中 国 地 质 大 学

本科生课程论文

课程名称 计算机图形学 C

教师姓名 武云

本科生姓名 常文瀚

本科生学号 20181001095

本科生专业 计算机科学与技术

所 在 院 系 计算机学院

日 期： 2020 年 6 月 14 日

目 录

实验一：绘制椭圆形

一、实验目的与要求

1.目的：通过实践，理解中点画椭圆的算法如何运行，巩固课堂知识。

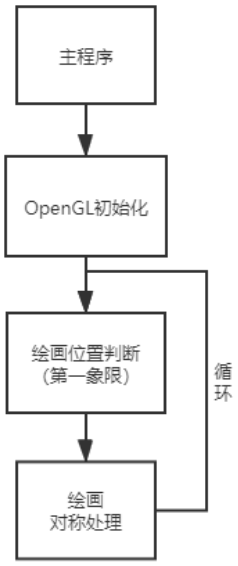
2.要求：在 Windows10 的环境下，使用 C++编程语言与 OpenGL 结合进行编程。

二、实验内容

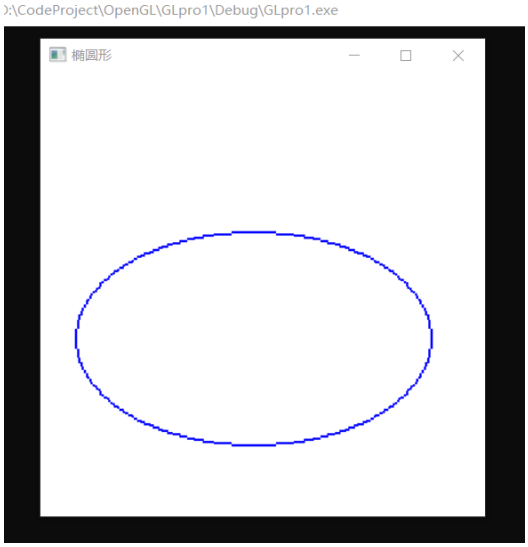
（1）模块设计：该程序包括绘制模块，对称处理模块，条件判断模块

（2）流程设计：先通过算法主体对绘制条件进行初步判断，计算出绘制的位置，在这之后调用像素绘制函数进行颜色的更改，其中需要对像素绘制函数进行相对于 X 轴与 Y 轴对称的处理，以达到该算法可以在四个象限同时绘制的效果。

（3）模块间的调用关系：



三、实验结果



1-1 绘制效果

四、体会

实验一是我对本学期圆与椭圆绘制算法的回顾，我希望可以循序渐进逐，渐加深难度通过三个实验来重新回忆起计算机图形学这门课程的知识。通过这个实验我重新回忆了 Bresenham 算法绘制直线以及通过 OpenGL 中点绘制圆与椭圆形算法的具体流程，与一些需要注意的细节，这些基本算法在工业生产中一定也十分重要，所以需要牢牢掌握。

