**计算机图形学实验**

**姓 名： 王印**

**学 号：20201050470**

**专 业：计算机科学于技术**

**教 师：钱文华**

实验一 直线段生成算法

时间：2022.3.16

地点：信息学院2202

1. 实验内容：

熟悉OPENGL，通过实例程序生成直线段

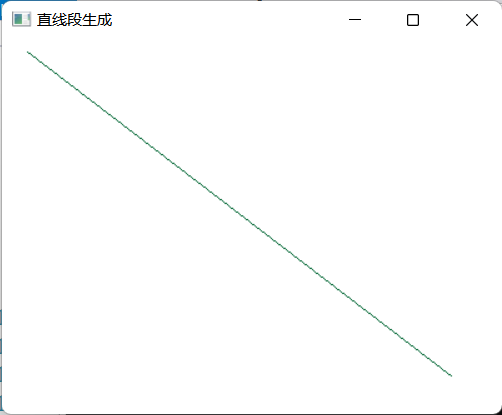
1. 实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

1. 实验代码：

|  |
| --- |
| //直线段生成算法  #include <GL/glut.h>  void init(void){  glClearColor(1.0, 1.0, 1.0, 0.0);  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0.0, 200.0, 0.0, 150.0);  }  void lineSegment(void) {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.4, 0.2);  glBegin(GL\_LINES);  glVertex2i(180, 15);  glVertex2i(10, 145);  glEnd();  glFlush();  }  void main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(50, 100);  glutInitWindowSize(400, 300);  glutCreateWindow("直线段生成");  init();  glutDisplayFunc(lineSegment);  glutMainLoop();  } |

4、实验结果：



实验二 DDA直线生成算法

时间：2022.3.23

地点：信息学院2202

1、实验内容：

熟悉OPENGL，通过DDA、中点算法生成直线段

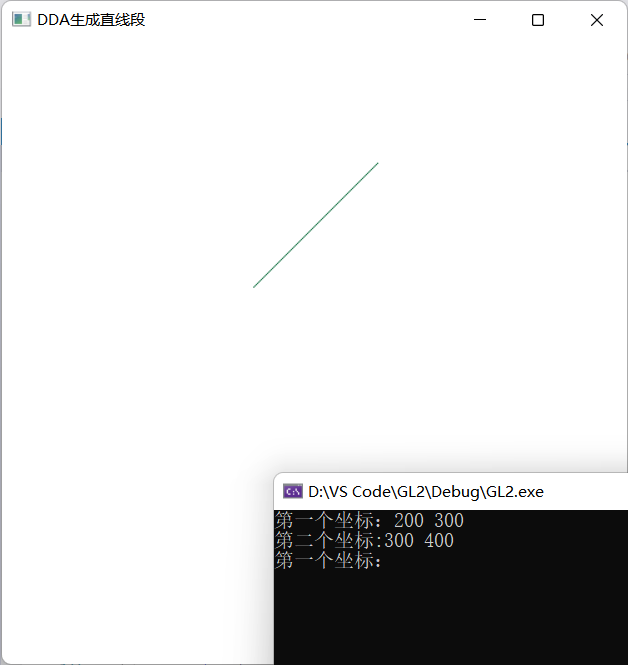
2、实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

|  |
| --- |
| //DDA直线生成算法  #include <stdlib.h>  #include <math.h>  #include<GL/glut.h>  #include <stdio.h>  inline int round(const float a) {  return int(a + 0.5);  }  void setPixel(GLint x, GLint y) {  glBegin(GL\_POINTS);  glVertex2i(x, y);  glEnd();  }  void lineDDA(int x0, int y0, int xEnd, int yEnd) {  int dx = xEnd - x0;  int dy = yEnd - y0;  int steps, k;  float xIncrement, yIncrement, x = x0, y = y0;  if (fabs(dx) > fabs(dy)) {  steps = fabs(dy);  }  else  steps = fabs(dx);  xIncrement = float(dx) / float(steps);  yIncrement = float(dy) / float(steps);  setPixel(round(x), round(y));  for (k = 0; k < steps; k++) {  x += xIncrement;  y += yIncrement;  setPixel(round(x), round(y));  }  }  //初始化方法  void Init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0);  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0.0, 500.0, 0.0, 500.0);  }  void lineSegment(void) {  int x1, y1, x2, y2;  //glViewport(0, 0, 500, 500);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.4, 0.2);  glBegin(GL\_LINES);  printf("第一个坐标：");  scanf("%d %d", &x1, &y1);  printf("第二个坐标:");  scanf("%d%d", &x2, &y2);  lineDDA(x1, y1, x2, y2);  glEnd();  glFlush();  }  void main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(0, 0);  glutInitWindowSize(500, 500);  glutCreateWindow("DDA生成直线段");  glutDisplayFunc(lineSegment);  Init();  glutMainLoop();  } |

4、实验结果：



实验三 BresenBam直线生成算法

时间：2022.3.30

地点：信息学院2202

1、实验内容：

熟悉OPENGL，通过Bresenham中点、改进Bresenham算法生成直线段

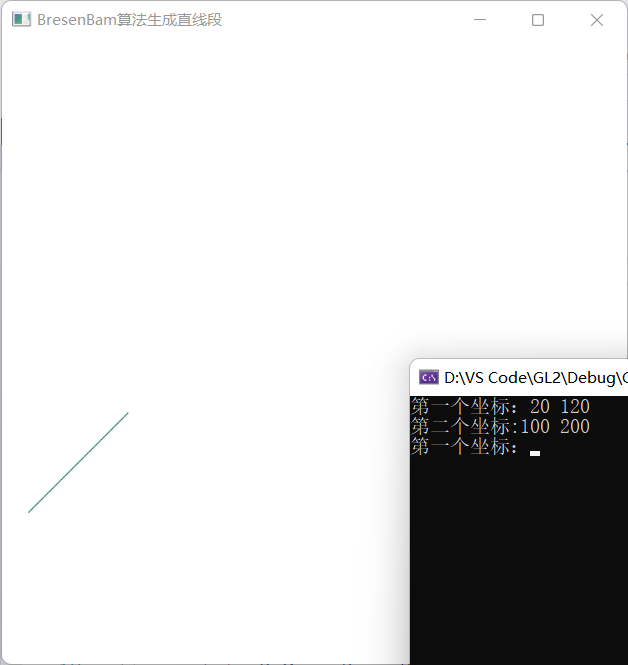
2、实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

|  |
| --- |
| //BresenBam算法生成直线段  #include <stdlib.h>  #include <math.h>  #include <GL/glut.h>  #include <stdio.h>  void setPixel(GLint x, GLint y) {  glBegin(GL\_POINTS);  glVertex2i(x, y);  glEnd();  }  void lineBres(int x0,int y0,int xEnd,int yEnd){  int dx = fabs(xEnd - x0);  int dy = fabs(yEnd - y0);  int p = 2 \* dy - dx;  int twoDy = 2 \* dy;  int twoDyMinusDx = 2 \* (dy - dx);  int x, y;  if (x0 > xEnd) {  x = xEnd;  y = yEnd;  xEnd = x0;  }  else {  x = x0;  y = y0;  }  setPixel(x, y);  while(x<xEnd){  x++;  if (p < 0) {  p = p + twoDy;  }  else {  y++;  p = p + twoDyMinusDx;  }  setPixel(x, y);  }  }  void Init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0);  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0.0, 500.0, 0.0, 500.0);  }  void lineSegment(void) {  int x1, y1, x2, y2;  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.4, 0.2);  glBegin(GL\_LINES);  printf("第一个坐标：");  scanf("%d%d", &x1, &y1);  printf("第二个坐标:");  scanf("%d%d", &x2, &y2);  lineBres(x1, y1, x2, y2);  glEnd();  glFlush();  }  void main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(0, 0);  glutInitWindowSize(500, 500);  glutCreateWindow("BresenBam算法生成直线段");  glutDisplayFunc(lineSegment);  Init();  glutMainLoop();  } |

