计算机图形学六月报告

171860659　　吴紫航　　联系邮箱：401986905@qq.com

2020 年 6 月 23 日

一. 综述

计划完成一个功能完善的图形学系统。实现各种图形学的算法，完成绘图软件的常见功能。如常见图形的绘制、平移、旋转、放缩、裁剪、填充等。系统分成两个部分，gui 部分以图形界面的形式进行交互；cli 部分以读取文件命令行的形式进行交互。

完成的内容（额外功能为加粗部分）

Gui 系统：新建画布、**打开图片**、保存图片、**另存为**、**关于信息**、退出画布、绘制直线（DDA 和 bresenham）、绘制多边形（DDA 和 bresenham）、**绘制矩形**、**绘制三角形**、绘制椭圆（中点法）、绘制曲线（Bezier 和 B-spline）、**绘制字符**、设置画笔颜色、**粗细设置**、**铅笔**、**刷子**、**橡皮**、**选择图元（可调整图元的控制点和复制粘贴）**、**快捷键**、平移图元、旋转图元、放缩图元、裁剪直线图元（Cohen-Sutherland 、 Liang-Barsky 和 **Nicholl-Lee-Nicholl** ）、 **裁剪多边形图元（Sutherland-Hodgeman）**、**填充图元（多边形扫描线填充和区域种子填充）**、**撤销操作**、**重做操作**、清空画布、**退出时如画板有修改则提示保存**、**状态栏显示当前鼠标像素位置、操作序列和状态、画板大小等**

Cli 系统：识别文件指令、图元对象的生成建模、画布的绘制和保存

二. 算法介绍

1.DDA

1）原理

a）DDA 的核心是根据斜率和 x 或 y 方向单位的间隔（1 个单位），来计算 y 或 x 方向的间隔。因为像素是整数，所以每一步结果都需要取整操作。

b）斜率绝对值小于 1 时，在 X 方向取样；斜率绝对值大于 1 时，在 Y 方向取样。这样的目的是为了保证直线不会太稀疏而失真。

c) 增量的取值取决于直线的生成方向和坐标轴的方向的关系，两个方向相同取+1；两个方向相反取-1

2) 过程

a)当斜率 m 绝对值在[0,1]区间，从左端点到右端点，在 x 方向取样，$\Delta x = 1$, 顺序计算 y 值：$y_{k+1} = y_k + m,\ k \in Z$, 计算的结果需要取整

b)当斜率 m 绝对值在[0,1]区间, 从右端点到左端点, 在 x 方向取样, $\Delta x = -1$, 顺序计算 y 值：$y_{k+1} = y_k - m,\ k \in Z$, 计算的结果需要取整

c)当斜率 m 绝对值在[1,+ ∞]区间，从左端点到右端点，在 y 方向取样，$\Delta y = 1$, 顺序计算 x 值：$x_{k+1} = x_k + \frac{1}{m},\ k \in Z$, 计算的结果需要取整

d) 当斜率 m 绝对值在[1,+ ∞]区间, 从右端点到左端点, 在 y 方向取样, $\Delta y = -1$,顺序计算 x 值：$x_{k+1} = x_k - \frac{1}{m},\ k \in Z$, 计算的结果需要取整

3)特点

a)和直线方程法相比，消除乘法操作，速度更快

b)结果需要取整，有累计误差

c)存在浮点运算，比较耗时

4）注意事项

实际代码处理的时候要注意两点：

a)注意斜率不存在的特殊情况，按 x 不变，y 步长为 1 处理，从下到上生成

b)注意处理斜率绝对值刚好是 1 的情况，可以直接归类在 x 方向取样

2.bresenham

1）原理

a)按个人理解，bresenham 算法的核心是将原本增量计算的问题（涉及浮点数和取整操作）转化为决策问题（抉择随着 x 或 y 改变一个单位，y 或 x 是否改变一个单位）。完全是整数运算，没有取整操作，只需要初始化然后迭代即可。

