**计算机图形学实验**

**姓 名：王志元**

**学 号：20201050318**

**专 业：计算机科学与技术**

**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

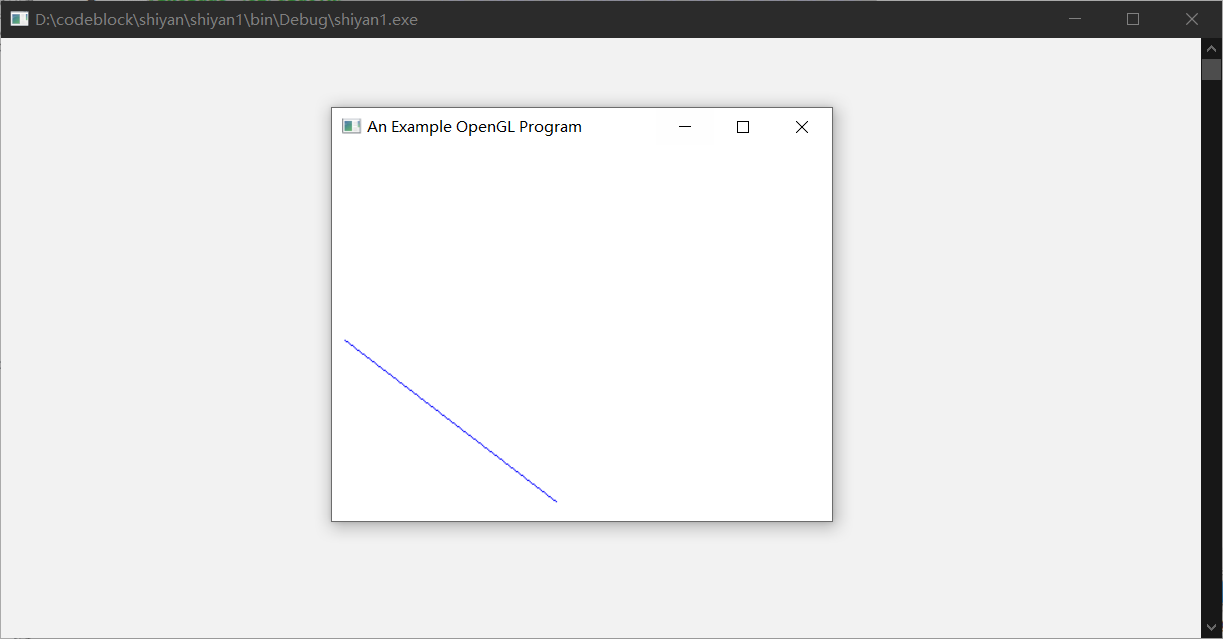
1. 实验内容：
2. 在CodeBlocks环境下安装OpenGL（使用GLUT）
   1. 安装GLUT
   2. CodeBlocks上新建GLUT工程
   3. 编写GLUT程序

2、绘制直线

1. 实验目的：
2. 掌握OpenGL glut的安装;
3. 掌握OpenGL编程初步;
4. 熟悉OpenGL glut下的编程框架;
5. 使用OpenGL绘制点线等图元。
6. 实验代码：

|  |
| --- |
| #include <windows.h>  #include <GL/glut.h> // (or others, depending on the system in use)  void init (void)  {  glClearColor (1.0, 1.0, 1.0, 0.0); // 指定清空颜色（背景色）为白色  gluOrtho2D (0.0, 400.0, 0.0, 300.0); //指定二维坐标系中被显示的区域  }  void display (void)  {  glClear (GL\_COLOR\_BUFFER\_BIT); // 清空显示窗口  glColor3f (0.0, 0.0, 1.0); // 指定前景色（当前绘制颜色）为蓝色  glBegin (GL\_LINES);  glVertex2i (180, 15); // 指定顶点  glVertex2i (10, 145);  glEnd ( );  glFlush ( ); // 使绘制立即反映到屏幕上  }  int main (int argc, char\*\* argv)  {  glutInit (&argc, argv); // 初始 GLUT.  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB); //设定显示模式  glutInitWindowPosition (50, 100); // 设定窗口位置  glutInitWindowSize (400, 300); // 设定窗口大小  glutCreateWindow ("An Example OpenGL Program"); // 用前面指定参数创建窗口，并定义窗口名称  init ( ); // 进行一些初始化工作  glutDisplayFunc (display); // 指定绘制的回调函数  glutMainLoop ( ); // 进入无穷循环，等待事件处理 |

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

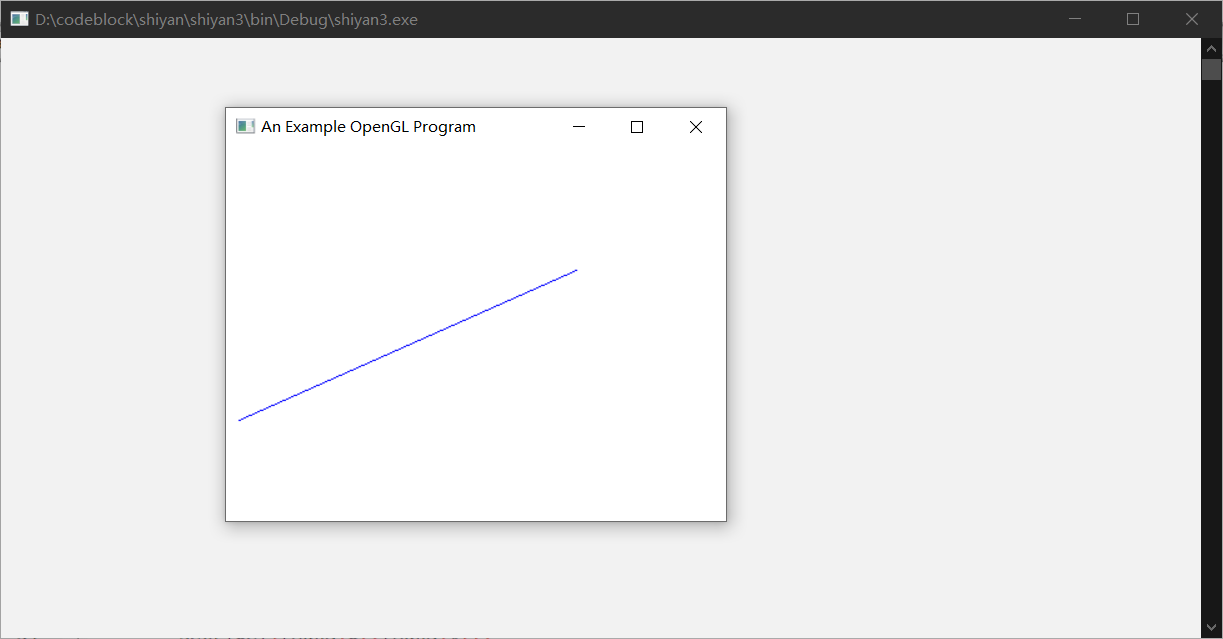
1、实验内容：

2、实验目的：

3、实验代码：

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include <GL/glut.h> // (or others, depending on the system in use)  void init(void){  glClearColor(1.0,1.0,1.0,0.0);  gluOrtho2D(0.0,400.0,0.0,300.0);  }  inline int round (const float a){  return int(a + 0.5);  }  void setPixel(GLint x,GLint y){  glBegin(GL\_POINTS);  glVertex2i(x,y);  glEnd();  glFlush();  }  void lineDDA(){  int x0,xEnd,y0,yEnd;  x0 = 280;  xEnd = 10;  y0 = 200;  yEnd = 80;  int dx = xEnd - x0;  int dy = yEnd - y0;  int steps;  int k;  float xIncrement,yIncrement;  float x = x0;  float y = y0;  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0,0.0,1.0);  if(fabs(dx)>fabs(dy)){  steps = fabs(dx);  }else{  steps = fabs(dy);  }  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));  }  }  int main (int argc, char\*\* argv)  {  glutInit (&argc, argv); // 初始 GLUT.  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB); //设定显示模式  glutInitWindowPosition (50, 100); // 设定窗口位置  glutInitWindowSize (400, 300); // 设定窗口大小  glutCreateWindow ("An Example OpenGL Program"); // 用前面指定参数创建窗口，并定义窗口名称  init ( ); // 进行一些初始化工作  glutDisplayFunc(lineDDA);  glutMainLoop();  } |

4、实验结果：



实验三 Bresenham、改进Bresenham算法生成直线段实验

时间：2022.3.30

地点：信息学院2202

1. 实验内容：

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

2、实验目的：

1、安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

Bresenham算法

|  |
| --- |
| #include <windows.h>  #include <GL/glut.h>  #include <stdlib.h>  #include <math.h>  #include <stdio.h>  void init(void){  glClearColor(1.0,1.0,1.0,0.0);  gluOrtho2D(0.0,400.0,0.0,300.0);  }  void setPixel(GLint x,GLint y){  glBegin(GL\_POINTS);  glVertex2i(x,y);  glEnd();  glFlush();  }  void lineBres(){  int x0 = 100;  int y0 = 1;  int xEnd = 10;  int yEnd = 100;  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+=twoDy;  }else{  y++;  p+=twoDyMinusDx;  }  setPixel(x,y);  }  }  int main (int argc, char\*\* argv)  {  glutInit (&argc, argv); // 初始 GLUT.  glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB); //设定显示模式  glutInitWindowPosition (50, 100); // 设定窗口位置  glutInitWindowSize (400, 300); // 设定窗口大小  glutCreateWindow ("An Example OpenGL Program"); // 用前面指定参数创建窗口，并定义窗口名称  init ( ); // 进行一些初始化工作  glutDisplayFunc(lineBres);  glutMainLoop();  } |

改进Bresenham算法:

#include<GL/glut.h>

#include<stdio.h>

#include<math.h>

#include<iostream>

using namespace std;

void setPixel(GLint x, GLint y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

glFlush();

}

void lineBres() {

int x0, y0, xEnd, yEnd;

cout << "Please input x0,y0,xEnd,yEnd:" << endl;

cin >> x0 >> y0 >> xEnd >> yEnd;

int dx = fabs(xEnd - x0), dy = fabs(yEnd - y0);

float k = dy / (dx \* 1.0), d = 0;

int x, y;

x = x0; y = y0;

setPixel(x, y);

while (x < xEnd) {

x++;

d = d + k;

if (d >= 0.5) { y++; d = d - 1; }

setPixel(x, y);

}

}

void display(void) {

glClearColor(1.0, 1.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0, 0, 500, 500);

lineBres();

glFlush();

}

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

glutInit(&argc, argv); // 初始 GLUT.

