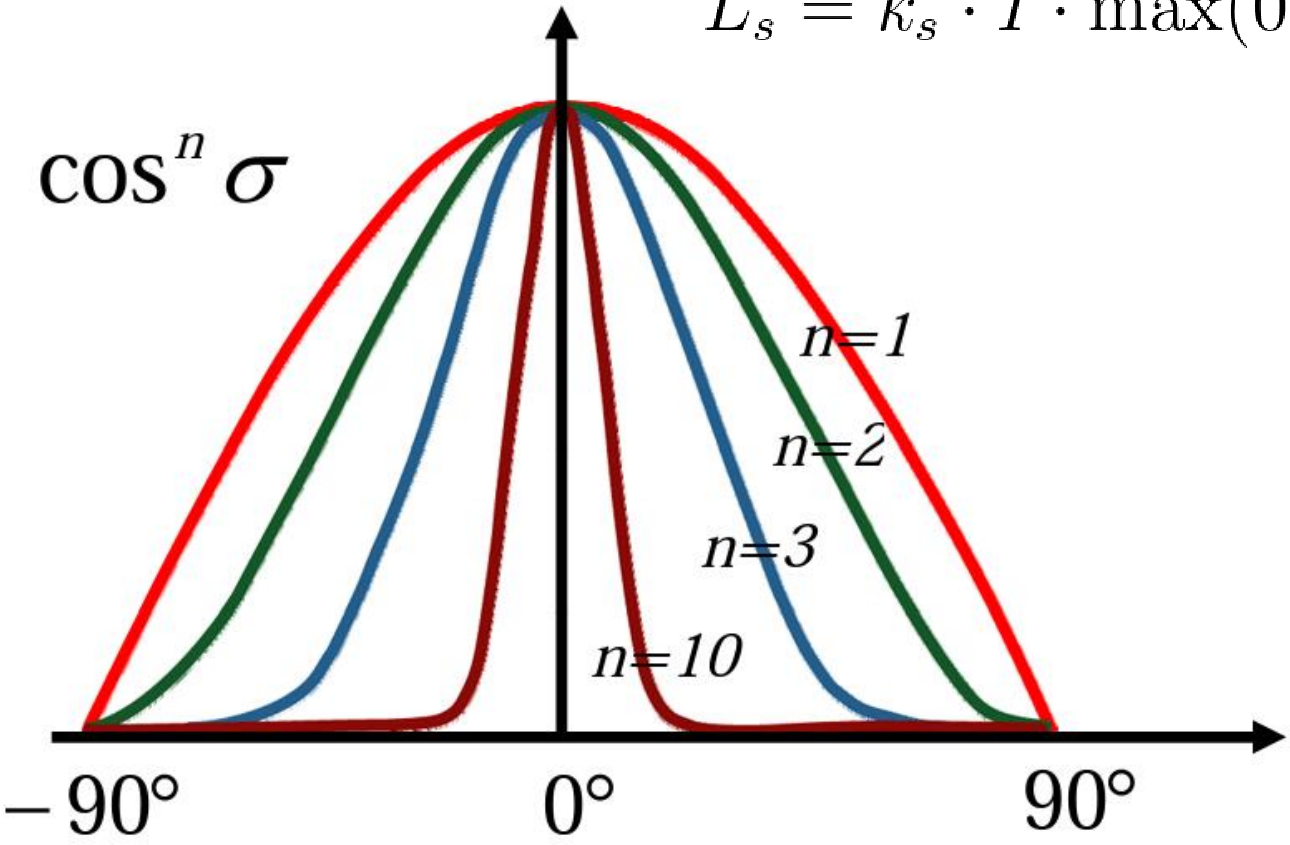
Example:
green ball: (0.2, 0.7, 0.2)
blue ball: (0.2, 0.2, 0.7)



Recap: Phong Lighting Model (cont.)

- Increase *n* narrows the lobe

$$L_s = k_s \cdot I \cdot \max(0, \cos\sigma)^{\boxed{n}}$$



Data Structure: Shaders

- Defined in ***shaderprog.h / shaderprog.cpp***
- Add **base class** ***"ShaderProg"***
- Add **inherited class** ***"FillColorShaderProg"***

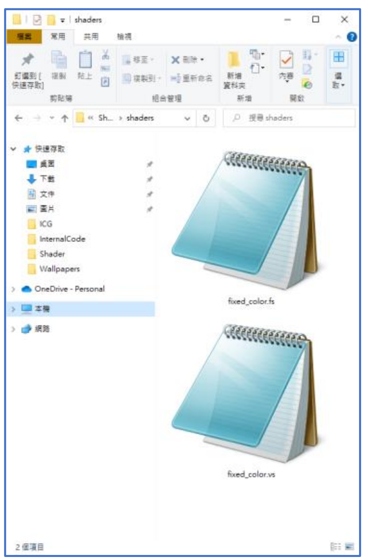
  Shader files:
  - Vertex shader: ***"fixed_color.vs"***
  - Fragment shader: ***"fixed_color.fs"***
- Add **inherited class** ***"GouraudShadingDemoShaderProg"***

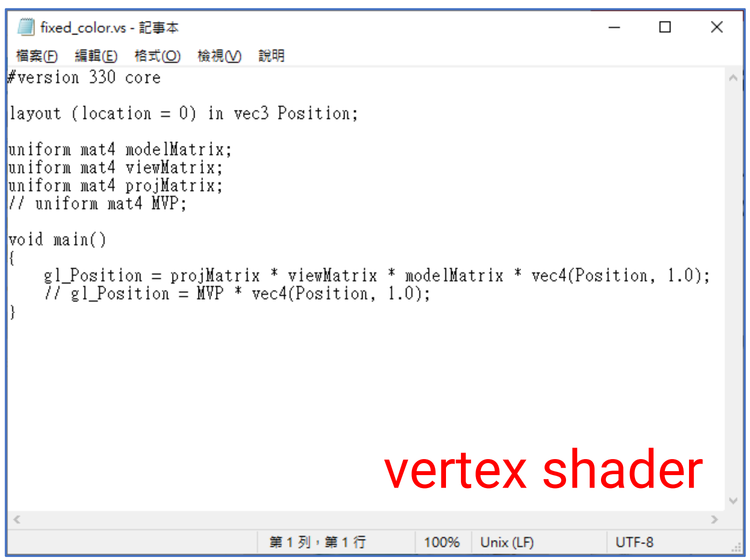
  Shaders files:
  - Vertex shader: ***"gouraud_shading_demo.vs"***
  - Fragment shader: ***"gouraud_shading_demo.fs"***

Recap: Shader

- Shaders: small C-like program that runs in a **per-vertex (Vertex Shader)** or **per-fragment (Fragment Shader)** manner **on the GPU in parallel**

the file extension does not matter!





