|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **计算机图形学实验报告（三）** | | | | | | |
| 学 号 | 姓 名 | | 班 级 | | 报告日期 | |
| 180400715 | 杨卓宸 | | 1804102 | | 2020/10/29 | |
| 实验内容 | 利用Cohen-Sutherland 直线裁剪算法实现直线段的剪切  实现直线段裁剪的中点分割算法 | | | | | |
| 实验目的 | 1.建立一个MFC单文档程序，在窗口中间部位绘制一个矩形用来模拟窗口。  2.通过鼠标交互绘制直线段（橡皮筋直线，当按两次鼠标左键确定一条直线段）。  3.当按下鼠标右键时利用直线段裁剪算法对绘制的直线段进行裁剪，保留并显示裁剪之后的直线段。 | | | | | |
| 实验预备知识 | Cohen-Sutherland 直线裁剪算法：  假设裁剪窗口是标准矩形，由上（y=wyt）、下（y=wyb）、左（x=wxl）、右（x=wxr）四条边组成，如下图所示。  Cohen-Sutherland 直线裁剪算法：  假设裁剪窗口是标准矩形，由上（y=wyt）、下（y=wyb）、左（x=wxl）、右（x=wxr）四条边组成，如下图所示。 延长窗口四条边形成 9个区域。根据被裁剪直线的任一端点 P(x，y)所处的窗口区域位置，可以赋予一组4位二进制区域码C4C3C2C1。    编码定义规则：  第一位C1：若端点位于窗口之左侧，即 X<Wxl，则 C1=1，否则 C1=0。  第二位C2：若端点位于窗口之右侧，即 X>Wxr，则 C2=1，否则 C2=0。  第三位C3：若端点位于窗口之下侧，即 Y<Wyb，则 C3=1，否则 C3=0。  第四位C4：若端点位于窗口之上侧，即 Y>Wyt，则 C4=1，否则 C4=0。  裁剪步骤：  1. 若直线的两个端点的区域编码都为0，即 RC1|RC2=0（二者按位相或的结果为0，即 RC1=0 且RC2=0），说明直线两端点都在窗口内，应“简取”。  2. 若直线的两个端点的区域编码都不为0，即 RC1&RC2≠0（二者按位相与的结果不为0，即 RC1≠0且 RC2≠0，即直线位于窗外的同一侧，说明直线的两个端点都在窗口外，应“简弃”。  3. 若直线既不满足“简取”也不满足“简弃”的条件，直线段必然与窗口相交，需要计算直线与窗口边界的交点。交点将直线分为两段，其中一段完全位于窗口外，可“简弃”。对另一段赋予交点处的区域编码，再次测试，再次求交，直至确定完全位于窗口内的直线段为止。  4. 实现时，一般按固定顺序左（x=wxl）、右（x=wxr）、下（y=wyb）、上（y=wyt）求解窗口与直线的交点。  延长窗口四条边形成 9个区域。根据被裁剪直线的任一端点 P(x，y)所处的窗口区域位置，可以赋予一组4位二进制区域码C4C3C2C1。  编码定义规则：  第一位C1：若端点位于窗口之左侧，即 X<Wxl，则 C1=1，否则 C1=0。  第二位C2：若端点位于窗口之右侧，即 X>Wxr，则 C2=1，否则 C2=0。  第三位C3：若端点位于窗口之下侧，即 Y<Wyb，则 C3=1，否则 C3=0。  第四位C4：若端点位于窗口之上侧，即 Y>Wyt，则 C4=1，否则 C4=0。  裁剪步骤：  1. 若直线的两个端点的区域编码都为0，即 RC1|RC2=0（二者按位相或的结果为0，即 RC1=0 且RC2=0），说明直线两端点都在窗口内，应“简取”。  2. 若直线的两个端点的区域编码都不为0，即 RC1&RC2≠0（二者按位相与的结果不为0，即 RC1≠0且 RC2≠0，即直线位于窗外的同一侧，说明直线的两个端点都在窗口外，应“简弃”。  3. 若直线既不满足“简取”也不满足“简弃”的条件，直线段必然与窗口相交，需要计算直线与窗口边界的交点。交点将直线分为两段，其中一段完全位于窗口外，可“简弃”。对另一段赋予交点处的区域编码，再次测试，再次求交，直至确定完全位于窗口内的直线段为止。  4. 实现时，一般按固定顺序左（x=wxl）、右（x=wxr）、下（y=wyb）、上（y=wyt）求解窗口与直线的交点。 | | | | | |
| 实验过程描述 | 1.Cohen-Sutherland算法  void CTestView::CohenSutherland()//Cohen－Sutherland算法  {  BOOL isChanged;  double x, y;  RC0 = codePoint(Pointx[0], Pointy[0]);  RC1 = codePoint(Pointx[1], Pointy[1]);  while (TRUE)  {  isChanged = FALSE;  if (0 == (RC0 | RC1))  {  return;  }  else if (0 != (RC0 & RC1))  {  Pointx[0] = Pointy[0] = Pointx[1] = Pointy[1] = 0;  return;  }  else  {  if (0 == RC0)  //如果P0点在窗口内，交换P0和P1,保证p0点在窗口外  {  //交换点的坐标值  double cPointx, cPointy;  cPointx = Pointx[0]; cPointy = Pointy[0];  Pointx[0] = Pointx[1]; Pointy[0] = Pointy[1];  Pointx[1] = cPointx; Pointy[1] = cPointy;  //交换点的编码值  unsigned int TRC;  TRC = RC0; RC0 = RC1; RC1 = TRC;  }  //按左、右、下、上的顺序裁剪  if (RC0 & 1)//P0点位于窗口的左侧  {  x = wxl;//求交点y  y = Pointy[0] + (Pointy[1] - Pointy[0])\*(x - Pointx[0]) / (Pointx[1] - Pointx[0]);  Pointx[0] = x; Pointy[0] = y;  isChanged = TRUE;  