b)x_{k+1}处的 y 坐标为$mx_{k+1}+b$，屏幕坐标$y_{k+1}=y_k+1; x_{k+1}=x_k+1$

c)候选像素的中点和实际数学路径的位置关系，由决策参数 pk 的正负决定。

d)pk>0，则候选像素的中点在实际位置下方，故选择高像素；反之低像素

e）由于只关心 pk 正负，所以可以通过迭代方式消除常数，简化计算

2) 过程（以 m 绝对值小于 1 为例）

a)输入两个端点，把左端点存储到帧缓冲器，绘制第一个点

b)起始决策参数是$p_0=2\Delta y-\Delta x$，计算常量Δx、Δy、$2\Delta y$、$2\Delta x$

c)从 k=0 开始，每个离散取样点判断以下两步

e)如果 pk<0 则绘制$(x_{k+1}y_k), p_{k+1}=p_k+2\Delta y$

f) 如果 pk>0 则绘制$(x_{k+1}y_{k+1}), p_{k+1}=p_k+2\Delta y-2\Delta x$

g) k=k+1

h) 重复 c-g 共Δx次

3)特点

没有取整操作和浮点运算，速度比 DDA 快。

4）注意事项

a)当斜率小于 1，按 x 方向取样；当斜率大于 1，按 y 方向取样

b)常量Δx、Δy、$2\Delta y$、$2\Delta x$都是绝对差值，大于 0

c)起始端点决定直线生成方向和相应坐标值的增量正负

d)对斜率不存在和对角线特殊处理即可

3.中点椭圆

1）原理

a)和 bresenham 算法类似的是，都是判断候选像素的中点和曲线的相对位置关系。并且把计算增量的问题转化为迭代和决策的问题

b)定义椭圆函数$F_{ellipse} = {r_y}^2x^2 + {r_x}^2y^2 - {r_x}^2{r_y}^2$，当其小于零，候选中点在椭圆内，当其等于零，则在椭圆上，否则在椭圆外。因此其可以作为决策参数

c)对椭圆第一象限分成两部分处理，当斜率绝对值小于 1，按 x 方向离散取样；当斜率绝对值大于 1，在 y 方向离散取样。对于其他象限只需要对称取点即可。

d)根据导数知识，通过判断${r_y}^2x \geq {r_x}^2y^2$来确定什么时候从斜率绝对值大于 1 变成小于 1

2) 过程

a)确定椭圆的长半轴 rx、短半轴 ry 长度，确定中心位置（xc，yc）

b)第一个点为（0，ry），决策参数 pl 为$ry^2 - rx^2ry + \frac{rx^2}{4}$

c)对于每个位置循环决策以下几步

e）如果 pl<0,选择(x_{k+1}, y_k),pl+=$2ry^2x_{k+1} + ry^2$

f) 否则选择$(x_{k+1}, y_k - 1)$,pl+=$2ry^2x_{k+1} - 2rx^2y_{k+1} + ry^2$

g) ef 循环至${r_y}^2x \geq {r_x}^2y^2$，假设区域 1 最后一个点是(x_0, y_0)

h) 初始化区域 2 的初值 pl=$ry^2(x_0 + 0.5)^2 + rx^2(y_0 - 1)^2 - {r_x}^2{r_y}^2$

i) 循环以下

j) 如果 pl>0,选择$(x_k, y_k - 1)$,pl+=$-2rx^2y_{k+1} + rx^2$

k) 否则选择$(x_{k+1}, y_k - 1)$,pl+=$2ry^2x_{k+1} - 2rx^2y_{k+1} + rx^2$

l) 循环至 ry=0，再用对称性确定其他象限，并进行平移操作即可

3)特点

简化复杂的数学运算，不用取整操作，只要迭代决策即可。

4）注意事项

圆锥曲线的生成都可以用中点法的思路来进行

4.bezier 绘制曲线

1）原理

个人对算法的理解是基于递归的。给定一个参数 t。t 可以在[0,1]内任意取值。当只有一个控制点的时候，输出的结果点就是该控制点；当有 k 个控制点的时候，输出的结果点由 t*(前 k-1 个控制点)+（1-t）*（后 k-1 个控制点）。因此，给定一个参数 t 和 n 个控制点，可以唯一确定一个输出点，当参数 t 从[0,1]取样，会

生成一系列结果输出点，这些结果点的绘制效果就是 bezier 曲线。

2) 过程

以上原来概括成算法（de Casteljau 算法）就是

$$p_i^k = \begin{cases} p_i & k=0 \\ (1-t)p_i^{k-1} + tp_{i+1}^{k-1} & k=1,2,\ldots,n, i=0,1,\ldots,n-k \end{cases}$$

3)特点

可以拟合任何数目的控制点；控制点较多的时候运算相对比较慢（实际代码运行结果表明的）；对曲线的变换只作用于控制点即可；曲线位于控制点的凸包内；移动一个控制点，对曲线一个局部影响比较大，其他局部影响相对小

4）注意事项

实际代码实现的时候比较了递归方案和循环方案的效率。发现采用循环方式会快很多。循环方案的原理，基于 Bernstein 基函数，即 n 次 Bernstein 基函数多项式 $BEZ_{i,n} = C(n,i)u^i(1-u)^{n-i}$，$C(n,i)$为组合数。实际测试比较的时候循环方案比递归方案快了大概 50 倍!

5.B-Spline 绘制曲线

1）原理

在 bezier 曲线生成可以利用基函数的方法。B-Spline 曲线也可以用这样的方式。假设有 n 个控制点，就需要 n 个基函数作为加权；同时需要一个节点向量 $[u_0, u_1, \ldots, u_m]$。对于每个节点向量区间范围内的参数 u，生成一个目标点。目标点的坐标就是控制点坐标关于基函数的加权总和。生成基函数的方法可以用 Cox-de Boor 递归公式。

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

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

2) 过程

按照个人理解，总体就分为 2 步:

a) 假设阶数是 k，对于 u∈[k-1,n)的每一个值，希望算出对应生成点

b) 每个生成点的计算过程中，对第 0 到第 n-1 项分别计算基函数并和控制点进行加权求和。当然这里实际上只要考虑第 int（u）+1-k 项到 int（u）项即可，因为其他项基函数肯定是 0，不用计算

