**电 子 科 技 大 学**

**实 验 报 告**

**一、实验四：OpenGL纹理映射编程实验**

**二、实验室名称：主楼A2-412 实验时间：14：30~17：55**

**三、实验目的：**

1、理解OpenGL纹理映射的相关原理；

2、掌握与纹理映射相关的OpenGL API函数及其用法；

3、熟悉纹理映射设置的基本步骤。

**四、实验原理：OpenGl**

**五、实验内容：**

**（一）编程任务1——绘制一个具有纹理和光照的房间**

为在第三次实验中生成的房间（如上图）添加数字图像生成的纹理。

要求：

1、 使用纹理对象；

2、 墙壁、地板和天花板的数字图像生成的纹理要不相同；

3、 屋内至少有一个点光源；

4、 可用右键菜单交互式的开启或关闭场景光照和纹理；

5、 可用右键菜单设置纹理环境（GL\_REPLACE和GL\_MODULATE）。分析这两种方式中哪种适宜于有光照的环境下。

**六、实验器材（设备、元器件）：电脑**

**七、实验步骤：**

1. 生成纹理对象并应用，将墙壁等划分为小块让光照明显
2. 设置菜单

**八、实验数据及结果分析：**

编程任务1：

程序功能：

程序实现了绘制一个具有光照和纹理的房间。

源代码：

#include <glut.h>

#include <stdio.h>

#include <math.h>

#define PI 3.14159265

void init();

void display();

void mykey(unsigned char, int, int);

void mymouse(int, int, int, int);

void CreateWall();

void CreateCeiling();

void CreateGround();

void CreateFurnishings(int);//0=球体

void CreateLightAndProxy(GLenum);

void DrawWorld(void);

void menu(int);

void move(int, int);

void setTex(int);

int bx;

float p1[] = { -2,3,-8,1 };

float p2[] = { 2,2,-8,1 };

GLint buff[100];

int by;

GLint id[3];//0=地板1=墙壁2=天花板

typedef struct obj {

GLfloat a[4];

GLfloat d[4];

GLfloat s[4];

GLfloat shin;

}obj;

obj ball = {

{1,0.6,0.07,1},

{1,0.6,0.07,1},

{1,0.6,0.07,1},

50

};

obj cone = {

{0.25,0.4,0.88,1},

{0.25,0.4,0.88,1},

{0.25,0.4,0.88,1},

50

};

obj wall = {

{0.85,0.625,0.86,1},

{0.85,0.625,0.86,1},

{0.85,0.625,0.86,1},

100

};

obj ceiling = {

{0.1875,0.5,0.08,1},

{0.1875,0.5,0.08,1},

{0.1875,0.5,0.08,1},

100

};

obj ground = {

{1,1,1,1},

{1,1,1,1},

{1,1,1,1},

100

};

obj light = {

{1,1,0,1},

{1,1,0,1},

{1,1,0,1},

50

};

void use(obj t) {

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, t.a);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, t.d);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, t.s);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, t.shin);

}

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

glutInit(&argc, argv);

glutInitWindowSize(800, 800);

glutInitWindowPosition(0, 0);

glutCreateWindow("第四次实验");

init();

glutDisplayFunc(display);

glutMouseFunc(mymouse);

glutMotionFunc(move);

glutMainLoop();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

DrawWorld();

glFlush();

}

void init() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_NORMALIZE);

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

float a[] = { 0.5,0.5,0.5,1 };

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(80, 1, 1, 30);

glutCreateMenu(menu);

glutAddMenuEntry("开启点光源", 0);

glutAddMenuEntry("关闭点光源", 1);

glutAddMenuEntry("开启聚光灯", 2);

glutAddMenuEntry("关闭聚光灯", 3);

glutAddMenuEntry("开启纹理", 4);

glutAddMenuEntry("关闭纹理", 5);

glutAddMenuEntry("GL\_REPLACE", 6);

glutAddMenuEntry("GL\_MODULATE", 7);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glGenTextures(3, id);

}

void setTex(int c) {

GLubyte image[4][4][3];

int t;

switch (c)

{

case 0:

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if ((i < 2 && j < 2) || (i >= 2 && j >= 2))

t = 255;

else

t = 0;

image[i][j][0] = (GLubyte)t;

image[i][j][1] = (GLubyte)t;

image[i][j][2] = (GLubyte)t;

}

}break;

case 1:

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if (i < 2)

t = 255;

else

t = 0;

image[i][j][0] = (GLubyte)t;

image[i][j][1] = (GLubyte)t;

image[i][j][2] = (GLubyte)t;

}

}break;

case 2:

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if ((i = j) || (i+j==3))

t = 125;

else

t = 0;

image[i][j][0] = (GLubyte)t;

image[i][j][1] = (GLubyte)t;

image[i][j][2] = (GLubyte)t;

}

}break;

}

glBindTexture(GL\_TEXTURE\_2D, id[c]);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, 4, 4, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, image);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);//需要增加对于超出部分像素的处理，否则无法显示

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

}

void mymouse(int button, int state, int x, int y)

{

GLint viewport[4];

int hit;

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

bx = x;

by = y;

glGetIntegerv(GL\_VIEWPORT, viewport);

glSelectBuffer(100, buff);

glRenderMode(GL\_SELECT);

glInitNames();

glPushName(0);

glMatrixMode(GL\_PROJECTION);

glPushMatrix();

glLoadIdentity();

gluPickMatrix(x, viewport[3] - y, 3, 3, viewport);

gluPerspective(80, 1, 1, 30);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 5, 10, 0, 5, 0, 0, 1, 0);

CreateLightAndProxy(GL\_SELECT);

glMatrixMode(GL\_PROJECTION);

glPopMatrix();

glFlush();

hit = glRenderMode(GL\_RENDER);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

}

void CreateWall()

{

setTex(1);

use(wall);

glNormal3f(0, 0, 1);

int huafen = 40;

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0 / huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(-10 + i \* step, j\*step, -10);

glTexCoord2d(1, 0);

glVertex3d(-10 + i \* step + step, j \* step, -10);

glTexCoord2d(1, 1);

glVertex3d(-10 + i \* step + step, j \* step+step, -10);

glTexCoord2d(0, 1);

glVertex3d(-10 + i \* step, j \* step + step, -10);

glEnd();

}

}

glNormal3f(-1, 0, 0);

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0 / huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(10, j \* step, -10 + i \* step);

glTexCoord2d(1, 0);

glVertex3d(10, j \* step, -10 + i \* step + step);

glTexCoord2d(1, 1);

glVertex3d(10, j \* step + step, -10 + i \* step + step);

glTexCoord2d(0, 1);

glVertex3d(10, j \* step + step, -10 + i \* step);

glEnd();

}

}

glNormal3f(0, 0, -1);

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0 / huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(10 - i \* step, j \* step, 10);

glTexCoord2d(1, 0);

glVertex3d(10 - i \* step-step, j \* step, 10);

glTexCoord2d(1, 1);

glVertex3d(10 - i \* step - step, j \* step + step, 10);

glTexCoord2d(0, 1);

glVertex3d(10 - i \* step, j \* step + step, 10);

glEnd();

}

}

glNormal3f(1, 0, 0);

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0 / huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(-10, j \* step, 10 - i \* step);

glTexCoord2d(1, 0);

glVertex3d(-10, j \* step, 10 - i \* step - step);

glTexCoord2d(1, 1);

glVertex3d(-10, j \* step + step, 10 - i \* step - step);

glTexCoord2d(0, 1);

glVertex3d(-10, j \* step + step, 10 - i \* step);

glEnd();

}

}

}

void CreateCeiling()

{

setTex(2);

use(ceiling);

glNormal3f(0, 1, 0);

int huafen = 20;

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0 / huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(-10 + i \* step, 10, 10 - j \* step);

glTexCoord2d(1, 0);

glVertex3d(-10 + i \* step + step, 10, 10 - j \* step);

glTexCoord2d(1, 1);

glVertex3d(-10 + i \* step + step, 10, 10 - j \* step - step);

glTexCoord2d(0, 1);

glVertex3d(-10 + i \* step, 10, 10 - j \* step - step);

glEnd();

}

}

}

void CreateGround()

{

setTex(0);

glNormal3f(0, 1, 0);

use(ground);

int huafen = 80;

for (int i = 0; i < huafen; i++) {

for (int j = 0; j < huafen; j++) {

double step = 20.0/huafen;

glBegin(GL\_POLYGON);

glTexCoord2d(0, 0);

glVertex3d(-10+i\*step, 0, 10-j\*step);

glTexCoord2d(0.1, 0);

glVertex3d(-10 + i \* step+step, 0, 10 - j \* step);

glTexCoord2d(0.1, 0.1);

glVertex3d(-10 + i \* step + step, 0, 10 - j \* step-step);

glTexCoord2d(0, 0.1);

glVertex3d(-10 + i \* step, 0, 10 - j \* step - step);

glEnd();

}

}

}

void CreateFurnishings(int choice)

{

glPushMatrix();

if (choice == 0) {

use(ball);

glutSolidSphere(1, 60, 60);