五、源程序

```
#include<glut.h>
#include<gl/GL.h>
#include<cmath>
using namespace std;

void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
/*
    画点
*/
void setPixel(int x, int y)
{
    glColor3f(0.0, 0.0, 1.0);          // 蓝色
    glPointSize(2.0f);
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    glEnd();
    glFlush();
}
/*
    椭圆 4 路对称
*/
void Ellipsepot(int x0, int y0, int x, int y)
{

    setPixel((x0 + x), (y0 + y));
    setPixel((x0 + x), (y0 - y));
    setPixel((x0 - x), (y0 - y));
    setPixel((x0 - x), (y0 + y));
}
/*
    中点画椭圆算法
*/
```

```
void MidPoint_Ellipse(int x0, int y0, int a, int b)
{
    double sqa = a * a;
    double sqb = b * b;
    double d = sqb + sqa * (0.25 - b);
    int x = 0;
    int y = b;
    Ellipsepot(x0, y0, x, y);
    // 1
    while (sqb * (x + 1) < sqa * (y - 0.5))
    {
        if (d < 0)
        {
            d += sqb * (2 * x + 3);
        }
        else
        {
            d += (sqb * (2 * x + 3) + sqa * ((-2) * y + 2));
            --y;
        }
        ++x;
        Ellipsepot(x0, y0, x, y);
    }
    d = (b * (x + 0.5)) * 2 + (a * (y - 1)) * 2 - (a * b) * 2;
    // 2
    while (y > 0)
    {
        if (d < 0)
        {
            d += sqb * (2 * x + 2) + sqa * ((-2) * y + 3);
            ++x;
        }
        else
        {
            d += sqa * ((-2) * y + 3);
        }
        --y;
        Ellipsepot(x0, y0, x, y);
    }
}

// 窗口大小改变时调用的登记函数
void ChangeSize(GLsizei w, GLsizei h)
{
```

```
    if (h == 0)         h = 1;
    // 设置视区尺寸
    glViewport(0, 0, w, h);
    // 重置坐标系统
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // 建立修剪空间的范围
    if (w <= h)
        glOrtho(0.0f, 250.0f, 0.0f, 250.0f * h / w, 1.0, -1.0);
    else
        glOrtho(0.0f, 250.0f * w / h, 0.0f, 250.0f, 1.0, -1.0);
}

void display(void)
{
    // 用当前背景色填充窗口，如果不写这句会残留之前的图像
    glClear(GL_COLOR_BUFFER_BIT);

    int x0 = 120, y0 = 100, a = 100, b = 60;
    MidPoint_Ellipse(x0, y0, a, b);
}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(200, 200);
    glutInitWindowSize(400, 400);
    glutCreateWindow("椭圆形");
    glutDisplayFunc(display);
    glutReshapeFunc(ChangeSize);
    init();
    glutMainLoop();
    return 0;
}
```

实验二：绘制可移动茶壶

一、实验目的与要求

1.目的：通过编程学习了解光照与程序框大小改变后对原图像的处理，并初步认识如何处理鼠标的触发对源程序的影响。

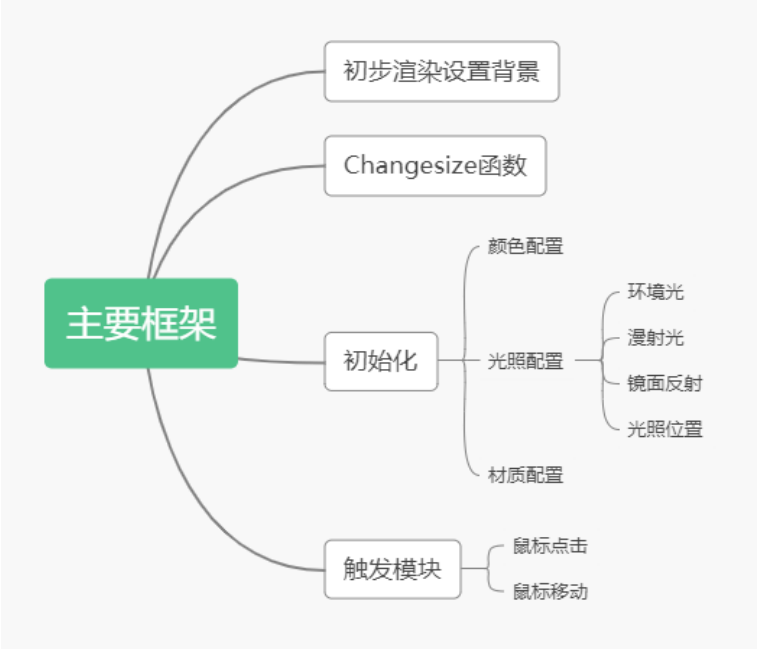
2.要求：在 Windows10 的环境下，使用 C++编程语言与 OpenGL 结合进行编程，尝试将 OpenGL 中处理光照与材质的函数添加到整个程序中。

