

VCL Lab 3 完整实验报告

王泽恺 2400013155

目录

1. Task 1 - Phong光照模型
2. Task 2 - 环境映射
3. Task 3 - 非真实感渲染
4. Task 4 - 阴影映射
5. Task 5 - 光线追踪

Task 1 - Phong光照模型

基本原理

完整公式：

$$I = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}}$$

各分量说明：

- **环境光 (Ambient)**：表示全局均匀的环境光照

$$I_{\text{ambient}} = k_a * I_{\text{env}}$$

其中 k_a 是环境光系数

- **漫反射 (Diffuse)**：表示光线在粗糙表面的漫反射，遵循Lambert定律

$$I_{\text{diffuse}} = k_d * \max(\theta, N \cdot L) * I_{\text{light}}$$

其中N是表面法线，L是指向光源的单位向量

- **镜面反射 (Specular)**：表示光线在光滑表面的镜面反射

$$I_{\text{specular}} = k_s * \max(\theta, R \cdot V)^{\text{shininess}} * I_{\text{light}}$$

其中R是反射方向，V是指向观察者的单位向量

Blinn-Phong改进：

Blinn-Phong模型改进了镜面反射的计算，使用半向量代替反射向量：

```
I_specular = k_s * max(0, N·H)^shininess * I_light
```

其中 $H = \text{normalize}(L + V)$ 是光源方向与视线方向的半向量。

Blinn-Phong的优点：

- 高光边界更平滑自然
- 计算量稍小（避免计算反射向量）
- 在某些情况下与物理光学更接近

效果分析

实现后的Phong模型能够正确渲染：

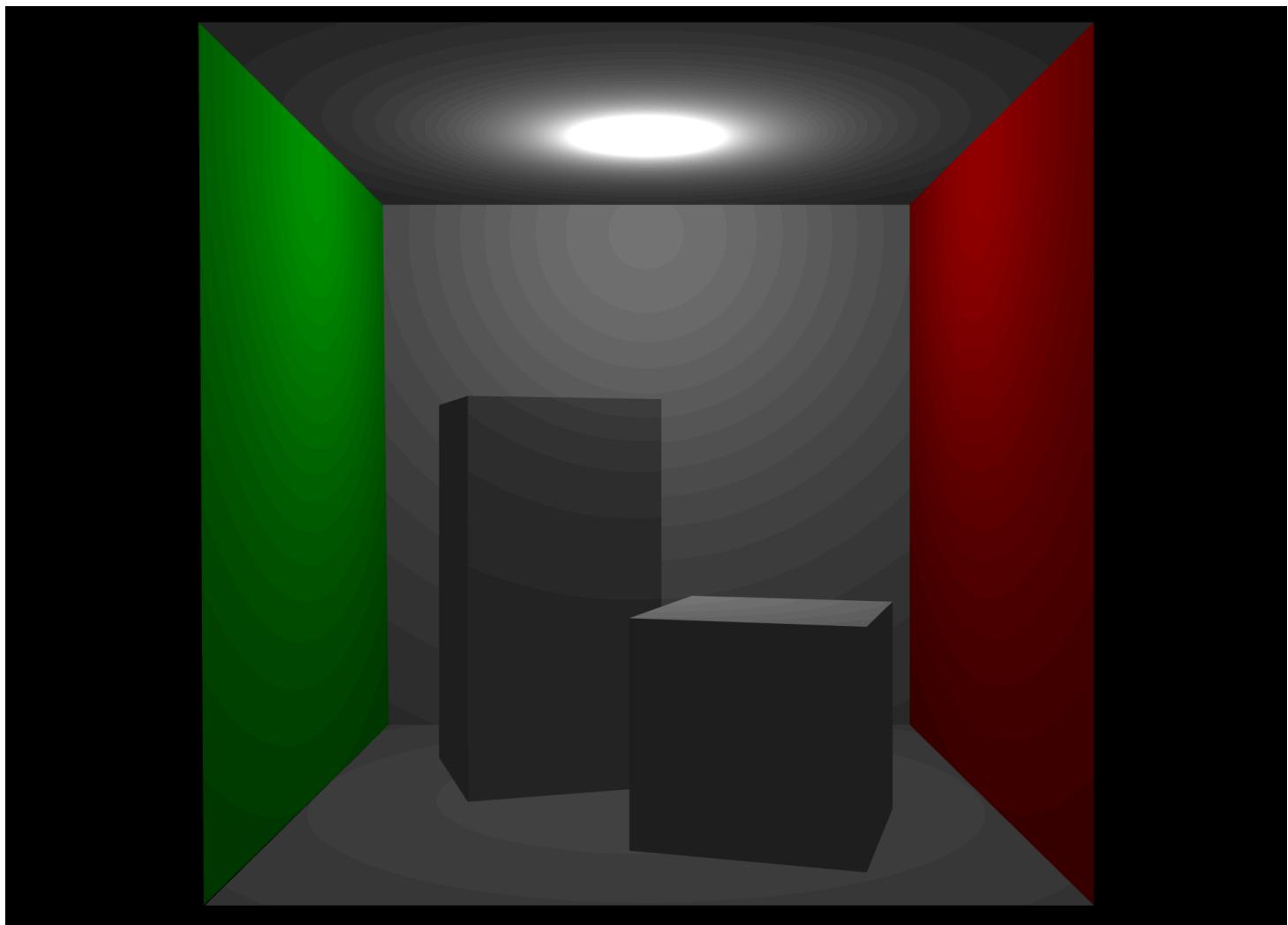
- 基本的光影效果
- 表面材质特性（通过 k_d , k_s , $shininess$ 参数控制）
- 多光源场景

