**计算机图形学**

**实验报告**

**exp2**

**目录**

[**Task1** 1](#_Toc509145986)

[功能说明： 1](#_Toc509145987)

[效果预览： 1](#_Toc509145988)

[源代码： 1](#_Toc509145989)

[**Task2** 5](#_Toc509145990)

[功能说明： 5](#_Toc509145991)

[效果预览： 5](#_Toc509145992)

[源代码： 5](#_Toc509145993)

**Task1**

功能说明：

实现了三角形的变色，缩放（切分精度）以及旋转功能

效果预览：

注：由于动态旋转，故没有截取到初始状态

另：为了保证旋转的流畅度，颜色的改变速度略快，可能略有观感不适

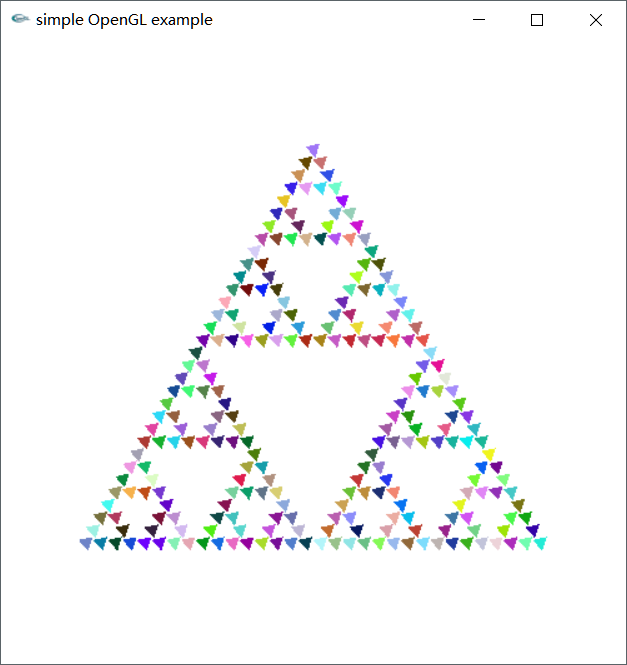
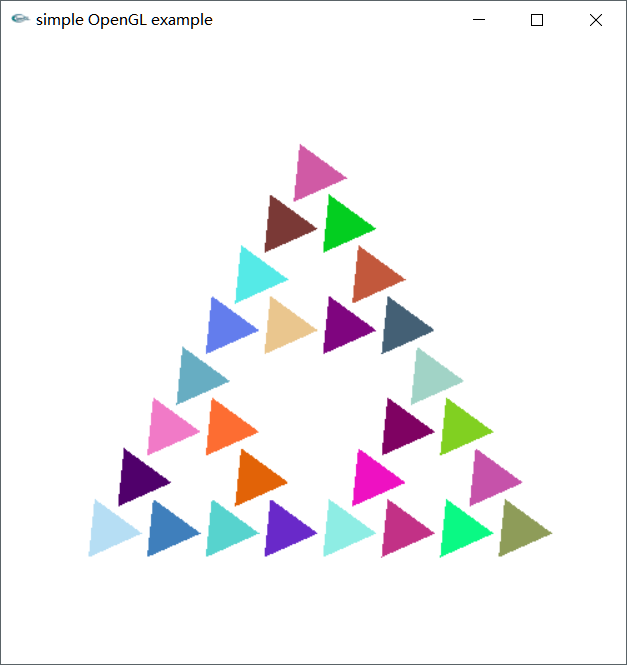
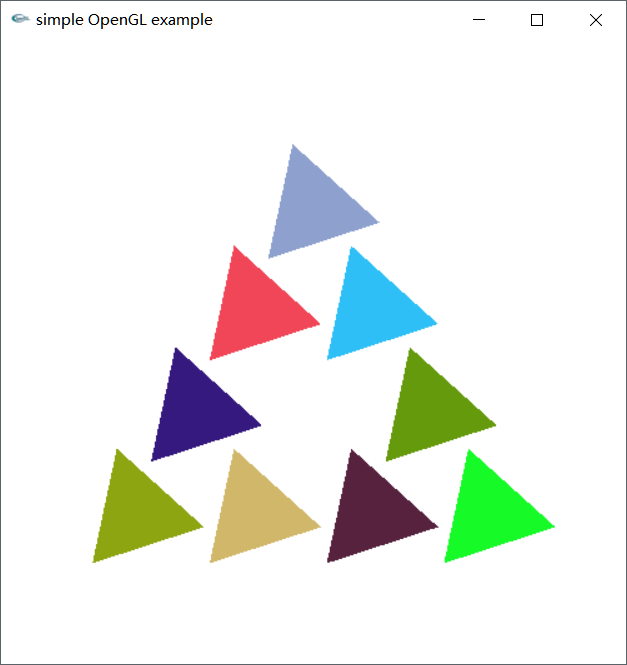