二、实验内容

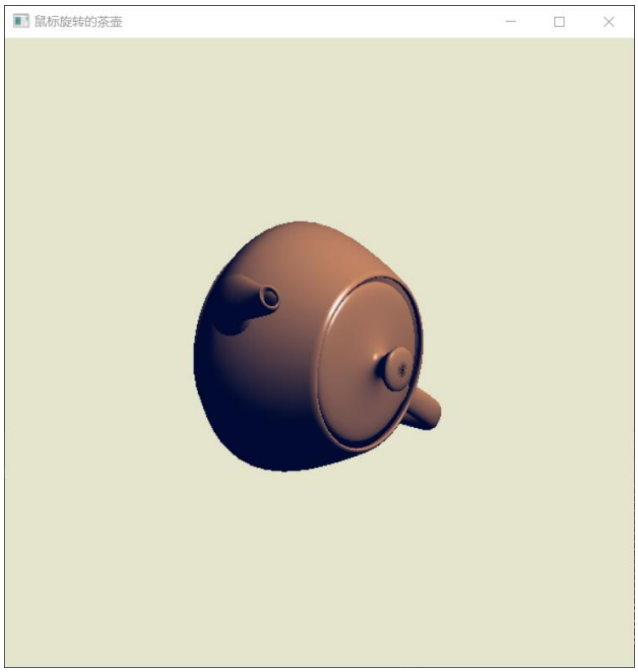
（1）模块设计：该程序包括初步渲染模块、窗口大小改变处理模块、图像初始化模块、触发模块

（2）流程设计：首先程序对画布进行初步渲染，设置好背景色，其次初始化茶壶的颜色、光照、材质，光照的初始化总包括了：环境光、漫射光、镜面反射、光照位置四个部分，同时 Changesize 函数要为程序框的大小变化处理做好准备，最后可以用触发模块应对鼠标对茶壶的拖动等操作，光照也会随之变化。

（3）模块间的调用关系：

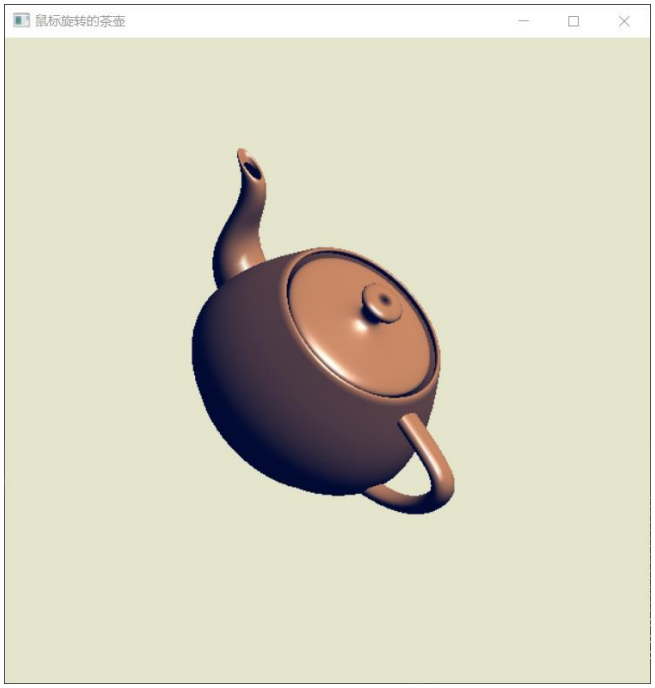


三、实验结果



2-1 绘制效果 1

2-2 绘制效果 2



2-3 绘制效果 3

四、体会

这一题的处理相比与之前课内的所有试验都更加繁琐一些，因为涉及到了光照、材质、颜色的同时处理，接触到了很多之前未曾使用过的 OpenGL 的函数。在不断查询网上教程、函数解释后终于实现了预期效果。OpenGL 的相关技术会在游戏研发的很多领域用到，对光照的渲染和触发机制的调整仅仅一个茶壶就会耗费很多时间，研发游戏时并对其进行测试时程序员耗费的精力一定更多，更加辛苦，我不由得对那些在大型企业进行游戏开发的程序员们肃然起敬，他们坚持不懈的精神值得学习

