《计算机图形学》10月报告

171850505，宋昱豪，yhsong.nju@gmail.com

2019 年 12 月 22 日

1 综述

基于计算机图形学设计了一套画图系统。完成了画图软件。主要实现了基于各种算法的图元生成，图元编辑，文件的保存打开，基于文档文件的指令序列读取完成画图任务，基于鼠标交互的图元绘制。并在此基础上对于软件界面进行了美化，具有一定可用性。

关键词： openGL　　Qt gui设计　　计算机图形学

2 算法介绍

2.1 直线绘制算法

2.1.1 DDA算法

数字差分分析法（Digital Differential Analyzer, DDA）是利用计算两个坐标方向的差分来确定线段显示的屏幕像素位置的线段扫描转换算法。DDA算法的核心是增量式的思想。即利用光栅的特性消除了直线方程中的乘法，在x和y方向使用合适的增量来逐步沿直线的路径推导出各像素的位置。

现假设直线的方程为

$$y = m \times x + b \tag{1}$$

假设斜率$m \leq 1$，在x的维度上单位间隔采样并求每个采样点对应的y值，可以发现在对于$x_{k+1} = x_k + 1$的x采样时，$y_{k+1} = y_k + m$。因此在屏幕光栅的特点下，我们采用增量式的方法计算出直线路径上各像素点的位置。而当斜率m>1时，我们在y的维度上进行采样，对于$y_{k+1} = y_k + 1$时，$x_{k+1} = x_k + \frac{1}{m}$。在起始点和终点位置不同时增量的符号可能会改变。

2.1.2 Bresenham算法

在DDA算法中，由于斜率为浮点数，而浮点增量的连续迭加中取整误差的积累会使得计算得到的像素位置与实际上的像素位置偏离。同时浮点计算也十分耗时。而Bresenham算法通过引入整形参量定义来衡量两候选像素与直线路径上实际点在方向上的偏移。并通过对该参量的符号判定来决定像素点的选择。

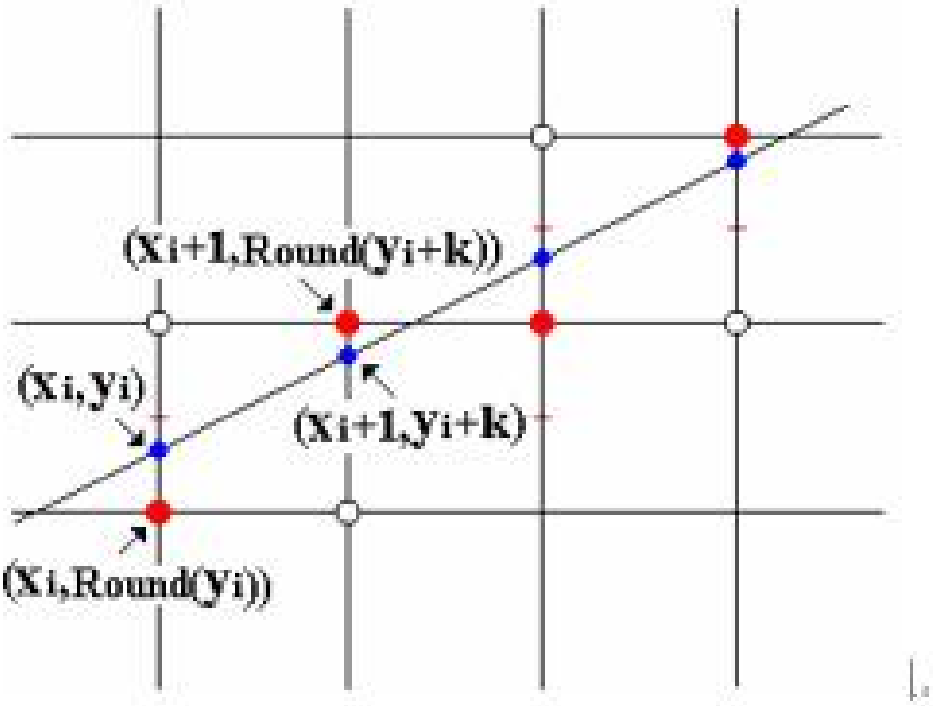


图 1: Brensenham 示意图

假设依然以x的维度为基础，设目前已经绘制到第k步在点（x_k,y_k），下一个点的位置在（x_{k+1},y_k）或者（x_{k+1},y_{k+1}）。由于点x_{k+1}处y的值可以表示为

$$y = m \times x_{k+1} + b = m \times (x_k + 1) + b \tag{2}$$

此时两个候选的像素和线段的数学路径的垂直偏移为

$$\begin{cases} d_1 = y - y_k = m \times (x_k + 1) + b - y_k \\ d_2 = y_{k+1} - y = y_k + 1 - m \times (x_k + 1) + b \end{cases} \tag{3}$$

