

Introduction to Computer Graphics



Shading 2

Ming Zeng
Software School,
Xiamen University

Contact Information:
zengming@xmu.edu.cn

第六章 明暗着色(Shading)

本章主要介绍光源作用在物体上呈现出不同明暗效果的物理过程。

- 基本概念

- 增强真实感的几种方式
- 为什么需要明暗着色
- 光源的类型、材料的类型

- Phong光照明模型

- 给定光源与材质、视点，如何计算光照效果
- Phong模型的改进版本——Blinn模型

- 多边形明暗处理

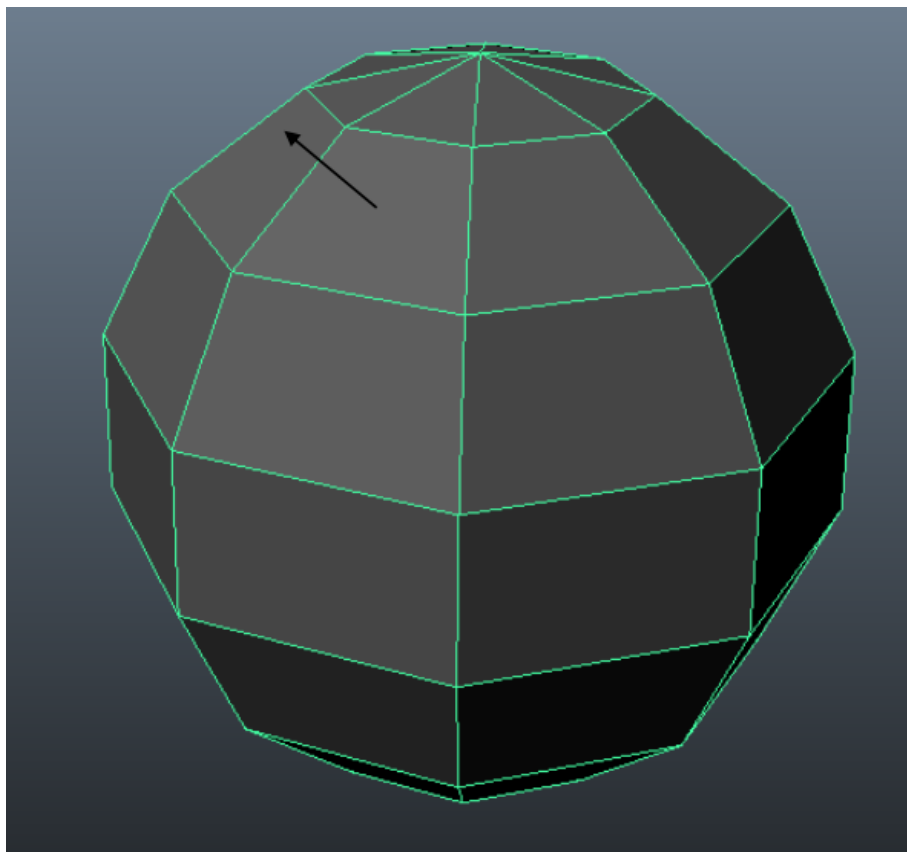
- OpenGL明暗处理

第三部分 多边形明暗处理

- 主要内容
 - 什么是多边形明暗处理
 - 多边形的明暗处理
 - 平面着色
 - Gouraud着色