五、源代码

```
#include<glut.h>
GLfloat rx = 0, ry = 0, angle = 0;
GLint winWidth = 600, winHeight = 600;
GLint mouseDx, mouseDy;

//////////显示函数
void Display() {
	////////GL_COLOR_BUFFER_BIT(用背景颜色填充)
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glRotatef(1, rx, ry, 0);
	glColor3f(0.0, 0.0, 1.0);
	glutSolidTeapot(120);
		///交换缓冲区
	glutSwapBuffers();
}

// 设置渲染状态
void SetupRC(void)
{
	glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
}

void ChangeSize(int w, int h)
{
	winWidth = w;
	winHeight = h;
	// 设置观察视野为窗口大小（用 FLASH 里面的话来说应该叫设置摄象机视野）
	glViewport(0, 0, w, h);
	// 重置坐标系统，指定设置投影参数
	glMatrixMode(GL_PROJECTION);
	///////调用单位矩阵，去掉以前的投影参数设置
	glLoadIdentity();
	//////设置投影参数
	glOrtho(-w / 2, w / 2, -h / 2, h / 2, -w, w);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

void init() //初始化背景颜色，光照，材质等
{
	glClearColor(0.9, 0.9, 0.8, 1.0); //初始背景色
	/********* 光照处理 **********/
	GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
```

```
    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position0[] = { 100.0, 100.0, 100.0 ,0 };
    //定义光位置得齐次坐标(x,y,z,w),如果 w=1.0,为定位光源（也叫点光源），
    //如果 w＝0，为定向光源（无限光源），定向光源为无穷远点，因而产生光为
    //平行光。
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient); //环境光
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse); //漫射光
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular); //镜面反射
    glLightfv(GL_LIGHT0, GL_POSITION, light_position0); //光照位置
    /******** 材质处理 ***********/

    GLfloat mat_ambient[] = { 0.0, 0.2, 1.0, 1.0 };
    GLfloat mat_diffuse[] = { 0.8, 0.5, 0.2, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 100.0 }; //材质 RGBA 镜面指数，数值在 0～128 范围内

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glEnable(GL_LIGHTING); //启动光照
    glEnable(GL_LIGHT0); //使第一盏灯有效
    glEnable(GL_DEPTH_TEST); //测试深度缓存
}

void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse) {
    if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
        mouseDx = xMouse;
        mouseDy = yMouse;
    }
    if (button == GLUT_LEFT_BUTTON && action == GLUT_UP) {
        glutPostRedisplay();
    }
}

////////////////鼠标移动
void MouseMove(GLint xMouse, GLint yMouse) {
    //angle+=2;
    if (xMouse > mouseDx&& yMouse == mouseDy)
    {
        rx = 0; ry = 1;
    }
    else if (xMouse < mouseDx && yMouse == mouseDy)
```

```
    {
        rx = 0; ry = -1;
    }
    else if (xMouse == mouseDx && yMouse < mouseDy)
    {
        rx = -1; ry = 0;
    }
    else if (xMouse == mouseDx && yMouse > mouseDy)
    {
        rx = 1; ry = 0;
    }
    else if (xMouse > mouseDx&& yMouse < mouseDy)
    {
        rx = -1; ry = 1;
    }
    else if (xMouse > mouseDx&& yMouse > mouseDy)
    {
        rx = 1; ry = 1;
    }
    else if (xMouse < mouseDx && yMouse < mouseDy)
    {
        rx = -1; ry = -1;
    }
    else if (xMouse<mouseDx && yMouse>mouseDy)
    {
        rx = 1; ry = -1;
    }
    mouseDx = xMouse;
    mouseDy = yMouse;
    glutPostRedisplay();
}

// 主函数
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    //设置显示模式
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    //设置窗口大小像素
    glutInitWindowSize(600, 600);
    ////设置窗口出现在屏幕的位置
    glutInitWindowPosition(300, 160);
    //建立一个叫 OpenGL 的窗口
    glutCreateWindow("鼠标旋转的茶壶");
```

```
    //调用函数 Display 进行绘制
    glutDisplayFunc(Display);
    //////调用鼠标移动函数
    //glutPassiveMotionFunc(PassiveMouseMove);
    glutMotionFunc(MouseMove);
    //////调用鼠标点击函数
    glutMouseFunc(MousePlot);
    //如果窗口大小改变则调用函数 ChangeSize 重新进行绘制
    glutReshapeFunc(ChangeSize);
    init();
    glutMainLoop();
    return 0;
}
```