3)特点

a) $N_{i,p}(u)$是 u 上的 p 次多项式

b) 所有的 i，p 和 u，$N_{i,p}(u)$都是非负的

c）局部性，即对于每个参数 u，计算基函数只考虑$[u_i, u_{i+p+1})$的范围，这样改变控制点只能局部性的影响周边生成点的位置!

d)在任意区间$[u_i, u_{i+1})$最多有 p+1 个 p 次基函数不为 0

e)单位分解：即所有非零的 p 次基函数在区间$[u_i, u_{i+1})$*上的和是* 1

4）注意事项

大作业的要求是生成 4 阶均匀 b 样条，因此节点向量的每个元素是等差的，并且 4 阶均匀 b 样条的基函数我们可以直接运算好，省去了大量的递归调用！我们知道，4 阶 b 样条的每个参数 u，生成的基函数最多只有 4 个不为 0。我们把基函数的结果直接存到一个列表中。假设 t 是 u 参数的小数部分。列表为$\frac{1}{6}[-t^3 + 3t^2 - 3t + 1, 3t^3 - 6t^2 + 4, -3t^3 + 3t^2 + 3t + 1, t^3]$。最后只需要对这个四个基函数作为权值和 4 个控制点进行加权，就可以生成对应生成点。并且这 4 个生成点只受

附近的 4 个控制点的影响!

6.图元变换

1）原理

个人的理解，跟各种图元绘制算法不同，图元的变换是属于建模的领域，并不是一个绘制过程的算法。每次变换就是对模型的修改，绘制过程还是按原本的逻辑。因此，具体实现的时候，变换算法是内嵌在图元类之中的方法。

2）过程

a) 平移变换

对于点（x,y），经过平移变换（tx,ty）后

坐标变为$(\mathrm{x}+\mathrm{tx},\mathrm{y}+\mathrm{ty})$

b) 旋转变换

对于（x,y)、中心（xr,yr）和旋转角θ

坐标变为$(\mathrm{xr}+(\mathrm{x}-\mathrm{xr})\cos\theta-(\mathrm{y}-\mathrm{yr})\sin\theta,\mathrm{yr}+(\mathrm{x}-\mathrm{xr})\sin\theta+(y-yr)cos\theta)$

c) 缩放变换

对于点（x,y)、中心（xr,yr）和放缩系数 s

坐标变为$(\mathrm{xr}+(\mathrm{x}-\mathrm{xr})\mathrm{s},\mathrm{yr}+(y-yr)\ s)$

3）注意事项

变换图元时只需要修改模型即可，比如直线修改两个端点，曲线修改控制点，椭圆修改左上角点和右下角点等

7.Cohen-Sutherland 线段裁剪算法

1）原理

通过编码测试减少计算交点的次数，即编码算法。线段端点按区域赋给四位二进制码，每一个位分别表示是否在左右下上边界以外。这样给定一个点可以得到区位码。通过区位码的交和或等操作能快速判断位置关系。

2）过程

A) 计算端点区位码 code0,code1,进入循环反复判断

B) 如果 code0|code1==0，则说明全在内部，故完全保存直线，结束算法

C) 如果 code&code1==1，则说明全在外部，故完全舍弃，结束算法

D）否则有可能需要进行裁剪操作

a) 如果 code0 端点在外部，按 code0 左右下上顺序计算其和四个边界的交点情况（考虑 code0 值为 1 的位对应的的边界），用交点代替 code0 端点

b) 如果 code0 端点在内部，则 code1 端点在外部，此时按 code1 左右下上顺序计算其和四个边界的交点情况（code1 值为 1 的位对应的的边界），用交点代替 code1 端点

c) 进入 B)步骤继续循环

3）特点

适合大部分线段完全可见或完全不可见的情况

4）注意事项

循环中每次裁剪一次就要进入下一个循环，不要在一次循环中多次裁剪。比如在一次循环中裁剪了两个端点，如果两个端点分别在左边界延长线和下边界延长线的话，算法将陷入无限递归的死循环。这也是编程时遇到的一个 bug

8.Liang-Barsky 线段裁剪算法

1）原理

把 2 维问题转化为 1 纬问题。思路是通过参数表示直线的方式。一维的参数 u 能唯一确定直线上的一个点。并且当 u 取 0 和 1 的时候刚好是两个端点。直线和四个边界有四个交点（先不考虑平行情况），对这六个点进行分组，按进入边和离开边分组。进入边的交点设为 AB，离开边的交点设位 CD，则在 uA、uB、0 中取最大的作为裁剪后的始端点；从 uC、uD、1 中选最小的作为裁剪后的末端点。