图1-图4.Task1程序预览

源代码：

/\*

\* Sierpinski

\* 实现了变色、缩放（改变切分层数）、旋转

\* 为了保证旋转的流畅性，颜色改变频率较快，略有不适

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <cmath>

#include <cstdlib>

#include <gl/glut.h>

//设置PI值

const double PI = 3.14159265358979323846;

//设置PI/30

const double PIDIV30 = PI / 30.0;

//设置初始旋转角

const double ANGLE1 = PI / 2;

const double ANGLE2 = PI \* 7 / 6;

const double ANGLE3 = PI \* 11 / 6;

//边长为60的等边三角形

GLfloat vertices[3][2] = { { 10.0,15.0 },{ 40.0, 66.9615 },{ 70.0, 15.0 } };

//标记三角形中心到定点的距离是否已经计算

bool beSetValue;

//三角形中心到顶点的距离

double coreDistant;

//旋转比例：三角形旋转角相对PI/30的倍数

int shapeAngle = -1;

//指定切分次数以改变缩放

int sizeNum;

//设置属性

void myinit() {

//设置背景为白色

glClearColor(1.0, 1.0, 1.0, 1.0);

//设置观察参数

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

gluOrtho2D(0.0, 80.0, 0.0, 80.0);

glMatrixMode(GL\_MODELVIEW);

}

//指定划分次数为[2,5]次

int setSizeNum() {

int sizeNum = rand() % 4 + 2;

return sizeNum;

}

//获取随机颜色值，各保留三位小数

void setColor() {

int rand();

double colorR = rand() % 1000 / 1000.0;

double colorG = rand() % 1000 / 1000.0;

double colorB = rand() % 1000 / 1000.0;

glColor3f(colorR, colorG, colorB);

}

//获取三角形中心到顶点的距离

double getDistCoreToVer(GLfloat \*a, GLfloat \*b) {

double sideLen = sqrt((a[0] - b[0])\*(a[0] - b[0]) + (a[1] - b[1])\*(a[1] - b[1]));

return sideLen \* sqrt(3.0) / 3.0;

}

//设置旋转以后的坐标点

void changePosition(GLfloat \*core) {

GLfloat a[2], b[2], c[2];

a[0] = core[0] + cos(ANGLE1 + shapeAngle \* PIDIV30)\*coreDistant;

a[1] = core[1] + sin(ANGLE1 + shapeAngle \* PIDIV30)\*coreDistant;

b[0] = core[0] + cos(ANGLE2 + shapeAngle \* PIDIV30)\*coreDistant;

b[1] = core[1] + sin(ANGLE2 + shapeAngle \* PIDIV30)\*coreDistant;

c[0] = core[0] + cos(ANGLE3 + shapeAngle \* PIDIV30)\*coreDistant;

c[1] = core[1] + sin(ANGLE3 + shapeAngle \* PIDIV30)\*coreDistant;

glVertex2fv(a);

glVertex2fv(b);

glVertex2fv(c);

}

//绘制三角形顶点

void triangle(GLfloat \*a, GLfloat \*b, GLfloat \*c) {

if (!beSetValue) {

coreDistant = getDistCoreToVer(a, b);

beSetValue = true;

}

//获取中心坐标

GLfloat core[2] = { (a[0] + b[0] + c[0]) / 3.0, (a[1] + b[1] + c[1]) / 3.0 };

//设置旋转以后的坐标点

changePosition(core);

