

计算机图形学

Computer Graphics

OpenGL 中的光照模型

福州大学数计学院 软件工程系 陈昱

目的

- 学习 OpenGL 中光照模型的使用，包括：
 - 法向量的定义
 - 创建、定位和启用光源
 - 选择光照模型
 - 定义物体的材质
 - OpenGL 的反射计算公式
- 学习 OpenGL 中着色方法的使用

OpenGL 中的光照模型

场景中增加光照的步骤

1. 定义每个物体的每个顶点的法向量
 - 这些法线决定了物体相对于光源的方向
2. 创建和选择一个或多个光源，并设置它们的位置
3. 设置光照模型（lighting model）
 - 它定义了全局环境光的层次以及观察者的有效位置（便于进行光照计算）
4. 定义场景中物体的材质属性

例：渲染光照下的球体

```
#include <GL/glut.h>
```

```
#include <stdlib.h>
```

```
/* Initialize material property, light source, lighting model,  
 * and depth buffer.
```

```
*/
```

```
void init(void)
```

```
{
```

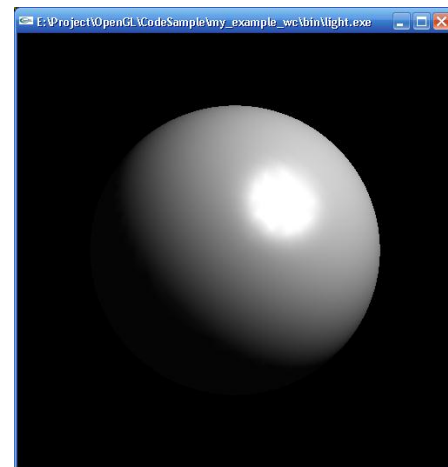
```
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
    GLfloat mat_shininess[] = { 50.0 };
```

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

```
    GLfloat white_light[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
    GLfloat lmodel_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
```



材质设定

```
glClearColor (0.0, 0.0, 0.0, 0.0);
```

```
glShadeModel (GL_SMOOTH);
```

```
glMaterialfv(GL_FRONT, GL_SPECULAR,  
             mat_specular);
```

```
glMaterialfv(GL_FRONT, GL_SHININESS,  
             mat_shininess);
```

光源设定

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

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

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

```
glLightModelfv(GL_LIGHT_MODEL_AMB  
IENT, lmodel_ambient);
```

启用光源

```
glEnable(GL_LIGHTING);  
glEnable(GL_LIGHT0);  
glEnable(GL_DEPTH_TEST);  
}
```


显示函数

```
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT);
    glutSolidSphere (1.0, 20, 16);
    glFlush ();
}
```

投影和视口设定

```
void reshape (int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
            1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
    else
        glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
            1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

主函数

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB |
        GLUT_DEPTH);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

1、定义法向量

定义法向量

- 物体的法向量决定了它相对于光源的方向
- 对于物体的每个顶点，OpenGL 使用法线判断这个顶点从每个光源所反射的光线数量
- OpenGL 中既可以为多边形指定一条共同的法线，也可以为多边形的每个顶点分别指定一条法线

定义法向量

- `glNormal3{bsidf} (Type nx,Type ny,Type nz)`
- `glNormal3{bsidf}v (constType *v)`
- 设定当前的法向量，成为下一个 `glVertex*()` 定义的顶点的法向量
- 除了顶点之外，不能为其他地方分配法线

定义法向量

- 范例：