}

else {

use(cone);

glRotatef(-90, 1, 0, 0);

glutSolidCone(1, 3, 50, 50);

}

glPopMatrix();

}

void CreateLightAndProxy(GLenum c)

{

float a1[] = { 0.1,0.1,0.1,1 };

float d1[] = { 1,1,0.8,1 };

float s1[] = { 1,1,1,1 };

glLightfv(GL\_LIGHT0, GL\_POSITION, p1);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, a1);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, d1);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, s1);

float a2[] = { 0.1,0.1,0.1,1 };

float d2[] = { 1,1,1,1 };

float s2[] = { 1,1,1,1 };

float dir[] = { -1,-1,1 };

glLightfv(GL\_LIGHT1, GL\_POSITION, p2);

glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, dir);

glLightf(GL\_LIGHT1, GL\_SPOT\_CUTOFF, 30);

glLightfv(GL\_LIGHT1, GL\_AMBIENT, a2);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, d2);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, s2);

if (c == GL\_SELECT)

glLoadName(1);

use(light);

glPushMatrix();

glTranslatef(p1[0], p1[1], p1[2]);

glutSolidCube(1);

glPopMatrix();

if (c == GL\_SELECT)

glLoadName(2);

use(light);

glPushMatrix();

glTranslatef(p2[0], p2[1], p2[2]);

glutSolidSphere(0.5, 50, 50);

glPopMatrix();

}

void DrawWorld(void)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(80, 1, 1, 30);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 5, 10, 0, 5, 0, 0, 1, 0);

glPushMatrix();

CreateWall();

CreateCeiling();

CreateGround();

glTranslatef(0, 1, -5);

CreateFurnishings(0);

glTranslatef(-5, -1, 5);

CreateFurnishings(1);

glPopMatrix();

CreateLightAndProxy(GL\_RENDER);

}

void menu(int t)

{

switch (t)

{

case 0:glEnable(GL\_LIGHT0); break;

case 1:glDisable(GL\_LIGHT0); break;

case 2:glEnable(GL\_LIGHT1); break;

case 3:glDisable(GL\_LIGHT1); break;

case 5:glDisable(GL\_TEXTURE\_2D); break;

case 6:glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE); break;

case 7:glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE); break;

}

glutPostRedisplay();

}

void move(int x, int y)

{

if (buff[3] == 1) {

p1[0] += (x - bx) / 35.0;

p1[1] += (by - y) / 35.0;

}

else {

p2[0] += (x - bx) / 35.0;

p2[1] += (by - y) / 35.0;