Recap: Fill Color Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 Position;
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projMatrix;

void main() {
    gl_Position = projMatrix * viewMatrix *
                  modelMatrix * vec4(Position, 1.0);
}
```

Vertex attribute
- **glEnableVertexAttribArray(0)**

uniform variables communicated with the CPU
- Get location by **glGetUniformLocation**
- Set value by **glUniformXXX**

the main program **executed per vertex**

built-in variable for the Clip Space coordinate

Recap: Fill Color Fragment Shader

```
#version 330 core
uniform vec3 fillColor;

out vec4 FragColor;

void main() {
    FragColor = vec4(fillColor, 1.0);
}
```

uniform variables communicated with the CPU

- Get location by **glGetUniformLocation**
- Set value by **glUniformXXX**

Output: fragment data

the main program **executed per fragment**

Compute Lighting in Shader

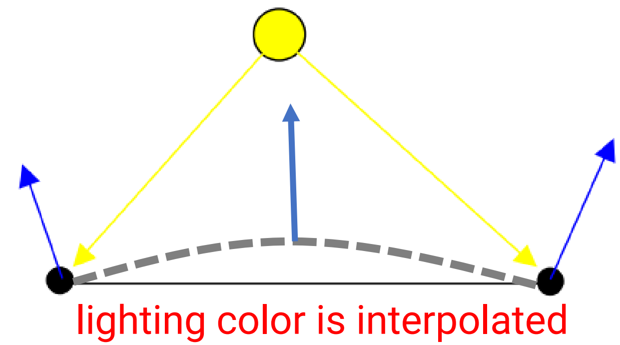
- Lighting and shading can be implemented either in the
  - **Vertex shader** **(Gouraud shading)**
    (compute per vertex and interpolate color)
    or
  - **Fragment shader** **(Phong shading)**
    (interpolate vertex attributes and compute per fragment)

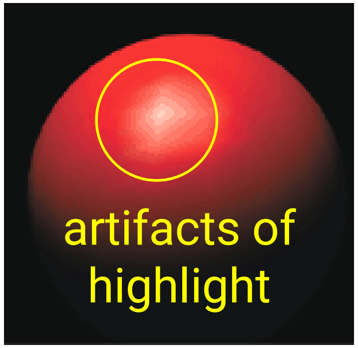
- It can also be implemented in **all coordinate spaces**, such as world space or camera space
  - Just remember that all objects should use the **SAME** coordinate space

Recap: Gouraud Shading and Phong Shading

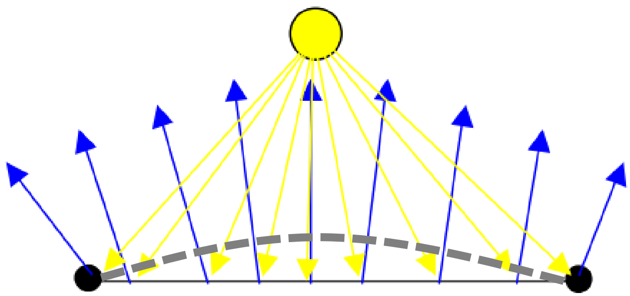
- **Gouraud shading**: compute lighting at vertices and interpolate the lighting color
- **Phong shading**: interpolate normal and compute lighting

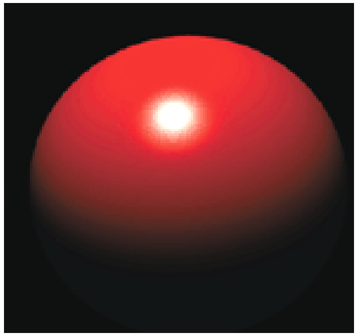
Gouraud shading





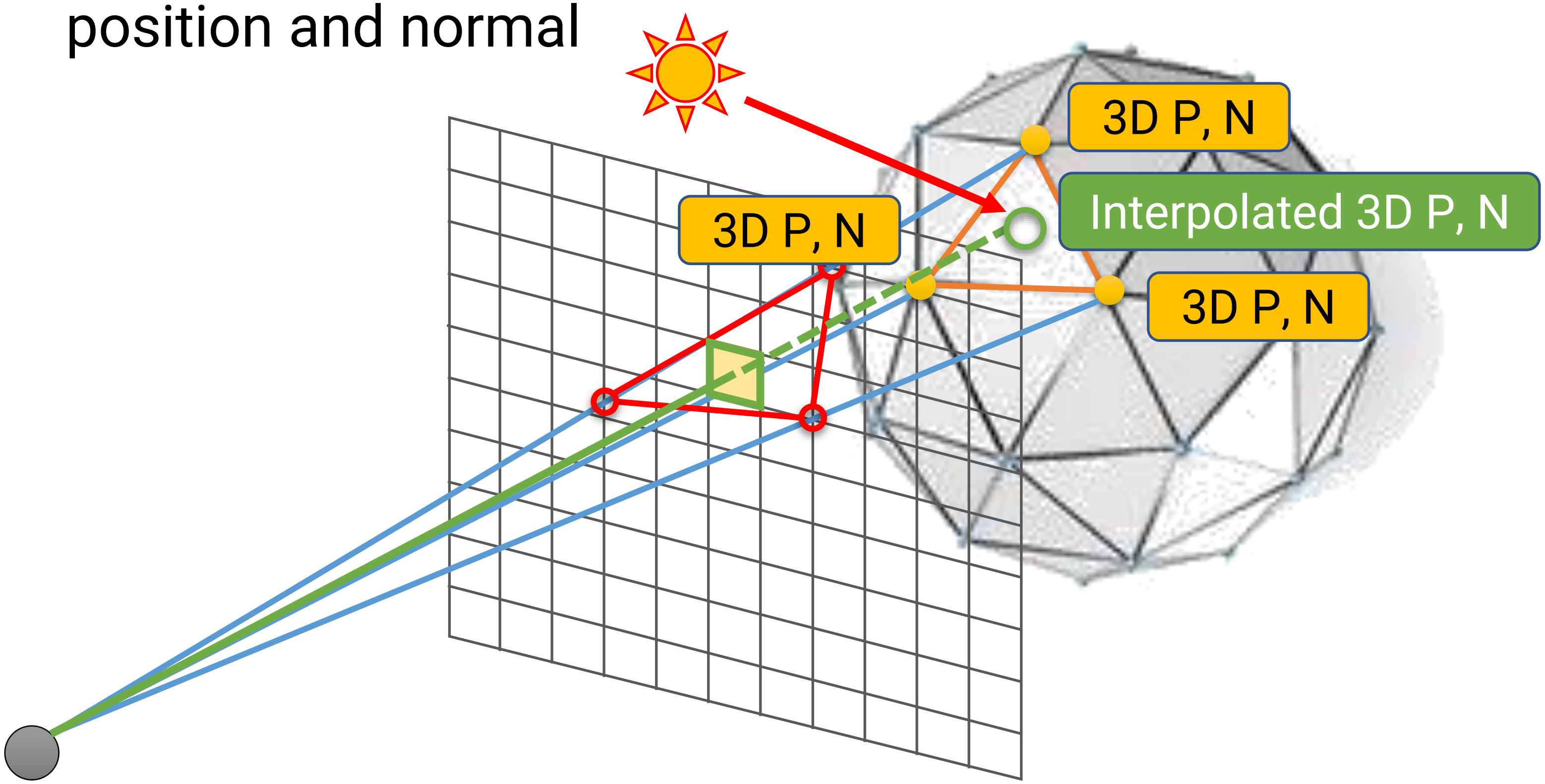
Phong shading





Recap: Vertex Attribute Interpolation

- **Interpolate geometry attributes**
  - Compute lighting at each fragment (in the fragment shader) requires per-fragment geometry attributes such as 3D position and normal



Recap: Vertex Attribute Interpolation (cont.)

- 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

world matrix for transforming normal

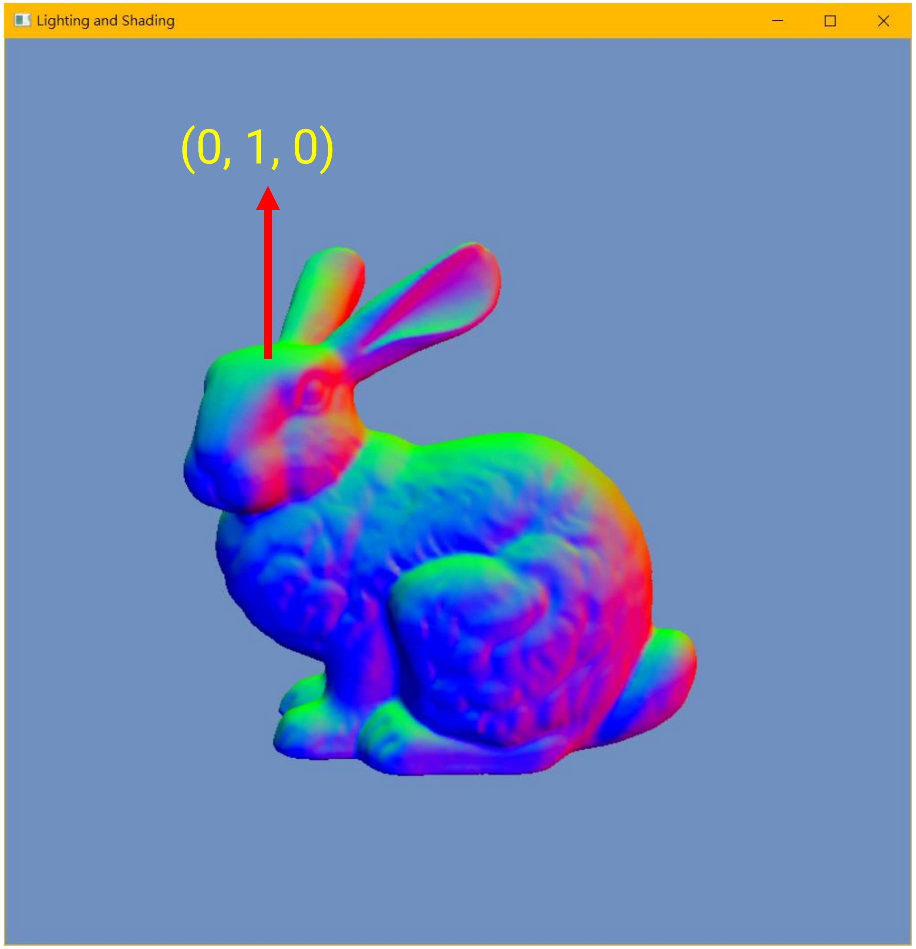
Fragment Shader

