**opengl光照模型与面绘制算法**

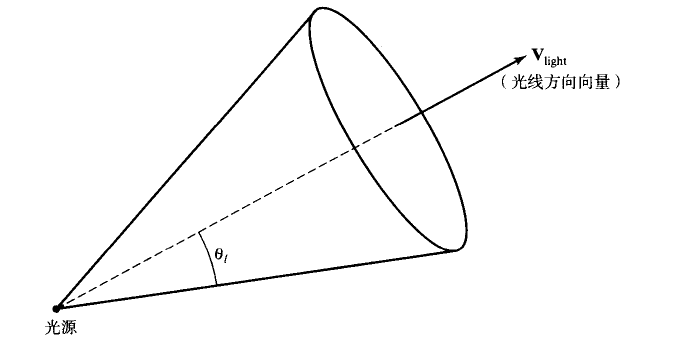
光照模型主要用于对象表面某光照位置的颜色计算。表面绘制算法是使用光照模型为对象的所有投影位置确定像素颜色

1）光源：任一发出辐射能量的对象称为一个光源，大多数的发光体仅作为场景中的光照源，但是在有些应用中，我们可能希望建立既是光源又是光反射体的对象。比如，在一个塑料球内放置一个灯泡，这样在球表面上既发光也反射光

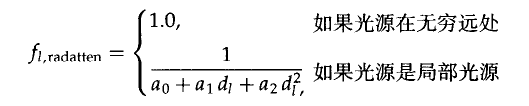
点光源：通过给出其位置和发射光颜色来定义

无穷远光源：使用一个颜色值和从该光源发出的一个固定光线方向来模拟一个无穷远距离光源，在光照计算中，仅需要一个发射方向的向量以光源颜色而不需要光源位置

方向光源：除了位置和颜色之外还需要确定一个向量方向和从该向量方向开始的角度范围



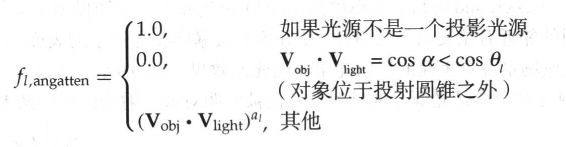
辐射强度衰减：衰减因子为1/(a0+a1\*d+a2\*d^2),但是对于无穷远处的光源不能使用强度衰减来计算，因为场景中的各点离很远的光源位置几乎都相等，因此：



角强度衰减：对于一个方向光源，可以按照从点光源位置出发的光强度角计算衰减，即可模拟一个光锥，沿其圆锥轴光强最大，离开圆锥轴时强度逐渐减弱，常用的方向光源角强度衰减函数是：



这种情况下还需要考虑几个特殊情况，如果光源不是方向光源（即不是投影光源）则没有角强度衰减，同样，如果一个对象位于投影光锥体外的任何地方，则它得不到该光源的光照:



表面光照效果：光照模型使用使用为表面设定的各种光学特性来计算表面的光照效果，这些特性包括透明度、颜色发射系数以及各种表面纹理参数

当光入射到不透明表面时，一部分被发射而一部分被吸收，入射光中北表面反射的总量依赖于材料类型，光亮的表面将大部分的入射光反射，而惠安的表面吸收较多的入射光，对于透明的表面，入射光的补一份穿过该材料

漫反射：对象的漫反射光的颜色称为该对象的颜色。例如，蓝色对象将白色光中的蓝色部分反射而将其他成分吸收，如果蓝色对象在红色光下观察，则它表现为黑色，因为所有入射光全部被吸收

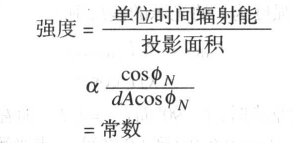
基本光照模型：

精确的光照模型按照入射光能量与对象的材料组成之间的交互结果进行计算，为了简化表面光计算，可使用前面泰伦的光照效果的物理过程近似表示，基本光照模型中的发光体一般限于点光源，然而，许多图形软件提供处理方向光源和扩展光源的函数。

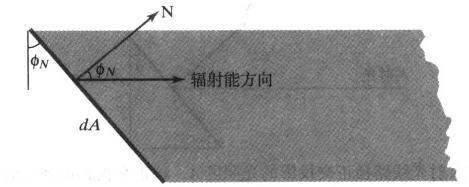
**Lambert漫反射模型**

该模型属于经验模型，主要用来简单模拟粗糙物体表面的光照现象。

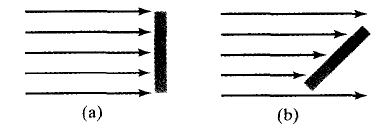
该定律表明：在与对象表面法向量夹角为¢N的方向上，每个面积为dA的平面单位所发散的光线与cos¢N成正比。该方向的光强度可用单位时间辐射能总量除以表面积在辐射方向的投影来计算