}

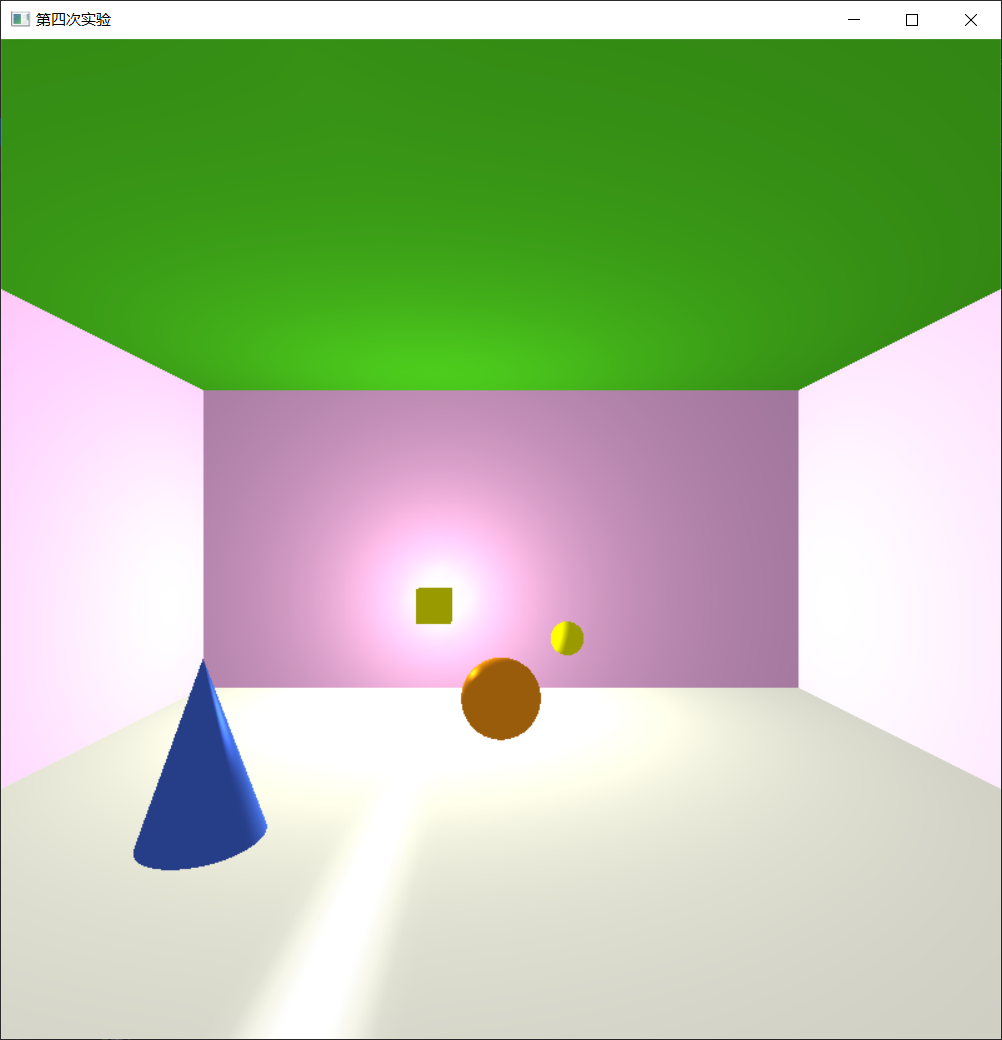
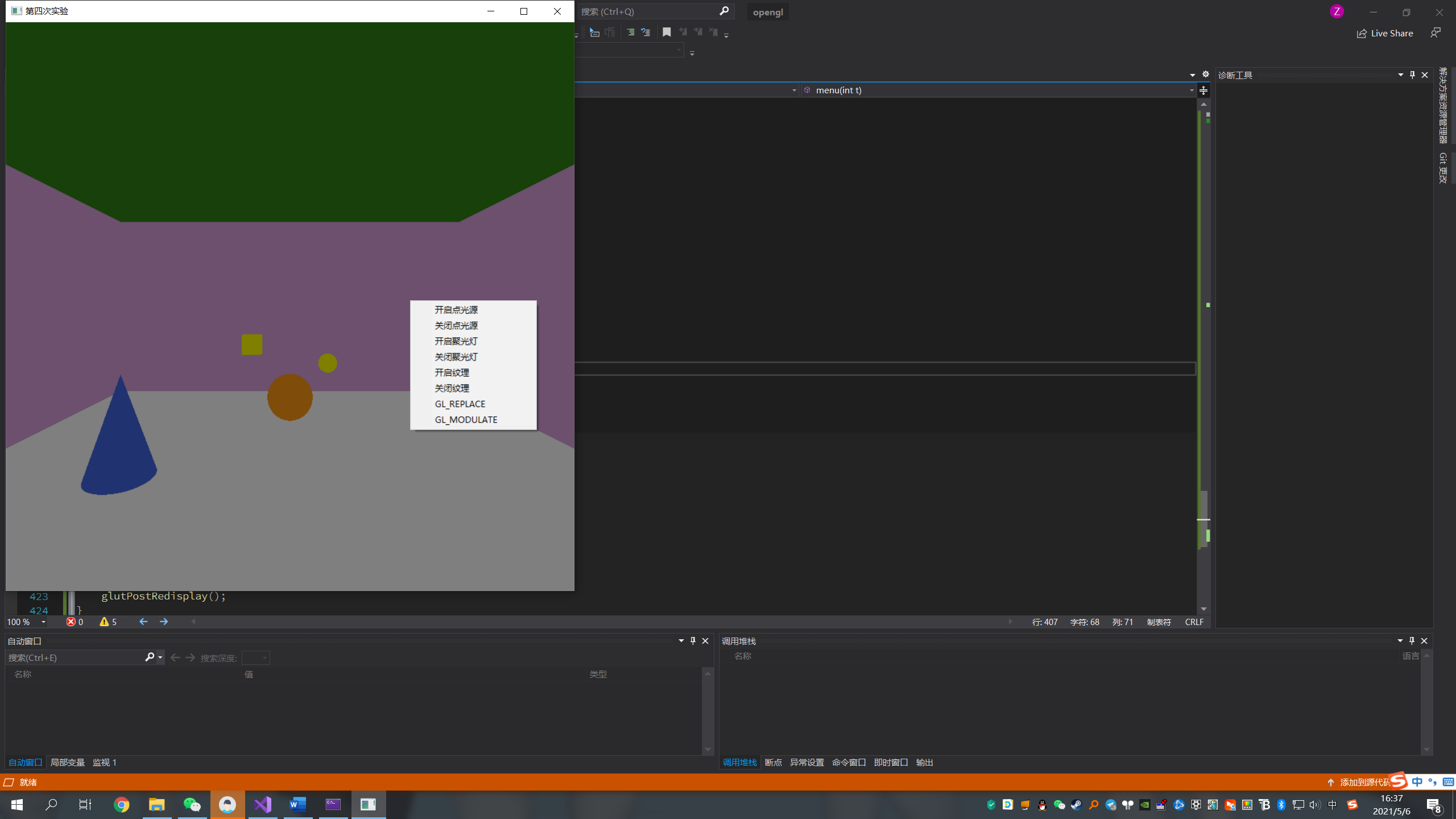