```
#version 330 core

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

out vec4 FragColor;


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

Ensure the interpolated normal has a unit length

Recap: Vertex Attribute Interpolation (cont.)

visualize world-space position as color



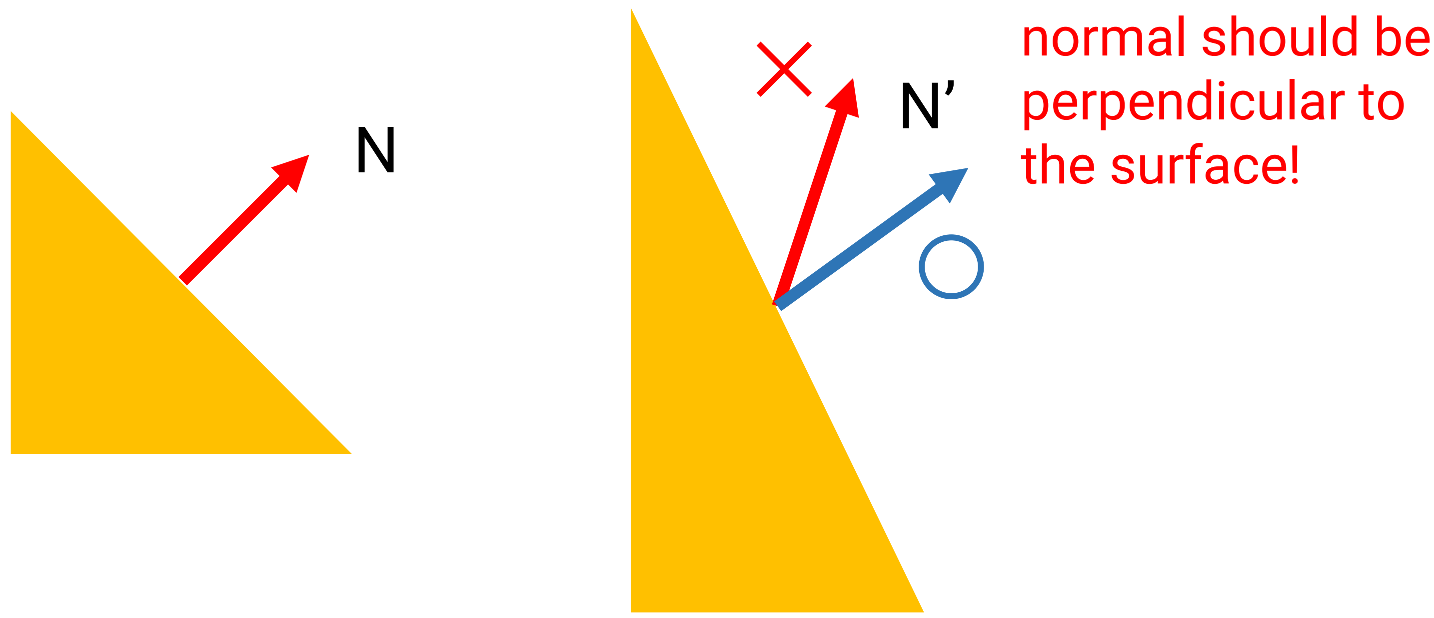
visualize world-space normal as color

Normal Matrix

- To transform a point from **Object Space** to **World Space**, we multiply its object-space position by the **world** (**model**) matrix
- How about the **vertex normal**?
  - We also need to transform the object-space normal to World Space for lighting computation
  - Could we also multiply the object-space normal by the world matrix?