 - Phong着色

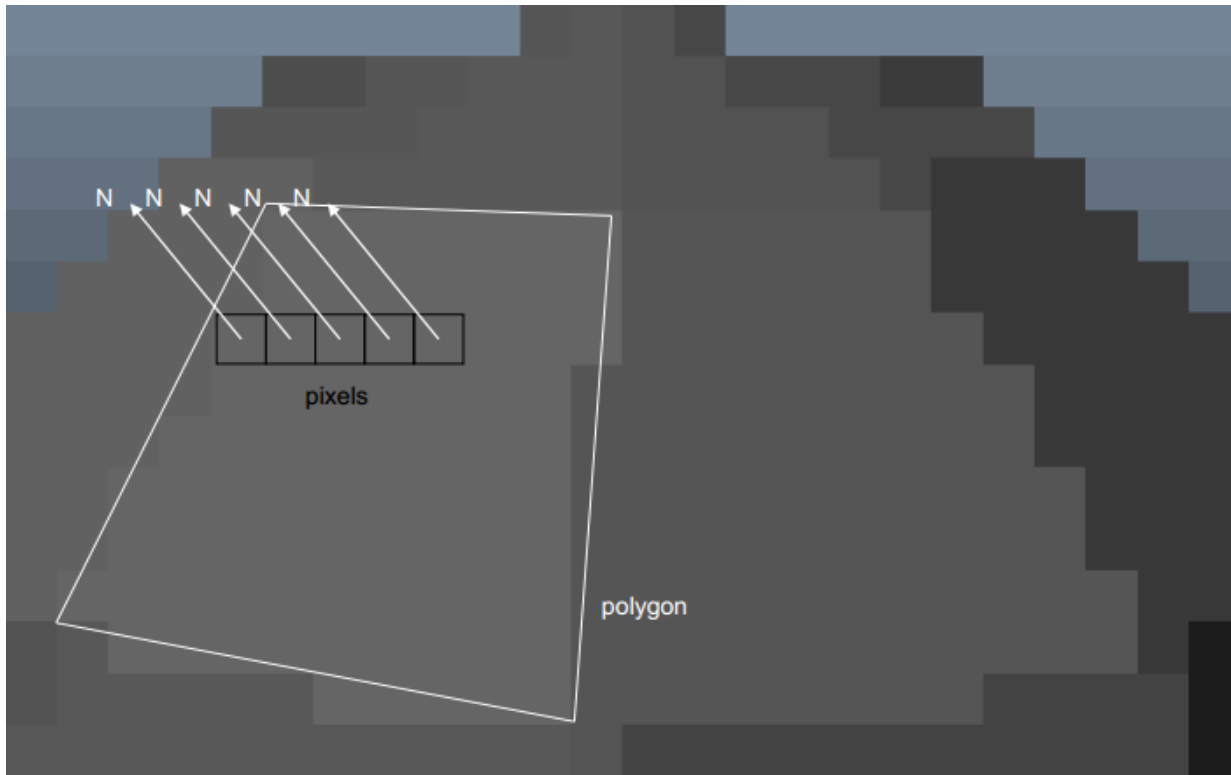
什么是多边形明暗处理

- 球面：用若干个多边形近似逼近



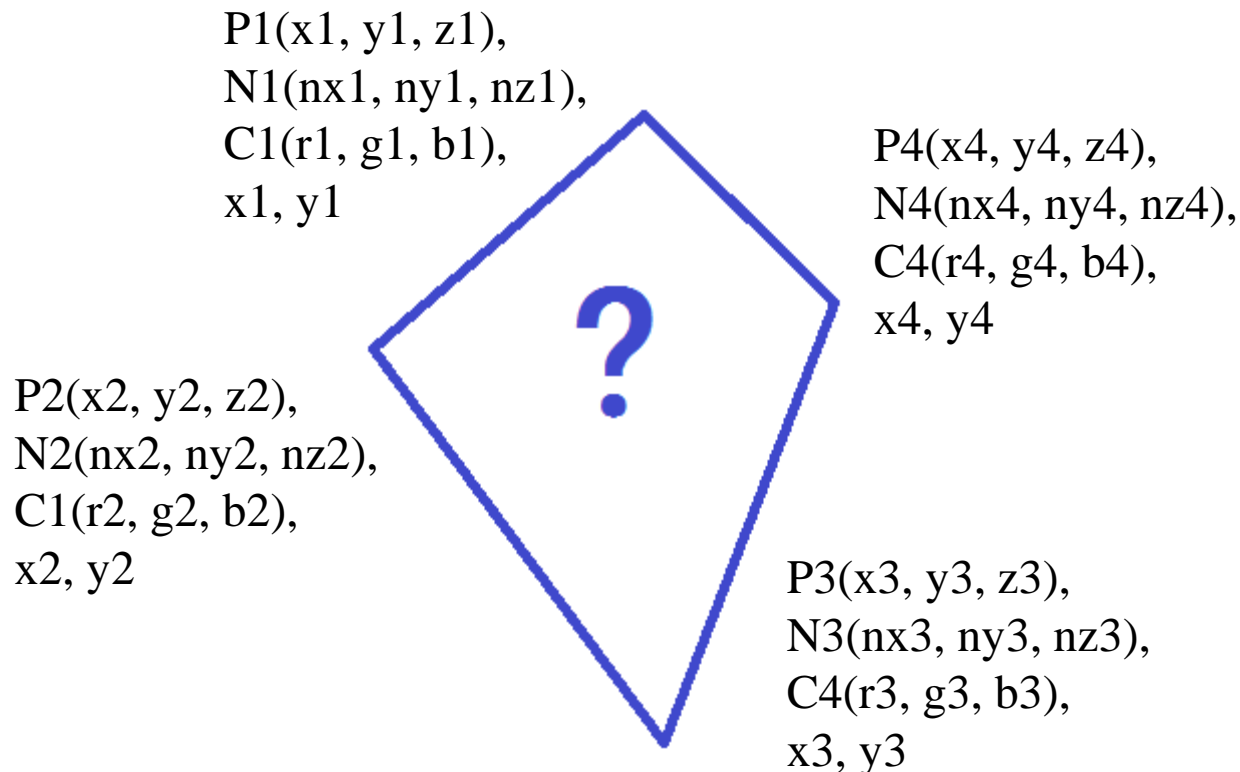
什么是多边形明暗处理

- 白色区域是某个多边形。多边形明暗处理就是为了确定多边形内部的每个像素是何种颜色。

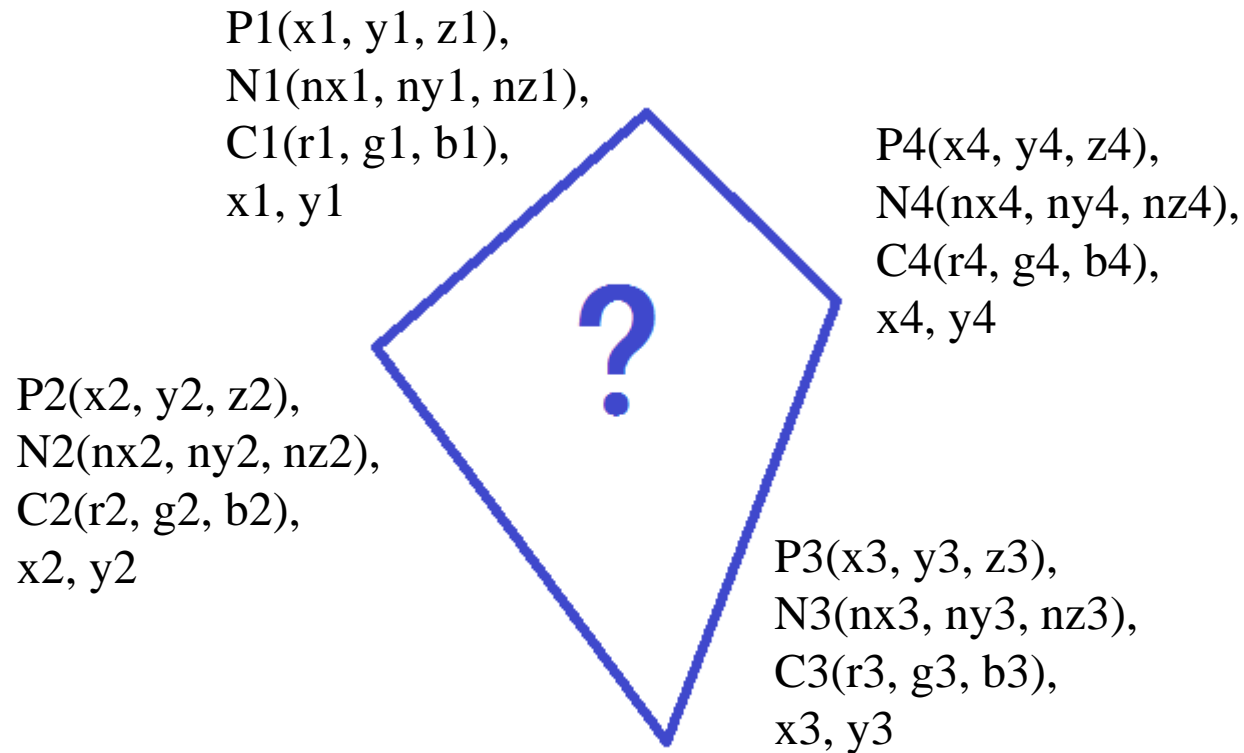


问题

- 已经求得帧缓冲区（画布）中多边形四个顶点的位置，以及相应的颜色、法向，如何求多边形内各个像素的颜色？

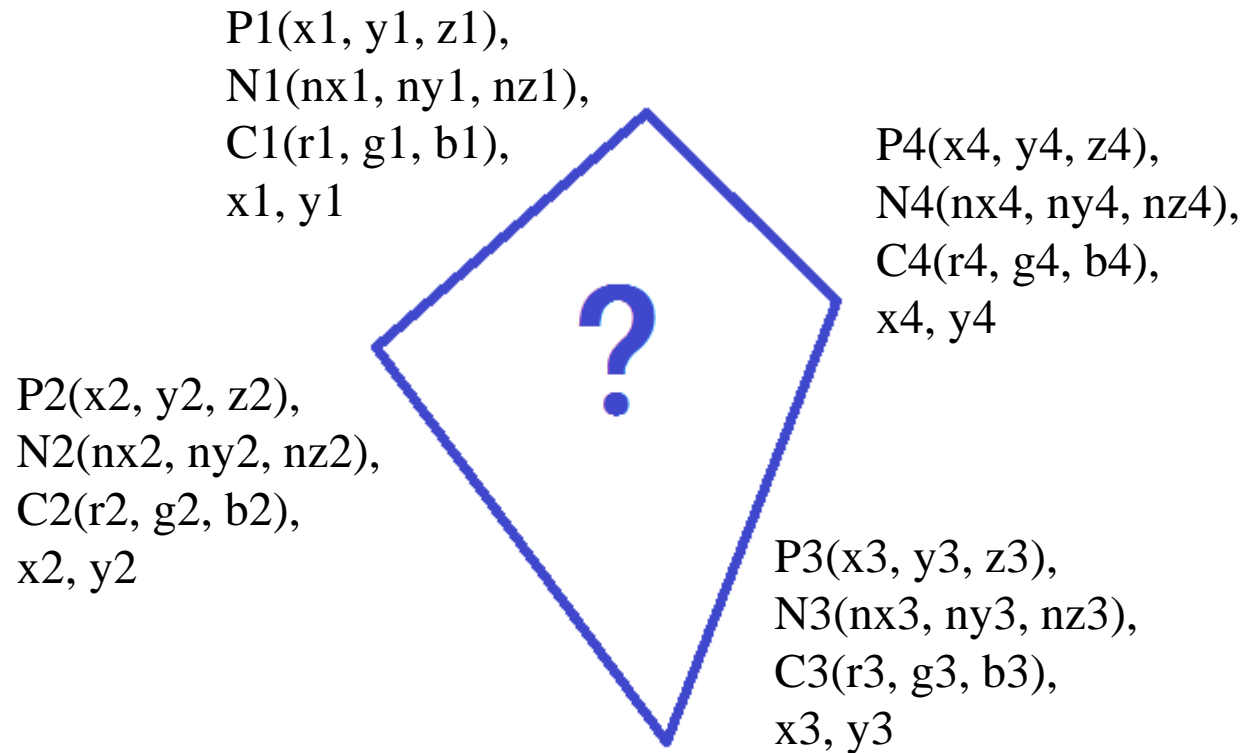


问题



- x_i, y_i 可以通过顶点变换求得

问题

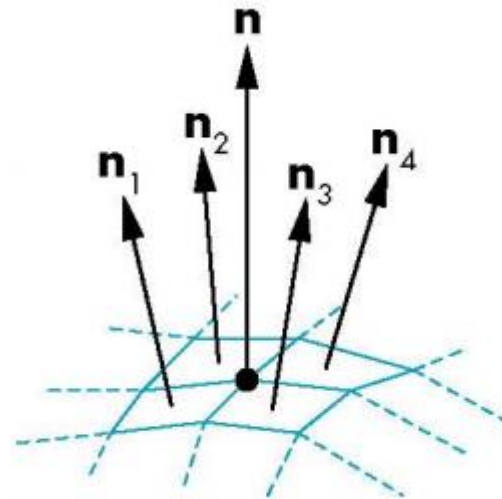


- Ni 可以对相邻平面求法向平均