4、实验结果：



实验四 描线填充算法填充多边形

时间：2022.4.6

地点：信息学院2202

1、实验内容：

熟悉OPENGL，用扫描线填充算法填充多边形。

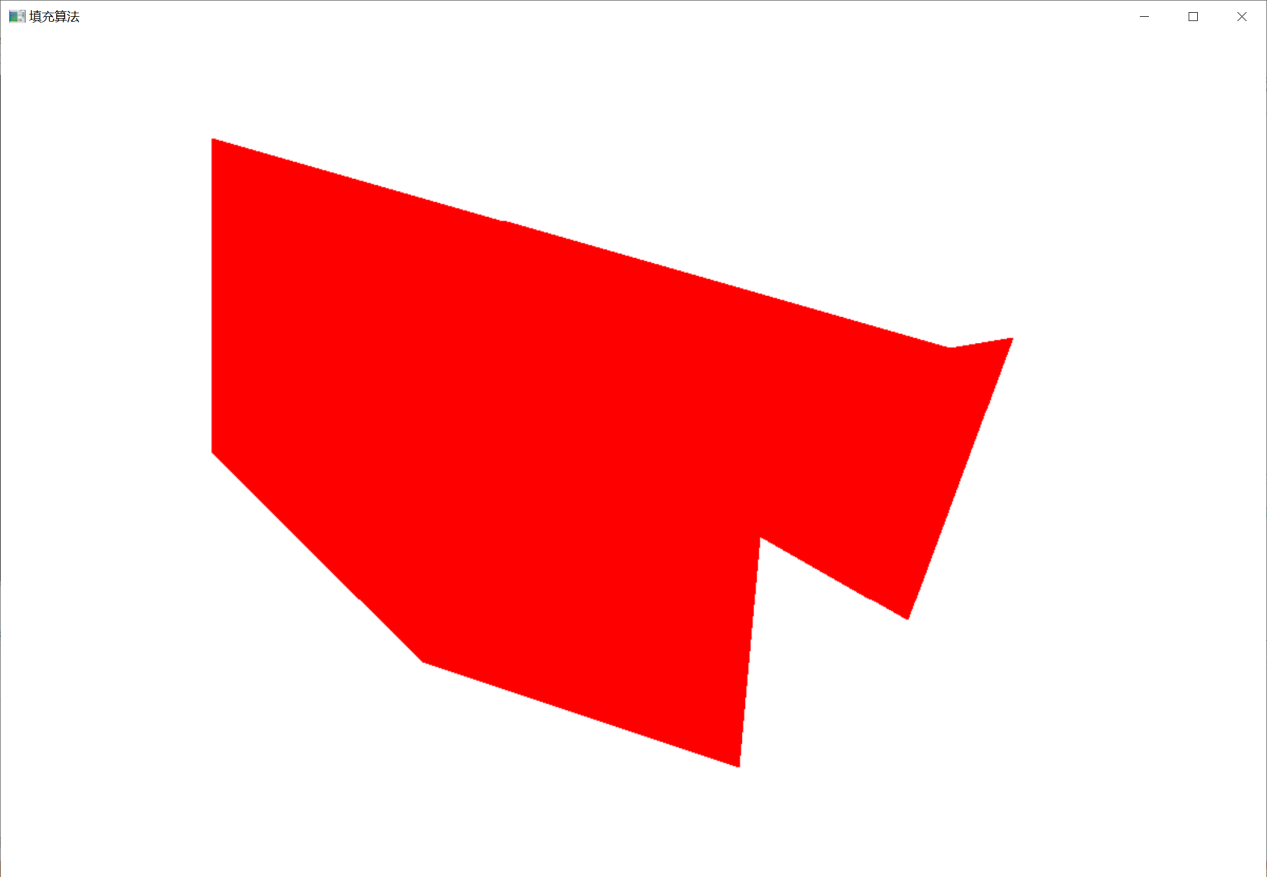
2、实验目的：

验证扫描线填充算法，指定任意的多边形边数，填充多边形。

3、实验代码：

|  |
| --- |
| #include <iostream>  #include <vector>  #include "GL/glut.h"  using namespace std;  //定义用于边表ET和活动边表AET的通用类Edge  class Edge  {  public:  int ymax;  float x;  float dx;  Edge\* next;  };  //定义用于表示像素点坐标的类Point  class Point  {  public:  int x;  int y;  Point(int x, int y)  {  this->x = x;  this->y = y;  }  };  /////////////////////请使用对应Demo/////////////////////  //窗体宽高  //Demo1  //const int windowWidth = 18;  //const int windowHeight = 12;  //Demo2  //const int windowWidth = 180;  //const int windowHeight = 120;  //Demo3、Demo4、Demo5  const int windowWidth = 1200;  const int windowHeight = 900;  //多边形顶点  //Demo1  //vector<Point> vertices = { Point(2, 5), Point(2, 10), Point(9, 6), Point(16, 11), Point(16, 4), Point(12, 2), Point(7, 2) };  //Demo2  vector<Point> vertices = { Point(200, 500), Point(200, 800), Point(900, 600), Point(960, 610), Point(860, 340), Point(720, 420), Point(700, 200),Point(400,300) };  //vector<Point> vertices = { Point(200, 500), Point(200, 800), Point(900, 600) };  //vector<Point> vertices = { Point(200, 500), Point(200, 800), Point(900, 600), Point(960, 610) };  //Demo3 多边形  //vector<Point> vertices = { Point(200, 500), Point(200, 1000), Point(900, 600), Point(1600, 1100), Point(1600, 400), Point(1200, 200), Point(700, 200) };  //Demo4 箭头  //vector<Point> vertices = { Point(395, 887), Point(479, 998), Point(1199, 433), Point(1101, 867), Point(1294, 715), Point(1417, 171), Point(857, 163), Point(668, 314), Point(1111, 321) };  //Demo5 闪电  //vector<Point> vertices = { Point(566, 970), Point(685, 1020), Point(754, 683), Point(985, 768), Point(1037, 481), Point(1208, 546), Point(1233, 179), Point(1140, 440), Point(951, 386), Point(899, 662), Point(668, 562) };  //边表  Edge\* ET[windowHeight];  //活动边表  Edge\* AET;  void init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0);  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);  }  void polygonScan()  {  //计算最高点的y坐标  int maxY = 0;  for (int i = 0; i < vertices.size(); i++)  {  if (vertices[i].y > maxY)  {  maxY = vertices[i].y;  }  }  //初始化ET和AET  Edge\* pET[windowHeight];  for (int i = 0; i < maxY; i++)  {  pET[i] = new Edge();  pET[i]->next = nullptr;  }  AET = new Edge();  AET->next = nullptr;  //清空显示窗口并设置画点颜色为红色  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(1.0, 0.0, 0.0);  glBegin(GL\_POINTS);  //建立边表ET  for (int i = 0; i < vertices.size(); i++)  {  //取出当前点1前后相邻的共4个点，点1与点2的连线作为本次循环处理的边，另外两个点点0和点3用于计算奇点  int x0 = vertices[(i - 1 + vertices.size()) % vertices.size()].x;  int x1 = vertices[i].x;  int x2 = vertices[(i + 1) % vertices.size()].x;  int x3 = vertices[(i + 2) % vertices.size()].x;  int y0 = vertices[(i - 1 + vertices.size()) % vertices.size()].y;  int y1 = vertices[i].y;  int y2 = vertices[(i + 1) % vertices.size()].y;  int y3 = vertices[(i + 2) % vertices.size()].y;  //水平线直接舍弃  if (y1 == y2)  continue;  //分别计算下端点y坐标、上端点y坐标、下端点x坐标和斜率倒数  int ymin = y1 > y2 ? y2 : y1;  int ymax = y1 > y2 ? y1 : y2;  float x = y1 > y2 ? x2 : x1;  float dx = (x1 - x2) \* 1.0f / (y1 - y2);  //奇点特殊处理，若点2->1->0的y坐标单调递减则y1为奇点，若点1->2->3的y坐标单调递减则y2为奇点  if (((y1 < y2) && (y1 > y0)) || ((y2 < y1) && (y2 > y3)))  {  ymin++;  x += dx;  }  //创建新边，插入边表ET  Edge\* p = new Edge();  p->ymax = ymax;  p->x = x;  p->dx = dx;  p->next = pET[ymin]->next;  pET[ymin]->next = p;  }  //扫描线从下往上扫描，y坐标每次加1  for (int i = 0; i < maxY; i++)  {  //取出ET中当前扫描行的所有边并按x的递增顺序（若x相等则按dx的递增顺序）插入AET  while (pET[i]->next)  {  //取出ET中当前扫描行表头位置的边  Edge\* pInsert = pET[i]->next;  Edge\* p = AET;  //在AET中搜索合适的插入位置  while (p->next)  {  if (pInsert->x > p->next->x)  {  p = p->next;  continue;  }  if (pInsert->x == p->next->x && pInsert->dx > p->next->dx)  {  p = p->next;  continue;  }  //找到位置  break;  }  //将pInsert从ET中删除，并插入AET的当前位置  pET[i]->next = pInsert->next;  pInsert->next = p->next;  p->next = pInsert;  }  //AET中的边两两配对并填色  Edge\* p = AET;  while (p->next && p->next->next)  {  for (int x = p->next->x; x < p->next->next->x; x++)  {  glVertex2i(x, i);  }  p = p->next->next;  }  //删除AET中满足y=ymax的边  p = AET;  while (p->next)  {  if (p->next->ymax == i)  {  Edge\* pDelete = p->next;  p->next = pDelete->next;  pDelete->next = nullptr;  delete pDelete;  }  else  {  p = p->next;  }  }  //更新AET中边的x值，进入下一循环  p = AET;  while (p->next)  {  p->next->x += p->next->dx;  p = p->next;  }  }  glEnd();  glFlush();  }  int main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(50, 100);  glutInitWindowSize(windowWidth, windowHeight);  glutCreateWindow("填充算法");  init();  glutDisplayFunc(polygonScan);  glutMainLoop();  return 0;  } |

4、实验结果：



实验五 填充算法实验

时间：2022.4.13

地点：信息学院2202

1、实验内容：

圆的扫描转换，输入圆的半径，画出圆，

种子填充算法，输入多边形，种子位置，填充多边形。

2、实验目的：

（1）输入圆的半径，画出圆。

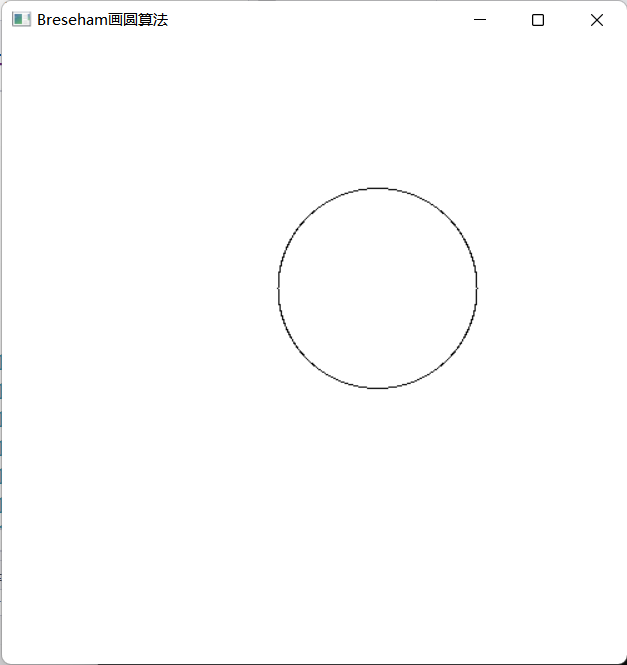
（2）输入多边形，种子点位置，填充多边形。

1. 实验代码：

3.1画圆算法：

|  |
| --- |
| //Bresenham算法画圆  #include<windows.h>  #include<GL/glut.h>  #include<stdio.h>  void init(int argc, char\*\* argv)  {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);      glutInitWindowPosition(50, 100);      glutInitWindowSize(400, 400);      glutCreateWindow("Bresenham算法画圆");      glClearColor(1.0, 1.0, 1.0, 0);      glMatrixMode(GL\_PROJECTION);      gluOrtho2D(0, 400, 0, 400);  }  void Bresenham\_Circle(int xc, int yc, int r)  {      int x, y, d;      x = 0;      y = r;      d = 3 - 2 \* r;      glVertex2i(x + xc, y + yc);      while (x < y)      {          if (d < 0)          {              d = d + 4 \* x + 6;          }          else          {              d = d + 4 \* (x - y) + 10;              y--;          }          x++;          glVertex2i(x + xc, y + yc);          glVertex2i(y + xc, x + yc);          glVertex2i(y + xc, -x + yc);          glVertex2i(x + xc, -y + yc);          glVertex2i(-x + xc, -y + yc);          glVertex2i(-y + xc, -x + yc);          glVertex2i(-x + xc, y + yc);          glVertex2i(-y + xc, x + yc);      }  }  void myDisplay(void)  {      glClear(GL\_COLOR\_BUFFER\_BIT);      glColor3f(0.0, 0.4, 0.2);      glPointSize(2);      glBegin(GL\_POINTS);      int r;      printf("请输入圆的半径：");      scanf("%d", &r);      Bresenham\_Circle(200, 200, r);      glEnd();      glFlush();  }  int main(int argc, char\*\* argv)  {      init(argc, argv);      glutDisplayFunc(myDisplay);      glutMainLoop();      return 0;  } |

