吉林大学

软件学院

《图形学与人机交互》

实验报告

班级： 6

学号：55210629

姓名：张耕儒

2023-2024学年第1学期

|  |  |  |  |
| --- | --- | --- | --- |
| 实验项目 | 图形学算法实现与演示 | | |
| 实验性质 | 🗹演示性实验 □验证性实验  🗹操作性实验 □综合性实验 | | |
| 实验地点 | 王湘浩楼A110 | 机器编号 |  |
| 一、系统实现的功能  实现了图形绘制中的：1.绘制矩形 2.绘制圆形 3.设置颜色  区域填充中的：1.绘制多边形 2.设置颜色 3.设置填充颜色  绘制曲线中的：1.绘制Bezier曲线。  三维变换中的：1.绘制立方体。2.沿X、Y、Z轴平移。  3.绕X、Y、Z轴旋转。4.设置平移步长及旋转角度。  二、 实现的图形学算法  对于所有绘制类，我采用先写一个基类DrawMode，在DrawMode中定义了点的结构体，存储点的容器，以及绘制橡皮线，绘制，更新，判断结束的函数。  struct Point  {  Point(int x, int y) :x(x), y(y) {}  Point() { x = 0, y = 0; }  int x;  int y;  };  class DrawMode  {  public:  virtual void drag(CDC\* pDC, Point oldPoint, Point newPoint, COLORREF color);  virtual void draw(CDC\* pDC, Point startPoint, Point endPoint, COLORREF color)=0;  virtual void updata(CDC\* pDC, Point point, COLORREF color);  virtual bool isOver();//判断是否结束  //记录鼠标左键点击时的坐标  vector<Point> points;  };   * 绘制直线采用了DDA直线绘制算法。 * 矩形中通过两点，分别绘制上下，左右两条边。 * 绘制圆形采用了Bersenham画圆。   void DrawCircle::draw(CDC\* pDC, Point startPoint, Point endPoint, COLORREF color)  {  //Bersenham画圆  int dx = endPoint.x - startPoint.x;  int dy = endPoint.y - startPoint.y;  double R;  R = sqrt(dx \* dx + dy \* dy) / 2;  circleX = (startPoint.x + endPoint.x) / 2;  circleY = (startPoint.y + endPoint.y) / 2;  int x = 0, y = R;  int p = 3 - 2 \* R;  for (; x <= y; x++)  {  pDC->SetPixel(circleX + x, circleY + y, color);  pDC->SetPixel(circleX + x, circleY - y, color);  pDC->SetPixel(circleX - x, circleY + y, color);  pDC->SetPixel(circleX - x, circleY - y, color);  //........................................................  pDC->SetPixel(circleX + y, circleY + x, color);  pDC->SetPixel(circleX + y, circleY - x, color);  pDC->SetPixel(circleX - y, circleY + x, color);  pDC->SetPixel(circleX - y, circleY - x, color);  if (p >= 0)  {  p += 4 \* (x - y) + 10;  y--;  }  else  p += 4 \* x + 6;  }  }   * 多边形的颜色填充实现了两种算法，首先实现的是判断一个点是否在多边形内，如果在，即绘制。但是实现后发现该算法效率较低。然后实现了边标志算法及栅栏改进。在实现过程中，利用unordered\_map实现MASK矩阵。   算法1：  首先用最大x,y, 最小x,y,求出一个最小覆盖的矩形，然后用int inner(Point p);函数判断这个矩形中的点是否在多边形中，如果在，就setPixel。  for (int i = 0; i < RemainderPoint.size(); i++)  if(inner(RemainderPoint[i])&&zgr[RemainderPoint[i].y%10][RemainderPoint[i].x%20]==1)  pDC->SetPixel(RemainderPoint[i].x, RemainderPoint[i].y, fillcolor);  算法2：  实现边标志算法及栅栏改进。算法中的MASK利用unordered\_map实现，重写hash\_name 和 equal\_point是为了能把Point类作为map的键所重写的哈希函数及判等函数。  unordered\_map<Point, bool, hash\_name, equal\_point>MASK;  for (int y = minPoint.y; y <maxPoint.y; y++)  {  bool inside = false;  for (int x = minPoint.x; x < maxPoint.x; x++)  {  if (MASK[Point(x, y)])  inside = !inside;  if (inside&&zgr[y%10][x%20])  {  if (y == minPoint.y)//如果是最上面的一行，不绘制  continue;  pDC->SetPixel(x, y, fillcolor);  }  }  }   * 绘制Bezier曲线采用分裂算法。