Blinn-Phong相比Phong的优势在实际效果中体现为：

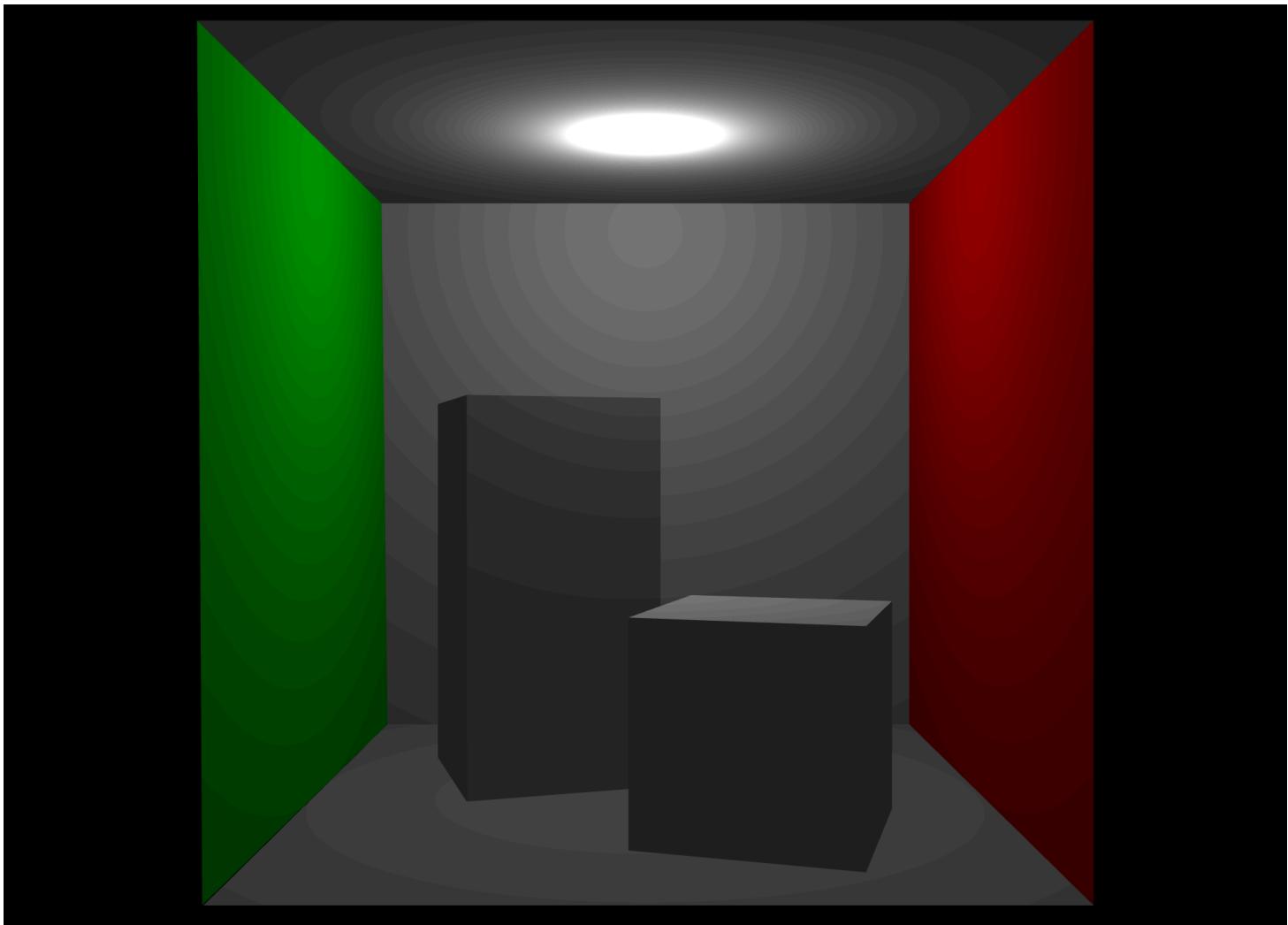
- 高光边界更加柔和
- 在不同视角下更稳定

示意图

phong：



blinn phong:



问题回答

Q1: 顶点着色器和片段着色器的关系？数据传递方式？

答：

- **顶点着色器**: 对每个顶点执行一次，输出顶点的变换后位置和其他属性
- **片段着色器**: 对每个像素（片段）执行一次，输入是顶点着色器输出经过光栅化插值后的值，计算该像素的最终颜色
- **关系**: 它们之间的关系是生产者-消费者模型：顶点着色器产生输出，片段着色器消费这些输出。
- **声明方式**:

在顶点着色器中用**out**关键字声明输出变量：

在片段着色器中用**in**关键字声明对应的输入变量

- **链接规则**:

名称相同：变量名必须完全一致

类型相同：数据类型必须匹配（如**vec3**, **vec2**, **float**等）

自动链接：在着色器程序链接阶段，OpenGL自动将同名同类型的变量连接起来

- 光栅化插值：

顶点着色器输出的数据并不是直接传递到片段着色器，而是经过光栅化阶段的插值：

顶点阶段：顶点着色器为每个顶点（如三角形的3个顶点）输出值

光栅化：将三角形转换为像素时，对顶点值进行透视正确的线性插值

片段阶段：片段着色器接收到的是插值后的值

Q2: `if (diffuseFactor.a < .2) discard;` 的作用？

答：

- **作用：**丢弃透明度小于0.2的片段，实现透明测试(Alpha Testing)

- **为什么不用 `== 0.0` :**

- 浮点数精度问题，`== 0.0` 容易因精度误差判断失败
- 使用范围判断 (`< 0.2`) 更稳健
- 允许半透明但不可见的材质被过滤

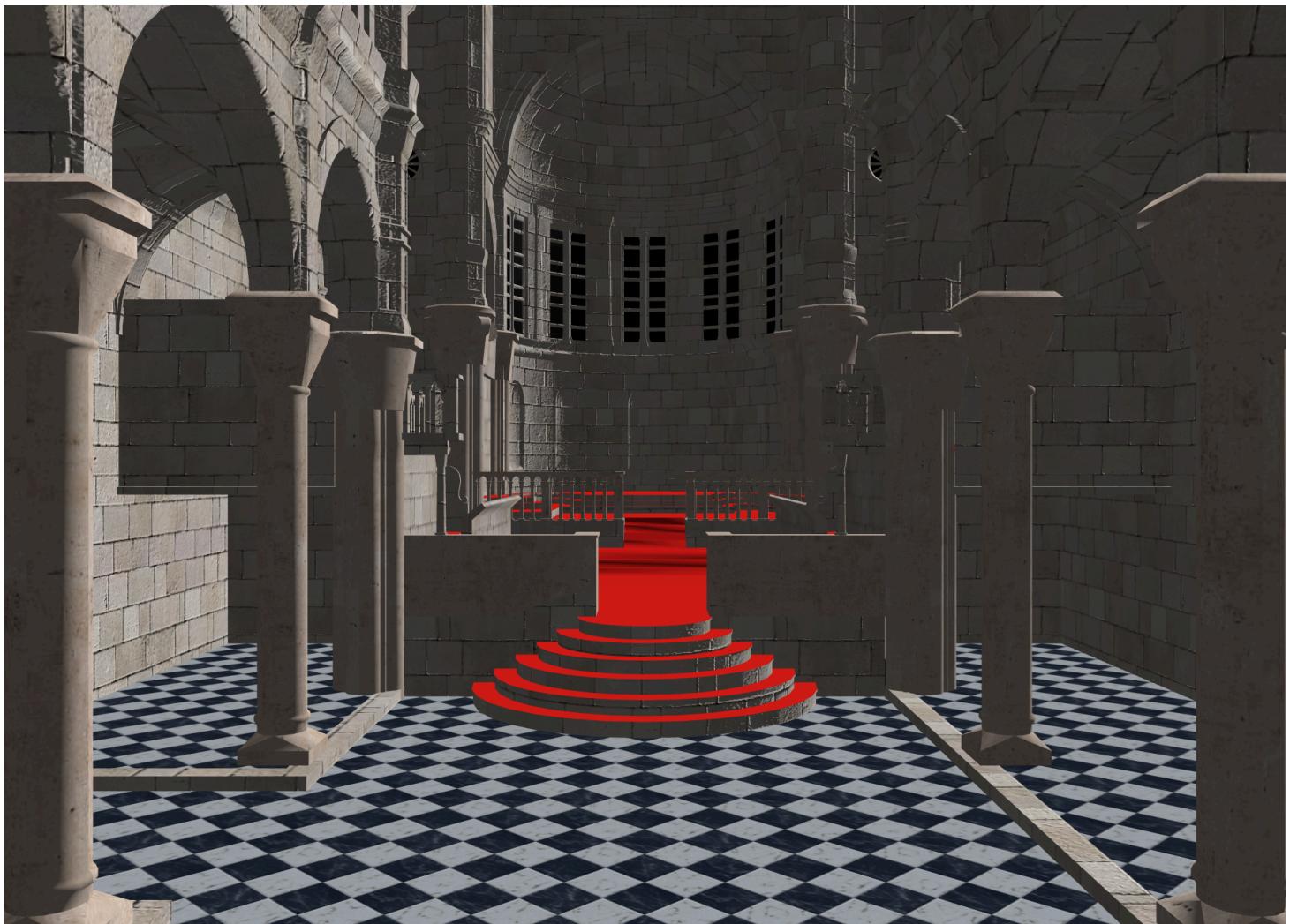
Bonus: 凸凹映射(Bump Mapping)

