2018-2019年度第二学期 00106501

计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第二节 OpenGL中的明暗处理

在OpenGL中应用明暗处理的步骤

- 启用明暗处理功能，并选择模式
- 指定法向量
- 指定材料属性
- 指定光源
- 注意：本节内容仅适用于Compatibility profile（与明暗处理相关的函数，在Core profile中都被废弃了，需要自己编写顶点着色器和片元着色器实现明暗处理）

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()命令被忽略
- 必须单独激活每个光源
  - glEnable(GL_LIGHTi), i = 0, 1, …7
- 可以选择光照模型的参数
  - glLightModel{if}[v](参数，值)

glLightModel*()

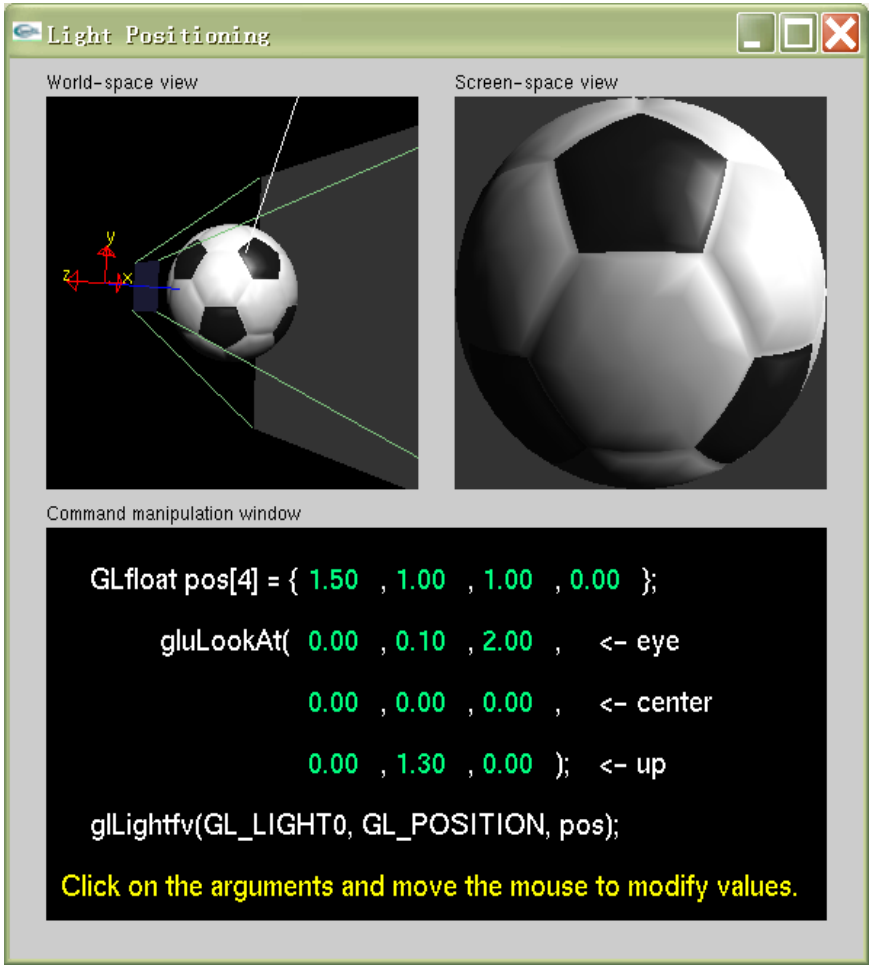
- 参数及其默认值，意义
  - GL_LIGHT_MODEL_AMBIENT，(0.2,0.2,0.2,1.0)，整个场景中的环境光强
  - GL_LIGHT_MODEL_LOCAL_VIEWER，0.0或GL_FALSE，在计算中不应用无穷远视点的假设简化计算
  - GL_LIGHT_MODEL_TWO_SIDED，0.0或GL_FALSE，单独对多边形的两面进行明暗处理
  - GL_LIGHT_MODEL_COLOR_CONTROL，GL_SINGLE_COLOR，镜面光是否与漫反射和环境光分开计算

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)可以使OpenGL自动进行单位化，当然是以损失效率为代价

光源位置定位的Tutor

- Nate Robin's lightposition tutor



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
- 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