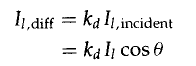
这样，对于朗伯反射，光强度在所有观察方向都相同

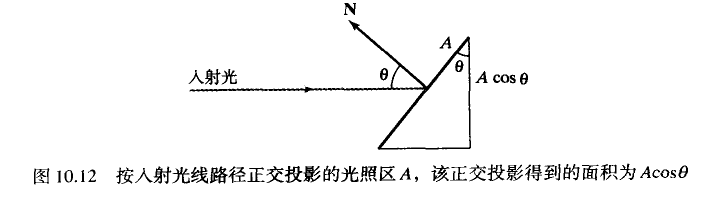


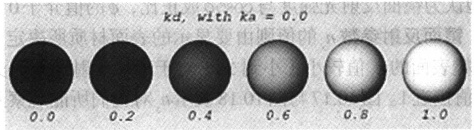
与表面相交的光线数与该表面投影到入射光方向的面积成正比，如下图：



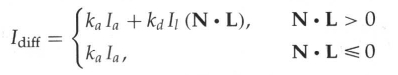
如果把入射光方向与表面法线方向之间的入射角称为θ，则垂直于光线方向的面片投影面积与cosθ成正比，因此一个强度为I的光源入射光总量为

 Kd数与0-1之间的常数

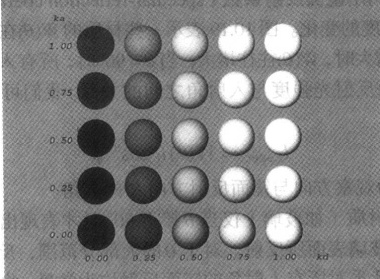




组合环境光和点光源的强度计算，可以得到一个的表面位置的全部漫反射表达。另外，许多图形软件包引入为每一表面指定的环境光发射系数Ka来修改环境光强度Ia，这使得我们能够简单地利用该参数来调节场景中的光照效果。使用参数Ka，可将单个点光源的全部漫反射公式表示为：



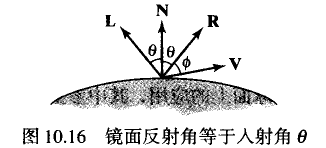
其中Ka和Kd都依赖于表面材质，对单色光照效果而言其值介于0和1之间



把环境光模型添加进来，最后，Lambert光照模型可写为：  
I= IaKa + Id Kdcosθ= IaKa + Id Kd(L·N)  
该模型包含环境光和漫反射光

**Phong镜面反射模型**

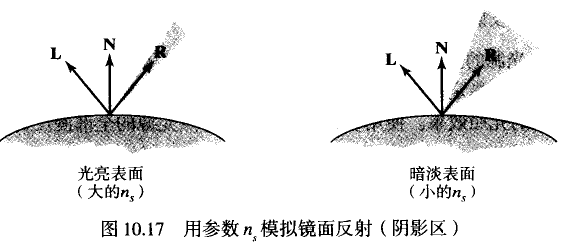
在光滑表面上看到的高光或镜面反射是由接近镜面反射角的一个会聚区域内入射光的全部或绝大部分称为反射光所导致。



对于一个理想的反射体（镜子），入射光仅在镜面反射方向有反射现象，此时仅当V和R重合时才能观察到反射光线（即¢=0）

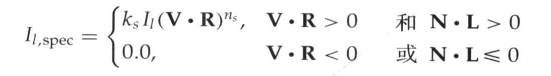
非理想反射体系统的镜面发射方向分布在向量R周围的有限范围内。较光滑表面的镜面反射范围较小，而粗糙的对象表面则有较大的镜面反射范围。

Phong Bui Tuong提出了镜面反射模型，该模型认为镜面反射光照强度与成正比，¢介于0-90之间。**镜面发射参数ns的值则由要显示的表面材质所决定。**光滑表面的ns较大（如100或更大），而粗糙表面的ns较小（小到1）。对于理想反射器ns是无限的。而粗糙对象表面。如粉笔或煤渣的ns值接近1