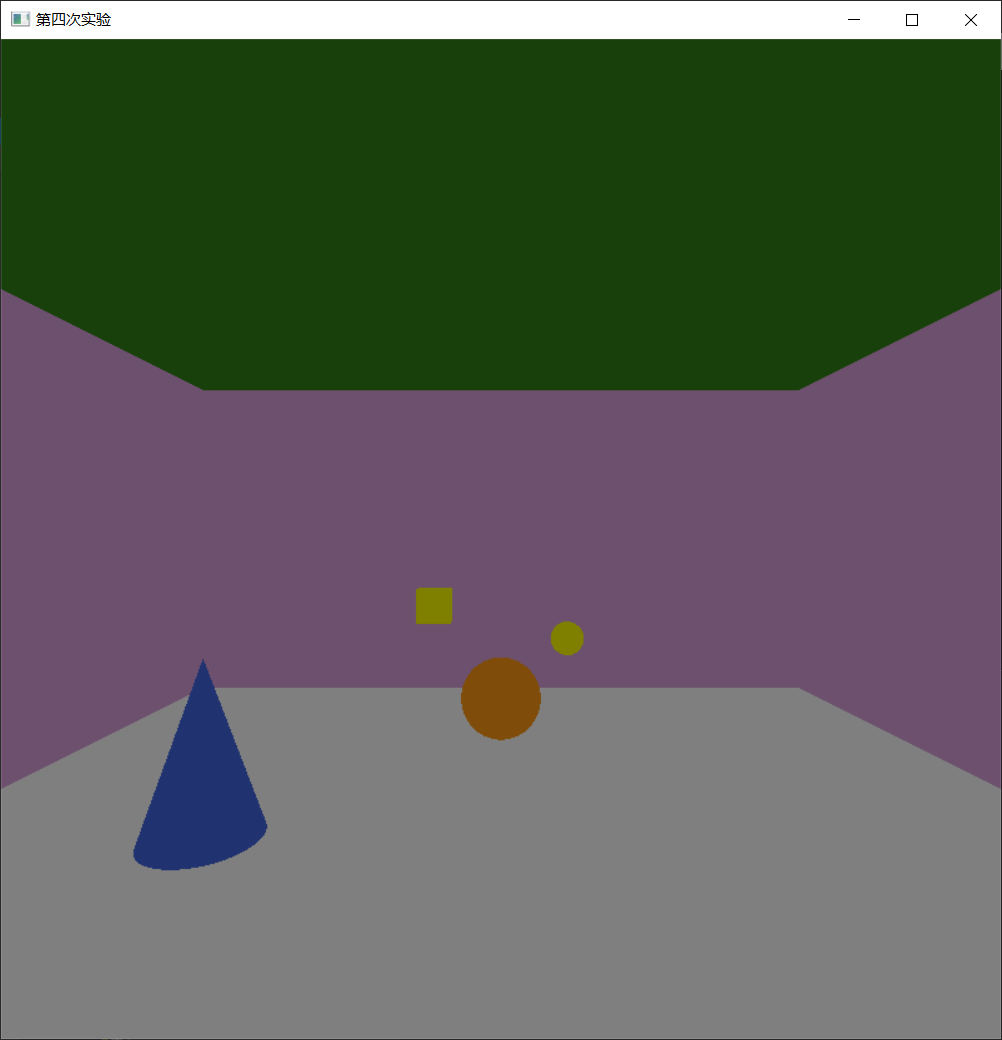
bx = x; by = y;