```
glBegin(GL_POLYGON);  
    glNormal3fv(n0);  
    glVertex3fv(v0);  
    glNormal3fv(n1);  
    glVertex3fv(v1);  
    glNormal3fv(n2);  
    glVertex3fv(v2);  
glEnd();
```
- 在刚才的程序中，定义球体的法线是在 `glutSolidSphere()` 函数内完成的

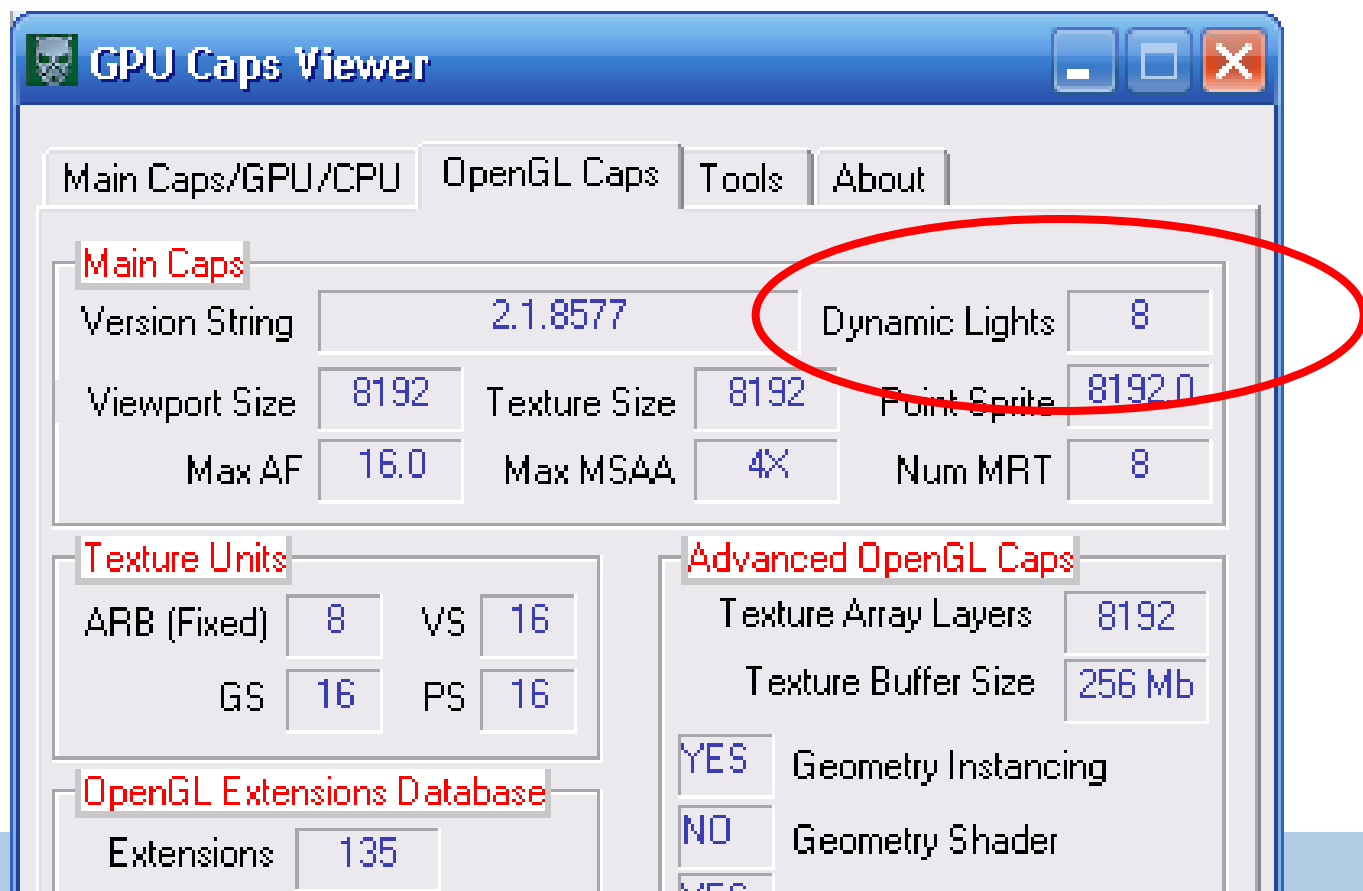
注意：

- 为了进行正确的光照计算，表面法线必须为单位长度
- 我们还必须保证对物体所进行的模型视图变换没有对表面法线进行缩放
- 启用 `glEnable(GL_NORMALIZE)`，OpenGL 可以在变换时自动对法向量进行规范化

2、创建、定位和启用光源

启用 OpenGL 的光源功能

- OpenGL 提供八个光源，名称分别为 GL_LIGHT0, ..., GL_LIGHT7



启用光源

- 若要使用某个光源，你必须在设定其属性后，用 `glEnable()` 函数来启动它，
- 例如：`glEnable(GL_LIGHT0);`
将启动光源 `GL_LIGHT0` 的功能
- 在刚才的程序中，只指定了一个白色的光源，其他光源的默认颜色为黑色

光源的创建

- 光源的属性通过 `glLight` 函数设定：

`void glLight{if} (GLenum light, GLenum pname, Type param)`

`void glLight{if}v (GLenum light, GLenum pname, Type* param)`

第一个函数用来指定数值型的属性

第二个函数用来指定数组型的属性

参数 *light* 选择光源 (`GL_LIGHT0`, ..., 或 `GL_LIGHT7`)

参数 *pname* 选取欲设定的参数，如颜色、种类、衰减率、位置、方向等等

参数 *param* 属性值

光源的颜色与强度

- 颜色由RGBA体现，强度由RGB算得的亮度决定

参数名称	默认值	意义
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	光源的环境光强度
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) or (0.0, 0.0, 0.0, 1.0)	光源的漫反射光强度。0 号光源预设为白光，其余光源预设为不发光。
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) or (0.0, 0.0, 0.0, 1.0)	光源的镜面反射光强度。0 号光源预设为白光，其余光源预设为不发光。

例：设定光源的各种反射光的颜色与强度

```
GLfloat light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };  
                          /* RGBA */  
GLfloat light_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };  
GLfloat light_specular[] = { 0.0, 0.0, 0.5, 1.0 };  
  
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);  
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);  
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
```

GL_AMBIENT

- GL_AMBIENT 参数表示一个特定的光源在场景中所添加的环境光的 RGBA
- 在默认情况下是不存在环境光的
 - 默认值 (0.0, 0.0, 0.0, 1.0)

GL_DIFFUSE

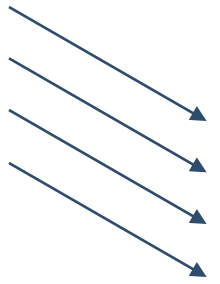
- GL_DIFFUSE 效果接近我们所想象的“光的颜色”，它定义了特定光源在场景中所添加的漫反射光的 RGBA
- GL_LIGHT0 默认的 GL_DIFFUSE 为 (1.0, 1.0, 1.0, 1.0)，产生一种明亮的白光
- 其余光源默认值是 (0.0, 0.0, 0.0, 1.0)，预设设为不发光

GL_SPECULAR

- GL_SPECULAR 影响物体上镜面高光的颜色
- 在现实世界中，玻璃瓶这样的物体具有和照射它表面的光线颜色相同的镜面反射颜色（通常是白色）
- 因此，通常将 GL_SPECULAR 参数值和 GL_DIFFUSE 设置成一样
- GL_LIGHT0 默认的 GL_SPECULAR 为 (1.0, 1.0, 1.0, 1.0)，其余光源默认值是 (0.0, 0.0, 0.0, 1.0)

光源类型

- OpenGL 提供以下三种灯光类型：

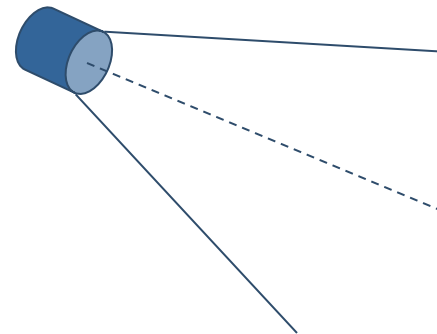
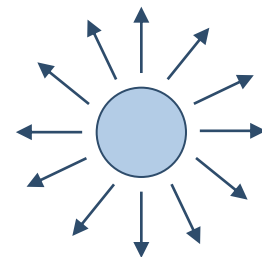


- 平行光源

- directional light，方向性光源
- 光线是平行的，朝一个方向照射，而且光的强度不因距离而改变，如室外的阳光
- 如果光源位于无穷远处，当光线到达物体表面时，可以认为所有光线都是平行的

光源类型

- 点光源 (point light)
 - 理想的点光源会向各个方向均匀发射光线
 - 而且光线的强度随距离加长而减弱
 - 如灯泡
- 聚光灯 (spot light)
 - 把光发射到一个很窄的角度范围里，而且光强随距离加长而减弱。此外，圆锥体边缘的光强也比较暗
 - 如台灯



光源类型

- 前述的点光源与探照光源合称为位置光源（positional light）
- 平行光源和位置光源是靠下面的参数来区分：

参数名称	默认值	意义
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	灯光的照射方向 或所在位置

光源类型

- 假定 `GL_POSITION` 的值是 (x, y, z, w)
- 若 $w = 0$ ，则灯光是平行光源，而且照射的方向是向量方向 (x, y, z)
- 若 $w \neq 0$ ，则光源是位置光源，而且光源是位于齐次坐标 (x, y, z, w)
- 从上面 `GL_POSITION` 的默认值知，默认的灯光是平行光源，而且朝 $(0, 0, 1)$ 方向照射

设置点光源

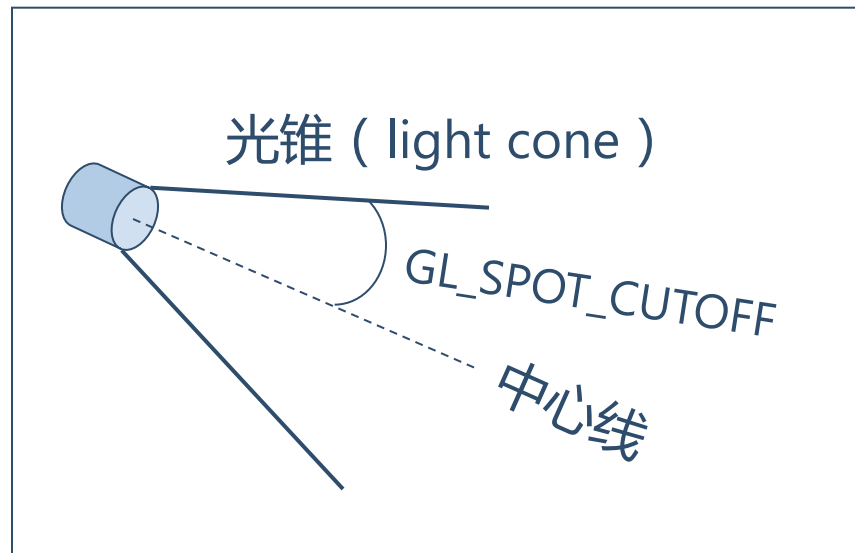
- 点光源只需设置 $w \neq 0$
- 下面的代码设置在 (0.0, 0.0, 10.0) 位置上的点光源