}

//切分三角形

void divideTriangle(GLfloat \*a, GLfloat \*b, GLfloat \*c, int k) {

GLfloat ab[2], ac[2], bc[2];

int j;

if (k > 0) {

for (j = 0; j < 2; j++) ab[j] = (a[j] + b[j]) / 2;

for (j = 0; j < 2; j++) ac[j] = (a[j] + c[j]) / 2;

for (j = 0; j < 2; j++) bc[j] = (b[j] + c[j]) / 2;

divideTriangle(a, ab, ac, k - 1);

divideTriangle(c, ac, bc, k - 1);

divideTriangle(b, bc, ab, k - 1);

}

else {

setColor();

triangle(a, b, c);

}

}

void display() {

//改变旋转角比重

shapeAngle = (shapeAngle + 1) % 20;

if (shapeAngle == 0) {

//重置标记

beSetValue = false;

sizeNum = setSizeNum();

}

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT);

//开始绘图

glBegin(GL\_TRIANGLES);

divideTriangle(vertices[0], vertices[1], vertices[2], sizeNum);

glEnd();

glFlush();

//交换通道

glutSwapBuffers();

}

//设置动画效果

void changeTriangle(int value) {

display();

glutTimerFunc(100, changeTriangle, 1);

}

//按下Esc退出

void pressToExit(unsigned char key, int x, int y) {

if (key == 27)

exit(0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(pressToExit);

glutDisplayFunc(display);

glutTimerFunc(100, changeTriangle, 1);

glutMainLoop();

return 0;

}

**Task2**

功能说明：

实现了砖块移动和小球的反弹以及失败判定。

为了模拟物理效果，实现了砖块移动对反弹角度的影响。

效果预览：

注：键盘操作时，响应略有延迟，暂未找到原因

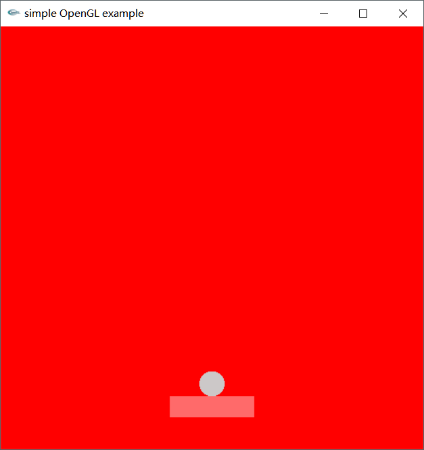
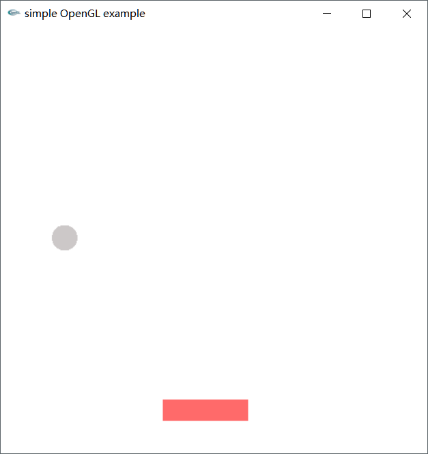
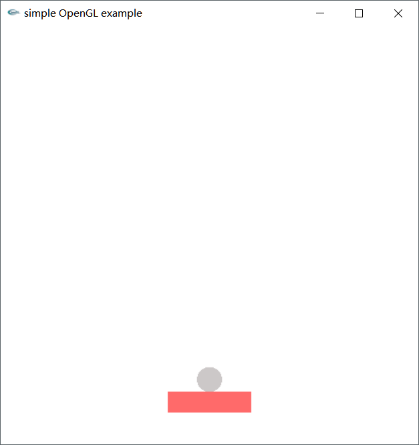


图5.开始界面 图6.游戏界面 图7.失败提示

源代码：

/\*

\* 砖块挡反弹球

\* 实现了砖块的移动和球的反弹以及游戏失败的判断

\* 为了模拟实际物理效果，实现了小球在运动的砖块上反弹时，反射角的改变

\* 键盘响应速度慢，暂未找到原因

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <cmath>

#include <cstdlib>

#include <gl/glut.h>

//设置砖块中心

GLfloat brickCenter[2];

//设置砖块宽度一半

const GLfloat HALFBRICKWIDTH = 10;