2）过程

A) 输入两个端点坐标 x0、y0、x1、y1 和四个边界 xmin、xmax、ymin、ymax

B) 计算

a) p0=x0-x1; p1=-p0; 2=y0-y1; p3=-p2;

b) q0=x0-xmin; q1=xmax-x0; q2=y0-ymin; q3=ymax-y0

C) 对四个边循环判断如下

a）如果进入该边，则 u0 取 u0 和 q[i]/p[i]的最大值，并保证 u0 仍小于 u1，否则舍弃直线

b) 如果离开该边，则 u1 取 u1 和 q[i]/p[i]的最小值，并保证 u1 仍大于 u0，否则舍弃直线

c）如果和该边平行，则判断 q[i]是否>0,如果否，则说明直线在该边界外平行于此边界，故舍弃直线

D) 离开 4 次循环后根据裁剪后的 u0 和 u1 定位出新的两个端点的值，完成算法

3）特点

把 2 维转换为 1 维，使用参数表示点，偏数学的思路

4）注意事项

a) 要考虑 u0 始终小于 u1，因为直线只可能先进入矩形区域再离开矩形区域

b) 此外还要考虑直线和某个边界平行的情况，如果平行的话，和边界交点就是 4 个，而不是 6 个了

9.多边形扫描线填充算法

1）原理

找到画板每一行扫描线和多边形的交点，对交点从左到右排序，因为直线和多边形的位置关系必然是先进再出、再进再出，因此可以按奇偶分组。从奇数到偶数进行交点对的分组，每个点对之间进行填充即可。数据结构方面，需要建立有序边表 OET，其中按扫描线的行建立对应的一个个吊桶，边挂在与该边低端 y 值相同的扫描线的吊桶中。边表中的每一项表示一个边的信息，包括这个边最高的 ymax，较低点的横坐标，斜率的倒数。每个吊桶可以用链表的方式。本人使用的是列表 List 的方式。另一个数据结构活化边表 AET 记录着与当前扫描到的行相交的边的信息。刚开始为空。结束时也为空。在本图形系统中，用于给多边形进行填充。

2）过程

A) 初始化有序边表 OET，用来记录每个边。

B）找到 OET 的非空桶的最小 y 值，从这开始往纵坐标正向进行行填充

C）设置活化边表 AET 为空，把当前 y 对应的 OET 桶中的边加入 AET 表中

D）while AET 表非空 do

Begin

对 AET 表中的 xmin 值按升序排列

按照 AET 表中交点前后次序，在每对奇偶交点间的 x 段予以填充

If 扫描线 y=ymax then 从 AET 表中删除对应的边

对 AET 表中的其他边，计算其与下一行扫描线的交点：x=x+1/k（增量计算）

计算下一行扫描线 y=y+1

按照扫描线的 y 值把 OET 表中相应桶的边加入 AET 表中

End

E)结束算法

3）特点

数据结构相对复杂，实现过程的细节也比较多。但是中心思想不难，围绕着扫描线，按行进行交点匹配、填充。

4）注意事项

a) 在把边加入有序边表初始化的时候，需要解决一种特殊情况：即扫描线可能会与端点相交。如果端点是局部极值点，则不用特殊处理，但如果端点是局部极值点，需要进行端点分离，把较高的线的低端缩短一个单位。更甚的情况，如果有斜率为 0 的边，考虑这个边是不是局部最小值，如果是，则不用处理，否则把相邻的较高的边的低端点向上缩短一个单位。

b) 在活化边表计算交点的时候，采用的是增量计算的方式，极大减少了计算量

c）对于活化边表的边的排序，个人认为插入排序最适合，因为边的增量计算，大部分情况，AET 中还是维持总体有序的状态，因此插入排序比较理想。

10.区域种子填充算法

1）原理

从一种子点开始，递归的搜索附近，不断判断点是否在图元内部，如果在则染色并继续搜索，否则停止当前分支。在本绘图系统中用于给椭圆进行填充

2）过程

A) 种子压入栈（椭圆中心作为种子）

B）当栈非空

从栈中弹出一个元素

将其染色

检查附近 4 连通像素是否已被处理过或者是否在椭圆内部

如果未被处理过且在椭圆内部，加入栈，否则忽略

C）完成填充

3）特点

算法简单容易实现，适合用于方便判断点是否在图元内部的情况

4）注意事项

a) 如果直接递归处理容易导致栈溢出，因此生成一个专门的栈改成循环

b）由于椭圆的对称性，可以从原本向四个方向遍历，改成相右和上遍历，这样只需要原本 1/4 的搜索量即可完成，算是一个不小的优化。

11.Nicholl-Lee-Nicholl 线段裁剪算法

1）原理

把裁剪窗口附近划分成多个区域，通过线段斜率和区域边界斜率等方法判断两个端点所在的区域的各个情况，从而避免多次裁剪和求交计算

2）过程

A) 先计算区位码，排除肯定在区域外和内部的情况

B）确定第一个端点所在的区域。有 5 种主要情况，在裁剪框内部、在裁剪框左边、在裁剪框上边、在裁剪框左上角区域且在对角线延长线下方、在裁剪框左上角区域且在对角线延长线上方