```
GLfloat light_position[] =  
    { 0.0, 0.0, 10.0, 1.0 };  
glLightfv(GL_LIGHT0, GL_POSITION,  
    light_position);
```

设置聚光灯

属性名称	默认值	意义
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	聚光灯照射方向向量 (x,y,z)
GL_SPOT_CUTOFF	180.0	聚光灯照射的切角
GL_SPOT_EXPONENT	0.0	聚光灯的聚光指数

- 注意：
- GL_SPOT_CUTOFF 的值只能设在 0 到 90 度之间
- GL_SPOT_EXPONENT 的值愈大，聚光度愈强，反之愈弱



光的衰减

- 对于现实世界的光照，随着光源距离的增加，光的强度也随之衰减
- OpenGL 可对位置光源所发出的光进行衰减，把光源的强度乘以衰减因子，对它进行衰减：

$$\text{衰减因子} = \frac{1}{k_c + k_l d + k_q d^2}$$

- d 为光源和顶点之间的距离

光的衰减

$$\text{衰减因子} = \frac{1}{k_c + k_l d + k_q d^2}$$

- 系数 K_c K_l K_q 可以通过 *glLight*()* 进行设置

属性名称	默认值	符号
GL_CONSTANT_ATTENUATION	1.0	k_c
GL_LINEAR_ATTENUATION	0.0	k_l
GL_QUADRATIC_ATTENUATION	0.0	k_q

注意

- OpenGL 默认的光源并无衰减（衰减因子公式=1）
- 光源的环境光、漫反射光和镜面光的反射强度都会进行衰减
 - 只有全局环境光的强度没有衰减
- 方向性光源（平行光源）没有衰减
- 衰减需要在每次计算颜色时再进行一次除法，因此会影响性能

例：

设定强度与距离成反比的点光源

```
GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };  
glLightfv(GL_LIGHT0, GL_POSITION, light_position);  
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.0);  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
```

设定强度与距离平方成反比的点光源

```
GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };  
glLightfv(GL_LIGHT0, GL_POSITION, light_position);  
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.0);  
glLightf(GL_LIGHT0, GL_QUADRTIC_ATTENUATION, 1.0);
```

控制光源的位置

```
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };  
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- OpenGL 处理光源位置和方向的方式类似于处理几何图元的位置
- 控制光源的矩阵变换和控制图元的矩阵变化相同
- 当我们调用 `glLight*()` 函数指定一个光源的位置和方向时，这个光源的位置或方向将根据当前的模型视图矩阵进行变换
 - 投影变换对光源的位置和方向没有影响

3、设定光照模型属性

设定光照模型属性

- OpenGL 的光照模型设定包含下面4个部分：
 - 全局的环境光强度
 - 视点是否位于场景内还是位于无限远处
 - 物体正面和背面是否应执行不同的光照计算
 - 是否要把镜面反射光与环境光及漫反射光分开处理，并在纹理贴图之后才计算

