《计算机图形学》系统报告

181860155 朱晓晴 heloize@126.com

2020 年 12 月

1 综述

在 9 月提交中，主要完善了 cg_cli.py 和 cg_algorithms.py 两个模块。在 cg_cli.py 中，补充了对 drawPolygon、drawEllipse 等指令的解析。在 cg_algorithms.py 中，完成了线段绘制算法（draw_line 函数）和椭圆绘制算法（draw_ellipse 函数）。

在 10 月提交中，主要完善了 cg_cli.py 和 cg_algorithms.py 两个模块。在 cg_cli.py 中，补充了对 drawCurve、translate、rotate、scale 和 clip 等指令的解析。在 cg_algorithms.py 中，完成了曲线绘制算法（draw_curve 函数）、平移算法（translate 函数）、旋转算法（rotate 函数）、缩放算法（scale 函数）和线段裁剪算法（clip 函数）。至此，所有指令的算法已全部完成。

在 11 月提交中，主要完成了 cg_gui.py 中图形界面的基本功能，即将 cg_cli.py 中的所有操作可视化。同时，根据图形界面的显示需要，在 cg_algorithms.py 中添加了相应的函数，例如 draw_polygon_gui。至此，要求实现的功能已全部完成。

在 12 月提交中，对已有系统进行了优化，如修复 bug、修改图形界面交互等。此外，对系统功能进行了拓展，添加了多边形填充、多边形裁剪等功能。

2 算法介绍

2.1 算法原理

2.1.1 绘制线段

要求：根据给定两点 (x_0, y_0) 和 (x_1, y_1) 绘制线段。

绘制线段共需完成 2 种算法：DDA 算法和 Bresenham 算法。

DDA 算法

斜率 $m = \frac{y_1 - y_0}{x_1 - x_0}$，若 $|m| \leqslant 1$，以 $x_0 < x_1$ 为例进行说明。以单位间隔 $(x_{k+1} - x_k = 1)$ 对 x 进行采样，并计算对应的 y 值:

$y_{k+1} = y_k + m \ (k = 0, 1, ...)$

若 $|m| > 1$，以 $y_0 < y_1$ 为例进行说明。以单位间隔 $(\Delta y = 1)$ 对 y 进行采样，并计算对应的 x 值:

$x_{k+1} = x_k + \frac{1}{m} \ (k = 0, 1, ...)$

Bresenham 算法

将 (x_0, y_0) 作为第一个点，$m \geqslant 0$，决策参数初值为 $p_0 = 2\Delta y - \Delta x$；$m < 0$，决策参数初值为 $p_0 = 2\Delta y + \Delta x$。

$|m| \leqslant 1$ 时，以 $x_0 < x_1$ 的情况为例进行说明。

若 $m \geqslant 0$，在每个 x_k 处进行检测 p_k：

$p_k \geqslant 0$，下一个点为 $(x_{k+1}, y_k + 1)$，下一个决策参数 $p_{k+1} = p_k + 2\Delta y - 2\Delta x$；

$p_k < 0$，下一个点为 (x_{k+1}, y_k)，下一个决策参数 $p_{k+1} = p_k + 2\Delta y$。

若 $m < 0$，在每个 x_k 处进行检测 p_k：

$p_k \geqslant 0$，下一个点为 (x_{k+1}, y_k)，下一个决策参数 $p_{k+1} = p_k + 2\Delta y$；

$p_k < 0$，下一个点为 $(x_{k+1}, y_k - 1)$，下一个决策参数 $p_{k+1} = p_k + 2\Delta y + 2\Delta x$。

$|m| > 1$ 时，以且 $y_0 < y_1$ 的情况为例进行说明。

若 $m \geqslant 0$，在每个 y_k 处进行检测 p_k：

$p_k \geqslant 0$，下一个点为 $(x_k + 1, y_{k+1})$，下一个决策参数 $p_{k+1} = p_k + 2\Delta x - 2\Delta y$；

$p_k < 0$，下一个点为 (x_k, y_{k+1})，下一个决策参数 $p_{k+1} = p_k + 2\Delta x$。

若 $m < 0$，在每个 y_k 处进行检测 p_k：

$p_k \geqslant 0$，下一个点为 (x_k, y_{k+1})，下一个决策参数 $p_{k+1} = p_k + 2\Delta x$；

$p_k < 0$，下一个点为 $(x_k - 1, y_{k+1})$，下一个决策参数 $p_{k+1} = p_k + 2\Delta x + 2\Delta y$。