C）从第一个端点开始向矩形裁剪框四个角发出射线，把二维平面划分，此时另一个端点的位置决定了和窗口边的相交关系

D）确定另一个端点的区域，此时知道和矩形裁剪框哪个边相交

E）求交点，确定裁剪后结果

3）特点

编码过程比较琐碎，情况分类细致。运行时避免多次裁剪计算，效率高

4）注意事项

a) 对于所有斜率可能不存在的情况都要特殊考虑。比如第一个点在矩形裁剪框左边界上，发出四个射线有两条斜率不存在，或者裁剪线段斜率不存在的情况

b）对所有边界区域要进行考虑，如线段端点刚好在对角线延长线或边界延长线上

c）对于第一个端点在右、下边界的情况或其他三个角区域的情况，都可以通过对称变换，变换到左、上、左上角区域。最后检查区位码是否要变换回来即可

12.Sutherland-Hodgeman 多边形裁剪算法

1）原理

每次用窗口的一条边界（包括延长线）对要裁剪的多边形进行裁剪。裁剪时，顺序的测试多边形的各个顶点，保留边界内部的顶点，同时删除边界外的顶点，适当时插入新的顶点，即交点和窗口顶点，从而得到一个新的多边形

2）过程

以左边界为例

A）对于每个端点进行判断：

如果上一个端点在界外且本端点在界外，则不加入点

如果上一个端点在界外，本端点在界内，则加入交点 q 和本端点

如果上一个端点在界内，本端点在界外，则加入交点 q

如果上一个端点在界内，本端点在界内，则加入本端点

B）完成后所有加入的点构成了左边界裁剪的结果

C）再分别对右上下三个边界执行 AB 过程即可。

3）特点

简单、经典且容易理解。

4）注意事项

对于第一个端点，上一个端点是末端点

12.字符图元绘制算法

1）原理

字符的显示是基于点阵字符库的。使用的点阵字库是 hzk16。单个字符，由 32 个 char 字节构成，每个字节 8 位串，则标记了 16x16 个点的像素信息矩阵。程序中直接构造了实现的字符图元的点阵，然后通过点阵信息进行绘制

2）过程

A) 根据待绘制的字符查询并设置好像素点阵

B）根据像素点阵循环按位判断是否为 1，如是则加入为待绘制的点

3）特点

绘制原理不难，主要是字库文件的搜集花了点时间

三.Gui 系统介绍

1.总体框架

1）界面通过鼠标事件接收识别用户的命令

2）程序存储 recordList，recordList 是一个二维列表，其每个元素 record 都是一个图元的列表，记录着当前画布上有哪些图元。之所以是二维列表是为了方便回溯和恢复操作。

3) 如果识别到用户的指令，需要对图元进行增删查改，则要先拷贝新建新的 record，然后对新的 record 进行修改。

4) 当 record 被修改后，建图元的模型就已经被更改了，此时只要绘制即可。框架中封装了刷新接口，传入当前所在的记录索引，刷新整个画布

5）刷新绘制过程需要调用图形学的具体算法，即通过 record 列表里每个图

元，找到需要绘制的点。算法单独作为一个模块存放在 cg_algorithms 中

5）至此完成交互、建模、绘制整个过程：即接收命令->修改模型->刷新画布->完成修改

2.模块划分

模块名称	包含类	模块功能
cg_algorithms	MyAlgorithms	具体的**图元绘制算法**，在刷新画布的时候调用
cg_cli	CliControl	对命令行文件读取和执行进行控制，是高层的接口
cg_gui	MyWindow	最高层窗口，内部包含一系列的控件，直接和用户进行交互的模块。
Figure	FigureType（枚举）、MyPoint 、MyFigure、MyLine、MyPolygon 、MyOval、MyCurve、MyPencilFigure 、MyBrushFigure 、MyEraserFigure、MyCharactor 、DrawingProcess	DrawingProcess 负责对图元对象的存储进行管理，方便回溯和恢复; 其他对象负责对图元进行建模; MyFigure 是图元高层的类，是其他图元类的父类，作为高层的接口统一进行调用。此外，**用于图元变换的方法**也封装在图元对象的内部 FigureType 是一个枚举类，标识图元类型
Mainwindow	Ui_Mainwindow	由 ui designer 界面工具生成的类，作为 MyWindow 的父类，方便界面的美化和排版
MyCanvas	MyCanvas DrawingState	MyCanvas 是画布对象，继承于 Qlabel,通过设置标签图片达到画布的效果，对画布的绘制即修改标签上图片的像素内容。 DrawingState 用于描述绘制过程的状态机，每个绘制过程都有各自的状态机，这是绘制交互的核心
Myresouce	Qt_resource_data	由 qt 工具自动生成的管理图片资源的模块
NewInputDialog	NewInputDialog	新建画布 New 的时候出现的对话框，用来输入新建画布的规格（高和宽）