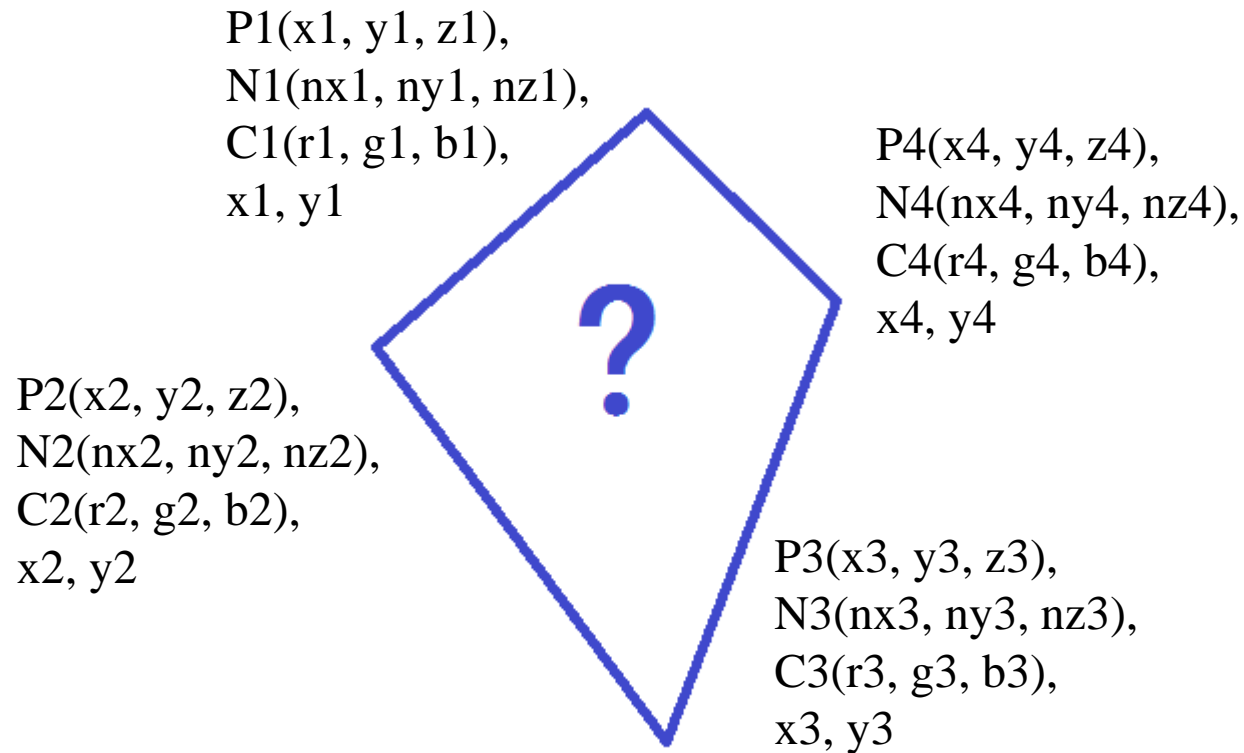
如何求法向

- 在网格中每个顶点处有几个多边形交于该点，每个多边形有一个法向，取这几个法向的平均得到该点的法向

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$



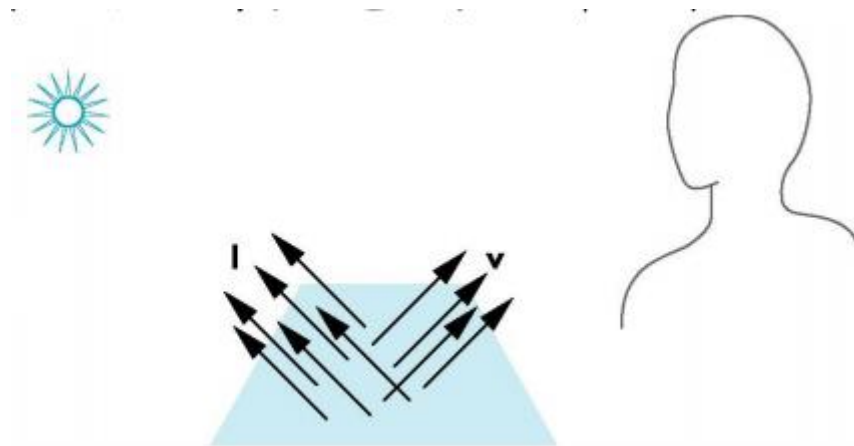
问题



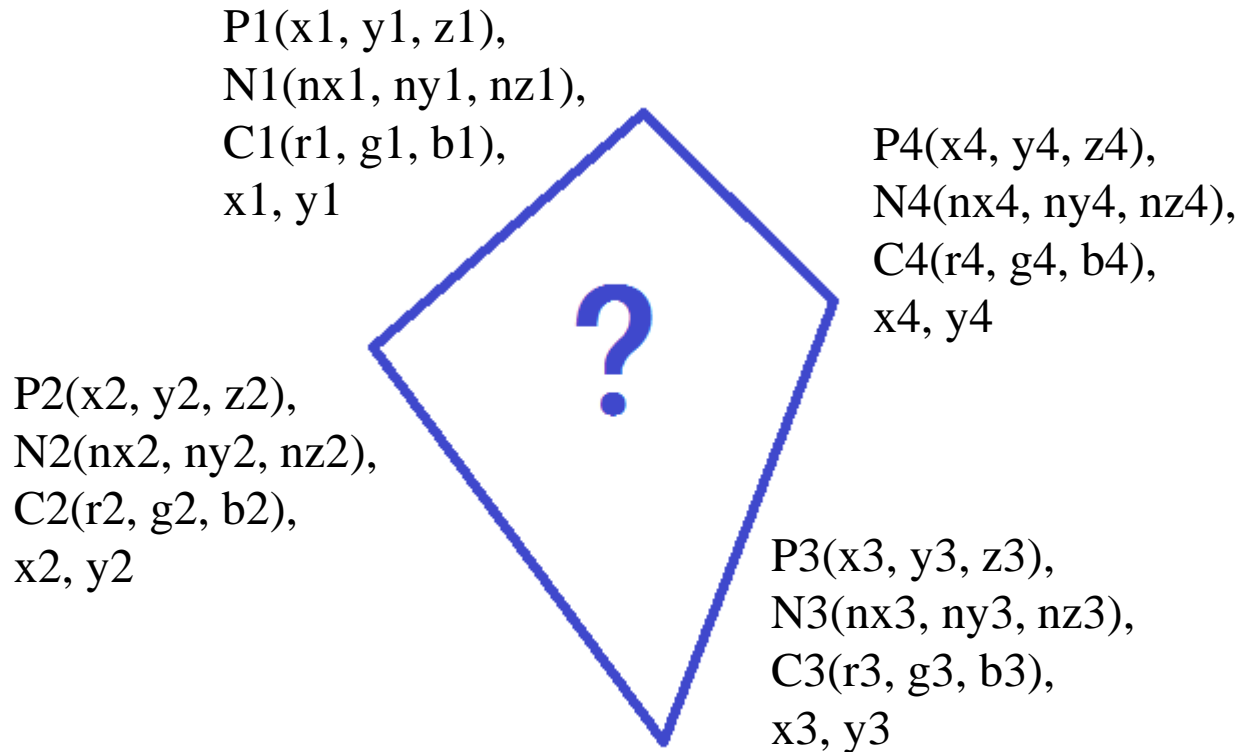
- C_i 可以对每个顶点用光照明模型求得

解决方法1——平面着色法 (flat shading)

- 在同一多边形上法向 \mathbf{n} 为常向量
- 假设视点在无穷远，视点方向 \mathbf{v} 是常向量
- 假设光源在无穷远，入射方向 \mathbf{l} 也是常向量
- 从而对于每个多边形，只需要计算其上一点的颜色，其它点的颜色与它相同