2.1.2 绘制椭圆

要求：根据给定的椭圆矩形包围框左上角坐标 (x_0, y_0) 和右下角坐标 (x_1, y_1) 绘制椭圆。

中点圆生成算法

计算出椭圆中心 (x_c, y_c)，长短轴径 r_x 和 r_y。

（1）区域 1 中（| 切线斜率 |⩽ 1）

计算得中心在原点的椭圆上的第一个点 $(0, r_y)$。在区域 1 每个 x_k 处，计算相应的决策参数：

$$p1_k = r_y^2(x_k + 1)^2 + r_x^2(y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$

并对决策参数进行检测：

$p1_k < 0$，下一个点为 (x_{k+1}, y_k)；

$p1_k \geqslant 0$，下一个点为 $(x_{k+1}, y_k - 1)$。

循环直到 $2r_y^2 x \geqslant 2r_x^2 xy$。

（2）区域 2 中（| 切线斜率 |> 1）

在区域 2 每个 y_k 处，计算相应的决策参数：

$$p2_k = r_y^2(x_k + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

并对决策参数进行检测：

$p2_k \leqslant 0$，下一个点为 (x_k, y_{k+1})；

$p2_k > 0$，下一个点为 $(x_k - 1, y_{k+1})$。

循环直到 $y = 0$。

最后，计算出其他三个象限中的点，并将所有的点平移到中心为 (x_c, y_c) 的椭圆轨迹上。

2.1.3 绘制曲线

要求：根据给定的若干控制点绘制曲线。

绘制曲线共需完成 2 种算法：Bezier 算法和 B-spline 算法。

Bezier 曲线

给定任一参数 u，$u \in [0,1]$，利用 de Castelijau 递推算法来产生曲线上的点。计算公式为：

$$P_i^r = \begin{cases} P_i & r = 0 \\ (1-u)P_i^{r-1} + uP_{i+1}^{r-1} & r = 1,2,...,n;\ i = 0,1,...,n-r \end{cases}$$

$r = 0$ 时，对应的顶点是曲线的控制点；r 不断增加时，每两个顶点生成一个新的顶点，对应的顶点数递减，直到只剩下一个顶点。

在 [0,1] 内对 u 取值，对任一 u 的取值，运行 de Castelijau 递推算法，得到 Bezier 曲线上的一个点。最后，即可得到 Bezier 曲线。

B-spline 曲线

实验要求 B-Spline 绘制出的曲线为三次均匀 B 样条曲线，设共给定 $n+1$ 个控制点。

在定义域 $[u_3, u_{n+1}]$ 中对 u 取值，对任一 u 的取值，利用 deBoox-Cox 递推公式

$$B_{i,k}(u) = [\frac{u - u_i}{u_{i+k-1} - u_i}]B_{i,k-1}(u) + [\frac{u_{i+k} - u}{u_{i+k} - u_{i+1}}]B_{i+1,k-1}(u)$$

$$B_{i,1}(u) = \begin{cases} 1 & u \in [u_i, u_{i+1}] \\ 0 & u \notin [u_i, u_{i+1}] \end{cases}$$

计算出每个顶点的 B-Spline 基函数 $B_{i,k}(u)$。再根据 B-Spline 曲线公式

$$P(u) = \sum_{i=0}^{n} P_i B_{i,k}(u), u \in [u_3, u_{n+1}]$$

计算得曲线上的某一点。最后，即可得到 B-Spline 曲线。

2.1.4 图元平移

要求：根据给定的平移向量 (dx, dy) 平移指定图元。

对于指定图元的任一图元参数 $P_1(x_1, y_1)$，根据以下公式计算出新点 $P_2(x_2, y_2)$ 的坐标：

$$\begin{cases} x_2 = x_1 + dx \\ y_2 = y_1 + dy \end{cases}$$

2.1.5 图元旋转

要求：根据给定的旋转中心 (x, y) 和顺时针旋转角度 r 旋转指定图元。

首先，以 (x, y) 为原点旋转图元。对于指定图元的任一图元参数 $P_1(x_1, y_1)$，根据以下公式计算出点 $P_2(x_2, y_2)$ 的坐标：

$$\begin{cases} x_2 = (x1 - x)cos(-r) - (y1 - y)sin(-r) \\ y_2 = (x1 - x)sin(-r) + (y1 - y)cos(-r) \end{cases}$$

