華中科技大學

课 程 实 验 报 告

课程名称： 计 算 机 图 形 学

专业班级： 计科 1305

学　　号： U201214946

姓　　名： 赵子昂

指导教师： 徐海银　吕新桥

报告日期： 2015 年 11 月 7 日

计算机科学与技术学院

实验一：分形图形绘制

一、实验目的

(1)理解OpenGL 中glut 程序框架；

(2)掌握二维基本图形绘制算法；

(3)利用二维基本图形绘制算法，扩展对其他复杂图形的绘制理解。

二、实验内容

1、实验算法

(1)中点 Bresenham 画线算法。本算法的基本原理为，每次在最大位移方向上走一步，而另一个方向是走步还是不走步取决于误差项的判别。故算法开始即判断输入的两个坐标的大小来确定起始点。之后求出 dx 和 dy，判式选择由于之前算法里已有比较好的范例 d=dx-2*dy，d 的增量为 2*dx-2*dy 和-2*dy，之后根据判式 d 的符号即可判断点的位置和增量的选取，如此循环直至画线结束。

(2)分形三角形实现算法。本算法是基于 IFS（Iterated Function System）来实现的 Sierpinski 三角形。IFS 将待生成的图像看做是由许多与整体相似的（自相似）或经过一定变换与整体相似的（自仿射）小块拼贴而成。而此处的基本算法思想是，取一个三角形，求出其三个中点，然后以两个中点和其中一个顶点作出一个新的三角形，然后生成的所有的三角形递归此过程。递归次数越多，所得到的图形越复杂。

2、源程序

```
#include "windows.h"
#include <iostream>
#include <math.h>
#include <GL/glut.h>
#include "stdlib.h"
#include "windows.h"
#define ROUND(a) ((int )(a+0.5))   //求某个数的四舍五入值
using namespace std;
int xf,yf,xl,yl;


void init(void)
{
```

```
 glClearColor(1.0,1.0,1.0,0.0);//指定窗口的背景色为白色
 glMatrixMode(GL_PROJECTION);//对投影矩阵进行操作
 gluOrtho2D(0.0,600.0,0.0,600.0);//使用正投影
}

//绘制直线的函数
void lineDDA (GLint xa, GLint ya, GLint xb, GLint yb)
{
 GLint dx=xb-xa, dy=yb-ya;      //计算 x,y 方向的跨距
 int steps,k;      //定义绘制直线像素点的步数
 float xIcre, yIcre,x=xa, y=ya; //定义步长的增量，取 X，Y 方向跨距较大的值为步数
 if (abs(dx)>abs(dy)) steps=abs(dx);
  else steps=abs(dy);                  //根据步数来求步长增量
  xIcre=dx/(float)steps;
  yIcre=dy/(float)steps;               //从起点开始绘制像素点

  for(k=0;k<=steps; k++)
  {
      glBegin(GL_POINTS);
      glVertex2f(x,y);
      glEnd();
         x+=xIcre;
         y+=yIcre;

  }
}

void MidBresenhamLine (GLint xa, GLint ya, GLint xb, GLint yb)
{
 GLint dx,dy,d,UpIncre,DownIncre,x,y;
 if(xa>xb){                           //比较 xa，ya 来判断较小点为起点
      x=xb;xb=xa;xa=x;
      y=yb;yb=ya;ya=x;
 }
 x=xa;y=ya;
 dx=xb-xa, dy=yb-ya;                  //计算 XY 方向的跨距
 d=dx-2*dy;                           //判式 d
 UpIncre=2*dx-2*dy;DownIncre=-2*dy;    //d 的增量
 while(x<=xb){                        //从起点开始绘制像素点
      glBegin(GL_POINTS);
      glVertex2f(x,y);
      glEnd();
      x++;
      if(d<0){
```

```
            y++;
            d+=UpIncre;
        }
        else d+=DownIncre;
    }
}
void Fractal(GLfloat xa,GLfloat ya,GLfloat xb,GLfloat yb,GLfloat xc,GLfloat yc,GLint n)
{
    GLfloat xab,yab,xac,yac,xbc,ybc;      //定义三角形三个中点的坐标
    if(n>0){
        xab=(xa+xb)/2;                          //计算坐标
        yab=(ya+yb)/2;
        xac=(xa+xc)/2;
        yac=(ya+yc)/2;
        xbc=(xb+xc)/2;
        ybc=(yb+yc)/2;
        Fractal(xa,ya,xab,yab,xac,yac,n-1);       //对两个重点和一个顶点构成的三角形递归
        Fractal(xab,yab,xb,yb,xbc,ybc,n-1);
        Fractal(xac,yac,xbc,ybc,xc,yc,n-1);
    }
    else
    {
        glBegin(GL_TRIANGLES);                  //从起点开始绘制三角形
        glVertex2f(xa,ya);
        glVertex2f(xb,yb);
        glVertex2f(xc,yc);
        glEnd();
    }
void lineSegment(void)
{
    glClear(GL_COLOR_BUFFER_BIT);//设定颜色缓存中的值
    glColor3f(1.0,0.0,0.0); //设置直线颜色为红色
    MidBresenhamLine (xf,yf,xl,yl);       //调用中点 Bresenham 函数
    glFlush();                                  //立即执行
    glClear(GL_COLOR_BUFFER_BIT);         //清除屏幕
    getchar();
    Fractal(0.0,0.0,600.0,0.0,300.0,300.0*sqrt(3),6);       //调用分形函数
    glFlush();  //立即执行
}
int main(int argc,char ** argv)
{
    glutInit(&argc, argv);
    //初始化 GLUT 库 OpenGL 窗口的显示模式
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //初始化窗口的显示模式
```

```
glutInitWindowSize(600,600);                    //设置窗口的尺寸
glutInitWindowPosition(100,100);                //设置窗口的位置
glutCreateWindow("DDA 算法绘制直线");
cout<<"Bresenham 算法绘制直线"<<endl;
//输入直线的起点坐标
cout<<"please input the first point(x0,y0) of the line(range from 0 to 600): "<<endl;
        cin>>xf;
        cin>>yf;
//输入直线的终点坐标
      cout<<"please input the last point(x1,y1) of the line(range from 0 to 600): "<<endl;
cin>>xl;
cin>>yl;
   init();                          //初始化
glutDisplayFunc(lineSegment);  //执行画图程序
glutMainLoop();                 //启动主 GLUT 事件处理循环
}
```