平面着色法



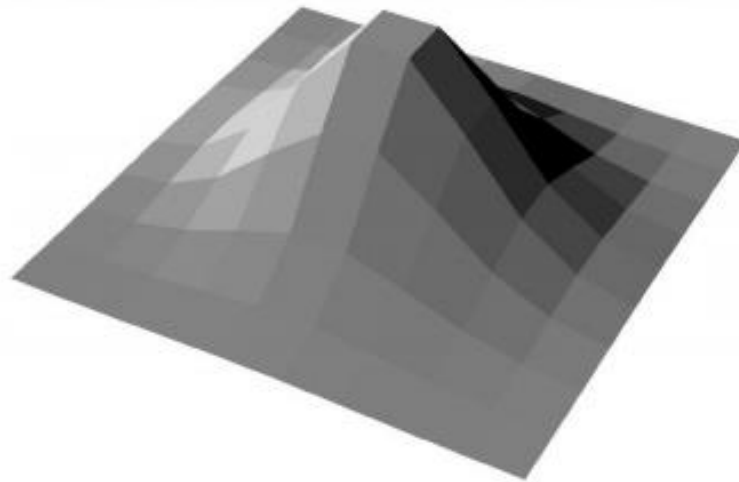
- ? 处的颜色用某一个顶点的颜色代替

OpenGL平面着色法

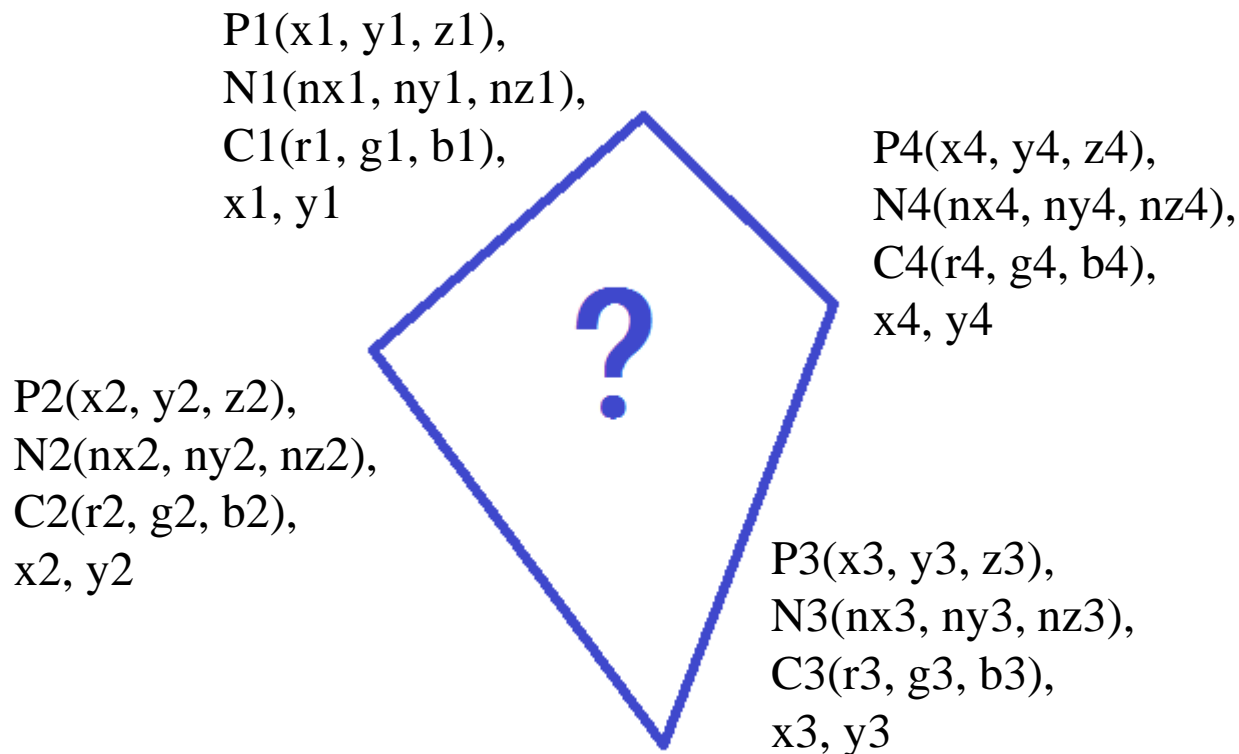
- OpenGL中设置平面着色：
glShadeModel(GL_FLAT);
- 如何选择多边形的法向或颜色：
 - 单个多边形(GL_POLYGON) 第1个顶点
 - 独立三角形(GL_TRIANGLES) 第 $3i$ 个顶点
 - 独立四边形(GL_QUADS) 第 $4i$ 个顶点
 - 四边形带(GL_QUAD_STRIP) 第 $2i+2$ 个顶点
 - 三角形带或三角形扇 第 $i+2$ 个顶点

特点

- 网格中每个多边形的颜色不同
 - 如果多边形网格表示的是一个光滑曲面，那么这种效果显然是不令人满意的



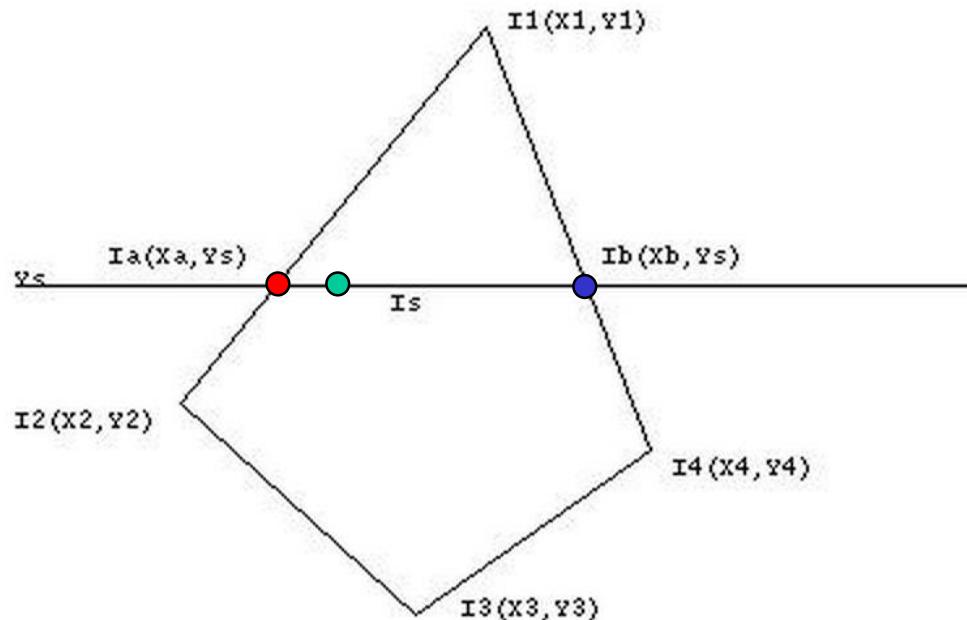
解决方法2——Gouraud着色法(Gouraud shading)