再以 (x,y) 为平移向量，平移 $P_2(x_2,y_2)$ 得到新点 $P_3(x_3,y_3)$：

$$\begin{cases} x_3 = x_2 + x \\ y_3 = y_2 + y \end{cases}$$

2.1.6 图元缩放

要求：根据给定的缩放中心 (x,y) 和缩放倍数 s 缩放指定图元。

首先，以 (x,y) 为原点缩放图元。对于指定图元的任一图元参数 $P_1(x_1,y_1)$，根据以下公式计算出点 $P_2(x_2,y_2)$ 的坐标：

$$\begin{cases} x_2 = (x1 - x) * s \\ y_2 = (x1 - x) * s \end{cases}$$

再以 (x,y) 为平移向量，平移 $P_2(x_2,y_2)$ 得到新点 $P_3(x_3,y_3)$：

$$\begin{cases} x_3 = x_2 + x \\ y_3 = y_2 + y \end{cases}$$

2.1.7 线段裁剪

要求:根据给定的裁剪窗口的左上角顶点坐标 (x_{min},y_{max}) 和右下角顶点坐标 (x_{max},y_{min}) 裁剪指定线段。

线段裁剪共需完成 2 种算法：Cohen-Sutherland 算法和 Liang-Barsky 算法。

Cohen-Sutherland 算法

Cohen-Sutherland 算法的流程如图 1 所示，对线段端点 (x_0,y_0) 和 (x_1,y_1) 编码的规则如图 2 所示。

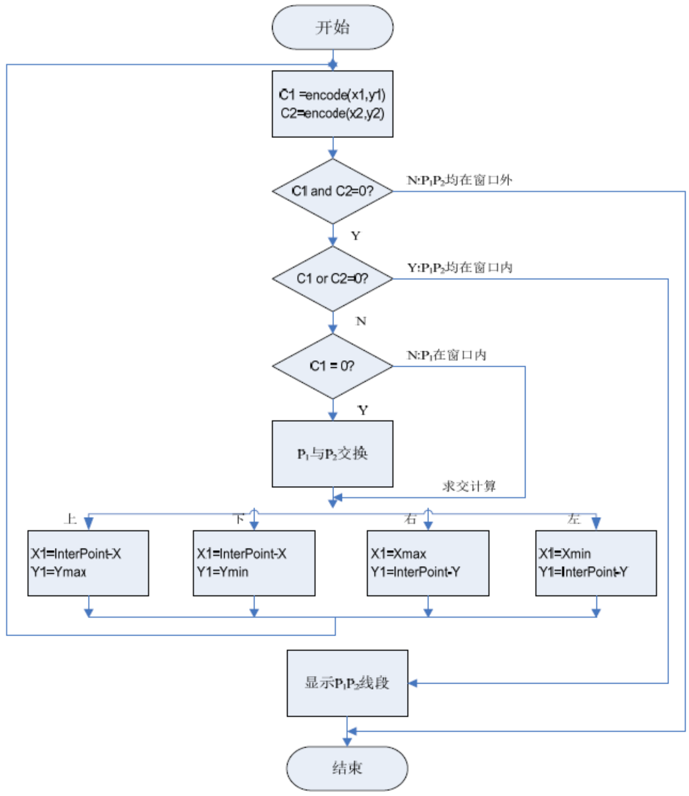


图 1: Cohen-Sutherland 算法流程图

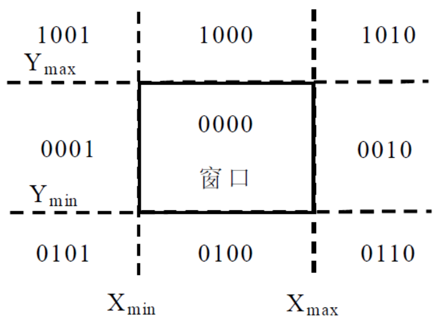


图 2: 区域编码

Liang-Barsky 算法

定义如下 8 个参数：

$$\begin{cases} p_1 = -\Delta x, q_1 = x_0 - x_{min} \\ p_2 = \Delta x, q_2 = x_{max} - x_0 \\ p_3 = -\Delta y, q_3 = y_0 - y_{min} \\ p_4 = \Delta y, q_4 = y_{max} - y_0 \end{cases}$$

初始化 $u_1 = 0$，$u_2 = 1$。对任一组 p_k 和 q_k，做如下检测：

$p_k = 0$ 时，若 $q_k < 0$，线段在裁剪窗口外，舍弃该线段，算法结束；

$p_k < 0$ 时，令 u_1 为其本身和 $\frac{q_k}{p_k}$ 中的较大值；

$p_k > 0$ 时，令 u_2 为其本身和 $\frac{q_k}{p_k}$ 中的较小值。

参数更新后，若 $u_1 > u_2$，舍弃该线段，算法结束。

2.1.8 多边形填充

要求：给定多边形顶点坐标列表，填充该多边形（返回需着色的内部像素点集合）。

多边形填充采用扫描转换填充算法，该算法的基本处理方式为：对于每条穿过多边形的扫描线，计算出该扫描线与多边形各边的交点。一般情况下，存在偶数个这样的交点（若共享顶点的两条边位于扫描线同侧，将该点记为两个交点）。将这些交点从左至右配对，给每对交点区间内的像素点着色即可。