其中点的容器用vector实现。 * 绘制三维立体图形时，首先将坐标系原点平移到窗口中心，x正方向向右，y正方向向上，-z方向向屏幕内。用vector实现了点表和边表，另外定义了一个seenPoints用来存储经过坐标变换后的点表。   for (int i = 0; i < 8; i++)  {  seenPoints[i].set(watchPoint.getx() + d \* (points[i].getx() - watchPoint.getx()) / (points[i].getz() + d),  watchPoint.gety() + d \* (points[i].gety() - watchPoint.gety()) / (points[i].getz() + d), 0);  }  旋转和平移时用原始点表乘上平移/旋转矩阵，然后再进行透视投影变换。以绕x周旋转为例：  for (int i = 0; i < 8; i++)  {  y = this->points[i].gety();  z = this->points[i].getz();  this->points[i].setY(y \* cos(radian) - z \* sin(radian));  this->points[i].setZ(y \* sin(radian) + z \* cos(radian));  }  this->draw(pDC);  三、采用的交互方式  1.鼠标左键按下，记录点的位置。  void CCGWORK0629View::OnLButtonDown(UINT nFlags, CPoint point)  {  // TODO: 在此添加消息处理程序代码和/或调用默认值  drawmode->updata(this->GetDC(), Point(point.x, point.y), currColor());  CView::OnLButtonDown(nFlags, point);  }  2.鼠标右键按下，实现多边形填充。  void CCGWORK0629View::OnRButtonDown(UINT nFlags, CPoint point)  {  // TODO: 在此添加消息处理程序代码和/或调用默认值  Draw2dDrawpoly tmp;  Draw2dDrawpoly\* p;  type\_index tmp\_type = type\_index(typeid(tmp));  type\_index dm\_type = type\_index(typeid(\*this->drawmode));  CDC\* pDC = this->GetDC();  if (tmp\_type == dm\_type)  {  p = (Draw2dDrawpoly\*)this->drawmode;  p->finish(pDC, currFillColor());  }  CView::OnRButtonDown(nFlags, point);  }  3.鼠标移动，绘制橡皮线。  void CCGWORK0629View::OnMouseMove(UINT nFlags, CPoint point)  {  // TODO: 在此添加消息处理程序代码和/或调用默认值  if (!drawmode->isOver())  {  CDC\* pDC = this->GetDC();  drawmode->drag(pDC, drag, Point(point.x, point.y),currColor());  }  drag.x = point.x;  drag.y = point.y;  CView::OnMouseMove(nFlags, point);  }  4.在设置好平移/旋转后，分别按键盘上的A/D进行平移或旋转。  if (nChar == 65)  {  clear();  p->move(pDC, this->center, this->move\_step);  }  if (nChar == 68)  {  clear();  p->move(pDC, this->center, -this->move\_step);  }  四、实验结果  五、遇到的问题及解决办法  1.问题：实现边标志算法时，填充总会在最大y和最小y位置多两条线。  解决方法：最大y：在循环时跳出条件是y<maxPoint.y。最小y：  if (inside&&zgr[y%10][x%20])  {  if (y == minPoint.y)//如果是最上面的一行，不绘制  continue;  pDC->SetPixel(x, y, fillcolor);  } | | | |
| 2.问题：实现边标志算法时，用课件上递增x的方法会导致精度不够，部分区域不绘制，部分区域多绘制。  解决办法：把dx、dy均定义为浮点数，x += dx这种办法递增解决。  3.问题：绘制Bezier曲线，课件上收敛条件不适用于MFC环境，会导致无法收敛，vector移除。  解决方法：epsilon值设为1。  4.问题：画出的立方体与理想中透视投影不一致。  解决方法：平移坐标系至屏幕中心，y轴正向向上解决。  pDC->SetMapMode(MM\_ANISOTROPIC);  pDC->SetWindowExt(rect.Width(), rect.Height());  pDC->SetViewportExt(rect.Width(), -rect.Height());  pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2); | | | |