实验五 圆的扫描转换和种子点填充实验

时间：2022年4月13日

地点：信息学院2202

1、实验内容

圆的扫描转换和种子点填充算法的实现

2、实验目的

掌握图的扫描转换与种子点填充算法

3、实验代码

圆的扫描转换：

|  |
| --- |
| #include <windows.h>  #include<GL\glut.h>  #include<iostream>  #include<cmath>  #include <stdio.h>  using namespace std;  void init(void)  {  glClearColor(1.0, 1.0, 1.0, 0.0); // Set display-window color to white.  glMatrixMode(GL\_PROJECTION); // Set projection parameters.  gluOrtho2D(0.0, 200.0, 0.0, 150.0);  }  /\*  画点  \*/  void setPixel(int x, int y)  {  glColor3f(0.0, 0.0, 0.0);  glPointSize(2.0f);  glBegin(GL\_POINTS);  glVertex2f(x, y);  glEnd();  glFlush();  }  /\*  8路对称  \*/  void Cirpot(int x0, int y0, int x, int y)  {  // 1  setPixel((x0 + x), (y0 + y));  // 2  setPixel((x0 + y), (y0 + x));  // 3  setPixel((x0 + y), (y0 - x));  // 4  setPixel((x0 + x), (y0 - y));  // 5  setPixel((x0 - x), (y0 - y));  // 6  setPixel((x0 - y), (y0 - x));  // 7  setPixel((x0 - y), (y0 + x));  // 8  setPixel((x0 - x), (y0 + y));  }  /\*  中点画圆算法  \*/  void MidPoint\_Circle(int x0, int y0, int r)  {  int x = 0;  int y = r;  int d = 1 - r; // d = 1.25-r的取整的结果  Cirpot(x0, y0, x, y);  while (x < y)  {  if (d < 0)  {  d += 2 \* x + 3;  }  else  {  d += 2 \* (x - y) + 5;  y--;  }  x++;  Cirpot(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 = 100, y0 = 100, r = 80;  MidPoint\_Circle(x0, y0, r);  }  int main(int argc, char\* argv[])  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(200, 200);  glutInitWindowSize(400, 400);  glutCreateWindow("MidCircle");  glutDisplayFunc(display);  glutReshapeFunc(ChangeSize);  init();  glutMainLoop();  return 0;  } |

实验二：输入多边形，种子点位置，填充多边形

|  |
| --- |
| #define \_STDCALL\_SUPPORTED  #ifdef \_\_APPLE\_\_  #include <GLUT/glut.h>  #else  #include <GL/glut.h>  #endif  #include <stdlib.h>  typedef float Color[3];  //获取像素点的颜色  void getpixel(GLint x, GLint y, Color color) {  glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, color);  }  //画点函数  void setpixel(GLint x, GLint y) {  glBegin(GL\_POINTS);  glVertex2f(x, y);  glEnd();  }  //比较颜色是否相等  int compareColor(Color color1, Color color2) {  if (color1[0] != color2[0] || color1[1] != color2[1] || color1[2] != color2[2]) { return 0; }  else { return 1; }  }  void boundaryFill4(int x, int y, Color fillColor, Color boarderColor) {  Color interiorColor;  getpixel(x, y, interiorColor);  if (compareColor(interiorColor, fillColor) == 0 && compareColor(interiorColor, boarderColor) == 0) {  setpixel(x, y);  boundaryFill4(x + 1, y, fillColor, boarderColor);  boundaryFill4(x - 1, y, fillColor, boarderColor);  boundaryFill4(x, y + 1, fillColor, boarderColor);  boundaryFill4(x, y - 1, fillColor, boarderColor);  }  }  void boundaryFill8(int x, int y, Color fillColor, Color boarderColor) {  Color interiorColor, a, b, c, d;  getpixel(x, y, interiorColor);  getpixel(x + 1, y - 1, a);  getpixel(x, y - 1, b);  getpixel(x, y + 1, c);  getpixel(x - 1, y, d);  int i = 0;  if (compareColor(a, boarderColor) == 1) i++;  if (compareColor(b, boarderColor) == 1) i++;  if (compareColor(c, boarderColor) == 1) i++;  if (compareColor(d, boarderColor) == 1) i++;  if (i <= 1) {  if (compareColor(interiorColor, fillColor) == 0 && compareColor(interiorColor, boarderColor) == 0) {  setpixel(x, y);  boundaryFill8(x + 1, y, fillColor, boarderColor);  boundaryFill8(x, y - 1, fillColor, boarderColor);  boundaryFill8(x - 1, y, fillColor, boarderColor);  boundaryFill8(x, y + 1, fillColor, boarderColor);  boundaryFill8(x + 1, y - 1, fillColor, boarderColor);  boundaryFill8(x - 1, y - 1, fillColor, boarderColor);  boundaryFill8(x - 1, y + 1, fillColor, boarderColor);  boundaryFill8(x + 1, y + 1, fillColor, boarderColor);  }  }  }  void polygon() {  glBegin(GL\_LINE\_LOOP);  glLineWidth(5);  //此处修改坐标，绘制多边形  glVertex2f(100, 100);  glVertex2f(100, 200);  //glVertex2f(200, 200);  glVertex2f(200, 100);  glEnd();  }  void display(void) {  Color fillColor = {1.0, 1.0, 1.0};//设置填充颜色  Color boarderColor = {1.0, 0.0, 0.0};//边界颜色  glClear(GL\_COLOR\_BUFFER\_BIT);  glViewport(0, 0, 500, 500);  glColor3fv(boarderColor);  polygon();  glColor3fv(fillColor);  boundaryFill4(110, 110, fillColor, boarderColor);//设置起点坐标及颜色  //boundaryFill8(110, 110, fillColor, boarderColor);  glFlush();  }  int main(int argc, char \*\*argv) {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RED);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow("BoundaryFill1");  glClearColor(1, 1, 1, 0.0);  glMatrixMode(GL\_PROJECTION);//投影模型  gluOrtho2D(0.0, 500.0, 0.0, 500.0);  glutDisplayFunc(display);  glutMainLoop();  return 0;  } |

4、实验总结

Bresenham画圆算法与画线算法原理基本相同，主要是用X轴、Y轴及y=x和y=-x四条直线将圆分为八份，画圆时在绘制一块区域中的每个点时，对应的画出另外七个点，最后画完这一个八分之一圆弧就画完了整个圆。而种子填充算法是从区域内任一个种子像素位置（x，y）开始，由内向外将填充色扩散到整个多边形区域的填充过程。