glutPostRedisplay();

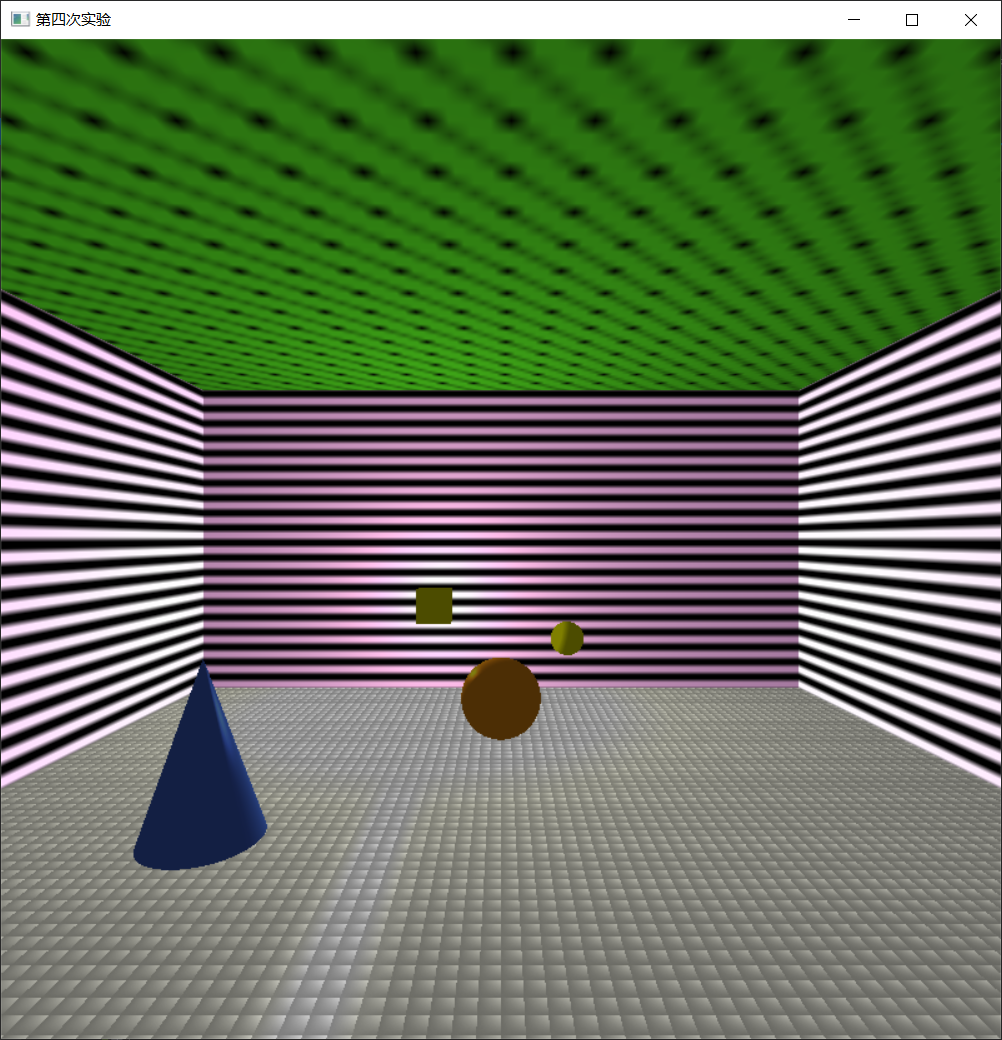
}

运行结果：

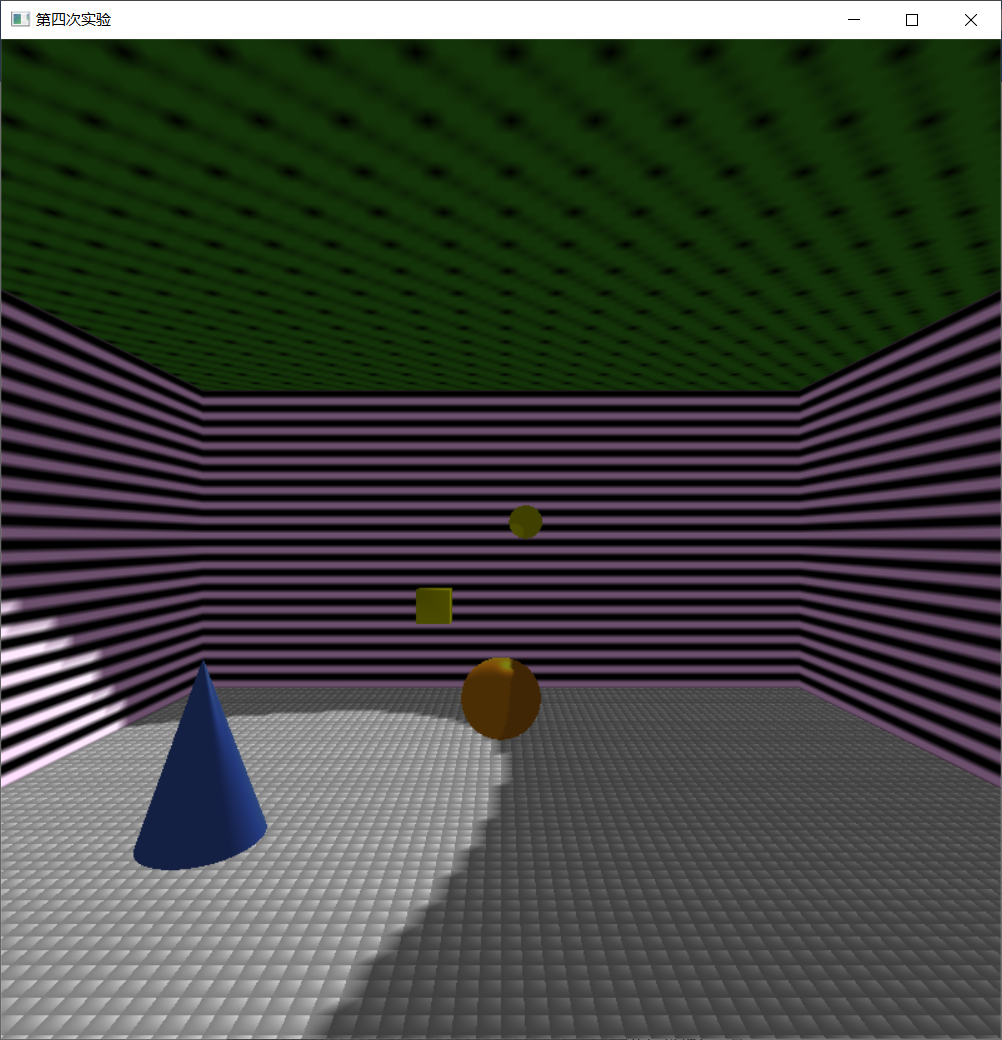