凸凹映射通过高度图改变每个像素的法线方向，模拟表面细节而无需增加几何复杂度。

实现原理：

1. 从高度图采样周围像素的高度值
2. 计算切线空间中的法线扰动
3. 变换回世界空间
4. 用扰动后的法线进行光照计算

效果图



Task 2 - 环境映射

基本原理

环境映射是一种高效的图像渲染技术，用立方体贴图(Cubemap)存储环境信息，可以：

- 快速渲染反射效果
- 提供逼真的背景
- 为场景提供全局光照信息

立方体贴图(Cubemap)：

- 由6张2D纹理组成（前、后、左、右、上、下）
- 通过3D坐标向量进行采样

- 适合表示全向的环境光照

实现流程

第一步：天空盒渲染 (skybox.vert)

第二步：环境贴图采样 (envmap.frag)

核心特性

- **立方体坐标系**: 使用3D方向向量代替2D纹理坐标
- **反射计算**: 通过 `reflect()` 函数计算反射方向
- **性能高效**: 相比实时追踪，效率提升10~100倍

效果图



Task 3 - 非真实感渲染(NPR)

基本概念

非真实感渲染(Non-Photorealistic Rendering)旨在模拟绘画、素描等艺术风格，而非追求照片级真实。

本任务采用**Gooch Shading**实现，结合以下特点：

1. 轮廓线：渲染背面并扩大，叠加正面
2. 颜色过渡：从冷色→中性色→暖色的渐变
3. 卡通效果：颜色分界线明显

实现方法

第一步：轮廓线渲染（提供）

第二步：Gooch Shading (npr.frag)

效果图



问题回答

Q1: 如何分别渲染反面和正面？

答：代码通过两次渲染 + 面剔除实现分别处理：

- 第一次渲染 (`glCullFace(GL_FRONT)`)：
只渲染背面
使用轮廓线着色器扩大顶点
形成轮廓线
- 第二次渲染 (`glCullFace(GL_BACK)`)：
只渲染正面
使用正常的Gooch着色器
覆盖中间部分，只露出边缘的轮廓线

Q2: 为什么不在世界坐标中沿法线移动顶点?

答:

- 会导致**顶点分离**: 相邻三角形的顶点不共享
- **不同光照**: 背面会被额外光照影响
- **法线方向问题**: 在世界坐标中, 不同大小的模型需要不同的偏移
- **会产生缝隙**: 扩大后的背面和正面之间有空隙

Task 4 - 阴影映射

基本原理

阴影映射(Shadow Mapping)是实时阴影的标准方法:

两步过程:

1. 阴影生成阶段 (从光源视角)

- 将场景从光源位置渲染
- 保存深度值为阴影贴图(Shadow Map)
- 记录光源能"看到"的最近深度

2. 着色阶段 (从相机视角)

- 计算着色点到光源的距离
- 与阴影贴图中的深度比较
- 若当前深度 > 贴图深度, 则在阴影中

效果图



问题回答

Q1: 深度贴图的投影矩阵选择?

答:

- **有向光源:** 使用**正交投影**(Orthographic Projection)
 - 光线平行, 没有透视效果
- **点光源:** 使用**透视投影**(Perspective Projection)
 - 光线从一点发出, 需要透视变换
 - 对每个面使用正确的投影矩阵

Q2: 为什么无需计算像素深度?