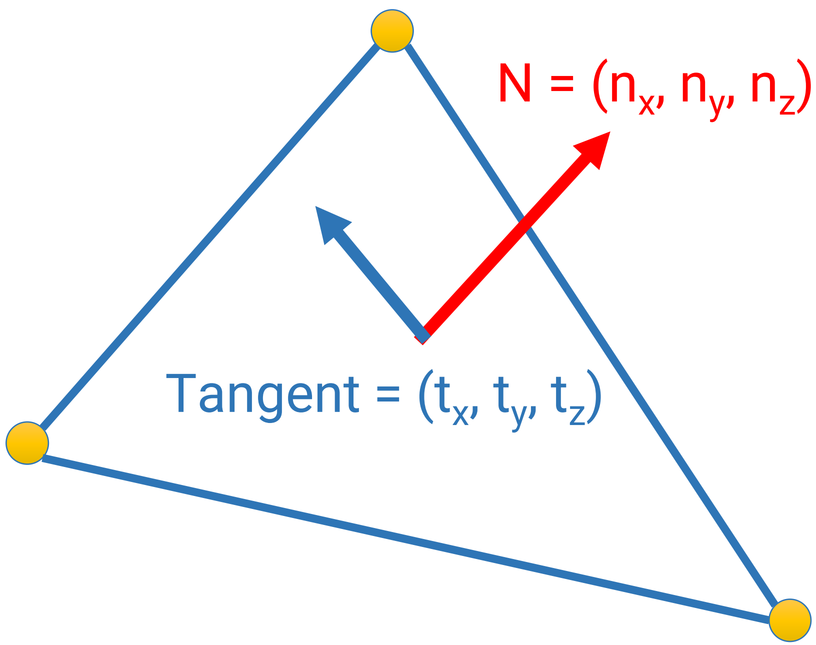
Normal Matrix (cont.)

- If the scaling in a world matrix is **uniform**, you can use the world matrix for transforming the normal directly
- However, if there is a **non-uniform** scaling, the matrix for transforming normal should be different



Normal Matrix (cont.)

- Derivation of the normal matrix



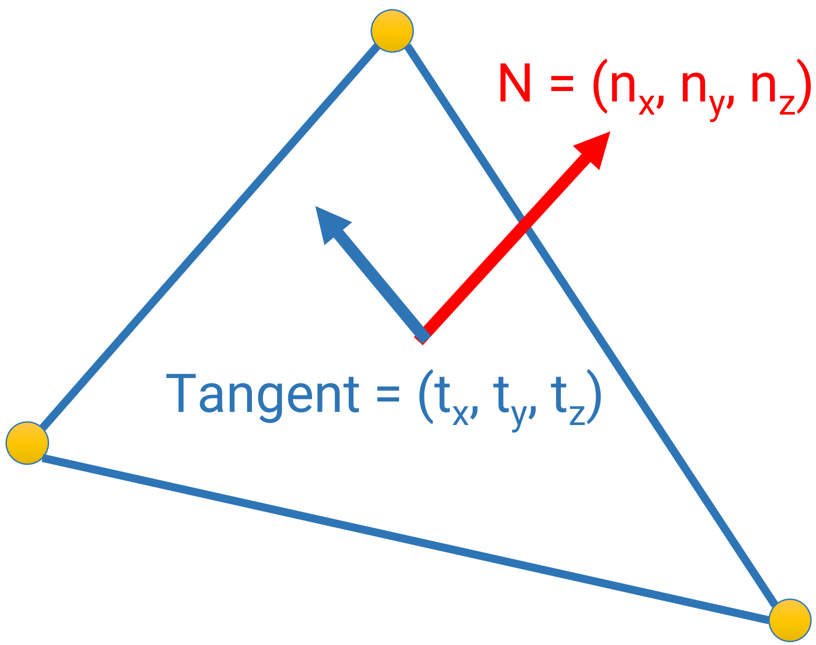
$$(n_x, n_y, n_z, 0) \cdot (t_x, t_y, t_z, 0) = 0$$

$$(n_x, n_y, n_z, 0) \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix} = 0$$

$$\underbrace{\boxed{(n_x, n_y, n_z, 0)\, M^{-1}}}_{\text{transform normal}} \underbrace{\boxed{M \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix}}}_{\text{transform vertex}} = 0$$

Normal Matrix (cont.)

- Derivation of the normal matrix



Note: if you want to compute lighting in **Camera Space**, the ***M*** should be the **modelview** matrix

$$\begin{pmatrix} n_x^{world} \\ n_y^{world} \\ n_z^{world} \\ 0 \end{pmatrix}^T = (n_x, n_y, n_z, 0)M^{-1}$$

$$(AB)^T = B^T A^T$$

$$\begin{pmatrix} n_x^{world} \\ n_y^{world} \\ n_z^{world} \\ 0 \end{pmatrix} = \boxed{(M^{-1})^T} \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix}$$

normal matrix
(the inverse transpose of world matrix)

Gouraud Shading 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;

(cont.)
```

Vertex attribute

- **glEnableVertexAttribArray(1)**

(you can refer to sphere.cpp)

Gouraud Shading Vertex Shader (cont.)

```
// Material properties.
uniform vec3 Ka;
uniform vec3 Kd;
uniform vec3 Ks;
uniform float Ns;
// Light data
uniform vec3 ambientLight;
uniform vec3 dirLightDir;
uniform vec3 dirLightRadiance;
uniform vec3 pointLightPos;
uniform vec3 pointLightIntensity;
```

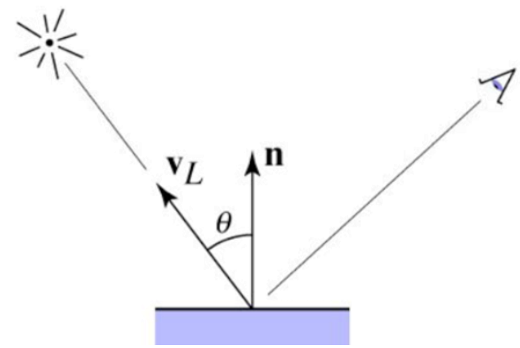
(cont.)

Gouraud Shading Vertex Shader (cont.)