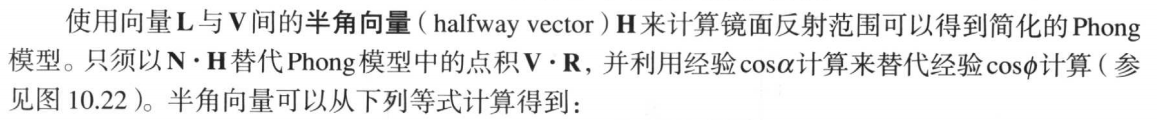


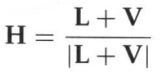


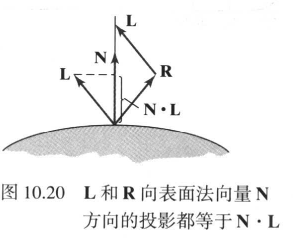
反射向量R的方向可通过向量L和N计算出来，可以通过点积N.L得到向量L在法向量方向上的投影，这也等于单位向量R在N方向上的投影

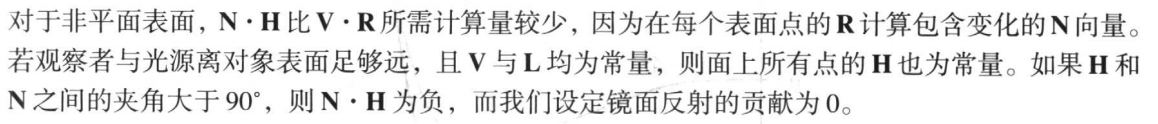


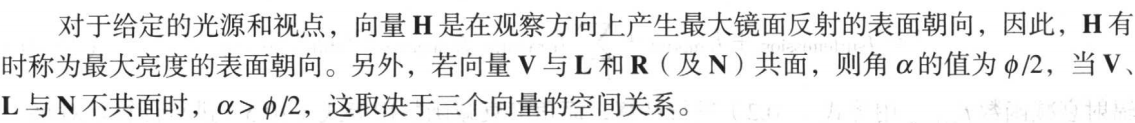






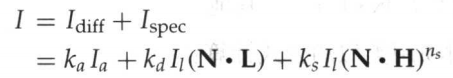






Lambert和Phong两个模型经常同时使用，因为在现实世界中，任何表面都会同时发生漫反射和镜面反射两种现象，因此我们就要使用两种模型分别计算两种发射后的光强（也就是顶点颜色值），使渲染的效率看起来真实一些，但是注意，这样做并不会带来真正真实的效果，毕竟这两种模型都是经验模型，考虑的都是在理想情况下。而Blinn-phong光照模型是基于Phong的修正模型。相比于Phong模型，Blinn-phong模型只是用N.H替代了V.R，但却获得了明显的提高，它能够提供比Phong更柔和。更平滑的高光，而且速度上更快，因此成为许多CG软件中默认的光照渲染方法，同时也被集成到大多数的图形芯片中，而且Opengl和DirecX 3D的渲染管线中，它也是默认的光照模型

漫反射和镜面发射合并：



Blinn-phong光照模型省去了计算发射光线方向向量的两个乘法运算，速度更快

**OpenGL光照**

光源的一般属性包括：镜面反射光线颜色、漫反射光线颜色、环境光线颜色与光源位置

光线的衰减：光源发出的光线强度会随着传播距离越来越大而变弱（无穷远光源除外）。光线强度会乘以一个衰减因子

衰减因子=1 / (K1 + K2 \* d + k3 \* d ^ 2) 其中d为光源距离

（无穷远光源的衰减因子为1）

方向光源发出的光线会随着偏移中心方向的角度增大而减弱

漫反射光：如果它从正面照射表面，看起来就更亮一些，反之，如果它写着掠过表面，看起来就暗一些，但是，当它撞击表面时，它会均与地向所有方向发散，因此，不管眼睛在哪个位置，漫反射光看上去总是一样亮

材质：材质是光照效果中的重要属性。材质描述了物体表面的光学物理属性，决定了光线在该表面光线反射的具体情况

物体表面的反射分为漫反射和镜面反射。物体反射各种类型光源的情况都可以分为漫反射和镜面反射两种

材质的属性包括：镜面反射颜色、漫反射颜色、环境光颜色、光洁度、自发光颜色

镜面反射颜色、漫反射颜色、环境光颜色分别与光源的镜面反射光颜色、光源的漫发射颜色、光源的环境光颜色共同决定物体表面的镜面发射颜色、漫反射颜色、环境光颜色。3种类型的结果分别计算，然后叠加共同确定发射表面像素值的颜色