3、实验结果

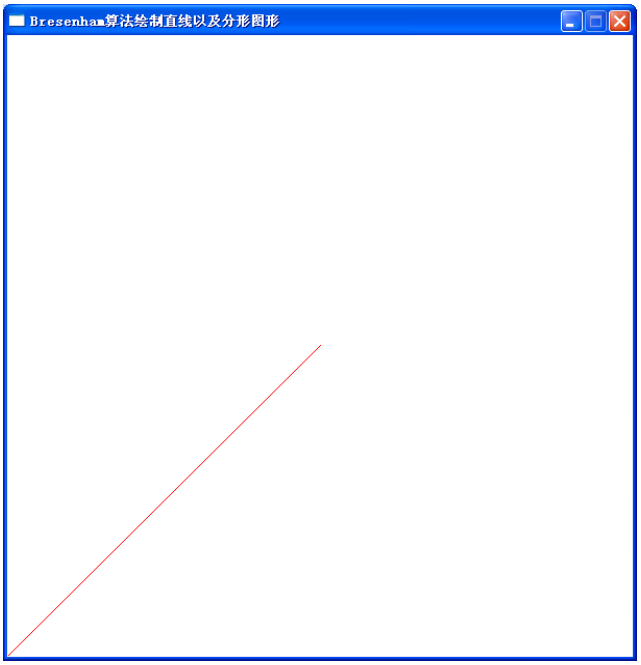

图 1-2 中点 Bresenham 算法绘制直线绘制结果

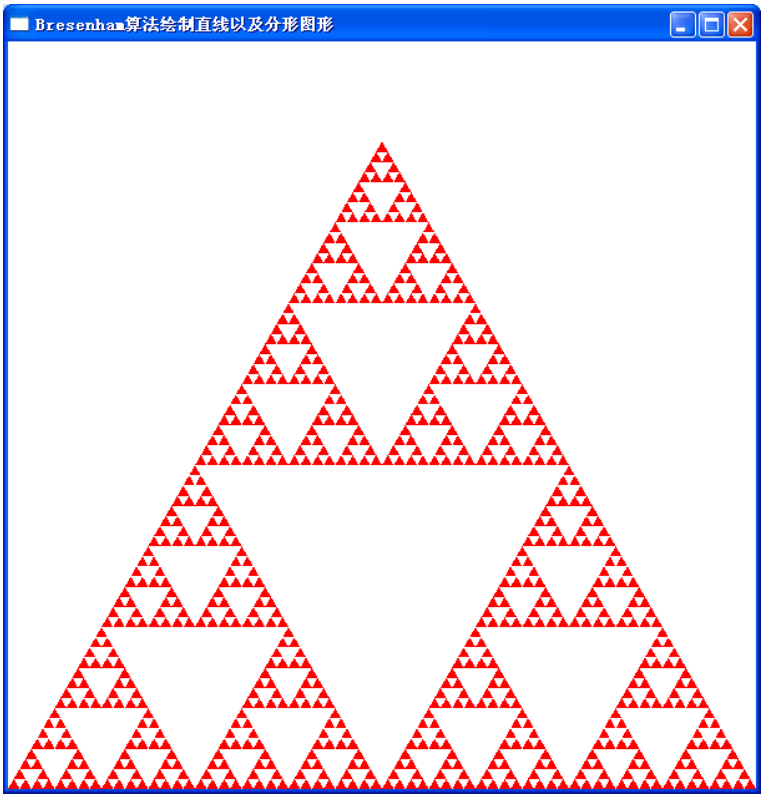

图 1-3 分形三角形递归 6 次结果

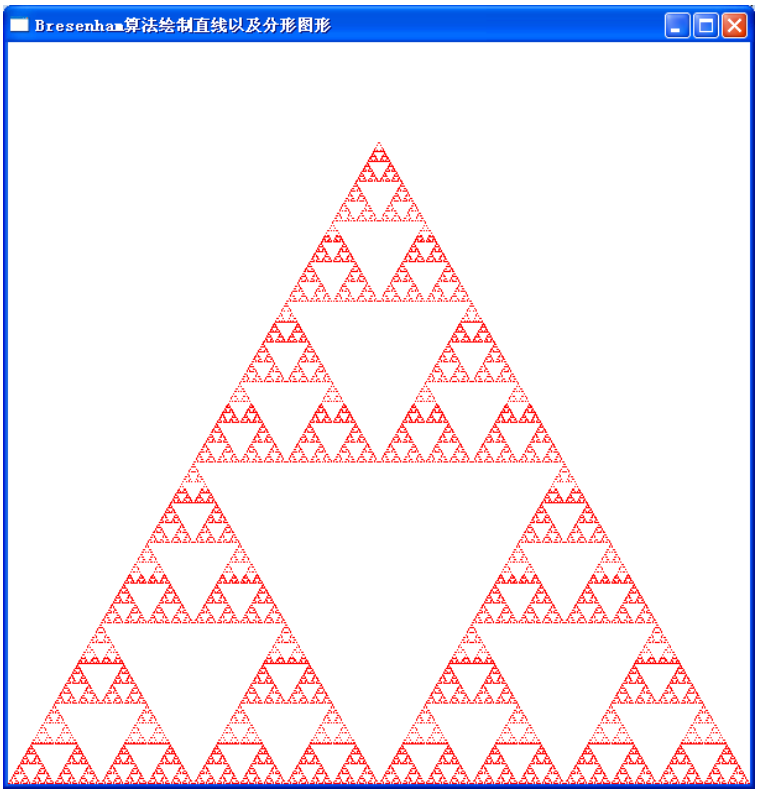

图 1-4 分形三角形递归 8 次结果

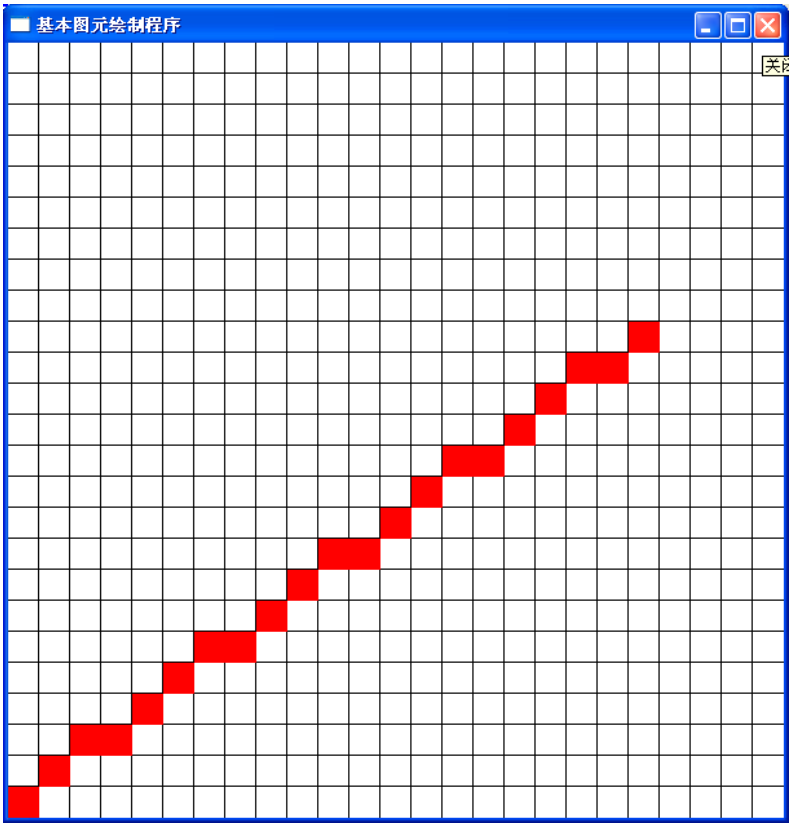

图 1-5 调大像素点后的 Bresenham 直线绘制结果

三、实验心得

本次上机是计算机图形学实验的第一次上机，总的来说算是完成的比较顺利。上机内容有两部分，第一部分是实现中点 Bresenham 画线算法，本来实验要求肯定是基于我们对次算法的理解来实现简单的算法，一开始我认为这个过程是比较轻松的，因为本身算法难度不大，但是到了取判式这一步时就出现了问题，由于对该算法理解不够深，很容易将系数弄错。之后迅速的纠正了错误。第二部分是分型图形的绘制。分形理论的最基本特点分数维度的视角和数学方法描述和研究客观事物，也就是用分形分维的数学工具来描述研究客观事物。它跳出了一维的线、二维的面、三维的立体乃至四维时空的传统藩篱，更加趋近复杂系统的真实属性与状态的描述，更加符合客观事物的多样性与复杂性。本次实验的要求为一个简单的分形三角形，发现其分布规律后根据谢尔宾斯基三角形的提出思想实现起来还是不难的。最后值得一提的是，老师过来想看出画出直线的锯齿状，于是原来的样例就出现了问题，这也是程序的不足，在另一个样例中对此作了补充，并且将图片也截图于实验结果中。希望以后学院能够重视图形学课程的建设，个人认为还是很重要的。

实验二：三维场景绘制

一、实验目的

建立带有光照模型的三维场景，三维场景的物体可以自定义，建议物体为法兰盘。场景里应具备三位观察点和视图变换，具备光照模型变化。

二、实验内容

1、实验算法