这两个点距离理想点的距离的差分为

$$d_1 - d_2 = 2 \times (x_k + 1) - 2 \times y_k + 2 \times b - 1 \tag{4}$$

为了达到消除浮点数运算的目的，令$m = frac\Delta y\Delta x$，设变量

$$p_k = \Delta x(d_1 - d_2) = 2\Delta y x_k - 2\Delta x y_k + c \tag{5}$$

同时由于c为常数，所以在循环计算中被消除，因为$\Delta x = 1$>0，所以p_k的符号与$d_1 - d_2$的符号相同。当y_k处的像素点比y_{k+1}处的像素点更加接近理想线段时，$p_k < 0$，反之则大于零。所以我们可以根据p_k的符号来决定k+1步走到的位置。

2.2 圆形绘制算法

采用中点圆生成算法，为了简化运算，只计算出八分之一的圆上的点，其他的点由计算单得到的八分之一圆旋转对称得到。

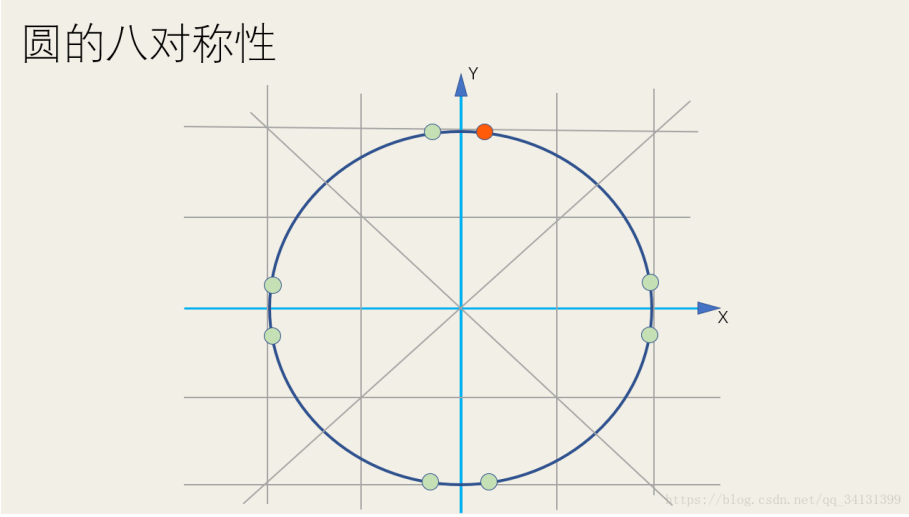


图 2: 圆的对称性示意图

在中点圆算法中，和画直线的Bresenham算法类似，判断的参数是由点到中心的距离计算得到的。计算的是两个候选点的中点到圆心的距离。

$$\begin{cases} p_{k+1} = f_{circle}(x_{k+2}, y_k - frac12) & p_k < 0 \\ p_{k+1} = f_{circle}(x_{k+2}, y_k - frac32) & p_k \geq 0 \end{cases} \tag{6}$$

计算推导可得

$$\begin{cases} p_{k+1} = p_k + 2x_k + 3 & p_k < 0 \\ p_{k+1} = p_k + 2x_{k+1} - 2y_k + 5 & p_k \geq 0 \end{cases} \tag{7}$$