Opengl确定某个顶点颜色可以分为两种情况：开启光照渲染和关闭光照渲染的情况。当关闭光照渲染的时候，顶点的颜色由opengl状态机绘制该物体时的颜色决定；当开启光照渲染时，绘制该物体的opengl状态机的颜色将对该物体上顶点的颜色没有任何影响，此时物体顶点的颜色由物体材质、光照叠加效果以及物体表面的纹理贴图（如果有纹理贴图的话）共同决定。

// 设置材质

GLfloat material\_ambient[] = {0.3f, 0.2f, 0.7f, 1.0f};

GLfloat material\_diffuse[] = {0.2, 0.8f, 1.0f, 1.0f};

glMaterialfv(GL\_FRONT, GL\_AMBIENT, material\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, material\_diffuse);

// 设置光照

GLfloat light\_ambient[] = {1.0f, 0.2f, 0.2f, 0.0f};

GLfloat light\_diffuse[] = {1.0f, 0.7f, 1.0f, 1.0f};

GLfloat light\_specular[] = {0.0f, 0.0f, 1.0f, 1.0f};

GLfloat light\_position[] = {-10.0f, 0.0f, 0.0f};

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glEnable(GL\_LIGHTING); // 开启光照渲染

glEnable(GL\_LIGHT0); // 开启光源

如果我们要打开全局光照，也就是在没有指定光源的时候，让场景中的物体也可以被照亮，可以使用下面的代码（已经测试）：

// 设置全局光照，是的物体在没有光照的情况下也可以被照亮

GLfloat global\_ambient[] = {0.5f, 0.2f, 0.2f, 1.0f};

glLightModelfv(GL\_AMBIENT\_AND\_DIFFUSE, global\_ambient);

**注意，在这种情况下也是需要开启光照渲染的**glEnable(GL\_LIGHTING); **在启动混合之前，这些颜色中的alpha成分是不会使用的，在此之前，可以完全忽略alpha段，在opengl中，被光照的物体并不会向其他物体发射光线**

由于物体有的表面相对于观察者来说是背向观察者的，即使光源照到这些表面上的顶点也不能被观察者看到，但是这些顶点也在渲染场景的时候，消耗了大量的计算时间，即使对最后的观察效果没有任何影响。所以可以设置背面的这些顶点不进行计算，添加如下代码：

// 关闭背面顶点的计算

glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_FALSE);

一个光源对场景中3D物体表面一个顶点的颜色贡献如下：

顶点颜色=全局环境光颜色+衰减因子\*（环境+漫反射+镜面反射）

考虑到材质自发光颜色属性和多个光源后，3D物体表面一个顶点的颜色为：

顶点颜色=材质自发光颜色+所有光源之和（全局环境光颜色+衰减因子\*（环境+漫反射+镜面反射））

环境光不依赖于光源的方向

漫反射光依赖于光源的方向和法线的方向

镜面反射光依赖于光源的方向、法线的方向和视角方向

**注意：在光照计算中，顶点法线必须是经过规范化的（normalize）**

光照中的纹理

当场景中的3D物体表面有纹理贴图时，纹理图像也将对点的最终颜色产生影响。在这种情况下，物体表面点的颜色由场景中的所有光源（包括环境光源），物体的材质，物体表面的纹理共同确定。渲染过程是先进行光照处理，再进行纹理处理，纹理处理会影响到最终3D物体表面像素点的颜色。

但是在这种情况下会发现镜面反射区可能被覆盖，为了解决这个问题，可以

glLightModefv(GL\_LIGHT\_MODEL\_COLOR\_CONTROL, GL\_SEPARATE\_SPECULAR\_COLOR),

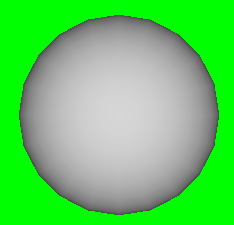
这样，每个顶点光照计算将产生两种颜色，主颜色和辅助颜色，前者包含所有非镜面发射光照的贡献，后者是所有镜面发射光照的总贡献。纹理映射的时候只将主颜色和纹理颜色混合起来，执行完纹理映射之后，再将主颜色和纹理颜色的混合结果与辅助颜色混合起来

光源GL\_LIGHT0与其他几个光源不同，它的

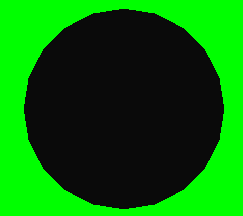
GL\_DIFFUSE、GL\_SPECULAR的默认值是（1.0，1.0,1.0，1.0）白光