本次实验的目的为实现光照场景，即建立三维场景后可参考 opengl 中提供的光照设定的参数函数来设定光源。物体为了避免过于简单看不出效果，此处选择了一个茶壶，而且为了体现场景变化设置了多个茶壶。对于场景的变化，此处选择了 gluLookAt 这个视点变换函数，由于相对运动的关系，在实际实验时就可以感受光源的变化。此处唯一需要多做考虑的是函数中 lx、ly、lz 的值和转动角度的三角函数关系。整个过程在了解了参数的设定后并不难实现。

2、源程序

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <windows.h>



// angle of rotation for the camera direction
float angle=0.0;
// actual vector representing the camera's direction
float lx=0.0f,lz=-1.0f,ly=0.0f;
// XZ position of the camera
float x=0.0f,z=5.0f,y=0.0f;


void renderScene(void)
{

    // Clear Color and Depth Buffers

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset transformations
    glLoadIdentity();
```

```
	// Set the camera
	gluLookAt(	x, 1.0f, z,
				x+lx, y+ly,	z+lz,
				0.0f, 1.0f,	0.0f);

	// Draw ground
	glColor3f(0.9f, 0.9f, 0.9f);
	glBegin(GL_QUADS);
	glVertex3f(-100.0f, 0.0f, -100.0f);
	glVertex3f(-100.0f, 0.0f,	100.0f);
	glVertex3f( 100.0f, 0.0f,	100.0f);
	glVertex3f( 100.0f, 0.0f, -100.0f);
	glEnd();