设定光照模型属性

- `glLightModel*()` 函数用于指定光照模型的所有属性

`void glLightModel{if} (GLenum pname, Type param)`

`void glLightModel{if}v (GLenum pname, Type*param)`

- `glLightModel*()` 函数具有两个参数：
 - 光照模型属性
 - 这个属性所需要设置的值

全局环境光

属性名称 `GL_LIGHT_MODEL_AMBIENT`

默认值 `(0.2, 0.2, 0.2, 1.0)`

- 除了用光源来提供环境光外，你也可以用这个光照模型的属性来提供全局的环境光
- 全景环境光的默认值使得即使没有光源，场景也不会区黑一片而看不到任何东西

例：设定全局的环境光颜色

```
GLfloat lmodel_ambient[] = { 0.5, 0.5, 0.5,  
    1.0 };
```

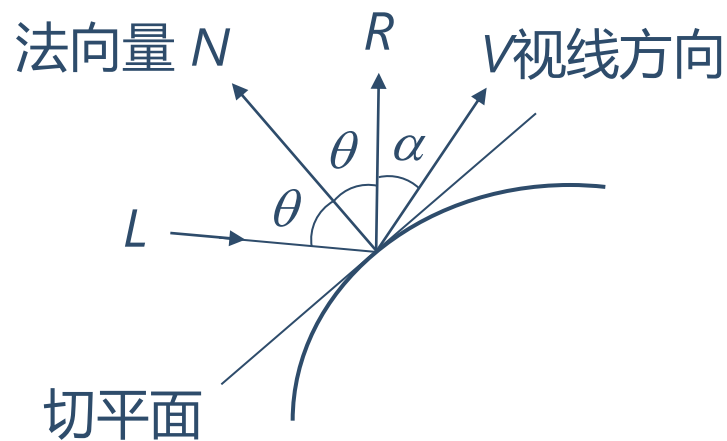
```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  
    lmodel_ambient);
```

- 注意：设置全局环境光不能使用非数组版本的函数

观察点的位置设定

- 属性名称 GL_LIGHT_MODEL_LOCAL_VIEWER
- 默认值 0.0 或 GL_FALSE

观察者和物体的相对位置会影响镜面反射
然而 OpenGL 在计算镜面反射时，默认地把观察者看作位在无限远处，使得每个顶点上的 α 角都相同，通过这个简化方法来减少计算量



$$I_s = I_p K_s (V \cdot R)^n$$

观察点的位置设定

- 但如此一来，真实感就降低了
- 可以把这个属性改成 `GL_TRUE`，将计算时的观察点设为局部（真实位置）来避免上面的问题
- 例：
`glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);`
- 不过，由于每个顶点都必须计算它和观察点的方向，性能会受到影响

双面光照

- 属性名称 `GL_LIGHT_MODEL_TWO_SIDE`
- 默认值 `0.0` 或 `GL_FALSE`
- 多面体的每一个面都有正面与背面之分。
OpenGL 通常只计算正面的反射光
- 但有的时候我们需要计算背面的反射光，例如：一个剖开的半球，因为看得到其内部的面，因此需计算它们背面的反射光

双面光照

- 在这个情况下，我们必须把这个属性设成 `GL_TRUE`，即调用函数：
- `glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);`
- OpenGL 将会反转表面法线的方向，表示背面多边形的法线
- 但额外的计算将使双面光照比默认的单面光照要慢一些

镜面反射光的计算顺序

- 属性名称 `GL_LIGHT_MODEL_COLOR_CONTROL`
- 默认值 `GL_SINGLE_COLOR`
- 在默认的情况下，OpenGL 先计算出环境光、漫反射光、镜面反射光 and 自发光，把它们加起来得到反射光的数值，然后再计算出纹理（ texture mapping ）的效果
- 有时这样的计算顺序会使得镜面反射的高光变得比较暗淡或纹理效果不佳

镜面反射光的计算顺序

- 为了避免这个问题，我们可以调用下面的函数把镜面反射高光的计算移到最后阶段来：
 - `glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR);`
- 若不使用纹理，我们不需要改变这个属性值。若要恢复默认值只要调用下面的函数即可：
 - `glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SINGLE_COLOR);`

启用光照

- 在 OpenGL 中需要显式启用或禁用光照
 - 如果未启用光照，当前的颜色就简单地映射到当前的顶点，不存在任何涉及法线、光源、光照模型和材质属性的计算
- 启用光照的方法：
`glEnable(GL_LIGHTING)`
- 禁用光照的方法：
`glDisable(GL_LIGHTING)`

4、定义物体的材质

设定物体的材质

- 除了光源的特性外，物体材质（material）的特性也会影响其反射的效果，物体材质可用函数`glMaterial*()`来设定：

`void glMaterial{if} (GLenum face, GLenum pname, Type param)`

`void glMaterial{if}v (GLenum face, GLenum pname, Type*param)`

参数`face` 选择 `GL_FRONT`, `GL_BACK`, 或
 `GL_FRONT_AND_BACK` 来应用材质。

参数`pname` 选取欲设定的材质属性，如颜色、高光强度、闪亮
 度、发光度

参数`param` 属性值

材质颜色

属性名称	默认值	意义
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	材质的环境光颜色
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	材质的漫反射光颜色

GL_AMBIENT_AND_DIFFUSE

- 大部分的物体具有相同的环境光颜色和漫反射光颜色，因此可以用属性 GL_AMBIENT_AND_DIFFUSE 来同时设定这两种颜色

材质颜色

- 例如：

```
GLfloat amt_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };  
glMaterialfv(GL_FRONT_AND_BACK,  
             GL_AMBIENT_AND_DIFFUSE, amt_amb_diff);
```

镜面反射光颜色

属性名称	默认值	意义
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	材质的镜面反射光颜色
GL_SHININESS	0.0	材质的镜面指数

- 镜面指数 GL_SHININESS 用来控制高光的集中范围，它的值必须介于 0.0 到 128.0 之间
- 值愈大，高光范围愈集中，亮度也愈强，反之，高光范围愈扩散，亮度也愈弱

例：设定物体高光的颜色与集中度

```
GLfloat mat_specular[] =  
    { 1.0, 1.0, 1.0, 1.0 };  
GLfloat low_shiness = 5.0;  
glMaterialfv(GL_FRONT, GL_SPECULAR,  
             mat_specular);  
glMaterialf(GL_FRONT, GL_SHININESS,  
            low_shiness);
```

发光颜色

属性名称	默认值	意义
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	材质的发光颜色
<ul style="list-style-type: none">光源在 OpenGL 中是一种隐形物体若想在场景中显示光源的形状（如灯泡），我们必须先制作灯泡的 3D 模型，摆在光源位置上，然后设定其材质属性 GL_EMISSION 来模拟出发光体的样子		