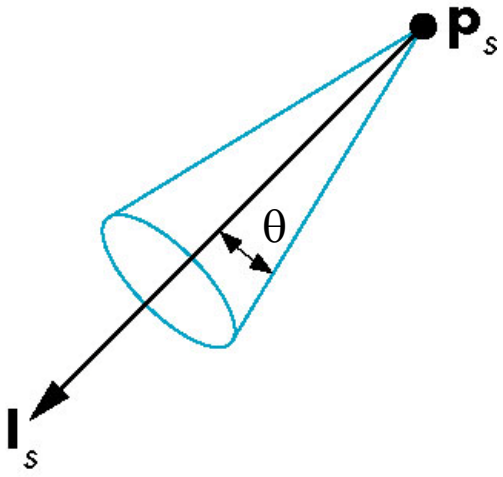
距离项的指定

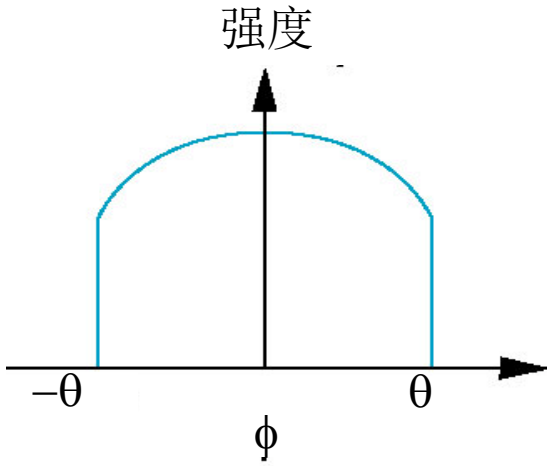
- 即光强反比于距离的因子 $a + bd + cd^2$
  - 默认值：$a = 1.0, b = c = 0.0$
  - 改变方法

```
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
  - 角度范围：GL_SPOT_CUTOFF
  - 衰减指数：GL_SPOT_EXPONENT
    - 正比于$\cos^{\alpha}\phi$





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中也可以定义一个有时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);
```

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

静态光源代码

```
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);
```

光源与对象分开移动

```
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();
}
```

与视点一起移动光源

- 为此需要在视图变换之前设置光源位置
  - 记住：光源位置是存贮在视点坐标系中，这是视点坐标系实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)
```

```
//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();
}
```

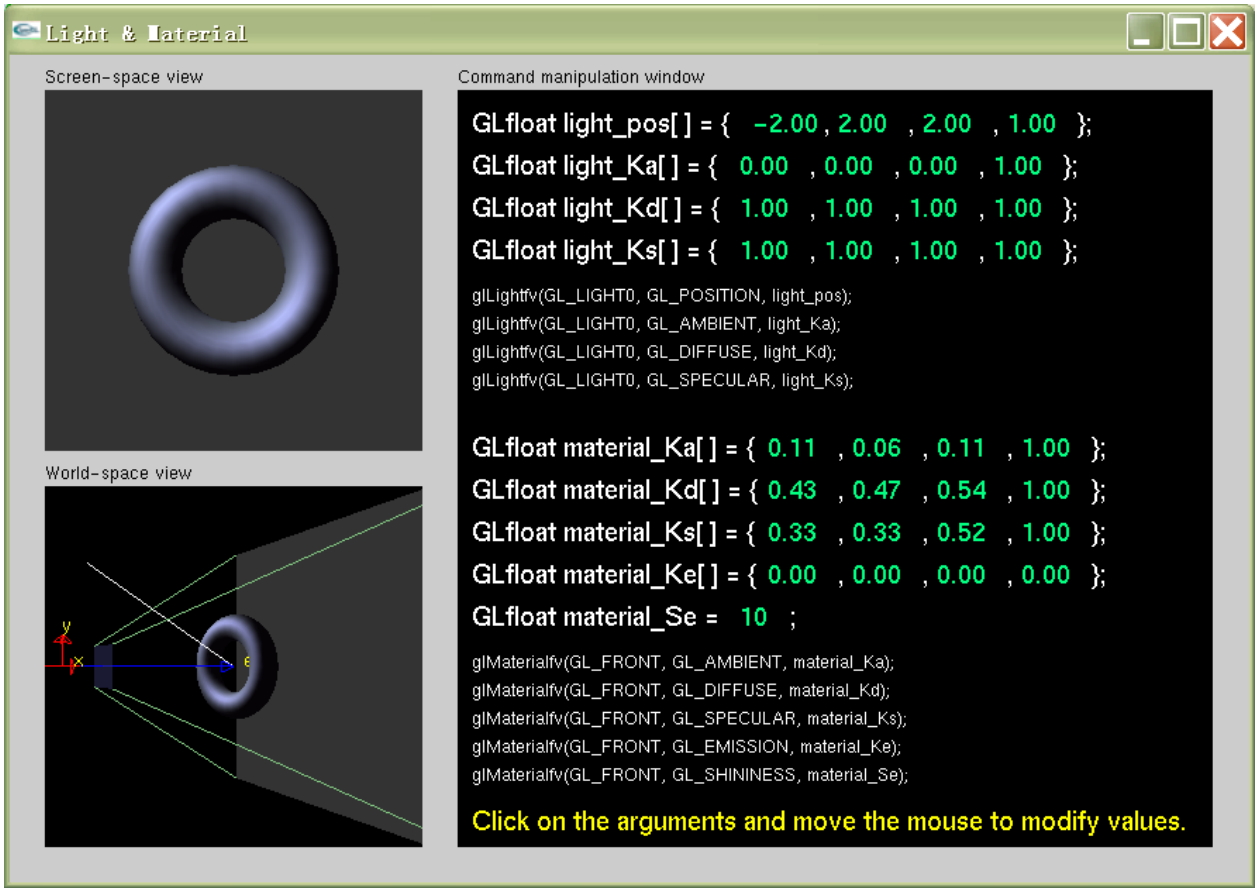
材料属性

- 材料属性也是OpenGL状态的一部分，与简单光照模型中的各项是匹配的
- 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);
```

