实验六 二维图形几何变换实验

时间：2022年4月27日

地点：信息学院机房

1、实验内容

实验一：使用opengl，对二维几何图形进行几何变换

实验二：鼠标函数实验

2、实验目的

验证二维几何变换实验，平移、比例、旋转、错切、对称变换，其中三种

3、实验代码

实现平移 旋转 比例三种功能

|  |
| --- |
| #include <windows.h>  #include <GL/glut.h>  #include <stdlib.h>  #include <math.h>  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, 0.0);  }  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 rotate2D(wcPt2D pivotPt, GLfloat theta)  {  Matrix3x3 matRot;  matrix3x3SetIdentity(matRot);  matRot[0][0] = cos(theta);  matRot[0][1] = -sin(theta);  matRot[0][2] = pivotPt.x \* (1 - cos(theta)) +  pivotPt.y \* sin(theta);  matRot[1][0] = sin(theta);  matRot[1][1] = cos(theta);  matRot[1][2] = pivotPt.y \* (1 - cos(theta)) -  pivotPt.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 transformVerts2D(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 displayFcn()  {  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);  rotate2D(pivPt, theta);  translate2D(tx, ty);  transformVerts2D(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);  }  int main(int argc, char \*\*argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(50, 50);  glutInitWindowSize(winWidth, winHeight);  glutCreateWindow("Geometric Transformation Sequence");  init();  glutDisplayFunc(displayFcn);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  return 0;  } |

使用鼠标控制完成裁剪算法：

|  |
| --- |
| #include<windows.h>  #include<iostream>  #include<gl\glut.h>  #include<utility>  #include<vector>  #include<math.h>  using namespace std;  const int screenWidth = 800;  const int screenHeight = 600;  const int windowPositionX = 300;  const int windowPositionY = 150;  class Point {  public:  double x, y;  Point() {}  Point(double x, double y) {  this->x = x;  this->y = y;  }  };  vector<Point> vertice; //顶点  //线段  vector<pair<Point, Point> > V;  //用于裁剪的窗口  class Window {  public:  int x, y;  int width;  int height;  int l, r, t, b; //边界：左右上下  Window(int x, int y, int width, int height) {  this->x = x;  this->y = y;  this->width = width;  this->height = height;  l = x;  r = x + width;  t = y + height;  b = y;  }  };  Window myWindow(250, 200, 300, 200);  //画裁剪窗口  void draw() {  glBegin(GL\_LINE\_LOOP);  glVertex2i(myWindow.l, myWindow.b);  glVertex2i(myWindow.l, myWindow.t);  glVertex2i(myWindow.r, myWindow.t);  glVertex2i(myWindow.r, myWindow.y);  glEnd();  }  //画点函数  void draw\_a\_point(int x, int y)  {  glBegin(GL\_POINTS);  glColor3f(0, 0, 0);  glVertex2f(x, y);  glEnd();  glFlush();  }  //判断位置  void chopLine(Point& p, unsigned char code, double dely, double delx) {  if (code & 1) { //0001 左方，位与运算，结果不为0时证明在窗口左方有交点  p.y += (myWindow.l - p.x) \* dely / delx;  p.x = myWindow.l;  }  else if (code & 2) { //0010 右方  p.y += (myWindow.r - p.x) \* dely / delx;  p.x = myWindow.r;  }  else if (code & 4) { //0100 下方  p.x += (myWindow.b - p.y) \* delx / dely;  p.y = myWindow.b;  }  else { //1000 上方  p.x += (myWindow.t - p.y) \* delx / dely;  p.y = myWindow.t;  }  }  //按位或，生成编号  void generateCode(Point& p, unsigned char& code) {  if (p.x < myWindow.l) code |= 1;  if (p.y > myWindow.t) code |= 8;  if (p.x > myWindow.r) code |= 2;  if (p.y < myWindow.b) code |= 4;  }  //裁剪  int cut(pair<Point, Point>& tmp) {  unsigned char code1;  unsigned char code2;  int k = 0;  do {  code1 = 0;  code2 = 0;  generateCode(tmp.first, code1);  generateCode(tmp.second, code2);  if ((code1 | code2) == 0) { //完全在窗口里面（0000|0000）  return 1;  }  else if ((code1 & code2) != 0) { //在某条边界同侧，即完全在窗口外面  return 0;  }  if (code1 != 0) {  chopLine(tmp.first, code1, tmp.second.y - tmp.first.y, tmp.second.x - tmp.first.x);  }  if (code2 != 0) {  chopLine(tmp.second, code2, tmp.second.y - tmp.first.y, tmp.second.x - tmp.first.x);  }  k++;  } while (1);  }  void func() {  pair<Point, Point> tmp(Point(vertice[0].x, vertice[0].y), Point(vertice[1].x, vertice[1].y));  V.push\_back(tmp);  glBegin(GL\_LINES);  glColor3f(0.0f, 0.0f, 0.0f);  glVertex2f(V[0].first.x, V[0].first.y);  glVertex2f(V[0].second.x, V[0].second.y);  glColor3f(1.0f, 0.0f, 0.0f);  int a = cut(V[0]);  if (a == 1) {  glVertex2f(V[0].first.x, V[0].first.y);  glVertex2f(V[0].second.x, V[0].second.y);  }  glEnd();  glFlush();  //V.pop\_back();  //ertice.pop\_back();  V.clear();  vertice.clear();  }  void mydisplay(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT); //clear the screen  glColor3f(0.0f, 0.0f, 0.0f);  glLineWidth(3.0);  draw();  //func();  glFlush();  }  void mymouse(int button, int state, int x, int y)  {  if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)  {  draw\_a\_point(x, screenHeight - y);  Point p(x, screenHeight - y);  vertice.push\_back(p);  cout << "起点" << ": (" << x << ", " << y << ")" << endl;  }  if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)  {  draw\_a\_point(x, screenHeight - y);  Point p(x, screenHeight - y);  vertice.push\_back(p);  cout << "终点" << ": (" << x << ", " << y << ")" << endl;  cout << "开始剪裁" << endl;  func();  }  }  int main(int argc, char\*\* argv)  {  cout << "先点击鼠标左键画起点；" << endl << "再点击鼠标右键画终点并开始裁剪线段；" << endl;  glutInit(&argc, argv); //initialize the toolkit  glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);//t the display mode  glutInitWindowSize(screenWidth, screenHeight); //t the window size  glutInitWindowPosition(windowPositionX, windowPositionY); //t the window position  glutCreateWindow("Cohen-Sutherland 裁剪算法"); //open the screen window  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0, screenWidth, 0, screenHeight);  glClearColor(1, 1, 1, 1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glutDisplayFunc(&mydisplay); //register the callback functions  glutMouseFunc(&mymouse);  glutMainLoop(); //get into a perpetual loop  } |

4、实验总结

在设计图形变换功能时，主要时通过设计好3\*3的变换矩阵与x y坐标相乘从而达到图形二维变换的功能。而鼠标控制则是调用了几个简单的控制函数可以做到。