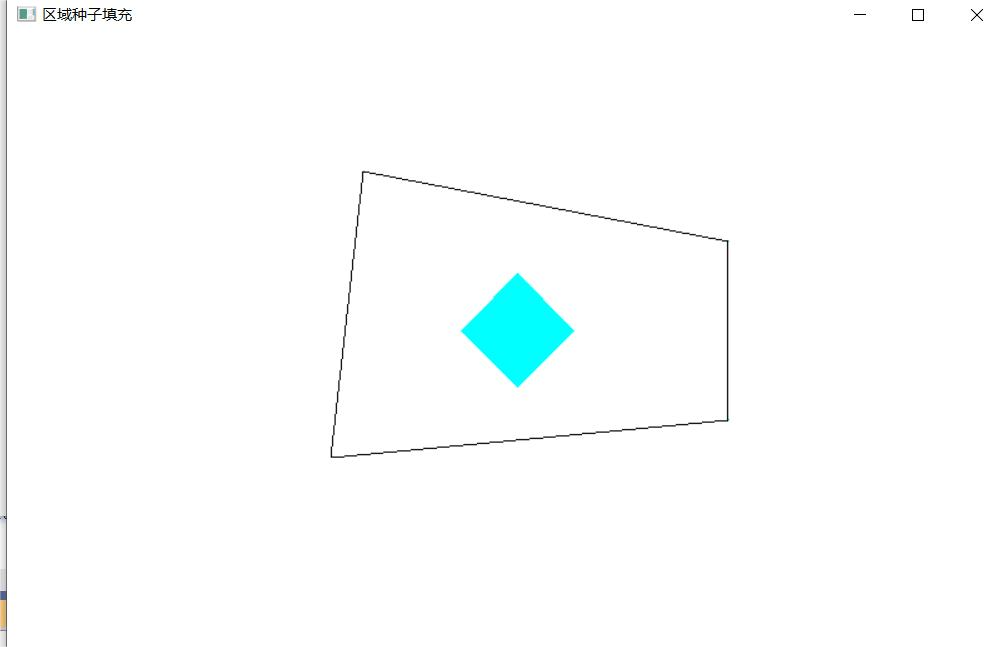
实验结果：



3.2种子填充

|  |
| --- |
| //系统递归栈太慢了，用stack或者queue会好一点，本质一样，都是bfs  #include<iostream>  #include<vector>  #include<queue>  #include<stack>  #include<GL/glut.h>  using namespace std;  int window\_width = 800, window\_height = 600;  struct point  {  int x, y;  point() {}  point(int xx, int yy) :  x(xx), y(yy) {}  };  vector<point> vertice; //顶点  float newcolor[3] = { 0, 1, 1 }; //填充颜色  float boundarycolor[3] = { 0, 0, 0 }; //边界颜色  void draw\_a\_point(int x, int y);  bool is\_equal(float\* a, float\* b);  void BoundaryFill4(int x, int y);  void BoundaryFill4\_Stack(int x, int y);  void mymouse(int button, int state, int x, int y);  void KeyBoards(unsigned char key, int x, int y);  //画点函数  void draw\_a\_point(int x, int y)  {  glBegin(GL\_POINTS);  glColor3fv(newcolor);  glVertex2f(x, y);  glEnd();  glFlush();  }  //判断函数  bool is\_equal(float\* a, float\* b)  {  return a[0] == b[0] && a[1] == b[1] && a[2] == b[2];  }  //4连通区域的递归填充算法  /\*void BoundaryFill4(int x, int y)  {  float color[3];  glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, color);  cout << x << "," << y << ":" << color[0] << color[1] << color[2] << endl;  if (!is\_equal(color, newcolor) && !is\_equal(color, boundarycolor))  {  draw\_a\_point(x, y);  BoundaryFill4(x, y + 1);  BoundaryFill4(x, y - 1);  BoundaryFill4(x + 1, y);  BoundaryFill4(x - 1, y);  }  \*/  //用队列queue实现的 4连通区域的填充算法  void BoundaryFill4(int x, int y)  {  queue<point> q;  q.push(point(x, y));  while (!q.empty())  {  point now = q.front();  q.pop();  int nowx = now.x, nowy = now.y;  float color[3];  glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color); //读取像素颜色  if (!is\_equal(color, newcolor) && !is\_equal(color, boundarycolor))  {  draw\_a\_point(nowx, nowy);  q.push(point(nowx, nowy + 1)); //下方  q.push(point(nowx, nowy - 1));  q.push(point(nowx + 1, nowy));  q.push(point(nowx - 1, nowy)); //左方  }  }  }  //用栈实现的 扫描线形式的填充算法  void BoundaryFill4\_Stack(int x, int y)  {  stack <point> q;  q.push(point(x, y));  while (!q.empty())  {  point now = q.top();  q.pop();  int nowx = now.x, nowy = now.y;  float color[3];  glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);  if (!is\_equal(color, newcolor) && !is\_equal(color, boundarycolor))  {  draw\_a\_point(nowx, nowy);  q.push(point(nowx, nowy + 1));  q.push(point(nowx, nowy - 1));  q.push(point(nowx + 1, nowy));  q.push(point(nowx - 1, nowy));  }  }  }  void mymouse(int button, int state, int x, int y)  {  if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)  {  draw\_a\_point(x, window\_height - y);  point p(x, window\_height - y);  vertice.push\_back(p);  cout << "顶点" << vertice.size() << ": (" << x << ", " << y << ")" << endl;  }  if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)  {  glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色  glColor3fv(boundarycolor);  glBegin(GL\_LINES);  for (int i = 0; i < vertice.size(); i++)  {  if (i == vertice.size() - 1)//画完最后一个点，使其闭合  {  glVertex2f(vertice[0].x, vertice[0].y);  glVertex2f(vertice[i].x, vertice[i].y);  }  else  {  glVertex2f(vertice[i].x, vertice[i].y);  glVertex2f(vertice[i + 1].x, vertice[i + 1].y);  }  }  glEnd();  glFlush();  vertice.clear();  }  if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)  {  cout << "center: (" << x << ", " << y << ")" << endl;  BoundaryFill4(x, window\_height - y);  //BoundaryFill4\_Stack(x, window\_height - y);  }  }  void KeyBoards(unsigned char key, int x, int y)  {  if (key == 32)  {  BoundaryFill4(x, window\_height - y);  glFlush();  }  }  void display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.4, 0.2);  glPointSize(1);  glBegin(GL\_POINTS);  //BoundaryFill4(m, window\_height - n);  glEnd();  glFlush();  }  int main(int argc, char\* argv[])  {  cout << "点击鼠标左键画点；" << endl << "点击鼠标右键结束画点，形成多边形；" << endl << "点击鼠标中键确定区域填充种子点位置。" << endl;  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(400, 150);  glutInitWindowSize(window\_width, window\_height);  glutCreateWindow("区域种子填充");  glMatrixMode(GL\_PROJECTION);/\*设置为投影类型模式和其他观察参数\*/  //glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();/\*设置为投影类型模式和其他观察参数\*/  //gluOrtho2D(0, window\_width, window\_height, 0);/\*设置为投影类型模式和其他观察参数，观察窗口的大小要与画布大小一致，所以直接设置全局变量即可\*/  gluOrtho2D(0, window\_width, 0, window\_height); //上面的不可以！！！  glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色  glClear(GL\_COLOR\_BUFFER\_BIT);  glutDisplayFunc(&display);//自己加的，回调函数  glutMouseFunc(&mymouse);  //glutKeyboardFunc(&KeyBoards);  glutMainLoop();  return 0;  } |

1. 实验结果：



实验六 填充算法实验

时间：2022.4.13

地点：信息学院2202

1、实验内容：

教材P161，二维几何变换算法（平移、比例、旋转、对称）

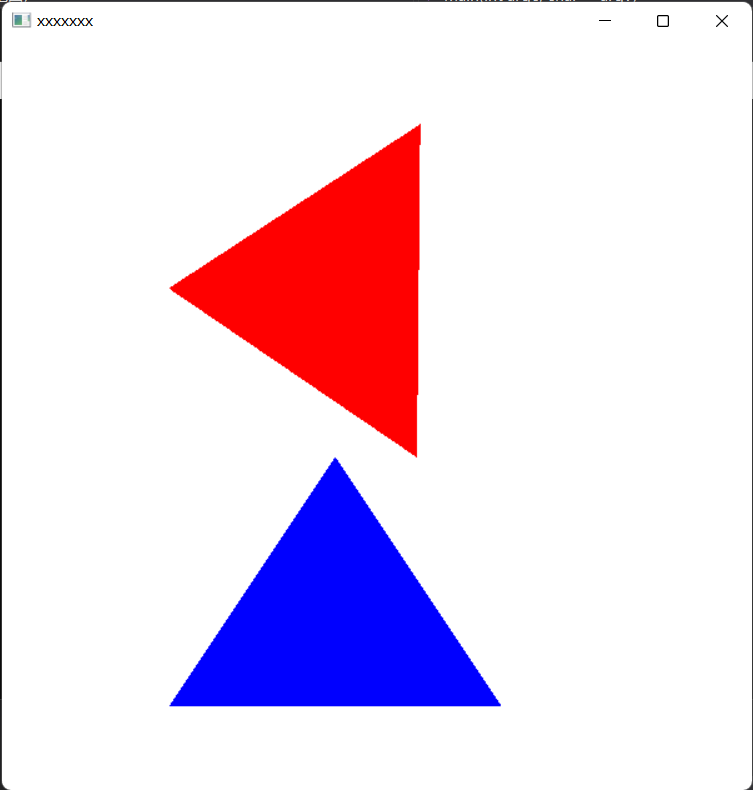
2、实验目的：

验证二维几何变换，熟悉变换矩阵。

3、实验代码：

|  |
| --- |
| //教材P161，二维几何变换算法（平移、比例、旋转、对称）  #include<GL/glut.h>  #include<stdlib.h>  #include<math.h>  /\*Set initial display-window size\*/  GLsizei winWidth= 600, winHeight = 600;  GLfloat xwcMin = 0.0, xwcMax = 225.0;  GLfloat ywcMin = 0.0, ywcMax = 225.0;  class wcPt2D{  public:  GLfloat x, y;  };  typedef GLfloat Matrix3x3[3][3];  Matrix3x3 matComposite;  const GLdouble pi=3.14159;  void init(void) {  glClearColor(1.0, 1.0, 1.0, 1.0);  }  void matrix3x3SetIdentity(Matrix3x3 matIdent3x3 ){  GLint row, col;  for (row = 0; row < 3; row++) {  for (col = 0; col < 3; col++) {  matIdent3x3[row][col] = (row == col);  }  }  }  void matrix3x3PreMultiply(Matrix3x3 m1,Matrix3x3 m2){  GLint row, col;  Matrix3x3 matTemp;  for (row = 0; row < 3; row++) {  for (col = 0; col < 3; col++) {  matTemp[row][col] = m1[row][0] \* m2[0][col] + m1[row][1] \* m2[1][col] + m1[row][2] \* m2[2][col];  }  }  for (row = 0; row < 3; row++) {  for (col = 0; col < 3; col++) {  m2[row][col] = matTemp[row][col];  }  }  }  void translate2D(GLfloat tx, GLfloat ty) {  Matrix3x3 matTransl;  matrix3x3SetIdentity(matTransl);  matTransl[0][2] = tx;  matTransl[1][2] = ty;  matrix3x3PreMultiply(matTransl, matComposite);  }  void roate2D(wcPt2D pivoPt, GLfloat theta) {  Matrix3x3 matRot;  matrix3x3SetIdentity(matRot);  matRot[0][0] = cos(theta);  matRot[0][1] = -sin(theta);  matRot[0][2] = pivoPt.x \* (1 - cos(theta)) + pivoPt.y\* sin(theta);  matRot[1][0] = sin(theta);  matRot[1][1] = cos(theta);  matRot[1][2] = pivoPt.y \* (1 - cos(theta)) - pivoPt.x \* sin(theta);  matrix3x3PreMultiply(matRot, matComposite);  }  void scale2D(GLfloat sx, GLfloat sy, wcPt2D fixedPt) {  Matrix3x3 matScale;  matrix3x3SetIdentity(matScale);  matScale[0][0] = sx;  matScale[0][2] = (1 - sx) \* fixedPt.x;  matScale[1][1] = sy;  matScale[1][2] = (1 - sy) \* fixedPt.y;  matrix3x3PreMultiply(matScale, matComposite);  }  void transfromVers2D(GLint nVerts, wcPt2D\* verts) {  GLint k;  GLfloat temp;  for (k = 0; k < nVerts; k++) {  temp = matComposite[0][0] \* verts[k].x + matComposite[0][1] \* verts[k].y + matComposite[0][2];  verts[k].y = matComposite[1][0] \* verts[k].x + matComposite[1][1] \* verts[k].y + matComposite[1][2];  verts[k].x = temp;  }  }  void triangle(wcPt2D\* verts) {  GLint k;  glBegin(GL\_TRIANGLES);  for (k = 0; k < 3; k++) {  glVertex2f(verts[k].x, verts[k].y);  }  glEnd();  }  void displyFcn(void) {  GLint nVerts = 3;  wcPt2D verts[3] = { {50.0,25.0},{150.0,25.0},{100.0,100.0} };  wcPt2D centroidPt;  GLint k, xSum = 0, ySum = 0;  for (k = 0; k < nVerts; k++) {  xSum += verts[k].x;  ySum += verts[k].y;  }  centroidPt.x = GLfloat(xSum) / GLfloat(nVerts);  centroidPt.y = GLfloat(ySum) / GLfloat(nVerts);  wcPt2D pivPt, fixedPt;  pivPt = centroidPt;  fixedPt = centroidPt;  GLfloat tx = 0.0, ty = 100.0;  GLfloat sx = 0.5, sy = 0.5;  GLdouble theta = pi / 2.0;  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.0, 1.0);  triangle(verts);  matrix3x3SetIdentity(matComposite);  scale2D(sx, sy, fixedPt);  roate2D(pivPt, theta);  translate2D(tx, ty);  transfromVers2D(nVerts, verts);  glColor3f(1.0, 0.0, 0.0);  triangle(verts);  glFlush();  }  void winReshapeFcn(GLint newWidth, GLint newHeight) {  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);  glClear(GL\_COLOR\_BUFFER\_BIT);  }  void main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(50, 50);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("xxxxxxxx");  init();  glutDisplayFunc(displyFcn);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