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //设定显示模式

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("lineBres"); // 用前面指定参数创建窗口，并定义窗口名称

glutDisplayFunc(display); // 指定绘制的回调函数

glColor3f(0.0, 0.0, 1.0);//颜色

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

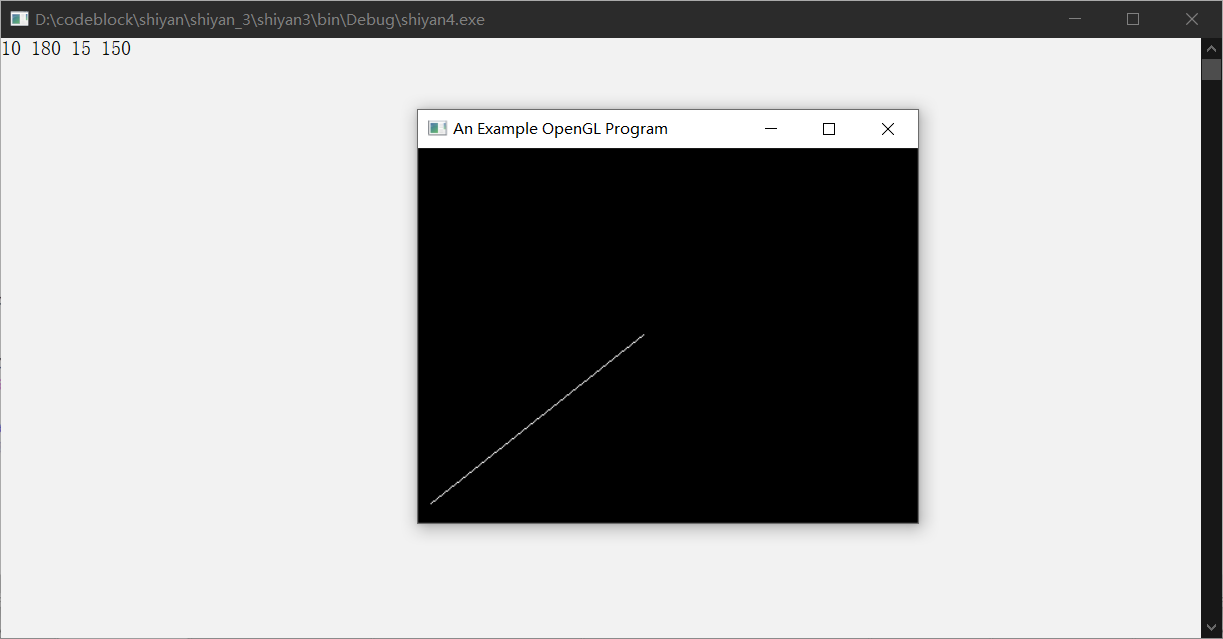
glutMainLoop();

return 0;

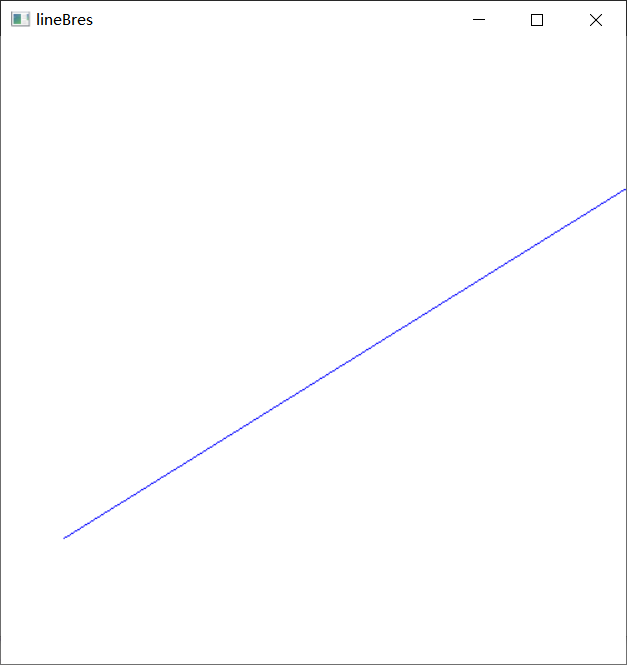
}

4、实验结果：

Bresenham算法：



改进Bresenham算法：



实验四 填充算法实验

时间：2022.4.6

地点：信息学院2202

1. 实验内容：

1、教材P66，填充六边形

2、实验目的：

1、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：（教材P66，填充六边形）

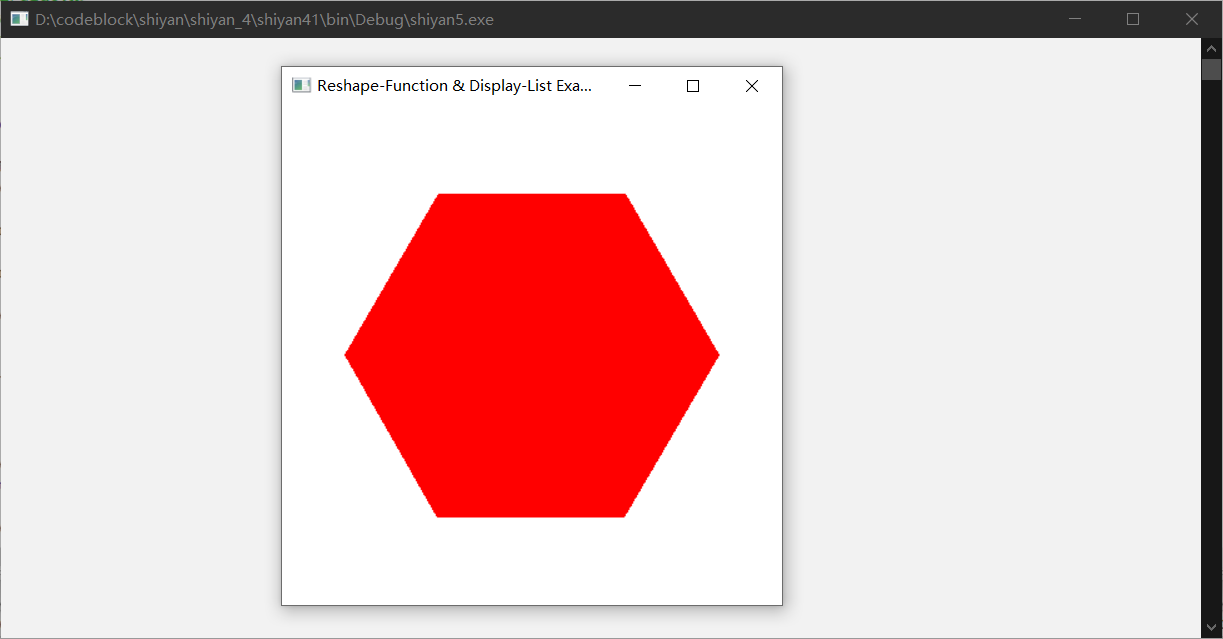
|  |
| --- |
| #include <windows.h>  #include <GL/glut.h>  #include <math.h>  #include <stdlib.h>  #include <stdio.h>  const double TWO\_PI = 6.2831853;  GLsizei winWidth = 400, winHeight = 400;  GLuint regHex;  class screenPt{  private:  GLint x,y;  public:  screenPt(){  x=y=0;  }  void setCoords(GLint xCoord,GLint yCoord){  x = xCoord;  y = yCoord;  }  GLint getx() const{  return x;  }  GLint gety() const{  return y;  }  };  static void init(void){  screenPt hexVertex,circCtr;  GLdouble theta;  GLint k;  circCtr.setCoords(winWidth/2,winHeight/2);  glClearColor(1.0,1.0,1.0,0.0);  regHex = glGenLists(1);  glNewList(regHex,GL\_COMPILE);  glColor3f(1.0,0.0,0.0);  glBegin(GL\_POLYGON);  for (k=0;k<6;k++){  theta = TWO\_PI \* k / 6.0;  hexVertex.setCoords(circCtr.getx()+150\*cos(theta),circCtr.gety()+150\*sin(theta));  glVertex2i(hexVertex.getx(),hexVertex.gety());  }  glEnd();  glEndList();  }  void regHexagon (void){  glClear(GL\_COLOR\_BUFFER\_BIT);  glCallList(regHex);  glFlush();  }  void winReshapeFcn (int newWidth,int newHeight){  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0.0,(GLdouble) newWidth,0.0,(GLdouble) newHeight);  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("Reshape-Function & Display-List Example");  init();  glutDisplayFunc(regHexagon);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  return 0;  } |