```
// Data pass to fragment shader
out vec3 iLightingColor;

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

    // Compute vertex lighting in view space.
    vec4 tmpPos = viewMatrix * worldMatrix * vec4(Position, 1.0);
    vec3 vsPosition = tmpPos.xyz / tmpPos.w;
    vec3 vsNormal = (normalMatrix * vec4(Normal, 0.0)).xyz;
    vsNormal = normalize(vsNormal);
```

(cont.)

Gouraud Shading Vertex Shader (cont.)




```
// Ambient light.
vec3 ambient = Ka * ambientLight;
// ------------------------------------------------------
// Directional light.
vec3 vsLightDir = (viewMatrix * vec4(-dirLightDir, 0.0)).xyz;
vsLightDir = normalize(vsLightDir);
// Diffuse and Specular.
vec3 diffuse =
        Diffuse(Kd, dirLightRadiance, vsNormal, vsLightDir);
vec3 specular = Specular();
vec3 dirLight = diffuse + specular;
```

(cont.)

Gouraud Shading Vertex Shader (cont.)

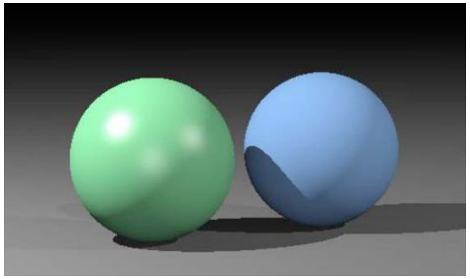
```
// Point light.
tmpPos = viewMatrix * vec4(pointLightPos, 1.0);
vec3 vsLightPos = tmpPos.xyz / tmpPos.w;
vsLightDir = normalize(vsLightPos - vsPosition);
float distSurfaceToLight = distance(vsLightPos, vsPosition);
float attenuation = 1.0f / (distSurfaceToLight * distSurfaceToLight);
vec3 radiance = pointLightIntensity * attenuation;
// Diffuse and Specular.
diffuse = Diffuse(Kd, radiance, vsNormal, vsLightDir);
specular = Specular();
vec3 pointLight = diffuse + specular;
```

(cont.)

Recap: Multiple Lights

- Compute the contribution from a light to a point by including **ambient**, **diffuse**, and **specular** components

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a \cdot I_a + I(k_d \cdot \max(0, N \cdot vL) + k_s \cdot \max(0, N \cdot vH)^n) \end{aligned}$$



- If there are ***s*** lights, just sum over all the lights because the lighting is **linear**

$$L = k_a \cdot I_a + \sum_i^s (I_i(k_d \cdot \max(0, N \cdot vL_i) + k_s \cdot \max(0, N \cdot vH_i)^n))$$

Gouraud Shading Vertex Shader (cont.)

```
    // Put all lights together.
    iLightingColor = ambient + dirLight + pointLight;
}

vec3 Diffuse(vec3 Kd, vec3 I, vec3 N, vec3 lightDir) {
    return Kd * I * max(0, dot(N, lightDir));
}
vec3 Specular( /* Put the parameters here. */ ) {
    // Try to implement yourself!
    return vec3(0.0, 0.0, 0.0);
}
```

Gouraud Shading Fragment Shader

```
#version 330 core

in vec3 iLightingColor;  (has been interpolated)

out vec4 FragColor;


void main()
{
    FragColor = vec4(iLightingColor, 1.0);
}
```

Recap: Setting Parameters to Shaders

```
locMVP = glGetUniformLocation(shaderProgId, "MVP");
glUniformMatrix4fv(locMVP, 1, GL_FALSE, glm::value_ptr(MVP));
```

CPU

2

1

Vertex Shader

GPU

```
#version 330 core
layout (location = 0) in vec3 Position;
uniform mat4 MVP;
void main() {
    gl_Position = MVP * vec4(Position, 1.0);
}
```

Data Structure: Shaders (cont.)

- Base class for creating a shader program

```
// ShaderProg Declarations.
class ShaderProg
{
public:
    // ShaderProg Public Methods.
    ShaderProg();
    ~ShaderProg();

    bool LoadFromFiles(const std::string vsFilePath, const std::string fsFilePath);
    void Bind() { glUseProgram(shaderProgId); };
    void UnBind() { glUseProgram(0); };

    GLint GetLocMVP() const { return locMVP; }
```

call private methods, **LoadShaderTextFromFile** and **AddShader**

all shaders need this

(cont.)

Data Structure: Shaders (cont.)

- Base class for creating a shader program

(cont.)

```
protected:
    // ShaderProg Protected Methods.
    virtual void GetUniformVariableLocation();

    // ShaderProg Protected Data.
    GLuint shaderProgId;

private:
    // ShaderProg Private Methods.
    GLuint AddShader(const std::string& sourceText, GLenum shaderType);
    static bool LoadShaderTextFromFile(const std::string filePath, std::string& sourceText);

    // ShaderProg Private Data.
    GLint locMVP;
};
```

each shader has different parameters, so make it **virtual for overriding**

Data Structure: Shaders

- Inherited class for Gouraud Shading

```
// GouraudShadingDemoShaderProg Declarations.
class GouraudShadingDemoShaderProg : public ShaderProg
{
public:
    // GouraudShadingDemoShaderProg Public Methods.
    GouraudShadingDemoShaderProg();
    ~GouraudShadingDemoShaderProg();

    GLint GetLocM() const { return locM; }
    GLint GetLocV() const { return locV; }
    GLint GetLocNM() const { return locNM; }
    GLint GetLocKa() const { return locKa; }
    GLint GetLocKd() const { return locKd; }
    GLint GetLocKs() const { return locKs; }
    GLint GetLocNs() const { return locNs; }
    GLint GetLocAmbientLight() const { return locAmbientLight; }
    GLint GetLocDirLightDir() const { return locDirLightDir; }
    GLint GetLocDirLightRadiance() const { return locDirLightRadiance; }
    GLint GetLocPointLightPos() const { return locPointLightPos; }
    GLint GetLocPointLightIntensity() const { return locPointLightIntensity; }
```

locations of uniform matrix variables

locations of uniform material variables

locations of uniform light data variables

Data Structure: Shaders (cont.)