1. 实验结果：



实验七 GLUT鼠标函数实验、反走样技术

时间：2022.4.26

地点：信息学院2202

1、实验内容：

教材P458，GLUT鼠标函数，使用opengl，实现任一反走样技术。

2、实验目的：

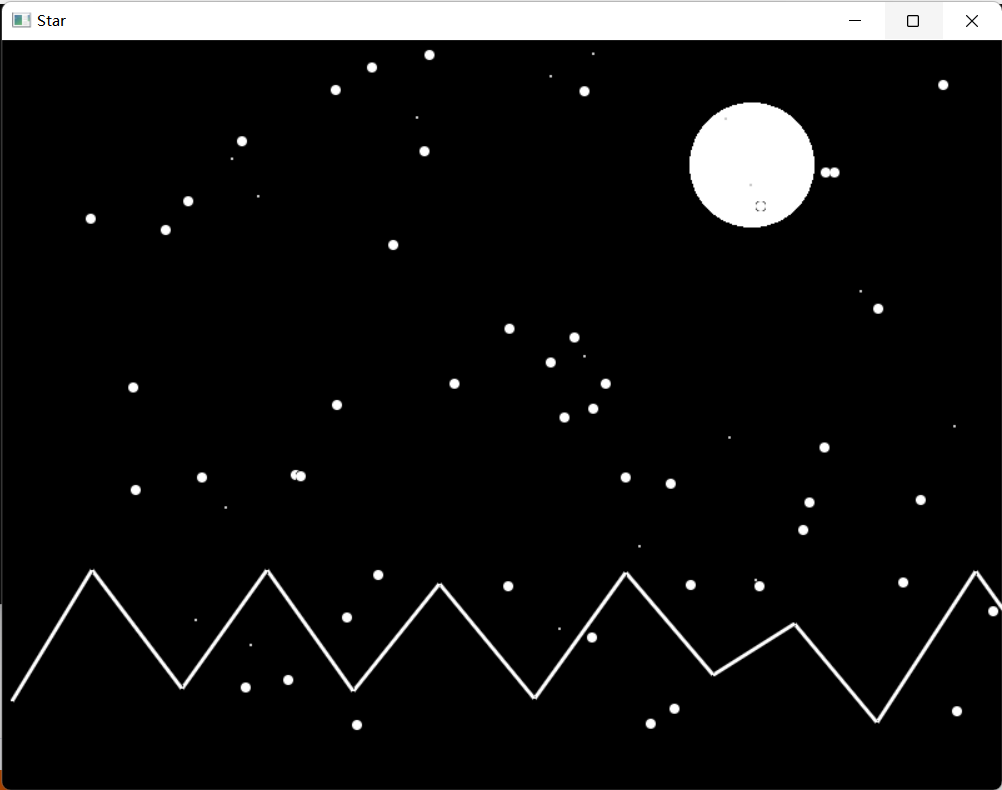
调用鼠标函数完成相应功能，2-3个程序。

3、实验代码：

3.1实现任一反走样技术

|  |
| --- |
| //实现任一反走样技术  #include<windows.h>  #include<gl/glut.h>  #include<math.h>  #include<iostream>  using namespace std;  #define x\_Screen 800  #define y\_Screen    600  #define little 50  #define middle 20  #define large   8  void myBackground()  {      glClearColor(0.0, 0.0, 0.0, 1.0);      glColor3f(1.0, 1.0, 1.0);  }  void myDisplay()  {      glEnable(GL\_DEPTH\_TEST);      //如果没有抗锯齿，则点为方形的。如果我们启动抗锯齿设置，则点是一个圆点。      glEnable(GL\_POINT\_SMOOTH);      glEnable(GL\_LINE\_SMOOTH);      glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST); // Make round points, not square points      glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);  // Antialias the lines      glEnable(GL\_BLEND);      glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);      glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);      int i;      glBegin(GL\_POINTS);      for (i = 0; i < little; i++)          glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);      glEnd();      glPointSize(2);      glBegin(GL\_POINTS);      for (i = 0; i < middle; i++)          glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);      glEnd();      glPointSize(8);      glBegin(GL\_POINTS);      for (i = 0; i < large; i++)          glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);      glEnd();      glBegin(GL\_POLYGON);      for (i = 0; i < 64; i++)          glVertex2f(600 + 50.0 \* cos((float)i / 10), 500 + 50.0 \* sin((float)i / 10));      glEnd();      glLineWidth(3);      glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);      glEnable(GL\_BLEND);      glEnable(GL\_LINE\_SMOOTH);      glBegin(GL\_LINE\_STRIP);      for (i = 0; i < 19; i++) {          glVertex2f(rand() % 10 + i \* 70, rand() % 50 + 50.0 + (i % 2) \* 80);      }      glEnd();      glutSwapBuffers();  }  void myChange(int w, int h)  {      glViewport(0, 0, w, h);      glMatrixMode(GL\_PROJECTION);      glLoadIdentity();      gluOrtho2D(0.0, x\_Screen, 0.0, y\_Screen);      glMatrixMode(GL\_MODELVIEW);      glLoadIdentity();  }  void main(int argc, char\*\* argv)  {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);      glutInitWindowSize(x\_Screen, y\_Screen);      glutCreateWindow("StarStar");      glutDisplayFunc(myDisplay);      glutReshapeFunc(myChange);      myBackground();      glutMainLoop();  } |

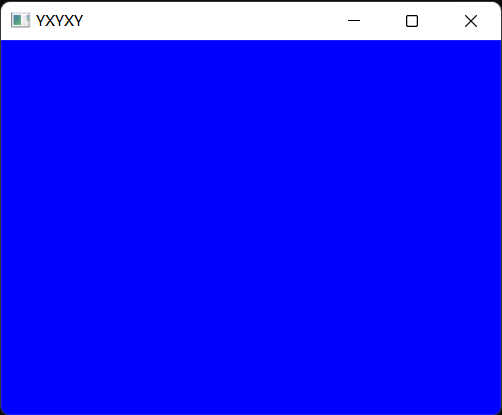
实验结果：



3.2教材P458的算法，GLUT鼠标函数

|  |
| --- |
| //教材P458的算法，GLUT鼠标函数  #include<GL/glut.h>  GLsizei winWidth = 400, winHeight = 300;  void init(void) {  glClearColor(0.0, 0.0, 1.0, 1.0);  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0.0, 200.0, 0.0, 150.0);  }  void displayFcn(void) {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(1.0, 0.0, 0.0);  glPointSize(3.0);  }  void winReshapeFcn(GLint newWidth, GLint newHeight) {  glViewport(0, 0, newWidth, newHeight);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0.0, GLdouble(newWidth), 0.0, GLdouble(newHeight));  winWidth = newWidth;  winHeight = newHeight;  }  void plotPoint(GLint x, GLint y) {  glBegin(GL\_POINTS);  glVertex2i(x, y);  glEnd();  }  void mousePtPlot(GLint button, GLint action, GLint xMouse, GLint yMouse) {  if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {  plotPoint(xMouse, winHeight - yMouse);  }  glFlush();  }  void main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(100, 100);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("YXYXY");  init();  glutDisplayFunc(displayFcn);  glutReshapeFunc(winReshapeFcn);  glutMouseFunc(mousePtPlot);  glutMainLoop();  } |

实验结果：



实验八 二维图像裁剪实验

时间：2022.5.4

地点：信息学院2202

1、实验内容：

教材P458，GLUT鼠标函数

实验内容1：使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪

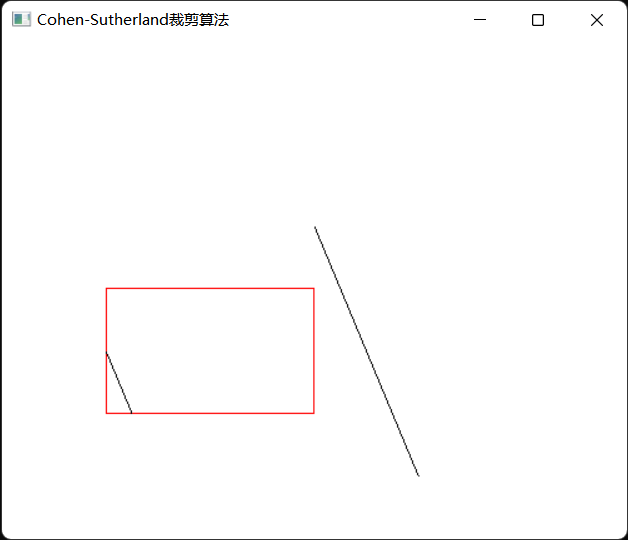
2、实验目的：

调用鼠标函数完成相应功能，3个程序。验证Cohen-Sutherland裁剪算法，从键盘输入任意的直线段，用指定的裁剪窗口裁剪直线段。

3、实验代码：

|  |
| --- |
| //Cohen-Sutherland线段裁剪算法对直线段进行裁剪。  #include <GL/glut.h>  #include <cstdio>  #define LEFT 1//0001  #define RIGHT 2//0010  #define BOTTOM 4//0100  #define TOP 8//1000  int x1 = 150, y1 = 50, x2 = 50, y2 = 250, XL = 100, XR = 300, YB = 100, YT = 200;  //(x1,y1)、(x2,y2)为直线段的端点，XL为左边界，XR为右边界，YB为下边界，YT为上边界  int x1\_init = 150, y1\_init = 50, x2\_init = 50, y2\_init = 250;  //将直线段端点备份，以便画出裁剪前的直线段  void encode(int x, int y, int& c)  {//|或运算两者有一个为1结果就为1      c = 0;      if (x < XL) c |= LEFT;//按照数字进行或运算，左边界      else if (x > XR) c |= RIGHT;//右边界      if (y < YB) c |= BOTTOM;//下边界      else if (y > YT) c |= TOP;//上边界  }  //上左:1001; 中左:0001; 下左:0101;  //上中:1000; 中中:0000 下中:0100  //上右:1010  中右:0010  下右:0110  void CS\_LineClip()  //Cohen-Sutherland裁剪算法  {      int x, y;      int code1, code2, code;      encode(x1, y1, code1);//先求出端点所在的区号      encode(x2, y2, code2);//先求出端点所在的区号      //循环处理      while (code1 != 0 || code2 != 0)//只有两个端点同时都在矩形区域里才不使用函数      {          if (code1 & code2)//两个都为1才成立，这两个都在同一个区域中 直接不进行处理              return;          if (code1 != 0)//如果              code = code1;          else              code = code2;          if (LEFT & code)//线段与左边界相交(如果          {              x = XL;//求出的左边界的交点的x值              y = y1 + (y2 - y1) \* (XL - x1) / (x2 - x1);//求出左边交点的y值          }          else if (RIGHT & code)//线段与右边界相交          {              x = XR;//求出右边界的交点的x值              y = y1 + (y2 - y1) \* (XR - x1) / (x2 - x1);//根据斜率计算出右边界的交点的y值          }          else if (BOTTOM & code)//线段与下边界相交          {              y = YB;//求出下边界的交点的y值              x = x1 + (x2 - x1) \* (YB - y1) / (y2 - y1);//求出下边界的交点的x值根据斜率          }          else if (TOP & code)//线段与上边界相交          {              y = YT;              x = x1 + (x2 - x1) \* (YT - y1) / (y2 - y1);          }          if (code == code1)          {              x1 = x; y1 = y;  encode(x1, y1, code1);//将区域 外的端点覆盖掉          }          else          {              x2 = x; y2 = y; encode(x2, y2, code2);//将区域 外的端点覆盖掉          }      }  }  void mydisplay()  //显示函数  {      //绘制方形边界      glClear(GL\_COLOR\_BUFFER\_BIT);      glColor3f(1.0, 0.0, 0.0);//设置背景颜色      glPointSize(2);//设置点的大小      glBegin(GL\_LINE\_LOOP);//绘制区域的操作      glVertex2i(XL, YT);//绘制矩形      glVertex2i(XL, YB);      glVertex2i(XR, YB);      glVertex2i(XR, YT);      glEnd();      glFlush();      //绘制未裁剪前的线段      glBegin(GL\_LINES);//绘制直线的操作      glVertex2i(x1\_init, y1\_init);      glVertex2i(x2\_init, y2\_init);      glEnd();      glFlush();      //绘制裁剪后的线段      glColor3f(0.0, 0.0, 0.0);      glBegin(GL\_LINES);      glVertex2i(x1, y1);      glVertex2i(x2, y2);      glEnd();      glFlush();      //绘制剪裁区域外的直线段      x1\_init = 400, y1\_init = 50, x2\_init = 300, y2\_init = 250;      glColor3f(0.0, 0.0, 0.0);      glBegin(GL\_LINES);      glVertex2i(x1\_init, y1\_init);      glVertex2i(x2\_init, y2\_init);      glEnd();      glFlush();  }  int main(int argc, char\* argv[])  {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);      glutInitWindowPosition(500, 100);//窗口位置      glutInitWindowSize(500, 400);      glutCreateWindow("Cohen-Sutherland裁剪算法");      glClearColor(1.0, 1.0, 1.0, 0.0);  //设置背景颜色      glMatrixMode(GL\_PROJECTION);       // 设置投影参数      gluOrtho2D(0.0, 600.0, 0.0, 400.0); // 设置场景的大小      CS\_LineClip();  //执行一次裁剪算法      glutDisplayFunc(&mydisplay);//调用绘制函数      glutMainLoop();      return 0;  } |