3.额外功能

1）打开图片：读入已有图片文件，在此基础上进行画布绘制

2）保存和另存：即维护了文件和画布的相互联系

3）软件关于信息：查看关于画板作者的相关信息

4）铅笔：能使用铅笔操作模式在画板上绘制

5）刷子：能使用刷子操作模式在画板上绘制

6）橡皮：能使用橡皮操作模式进行擦除

7）三角形：能直接绘制三角形

8）矩形：能直接绘制矩形

9）粗细设置：能对所有绘制操作的粗细进行设置

10）选择图元：选择模式下可以对图元各个控制点进行拖动调整

11）复制粘贴：在选择模式下可以进行图元的复制粘贴

12）快捷键：复制粘贴和图元的选择切换都有快捷键

13）撤销和恢复：对每一步操作保存管理，方便回溯

14）清空画布：只清空图元而不重建画板大小，不同于新建操作

15）状态栏显示当前鼠标像素位置、操作序列和状态、画板大小

16）当退出画板时，如果有未保存的修改，会进行提示保存

17）多边形扫描线填充算法填充多边形

18）区域种子填充算法填充椭圆

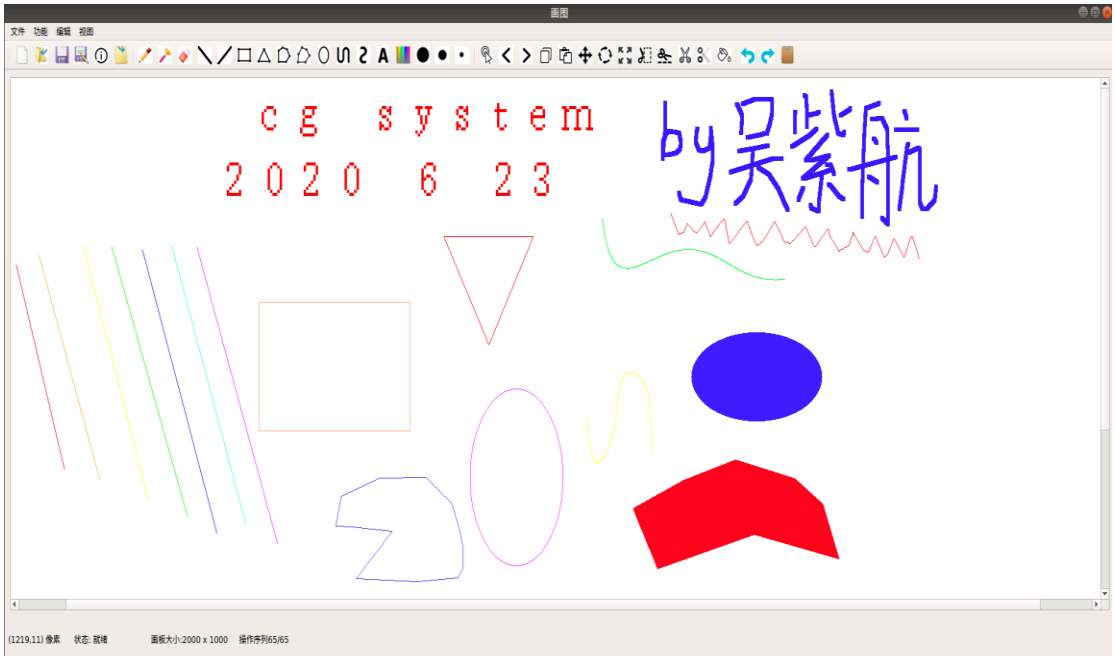
19）Nicholl-Lee-Nicholl 线段裁剪算法

20）Sutherland-Hodgeman 多边形裁剪算法

21）字符图元的绘制功能（26 个小写字母、10 个数字和空格符）

4.界面设计

Gui 界面几乎按照传统的画图软件的风格进行设计，效果如下：



四.Cli 系统介绍

Cli 系统的代码逻辑比较简单，和 gui 一样遵循着交互->建模->绘制的流程

1）封装了 CliControl 的类。Main 中接收输入文件和输出文件夹的地址，传入到 CliControl 的实例对象中。然后调用 CliControl 的 Work 方法进行任务处理。

2）Work 方法中只有一个循环，即反复的读文件下一行->执行这一行

3）执行每一行的命令的时候使用了 CliControl 的 ParseLine 方法，本质是一个 switch，根据每一行的内容，调用 CliControl 具体的指令方法

4）CliControl 中具体的指令方法有：ResetCanvas, SaveCanvas, SetColor, DrawLine, DrawPolygon, DrawOval, DrawCurve,