利用多边形各边的连贯特性，计算交点时可采用增量计算法来减少计算量。如图 3 所示，两条相邻扫描线间 y 坐标的变化为：

$$y_{k+1} - y_k = 1$$

扫描线交点的横坐标值 x_{k+1} 可从前一条扫描线交点的横坐标 x_k 计算得：

$$x_{k+1} = x_k + \frac{1}{m}$$

其中，m 为 (x_k, y_k) 和 (x_{k+1}, y_{k+1}) 所在边的斜率。

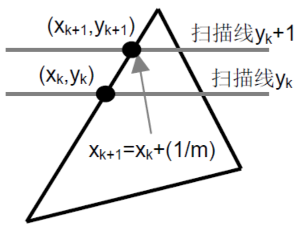


图 3: 边的连贯性

多边形扫描转换填充算法的伪代码如下：

1. 建立有序边表 NET。对于每个多边形顶点（按顺时针或逆时针顺序给出），如果该点的纵坐标小于前一个顶点的纵坐标，则将三元组（该点横坐标，前一个点的纵坐标，两点确定的直线斜率的倒数），即 $(x, ymax, k)$，加入该点纵坐标对应的扫描线在 NET 中的表项中。对后一个顶点进行同样的操作。

2. 对每一条扫描线（自下而上），用增量计算法更新活性边表 AET 表项的 x 值。

3. 删除 $ymax$ 等于当前扫描线 y 值的 AET 表项。

4. 如果扫描线对应的 NET 表项非空，将其中的三元组合并到 AET 中，并按 x 值升序对 AET 中所有三元组排序。

5. 给 AET 中排好序的三元组配对，将每对交点间的像素点加入需着色像素点的集合。返回第 2 步，直到遍历完所有扫描线。

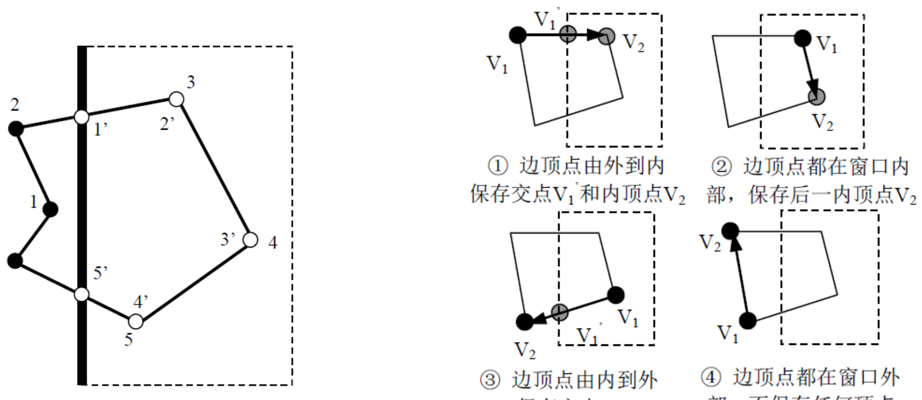
2.1.9 多边形裁剪

要求：给定多边形顶点坐标列表和裁剪窗口对角顶点坐标，返回裁剪后的多边形顶点坐标列表。

多边形裁剪采用了 Sutherland-Hodgman 算法，该算法对裁剪窗口的每一条边进行如下处理：

1. 如图 4(a) 所示，以裁剪窗口的某一条边为界，将画布区域分为内侧和外侧。

2. 如图 4(b) 所示，按序处理原多边形的每一条边，输出裁剪后的顶点坐标列表。若后续要继续处理其他裁剪窗口便捷，将此步输出的顶点列表作为新的输入。



(a) 输出顶点的确定方式

(b) 输出顶点的确定规则

图 4: 裁剪顶点的处理

2.2 对比分析

2.2.1 绘制线段

Naive 算法以单位间隔对 x 取样，并根据直线方程计算出相应的 y 坐标，以此得到线段上所有的点。这一过程需要做大量乘法运算，算法运行速度慢，且对硬件要求较高。此外，Naive 算法不对斜率绝对值大于 1 和小于 1 的两种情况区别处理，而是统一地以单位间隔对 x 取样。因而，当斜率绝对值大于 1 时，易出现直线取样点稀疏的问题。