实验代码：（验证扫描线填充，任意的多边形边数）

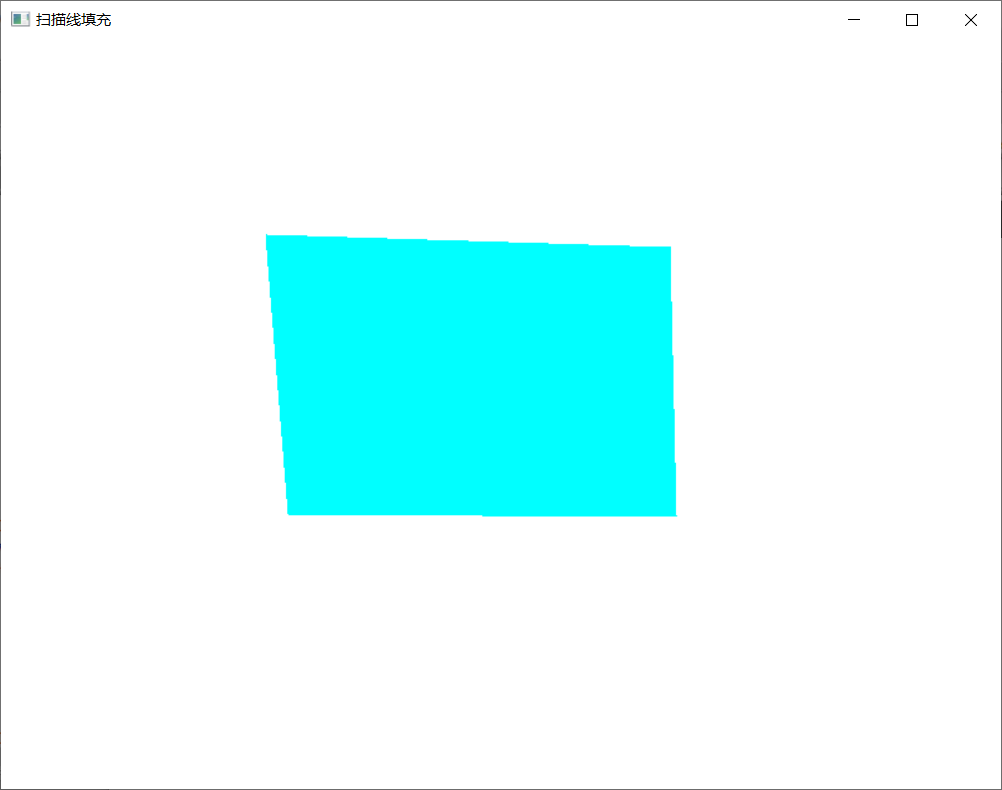
|  |
| --- |
| #include <windows.h>  #include<iostream>  #include<gl/glut.h>  #include<algorithm>  #include<vector>  #include<stack>  #include<queue>  using namespace std;  const int window\_width = 800, window\_height = 600;  const int maxn = 99999;  struct point  {  float x, y;  point(){}  point(int xx, int yy):  x(xx), y(yy) {}  };  vector<point> vertice; //顶点  typedef struct XET  {  float x;  float dx; // 从当前扫描线到下一条扫描线间x的增量，即斜率的倒数  float ymax; //该边所交的最高扫描线的坐标值ymax  XET\* next;  }AET, NET; //AET 活性边表； NET新边表  void draw\_a\_point(int x, int y);  void PolyScan();  void mymouse(int button, int state, int x, int y);  void display();  int main(int argc, char\* argv[])  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowPosition(100, 50);  glutInitWindowSize(window\_width, window\_height);  glutCreateWindow("扫描线填充");  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0, window\_width, 0, window\_height);  glClearColor(1, 1, 1, 1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glutMouseFunc(&mymouse);  glutDisplayFunc(&display);  glutMainLoop();  return 0;  }  //画点函数  void draw\_a\_point(int x, int y)  {  glBegin(GL\_POINTS);  glColor3f(0, 1, 1);  glVertex2f(x, y);  glEnd();  glFlush();  }  void PolyScan()  {  /\*得到最高点的y坐标\*/  int Max\_Y = 0;  for (int i = 0; i < vertice.size(); i++)  /\*Max\_Y = max(Max\_Y, vertice[i].y);\*/  if (vertice[i].y > Max\_Y)  Max\_Y = vertice[i].y;  //初始化AET表  AET\* pAET = new AET;  pAET->next = NULL;  //初始化NET表  NET\* pNET[800]; //吊桶  for (int i = 0; i <= Max\_Y; i++)  {  pNET[i] = new NET;  pNET[i]->next = NULL;;  }  //扫描并且建立NET表  int len = vertice.size(); //顶点个数  for (int i = 0; i <= Max\_Y; i++)  {  for (int j = 0; j < len; j++) //扫描每个点  {  if (i == vertice[j].y)  {  //如果一个点和前一个点有一条边相连，则该点和后面一个点也相连  //！这个式子 便于最后一个顶点和第一个点相连 和 防止出现负数  //判断当前点的高低，使ymax、DX、DY的计算有变化  if (vertice[(j - 1 + len) % len].y > vertice[j].y)  {  //前一个点在当前点的上方  NET\* p = new NET;  p->x = vertice[j].x;  p->ymax = vertice[(j - 1 + len) % len].y;//与当前扫描线相交的活性边 的 最高点即为相邻顶点的y  float DX = vertice[(j - 1 + len) % len].x - vertice[j].x;  float DY = vertice[(j - 1 + len) % len].y - vertice[j].y;  p->dx = DX / DY;//dx为直线斜率的倒数  p->next = pNET[i]->next;  pNET[i]->next = p;  }  if (vertice[(j + 1) % len].y > vertice[j].y)  {  //后一个点在当前点的上方  NET\* p = new NET;  p->x = vertice[j].x;  p->ymax = vertice[(j + 1) % len].y;  float DX = vertice[(j + 1) % len].x - vertice[j].x;  float DY = vertice[(j + 1) % len].y - vertice[j].y;  p->dx = DX / DY;//dx为直线斜率的倒数  p->next = pNET[i]->next;  pNET[i]->next = p;  }  }  }  }  //建立并且更新活性边表AET  //各条扫描线i  for (int i = 0; i <= Max\_Y; i++)  {  /\*把新边表NET[i] 中的边结点用插入排序法插入AET表，使之按x坐标递增顺序排列\*/  //计算每条扫描线上不同线产生的新的交点x，更新AET  NET\* p = pAET->next;  while (p)  {  p->x = p->x + p->dx; //更新x坐标  p = p->next;  }  //断表排序,不再开辟空间  AET\* tq = pAET;  p = pAET->next;  tq->next = NULL;  while (p)//顺着链表往下走  {  //找到第一个比它大的数字tq->next->next->x，则从p->next到tq->next都是比p->x小的  while (tq->next != NULL && tq->next->x <= p->x)  tq = tq->next;  //插入p到tq和tq->next之间  NET\* t = p->next;  p->next = tq->next;  tq->next = p;  p = t;  tq = pAET;//回到头  }  /\*(改进算法) 取消求交，减少计算量\*/  //先从AET表中删除ymax==i的结点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //像素的取舍问题，保证多边形的“下闭上开”，避免填充扩大化（交点的个数应保证为偶数个）  AET\* q = pAET;  p = q->next;  while (p)  {  if (p->ymax == i)  {  q->next = p->next;  delete p;  p = q->next;  }  else  {  q = q->next;  p = q->next;  }  }  //若NET中有新点，将其用插入法插入AET，按x递增的顺序排列  p = pNET[i]->next;  q = pAET;  while (p)  {  while (q->next && p->x >= q->next->x)  q = q->next;  //插入p  NET\* t = p->next;  p->next = q->next;  q->next = p;  p = t;  q = pAET;//回到头  }  //配对后填充颜色  p = pAET->next;  while (p && p->next != NULL)  {  for (float j = p->x; j <= p->next->x; j++)  {  //扫描线画点  draw\_a\_point(j, i);  //cout << "(" << j << ", " << i << ")" << endl;  }  p = p->next->next;//考虑端点情况  }  }  glFlush();  }  void mymouse(int button, int state, int x, int y)  {  //左键  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 << ", " << window\_height - y << ")" << endl;  }  //右键  if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)  {  glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色  glColor3f(0, 1, 1);  //绘制多边形  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();  }  //鼠标中间  if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)  {  //cout << "center: (" << x << ", " << y << ")" << endl;  //BoundaryFill4(x, window\_height - y);  //BoundaryFill4\_Stack(x, window\_height - y);  cout << "多边形顶点个数为" << vertice.size() << "。 " << "开始扫描线填充。" << endl;  PolyScan();  }  }  void display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0, 0.4, 0.2);  glPointSize(1);  glBegin(GL\_POINTS);  PolyScan();  glEnd();  glFlush();  } |