而其他的光源默认值是（0.0,0.0,0.0,1.0）

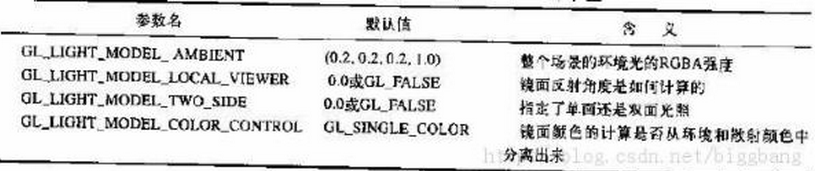
（已经测试）发现直接调用glEnable(GL\_LIGHT0);



glEnable(GL\_LIGHT1);



glLightModelfv(GLenum pname, const GLfloat \*params)

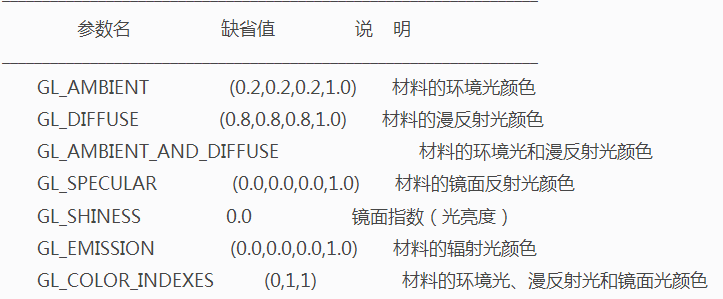


全局环境光是一种零成本的在opengl的光照计算结果上添加一个偏移量的方式，其默认的RGBA值是（0.2,0.2,0.2,1.0）

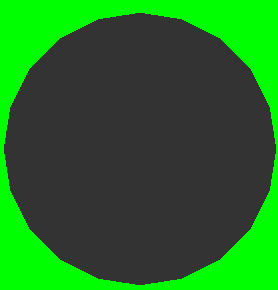
Gouraud Shading与Phong Shading

在图形渲染中有两种着色模式，高洛德着色与冯氏着色。高洛德着色也被称为per-vertex着色，它是在顶点着色阶段对顶点进行颜色计算，然后在光栅化阶段对这些顶点颜色进行线性插值形成片元的颜色；冯氏着色也被称为per-pixel像素着色，它是在片元着色阶段对每个片元（像素）进行颜色计算，无疑，插值的颜色效果没有针对每个片元进行颜色计算的效果好（除非你的图元切分到像素近似大小，不过这样GPU肯定吃不消，计算量巨大）。

Gouraud的消耗与模型的精细度相关，phong的消耗与render target分辨率相关



如果只开启了全局环境光，发现下面图的每个像素值均为51，



实验得出最终顶点的环境光=全局环境光\*顶点环境光+光源环境光\*顶点环境光

GLfloat globalAmebient[] = {1.0f, 1.0f, 1.0f, 1.0f};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, globalAmebient);

// 添加光源

GLfloat lightPosition[] = {-100.0f, 0.0f, 10.0f, 1.0f};

GLfloat lightAmbientColor[] = {1.0f, 0.0f, 0.0f, 1.0f};

GLfloat lightDiffuseColor[] = {0.0f, 0.0f, 0.0f, 1.0f};

GLfloat lightSpecularColor[] = {1.0f, 1.0f, 1.0f, 1.0f};

glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbientColor);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuseColor);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecularColor);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

//glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);

//glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

// 设置材质

GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f};

GLfloat diffuseColor[] = {0.0f, 0.0f, 0.0f, 1.0f};

GLfloat specularColor[] = {1.0f, 1.0f, 1.0f, 1.0f};

glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambientColor);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuseColor);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specularColor);

**glMateriali(GL\_FRONT, GL\_SHININESS, 10); // 镜面指数,这个特别重要**

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);



**注意：上面有一个参数特别重要GL\_SHININESS，表示镜面指数，取值范围为0-128，该值越小，表示材质越粗糙，点光源发射的光线照射到上面，也可以产生较大的亮点，该值越大，表示材质类似于镜面，光源照射到上面后，产生较小的亮点**

**将上面的10改为100**

**从上面的表中可以看出，该值的默认值为0**

**设置材质属性的方式有两种：**

1 在指定每个多边形或每组多边形之前使用glMaterial函数

GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f};

GLfloat diffuseColor[] = {0.0f, 0.0f, 0.0f, 1.0f};