//设置砖块高度一半

const GLfloat HALFBRICKHEIGHT = 2.5;

//设置砖块所在高度

const GLfloat BRICKHEIGHT = 12.5;

//砖块颜色

const GLfloat BRICKCOLOR[3] = { 1, 106 / 255.0, 106 / 255.0 };

//设置砖块移动幅度

const GLfloat BRICKSTEP = 2;

//设置砖块所能到达的边界

const GLfloat BRICKLEFTSIDE = HALFBRICKWIDTH;

const GLfloat BRICKRIGHTSIDE = 100 - HALFBRICKWIDTH;

//设置PI值

const GLfloat PI = 3.14159;

//设置直角

const GLfloat HALFPI = PI / 2;

//设置小球绘制精度

const int N = 100;

//设置小球中心

GLfloat ballCenter[2];

//设置小球半径

const GLfloat BALLRADIUS = 3;

//设置小球初始高度

const GLfloat INITBALLHEIGHT = BRICKHEIGHT + BALLRADIUS;

//设置小球所能到达的边界

const GLfloat BALLLEFTSIDE = BALLRADIUS;

const GLfloat BALLRIGHTSIDE = 100 - BALLRADIUS;

const GLfloat BALLMAXHEIGHT = 100 - BALLRADIUS;

//设置小球颜色

const GLfloat BALLCOLOR[3] = { 205 / 256.0, 201 / 256.0, 201 / 256.0 };

//设置小球移动幅度

const GLfloat BALLSTEP = 0.15;

//设置小球移动方向

GLfloat ballDirectionVector[2];

//设置角度改变幅度

const GLfloat ANGLESTEP = PI / 18;

//记录当前移动角度

GLfloat nowAngle;

//设置游戏开始标记

bool startGame = false;

//设置左方向键按下标记

bool leftOn = false;

//设置右方向键按下标记

bool rightOn = false;

//设置刚刚发球的标记

//防止小球在砖块上反复改变角度

bool justStart = false;

//改变小球中心位置

void setBallCenter(GLfloat x, GLfloat y) {

ballCenter[0] = x;

ballCenter[1] = y;

}

//改变砖块中心位置

void setBrickCenter(GLfloat x, GLfloat y = 10) {

brickCenter[0] = x;

brickCenter[1] = y;

}

//改变小球移动方向

void setBallDirectionVector(GLfloat angle) {

nowAngle = angle;

ballDirectionVector[0] = cos(angle);

ballDirectionVector[1] = sin(angle);

}

//判断小球是否成功被接到

bool beingTouch() {

bool heightSatisfied1 = (ballCenter[1] - INITBALLHEIGHT <= BALLSTEP);

bool heightSatisfied2 = (ballCenter[1] - INITBALLHEIGHT >= 0);

bool widthSatisfied = (abs(ballCenter[0] - brickCenter[0]) <= HALFBRICKWIDTH);

return heightSatisfied1 && heightSatisfied2 && widthSatisfied;

}

//判断小球是否未被接到但与砖块接触

bool missButTouch() {

bool heightSatisfied = (abs(brickCenter[1] - ballCenter[1]) < HALFBRICKHEIGHT);

bool widthSatisfied1 = (abs(ballCenter[0] - brickCenter[0]) < (HALFBRICKWIDTH + BALLRADIUS));

bool widthSatisfied2 = (abs(ballCenter[0] - brickCenter[0]) > HALFBRICKWIDTH);

return heightSatisfied && widthSatisfied1 && widthSatisfied2;

}

//设置初始属性

void myinit() {

//设置背景为白色

glClearColor(1.0, 1.0, 1.0, 1.0);

//设置观察参数

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

gluOrtho2D(0.0, 100.0, 0.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

//设置砖块和小球初始位置

setBrickCenter(50, 10);

setBallCenter(50, INITBALLHEIGHT);

}

void displayScene() {

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT);

//设置小球颜色

glColor3f(BALLCOLOR[0], BALLCOLOR[1], BALLCOLOR[2]);

//开始绘制小球

glBegin(GL\_POLYGON);

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

glVertex2f(BALLRADIUS\*cos(2 \* PI / N \* i) + ballCenter[0], BALLRADIUS\*sin(2 \* PI / N \* i) + ballCenter[1]);

glEnd();