答:

- OpenGL的**深度缓冲**自动计算每个片段的深度值

- 渲染管线流程：

顶点着色器 → 图元装配 → 光栅化 → [深度计算] → 片段着色器 → 深度测试 → 帧缓冲

↑
自动进行

- 片段着色器无需显式计算，深度值自动保存到深度纹理

Task 5 - 光线追踪

算法原理

Whitted光线追踪是1980年Bui Tuong Whitted提出的递归光线追踪算法，通过反向追踪光线来计算全局光照效果。

核心思想：

- 从相机发射光线穿过每个像素
- 计算光线与场景的交点
- 在交点处计算局部光照（Phong模型）
- 发射阴影光线判断是否在阴影中
- 生成反射/折射光线递归追踪
- 累积所有贡献得到最终颜色

Whitted算法的关键特点：

- 采用**后向追踪**（Backward Tracing）：从相机出发而非光源，效率更高[1]
- 使用**递归光线树**（Ray Tree）：主光线可以产生反射、折射和阴影光线，这些次级光线又可以产生新的光线[2]
- 通过**简化光照积分**：只考虑光源方向、完美反射和折射方向，将积分转化为有限项的和[3]

光线-三角形相交

使用**重心坐标法**(Barycentric Coordinates)求解相交。

数学推导：

光线方程：

$$P(t) = O + t * D$$

三角形平面方程：

$$(P - p1) \cdot n = 0$$

其中 $n = (p2 - p1) \times (p3 - p1)$

联立求解得到交点参数t和重心坐标(u, v)。

重心坐标定义：

对于三角形内的点p：

$$p = (1-u-v) * p1 + u * p2 + v * p3$$

其中 $u, v \geq 0$ 且 $u + v \leq 1$ 。

算法步骤：

1. 计算光线与三角形所在平面的交点参数t
2. 验证t是否为正（交点在光线前方）
3. 计算交点的重心坐标(u, v)
4. 验证交点是否在三角形内部 ($u \geq 0, v \geq 0, u+v \leq 1$)

重心坐标的优势：

- 可以用于插值顶点属性（法线、纹理坐标、颜色等）
- 判断点是否在三角形内变得简单
- 数值稳定性好

Phong光照与阴影

局部光照计算流程：

在光线与表面相交后，需要计算该交点的局部光照，这遵循Phong光照模型：[4][1]

1. 环境光 (Ambient)

- 表示全局均匀的背景光照
- 防止阴影区域完全黑暗

2. 遍历所有光源计算直接光照

对于每个光源，执行以下步骤：

a) 计算光源方向和衰减

- **点光源**:
 - 光源方向: $L = \text{normalize}(\text{lightPos} - \text{hitPoint})$
 - 衰减: $\text{attenuation} = 1.0 / \text{distance}^2$ (距离平方反比定律)
- **方向光**:
 - 光源方向: $L = \text{normalize}(\text{lightDirection})$
 - 衰减: $\text{attenuation} = 1.0$ (无衰减)

b) 阴影检测 (Shadow Ray)

这是Whitted算法的核心特性之一：[2][1]

- 从交点沿光源方向发射**阴影光线**
- 检查光线是否在到达光源前击中其他物体
- 如果有遮挡，该光源不对当前点贡献光照

阴影光线的关键技术点:

- **起点偏移**: 阴影光线起点要沿法线方向偏移小量 (如 $\text{hitPoint} + \epsilon \times \text{normal}$)，避免自相交
- **透明度过滤**: 只有不透明物体 ($\alpha \geq 0.2$) 才会产生阴影
- **距离判断**: 对于点光源，只有遮挡物距离小于光源距离才算阴影

c) 计算Phong光照分量

如果该点不在阴影中，计算两个光照分量：

漫反射 (Diffuse):

- 模拟光线在粗糙表面的散射
- 公式: $\text{diffuse} = k_d \times \max(0, \mathbf{N} \cdot \mathbf{L}) \times I_{\text{light}} \times \text{attenuation}$
- 符合Lambert余弦定律：光照强度与入射角余弦成正比

镜面反射 (Specular):

- 模拟光滑表面的高光效果
- 反射方向: $R = 2(N \cdot L)N - L$
- 公式: $\text{specular} = k_s \times \max(0, R \cdot V)^{\text{shininess}} \times I_{\text{light}} \times \text{attenuation}$
- shininess越大, 高光越锐利

d) 累加光照贡献

```
totalLight += attenuation * (diffuse + specular) * I_light
```

效果图