例：设定材质的发光颜色

- `Glfloat mat_emission[] =
 { 0.3, 0.3, 0.2, 0.0 };`
- `glMaterialfv(GL_FRONT, GL_EMISSION,
 mat_emission);`

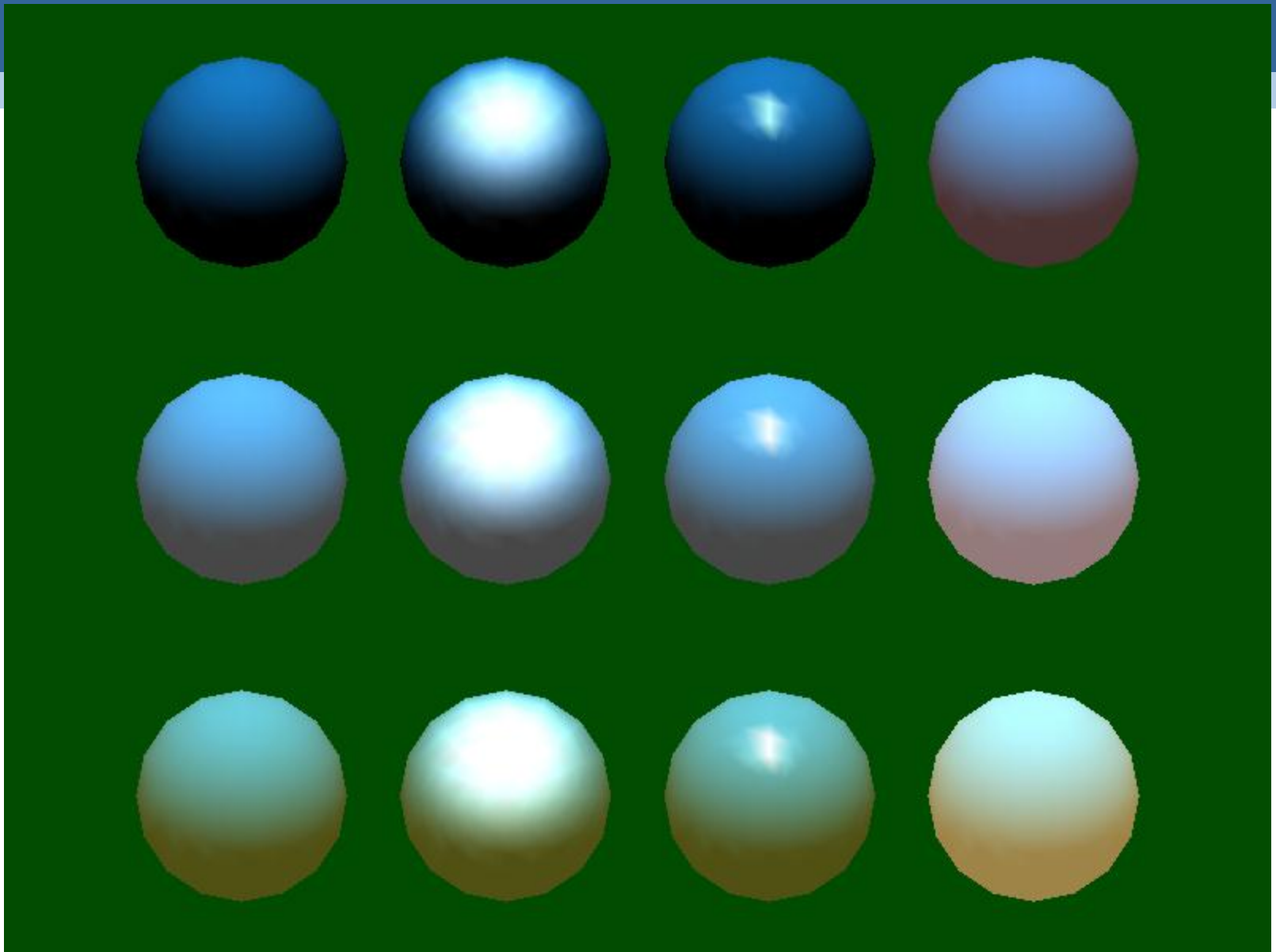
例：改变物体的材质（redbook 'material.c'）

```
void display(void)
{
    GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
    GLfloat mat_ambient_color[] = { 0.8, 0.8, 0.2, 1.0 };
    GLfloat mat_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat no_shininess[] = { 0.0 };
    GLfloat low_shininess[] = { 5.0 };
    GLfloat high_shininess[] = { 100.0 };
    GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
/* draw sphere in first row, first column
 * diffuse reflection only; no ambient or specular
 */
glPushMatrix();
glTranslatef (-3.75, 3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
```

```
/* draw sphere in first row, second column
* diffuse and specular reflection; low shininess; no ambient
*/
glPushMatrix();
glTranslatef (-1.25, 3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
```



redbook 'teapot.c'



A = (0.0215, 0.1745, 0.0215)
D = (0.07568, 0.61424, 0.07568)
S = (0.633, 0.727811, 0.633), shininess = 0.6



A = (0.135, 0.2225, 0.1575)
D = (0.54, 0.89, 0.63),
S = (0.316228, 0.316228, 0.316228), shininess = 0.1



A = (0.05375, 0.05, 0.06625)
D = (0.18275, 0.17, 0.22525),
S = (0.332741, 0.328634, 0.346435), shininess = 0.3



A = (0.25, 0.20725, 0.20725)
D = (1, 0.829, 0.829),
S = (0.296648, 0.296648, 0.296648), shininess = 0.088



A = (0.1745, 0.01175, 0.01175)
D = (0.61424, 0.04136, 0.04136),
S = (0.727811, 0.626959, 0.626959), shininess = 0.6



A = (0.1, 0.18725, 0.1745)
D = (0.396, 0.74151, 0.69102),
S = (0.297254, 0.30829, 0.306678), shininess = 0.1



$A = (0.329412, 0.223529, 0.027451)$
 $D = (0.780392, 0.568627, 0.113725)$
 $S = (0.992157, 0.941176, 0.807843), \text{ shininess} = 0.21794872$



$A = (0.2125, 0.1275, 0.054)$
 $D = (0.714, 0.4284, 0.18144)$
 $S = (0.393548, 0.271906, 0.166721), \text{ shininess} = 0.2$