//设置砖块颜色

glColor3f(BRICKCOLOR[0], BRICKCOLOR[1], BRICKCOLOR[2]);

//开始绘制砖块

glBegin(GL\_POLYGON);

glVertex2f(brickCenter[0] - HALFBRICKWIDTH, brickCenter[1] + HALFBRICKHEIGHT);

glVertex2f(brickCenter[0] + HALFBRICKWIDTH, brickCenter[1] + HALFBRICKHEIGHT);

glVertex2f(brickCenter[0] + HALFBRICKWIDTH, brickCenter[1] - HALFBRICKHEIGHT);

glVertex2f(brickCenter[0] - HALFBRICKWIDTH, brickCenter[1] - HALFBRICKHEIGHT);

glEnd();

glFlush();

//交换通道

glutSwapBuffers();

}

//设置小球移动轨迹

void setBallTrail() {

if (!startGame)

return;

GLfloat x = ballCenter[0] + ballDirectionVector[0] \* BALLSTEP;

GLfloat y = ballCenter[1] + ballDirectionVector[1] \* BALLSTEP;

if (x<(BALLLEFTSIDE + BALLSTEP) || x>(BALLRIGHTSIDE - BALLSTEP))

setBallDirectionVector(PI - nowAngle);

else if (y > (BALLMAXHEIGHT - BALLSTEP))

setBallDirectionVector(2 \* PI - nowAngle);

setBallCenter(x, y);

if (leftOn || rightOn)

return;

if (justStart) {

justStart = false;

//设置背景为白色

glClearColor(1.0, 1.0, 1.0, 1.0);

return;

}

if (beingTouch()) {

//成功接到球的情况

setBallDirectionVector(2 \* PI - nowAngle);

}else if (missButTouch()) {

//未成功接球但触碰到球的情况

setBallDirectionVector(PI - nowAngle);

}

}

//设置小球动画效果

void modifyScene(int value) {

setBallTrail();

displayScene();

glutTimerFunc(1, modifyScene, 1);

}

//响应左方向键

void responLeft() {

GLfloat newPosition = brickCenter[0] - BRICKSTEP;

if (newPosition >= BRICKLEFTSIDE)

setBrickCenter(newPosition);

if (!startGame) {

justStart = true;

startGame = true;

setBallDirectionVector(3 \* PI / 4);

}else if (beingTouch()) {

//成功接到球的情况

setBallDirectionVector(2 \* PI - nowAngle + ANGLESTEP);

}else if (missButTouch()) {

//未成功接球但触碰到球的情况

setBallDirectionVector(PI - nowAngle);

}

}

//响应右方向键

void responRight() {

GLfloat newPosition = brickCenter[0] + BRICKSTEP;

if(newPosition <= BRICKRIGHTSIDE)

setBrickCenter(newPosition);

if (!startGame) {

justStart = true;

startGame = true;

setBallDirectionVector(PI / 4);

}else if (beingTouch()){

//成功接到球的情况

setBallDirectionVector(2 \* PI - nowAngle - ANGLESTEP);

}else if (missButTouch()) {

//未成功接球但触碰到球的情况

setBallDirectionVector(PI - nowAngle);

}

}

//控制砖块移动

void controlGame(int key, int x, int y) {

switch (key) {

//点击键盘左右控制砖块方向

case GLUT\_KEY\_LEFT:

leftOn = true;

responLeft();

leftOn = false;

break;

case GLUT\_KEY\_RIGHT:

rightOn = true;

responRight();

rightOn = false;

break;

default:

break;

}

}

void gameFailed() {

if (ballCenter[1] >= -BALLRADIUS)

return;

//重置标记

startGame = false;

//设置背景为红色

glClearColor(1.0, 0, 0, 1.0);

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT);

//设置砖块和小球初始位置

setBrickCenter(50, 10);

setBallCenter(50, INITBALLHEIGHT);

}

//按下Esc退出

void pressToExit(unsigned char key, int x, int y) {

if (key == 27)

exit(0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(pressToExit);

glutSpecialFunc(controlGame);

glutDisplayFunc(displayScene);

glutTimerFunc(5, modifyScene, 1);

glutIdleFunc(gameFailed);

glutMainLoop();

return 0;

}