在计算中设置p_0为$5 - 4 \times r$ 初始化第一个点为(0,r)开始计算

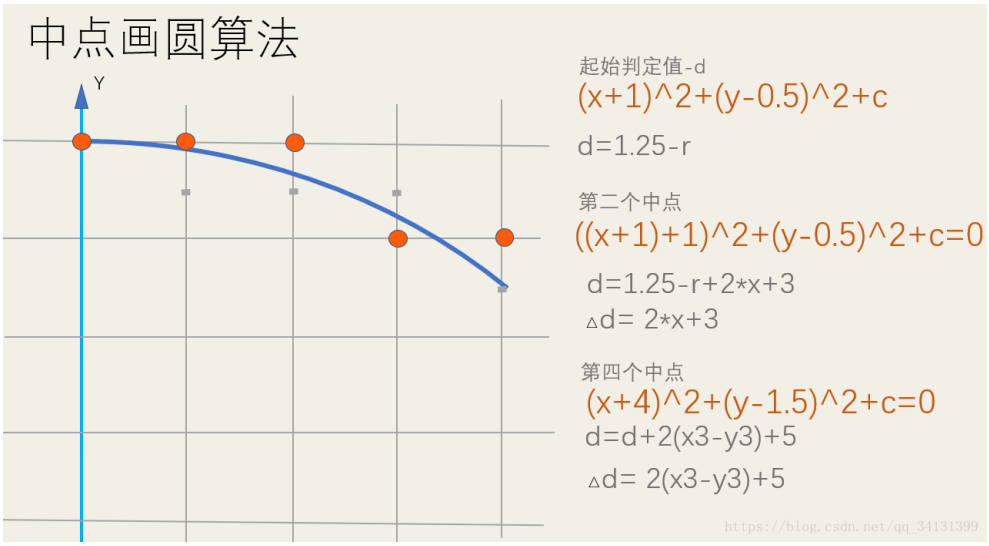


图 3: 中点圆画法示意图

2.3 中点椭圆绘制算法

椭圆绘制算法类似于中点圆生成算法，但是中点椭圆绘制算法中只能计算出四分之一的图形，剩下的用对称可以得到。在这四分之一图形中我们又要分成两个部分来计算点的位置。我们以一个以原点为中心的椭圆为例，我们只需要计算其第一象限内的点的位置。区域一即为其切线斜率小于1的部分，区域二为大于1的部分。在区域一中，我们有参数

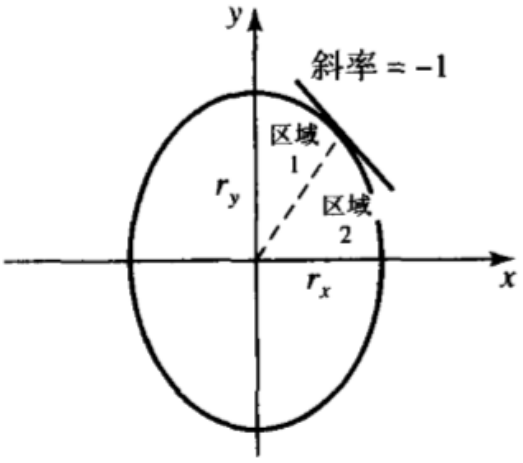


图 4: 椭圆对称性示意图

$$\begin{cases} p_{k+1} = p_k + 2r_y^2 x_k + 3r_y^2 & p_k < 0 \\ p_{k+1} = p_k + 2r_y^2 x_{k+1} - 2r_x^2 y_k + 3r_y^2 & p_k \geq 0 \end{cases} \tag{8}$$

在区域二中参数变化为

$$\begin{cases} p_{k+1} = p_k - 2r_x^2 y_k + 3r_x^2 & p_k < 0 \\ p_{k+1} = p_k + 2r_y^2 x_{k+1} - 2r_x^2 y_k + 3r_x^2 & p_k \geq 0 \end{cases} \tag{9}$$

