#include <iostream>

#include <windows.h>

#include <GL/glut.h>

#include <math.h>

using namespace std;

//凡是需要修改的地方都由//-------------这种的标记

//实验思路，从简单的入手，就是shading一个立方体，立方体规格大小为10\*10\*10，这里只投影三个面，具体的像素颜色储存在一个三维颜色矩阵中

//base\_color 统一设置为100方便计算，之后绘制到屏幕上的时候在加以改变

float base\_color=100; //物体的基本色彩值为100，这个算出来到每一个体素的颜色值之后可以采用同一种方法映射到opengl可以表示的范围

bool Phong=false;

float \*\*\*object\_color=NULL;

int \*lightpos; //一个一维数组记录光源位置，光源位置是40,40,40

int \*eyepos; //一个一个数组记录视点位置，视点位置是30，20，30

//根据实现设定的光源位置和立方体位置，这里只有三个面会有光线照射，也只有这三个面 会被我们设置视点所看到，所以只要着重计算者三个面就行了

//这三个面分别是x=10的一面，y=10的一面，z=10的一面

void Diffuse\_color(float mat\_diffuse,float light\_diffuse,float \*\*\* object\_color,int \* lightpos )

//此函数用来计算漫反射的颜色，输入为材料漫反射系数、光源漫反射系数、物体的位置，物体每一个像素对应的基本颜色，光源的位置

//通过修改全局变量数组中的每一个变量来实现函数作用的实现

//调用这个方法的时候先将整个diffuse\_color存储到颜色数组中，后两个函数将分别的将颜色值加上去

{

float diffuse\_color = 0;

//这里假定base\_color是一个全局变量且物体是一个基本颜色为这个数值的匀色物体

diffuse\_color=base\_color\*mat\_diffuse\*light\_diffuse;

int \*vec\_pot\_light=new int[3];

int \*nor\_vec=new int[3];

float coefficient;

for(int j =0;j<10;j++)

{

for(int k=0;k<10;k++)

{

//计算x=10的那个平面

//计算该店法线点乘该点到光源的单位向量，如果该值小于0，那么就用0代替此次结果

//ijk分别是这个点的三个维度上对用的坐标

vec\_pot\_light[0]=lightpos[0]-9;

vec\_pot\_light[1]=lightpos[1]-j;

vec\_pot\_light[2]=lightpos[2]-k;

//vet\_pot\_light数组现在存储的是方向向量，但是不是单位向量

//如果投影的是长方体的话，那么法向量就很容易的可以到，这里假设已经得到了 该点的法向量并且法向量为nor\_vec,

nor\_vec[0]=1;

nor\_vec[1]=0;

nor\_vec[2]=0;

coefficient=vec\_pot\_light[0]\*nor\_vec[0]+vec\_pot\_light[1]\*nor\_vec[1]+vec\_pot\_light[2]\*nor\_vec[2];

coefficient=coefficient/sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

if (coefficient<=0)

coefficient=0;

object\_color[9][j][k]=diffuse\_color\*coefficient;

//计算y=10的那个平面

vec\_pot\_light[0]=lightpos[0]-j;

vec\_pot\_light[1]=lightpos[1]-9;

vec\_pot\_light[2]=lightpos[2]-k;

//vet\_pot\_light数组现在存储的是方向向量，但是不是单位向量

//如果投影的是长方体的话，那么法向量就很容易的可以到，这里假设已经得到了 该点的法向量并且法向量为nor\_vec,

nor\_vec[0]=0;

nor\_vec[1]=1;

nor\_vec[2]=0;

coefficient=vec\_pot\_light[0]\*nor\_vec[0]+vec\_pot\_light[1]\*nor\_vec[1]+vec\_pot\_light[2]\*nor\_vec[2];

coefficient=coefficient/sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

if (coefficient<=0)

coefficient=0;

object\_color[j][9][k]=diffuse\_color\*coefficient;

//计算z=10的那个平面

vec\_pot\_light[0]=lightpos[0]-j;

vec\_pot\_light[1]=lightpos[1]-k;

vec\_pot\_light[2]=lightpos[2]-9;