```
protected:
    // GouraudShadingDemoShaderProg Protected Methods.
    void GetUniformVariableLocation();

private:
    // GouraudShadingDemoShaderProg Public Data.
    // Transformation matrix.
    GLint locM;
    GLint locV;
    GLint locNM;
    // Material properties.
    GLint locKa;
    GLint locKd;
    GLint locKs;
    GLint locNs;
    // Light data.
    GLint locAmbientLight;
    GLint locDirLightDir;
    GLint locDirLightRadiance;
    GLint locPointLightPos;
    GLint locPointLightIntensity;
};
```

override from the base class

Data Structure: Shaders (cont.)

- Inherited class for Gouraud Shading

```
GouraudShadingDemoShaderProg::GouraudShadingDemoShaderProg()
{
    locM = -1;
    locV = -1;
    locNM = -1;
    locKa = -1;
    locKd = -1;
    locKs = -1;
    locNs = -1;
    locAmbientLight = -1;
    locDirLightDir = -1;
    locDirLightRadiance = -1;
    locPointLightPos = -1;
    locPointLightIntensity = -1;
}

GouraudShadingDemoShaderProg::~GouraudShadingDemoShaderProg()
{}
```

Data Structure: Shaders (cont.)

- Inherited class for Gouraud Shading

```
void GouraudShadingDemoShaderProg::GetUniformVariableLocation()
{
    ShaderProg::GetUniformVariableLocation();
    locM = glGetUniformLocation(shaderProgId, "worldMatrix");
    locV = glGetUniformLocation(shaderProgId, "viewMatrix");
    locNM = glGetUniformLocation(shaderProgId, "normalMatrix");
    locKa = glGetUniformLocation(shaderProgId, "Ka");
    locKd = glGetUniformLocation(shaderProgId, "Kd");
    locKs = glGetUniformLocation(shaderProgId, "Ks");
    locNs = glGetUniformLocation(shaderProgId, "Ns");
    locAmbientLight = glGetUniformLocation(shaderProgId, "ambientLight");
    locDirLightDir = glGetUniformLocation(shaderProgId, "dirLightDir");
    locDirLightRadiance = glGetUniformLocation(shaderProgId, "dirLightRadiance");
    locPointLightPos = glGetUniformLocation(shaderProgId, "pointLightPos");
    locPointLightIntensity = glGetUniformLocation(shaderProgId, "pointLightIntensity");
}
```

Main Program

- The flow of the main program remains the same

```
int main(int argc, char** argv)
{
    // Setting window properties.
    Initialize window properties and GLEW

    // Initialization.
    SetupRenderState();
    SetupScene();
    CreateShaderLib();

    // Register callback functions.
    Register callback functions

    // Start rendering loop.
    glutMainLoop();

    return 0;
}
```

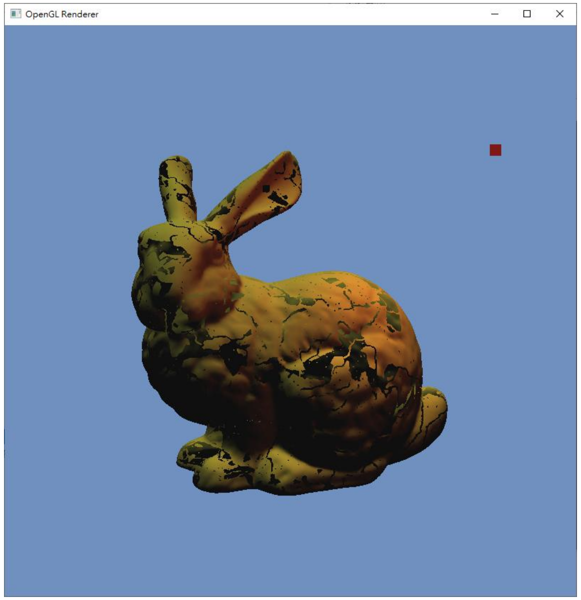
Main Program (cont.)

- Remember to enable “**depth test**” by calling

  glEnable(**GL_DEPTH_TEST**);

  Otherwise, the Z-buffer will not work

```
void SetupRenderState()
{
    // glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glEnable(GL_DEPTH_TEST);

    glm::vec4 clearColor = glm::vec4(0.44f, 0.57f, 0.75f, 1.00f);
    glClearColor(
        (GLclampf)(clearColor.r),
        (GLclampf)(clearColor.g),
        (GLclampf)(clearColor.b),
        (GLclampf)(clearColor.a)
    );
}
```



Main Program (cont.)

```
void SetupScene()
{
    // Scene object ---------------------------------------------------------------
    sphereMesh = new Sphere(32, 32, 0.5f);
    sceneObj.mesh = sphereMesh;

    // Scene lights ---------------------------------------------------------------
    // Create a directional light.
    dirLight = new DirectionalLight(dirLightDirection, dirLightRadiance);
    // Create a point light.
    pointLight = new PointLight(pointLightPosition, pointLightIntensity);
    pointLightObj.light = pointLight;
    pointLightObj.visColor = glm::normalize(((PointLight*)pointLightObj.light)->GetIntensity());

    // Create a camera and update view and proj matrices.
    camera = new Camera((float)screenWidth / (float)screenHeight);
    camera->UpdateView(cameraPos, cameraTarget, cameraUp);
    float aspectRatio = (float)screenWidth / (float)screenHeight;
    camera->UpdateProjection(fovy, aspectRatio, zNear, zFar);
}
```

Main Program (cont.)

```cpp
void CreateShaderLib()
{
    fillColorShader = new FillColorShaderProg();
    if (!fillColorShader->LoadFromFiles("shaders/fixed_color.vs", "shaders/fixed_color.fs"))
        exit(1);

    gouraudShadingShader = new GouraudShadingDemoShaderProg();
    if (!gouraudShadingShader->LoadFromFiles("shaders/gouraud_shading_demo.vs", "shaders/gouraud_shading_demo.fs"))
        exit(1);
}
```

render the object using “GouraudShadingShader” with object transform, object material, and lighting parameters

```cpp
void RenderSceneCB()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render a triangle mesh with Gouraud shading. ----------------------------------------------
    if (sceneObj.mesh ≠ nullptr) {
        // Update transform (assuming there might be dynamic transformations).
        glm::mat4x4 S = glm::scale(glm::mat4x4(1.0f), glm::vec3(1.5f, 1.5f, 1.5f));
        sceneObj.worldMatrix = S;
        glm::mat4x4 normalMatrix = glm::transpose(glm::inverse(camera->GetViewMatrix() * sceneObj.worldMatrix));
        glm::mat4x4 MVP = camera->GetProjMatrix() * camera->GetViewMatrix() * sceneObj.worldMatrix;
```

Main Program (cont.)