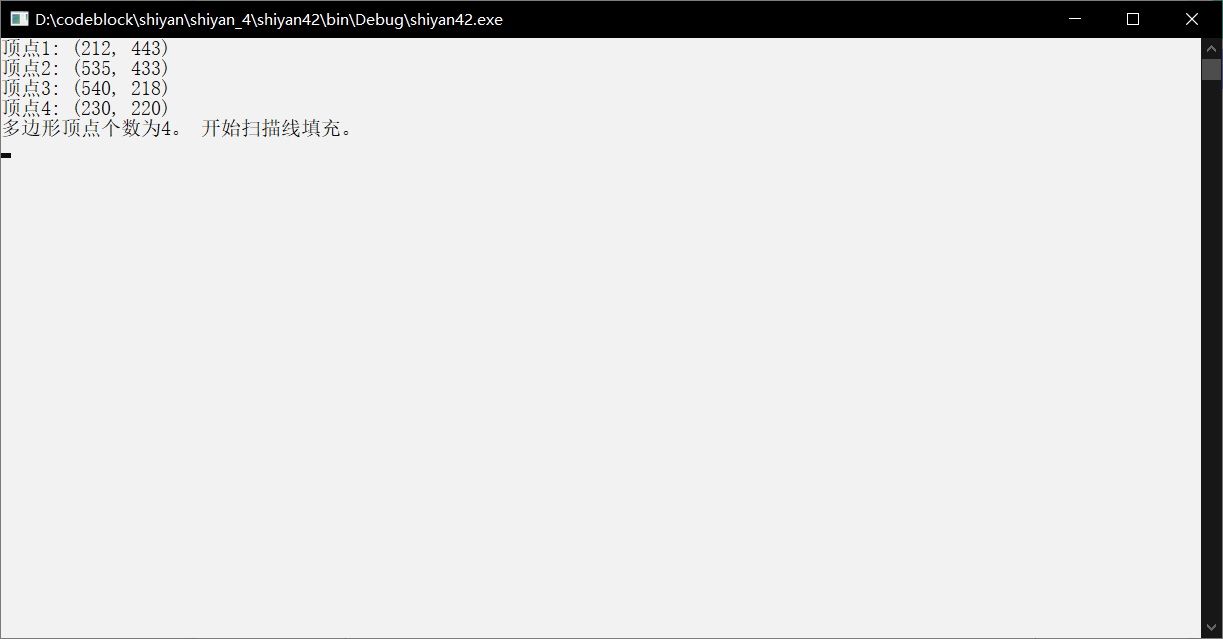
4、实验结果：

教材P66，填充六边形



验证扫描线填充，任意的多边形边数





实验五 圆扫描转换和种子点填充实验

时间：2022.4.13

地点：信息学院2202

1. 实验内容：

1、圆扫描转换

2、实验目的：

1、输入圆的半径，画出圆

3、实验代码：

|  |
| --- |
| #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, 1.0, 2.0); // 蓝色  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;  } |

种子点填充：

#include <GL/glut.h>

#include <fstream>

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(75, 125);

glVertex2f(125, 110);

glEnd();

}

void display(void) {

Color fillColor = { 1.0, 1.0, 0.0 };//填充颜色

Color boarderColor = { 0.0, 0.0, 1.0 };//边界颜色 glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0, 0, 500, 500);

glColor3fv(boarderColor);

polygon();

glColor3fv(fillColor);

//boundaryFill4(150, 150, 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(300, 300);

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);

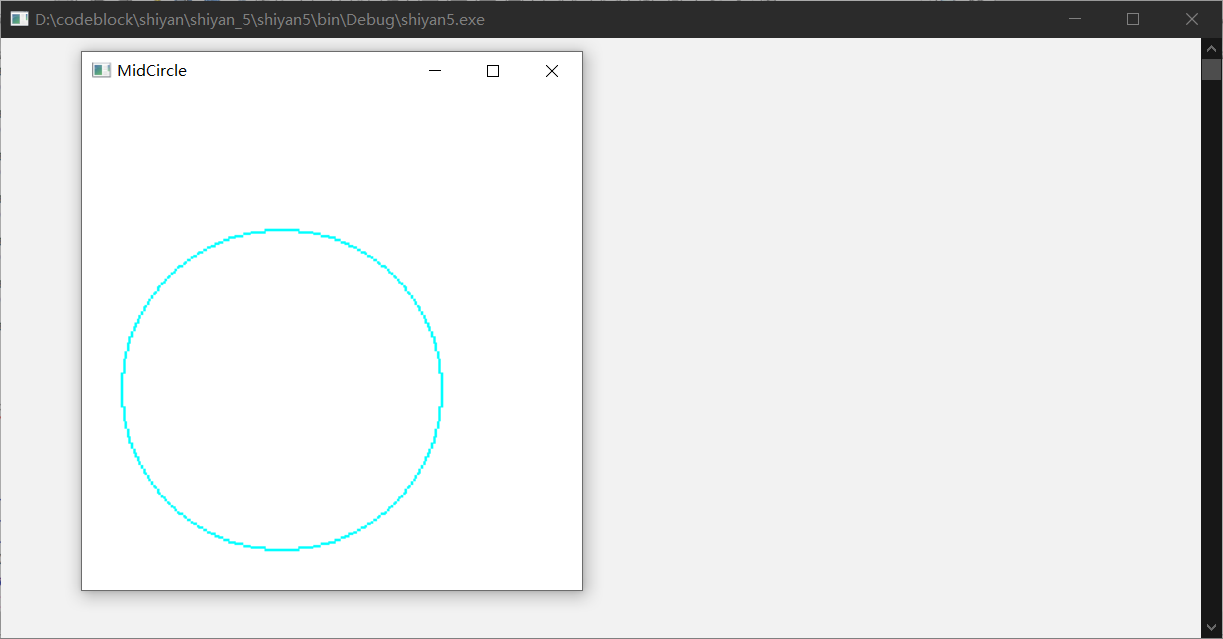
glutMainLoop();

return 0;

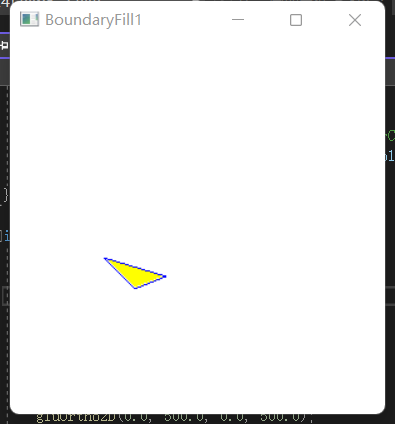
}

4、实验结果：

扫描转换：



种子点填充：



实验六 二维几何变换

时间：2022.4.20

地点：信息学院2202

1. 实验内容：

教材P161，二维几何变换算法

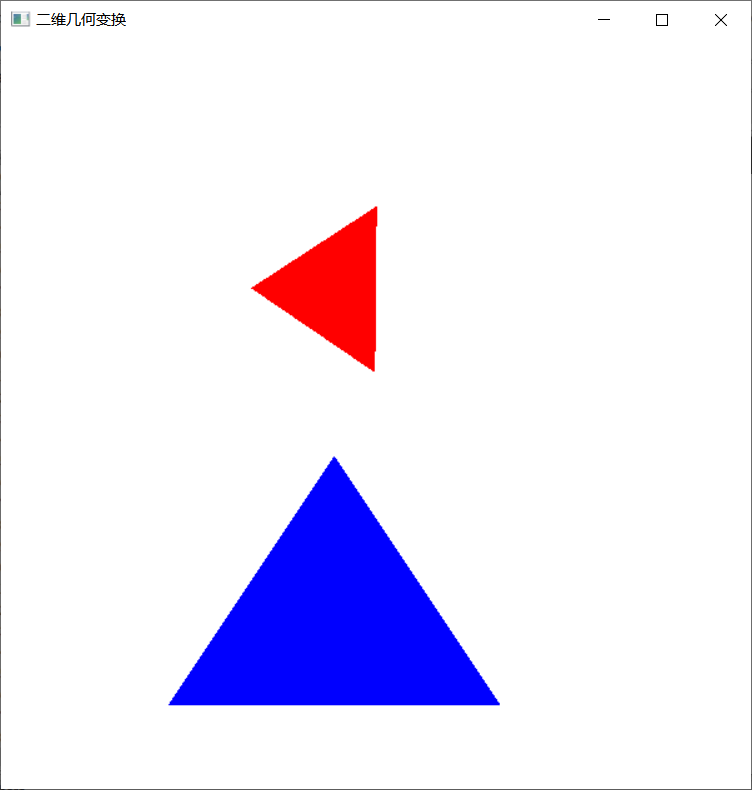
2、实验目的：

验证二维几何变换，熟悉变换矩阵

3、实验代码：

|  |
| --- |
| #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 = 3.12159;  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(void) {  GLint nVerts = 3;  wcPt2D verts[3] = { {50.0,25.0},{150.0,25.0},{100.0,100.0} };  wcPt2D centroidPt;  GLint k, xsun = 0, ysum = 0;  for (k = 0; k < nVerts; k++) {  xsun += verts[k].x;  ysum += verts[k].y;  }  centroidPt.x = GLfloat(xsun) / 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("xxxxxxx");  init();  glutDisplayFunc(displayFcn);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

1. 实验结果：



实验七 反走样技术

时间：2022年4月26日

地点：信息学院2202机房

1、实验内容：

教材P458，GLUT鼠标函数；

使用opengl，实现任意反走样技术。

2、实验目的：

调用鼠标函数完成相应功能，2-3个程序；

了解反走样技术并实现。

3、实验代码：

鼠标函数（1）：

#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);

}

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();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Mouse Plot Points");

init();

glutDisplayFunc(displayFcn);

glutReshapeFunc(winReshapeFcn);

glutMouseFunc(mousePtPlot);

glutMainLoop();

return 0;

}

鼠标函数（2）：

#include<gl/glut.h>

#include<stdlib.h>

GLsizei winWidth = 400, winHeight = 300;

GLint ptCtr = 0;

class scrPt {

public:

GLint x, y;

};

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);

}

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 drawLineSegment(scrPt endPt1, scrPt endPt2)