//vet\_pot\_light数组现在存储的是方向向量，但是不是单位向量

//如果投影的是长方体的话，那么法向量就很容易的可以到，这里假设已经得到了 该点的法向量并且法向量为nor\_vec,

nor\_vec[0]=0;

nor\_vec[1]=0;

nor\_vec[2]=1;

coefficient=vec\_pot\_light[0]\*nor\_vec[0]+vec\_pot\_light[1]\*nor\_vec[1]+vec\_pot\_light[2]\*nor\_vec[2];

coefficient=coefficient/sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

if (coefficient<=0)

coefficient=0;

object\_color[j][k][9]=diffuse\_color\*coefficient;

}

}

}

void Specular\_color(float mat\_specular,float light\_specular,int \*lightpos,int \*eyepos,float \*\*\* object\_color,float gloss)

//计算物体上每一个点的镜面反射颜色

{

float specular\_color=mat\_specular\*light\_specular;

int \*eye\_pot=new int[3];

float \*ref\_eye\_pot=new float[3];

int \*vec\_pot\_light=new int[3];

int \*nor\_vec=new int[3];

float length=0;

float coefficient=0;

for(int k =0;k<10;k++)

{

for(int j =0;j<10;j++)

{

//计算x=10的那个平面

//计算该店法线点乘该点到光源的单位向量，如果该值小于0，那么就用0代替此次结果

vec\_pot\_light[0]=lightpos[0]-9;

vec\_pot\_light[1]=lightpos[1]-k;

vec\_pot\_light[2]=lightpos[2]-j;

eye\_pot[0]=eyepos[0]-9;

eye\_pot[1]=eyepos[1]-k;

eye\_pot[2]=eyepos[2]-j;

//eye\_pot现在存储的是视点到该点的方向向量，但是不是单位向量，法向量是单位向量为nor\_vec

nor\_vec[0]=1;

nor\_vec[1]=0;

nor\_vec[2]=0;

length=sqrt(pow(eye\_pot[0],2)+pow(eye\_pot[1],2)+pow(eye\_pot[2],2));

//for(int i =0;i<3;i++) //将eyepot变为单位向量

// eye\_pot[i]/=length;

coefficient= eye\_pot[0]\*nor\_vec[0]+eye\_pot[1]\*nor\_vec[1]+eye\_pot[2]\*nor\_vec[2];

coefficient/=length;

length=sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

coefficient/=length;

for (int i =0;i<3;i++) //计算得到变换之后的eyepot的反射向量，计算参考的公式http://www.cnblogs.com/graphics/archive/2013/02/21/2920627.html

ref\_eye\_pot[i]=eye\_pot[i]-2\*coefficient\*nor\_vec[i];

coefficient=0;

for (int i =0;i<3;i++) //计算出了眼睛到该点的单位向量的反射向量和该点到光源的向量的乘积

coefficient+=ref\_eye\_pot[i]\*vec\_pot\_light[i];

coefficient/=length;

if (coefficient<0)

coefficient=0;

object\_color[9][k][j]+=pow(coefficient,gloss)\*specular\_color;

//计算y=10的那个平面

//计算该店法线点乘该点到光源的单位向量，如果该值小于0，那么就用0代替此次结果

vec\_pot\_light[0]=lightpos[0]-k;

vec\_pot\_light[1]=lightpos[1]-9;

vec\_pot\_light[2]=lightpos[2]-j;

eye\_pot[0]=eyepos[0]-k;

eye\_pot[1]=eyepos[1]-9;

eye\_pot[2]=eyepos[2]-j;