```
    gouraudShadingShader->Bind();

    // Transformation matrix.
    glUniformMatrix4fv(gouraudShadingShader->GetLocM(), 1, GL_FALSE, glm::value_ptr(sceneObj.worldMatrix));
    glUniformMatrix4fv(gouraudShadingShader->GetLocV(), 1, GL_FALSE, glm::value_ptr(camera->GetViewMatrix()));
    glUniformMatrix4fv(gouraudShadingShader->GetLocNM(), 1, GL_FALSE, glm::value_ptr(normalMatrix));
    glUniformMatrix4fv(gouraudShadingShader->GetLocMVP(), 1, GL_FALSE, glm::value_ptr(MVP));
    // Material properties.
    glUniform3fv(gouraudShadingShader->GetLocKa(), 1, glm::value_ptr(sceneObj.Ka));
    glUniform3fv(gouraudShadingShader->GetLocKd(), 1, glm::value_ptr(sceneObj.Kd));
    glUniform3fv(gouraudShadingShader->GetLocKs(), 1, glm::value_ptr(sceneObj.Ks));
    glUniform1f(gouraudShadingShader->GetLocNs(), sceneObj.Ns);
    // Light data.
    if (dirLight ≠ nullptr) {
        glUniform3fv(gouraudShadingShader->GetLocDirLightDir(), 1, glm::value_ptr(dirLight->GetDirection()));
        glUniform3fv(gouraudShadingShader->GetLocDirLightRadiance(), 1, glm::value_ptr(dirLight->GetRadiance()));
    }
    if (pointLight ≠ nullptr) {
        glUniform3fv(gouraudShadingShader->GetLocPointLightPos(), 1, glm::value_ptr(pointLight->GetPosition()));
        glUniform3fv(gouraudShadingShader->GetLocPointLightIntensity(), 1, glm::value_ptr(pointLight->GetIntensity()));
    }
    glUniform3fv(gouraudShadingShader->GetLocAmbientLight(), 1, glm::value_ptr(ambientLight));

    // Render the mesh.
    sceneObj.mesh->Render();

    gouraudShadingShader->UnBind();
}
```

Main Program (cont.)

```
    // Visualize the light with fill color. ------------------------------------------------
    // Bind shader and set parameters.
    PointLight* pointLight = pointLightObj.light;
    if (pointLight ≠ nullptr) {
        glm::mat4x4 T = glm::translate(glm::mat4x4(1.0f), (pointLight->GetPosition()));
        pointLightObj.worldMatrix = T;
        glm::mat4x4 MVP = camera->GetProjMatrix() * camera->GetViewMatrix() * pointLightObj.worldMatrix;

        fillColorShader->Bind();

        glUniformMatrix4fv(fillColorShader->GetLocMVP(), 1, GL_FALSE, glm::value_ptr(MVP));
        glUniform3fv(fillColorShader->GetLocFillColor(), 1, glm::value_ptr(pointLightObj.visColor));

        // Render the point light.
        pointLight->Draw();

        fillColorShader->UnBind();
    }
    // -------------------------------------------------------------------------------------

    glutSwapBuffers();
}
```

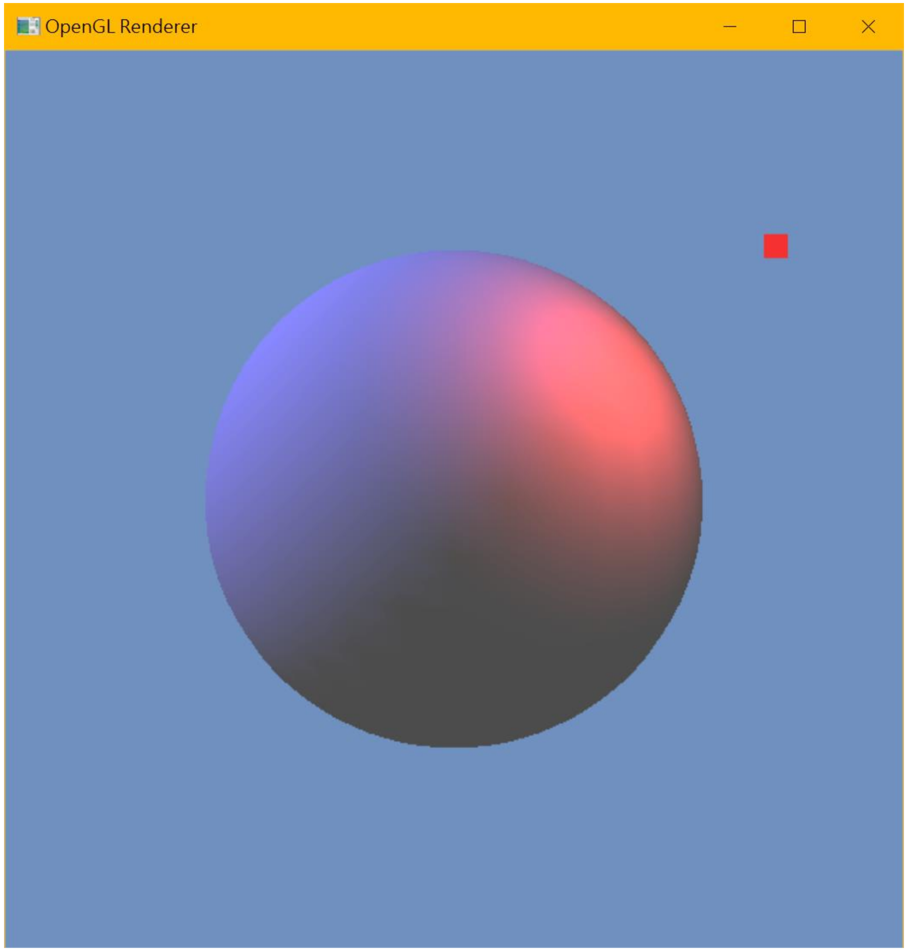
render the point light using “FillColorShader”

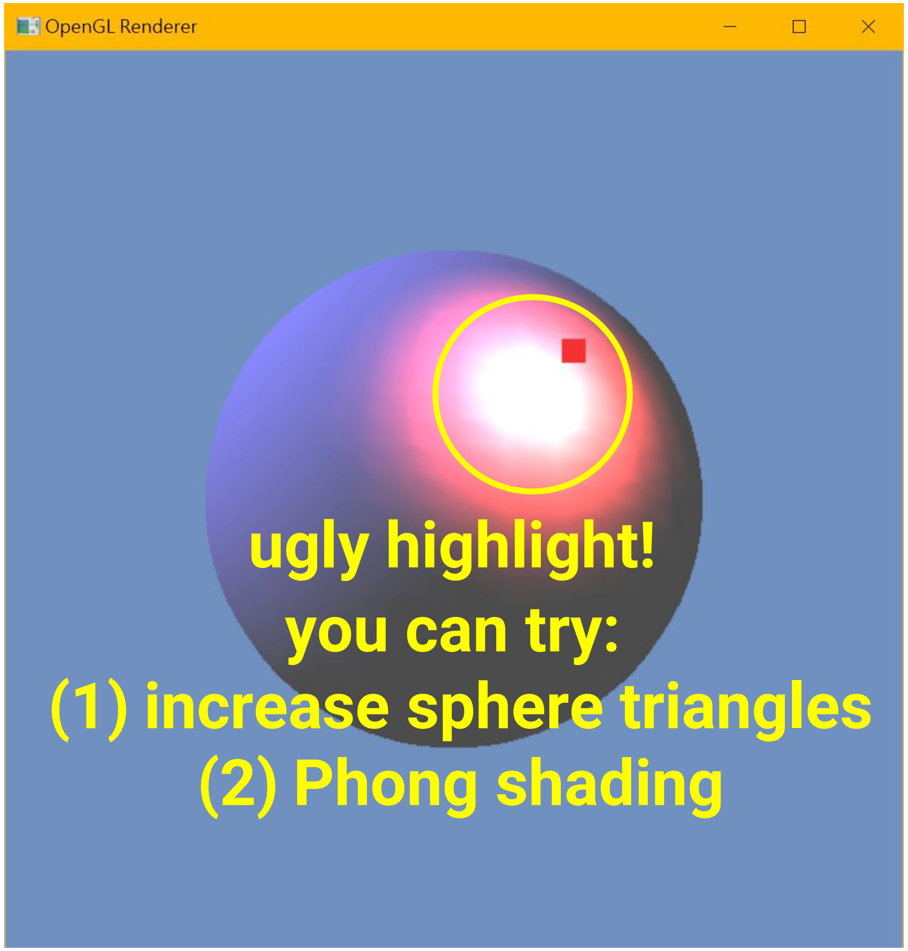
Main Program (cont.)