实验三：绘制椭圆形

一、实验目的与要求

1.目的：实现 Windows XP 系统的经典的锁屏 Windows 标志移动效果

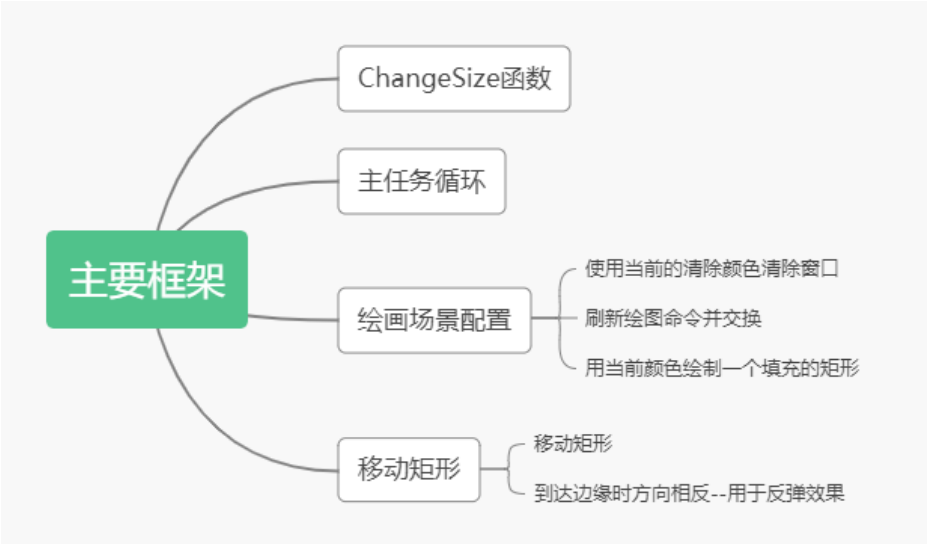
2.要求：在 Windows10 的环境下，使用 C++编程语言与 OpenGL 结合进行编程。

二、实验内容

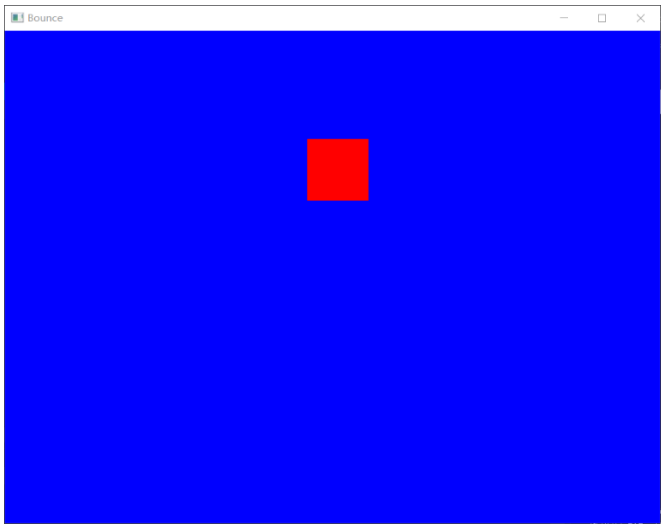
（1）模块设计：该程序包括绘画场景配置模块、移动矩形模块、ChangeSize 模块、主任务循环模块