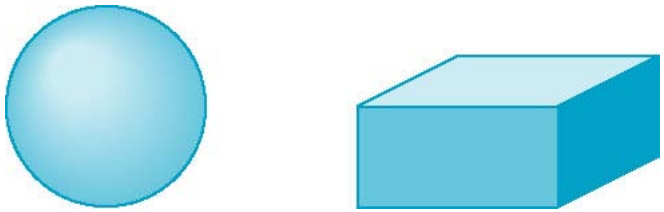
材料属性设置的Tutor

- Nate Robin's lightmaterial tutor

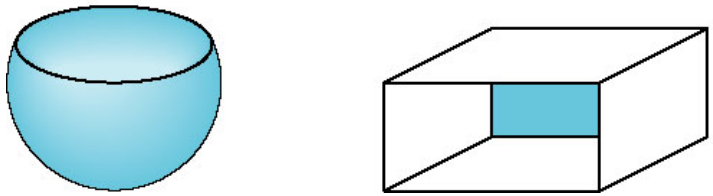


前面与后面

- 默认状态下只是把对象的前面进行明暗处理，对于凸对象这种处理结果是正确的
- 如果设置进行两面光照，那么OpenGL就会对曲面的双面进行明暗处理
- 每一面都可以具有自己的属性，面是用在glMaterialf()中的GL_FRONT, GL_BACK, 或者GL_FRONT_AND_BACK指定的



后面是不可见的



后面是可见的

发射项

- 在OpenGL中可以用材料的发射项来模拟一个光源
- 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);
```

用颜色指定代替材料属性指定

- 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：

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

其中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)

透明效果

- 材料属性是利用RGBA值指定的
- A值用来使表面透明
- 默认状态下不管A的值是多少，所有表面都是不透明的
- 后面我们会讲到如何激活融合(blending)功能以及这种功能的应用

效率

- 由于材料属性为状态的一部分，因此如果对许多表面采用大量的不同材料，那么系统的行为就会大打折扣
- 可以通过定义一个材料结构，利用它在初始化时设置所有材料，从而净化代码

```
typedef struct materialStruct {
  GLfloat ambient[4];
  GLfloat diffuse[4];
  GLfloat specular[4];
  GLfloat shineness;
} MaterialStruct;
```

- 这样可以通过指针选择一种材料

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

多边形的法向

- 多边形具有单个法向量
  - 应用简单光照模型在每个顶点处的明暗处理计算几乎完全相同
  - 对于无限远视点(默认)或者没有镜面光时，所有多边形的颜色一样
- 考虑球的模型
- 希望在每个顶点处具有不同的法向
  - 这个概念在数学上不太“正确”

光滑明暗处理

- 在每个顶点处设置新的法向
- 对于球的模型，这很容易做到
  - 如果球心在原点，那么n = p
- 现在进行光滑明暗处理
- 注意轮廓线处的结果

Thanks for your attention!