$A = (0.25, 0.25, 0.25)$
 $D = (0.4, 0.4, 0.4)$
 $S = (0.774597, 0.774597, 0.774597), \text{ shininess} = 0.6$



$A = (0.19125, 0.0735, 0.0225)$
 $D = (0.7038, 0.27048, 0.0828)$
 $S = (0.256777, 0.137622, 0.086014), \text{ shininess} = 0.1;$



$A = (0.24725, 0.1995, 0.0745)$
 $D = (0.75164, 0.60648, 0.22648)$
 $S = (0.628281, 0.555802, 0.366065), \text{ shininess} = 0.4$



$A = (0.19225, 0.19225, 0.19225)$
 $D = (0.50754, 0.50754, 0.50754)$
 $S = (0.508273, 0.508273, 0.508273), \text{ shininess} = 0.4$



$A = (0.0, 0.0, 0.0)$

$D = (0.01, 0.01, 0.01)$

$S = (0.50, 0.50, 0.50), \text{ shininess} = 0.25$



$A = (0.0, 0.1, 0.06)$

$D = (0.0, 0.50980392, 0.50980392)$

$S = (0.50196078, 0.50196078, 0.50196078), \text{ shininess} = 0.25$



$A = (0.0, 0.0, 0.0)$

$D = (0.1, 0.35, 0.1)$

$S = (0.45, 0.55, 0.45), \text{ shininess} = 0.25$



$A = (0.0, 0.0, 0.0)$

$D = (0.5, 0.0, 0.0)$

$S = (0.7, 0.6, 0.6), \text{ shininess} = 0.25$



$A = (0.0, 0.0, 0.0)$

$D = (0.55, 0.55, 0.55)$

$S = (0.70, 0.70, 0.70), \text{ shininess} = 0.25$



$A = (0.0, 0.0, 0.0)$

$D = (0.5, 0.5, 0.0)$

$S = (0.60, 0.60, 0.50), \text{ shininess} = 0.25$



$A = (0.02, 0.02, 0.02)$

$D = (0.01, 0.01, 0.01)$

$S = (0.4, 0.4, 0.4), \text{ shininess} = 0.078125$



$A = (0.0, 0.05, 0.05)$

$D = (0.4, 0.5, 0.5)$

$S = (0.04, 0.7, 0.7), \text{ shininess} = 0.078125$



$A = (0.0, 0.05, 0.0)$

$D = (0.4, 0.5, 0.4)$

$S = (0.04, 0.7, 0.04), \text{ shininess} = 0.078125$



$A = (0.05, 0.0, 0.0)$

$D = (0.5, 0.4, 0.4)$

$S = (0.7, 0.04, 0.04), \text{ shininess} = 0.078125$



$A = (0.05, 0.05, 0.05)$

$D = (0.5, 0.5, 0.5)$

$S = (0.7, 0.7, 0.7), \text{ shininess} = 0.078125$



$A = (0.05, 0.05, 0.0)$

$D = (0.5, 0.5, 0.4)$

$S = (0.7, 0.7, 0.04), \text{ shininess} = 0.078125$

物体材质的简易设定方式

- 若只改变物体材质的单一属性时，我们可用以下的函数：

`void glColorMaterial (GLenum face, GLenum mode)`

参数`face` 选择GL_FRONT, GL_BACK, 或
GL_FRONT_AND_BACK来应用材质。

参数`mode` 选取欲设定的材质属性：
GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR,
GL_AMBIENT_AND_DIFFUSE (默认值),
GL_EMISSION

物体材质的简易设定方式

- 使用前必须先调用 `glEnable(GL_COLOR_MATERIAL)` 来启动 `glColorMaterial()` 的功能
- 接下来用函数 `glColor*()` 来改变所选择的材质属性
- 不再需要此功能时，可调用 `glDisable(GL_COLOR_MATERIAL)` 将其关闭

改变物体材质的单一属性

```
glEnable(GL_COLOR_MATERIAL);  
glColorMaterial(GL_FRONT, GL_DIFFUSE);  
glColor3f(0.2, 0.5, 0.8);  
/* draw some objects here with diffuse color (0.2,  
   0.5, 0.8) */  
  
glColorMaterial(GL_FRONT, GL_SPECULAR);  
glColor3f(0.9, 0.0, 0.2);  
/* draw other objects here with specular color (0.9,  
   0.0, 0.2) */  
  
glDisable(GL_COLOR_MATERIAL);
```

OpenGL 的反射计算公式

$$I_{\text{vertex}} = I_{em} + I_{alm} \times K_{am} + \sum_{j=0}^{n-1} \left(\frac{1}{k_c + k_l d + k_q d^2} \right)_j \times (\text{spotlight effect})_j \times [I_{al} \times K_{am} + \max(\vec{L} \cdot \vec{N}, 0) \times I_{dl} \times K_{dm} + \max(\vec{H} \cdot \vec{N}, 0)^{\text{shininess}} \times I_{sl} \times K_{sm}]_j$$

I_{em} : emission_{material} I_{alm} : ambient_{light model}

I_{al} : ambient_{light} K_{am} : ambient_{material}

I_{dl} : diffuse_{light} K_{dm} : diffuse_{material}

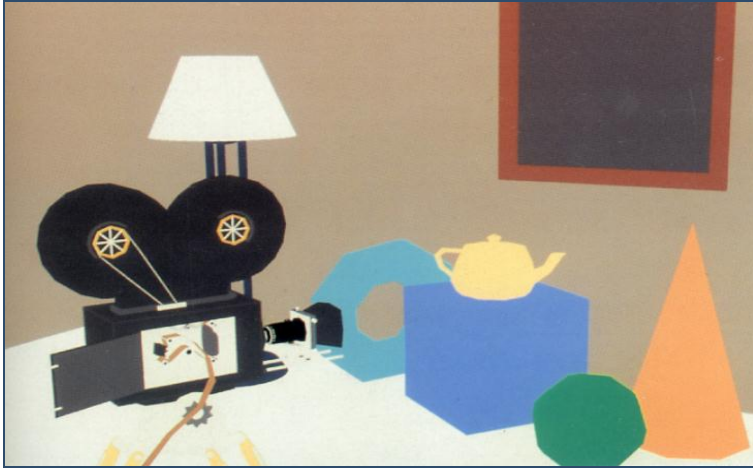
I_{sl} : specular_{light} K_{sm} : specular_{material}