{

glBegin(GL\_LINES);

glVertex2i(endPt1.x, endPt1.y);

glVertex2i(endPt2.x, endPt2.y);

glEnd();

}

void polyline(GLint button, GLint action, GLint xMouse, GLint yMouse)

{

static scrPt endPt1, endPt2;

if (ptCtr == 0) {

if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

endPt1.x = xMouse;

endPt1.y = winHeight - yMouse;

ptCtr = 1;

}

}

else

if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

endPt2.x = xMouse;

endPt2.y = winHeight - yMouse;

drawLineSegment(endPt1, endPt2);

endPt1 = endPt2;

}

else

if (button == GLUT\_RIGHT\_BUTTON)

exit(0);

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Draw Interactive Ployline");

init();

glutDisplayFunc(displayFcn);

glutReshapeFunc(winReshapeFcn);

glutMouseFunc(polyline);

glutMainLoop();

return 0;

}

反走样：

#include <gl/glut.h>

#pragma comment(lib, "glut32.lib")

#define NO 0

#define YES 1

int Drawing;

void Initialization()

{

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_DST\_ALPHA);

glEnable(GL\_POINT\_SMOOTH); //启用点反走样

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

glEnable(GL\_LINE\_SMOOTH); //启用直线反走样，初始化函数中所调用的那3条命令才真正起作用

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);

Drawing = NO;

}

void OnDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

if (Drawing == YES) //启用混合状态，初始化函数中所调用的那3条命令才真正起作用

{

glEnable(GL\_BLEND);

}

else //关闭混合状态

{

glDisable(GL\_BLEND);

}

glColor4f(0.0f, 1.0f, 0.0f, 1.0f); //当前绘图色为绿色

glLineWidth(10);

glBegin(GL\_LINE\_STRIP);

{

glVertex2f(-3.0f, -0.5f);

glVertex2f(0.0f, 0.5f);

glVertex2f(3.0f, -0.5f);

}

glEnd();

glPointSize(10);

glBegin(GL\_POINTS);

{

glVertex2f(0.0f, 1.0f);

}

glEnd();

glutSwapBuffers();

}

void OnReShape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //将当前矩阵指定为投影模式

glLoadIdentity();

if (h != 0)

{

GLfloat aspect = (float)w / (float)h;

if (w < h)

{

gluOrtho2D(-3, 3, -3 \* aspect, 3 \* aspect);

}

else

{

gluOrtho2D(-3 / aspect, 3 / aspect, -3, 3);

}

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void CreateMenu(void)

{

glutAddMenuEntry("正常显示", NO);

glutAddMenuEntry("反走样", YES);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void OnMenu(int value)

{

Drawing = value;

glutPostRedisplay();

}

void main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(300, 300);

glutCreateWindow("OpenGL反走样");

glutReshapeFunc(OnReShape);

glutDisplayFunc(OnDisplay);

glutCreateMenu(OnMenu);

Initialization();

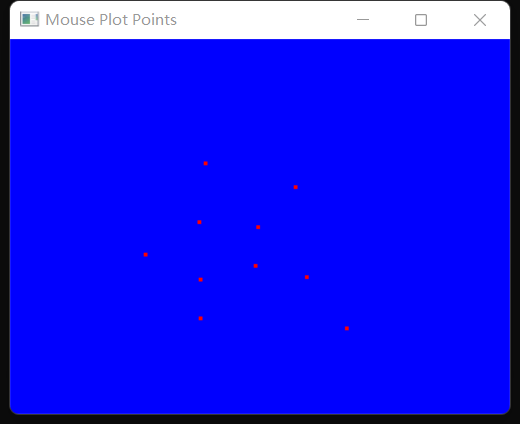
CreateMenu();

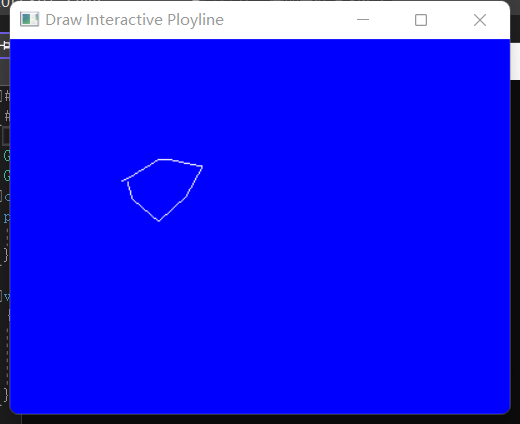
glutMainLoop();

}

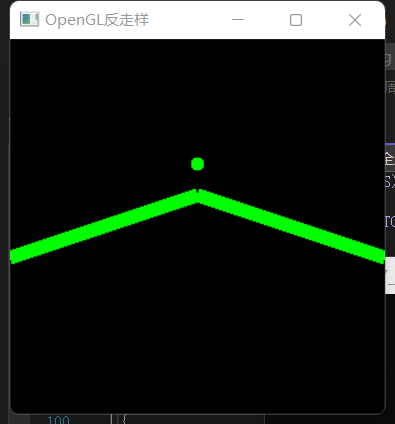
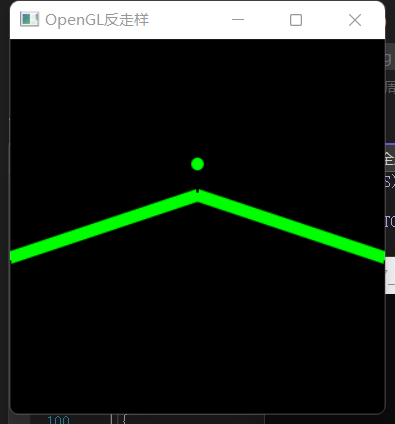
4、实验结果：

鼠标函数（1）：

鼠标函数（2）：



反走样：

  前 后

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：

1、使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪

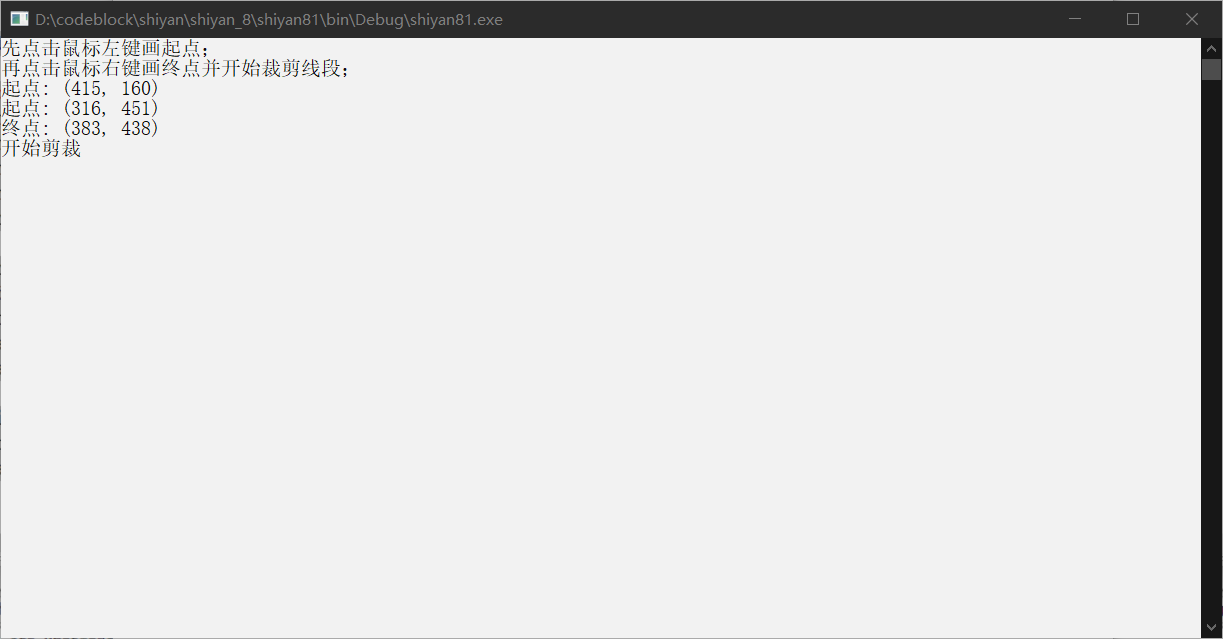
2、实验目的：

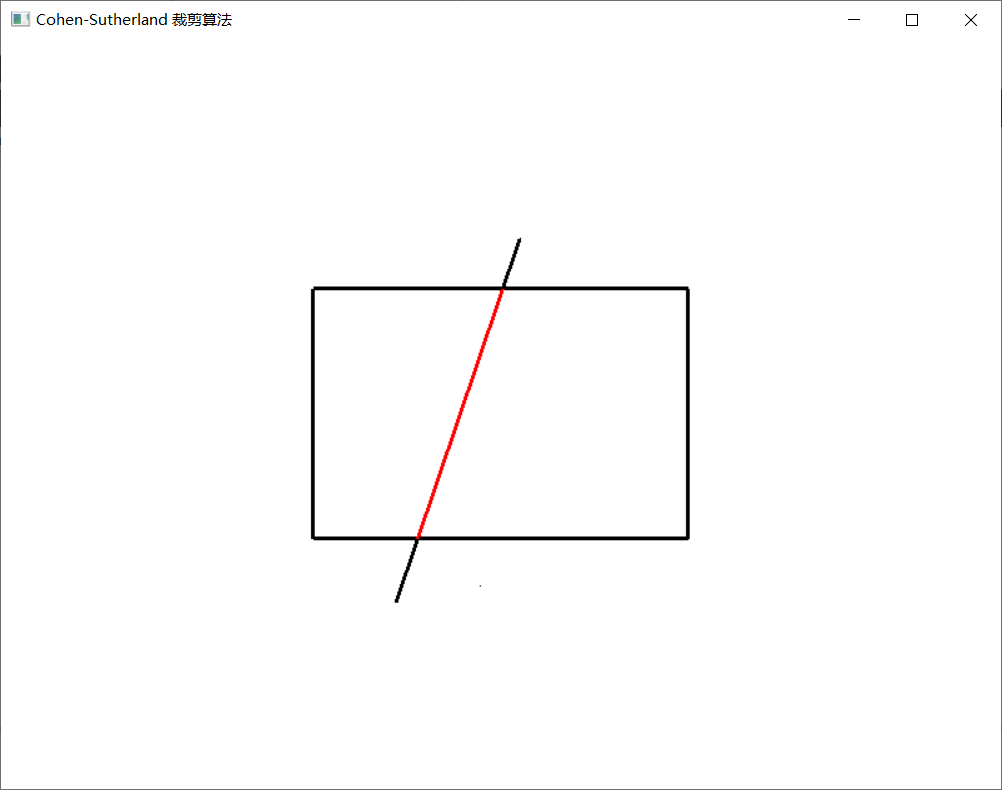
验证Cohen-Sutherland裁剪算法，从键盘输入任意的直线段，用指定的裁剪窗口裁剪直线段。