RC0 = codePoint(Pointx[0], Pointy[0]); RC1 = codePoint(Pointx[1], Pointy[1]);  }  if (RC0 & 2)//P0点位于窗口的右侧  {  x = wxr;//求交点y  y = Pointy[0] + (Pointy[1] - Pointy[0])\*(x - Pointx[0]) / (Pointx[1] - Pointx[0]);  Pointx[0] = x; Pointy[0] = y;  isChanged = TRUE;  RC0 = codePoint(Pointx[0], Pointy[0]); RC1 = codePoint(Pointx[1], Pointy[1]);  }  if (RC0 & 4)//P0点位于窗口的下侧  {  y = wyb;//求交点x  x = Pointx[0] + (Pointx[1] - Pointx[0])\*(y - Pointy[0]) / (Pointy[1] - Pointy[0]);  Pointx[0] = x; Pointy[0] = y;  isChanged = TRUE;  RC0 = codePoint(Pointx[0], Pointy[0]); RC1 = codePoint(Pointx[1], Pointy[1]);  }  if (RC0 & 8)//P0点位于窗口的上侧  {  y = wyt;//求交点x  x = Pointx[0] + (Pointx[1] - Pointx[0])\*(y - Pointy[0]) / (Pointy[1] - Pointy[0]);  Pointx[0] = x; Pointy[0] = y;  isChanged = TRUE;  RC0 = codePoint(Pointx[0], Pointy[0]); RC1 = codePoint(Pointx[1], Pointy[1]);  }  if (FALSE == isChanged)  {  return;  }  }  }  }  2.直线段裁剪的中点分割算法  void CTestView::midLineClip()  {  RC0 = codePoint(Pointx[0], Pointy[0]);  RC1 = codePoint(Pointx[1], Pointy[1]);  double p1\_x, p1\_y, p2\_x, p2\_y;  p1\_x = Pointx[0];  p1\_y = Pointy[0];  p2\_x = Pointx[1];  p2\_y = Pointy[1];  if (0 == (RC0 | RC1))  {  return;  }  //存在一个点在裁剪框内  else if (0 != (RC0 & RC1))  {  Pointx[0] = Pointy[0] = Pointx[1] = Pointy[1] = 0;  return;  }  else  {  if (RC0 == 0)  findIntersection(p1\_x, p1\_y, p2\_x, p2\_y);  else if (RC1 == 0)  findIntersection(p1\_x, p1\_y, p2\_x, p2\_y);  else  {  Pointx[0] = (p1\_x + p2\_x) / 2;  Pointy[0] = (p1\_y + p2\_y) / 2;  midLineClip();  }  }  }  //找出p1和p2之间边界的交点  double CTestView::findIntersection(double p1\_x, double p1\_y, double p2\_x, double p2\_y)  {  double p\_x, p\_y;  p\_x = (p1\_x + p2\_x) / 2;  p\_y = (p1\_y + p2\_y) / 2;  if (distance(p\_x, p\_y, p1\_x, p1\_y) < 1.5)  return p\_x;  RC0 = codePoint(Pointx[0], Pointy[0]);  RC1 = codePoint(Pointx[1], Pointy[1]);  RC = codePoint(p\_x, p\_y);  if (RC0 == 0)  if (p\_x == 0)  return findIntersection(p\_x, p\_y, p2\_x, p2\_y);  else  return findIntersection(p1\_x, p1\_y, p\_x, p\_y);  if (RC1 == 0)  if (p\_x == 0)  return findIntersection(p\_x, p\_y, p1\_x, p1\_y);  else  return findIntersection(p2\_x, p2\_y, p\_x, p\_y);  return p\_x;  }  double distance(double p1\_x, double p1\_y, double p2\_x, double p2\_y)  {  double result;  result = sqrt((p1\_x - p2\_x) \* (p1\_x - p2\_x) + (p1\_y - p2\_y) \* (p1\_y - p2\_y));  return result;  } | | | | | |
| 实验结果 | 1.直线与边框有两个交点    2.直线与边框有一个交点    3.直线在边框内部    4.直线在边框外部 | | | | | |
| 实验当中问题  及解决方法 | 问题：利用鼠标交互式画完直线后，按下鼠标右键进行剪切却没有任何变化。  解决：没有先对之前画的直线进行清空，之前的直线将新的直线段覆盖了，因此先对原来直线段进行清空，利用一个简单的方法，即用白色的画笔重新绘制一遍将原来的覆盖掉。 | | | | | |
| 成绩（教师打分） | 优秀 | 良好 | | 及格 | | 不及格 |