OpenGL 中的着色方法

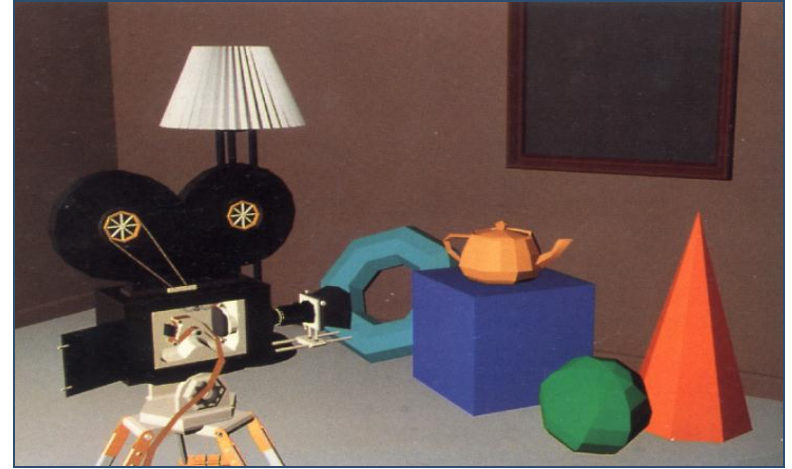
设定着色方法

- 要给 3D 对象模型着色，必须调用 `glShadeModel()` 来设定着色的方法
 - `void glShadeModel (GLenum mode)`
- 参数 `mode` 必须是下面两个常数值之一
 - `GL_FLAT`: 使用 Flat Shading
 - `GL_SMOOTH`: 使用 Gouraud Shading

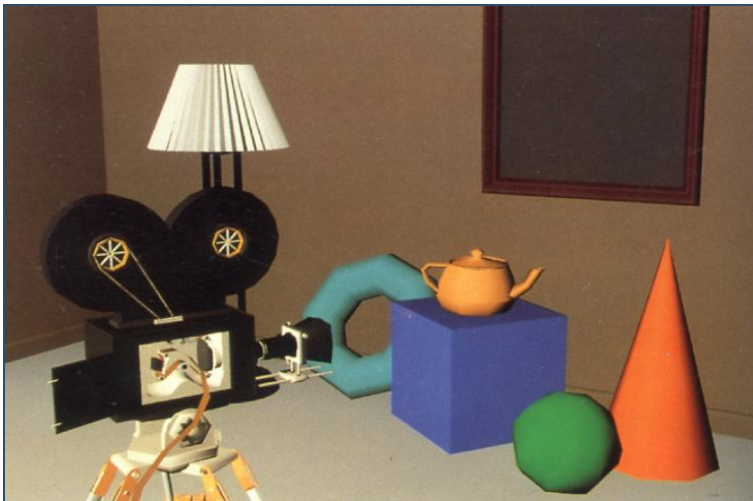
Shading Techniques



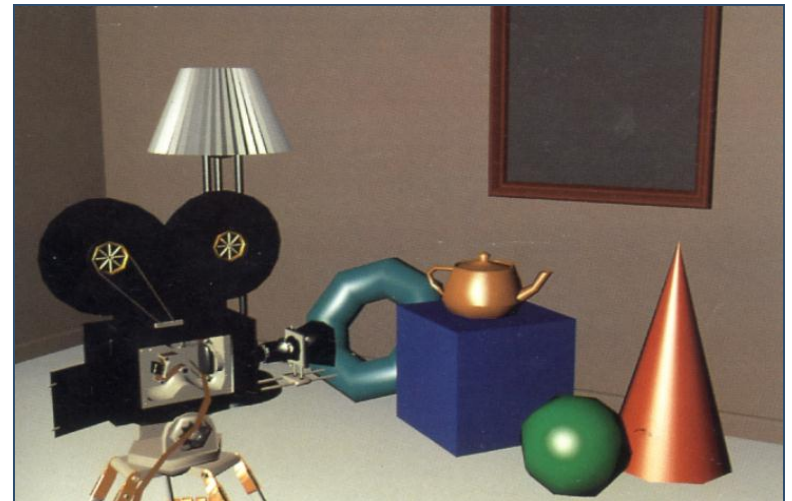
No Shading



Flat Shading



Gouraud Shading

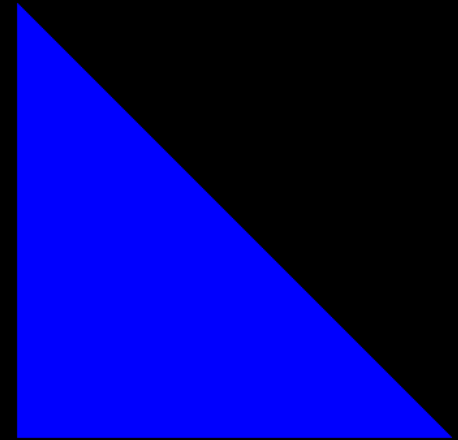


Phong Shading

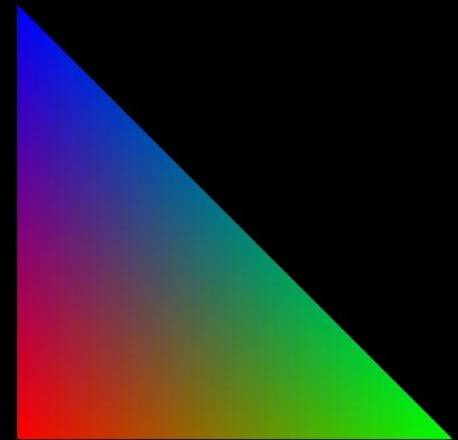
红皮书 smooth.c

```
void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_SMOOTH);
}
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);
    glBegin (GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0);
    glVertex2f (5.0, 5.0);
    glColor3f (0.0, 1.0, 0.0);
    glVertex2f (25.0, 5.0);
    glColor3f (0.0, 0.0, 1.0);
    glVertex2f (5.0, 25.0);
    glEnd();
    glFlush ();
}
```