GLfloat specularColor[] = {1.0f, 1.0f, 1.0f, 1.0f};

glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambientColor);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuseColor);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specularColor);

glMateriali(GL\_FRONT, GL\_SHININESS, 100);

glBegin(GL\_TRIANGLES);

glVertex3f(-15.0f, 0.0f, 30.0f);

glVertex3f(0.0f, 15.0f, 30.0f);

glVertex3f(0.0f, 0.0f, -56.0f);

glEnd();

在glMaterial调用之后所指定的所有图元都将受到最后一次所设置的值影响，直到再次调用glMaterial函数

2 颜色追踪方法（推荐）

可以告诉opengl仅仅通过调用glColor来设置材料属性，为了启动颜色追踪，需要调用

glEnable(GL\_COLOR\_MATERIAL);

接着，glColorMaterial函数根据glColor所设置的值来指定材料参数

GLfloat globalAmbient[] = {1.0f, 1.0f, 1.0f, 1.0f};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, globalAmebient);

glEnable(GL\_COLOR\_MATERIAL);

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

采用：

glColor4f(0.0f, 0.0f, 0.0f, 1.0f);

glutSolidSphere(10.0f, 50, 50);



glColor4f(0.5f, 0.5f, 0.5f, 1.0f);

glutSolidSphere(10.0f, 50, 50);



关于光照中光源的位置：

1）有待验证：顶点（基于局部坐标系） Model（基于世界坐标系） View（基于世界坐标系） Projection（基于眼坐标系） LightPosition（基于世界坐标系也可能基于眼坐标，取决于模型视图矩阵是先设置还是后设置？？？如果是之后设置是否应该是没有起到变化，而导致一直是用的默认的视点）

光源的坐标也会受到Model变换的影响（已经验证）

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 1.0f};



glTranslatef(-300.0f,0.0f, 0.0f);

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 1.0f};



在opengl中，至少支持8种独立的光源，它们可以出现在场景中的任何地方或者视景体之外，光源位置的控制也是通过模型视图矩阵实现的，包括平移、旋转、缩放（因为它会改变法向量，慎用）等，这样，光源可以看做一个定义在世界坐标系中的普通的几何对象。

在光照程序中：

gluLookAt(0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(-300.0f,0.0f, 0.0f);

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 1.0f};

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);



gluLookAt(-120.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(-300.0f,0.0f, 0.0f);

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 1.0f};

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);



对于普通的几何对象，我们在世界坐标系中定义它们，平移、旋转等也是在改变它们的世界坐标，这些坐标在提交给投影过程之前，opengl会根据当前世界坐标系和摄像机坐标系的关系把世界坐标系换换为摄像机坐标系，然后提交给投影过程，每个几何对象的坐标都会经过这种坐标变换

lightPosition包含了光的位置，这个数组最后一个值是1.0，表示指定的坐标是光源的位置，如果这个数组的最后一个值是0。0，则表示光是从无限远地方沿这个数组所指定的向量照射过来的

方向光源与聚光灯效果

在设置lightPosition的时候，第四个分量w为0，则表示光线从无穷远处向指定的位置发射而来，这就是方向光源。如果不为0，在x/w，y/w，z/w表示了光源的位置

使用方向光源的时候会发现光源的模型变换基本上没有什么影响

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 0.0f};



glTranslatef(-30000000.0f,0.0f, 0.0f);

GLfloat lightPosition[] = {150.0f, 0.0f, 0.0f, 0.0f};



方向性光源会均匀地照射在物体的表面，也就是说所有的光线都是平行的，反之，对于位置性光源，光线从光源发散开来，通常太阳可以近似地被认为是方向光源

聚光灯属性：

GL\_SPOT\_DIRECTION：表示一个向量，即光源发射的方向

GL\_SPOT\_EXPONENT：一个值，表示聚光的程度，为0时表示光照范围内向各个方向发射的光线强度相同，为正数时表示光照向中央集中，正对发射方向的位置受到更多光照，其他位置受到较少光照，数值越大，聚光效果就越明显

GL\_SPOT\_CUTOFF：一个值，表示一个角度，它是光源发射光线所覆盖角度的一半，其取值范围在0到90之间，也可以取180这个特殊值，取值为180时表示光源发射光线覆盖360度，即不适用聚光灯，向全周围发射。

GLfloat spotDirection[] = {0.0f, 0.0f, -1.0f};

glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, 60);

glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 10.0f);

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, spotDirection);



同时经过测试发现，聚光灯也受到模型视图矩阵变换的影响

发光体