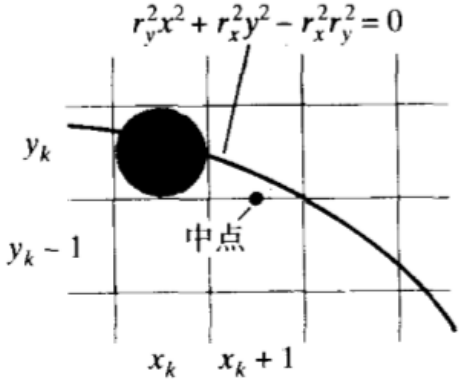


图 5: 中点椭圆画法示意图

2.4 矩形绘制算法

矩形绘制算法即根据获得到的两个点的坐标，分别作为矩形对角线的两个端点，生成其他端点并调用画线算法即可完成。

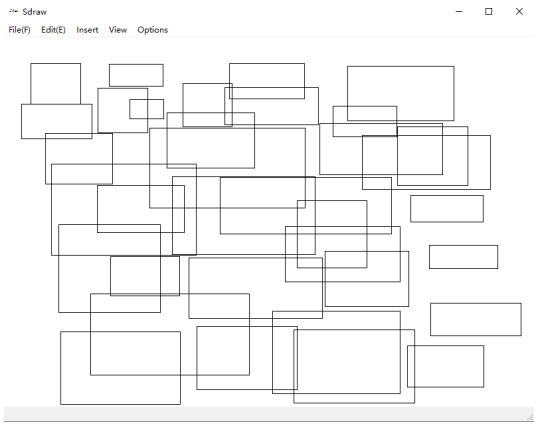


图 6: 矩阵示意图

2.5 曲线算法

此软件中实现了两种曲线绘制算法，即贝塞尔曲线和B样条曲线

2.5.1 贝塞尔曲线

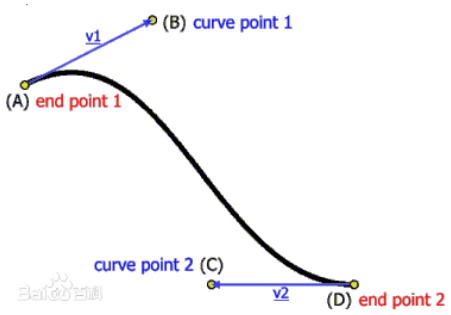


图 7: 贝塞尔曲线画法示意图

Bézier curve(贝塞尔曲线)是应用于二维图形应用程序的数学曲线。曲线定义：起始点、终止点（也称锚点）、控制点。通过调整控制点，贝塞尔曲线的形状会发生变化。 1962年，法国数学家Pierre Bézier第一个研究了这种矢量绘制曲线的方法，并给出了详细的计算公式，因此按照这样的公式绘制出来的曲线就用他的姓氏来命名，称为贝塞尔曲线。

Bezier曲线是参数多项式曲线,它由一组控制多边形折线(控制多边形)的顶点唯一定义,在控制多边形的各顶点中,只有第一个和最后一个顶点在曲线上,其他的顶点则用以定义曲线的导数,阶次和形状。Bezier曲线的数学基础是能够在第一个和最后一个顶点之间进行插值

的一个多项式混合函数,对于有n+1个控制点的Bezier曲线段用参数方程表示如下：

$$P(t) = \sum_{k=0}^{n} P_k BENk, n(t) \qquad t \in [0,1] \tag{10}$$

式中$P_k(x_k, y_k, z_k)$,k=0,1,2⋯..n是控制多边形的n+1个顶点,$BEN_{k,n}$(t)是Bernstein基函数。

$$BENk, n(t) = \frac{n!}{k! * (n-k)!} * t^k * (1-t)^{n-k} \tag{11}$$

2.5.2 B样条曲线

B-样条是贝塞尔曲线的一种一般化，B样条不能表示一些基本的曲线，比如圆，所以引入了NURBS,可以进一步推广为非均匀有理B-样条(NURBS)。三者关系可以表示为：

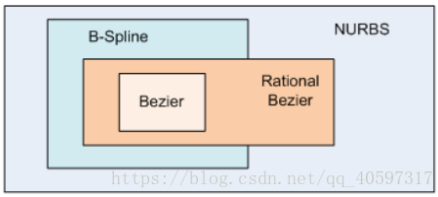


图 8: 曲线包含关系示意图

给定n+1个控制点，$P_0, P_1, ..., P_n \Gamma * \mathrm{J} U = u_0, u_1, ..., u_m$, p次B样条曲线由这些控制点和节点向量U 定义，其公式为：