实验结果：



实验九 三维图形几何变换实验

时间：2022.5.11

地点：信息学院2202

1、实验内容：

实验内容1：使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪，P204

实验内容2：教材P222，三维图形旋转、缩放变换、平移变换、错切变换、对称变换等任意变换。

2、实验目的：

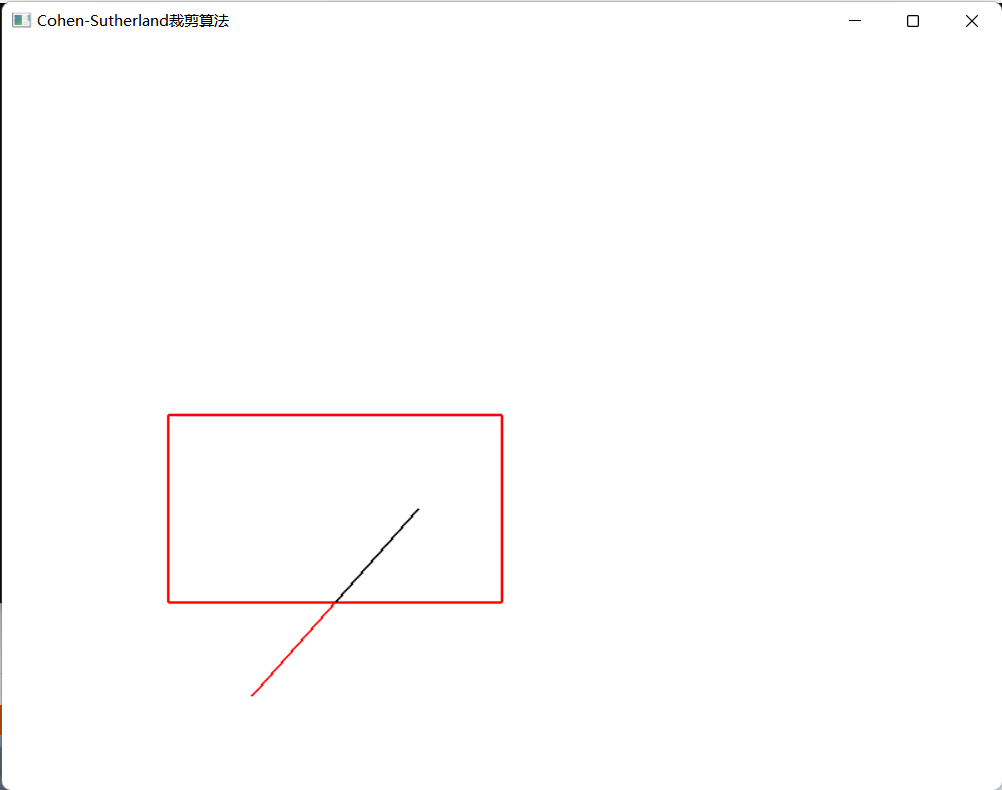
验证Cohen-Sutherland裁剪算法，从键盘输入任意的直线段，用指定的裁剪窗口裁剪直线段。调用函数完成三维图形几何变换。

3、实验代码：

3.1

|  |
| --- |
| //用Cohen-Sutherland线段裁剪算法对直线段进行裁剪，P204  #include <GL/glut.h> // OPenGL实用工具库  #define LEFT 1  #define RIGHT 2  #define BOTTOM 4  #define TOP 8  int x1 = 150, y1 = 50, x2 = 250, y2 = 150, XL = 100, XR = 300, YB = 100, YT = 200; //(x1,y1)、(x2,y2)为直线段的端点，XL为左边界，XR为右边界，YB为下边界，YT为上边界  int x1\_init = 150, y1\_init = 50, x2\_init = 250, y2\_init = 150; //将直线段端点备份，以便画出裁剪前的直线段  char encode(float x, float y)  {  char c = 0;  if (x < XL) c |= LEFT;  if (x > XR) c |= RIGHT;  if (y < YB) c |= BOTTOM;  if (y > YT) c |= TOP;  return c;  }  void CS\_LineClip(int& x1, int& y1, int& x2, int& y2, int XL, int XR, int YB, int YT) {  char code1 = encode(x1, y1);  char code2 = encode(x2, y2);  char code;  int x, y;  while (code1 || code2) {  if ((code1 & code2) != 0)return; //在外同侧  if (code1 != 0) code = code1;  else code = code2;  if ((LEFT & code) != 0) {  x = XL; y = y1 + (y2 - y1) \* (XL - x1) / (x2 - x1);  }  else if ((RIGHT & code) != 0) {  x = XR; y = y1 + (y2 - y1) \* (XR - x1) / (x2 - x1);  }  else if ((BOTTOM & code) != 0) {  y = YB; x = x1 + (x2 - x1) \* (YB - y1) / (y2 - y1);  }  else if ((TOP & code) != 0) {  y = YT; x = x1 + (x2 - x1) \* (YT - y1) / (y2 - y1);  }  if (code == code1) {  x1 = x; y1 = y; code1 = encode(x, y);  }  else { x2 = x; y2 = y; code2 = encode(x, y); }  }  }  void init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0); // 设置背景颜色  glMatrixMode(GL\_PROJECTION); // 设置投影参数  gluOrtho2D(0.0, 600.0, 0.0, 400.0); // 设置场景的大小  }  void draw(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(1.0, 0.0, 0.0); // 设置画图颜色  glLineWidth(2); // 设置边框宽度  glPointSize(2);  glPushMatrix();  //glTranslatef(dx, dy, 0); //图形变换操作须放在绘制之前  glBegin(GL\_LINE\_LOOP);  glVertex2i(XL, YT);  glVertex2i(XL, YB);  glVertex2i(XR, YB);  glVertex2i(XR, YT);  glEnd();  //绘制未裁剪前的线段  glBegin(GL\_LINES);  glVertex2i(x1\_init, y1\_init);  glVertex2i(x2\_init, y2\_init);  glEnd();  //绘制裁剪后的线段  glColor3f(0.0, 0.0, 0.0);  glBegin(GL\_LINES);  glVertex2i(x1, y1);  glVertex2i(x2, y2);  glEnd();  CS\_LineClip(x1, y1, x2, y2, XL, XR, YB, YT);  glPopMatrix();  glFlush(); // 处理绘图pipeLine  //glutSwapBuffers(); //GLUT\_DOUBLE  }  void main(int argc, char\*\* argv)  {  glutInit(&argc, argv); // 初始化GLUT环境  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); // GLUT显示模式：单缓冲区、RGB颜色模型  //glutInitWindowPosition(50, 100); // 窗口左上角的位置  glutInitWindowSize(800, 600); // 显示窗口的大小  glutCreateWindow("Cohen-Sutherland裁剪算法"); // 创建显示窗口，加上标题  init();  glutDisplayFunc(draw); // 调用绘图函数  glutMainLoop(); // 进入事件处理循环  } |