//eye\_pot现在存储的是视点到该点的方向向量，但是不是单位向量，法向量是单位向量为nor\_vec

nor\_vec[0]=0;

nor\_vec[1]=1;

nor\_vec[2]=0;

length=sqrt(pow(eye\_pot[0],2)+pow(eye\_pot[1],2)+pow(eye\_pot[2],2));

coefficient= eye\_pot[0]\*nor\_vec[0]+eye\_pot[1]\*nor\_vec[1]+eye\_pot[2]\*nor\_vec[2];

coefficient/=length;

length=sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

for (int i =0;i<3;i++) //计算得到变换之后的eyepot的反射向量，计算参考的公式http://www.cnblogs.com/graphics/archive/2013/02/21/2920627.html

ref\_eye\_pot[i]=eye\_pot[i]-2\*coefficient\*nor\_vec[i];

coefficient=0;

for (int i =0;i<3;i++) //计算出了眼睛到该点的单位向量的反射向量和该点到光源的向量的乘积

coefficient+=ref\_eye\_pot[i]\*vec\_pot\_light[i];

coefficient/=length;

if (coefficient<0)

coefficient=0;

object\_color[k][9][j]+=pow(coefficient,gloss)\*specular\_color;

//计算z=10的那个平面

//计算该店法线点乘该点到光源的单位向量，如果该值小于0，那么就用0代替此次结果

vec\_pot\_light[0]=lightpos[0]-k;

vec\_pot\_light[1]=lightpos[1]-j;

vec\_pot\_light[2]=lightpos[2]-9;

eye\_pot[0]=eyepos[0]-k;

eye\_pot[1]=eyepos[1]-j;

eye\_pot[2]=eyepos[2]-9;

//eye\_pot现在存储的是视点到该点的方向向量，但是不是单位向量，法向量是单位向量为nor\_vec

nor\_vec[0]=0;

nor\_vec[1]=0;

nor\_vec[2]=1;

length=sqrt(pow(eye\_pot[0],2)+pow(eye\_pot[1],2)+pow(eye\_pot[2],2));

coefficient= eye\_pot[0]\*nor\_vec[0]+eye\_pot[1]\*nor\_vec[1]+eye\_pot[2]\*nor\_vec[2];

coefficient/=length;

length=sqrt(pow(vec\_pot\_light[0],2)+pow(vec\_pot\_light[1],2)+pow(vec\_pot\_light[2],2));

for (int i =0;i<3;i++) //计算得到变换之后的eyepot的反射向量，计算参考的公式http://www.cnblogs.com/graphics/archive/2013/02/21/2920627.html

ref\_eye\_pot[i]=eye\_pot[i]-2\*coefficient\*nor\_vec[i];

coefficient=0;

for (int i =0;i<3;i++) //计算出了眼睛到该点的单位向量的反射向量和该点到光源的向量的乘积

coefficient+=ref\_eye\_pot[i]\*vec\_pot\_light[i];

coefficient/=length;

if (coefficient<0)

coefficient=0;

object\_color[k][j][9]+=pow(coefficient,gloss)\*specular\_color;

}

}

}

void Ambient\_color(float mat\_ambient,float light\_ambient,float \*\*\* object\_color )

//这里是对环境光颜色的变换

{

float ambient\_color=base\_color\*mat\_ambient\*light\_ambient;

for(int i =0;i<10;i++)

{

for (int j=0;j<10;j++)

{

object\_color[9][i][j]+=ambient\_color;

object\_color[i][9][j]+=ambient\_color;

object\_color[i][j][9]+=ambient\_color;

}

}

}

void Phongshading(float mat\_diffuse,float light\_diffuse,float mat\_specular,float light\_specular,float mat\_ambient,float light\_ambient,

float \*\*\*object\_color,int \*lightpos,int \*eyepos)

//利用6个输入分别为材料漫反射系、光源漫反射系数、材料镜面反射系数、光源镜面反射系数、材料环境光系数、灯光环境光系数来对物体上的每一个像素点进行相对应的匹配计算

//计算的公式参考http://www.cnblogs.com/qingsunny/archive/2013/03/02/2939765.html

{

Diffuse\_color(mat\_diffuse,light\_diffuse,object\_color,lightpos);

float gloss=1.2;

Specular\_color(mat\_specular,light\_specular,lightpos,eyepos,object\_color,gloss);

Ambient\_color(mat\_ambient,light\_ambient,object\_color);

}

void display()