	// Draw 36 teapot
	for(int i = -3; i < 3; i++)
		for(int j=-3; j < 3; j++)
		{
			glPushMatrix();
			glTranslatef(i*10.0,0,j * 10.0);

			glutSolidTeapot(1.0);
			glPopMatrix();
		}

	glutSwapBuffers();
}

void processSpecialKeys(int key, int xx, int yy)
{

	float fraction = 0.1f;

	switch (key)
	{						//Define specialkeys and the fluctuations from the key
	case GLUT_KEY_LEFT :		//key left
		angle -= 0.01f;
		lx = sin(angle);
		lz = -cos(angle);
		break;
	case GLUT_KEY_RIGHT :		//key right
		angle += 0.01f;
		lx = sin(angle);
		lz = -cos(angle);
```

```
            break;
        case GLUT_KEY_UP :                  //key up
            angle += 0.01f;
            ly = sin(angle);
            lz = -cos(angle);
            break;
        case GLUT_KEY_DOWN :                //key down
            angle -= 0.01f;
            ly = sin(angle);
            lz = -cos(angle);
            //x -= lx * fraction;
            //z -= lz * fraction;
            break;
        }
}



void init(void)

{
        // Initialize material property, light source, lighting model, * and depth buffer.
        GLfloat mat_ambient[] = {0.2, 0.2, 0.2, 1.0};
        GLfloat mat_diffuse[] = {0.8, 0.8, 0.8, 1.0} ;
        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 Light_Model_Ambient[] = { 0.2 , 0.2 , 0.2 , 1.0 };
        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);
        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_AMBIENT , Light_Model_Ambient ); //
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);

}

void changeSize(GLsizei w,GLsizei h)
```

```
{
    if(h==0) h=1;
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
        glOrtho(-5.5,5.5,-5.5*h/w,5.5*h/w,-10.0,10.0);
    else
        glOrtho(-5.5*w/h,5.5*w/h,-5.5,5.5,-10.0,10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}

int main(int argc, char** argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowSize (500, 500);

    glutInitWindowPosition (100, 100);

    glutCreateWindow (argv[0]);

    init ();
    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(renderScene);
    glutSpecialFunc(processSpecialKeys);

    glEnable(GL_DEPTH_TEST);

    return 0;