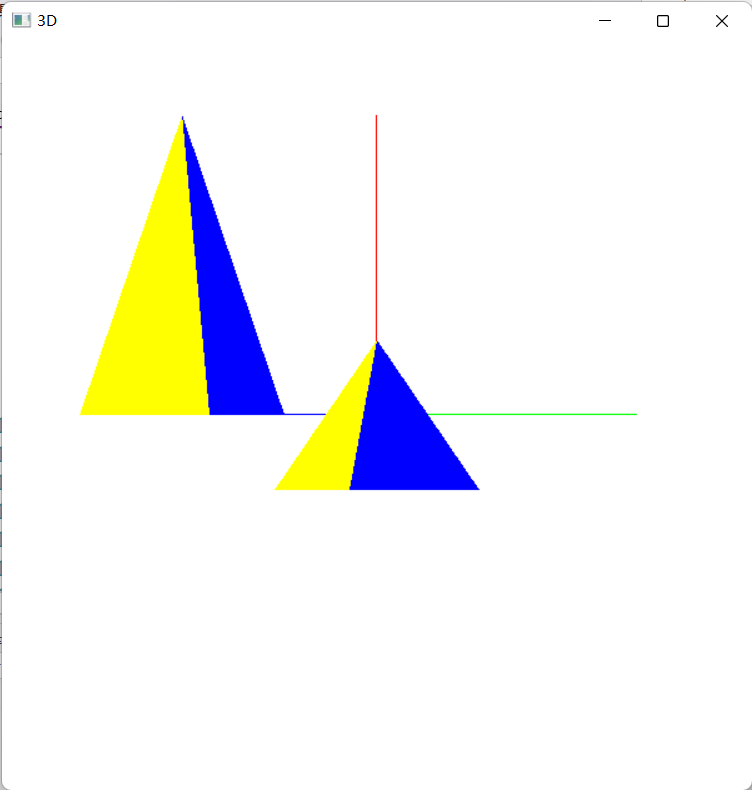
实验结果：



3.2

|  |
| --- |
| //教材P222，三维图形旋转、缩放变换、平移变换、错切变换、对称变换等任意变换。  #include <GL/glut.h>  #include<stdlib.h>  #include<math.h>  GLsizei winWidth = 600, winHeight = 600;  GLint nVerts = 12;//点的个数  GLfloat tx = 0, ty = 1, tz = 3;//存储要平移的距离  GLfloat angle = 60;//旋转角度  GLfloat scale = 2.0;//缩放比例  typedef GLfloat M4[4][4];  M4 matComposite;  class pt3D {  public:  GLfloat x, y, z;  };  pt3D verts[] = {  { 0.0, 1.0, 0.0 }, { -1.0, -1.0, 1.0 }, { 1.0, -1.0, 1.0 },  { 0.0, 1.0, 0.0 }, { 1.0, -1.0, 1.0 }, { 1.0, -1.0, -1.0 },  { 0.0, 1.0, 0.0 }, { 1.0, -1.0, -1.0 }, { -1.0, -1.0, -1.0 },  { 0.0, 1.0, 0.0 }, { -1.0, -1.0, -1.0 }, { -1.0, -1.0, 1.0 }  };//存放三维物体的各个点坐标，由于有4个三角形面，所以有12个点，设置顶点一致就好，拼接三角形，底面不做绘制  pt3D resultVerts[12];//存放变换后的矩阵，即最新的点坐标  class color {  public:  GLfloat r, g, b;  };  color colors[] = {  { 1.0, 0.0, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.0, 0.0, 1.0 }, { 1.0, 1.0, 0.0 }  };//存放每个面的颜色  void init()  {  glClearColor(1.0, 1.0, 1.0, 0.0);  glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0);  glMatrixMode(GL\_PROJECTION);  }  void m4SetIdentity(M4 matIdentity4x4)  {  GLint col, row;  for (row = 0; row < 4; row++) {  for (col = 0; col < 4; col++) {  matIdentity4x4[row][col] = (row == col);  }  }  }  void m4PreMultiply(M4 m1, M4 m2)  {  GLint row, col;  M4 matTemp;  for (row = 0; row < 4; row++) {  for (col = 0; col < 4; col++) {  matTemp[row][col] = m1[row][0] \* m2[0][col] + m1[row][1] \* m2[1][col] + m1[row][2] \* m2[2][col] + m1[row][3] \* m2[3][col];  }  }  for (row = 0; row < 4; row++) {  for (col = 0; col < 4; col++) {  m2[row][col] = matTemp[row][col];  }  }  }  void translate3D(GLfloat tx, GLfloat ty, GLfloat tz)  {  M4 matTranslate3D;  m4SetIdentity(matTranslate3D);  matTranslate3D[0][3] = tx;  matTranslate3D[1][3] = ty;  matTranslate3D[2][3] = tz;  m4PreMultiply(matTranslate3D, matComposite);  }  void transformVerts3D()  {  GLint k;  for (k = 0; k < nVerts; k++) {  resultVerts[k].x = matComposite[0][0] \* verts[k].x + matComposite[0][1] \* verts[k].y + matComposite[0][2] \* verts[k].z + matComposite[0][3];  resultVerts[k].y = matComposite[1][0] \* verts[k].x + matComposite[1][1] \* verts[k].y + matComposite[1][2] \* verts[k].z + matComposite[1][3];  resultVerts[k].z = matComposite[2][0] \* verts[k].x + matComposite[2][0] \* verts[k].y + matComposite[2][2] \* verts[k].z + matComposite[2][3];  }  }  //画一个三棱锥，每个面设不同的颜色  void draw(pt3D\* mat)  {  int j;  for (int i = 0; i < 4; i++) {  glColor3f(colors[i].r, colors[i].g, colors[i].b);  for (j = i \* 3; j < i \* 3 + 3; j++) {  glVertex3f(mat[j].x, mat[j].y, mat[j].z);  }  }  }  void displayFunc()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glRotatef(30, 0.0f, 1.0f, 0.0f);  glBegin(GL\_LINES);//画坐标轴，可以省略，主要是为了看旋转变化  glColor3f(1.0, 0.0, 0.0);//y轴红色  glVertex3f(0, 0, 0);  glVertex3f(0, 4, 0);  glColor3f(0.0, 1.0, 0.0);  glVertex3f(0, 0, 0);  glVertex3f(4, 0, 0);//x轴绿色  glColor3f(0.0, 0.0, 1.0);  glVertex3f(0, 0, 0);  glVertex3f(0, 0, 3);//z轴蓝色  glEnd();  glBegin(GL\_TRIANGLES); // 绘制三角形  draw(verts);  glEnd();  glLoadIdentity();//不可以省略，否则后面对其它三维物体的变换也会发生在上面已经绘制的物体中  GLfloat scaleBack = 1 / scale;  glRotatef(angle, 0.0, 1.0, 0.0);  glScalef(1.0, scale, 1.0);  //GLfloat tx = 0, ty = 1.0, tz = 3;  m4SetIdentity(matComposite);  translate3D(tx, ty, tz);  transformVerts3D();  glBegin(GL\_TRIANGLES);  draw(resultVerts);  glEnd();  glScalef(1.0, scaleBack, 1.0);  glLoadIdentity();  glFlush();  }  /\*  键盘设置，下面的还可以改善：当旋转或平移到一定数值时，回到原始状态  \*/  void processSpecialKeys(int key, int x, int y)  {  switch (key) {  case GLUT\_KEY\_UP:ty += 0.1; break;  case GLUT\_KEY\_DOWN:ty -= 0.1; break;  case GLUT\_KEY\_LEFT:tz += 0.1; break;  case GLUT\_KEY\_RIGHT:tz -= 0.1; break;  case GLUT\_KEY\_PAGE\_UP:scale += 0.1; break;  case GLUT\_KEY\_PAGE\_DOWN:scale -= 0.1; break;  case GLUT\_KEY\_INSERT:angle += 10; break;  case GLUT\_KEY\_END:tx = 0; ty = 1; tz = 3; angle = 60; scale = 2; break;//回到原始状态  default:break;  }  displayFunc();  }  void main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(50, 50);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("3D");  init();  glutDisplayFunc(displayFunc);  glutSpecialFunc(processSpecialKeys);  glutMainLoop();  } |

实验结果：



实验十 可编程着色实验

时间：2022.5.17

地点：信息学院2202

1、实验内容：

使用opengl，片元着色器着色，P523

实验目的：验证片元着色器算法，获得着色结果。

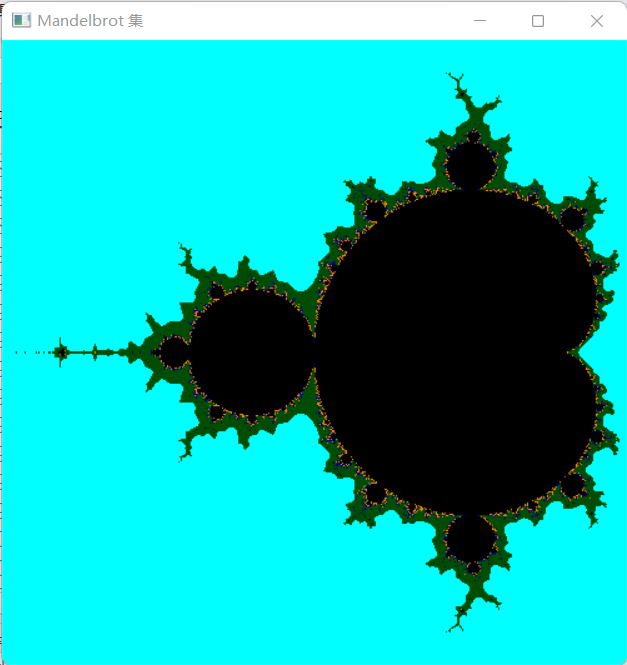
2、实验目的：

证片元着色器算法，获得着色结果。

3、实验代码：

|  |
| --- |
| //片元着色器着色  #include<GL/glut.h>  //设置一个窗口的大小  GLsizei winWidth = 500, winHeight = 500;  GLfloat xComplexMin = -2.00, xComplexMax = 0.50;  GLfloat yComplexMin = -1.25, yComplexMax = 1.25;  GLfloat complexWidth = xComplexMax - xComplexMin;  GLfloat complexHeight = yComplexMax - yComplexMin;  class complexNum {  public:      GLfloat x, y;  };  struct  color { GLfloat r, g, b; };  void init(void)  {      glClearColor(1.0, 1.0, 1.0, 0.0);  }  void plotpoint(complexNum z)  {      glBegin(GL\_POINTS);      glVertex2f(z.x, z.y);      glEnd();  }  complexNum complexSquare(complexNum z)  {      complexNum zSquare;      zSquare.x = z.x \* z.x - z.y \* z.y;      zSquare.y = 2 \* z.x \* z.y;      return zSquare;  }  GLint mandelSqTransf(complexNum z0, GLint maxIter)  {      complexNum z = z0;      GLint count = 0;      while ((z.x \* z.x + z.y \* z.y <= 4.0) && (count < maxIter)) {          z = complexSquare(z);          z.x += z0.x;          z.y += z0.y;          count++;      }      return count;  }  void mandelbrot(GLint nx, GLint ny, GLint maxIter)  {      complexNum z, zIncr;      color ptColor;      GLint iterCount;      zIncr.x = complexWidth / GLfloat(nx);      zIncr.y = complexHeight / GLfloat(ny);      for (z.x = xComplexMin; z.x < xComplexMax; z.x += zIncr.x)          for (z.y = yComplexMin; z.y < yComplexMax; z.y += zIncr.y) {              iterCount = mandelSqTransf(z, maxIter);              if (iterCount >= maxIter)                  ptColor.r = ptColor.g = ptColor.b = 0.0;              else if (iterCount > (maxIter / 10)) {                  ptColor.r = 1.0;                  ptColor.g = 0.5;                  ptColor.b = 0.0;              }              else if (iterCount > (maxIter / 10)) {                  ptColor.r = 1.0;                  ptColor.g = ptColor.b = 0.0;              }              else if (iterCount > (maxIter / 20)) {                  ptColor.b = 0.5;                  ptColor.r = ptColor.g = 0.0;              }              else if (iterCount > (maxIter / 100)) {                  ptColor.r = ptColor.b = 0.0;                  ptColor.g = 0.3;              }              else {                  ptColor.r = 0.0;                  ptColor.g = ptColor.b = 1.0;              }              glColor3f(ptColor.r, ptColor.g, ptColor.b);              plotpoint(z);          }  }  void displayFcn(void)  {      GLint nx = 1000, ny = 1000, maxIter = 1000;      glClear(GL\_COLOR\_BUFFER\_BIT);      mandelbrot(nx, ny, maxIter);      glFlush();  }  void winReshapeFcn(GLint newWidth, GLint newHeight) {      glViewport(0, 0, newHeight, newHeight);      glMatrixMode(GL\_PROJECTION);      glLoadIdentity();      gluOrtho2D(xComplexMin, xComplexMax, yComplexMin, yComplexMax);      glClear(GL\_COLOR\_BUFFER\_BIT);  }  void main(int argc, char\*\* argv)  {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);      glutInitWindowPosition(50, 50);      glutInitWindowSize(winWidth, winHeight);      glutCreateWindow("Mandelbrot 集");      init();      glutDisplayFunc(displayFcn);      glutReshapeFunc(winReshapeFcn);      glutMainLoop();  } |