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \tag{12}$$

上式中，$N_{i,p}(u)$为B样条基函数

设U 是m+1个非递减数的集合，$u_0 <= u_1 <= u_2 <= ... <= u_m \Theta u_i$称为节点（knots）, 集合U 称为节点向量（knot vector）, 半开区间$[u_i, u_{i+1})$ 是第i个节点区间（knot span）。注意某些u_i可能相等，某些节点区间会不存在。如果一个节点u_i出现 k 次 (即，$u_i = u_{i+1} = ... = u_{i+k-1}$), 其中 k ¿ 1, ui 是一个重复度（multiplicity）为k 的多重节点，写为 $u_i(k)$。否则，如果u_i只出现一次，它是一个简单节点。如果节点等间距(即，$u_{i+1} - u_i$是一个常数，对 0 ¡= i ¡= m - 1)，节点向量或节点序列称为均匀的；否则它是非均匀的。一般情况下，我们经常使用 $u_0 = 0 u_m = 1$，所以定义域是闭区间[0,1]。

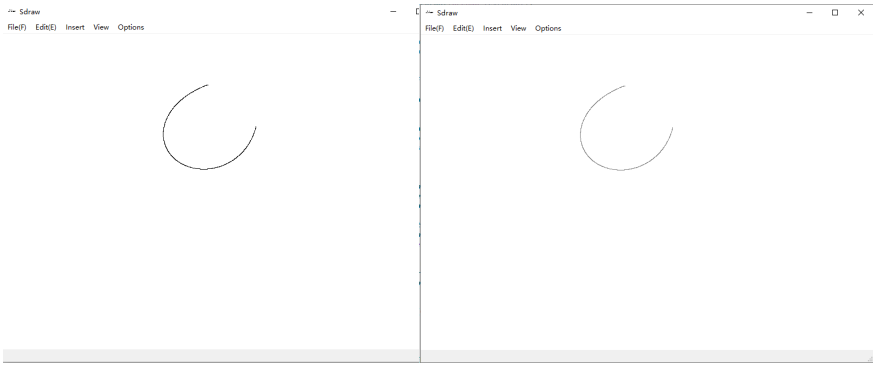
为了定义B-样条基函数，我们还需要一个参数，基函数的次数（degree）p，第i个p次B-样条基函数，写为$N_{i,p}(u)$，递归定义如下：

$$N_{i,0}(u) = \begin{cases} 1 & if u_i \le u \le u_{i+1} \\ 0 & otherwise \end{cases} \tag{13}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{14}$$

2.6 选中算法

选中即为，当用户鼠标点击的坐标所对应的像素点时，如果该像素点对应于画布上的图元，则选中该图元。选中算法为了用户操作的鲁棒性，当用户点击的坐标附近一定参数范围内存在相应的图元时，则选中该图元。程序中搜索相应图元位置则是在画图的画板中遍历搜索，当搜索到第一个距离点击点在操作范围内的图元时，则将标志变量choosenid标志为该图元的图元编号，同时将ischoose flag设置为true，表示成功选中了图元，在paintGL中，每次刷新画图时，如果发现当前遍历到的图元被选中的话，则将该图元颜色变化为灰色。



(a) 原图　　(b) 选中后

图 9: 选中算法示意图

2.7 删除算法

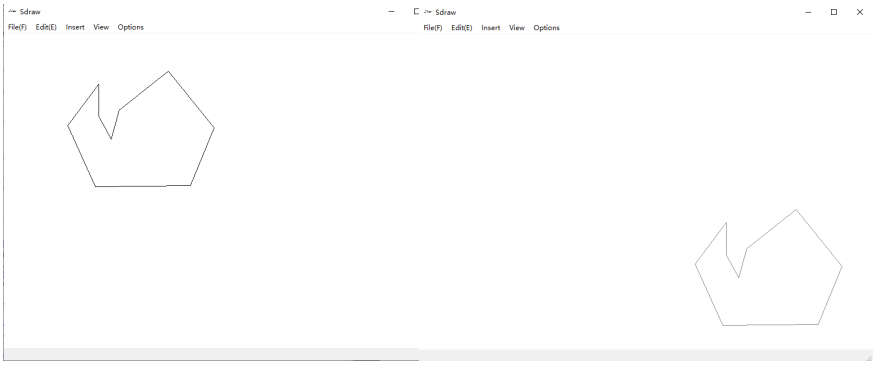
删除算法是在图形化界面中的服务的功能。在图形界面中必须基于选中算法来完成，该算法即遍历画布中所有的像素点，如果像素点的状态为选中，则从画布集合中删除该点。