{

//glClearColor(1.0f, 1.0f, 1.0f,0.1f);

float width=0.05,height=0.05,startx=0.3,starty=0.3;

glClear(GL\_COLOR\_BUFFER\_BIT);

for(int i =9;i>-1;i--)

{

for(int j=9;j>-1;j--)

{

if(Phong)

{

glColor3f(1.0f,1.0f,(object\_color[9][i][j]-70)/160);

glRectf(startx-i\*width,starty-(9-j)\*width,startx-i\*width+width,starty-(9-j)\*width+width);

glColor3f(1.0f,1.0f,(object\_color[i][9][j]-60)/200);

glRectf(startx+i\*width\*0.5-9\*width\*0.5-9\*width-width\*0.5,starty-(9-j)\*width+(9-i)\*width\*0.5,startx+i\*width\*0.5-9\*width\*0.5-9\*width,starty-(9-j)\*width+width+(9-i)\*width\*0.5);

glColor3f(1.0f,1.0f,(object\_color[i][j][9]-60)/200);

glRectf(startx-i\*width+(j-9)\*width\*0.5,starty+(9-j)\*width\*0.5,startx-i\*width+width+(j-9)\*width\*0.5,starty+(9-j)\*width\*0.5+width);

}

else

{

//利用线性差值每次递增那么多

glColor3f(1.0f,1.0f,-(object\_color[9][0][0]-object\_color[9][0][9])/160/10\*j+-(object\_color[9][0][0]-object\_color[9][9][0])/160/10\*i+

(object\_color[9][0][0]-70)/160);

glRectf(startx-i\*width,starty-(9-j)\*width,startx-i\*width+width,starty-(9-j)\*width+width);

glColor3f(1.0f,1.0f,-(object\_color[0][9][0]-object\_color[0][9][9])/160/10\*j+-(object\_color[0][9][0]-object\_color[9][9][0])/160/10\*i+

(object\_color[0][9][0]-70)/160);

glRectf(startx+i\*width\*0.5-9\*width\*0.5-9\*width-width\*0.5,starty-(9-j)\*width+(9-i)\*width\*0.5,startx+i\*width\*0.5-9\*width\*0.5-9\*width,starty-(9-j)\*width+width+(9-i)\*width\*0.5);

glColor3f(1.0f,1.0f,-(object\_color[0][0][9]-object\_color[0][9][9])/160/10\*j+-(object\_color[0][0][9]-object\_color[9][0][9])/160/10\*i+

(object\_color[0][0][9]-70)/160);

glRectf(startx-i\*width+(j-9)\*width\*0.5,starty+(9-j)\*width\*0.5,startx-i\*width+width+(j-9)\*width\*0.5,starty+(9-j)\*width\*0.5+width);

}

}

}

glFlush();

}

int main(int argc,char \*\*argv)

{

int winwidth=640,winheight=480;

glutInit(&argc,argv);

glutInitWindowSize (winwidth,winheight);

glutCreateWindow("Hello,world!");

float mat\_ambient=0.5,light\_ambient=0.5,mat\_specular=0.8,light\_specular=0.8,mat\_diffuse=0.5,light\_diffuse=0.5;

//初始化这个立方体的各个像素，虽然我们把内部的点也给初始化了，但是在计算需要shading的时候我们只需要计算表面上的体素就可以了

object\_color=new float\*\*[10];

for(int i =0;i<10;i++)

{

object\_color[i]=new float\*[10];

for (int j =0;j<10;j++)

object\_color[i][j]=new float[10];

}

lightpos=new int[3];

eyepos=new int[3];

lightpos[0]=50;

lightpos[1]=50;

lightpos[2]=50; //光源的位置

eyepos[0]=30;

eyepos[1]=20;

eyepos[2]=30; //视点的位置

//接下来只需要计算每一个表面的点的颜色就行了，但是计算所有的演示可以的

Phongshading(mat\_diffuse,light\_diffuse,mat\_specular,light\_specular,mat\_ambient,light\_ambient,object\_color,lightpos,eyepos);

glutDisplayFunc(display);

glutMainLoop();

}