实验结果：



实验十一 交互控制实验

时间：2022.5.17

地点：信息学院2202

1、实验内容：

使用opengl，完成鼠标、键盘交互操作

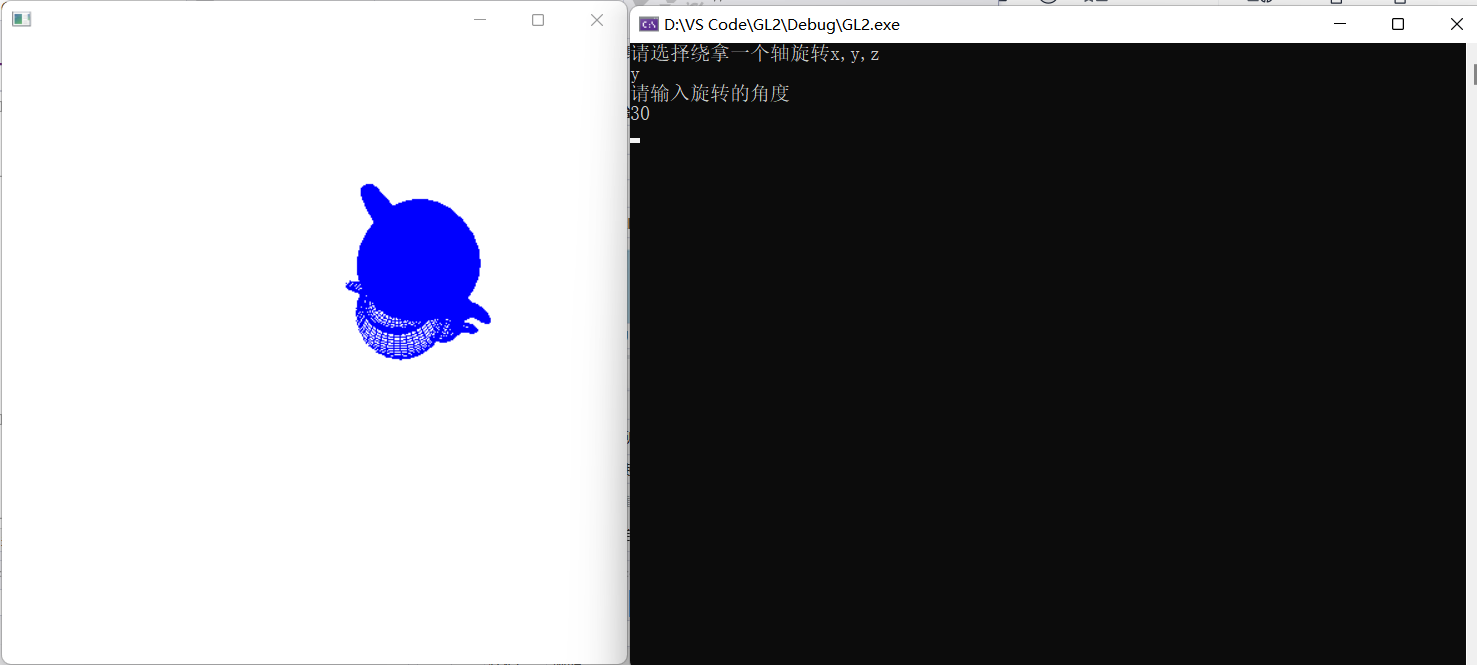
2、实验目的：

熟悉鼠标、键盘交互

3、实验代码：

|  |
| --- |
| //完成鼠标、键盘交互操作  #include<GL/glut.h>  #include<stdio.h>  GLsizei winWidth = 500, winHeight = 500;  char sixel;  float thera = 0;  float x = 0, y = 0, z = 0;  void init(void) {      glClearColor(1.0, 1.0, 1.0, 0.0);  }  void displayWirePolyhedra(float x, float y, float z, float thera) {      glClear(GL\_COLOR\_BUFFER\_BIT);      glColor3f(0.0, 0.0, 1.0);      gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);      glScalef(1.0, 1.0, 1.0);      glTranslatef(1.0, 2.0, 0.0);//下一个图形坐标      glutSolidTeapot(1.5);      //glutWireTeapot(1.5);//放大倍数      glScalef(1.0, 1.0, 1.0);//缩放比      glTranslatef(-1.0, -5.0, 0.0);//下一个图形坐标      glRotatef(thera, x, y, z);      glutWireTeapot(1.5);      //glutSolidTeapot(2.0);      glFlush();  }  void display() {      displayWirePolyhedra(x, y, z, thera);  }  void winReshapeFcn(GLint newWidth, GLint newHeight) {      glViewport(0, 0, newWidth, newHeight);      glMatrixMode(GL\_PROJECTION);      glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);      glMatrixMode(GL\_MODELVIEW);      glClear(GL\_COLOR\_BUFFER\_BIT);  }  void main(int argc, char\*\* argv) {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);      glutInitWindowPosition(100, 100);      glutInitWindowSize(winWidth, winHeight);      glutCreateWindow("   ");      init();      printf\_s("请选择绕拿一个轴旋转x,y,z \n");      scanf\_s("%c", &sixel);      getchar();      if (sixel == 'x') {          x = 1.0;          y = 0.0;          z = 0.0;          printf\_s("请输入旋转的角度\n");          scanf\_s("%f", &thera);      }      else if (sixel == 'y') {          x = 0.0;          y = 1.0;          z = 0.0;          printf\_s("请输入旋转的角度\n");          scanf\_s("%f", &thera);      }      else if (sixel == 'z') {          x = 0.0;          y = 0.0;          z = 1.0;          printf\_s("请输入旋转的角度\n");          scanf\_s("%f", &thera);      }      else {          printf\_s("输入有误\n");      }      glutDisplayFunc(display);      glutReshapeFunc(winReshapeFcn);      glutMainLoop();  } |

实验结果：



实验十二 三维观察实验

时间：2022.5.24

地点：信息学院2202

1、实验内容：

使用opengl，完成投影变换等实验；P 264

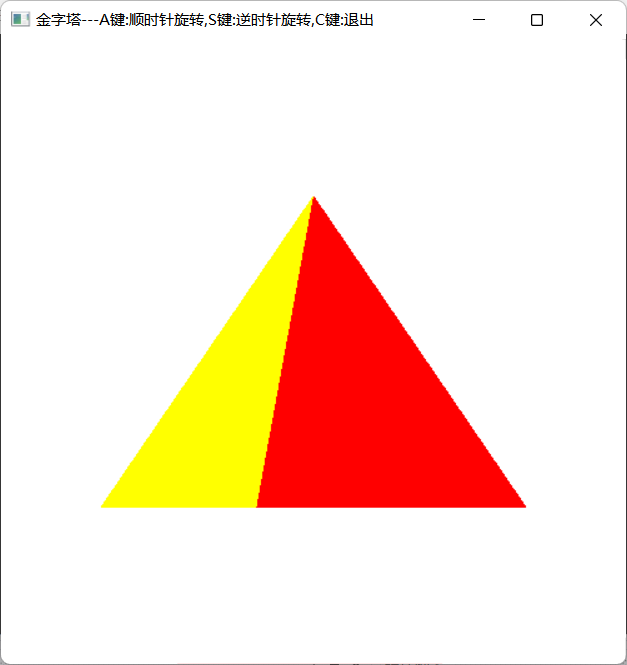
2、实验目的：

熟悉三维观察相关内容

3、实验代码：

|  |
| --- |
| //金字塔的投影变换实验  #include <stdlib.h>  #include <GL/glut.h>  float theta = 0.0;  void drawPyramid() //该金字塔在以原点为中心，边长为2的立方体范围内  {  glBegin(GL\_TRIANGLES);  glColor3f(1.0f, 0.0f, 0.0f); //前面为红色  glVertex3f(0.0f, 1.0f, 0.0f); //前面三角形上顶点  glVertex3f(-1.0f, -1.0f, 1.0f); //前面三角形左顶点  glVertex3f(1.0f, -1.0f, 1.0f); //前面三角形右顶点  glColor3f(0.0f, 1.0f, 0.0f); //右面为绿色  glVertex3f(0.0f, 1.0f, 0.0f); //右面三角形上顶点  glVertex3f(1.0f, -1.0f, 1.0f); //右面三角形左顶点  glVertex3f(1.0f, -1.0f, -1.0f); //右面三角形右顶点  glColor3f(0.0f, 0.0f, 1.0f); //背面为蓝色  glVertex3f(0.0f, 1.0f, 0.0f); //背面三角形上顶点  glVertex3f(1.0f, -1.0f, -1.0f); //背面三角形左顶点  glVertex3f(-1.0f, -1.0f, -1.0f); //背面三角形右顶点  glColor3f(1.0f, 1.0f, 0.0f); //左面为黄色  glVertex3f(0.0f, 1.0f, 0.0f); //左面三角形上顶点  glVertex3f(-1.0f, -1.0f, -1.0f); //左面三角形左顶点  glVertex3f(-1.0f, -1.0f, 1.0f); //左面三角形右顶点  glEnd();  glBegin(GL\_POLYGON); //金字塔底面正方形  glColor3f(0.5f, 0.5f, 0.5f); //底面为灰色  glVertex3f(-1.0f, -1.0f, 1.0f);  glVertex3f(1.0f, -1.0f, 1.0f);  glVertex3f(1.0f, -1.0f, -1.0f);  glVertex3f(-1.0f, -1.0f, -1.0f);  glEnd();  }  void display()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //清空颜色和深度缓存  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  //gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  glTranslatef(0.0f, 0.0f, -5.0f);  glRotatef(theta, 0.0f, 1.0f, 0.0f);  drawPyramid();  glutSwapBuffers();  }  void reshape(int w, int h) //重绘回调函数，在窗口首次创建或用户改变窗口尺寸时被调用  {  glViewport(0, 0, w, h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  //glFrustum(-1.0, 1.0, -1.0, 1.0, 3.1, 10.0);  //gluPerspective(45,1,0.1,10.0);  glOrtho(-2.0, 2.0, -2.0, 2.0, 2.0, 10.0);  }  void init()  {  glClearColor(1.0, 1.0, 1.0, 1.0);  glEnable(GL\_DEPTH\_TEST); //启动深度测试模式  }  void myKeyboard(unsigned char key, int x, int y)  {  if (key == 'a' || key == 'A')  theta += 5.0;  if (key == 's' || key == 'S')  theta -= 5.0;  if (key == 'c' || key == 'C')  exit(0);  if (theta > 360) theta -= 360;  if (theta < 0) theta += 360;  glutPostRedisplay(); //重新调用绘制函数  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGB);  glutInitWindowSize(500, 500);  glutInitWindowPosition(0, 0);  glutCreateWindow("金字塔---A键:顺时针旋转,S键:逆时针旋转,C键:退出");  glutReshapeFunc(reshape); //指定重绘回调函数  glutDisplayFunc(display);  glutKeyboardFunc(myKeyboard); //指定键盘回调函数  init();  glutMainLoop();  } |

实验结果：



实验十三 多面体实验

时间：2022.5.31

地点：信息学院2202

1、实验内容：

生成多面体线框图，P300

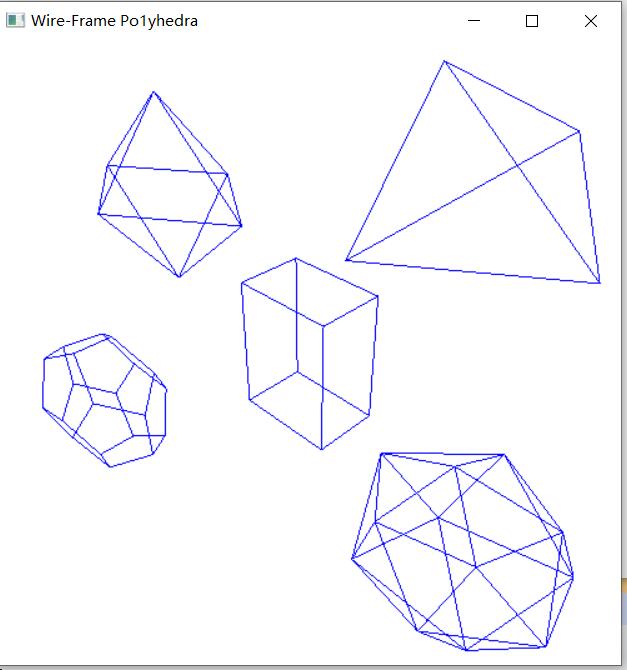
2、实验目的：

熟悉三维观察相关内容

3、实验代码：

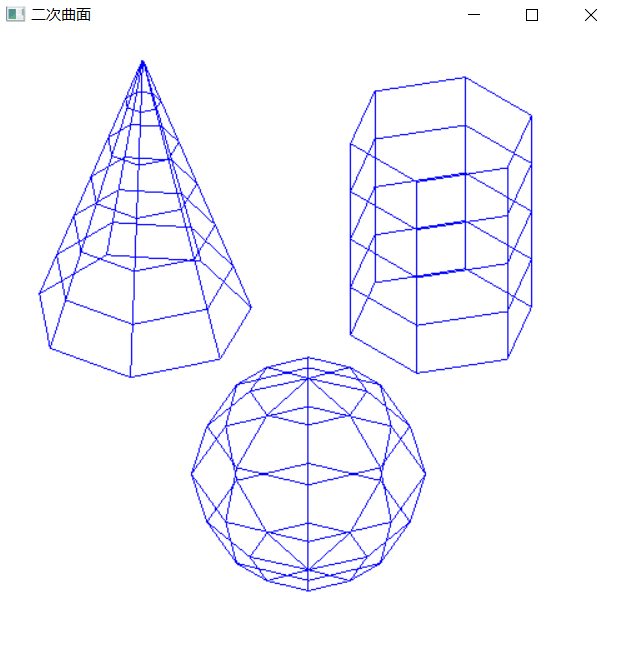
|  |
| --- |
| //P300生成多面体线框图  #include <GL/glut.h>  GLsizei winWidth = 500, winHeight = 500;  void init(void) {  glClearColor(1.0, 1.0, 1.0, 0.0);  }  void displayWirePolyhedra(void) {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.0, 1.0);  gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  glScalef(1.5, 2.0, 1.0);  glutWireCube(1.0);  glScalef(0.8, 0.5, 0.8);  glTranslatef(-6.0, -5.0, 0.0);  glutWireDodecahedron();  glTranslatef(8.6, 8.6, 2.0);  glutWireTetrahedron();  glTranslatef(-3.0, -1.0, 0.0);  glutWireOctahedron();  glScalef(0.8, 0.8, 1.0);  glTranslatef(4.3, -2.0, 0.5);  glutWireIcosahedron();  glFlush();  }  void winReshapeFcn(GLint newWidth, GLint newHeight) {  glViewport(0, 0, newWidth, newHeight);  glMatrixMode(GL\_PROJECTION);  glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);  glMatrixMode(GL\_MODELVIEW);  glClear(GL\_COLOR\_BUFFER\_BIT);  }  int main(int argc, char\*\* argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(100, 100);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("多面体");  init();  glutDisplayFunc(displayWirePolyhedra);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