2.8 移动算法

移动算法的完成较为简单，在图形界面中是依赖于选中算法来完成的，首先遍历画布集合中所有的点，如果发现有像素点的状态为选中，则将其位置加上参数需要的偏移量。在读文件操作中偏移量是已经给出的。但是在图形化界面中，偏移量是由用户操作完成的，所以我们设定用户一次点击释放鼠标左键为一次移动完成，开始点即为用户press左键所在的位置，结束的位置即为用户release鼠标左键所在的位置。用开始结束的位置信息来计算移动向量。传给移动函数，完成移动函数。

2.9 裁剪算法

裁剪算法由两种算法完成：



(a) 原图　　(b) 移动后

图 10: 移动算法示意图

2.9.1 CohenSutherland

Cohen-Sutherland 算法是最早、最流行的线段裁剪算法,核心思想是通过编码测试来减少要计算交点的次数。线段端点按区域赋以四位二进制码(区域码)。

区域码各位从右到左编号：

位1：上边界。

位2：下边界。

位3：右边界。

位4：左边界。

区域码位＝1：端点落在相应位置上,区域码位＝0：端点不在相应位置上。

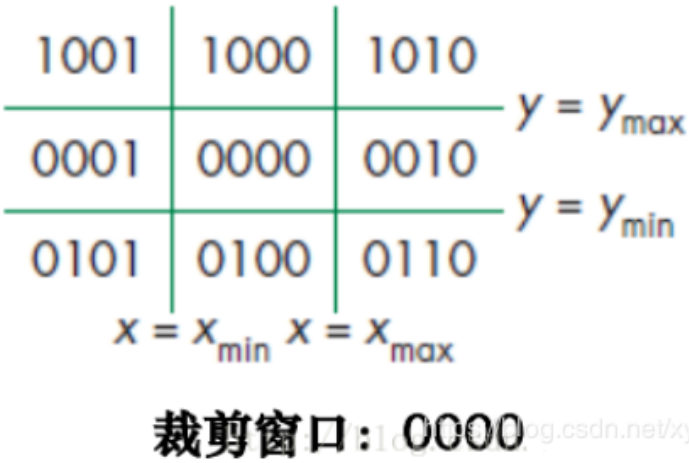


图 11: 编码

区域码位值通过将端点坐标值(x,y)与裁剪窗口边界坐标比较来确定。根据线段端点的区域码快速判断：1. 完全在窗口边界内的线段：两端点区域码均为0000；2.完全在裁剪矩形外的线段：两端点区域码同样位置都为1；对两个端点区域码进行逻辑与操作，结果不为0000。不能确定完全在窗口内外的线段，进行求交运算：按“左-右-上-下”顺序用裁剪边界检查线段端点：将线段的外端点与裁剪边界进行比较和求交，确定应裁剪掉的线段部

分；反复对线段的剩下部分与其它裁剪边界进行比较和求交，直到该线段完全被舍弃或找到位于窗口内的一段线段为止。

在程序的代码实现上，发现问题是在求交操作上，由于像素点在屏幕上的离散分布，所以程序求得的交点很可能不在原本的直线上，而且即使在原本的直线上，裁剪重新得到的直线与原本直线也可能不重合，所以在程序中需要使用删除原本的图元再重新画的操作。在计算交点逐步裁剪的过程中使用一个while循环，每次裁剪一段，直到直线的两端都在区域0000时结束。

2.9.2 Liang-barsky

设要裁剪的直线段为P_0 P_1, P_i的坐标为(x_i, y_i)，i=0, 1。 P_0P_1和窗口边界交于A、B、C和D四个点。我们知道，一条两端点为$P_1\Phi x_1$fi$y_1\Psi\Delta P_2\Phi x_2fiy_2\Psi$的线段可以用参数方程形式表示：

$$\begin{cases} x = x_1 + u(x_2 - x_1) = x_1 + u\delta x \\ y = y_1 + u(y_2 - y_1) = y_1 + u\delta y \end{cases} \tag{15}$$