- ? 处的颜色用顶点颜色的线性插值设置

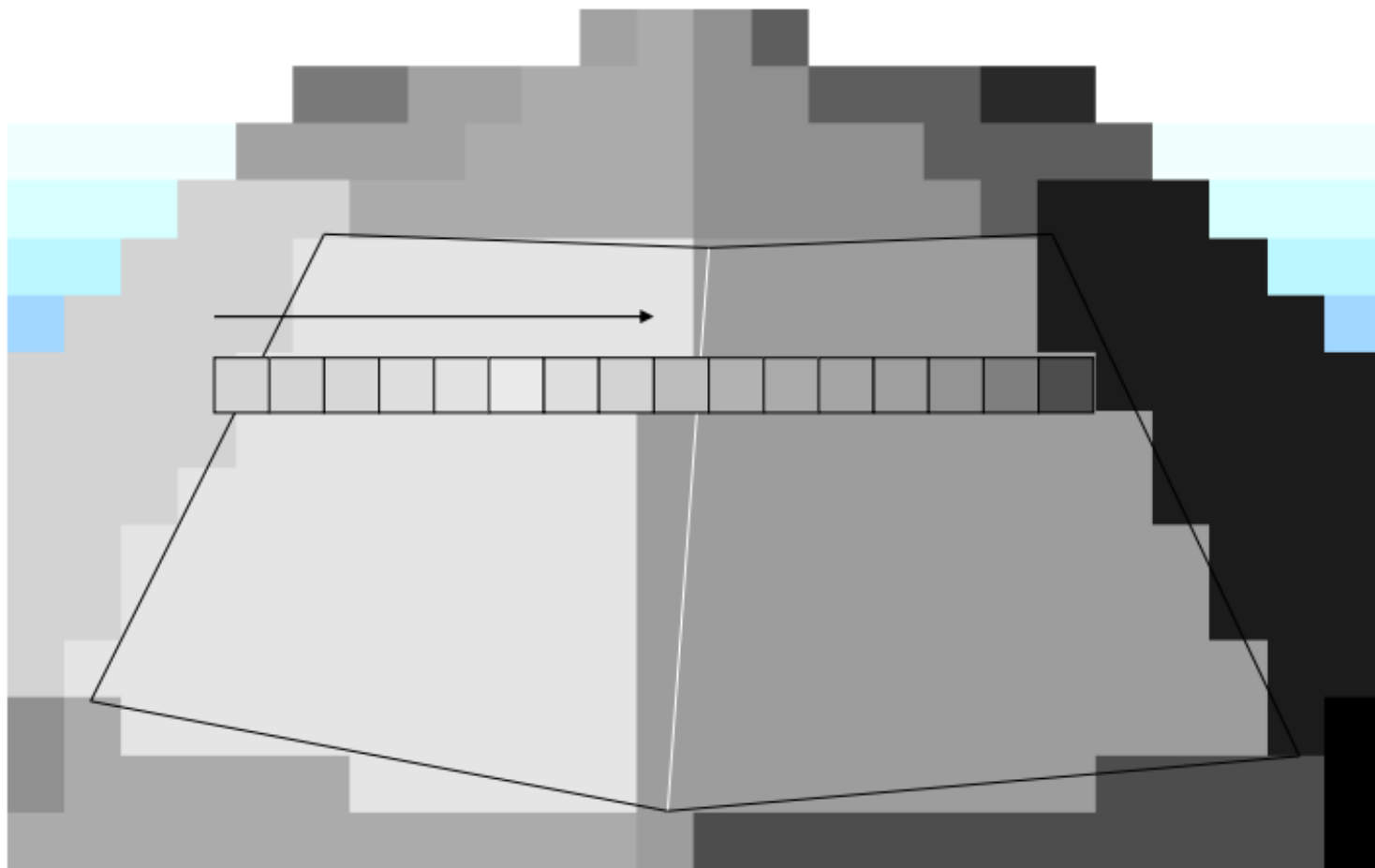
算法（对颜色进行双线性插值）

- 水平扫描线自上而下扫描
 - 对每条扫描线
 - 计算其与多边形的左右交点
 - 分别计算左右交点的颜色
 - 对扫描线上的每一点，计算其颜色



效果

- 同一个多边形上的颜色有渐变

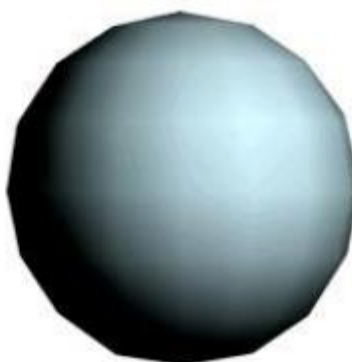
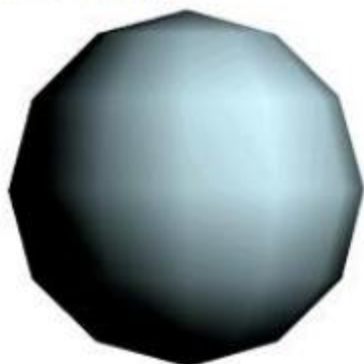


比较

Flat Shading

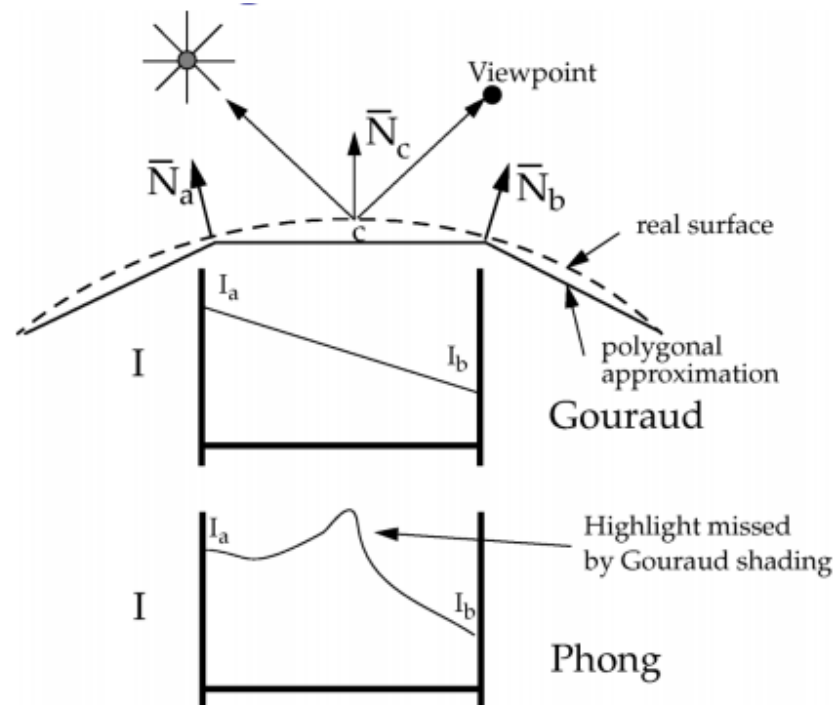


Gouraud Shading



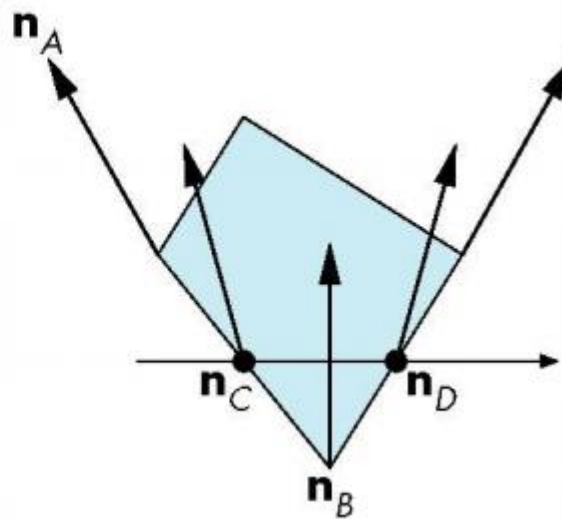
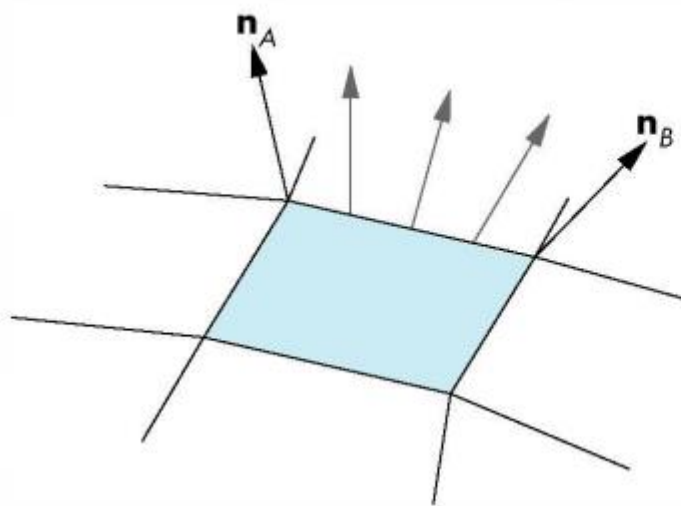
Gouraud着色的问题