3、实验代码：

|  |
| --- |
| #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();  //vertice.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);//set the display mode  glutInitWindowSize(screenWidth, screenHeight); //set the window size  glutInitWindowPosition(windowPositionX, windowPositionY); //set 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、实验结果：





实验九 三维图形几何变换

时间：2022年5月11日

地点：信息学院2202机房

1、实验内容：

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。

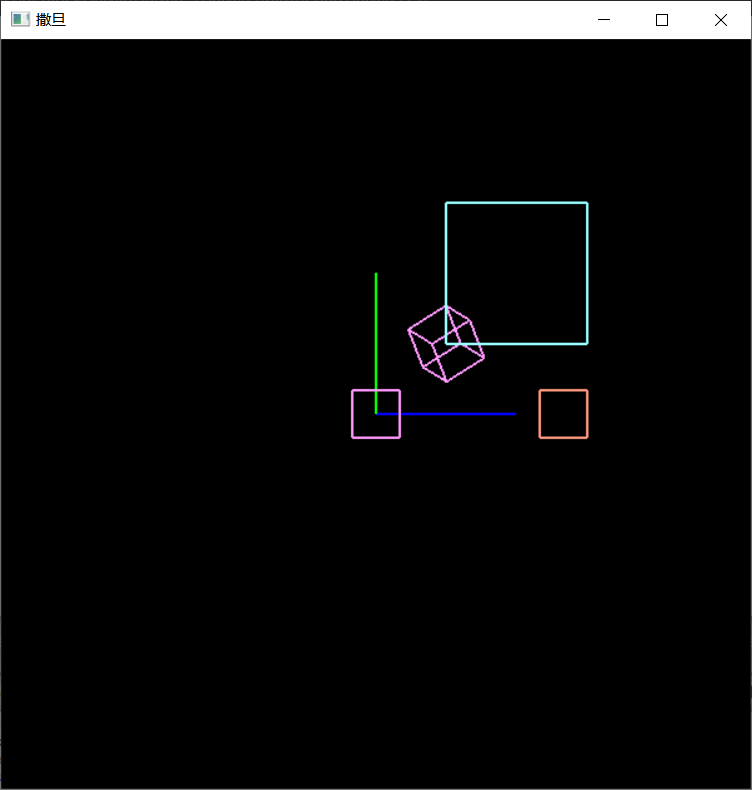
2、实验目的：

调用函数完成三维图形几何变换。

3、实验代码：

|  |
| --- |
| #include <windows.h>  #pragma once  #define GLUT\_DISABLE\_ATEXIT\_HACK //防止GLUT版本问题出错  #include <GL/glut.h>  int cx = 0, cy = 0, cz = 0;  float m\_xRotate = 0, m\_yRotate = 0, m\_zRotate = 0;  int spin = 0;  void Init()  {  glClearColor(0.0, 0.0, 0.0, 0.0); //设置背景颜色为黑色  glShadeModel(GL\_FLAT);  // glShadeModel选择平坦或光滑渐变模式。GL\_SMOOTH为缺省值，为光滑渐变模式，GL\_FLAT为平坦渐变模式。  }  void DrawCoordinate()  {  glLineWidth(2);  glBegin(GL\_LINES); //开始画线  //绘制一条从原点到x=300的蓝色线段  glColor3f(0.0, 0.0, 1.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(300.0, 0.0, 0.0);  //绘制一条从原点到y=300的绿色线段  glColor3f(0.0, 1.0, 0.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(0.0, 300.0, 0.0);  //绘制一条从原点到z=300的红色线段  glColor3f(1.0, 0.0, 0.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(0.0, 0.0, 300.0);  }  void DrawCube()  {  glClearColor(0.0, 0.0, 0.0, 0.0); //设置背景颜色为黑色  glClear(GL\_COLOR\_BUFFER\_BIT); //用背景颜色替换原有颜色  glLineWidth(2);  glBegin(GL\_LINES); //开始画线  //绘制一条从原点到x=300的蓝色线段  glColor3f(0.0, 0.0, 1.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(300.0, 0.0, 0.0);  //绘制一条从原点到y=300的绿色线段  glColor3f(0.0, 1.0, 0.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(0.0, 300.0, 0.0);  //绘制一条从原点到z=300的红色线段  glColor3f(1.0, 0.0, 0.0);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(0.0, 0.0, 300.0);  glEnd(); //结束绘制  glColor3f(1.0, 0.6, 1.0); //设置颜色为白色  glutWireCube(100.0f); //绘制一个边长为100个单位的线框立方体  glMatrixMode(GL\_MODELVIEW);  //-----在这里填写变换函数，分别实现平移变换、旋转变换和缩放变换  //1、平移  glPushMatrix();  glTranslatef(400.0, 0.0, 0.0); //沿X方向平移400个单位  glColor3f(1.0, 0.6, 0.5); //设置颜色为土色  glutWireCube(100.0f); //绘制一个边长为100个单位的线框立方体  glPopMatrix();  //2、旋转  glPushMatrix();  glTranslatef(150.0, 150.0, 150.0); //沿坐标系对角线方向平移150个单位  glRotatef(45, 1.0, 1.0, 1.0); //基于对角线方向旋转45度  glColor3f(1.0, 0.6, 1.0); //设置颜色为粉色  glutWireCube(100.0f); //绘制一个边长为100个单位的线框立方体  glPopMatrix();  //3、缩放  glPushMatrix();  glTranslatef(300.0, 300.0, 300.0); //沿坐标系对角线方向平移300个单位  glScalef(3.0, 3.0, 3.0); //对整体放大3倍  glColor3f(0.6, 1.6, 1.0); //设置颜色为青色  glutWireCube(100.0f); //绘制一个边长为100个单位的线框立方体  glPopMatrix();  //-----在这里填写变换函数，分别实现平移变换、旋转变换和缩放变换  glFlush();  }  void Reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  /\*if (w <= h)  {  glOrtho(-800.0, 800, -800.0 \* h/w, 800 \* h/w, -800, 800);  }  else  {  glOrtho(-800.0 \* w/h, 800 \* w/h, -800.0, 800, -800, 800);  }\*/  glOrtho(-800.0 \* w / h, 800 \* w / h, -800.0, 800, -800, 800);  }  void myMouseMotion(GLint x, GLint y)  {  m\_xRotate = cx - x;  m\_yRotate = cy - y;  cx = x, cy = y;  glMatrixMode(GL\_MODELVIEW);  glRotatef(m\_xRotate, 1.0, 0.0, 0.0);  glRotatef(m\_yRotate, 0.0, 1.0, 0.0);  glutPostRedisplay();  }  void mykeyboard(unsigned char key, int x, int y)  {  switch (key) {  case'd':  spin = spin + 90;  glMatrixMode(GL\_MODELVIEW);  glRotatef(spin, 0.0, 1.0, 0.0);  glutPostRedisplay();  break;  case'a':  spin = spin - 90;  glMatrixMode(GL\_MODELVIEW);  glRotatef(spin, 0.0, 1.0, 0.0);  glutPostRedisplay();  break;  case'w':  spin = spin + 90;  glMatrixMode(GL\_MODELVIEW);  glRotatef(spin, 1.0, 0.0, 0.0);  glutPostRedisplay();  break;  case's':  spin = spin - 90;  glMatrixMode(GL\_MODELVIEW);  glRotatef(spin, 1.0, 0.0, 0.0);  glutPostRedisplay();  break;  case'r':  glLoadIdentity();  glutPostRedisplay();  break;  case 27:  exit(0);  break;  }  }  int main(int argc, char\* argv[])  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE); //初始化颜色模式为RGB  glutInitWindowPosition(100, 100);  glutInitWindowSize(600, 600);  glutCreateWindow("撒旦");  Init();  glutDisplayFunc(DrawCube);  glutReshapeFunc(Reshape);  glutMotionFunc(myMouseMotion);  glutKeyboardFunc(mykeyboard);  glutMainLoop();  return 0;  } |