刚开始没有光照和纹理。通过菜单界面修改光照。开启点光源后。



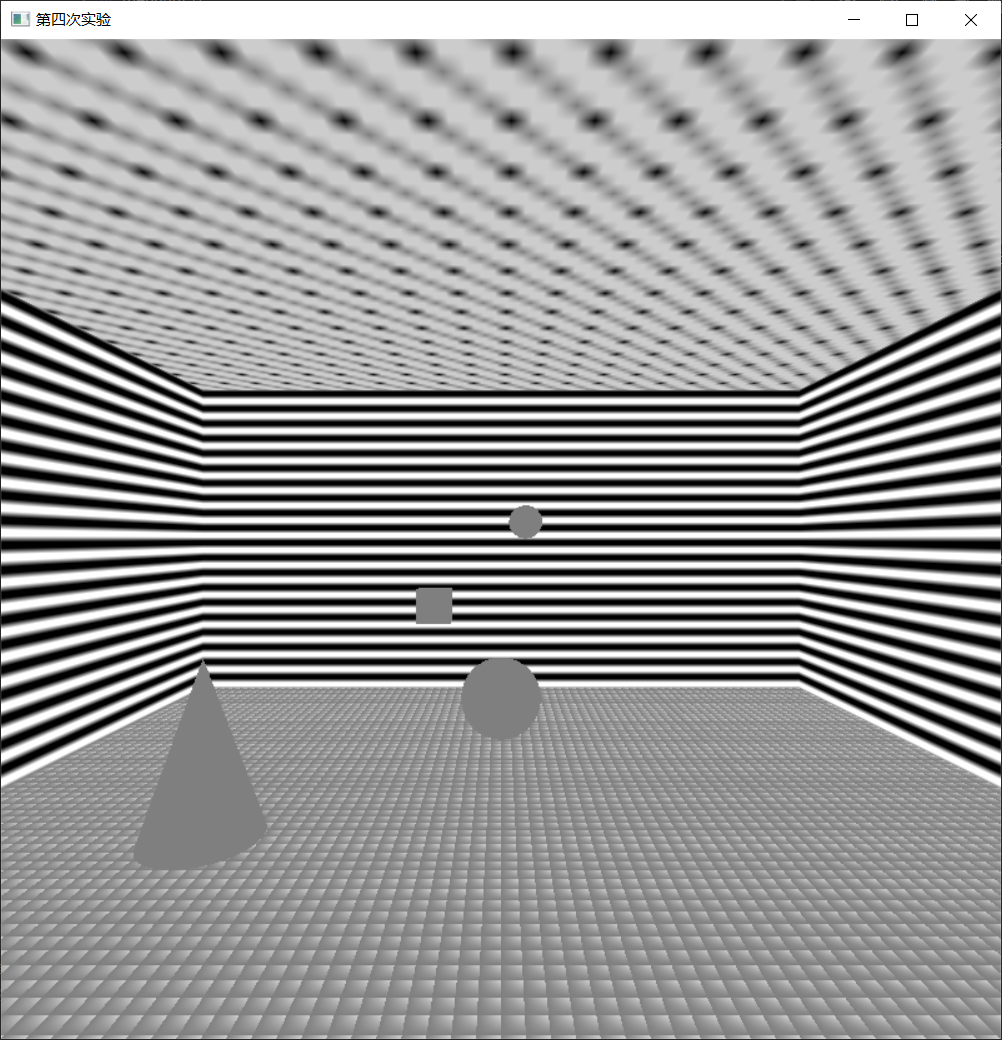
开启纹理后。



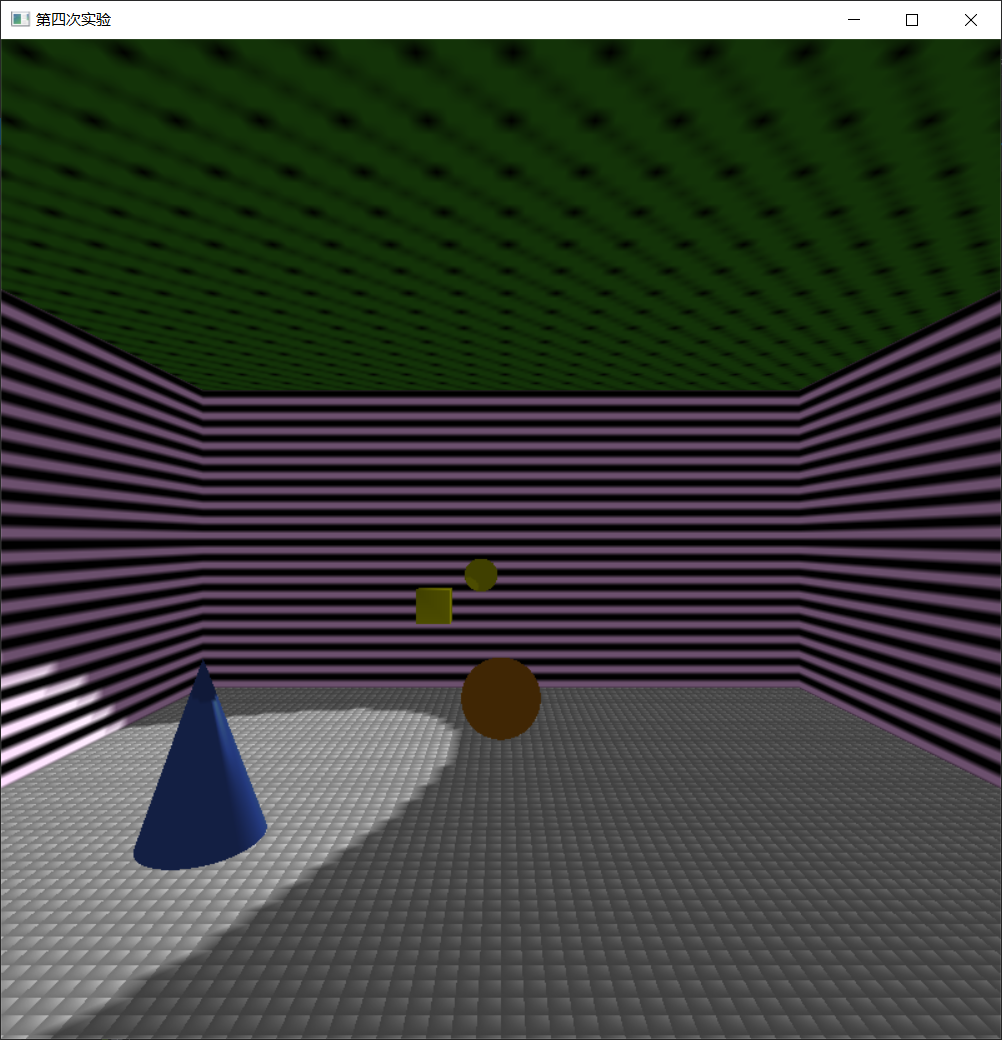
使用聚光灯效果更加明显。



开启GL\_REPLACE



开启GL\_MODULATE，恢复正常



**总结、改进建议及心得体会：**在有光照的情况下，GL\_MODULATE明显好于GL\_REPLACE。为了让光照的效果更加明显和正确，需要将物体划分成很多的小块，这样光照才明显。