- 高光丢失
- c点应有高光，a,b点没有高光
- 通过a,b点插值出c，丢失高光



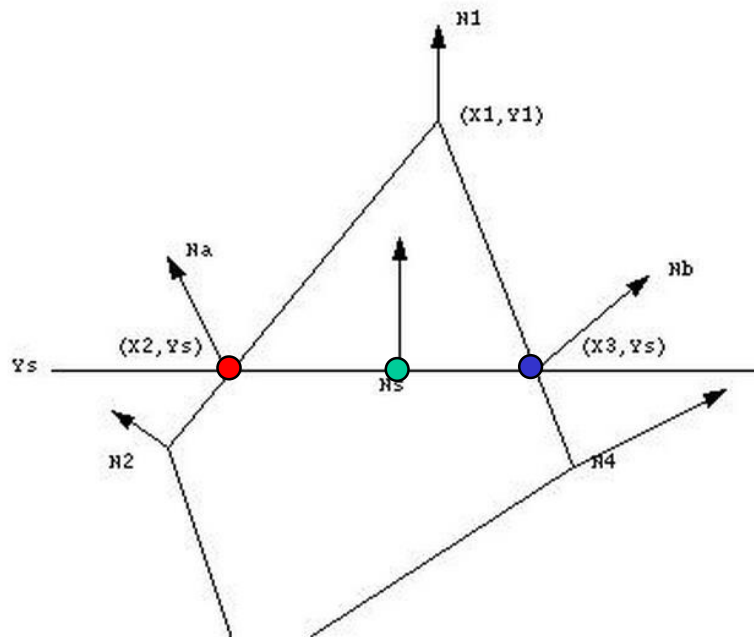
解决方法3 Phong着色

- 不直接对颜色插值，而是对法向插值，再用插值的法向计算光照

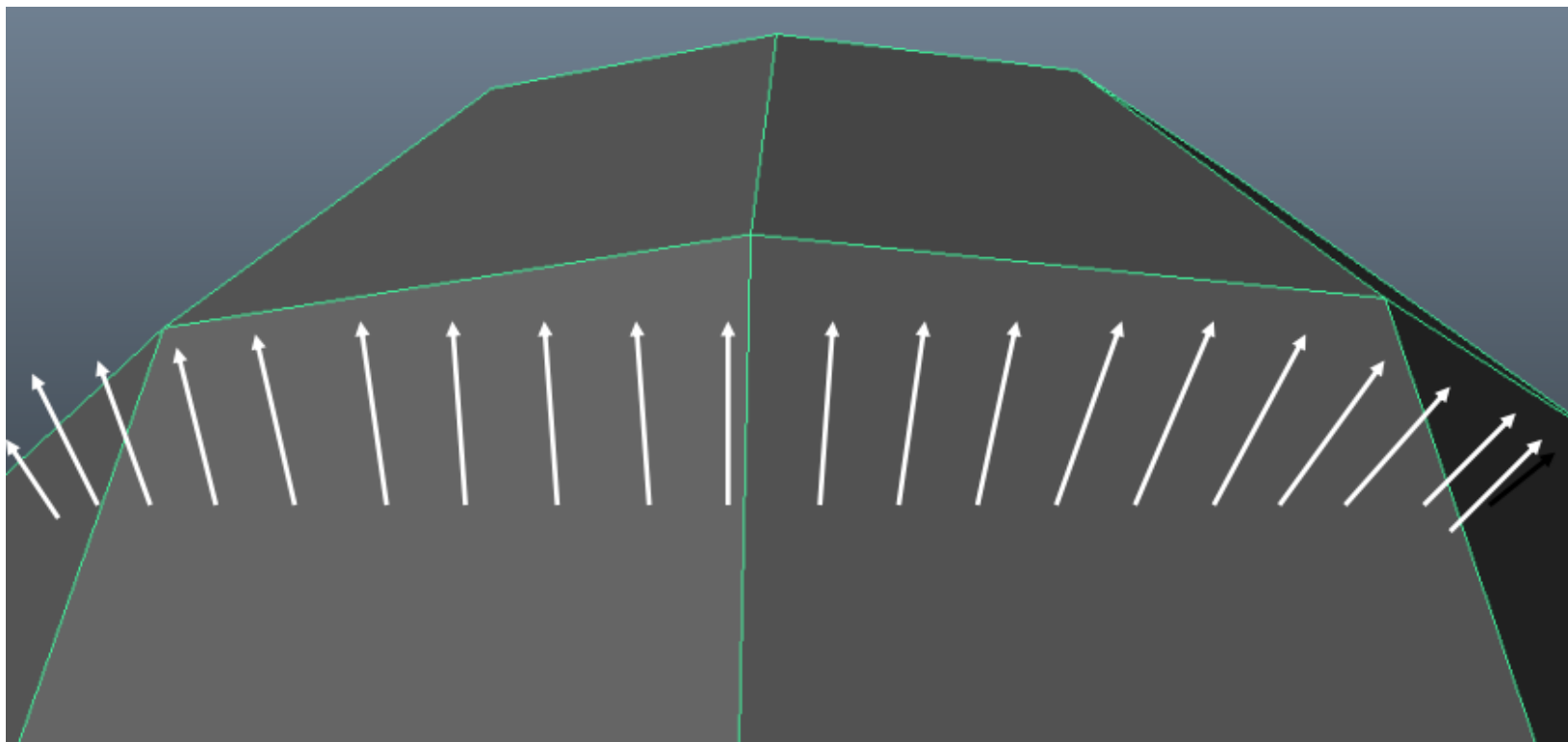


算法（对法向进行双线性插值）

- 水平扫描线自上而下扫描
 - 对每条扫描线
 - 计算其与多边形的左右交点
 - 分别计算左右交点的法向
 - 对扫描线上的每一点，计算其法向，再用光照模型计算该点的颜色

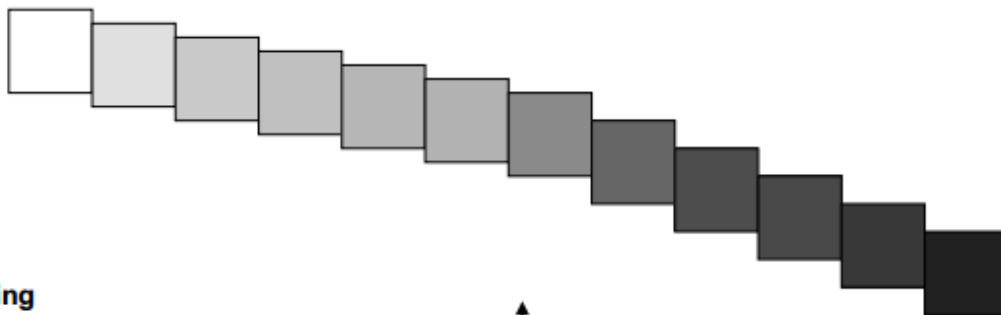


效果

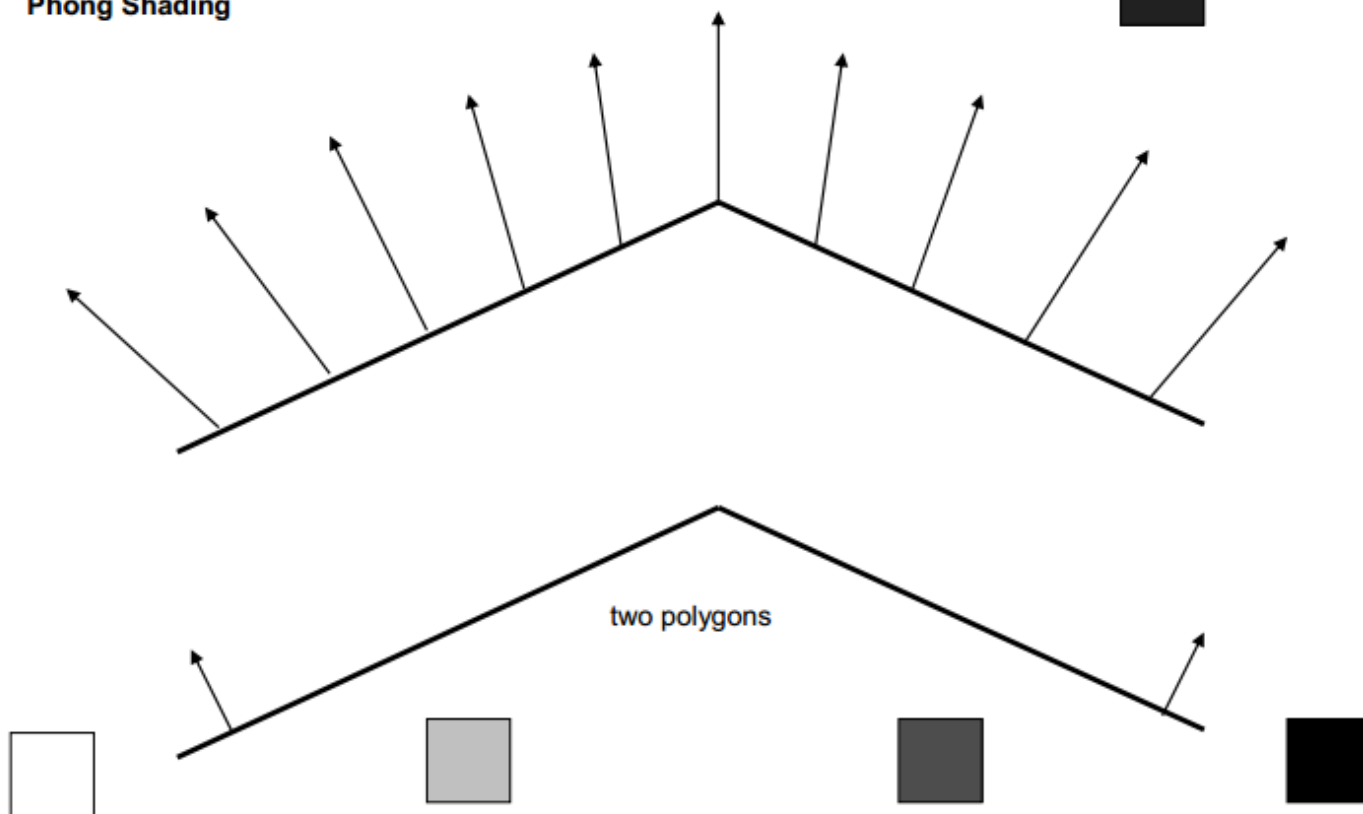


比较

Gouraud Shading



Phong Shading

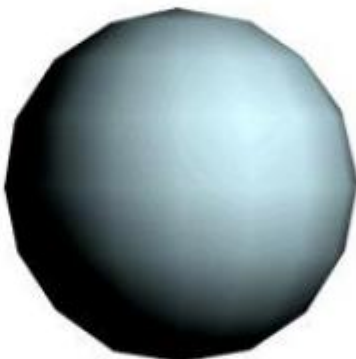
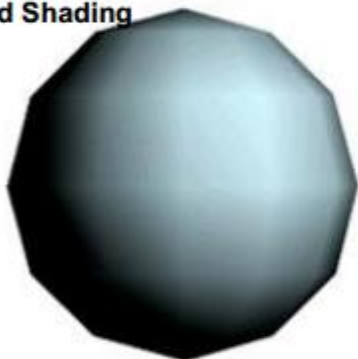


比较

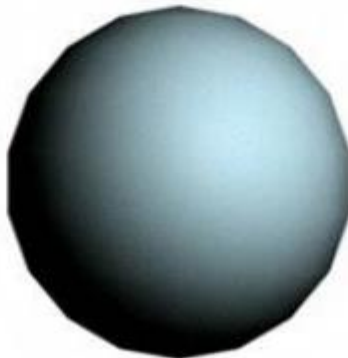
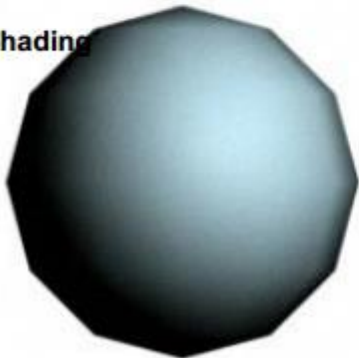
Flat Shading