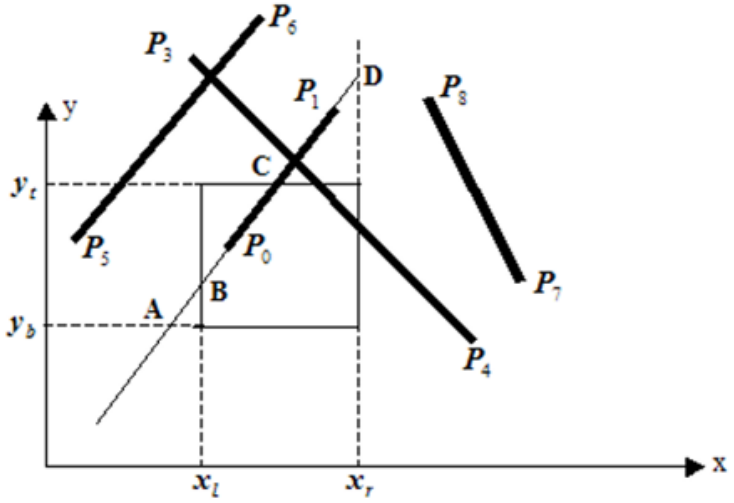


图 12: 交点

式中，$\delta x = x_2 - x_1$，$\delta y = y_2 - y_1$，参数u在0～1之间取值，P（x，y）代表了该线段上的一个点，其值由参数u确定，由公式可知，当u=0时，该点为$P_1\Phi x_1$fi$y_1\Psi$，当u=1时，该点为$P_2\Phi x_2$fi$y_2\Psi$。如果点P（x，y）位于由坐标Φx_{min}fi$y_{min}\Psi\Phi x_{max}$fi$y_{max}\Psi$所确定的窗口内，那么下式成立：

$$\begin{cases} x_{min} \le x_1 + u\delta x \le x_{max} \\ y_{min} \le y_1 + u\delta y \le y_{max} \end{cases} \tag{16}$$

这四个不等式可以表示为：

$$u * p_k \le q_k \qquad k = 1, 2, 3, 4 \tag{17}$$

任何平行于窗口某边界的直线，其pk=0，k值对应于相应的边界（k=1，2，3，4对应于左、右、下、上边界）。

如果$q_k < 0$，则线段完全在边界外，应舍弃该线段。

如果$q_k \geq 0$，则线段平行于窗口某边界并在窗口内。

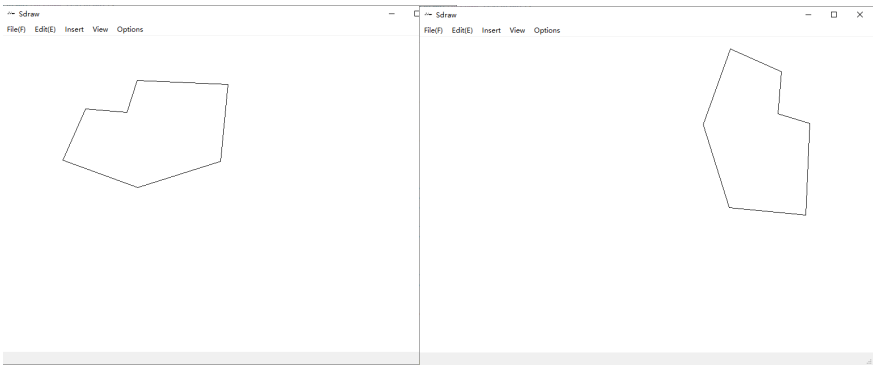
所以，当pk¡0时，线段从裁剪边界延长线的外部延伸到内部；当pk¿0时，线段从裁剪边界延长线的内部延伸到外部。

对于每条直线，可以计算出参数u1和u2，该值定义了位于窗口内的线段部分：1.u1的值由线段从外到内遇到的矩形边界所决定（pk¡0），对这些边界计算rk=qk/pk，u1取0和各个r值之中的最大值。 2.u2的值由线段从内到外遇到的矩形边界所决定（pk¿0），对这些边界计算rk=qk/pk，u2取1和各个r值之中的最小值。 3.如果u1¿u2，则线段完全落在裁剪窗口之外，应当被舍弃；否则，被裁剪线段的端点可以由u1和u2计算出来。