5）在图元绘制的时候，直接利用 gui 架构的 cg_algorithms 里的绘制算法

6) 图元变换的指令比较简单，只需要修改图元模型，因此没有单独写成方法，在 ParseLine 方法中直接修改即可。

7）Cli 系统的图元，存放在字典集合 item_dict 数据结构中。输入图元的 id 即可访问对应的图元对象

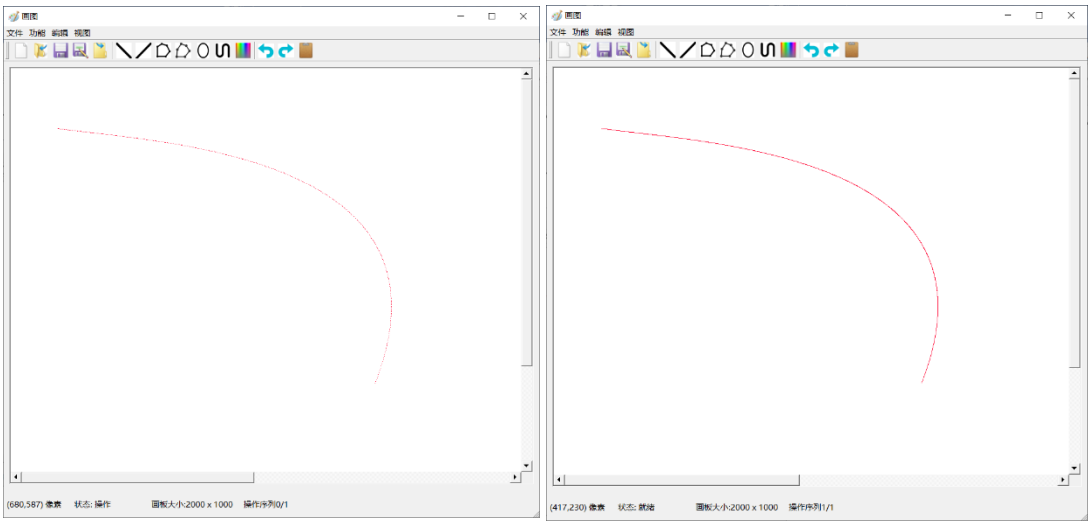
8) 在输出的时候，需要变换一下输出坐标，因为默认的坐标系 y 轴是向下为正，而要求的坐标系 y 轴是向上为正，只需要输出时把 y 变换为 height-1-y 即可

9）由于旋转算法的正方向，依据的是数学规定的正向，即当旋转角度输入为正的时候是逆时针旋转（数学上是从 x 轴向 y 轴），所以 cli 要特殊的把输入取相反数即可，这样旋转角度为正时就是逆时针旋转。

五.感受和总结

1）三月

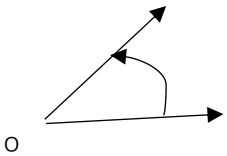
在系统实现的过程中，加深了对图形学算法的理解，顺便学习了一门新的编程语言 python，最直观的感受是 python 编程比较方便，但是代码效率比较低。比如绘制 Bezier 曲线的时候如果控制点太多就容易造成程序卡顿。如果用 c 的话速度会快很多，个人处于优化速度考虑，思考了可以在 Bezier 算法中内嵌 c 语言的方式，但是由于算法库只允许使用 math 库，因此这个方案行不通。折中的方案是在图元操作的时候使用虚化效果，等图片完成后再恢复步长密度。这样可以一定程度上简化计算又能保证结果的正确性。



绘制过程中　　绘制完成后

2）四月

实现完所有要求的图形学算法后，最明显的感受是：图形学算法需要很强的数学理解——包括微分几何、数值分析、线性代数等多个方面。实现过程的主要困难和突破并不是来自于编程细节，而是来自于对图形学算法的数学理解深度。比如在 b 样条曲线绘制算法中，**如果不能理解基函数的构造，就很难具体化的算出四阶均匀 b 样条基函数的结果矩阵**，从而很难写出高效简洁的代码了！此外，在旋转算法中，gui 程序的交互逻辑是通过中心点、鼠标拖动产生的直线，构造出两个向量如下。



o 是旋转中心，此时需要计算旋转夹角的正余弦。**通过向量的叉乘和点乘，可以直接得到旋转夹角的正弦值和余弦值，而不用调用 math 的 sin、cos 方法**，从减轻复杂度的层面，代码通过数学方法得到了优化。

3）五月

本月主要的任务是完善一些额外的功能。在图元算法方面基本没有任何改动，铅笔的思路本质也是一种图元，跟多边形类似，只是**不需要头尾连接**。刷子就是粗度较大的铅笔。三角形和矩形在构造的时候直接利用多边形的对象进行初始化即可。粗细的设置本质就是在绘制的时候将绘制点附件的矩形像素也加入待绘制列表中，达到修改粗细的效果。选择模式可以对选中的图元进行复制粘贴，同时这个模式还可以拖动修改各个图元的控制点，达到调整的目的。做这个功能最初的动机是**希望对两种曲线算法在拖动控制点时候的整体变化进行观察**。拖动的算法就是先排序找到鼠标位置最接近的那个控制点，选中后用拖动鼠标，得到位移向量，然后和平移图元的区别就是只需要移动被选中的那个控制点即可。**复制粘贴在拖动图元的时候无法进行，这样设计是为了防止回溯时候发生紊乱**。总的来说，完成的这个图形系统的功能还是比较完备的。当然还有一些可以考虑实现的功能，以后有机会还会继续完善，比如填充算法、3d 图元等等

4）六月

由于期末考试取消了，因此六月份继续的对图形学 gui 系统进行完善和补充。本月除了修复了一些隐蔽的 bug 以外，主要做的工作有三块。一个是填充算法，做了多边形扫描线填充和椭圆的种子区域填充。多边形扫描线填充算法算是实现的算法中，比较难理解的一个算法，特别是有序边表、活化边表结构的使用。种子区域填充相对就容易实现的多。第二个方面是裁剪功能，除了要求必做的两个直线裁剪算法，还增加了 Nicholl-Lee-Nicholl 直线裁剪算法和多边形 Sutherland-Hodgeman 裁剪算法。Nicholl-Lee-Nicholl 的算法原理不难，运行效率也很高，