（2）流程设计：先进行绘画场景配置，其中包括清除窗口、刷新绘图命令并交换，其次 TimerFunction 函数中包含了反弹效果的实现，并可以使小的矩形不断移动，ChangeSize 模块保证了当窗口大小发生人为改变时可以做到重新规划小矩形的大小和运行轨迹，最终主事件循环模块可以让整个程序一直运行下去。

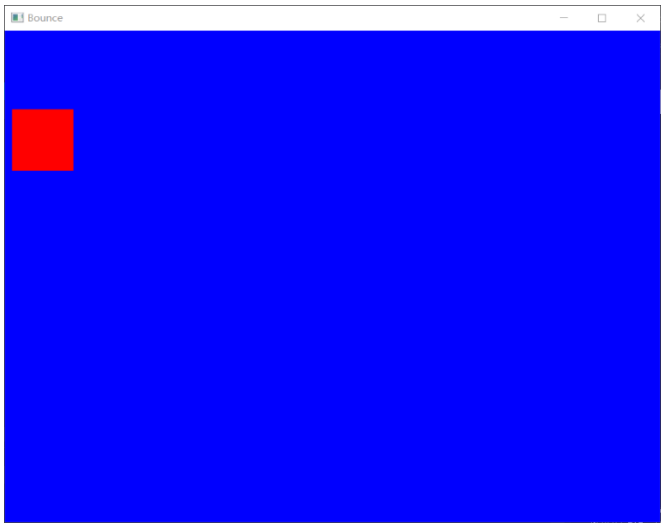
（3）模块间的调用关系：



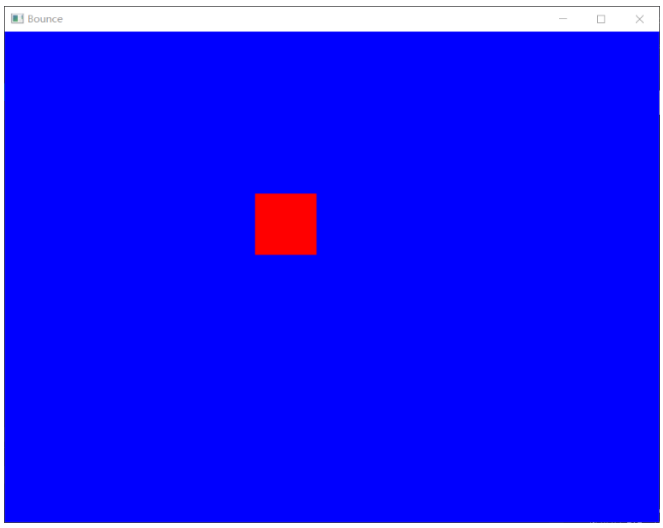
三、实验结果



3-1 运行效果 1



3-2 运行效果 2



3-3 运行效果 3

四、体会

通过这次为数不多的几天计算机实践学习，我巩固了一些关于 C++的知识，理解了我们计算机图形学的理论知识，这对我们将来到社会工作将会有莫大的帮助。同时它让我知道计算机图形的强大和瑰丽之处，虽然我们学的都是基本的生成算法，但是通过学习到的几个计算机图形学高级程序，我们才了解到计算机图形学可以做出非常华丽的视觉效果，而只要你努力，任何东西都不会太难。