2.10 旋转算法

旋转算法实现的是刚性旋转，所以我们只需要旋转改变图元的构造点，即对构造点进行操作。所以我们需要改变之前存储图元的方式，为了方便接下来此类操作，我们创造了一个图元字典，存储了图元的构造点数据，图元编号，图元所属模式等等属性。所以当需要对图元进行构造上的操作改变时，只需要从字典中检索即可。旋转算法只需要将原本的坐标乘一个矩阵，便可以得到所需要的新的点的位置。

$$\begin{cases} x = x_r + (x - x_r)cos(\theta) - (y - y_r)sin(\theta) \\ y = y_r + (x - x_r)sin(\theta) + (y - y_r)cos(\theta) \end{cases} \tag{18}$$



(a) 原图　　(b) 旋转90度后

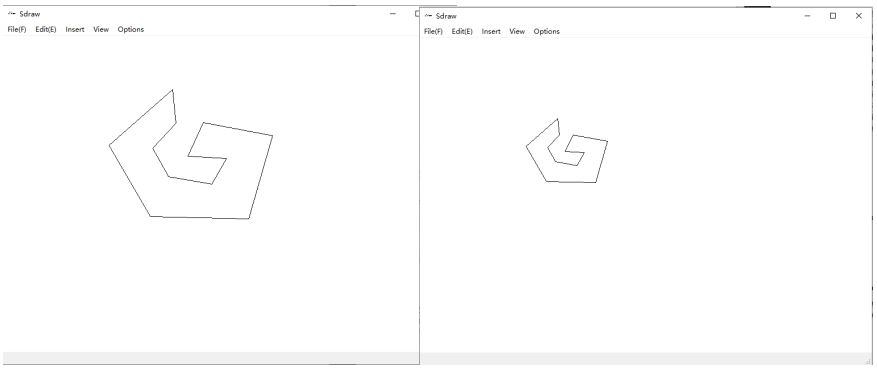
图 13: 旋转算法示意图

2.11 缩放算法

缩放算法类似于旋转算法，也是将点的坐标乘以矩阵，但是由于放大缩小点的数量不同，单纯的将图元中所有点乘矩阵会使得缩小时较小的面积上重复汇聚了较多的点，而放大时更大的空间上汇聚的点数依然与之前相同，导致图元上点过于稀疏，模糊不清，且多次放缩后会使得图元变形，所以我们采用类似旋转算法的方法，只对字典中图元的构造点

集进行操作。公式如下：

$$\begin{cases} x1 = x * s_x + x_f * (1 - s_x) \\ y1 = y * s_x + x_f * (1 - s_x) \end{cases} \tag{19}$$



(a) 原图　　(b) 缩放0.5后

图 14: 缩放算法示意图

3 系统介绍

3.1 系统架构

系统基于Qt gui设计完成了软件。主要分为两个模块，一个是主窗口模块，一个是图形库模块。

主窗口模块主要完成了gui的交互功能，图形库则主要完成了图元的绘制，包含了软件的各种操作。

主要思想是利用QOpenGLWidget自带的三个函数，paintGL,initializeGL,resizeGL, 来完成图元的显示，图元在机器上存储在一个QVector中，Vector采用的模板是自定义的Point类型，类型中有choosen作为flag来标识是否被选中，pid表示所在图元的编号，size表示画这一点时其大小，mode表示其是在什么图元绘制模式下被绘制的。color数组表示了其颜色的RGB。x,y表示了其所在位置。

paintGL在软件中负责将Vector中存储的数据画出。

图元绘制的基本函数则是点的绘制，注意在这里绘制指的是将Point对象存储到QVector中。基于点绘制的高级图元绘制则在之前的算法中已经涉及。

之后还涉及一些简单的系统功能的实现，如选中，我设计了当按下中键时取消所在绘制状态改为选择状态，此时当鼠标的位置在一个图元的像素附件，则求出此像素所归属图元并求出图元pid，再搜索所有pid与此pid相等的点，将choosen即flag一一标志为真。当用户点到无像素地区，则在一般软件中，将取消选中，所以我们也采用了此类逻辑。被选中的图元在显示时颜色变灰，并可以进行编辑一栏中的操作，如删除（可以按快捷键）。

于此同时还实现了文档读取软件开始绘制已实现功能，保存画布等功能。具体操作等详见使用手册。