4、运行结果：



实验十 着色及建模实验

时间：2022年5月17日

地点：信息学院2202机房

1、实验内容：

使用opengl，片元着色器着色，P523

2、实验目的：

验证片元着色器算法，获得着色结果。

3、实验代码：略（无法运行）

4、运行结果：略

实验十一 交互控制实验

时间：2022年5月24日

地点：信息学院2202机房

1、实验内容：

使用opengl，完成鼠标、键盘交互操作

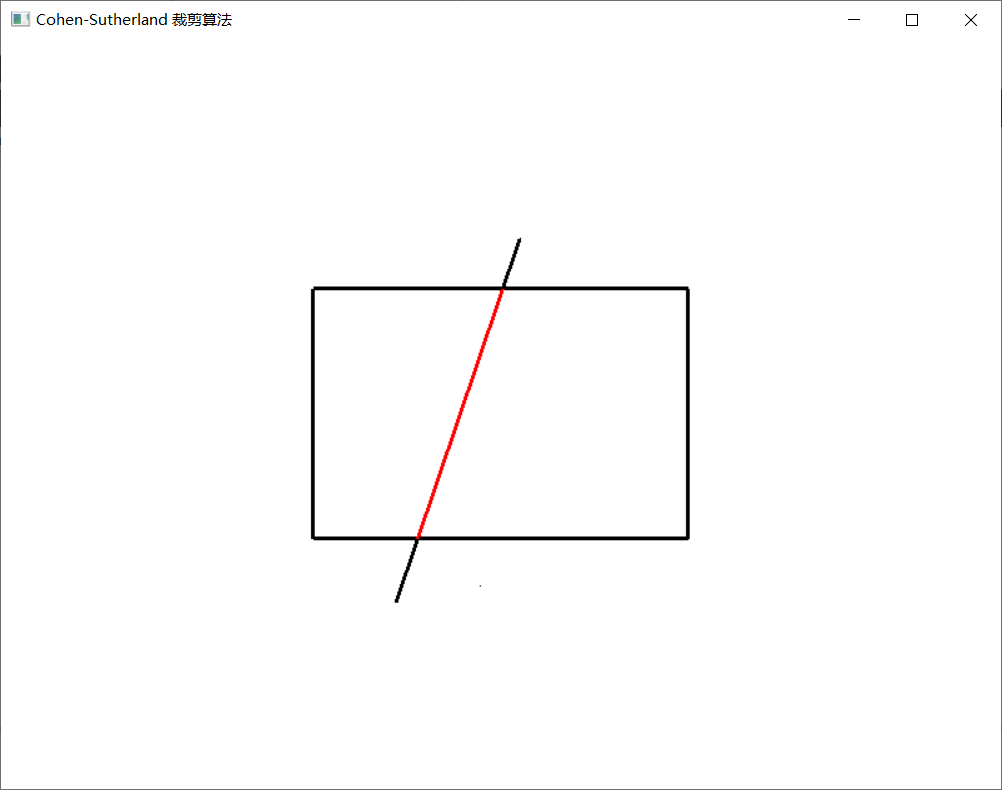
2、实验目的：

熟悉鼠标、键盘交互

3、实验代码：

|  |
| --- |
| #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();  //vertice.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);//set the display mode  glutInitWindowSize(screenWidth, screenHeight); //set the window size  glutInitWindowPosition(windowPositionX, windowPositionY); //set 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、运行结果：



实验十二 三维观察实验

时间：2022年5月25日

地点：信息学院2202机房

1、实验内容：

1、使用opengl，完成投影变换等实验

2、P 264

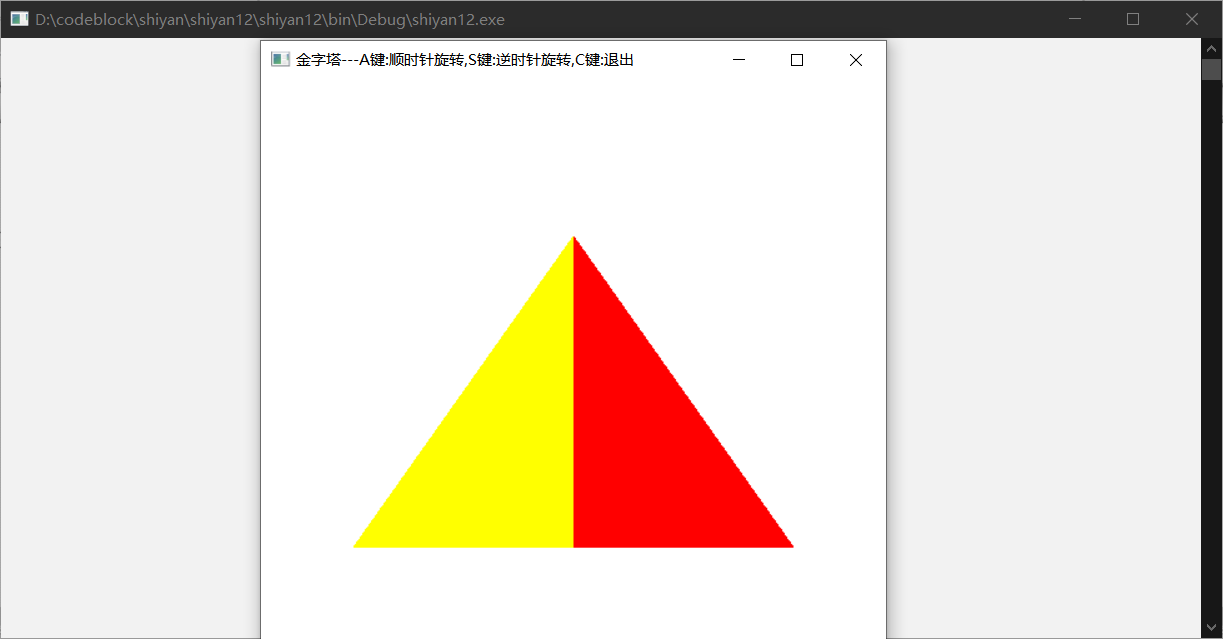
2、实验目的：

熟悉三维观察相关内容

3、实验代码：

|  |
| --- |
| #include <windows.h>  #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();  } |

4、运行结果：



实验十三 多面体实验

时间：2022年6月1日

地点：信息学院2202机房

1、实验内容：

生成多面体线框图，P300、P307

2、实验目的：

熟悉三维线框图相关内容

3、实验代码：

P300

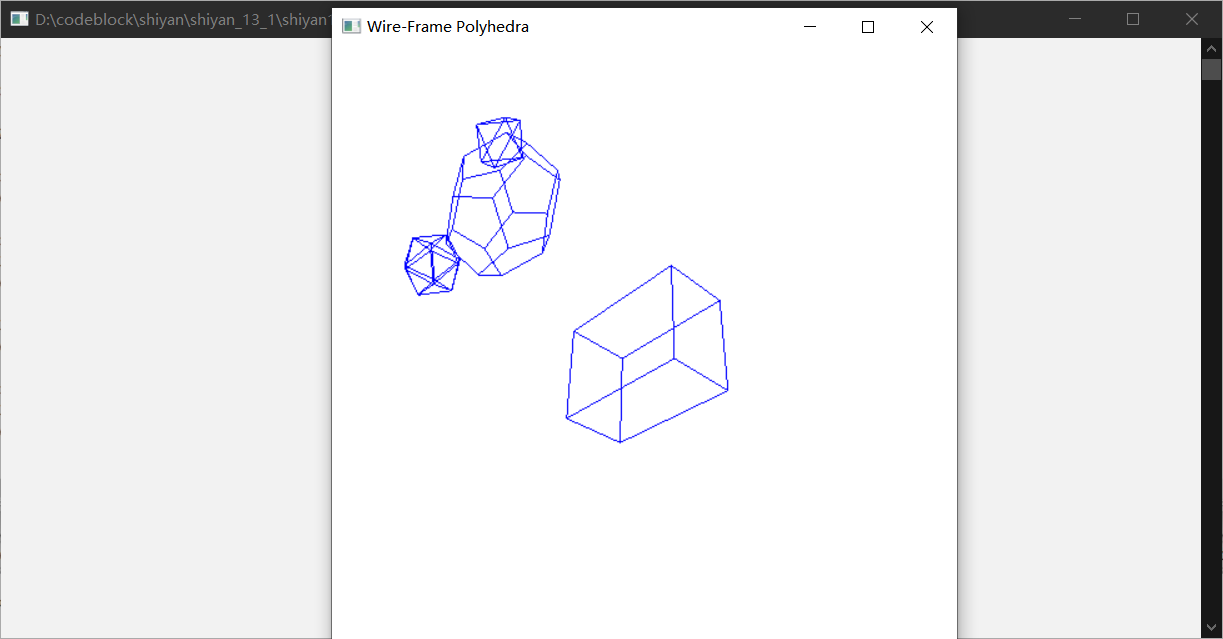
|  |
| --- |
| #include <windows.h>  #include <GL/glut.h>  GLsizei winWidth = 500,winHeight = 500;  void init(void){  glClearColor(1.0,1.0,1.0,0.0);  }  void displayWirePolyhedra(void){  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0,0.0,1.0);  gluLookAt(5.0,5.0,5.0,0.0,0.0,0.0,0.0,1.0,1.0);  glScalef(1.5,2.0,1.0);  glutWireCube(1.0);  glScalef(0.8,0.5,0.8);  glTranslatef(-6.0,-5.0,0.0);  glutWireDodecahedron();  glTranslatef(-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("Wire-Frame Polyhedra");  init();  glutDisplayFunc(displayWirePolyhedra);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