Gouraud Shading



Phong Shading

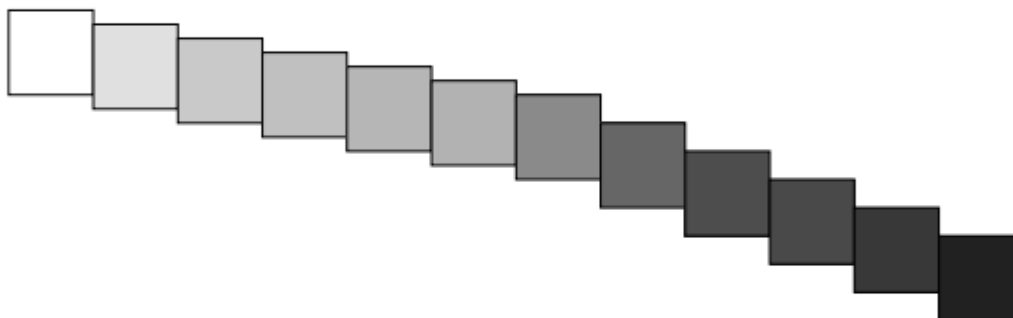


比较

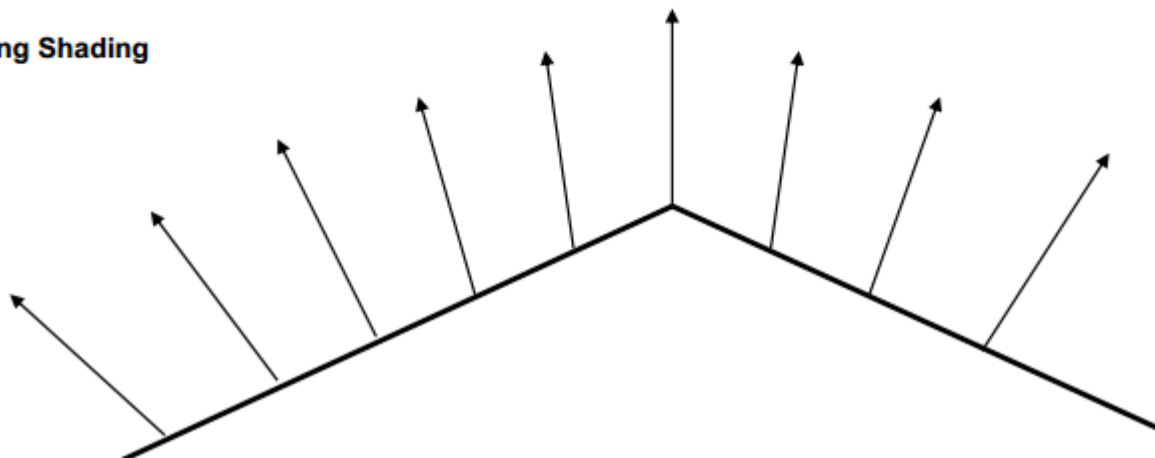
Flat Shading



Gouraud Shading



Phong Shading



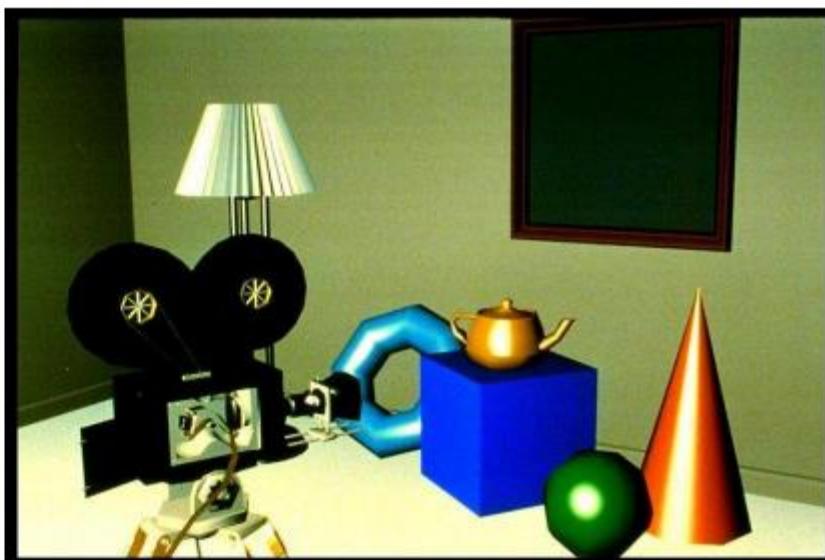
比较

- 得到的图形比应用Gouraud方法的结果更光滑
- 但是由于法向的计算还是很复杂，一般无法得到实时图形
 - 所花费时间通常是Gouraud方法的6到8倍
- OpenGL实现的是Gouraud方法

比较

- 如果用多边形网格逼近大曲率曲面，Phong方法的结果可能看起来光滑一些，而Gouraud方法就会使边有些明显
- Phong方法比Gouraud方法的复杂度高
 - 可以用片段处理器实现

比较



第四部分 OpenGL明暗处理

- OpenGL如何模拟光照
 - 改进的Phong光照模型
 - 对每个顶点计算颜色
 - 影响光照的因素
 - 表面材料属性
 - 光源属性
 - 光照模型属性

在OpenGL中应用明暗处理的步骤

1. 指定法向量
2. 启用明暗处理功能，并选择模式
3. 指定光源
4. 指定材料属性

法向量

- 在OpenGL中法向量是状态的一部分
- 利用`glNormal*()`设置， 例
 - `glNormal3d(x,y,z);`
 - `glNormal3dv(p);`
- 通常需要法向量为单位向量，这样余弦计算就非常直接
 - 变换会影响其长度，注意放缩并不保持其长度
 - `glEnable(GL_NORMALIZE)`或
`glEnable(GL_RESCALE_NORMAL)`可以使OpenGL自动进行单位化，当然以损失效率为代价

启用

- 明暗处理的计算由下述命令启用
glEnable(GL_LIGHTING)
 - 如果光照被激活, `glColor()`命令将被忽略
- 必须单独激活每个光源
 - **glEnable(GL_LIGHTi)**, $i = 0, 1, \dots, 7$

选择光照模型

- OpenGL的光照模型参数包括4个部分
 - 全局环境光强度
 - 观察点位于场景中还是无限远处
 - 物体的正反面是否执行不同的光照计算
 - 镜面光颜色是否从环境光和漫反射颜色中分离出来，并在纹理操作后再应用

glLightModel*()

- *void glLightModel{if}(GLenum pname, TYPE param);*
*void glLightModel{if}v(GLenum pname, TYPE *param);*
- 参数及默认值, 意义
 - **GL_LIGHT_MODEL_AMBIENT**, (0.2,0.2,0.2,1.0), 整个场景中的全局环境光强
 - **GL_LIGHT_MODEL_LOCAL_VIEWER**, 0.0或**GL_FALSE**, 在计算中应用无穷远视点的假设简化计算
 - **GL_LIGHT_MODEL_TWO_SIDED**, 0.0或**GL_FALSE**, 单独对多边形的两面进行明暗处理
 - **GL_LIGHT_MODEL_COLOR_CONTROL**, **GL_SINGLE_COLOR**(默认)或**GL_SEPARATE_SPECULAR_COLOR**, 镜面光是否与漫反射和环境光分开计算 (主要与纹理颜色相关)

全局环境光

- 环境光依赖于每个光源的颜色，因此需要对每个光源指定环境光强
 - 在白屋中的红灯会使生成红色环境光，当灯被关闭后这种成分就消失
- OpenGL中也可以定义一个对测试非常有用的全局环境光

```
GLfloat global_ambient[]={0.2,0,0,1};  
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  
global_ambient);
```

定义光源

- 每个光源都有环境光、漫射光和镜面光项

void glLight{if}(GLenum light, GLenum pname, TYPE param);

*void glLight{if}v(GLenum light, GLenum pname, TYPE *param);*

– light: GL_LIGHT0, GL_LIGHT1, ... , or
GL_LIGHT7

– pname:

- 颜色值: GL_AMBIENT, GL_DIFFUSE,
GL_SPECULAR

- 位置: GL_POSITION

- 衰减项: GL_CONSTANT_ATTENUATION,
GL_LINEAR_ATTENUATION,
GL_QUADRATIC_ATTENUATION

- 聚光灯参数: GL_SPOT_CUTOFF,
GL_SPOT_DIRECTION, GL_SPOT_EXPONENT

定义点光源

- 对于每个光源，可以设置漫反射光、镜面光和环境光的RGB值以及光源的位置

```
GLfloat diffuse0[]={1.0,0.0,0.0,1.0};
```

```
GLfloat ambient0[]={1.0,0.0,0.0,1.0};
```

```
GLfloat specular0[]={1.0,0.0,0.0,1.0};
```

```
GLfloat light0_pos[]={1.0,2.0,3.0,1.0};
```

```
glEnable(GL_LIGHTING);
```

```
glEnable(GL_LIGHT0);
```

```
glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
```

```
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
```

```
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
```

```
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

距离与方向

- 光源的颜色应当以RGBA模式定义
 - GL_AMBIENT的缺省值是(0.0, 0.0, 0.0, 1.0)
 - GL_DIFFUSE和 GL_SPECULAR缺省值, GL_LIGHT0是(1.0, 1.0, 1.0, 1.0), 而其他光源是(0.0, 0.0, 0.0, 1.0)
- 位置是以齐次坐标的形式给定
 - 如果 $w = 1.0$, 指定的是一个有限位置
 - 如果 $w = 0.0$, 指定的是一个平行光源, 所给定的是入射光方向。缺省为(0.0, 0.0, 1.0, 0.0)

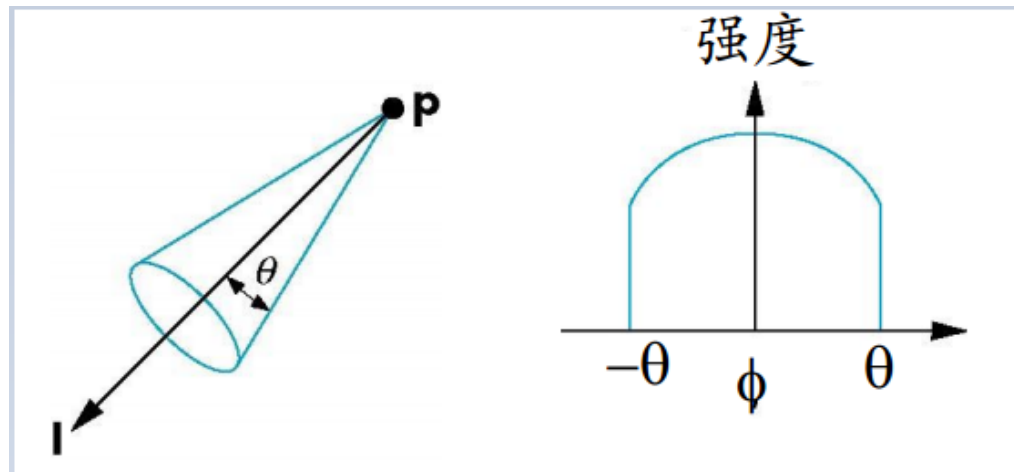
指定距离衰减项

- 即光强反比于距离的因子 $a + bd + cd^2$
 - 默认值: $a = 1.0, b = c = 0.0$
 - 改变方法

```
glLightf(GL_LIGHT0,  
GL_CONSTANT_ATTENUATION, 2.0);  
glLightf(GL_LIGHT0,  
GL_LINEAR_ATTENUATION, 1.0);  
glLightf(GL_LIGHT0,  
GL_QUADRATIC_ATTENUATION, 0.0);
```

聚光灯

- 应用glLightfv设置聚光灯的各项参数
 - 方向: GL_SPOT_DIRECTION, 缺省为 (0.0,0.0,-1.0)
 - 角度范围: GL_SPOT_CUTOFF, 缺省为 180.0
 - 衰减指数: GL_SPOT_EXPONENT, 缺省 0.0
 - 正比于 $\cos^\alpha \phi$



移动光源

- 光源是几何对象，它的位置或方向受模型视图矩阵的影响
- 把光源的位置和方向设置函数放置在不同的地方，可以达到不同的效果：
 - 和对象一起移动光源：所有的变换在光源位置和对对象定义之前调用
 - 固定对象，移动光源：先定义对象，进行变换后，再定义光源位置
 - 固定光源，移动对象：先定义光源位置，进行变换后，再定义对象
 - 分别移动光源和对象：采用矩阵堆栈

静止光源

```
glViewport(0,0,(GLsizei)w,(GLsizei)h);  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
if(w<=h)  
    glOrtho(-1.5,1.5,-1.5*h/w,1.5*h/w,-10.0,10.0);  
else  
    glOrtho(-1.5*w/h,1.5*w/h,-1.5,1.5,-10.0,10.0);  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
GLfloat light_pos[]={1.0,1.0,1.0,1.0};  
glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
```

光源与对象分别移动

```
static GLdouble spin;
void display(void){
    GLfloat light_pos[]={0.0,0.0,1.5,1.0};
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
    glPushMatrix();
    glRotated(spin,1.0,0.0,0.0);
    glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
    glPopMatrix();
    glutSolidTorus(0.275,0.85,8,15);
    glPopMatrix();
    glFlush();
}
```

光源与视点一起移动

为此需要在视图变换之前设置光源位置

记住：光源位置是存储在视点坐标系中，这是视点坐标系突现其作用的少数例子之一

```
GLfloat ligh_pos[]={0.0,0.0,0.0,1.0};  
glViewport(0,0,(GLsizei)w,(GLsizei)h);  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluPerspective(40.0,(GLfloat)w/h,1.0,100.0);  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glLightfv(GL_LIGHT0,GL_POSITION,light_pos);  
//... (to be continued)
```

光源与视点一起移动

```
//continued
static GLdouble ex,ey,ez,upx,upy,upz;
void display(void){
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
gluLookAt(ex,ey,ez,0.0,0.0,0.0,upx,upy,upz);
glutSolidTorus(0.275,0.85,8,15);
glPopMatrix();
glFlush();
}
```

材料属性

材料属性也是OpenGL状态的一部分，与改进Phong光照模型中的各项是匹配的

void glMaterial{if}(GLenum face, GLenum pname, TYPE param);

*void glMaterial{if}v(GLenum face, GLenum pname, TYPE *param);*

– face: GL_FRONT, GL_BACK,

GL_FRONT_AND_BACK

– pname:

- GL_AMBIENT, 缺省为(0.2, 0.2, 0.2, 1.0)
- GL_DIFFUSE, 缺省为(0.8, 0.8, 0.8, 1.0)
- GL_SPECULAR, 缺省为(0.0, 0.0, 0.0, 1.0)
- GL_SHININESS, 缺省为0.0, [0.0,128.0], 唯一的非向量参数
- GL_EMISSION, 缺省为(0.0, 0.0, 0.0, 1.0)

材料属性设置

应用glMaterial{if}[v]()设置

```
GLfloat ambient[]={0.2,0.2,0.2,1.0};  
GLfloat diffuse[]={1.0,0.8,0.0,1.0};  
GLfloat specular[]={1.0,1.0,1.0,1.0};  
GLint shine = 100;  
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);  
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);  
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);  
glMateriali(GL_FRONT, GL_SHININESS, shine);
```

材料自发射光项

- 在OpenGL中可以用材料的发射光项来模拟一个光源
- 该项的颜色不受任何其它光源或者变换的影响

```
GLfloat emission[]={0.0,0.3,0.3,1.0};  
glMaterialfv(GL_FRONT,GL_EMISSION,  
emission);
```


颜色指定与材料指定

- 通常当启用光照进行明暗处理后，原来的 `glColor*()` 命令失去原有的作用
- 如果调用了 `glEnable(GL_COLOR_MATERIAL)`，那么就会使光照模型中的几种光根据 `glColor*()` 中的指定确定颜色：

`void glColorMaterial(GLenum face, GLenum mode);`

– **face** 的取值 `GL_FRONT`, `GL_BACK` 与

`GL_FRONT_AND_BACK`(默认值)

– **mode** 的取值为 `GL_EMISSION`, `GL_AMBIENT`,

`GL_DIFFUSE`, `GL_SPECULAR` 与

`GL_AMBIENT_AND_DIFFUSE`(默认值)

颜色材料模式例子

```
glEnable(GL_COLOR_MATERIAL);  
glColorMaterial(GL_FRONT, GL_DIFFUSE);  
/* now glColor* changes diffuse reflection */  
glColor3f(0.2, 0.5, 0.8);  
/* draw some objects here */  
glColorMaterial(GL_FRONT, GL_SPECULAR);  
/* glColor* no longer changes diffuse reflection */  
/* now glColor* changes specular reflection */  
glColor3f(0.9, 0.0, 0.2);  
/* draw other objects here */  
glDisable(GL_COLOR_MATERIAL);
```

不需要使用glColorMaterial()时，确保禁用

多边形的明暗处理

- 对每个顶点进行明暗处理的计算
- 顶点的颜色变为顶点的明暗效果
- 默认状态下，多边形内部的颜色是顶点颜色的线性插值

`glShadeModel(GL_SMOOTH);`

- 如果调用了`glShadeModel(GL_FLAT);`，那么第一个顶点的颜色确定整个多边形的颜色