}
```

3、实验结果

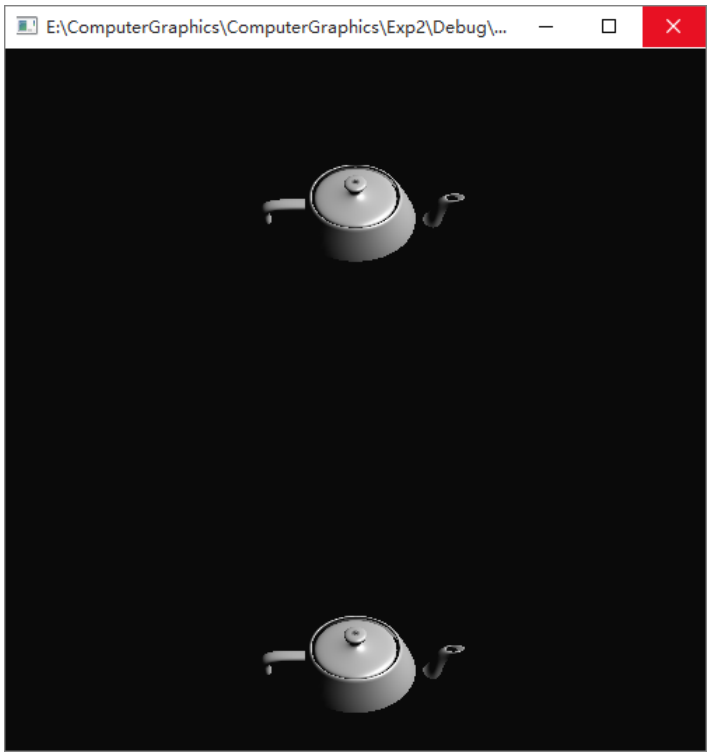

图 2-1 初始茶壶位置及表面光照

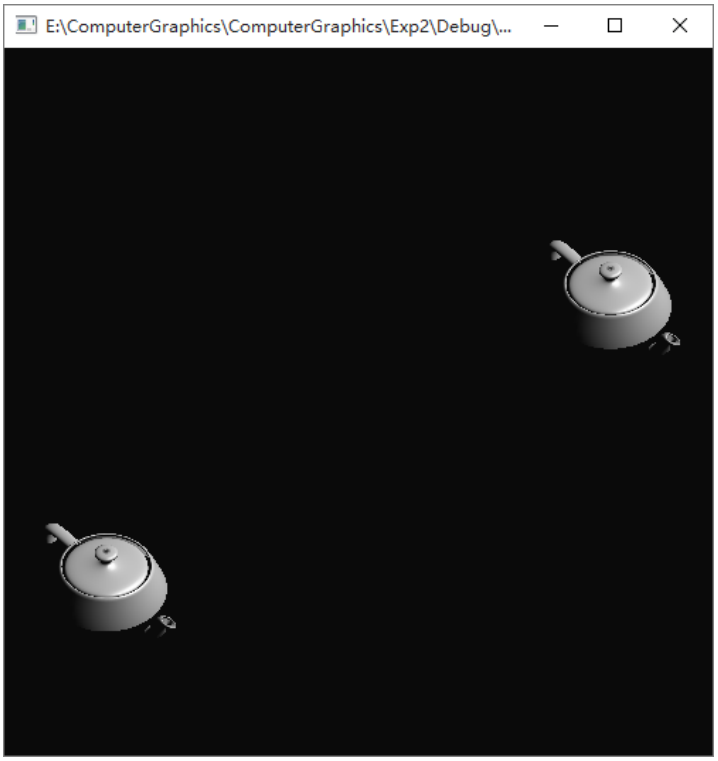

图 2-2 按方向键左后茶壶位置及表面光照变化

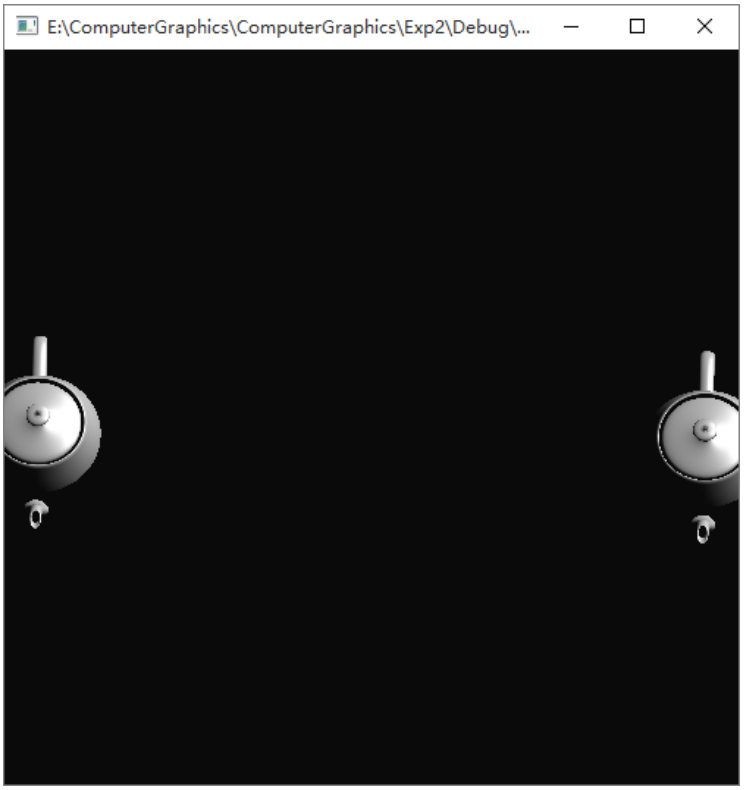

图 2-3 按方向键下后茶壶位置及表面光照变化

三、实验心得

本次实验的主要目的为绘制三维场景中的物体，难点在于增加光源和设置场景变换以达到物体表面光照变化的效果。一开始着手这个实验时其实是没有头绪的，其实也是没有认真看书的结果，全书最后一章其实介绍了关于 opengl 光源的相关设定。但是其实完全照搬树上也是有问题的，在对书上建立光源的过程仔细理解透彻之后，建立一个可以体现本实验中物体表面光源变化的可用光源不是很难。第二个难点在于实现场景的变化，此处考虑到图形库中提供好用的视角变换函数，在草稿上先建立一个虚拟的模型，在经过推算之后可以得出关于角度和增量的变化，一开始总调整不好转动的粒度，在慢慢调试之后旋转才变得平缓，也借助这种方法顺利的实现了光照的变化。其实毫无头绪的时候多去动手才是解决的办法，慢慢摸索就有方案了。再次希望学校重视图形学课程建设。