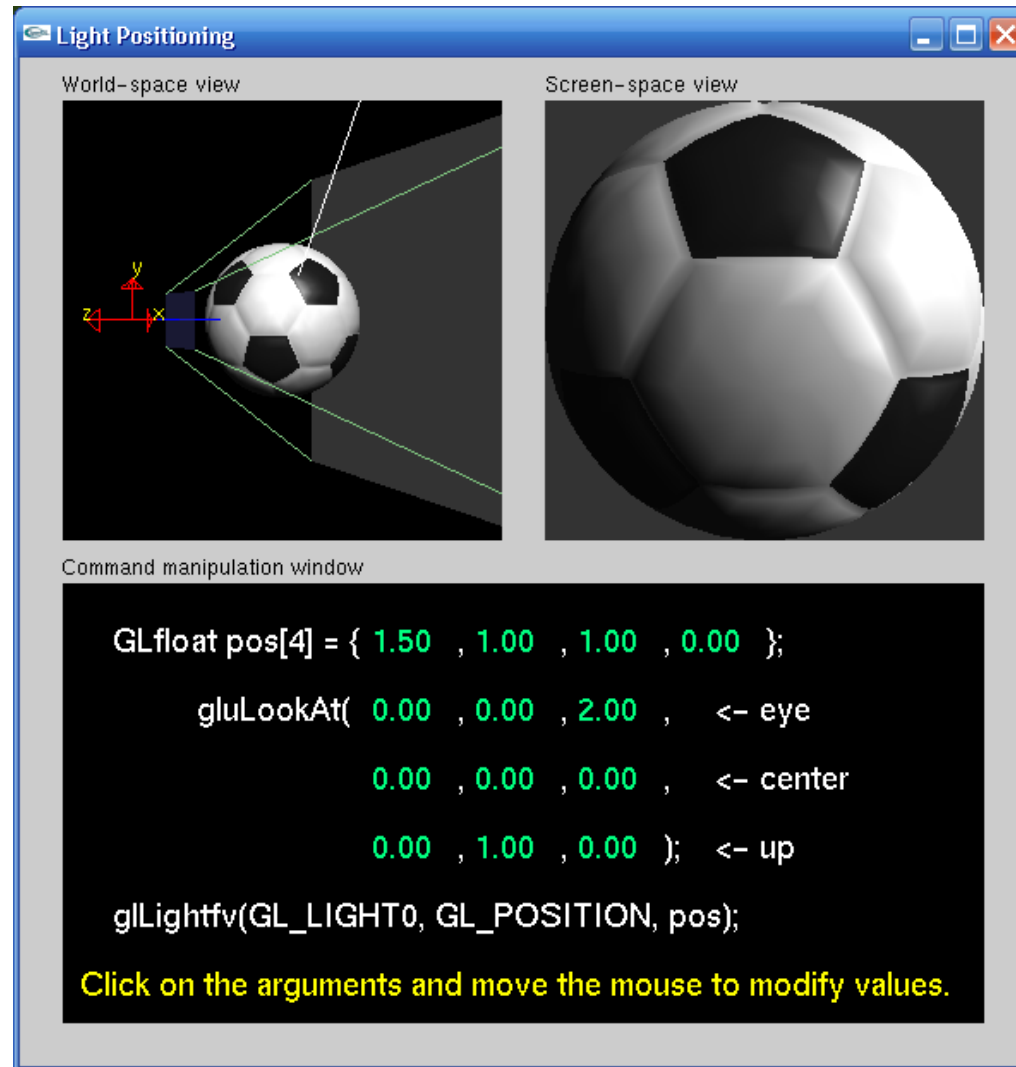
GL_FLAT



GL_SMOOTH



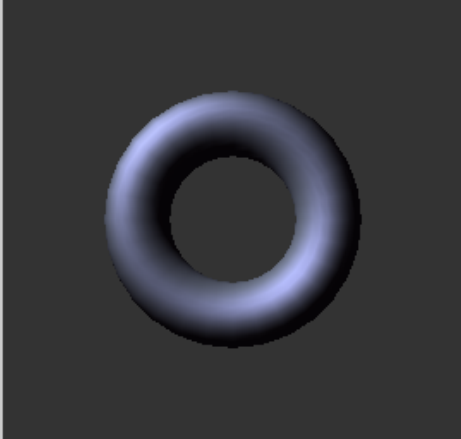
Nate Robin's Light Positioning Tutor



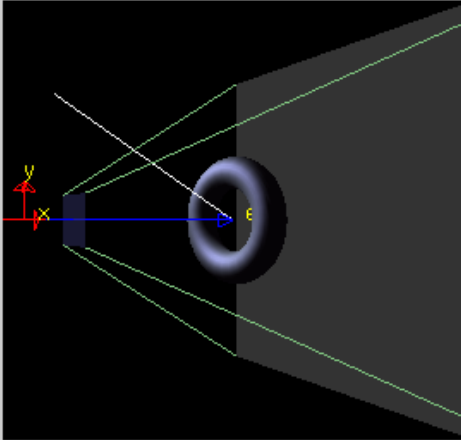
Nate Robin's Light & Material Tutor

Light & Material

Screen-space view



World-space view



Command manipulation window

```
GLfloat light_pos[] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[] = { 1.00 , 1.00 , 1.00 , 1.00 };

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[] = { 0.11 , 0.06 , 0.11 , 1.00 };
GLfloat material_Kd[] = { 0.43 , 0.47 , 0.54 , 1.00 };
GLfloat material_Ks[] = { 0.33 , 0.33 , 0.52 , 1.00 };
GLfloat material_Ke[] = { 0.00 , 0.00 , 0.00 , 0.00 };
GLfloat material_Se = 10 ;

glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);
```

Click on the arguments and move the mouse to modify values.

作业 hw5

- 请用 GLUT 的实体模型来绘制一张桌子，并在桌面上安排放置适当大小的一个 Utah 茶壶、一个圆球、与一个立方体。然后，**设定灯光与材质**，最后，把相机放在适当的位置以产生比较美观的画面。
- 本周内提交实验报告和 hw5