```
void ProcessSpecialKeysCB(int key, int x, int y)
{
    // Handle special (functional) keyboard inputs such as F1, spacebar, page up, etc.
    switch (key) {
    // Rendering mode.

    // Light control.
    case GLUT_KEY_LEFT:
        if (pointLight != nullptr)
            pointLight->MoveLeft(lightMoveSpeed);
        break;
    case GLUT_KEY_RIGHT:
        if (pointLight != nullptr)
            pointLight->MoveRight(lightMoveSpeed);
        break;
    case GLUT_KEY_UP:
        if (pointLight != nullptr)
            pointLight->MoveUp(lightMoveSpeed);
        break;
    case GLUT_KEY_DOWN:
        if (pointLight != nullptr)
            pointLight->MoveDown(lightMoveSpeed);
        break;
    default:
        break;
    }
}
```

interactively control the point light with the keyboard

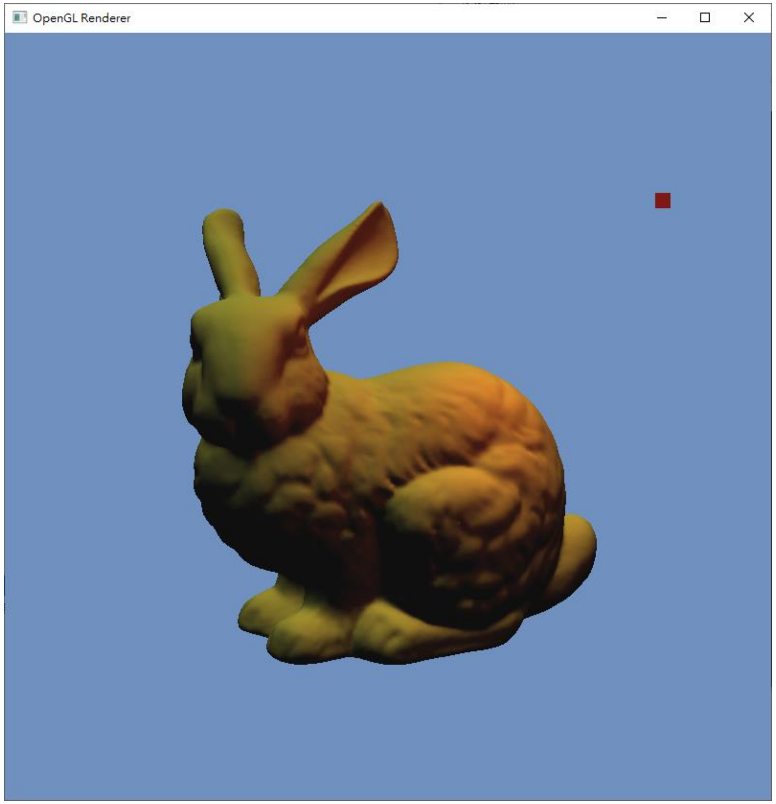
Results

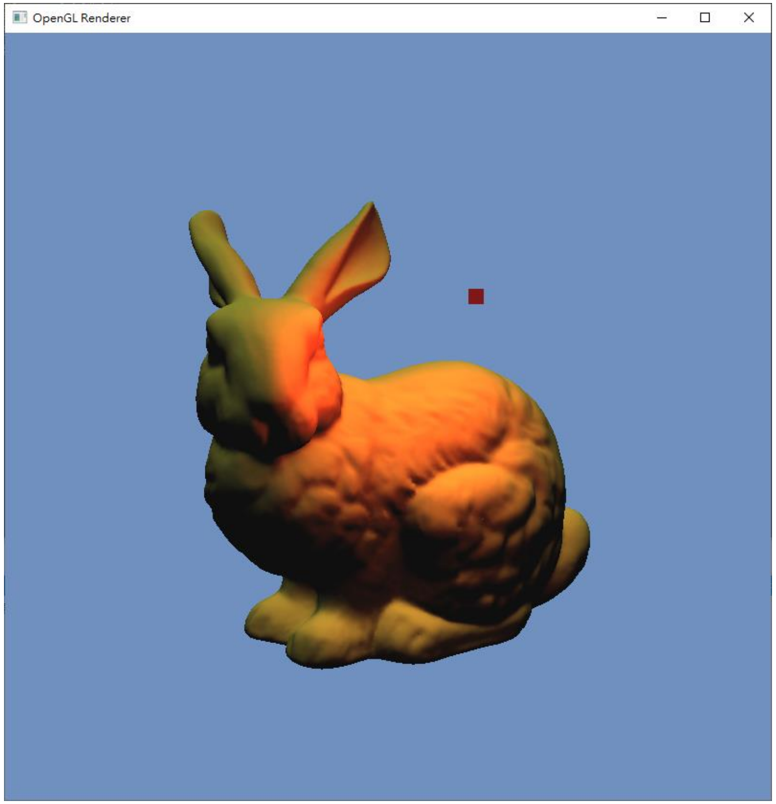

OpenGL Renderer


OpenGL Renderer
ugly highlight!
you can try:
(1) increase sphere triangles
(2) Phong shading

Results (cont.)

- Combine your **TriangleMesh** class in HW1
- Play with different light and material parameters





Practices

- Implement specular shading (HW2)
- Implement spotlight (HW2)
- Implement Phong shading (HW2)


END OF PRESENTATION
ANY QUESTIONS?