最后，还是很庆幸能学到计算机图形学这样的一门课程，在学习本课程的同时，已经涉及了很多的学科，让我提升编程能力、思维能力。

五、源代码

```
#include <glut.h>        // OpenGL toolkit
#include<gl/GL.h>
#include<gl/GLU.h>

// 初始化矩形位置和大小
GLfloat x = 0.0f;
GLfloat y = 0.0f;
GLfloat rsize = 25;

// x 和 y 方向上的步长
// (每次移动的像素)
GLfloat xstep = 1.0f;
GLfloat ystep = 1.0f;

// 跟随窗口改变长宽
GLfloat windowWidth;
GLfloat windowHeight;

////////////////////////////////////////////////////////
// 绘画场景
void RenderScene(void)
{
    // 使用当前的清除颜色清除窗口
    glClear(GL_COLOR_BUFFER_BIT);

    // 设置小方块颜色为红色
    glColor3f(1.0f, 0.0f, 0.0f);

    // 用当前颜色绘制一个填充的矩形
    glRectf(x, y, x + rsize, y - rsize);

    // Flush drawing commands and swap
    //刷新绘图命令并交换
```

```
    glutSwapBuffers();
}

////////////////////////////////////////////////////////
// 空闲时由 GLUT 库调用（不调整窗口大小或移动窗口）
void TimerFunction(int value)
{
    // Reverse direction when you reach left or right edge
    //到达左边缘或右边缘时方向相反--用于反弹效果
    if (x > windowWidth - rsize || x < -windowWidth)
        xstep = -xstep;

    // Reverse direction when you reach top or bottom edge
    //到达上边缘或下边缘时方向相反--用于反弹效果
    if (y > windowHeight || y < -windowHeight + rsize)
        ystep = -ystep;

    // Actually move the square
    //移动矩形
    x += xstep;
    y += ystep;

    // Check bounds. This is in case the window is made
    // smaller while the rectangle is bouncing and the
    // rectangle suddenly finds itself outside the new
    // clipping volume
    //检查边界，这是在窗口被建立的更小
    //当长方形在弹跳并且长方形忽然发现他跑到了新剪裁量外
    if (x > (windowWidth - rsize + xstep))
        x = windowWidth - rsize - 1;
    else if (x < -(windowWidth + xstep))
        x = -windowWidth - 1;

    if (y > (windowHeight + ystep))
        y = windowHeight - 1;
    else if (y < -(windowHeight - rsize + ystep))
        y = -windowHeight + rsize - 1;

    // Redraw the scene with new coordinates
    glutPostRedisplay();
    glutTimerFunc(33, TimerFunction, 1);      //需要在函数中再调用一次，才能保证循环
}

////////////////////////////////////////////////////////
```

```
// Setup the rendering state
void SetupRC(void)
{
    // Set clear color to blue
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

////////////////////////////////////////////////////////
// Called by GLUT library when the window has chanaged size
//修改窗口大小
void ChangeSize(int w, int h)
{
    GLfloat aspectRatio;

    // Prevent a divide by zero
    if (h == 0)
        h = 1;

    // Set Viewport to window dimensions
    //将视口设置为窗口尺寸
    glViewport(0, 0, w, h);

    // Reset coordinate system
    //重置坐标系
    glMatrixMode(GL_PROJECTION);      //GL_PROJECTION--投影模式
    glLoadIdentity();          //将当前点移到了屏幕中心：类似于一个复位操作

    // Establish clipping volume (left, right, bottom, top, near, far)
    aspectRatio = (GLfloat)w / (GLfloat)h;

    //glOrtho(left, right, bottom, top, near, far)
    //left 表示视景体左面的坐标, right 表示右面的坐标, bottom 表示下面的, top 表示上面的
    if (w <= h)
    {
        windowWidth = 100;
        windowHeight = 100 / aspectRatio;
        glOrtho(-100.0, 100.0, -windowHeight, windowHeight, 1.0, -1.0);
    }
    else
    {
        windowWidth = 100 * aspectRatio;
        windowHeight = 100;
        glOrtho(-windowWidth, windowWidth, -100.0, 100.0, 1.0, -1.0);
    }
```

```
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

////////////////////////////////////////////////////////
// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Bounce");
    glutDisplayFunc(RenderScene);     //设置绘画场景
    glutReshapeFunc(ChangeSize);      //窗口大小改变时调用
    glutTimerFunc(33, TimerFunction, 1);      //定时器，33 毫秒，回调函数指针 TimerFunction,区别值
    SetupRC();   //Set clear color to blue
    glutMainLoop();        // 让主事件无限循环
    return 0;
}
```