但是编码起来感觉十分繁琐，代码长度是其他两个算法的 4-5 倍。原因是分类情况比较多、边界条件繁杂。多边形裁剪也比较容易。最后是字体图元的绘制。本来是准备做完整的汉字字体图元，但是没想好要采取怎么样的交互模式，因此最终选择了实现字母和数字。过程是先把 hzk16 字库中的字母和数字的点阵算出来，然后直接初始化到代码当中即可。为了显示更清楚，绘制时以 4x4 的像素矩阵为一个点进行绘制，否则字符太小看不清楚。此外字符的被操作态和正常形态没有区别，只能进行平移，而不能进行旋转、裁剪、放缩等其他操作。

最后，本来还有一个立体图元的功能准备实现，但是感觉放在本系统中有点突兀。因此取消了计划。总的来说，本图形学系统实现了几乎所有的常见绘图软件的功能，还是比较满意的。

六.参考资料

1. https://www.cnblogs.com/clairvoyant/p/5540023.html 中点椭圆算法原理
2. https://www.jianshu.com/p/8f82db9556d2 Bezier 算法原理部分
3. https://blog.csdn.net/morninghapppy/article/details/7344737 贝塞尔曲线算法
4. https://blog.csdn.net/qq_40597317/article/details/81155571 B 样条曲线算法
5. https://www.cnblogs.com/Penglimei/p/9690372.html Cohen-Sutherland 裁剪算法
6. https://blog.csdn.net/sinat_34686158/article/details/78745492 Liang-Barsky 裁剪算法
7. https://blog.csdn.net/xiaofengcanyuexj/article/details/17404903 多边形扫描线填充算法
8. https://blog.csdn.net/comwise/article/details/16892765 hzk16 字库
9. https://blog.csdn.net/u013467442/article/details/24935055 Sutherland-Hodgeman 算法

七.进度日志

2020.3.1-0：28　GUI 基本框架和布局的搭建；完成 DDA 直线功能
2020.3.16-1：51 完成新建画布，可以修改画布的大小
2020.3.16-18：18 完成打开图片、保存、另存、颜色设置

2020.3.16-22：01 完成退出画布、清空画布、操作的撤销和重做、退出时的保存提示
2020.3.22-17：11 完成 Bresenham 直线功能
2020.3.23-12：49 完成 DDA 多边形功能
2020.3.23-13：24 完成 Bresenham 多边形功能
2020.3.25-19：28 完成椭圆功能
2020.3.25-23：04 完成 bezier 曲线功能
2020.3.27-17：15 对正在被操作或绘制的图元进行虚化显示

2020.4.24-12：11 完成 B-spline 曲线的绘制
2020.4.24-17：50 完成图元的平移功能
2020.4.24-18：05 修复在绘制多边形过程中点击其他按钮出现的状态机紊乱 bug
2020.4.25-11：56 完成图元的旋转功能
2020.4.25-13：06 当旋转椭圆时增加无法旋转的提示框功能
2020.4.25-14：14 完成图元的放缩功能
2020.4.25-15：16 修复图元缩成一个点以后，系统发生崩溃的 bug
2020.4.27-22：00 完成 Cohen-Sutherland 直线裁剪算法
2020.4.28-1：06 修复 Cohen-Sutherland 部分情况裁剪不正确的 bug
2020.4.28-12：15 完成 Liang-Barsky 直线裁剪算法
2020.4.28-22：51 裁剪非直线图元的提示框，完成 cli 模块
2020.4.29-11：40 修复了 cli 模块的部分 bug
2020.4.30-16：53 优化了 cli 模块的代码结构，使之更美观高效

2020.5.19-11：42 完成了矩形和三角形的绘制功能
2020.5.19-18：01 完成了铅笔功能、画板关于信息、粗细设置
2020.5.19-22：08 完成了刷子和橡皮功能
2020.5.21-14：44 完成了选择模式的复制粘贴功能（包括快捷键）
2020.5.21-15：49 完成了选择模式的对各个控制点进行拖动调整的功能
2020.5.21-16：22 修复了 cli 系统坐标系方向和默认绘制方向不同的 bug，新的方向和用户的视觉方向相一致（y 轴向上，角度顺时针为正）
2020.6.18-23：45 完成了多边形扫描线填充算法
2020.6.19-21：30 完成了填充椭圆的区域种子填充算法，修复了多边形扫描线填充算法出现毛刺的 bug（插入排序出了小问题）；修复了染色模式和选择模式操作序号延迟显示的 bug。
2020.6.20-22：18 完成 Nicholl-Lee-Nicholl 直线裁剪算法的调用接口
2020.6.21-17：29 完成 Nicholl-Lee-Nicholl 直线裁剪算法
2020.6.22-13：18 完成 Sutherland-Hodgman 多边形裁剪算法的接口和对左右边界裁剪的处理
2020.6.22-14：03 完成 Sutherland-Hodgman 多边形裁剪算法并修复三角形绘制时出现浮点数导致 bug
2020.6.22-19：18 修复多边形裁剪算法中逐点裁剪时对首尾顶点的线段处理错误导致的 bug
2020.6.22-22：03 完成了字符图元的绘制功能（26 个小写字母、10 个数字和空格）
2020.6.23-22：44 修复无法设置旋转点的 bug