P307

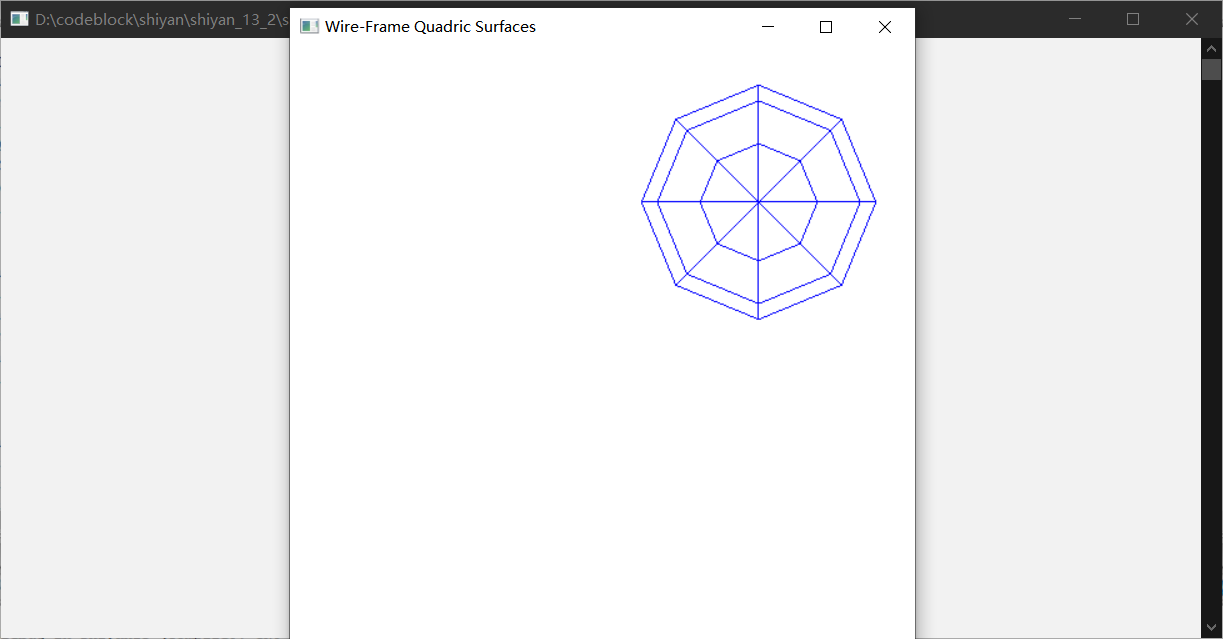
|  |
| --- |
| #include <windows.h>  #include <GL/glut.h>  GLsizei winWidth = 500,winHeight = 500;  void init(void){  glClearColor(1.0,1.0,1.0,0.0);  }  void wireQuadSurfs(void){  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3f(0.0,0.0,1.0);  glPushMatrix();  glTranslatef(1.0,1.0,0.0);  glutWireSphere(0.75,8,6);  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("Wire-Frame Quadric Surfaces");  init();  glutDisplayFunc(wireQuadSurfs);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

4、运行结果：

P300:



P307:



实验十四 曲线曲面生成实验

时间：2022年6月7日

地点：信息学院2202机房

1、实验内容：

生成曲线或者曲面，P323

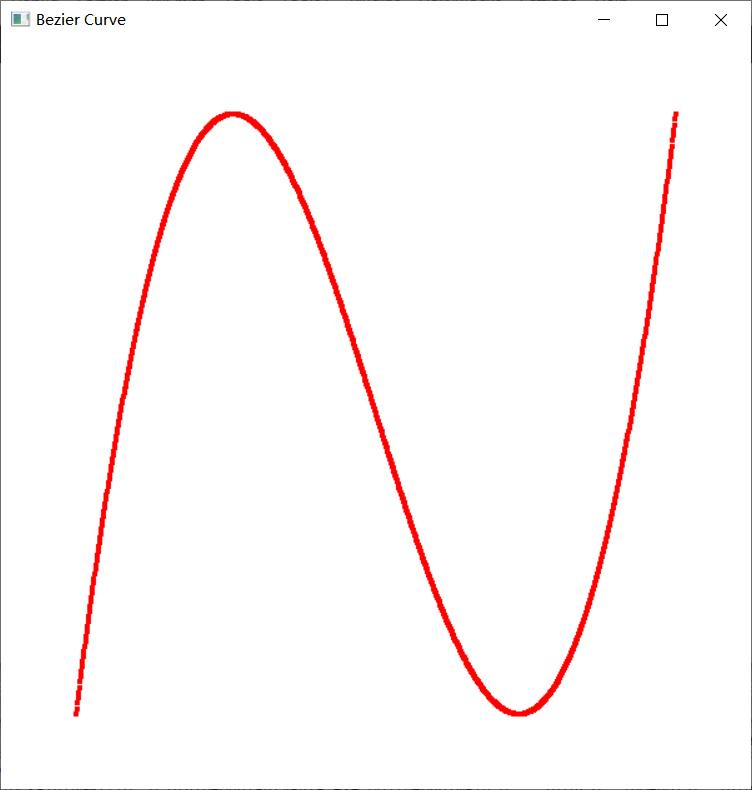
2、实验目的：

熟悉Bezier、样条等相关内容

3、实验代码：

|  |
| --- |
| #include<stdlib.h>  #include<math.h>  #include<windows.h>  #define GLUT\_DISABLE\_ATEXIT\_HACK  #include<gl/glut.h>  GLsizei winWidth = 600, winHeight = 600;  GLfloat xwcMin = -50.0, xwcMax = 50.0;  GLfloat 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 bezCurvePt)  {  glBegin(GL\_POINTS);  glVertex2f(bezCurvePt.x, bezCurvePt.y);  glEnd();  }  void binomialCoeffs(GLint n, GLint\* C)  {  GLint k, j;  for (k = 0; k <= n; k++) {  C[k] = 1;  for (j = n; j >= k + 1; j--)  C[k] \*= j;  for (j = n - k; j >= 2; j--)  C[k] /= j;  }  }  void computeBezPt(GLfloat u, wcPt3D\* bezPt, GLint nCtrlPts, wcPt3D\* ctrlPts, GLint\* C)  {  GLint k, n = nCtrlPts - 1;  GLfloat bezBlendFcn;  bezPt->x = bezPt->y = bezPt->z = 0.0;  for (k = 0; k < nCtrlPts; k++) {  bezBlendFcn = C[k] \* pow(u, k) \* pow(1 - u, n - k);  bezPt->x += ctrlPts[k].x \* bezBlendFcn;  bezPt->y += ctrlPts[k].y \* bezBlendFcn;  bezPt->z += ctrlPts[k].z \* bezBlendFcn;  }  }  void bezier(wcPt3D\* ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)  {  wcPt3D bezCurvePt;  GLfloat u;  GLint\* C, k;  C = new GLint[nCtrlPts];  binomialCoeffs(nCtrlPts - 1, C);  for (k = 0; k <= nBezCurvePts; k++) {  u = GLfloat(k) / GLfloat(nBezCurvePts);  computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);  plotPoint(bezCurvePt);  }  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, newHeight, 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("Bezier Curve");  init();  glutDisplayFunc(displayFcn);  glutReshapeFunc(winReshapeFcn);  glutMainLoop();  } |

4、运行结果：



实验十五 消隐实验

时间：2022年6月15日

地点：信息学院2202机房

1、实验内容：

完成消隐实验，采用Z-buffer算法完成消隐。

2、实验目的：

熟悉Z-buffer、画家算法等相关内容。

3、实验代码：

|  |
| --- |
| #include <windows.h>  #include <gl/glut.h>  void Initial()  {  glEnable(GL\_DEPTH\_TEST);  glFrontFace(GL\_CW);  glClearColor(1.0, 1.0, 1.0, 0.0);  }  void ChangeSize(int w, int h)  {  if (h == 0) h = 1;  glViewport(0, 0, w, h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  if (w <= h)  glOrtho(-4.0f, 4.0f, -4.0f \* h / w, 4.0f \* h / w, -4.0f, 4.0f);  else  glOrtho(-4.0f \* w / h, 4.0f \* w / h, -4.0f, 4.0f, -4.0f, 4.0f);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  void Display(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glColor3f(1.0, 0.0, 0.0);  glPushMatrix();  /\* 第一个茶壶使用了剔除\*/  glEnable(GL\_CULL\_FACE);  glCullFace(GL\_BACK); //剔除茶壶的后向面  glTranslatef(-2.0f, 0.0f, 0.0f);  glRotatef(180.0f, 0.0f, 1.0f, 0.0f);  GLdouble equ[4] = { -1.0f, 2.3f, 2.3f, 2.3f }; // equ中保存平面方程的系数  glClipPlane(GL\_CLIP\_PLANE0, equ); //glClipPlane定义裁减平面  glEnable(GL\_CLIP\_PLANE0);  glutSolidTeapot(1.0);  glPopMatrix();  /\* 第二个茶壶关闭了剔除操作\*/  glDisable(GL\_CULL\_FACE);  glTranslatef(2.0f, 0.0f, 0.0f);  glRotatef(180.0f, 0.0f, 1.0f, 0.0f);  glClipPlane(GL\_CLIP\_PLANE0, equ);  glEnable(GL\_CLIP\_PLANE0);  glutSolidTeapot(1.0);  glPopMatrix();  glDisable(GL\_CLIP\_PLANE0);  glFlush();  }  int main(int argc, char\* argv[])  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(400, 400);  glutCreateWindow("茶壶的剔除操作");  glutReshapeFunc(ChangeSize);  glutDisplayFunc(Display);  Initial();  glutMainLoop();  return 0;  } |

4、运行结果：