实验结果一：



|  |
| --- |
| //教材P307的程序，实现了生成多面体线框图。  #include <GL/glut.h>  GLsizei winWidth = 500, winHeight = 500;  void init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0);  }  void wireQuadSurfs(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.0, 1.0);  gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);  glPushMatrix();  glTranslatef(1.0, 1.0, 0.0);  glutWireSphere(0.75, 8, 6);  glPopMatrix();  glPushMatrix();  glTranslatef(1.0, -0.5, 0.5);  glutWireCone(0.7, 2.0, 7, 6);  glPopMatrix();  GLUquadricObj\* cylinder;  glPushMatrix();  glTranslatef(0.0, 1.2, 0.8);  cylinder = gluNewQuadric();  gluQuadricDrawStyle(cylinder, GLU\_LINE);  gluCylinder(cylinder, 0.6, 0.6, 1.5, 6, 4);  glPopMatrix();  glFlush();  }  void winReshapeFcn(GLint newWidth, GLint newHeight)  {  glViewport(0, 0, newWidth, newHeight);  glMatrixMode(GL\_PROJECTION);  glOrtho(-2.0, 2.0, -2.0, 2.0, 0.0, 5.0);  glMatrixMode(GL\_MODELVIEW);  glClear(GL\_COLOR\_BUFFER\_BIT);  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(100, 100);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("二次曲面");  init();  glutDisplayFunc(wireQuadSurfs);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  return 0;  } |

实验结果二：



实验十四 曲线曲面生成实验

时间：2022.6.7

地点：信息学院2202

1、实验内容：

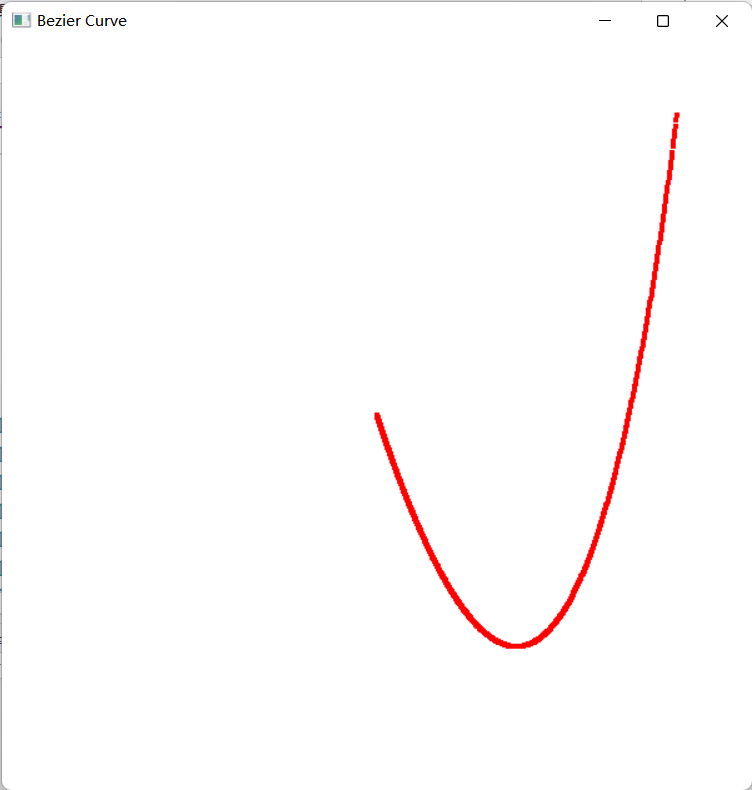
生成曲线或者曲面，P323

2、实验目的：

熟悉Bezier、样条等相关内容

3、实验代码：

|  |
| --- |
| //Bezier Curve算法  #include <GL/glut.h>  #include <stdio.h>  #include <math.h>  GLsizei winWIdth = 600, winHeight = 600;  GLfloat xwcMin = -50.0, xwcMax = 50.0, ywcMin = -50.0, ywcMax = 50.0;  class wcPt3D  {  public: GLfloat x, y, z;  };  void init(void)  {      glClearColor(1.0, 1.0, 1.0, 0.0);  }  void plotPoint(wcPt3D p)  {      glBegin(GL\_POINTS);      glVertex2f(p.x, p.y);      glEnd();  }  void binomialCoeffs(GLint n, GLint\* C)  {      GLint i, j;      for (i = 0; i <= n; i++)      {          C[i] = 1;          for (j = n; j >= i + 1; j--)          {              C[i] \*= j;          }          for (j = n - i; j >= 2; j--)          {              C[i] /= j;          }      }  }  void computeBezPt(GLfloat u, wcPt3D\* bezPt, GLint nCtrlPts, wcPt3D\* ctrlPts, GLint\* C)  {      GLint i, j = nCtrlPts - 1;      GLfloat bezBlendFcn;      bezPt->x = bezPt->y = bezPt->z = 0.0;      for (i = 0; i < nCtrlPts; i++)      {          bezBlendFcn = C[i] \* pow(u, j) \* pow(1 - u, j - i);          bezPt->x += ctrlPts[i].x \* bezBlendFcn;          bezPt->y += ctrlPts[i].y \* bezBlendFcn;          bezPt->z += ctrlPts[i].z \* bezBlendFcn;      }  }  //Bezier算法  void bezier(wcPt3D\* ctrlPts, GLint nCtrlPts, GLint nBezPts)  {      GLint\* C, k;      GLfloat u;      wcPt3D bezPt;      C = new GLint[nCtrlPts];      binomialCoeffs(nCtrlPts - 1, C);      for (k = 0; k <= nBezPts; k++)      {          u = GLfloat(k) / GLfloat(nBezPts);          computeBezPt(u, &bezPt, nCtrlPts, ctrlPts, C);          plotPoint(bezPt);      }      delete[] C;  }  void displayFcn(void)  {      GLint nCtrlPts = 4, nBezCurvePts = 1000;      wcPt3D ctrlPts[4] = {          {-40.0, -40.0, 0.0},          {-10.0, 200.0, 0.0},          {10.0, -200.0, 0.0},          {40.0, 40.0, 0.0} };      glClear(GL\_COLOR\_BUFFER\_BIT);      glPointSize(4);      glColor3f(1.0, 0.0, 0.0);      bezier(ctrlPts, nCtrlPts, nBezCurvePts);      glFlush();  }  void winReshapeFcn(GLint newWidth, GLint newHeight)  {      glViewport(0, 0, newWidth, newHeight);      glMatrixMode(GL\_PROJECTION);      glLoadIdentity();      gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);      glClear(GL\_COLOR\_BUFFER\_BIT);  }  void main(int argc, char\*\* argv)  {      glutInit(&argc, argv);      glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);      glutInitWindowPosition(50, 50);      glutInitWindowSize(winWIdth, winHeight);      glutCreateWindow("Bezier Curve");      init();      glutDisplayFunc(displayFcn);      glutReshapeFunc(winReshapeFcn);      glutMainLoop();  } |

实验结果：  


实验十五 消隐实验

时间：2022.6.15

地点：信息学院2202

1、实验内容：

完成消隐实验，采用Z-buffer算法完成消隐实验

2、实验目的：

熟悉Z-buffer、画家算法等相关内容

1. 实验代码：

|  |
| --- |
| import cv2  import numpy as np  import time  XLine = 1000  YLine = 700  img = np.zeros((YLine, XLine, 3), np.uint8) # 生成背景  pts = []  z\_buffer = np.zeros((XLine, YLine), np.uint32)  z\_buffer += 100000 # 无穷大  # 为了满足需求而设置的值  count = 0  def zfunc(x, y):  return x + y  def draw\_func(event, x, y, flags, param):  global pts  if event == cv2.EVENT\_LBUTTONDOWN:  print('(x:', x, ',y:', y, ')')  str1 = '(x:' + str(x) + ',y:' + str(y) + ')'  # cv2.putText(img,str1 , (x, y), cv2.FONT\_HERSHEY\_PLAIN,1.0, (0, 0, 255), thickness=1)  cv2.circle(img, (x, y), 1, (0, 255, 0), thickness=1)  pts.append([x, y])  # 活性边表与新边表中的节点  class Node:  def \_\_init\_\_(self, xmin, ymax, k1, next=None):  self.x = xmin # 也就是x的初始值  self.ymax = ymax  self.k1 = k1 # 也就是δx  self.next = next  def print(self):  t = self  while t != None:  print(t.x, t.ymax, t.k1, end="-->")  t = t.next  print()  def FillPoly(fpts, color):  global z\_buffer, count  count += 1  # cv2.fillPoly(img,fpts,color)  Ymin, Ymax = 1000000, 0  for pt in fpts:  if Ymin > pt[1]:  Ymin = pt[1]  if Ymax < pt[1]:  Ymax = pt[1]  # 构造新边表  NET = [x for x in range(Ymax + 1)]  for i in range(len(fpts)):  # 后面一个点的序号  j = i + 1 if i < len(fpts) - 1 else 0  if fpts[i][1] > fpts[j][1]:  ymax, ymin = fpts[i][1], fpts[j][1]  xmin = fpts[j][0]  else:  ymin, ymax = fpts[i][1], fpts[j][1]  xmin = fpts[i][0]  # 与x轴平行,跳过该边  if fpts[i][1] == fpts[j][1]:  continue  if fpts[i][0] == fpts[j][0]:  k1 = 0  else:  k1 = (fpts[i][0] - fpts[j][0]) / (fpts[i][1] - fpts[j][1])  if type(NET[ymin]) != type(66):  t = NET[ymin]  while t.next != None:  t = t.next  t.next = Node(xmin, ymax, k1)  else:  NET[ymin] = Node(xmin, ymax, k1)  for i, x in enumerate(NET):  if type(x) != type(66):  print(i)  x.print()  AET = Node(0, 0, 0)  print("YY")  for y in range(Ymin, Ymax + 1):  # print(y)  # 慢慢画  cv2.waitKey(3)  cv2.imshow('srchllo', img)  # 加入新边  if type(NET[y]) != type(66):  t = AET  while t.next:  t = t.next  t.next = NET[y]  # 填充颜色  # 按x排序  xs = []  t = AET  while t.next:  t = t.next  xs.append(t.x)  xs = sorted(xs)  for i in range(1, len(xs), 2):  for x in range(int(xs[i - 1]), int(xs[i])):  # z=zfunc(x,y) 这里是zbuffer消隐操作  z = count  if z == 3 and z\_buffer[x, y] == 1: # 这里是为了交作业 ヽ(ﾟ▽ﾟ)ノ  z\_buffer[x, y] = z  cv2.circle(img, (x, y), 1, color, thickness=1)  elif z < z\_buffer[x, y]:  z\_buffer[x, y] = z  cv2.circle(img, (x, y), 1, color, thickness=1)  # 删除并更新边  t = AET  while t.next:  ct = t.next  ct.x += ct.k1  if ct.ymax == y:  t.next = ct.next  t = t.next  if t == None:  break  cv2.namedWindow('srchello')  cv2.setMouseCallback('srchello', draw\_func)  while (1):  cv2.imshow('srchello', img)  code = cv2.waitKey(100)  if code == ord('q'): # 按下q退出  break  elif code == ord('e'): # 按下该键表示画完一个多边形  tmp = np.array(pts, np.int32)  tmp = [tmp.reshape((-1, 1, 2))]  fpts = pts  pts = []  # cv2.polylines(img,tmp,True,(255,255,255))  # 按下 r,g,b 填充多边形为相应的颜色  elif code == ord('r'):  FillPoly(fpts, (0, 0, 255))  elif code == ord('y'):  FillPoly(fpts, (0, 255, 255))  elif code == ord('b'):  FillPoly(fpts, (255, 0, 0))  cv2.destroyAllWindows() |

1. 实验结果：