DDA 算法针对上述缺陷做出了优化，它利用光栅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逐步推导出各像素点的位置，两种算法的对比如图 5 所示。然而，DDA 算法仍有不足之处，算法中的浮点运算和取整操作十分耗时，且取整误差的积累会导致长线段的像素位置偏离实际位置。

Bresenham 算法在每个已确定的像素点处计算决策参数，选择距离实际线段较近的点作为下一个绘制的点。相较于 DDA 算法，这一做法有利于控制绘制出的直线对实际线段的偏离程度。

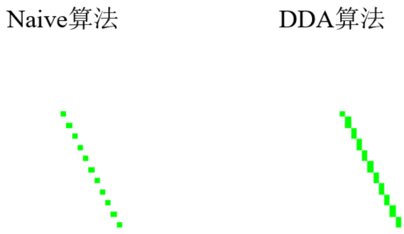


图 5: Naive 算法和 DDA 算法效果对比

2.2.2 绘制曲线

绘制曲线使用了 Bezier 曲线和 B-Spline 曲线，两者在基函数和局部控制能力上存在较大差异。

在基函数方面，Bezier 曲线基函数的次数始终等于控制顶点数减 1，灵活性不足。而 B-Spline 曲线基函数的次数则与控制顶点数无关。

在局部控制能力上，修改 Bezier 曲线的某一个控制顶点，会影响整条曲线的走向，Bezier 曲线在局部性上有所欠缺。而修改 B-Spline 曲线的某一个控制顶点，只影响曲线的某一部分，B-Spline 曲线的局部性较好。

此外，在开发系统图形界面的过程中，相较于其他绘制和编辑图元的算法，曲线绘制算法的性能较差，尤其是 B-Spline 算法。在图形界面的刷新频率下，若单条曲线的控制点较多，或是曲线数目较多，系统易出现卡顿。

在开发过程中，考虑过优化曲线绘制算法性能。查阅相关资料后，得知通过分段绘制低阶 Bezier 曲线，可以显著减少浮点运算的次数。但是，鉴于正常情况下在本系统中绘制的曲线阶数应当不会超过 20 阶，优化后的算法性能并不会显著提升。此外，若为了进一步优化算法性能，仅令分段后的曲线具有一阶连续性，那么曲线的局部性将会加强，但平滑程度显著降低。因此，本系统仍然沿用 2.1.3 节中介绍的曲线绘制算法，不再另做优化。

2.2.3 多边形裁剪

本系统的多边形裁剪算法采用 Sutherland-Hodgman 算法，此算法能够正确处理对凸多边形的裁剪。但对于凹多边形，则可能出现如图 6 所示的冗余线段。

出现这一情况的原因是 Sutherland-Hodgman 算法的输出为一个多边形，而不考虑裁剪后留下多个多边形的情况。因此，图 6(b) 中的两个多边形之间存在冗余线段。修正此算法的常见方法有对每条裁剪窗口边界进行拓扑检测，或是改用 Weiler-Atherton 双边裁剪算法，本系统暂未对 Sutherland-Hodgman 算法进行修正。

(a) 裁剪前

(b) 裁剪后

图 6: Sutherland-Hodgman 算法缺陷

系统未实现 Weiler-Atherton 双边裁剪算法，仅作简要介绍。Weiler-Atherton 算法按顺时针顺序建立多边形顶点表和裁剪窗口顶点表。对多边形的每一条边，求其与裁剪窗口的交点，将交点标记为“入点”或“出点”，并将交点插入到多边形顶点表和裁剪窗口顶点表的相应位置。

然后，任取一个未追踪过的入点，沿多边形顶点表和裁剪窗口顶点表追踪出该点所在的多边形。重复这一步骤，直到不存在未追踪过的入点。最终，得到裁剪后一个或多个多边形的顶点列表。

3 系统介绍

3.1 交互逻辑

系统沿用 demo 的框架，即命令行和图形界面两种运行方式。

以命令行方式运行时，系统逐行读取指令文件，解析指令并调用 cg_algorithms.py 模块中的相关算法。

以图形界面方式运行时，通过鼠标事件获取所需参数，调用 cg_algorithms.py 模块中的相关算法，再以可视化方式呈现绘制或编辑结果。

下面简要介绍图形界面各个功能的实现逻辑，各功能的详细交互方式参见系统说明书。

重置画布

利用 QDialog 类生成输入窗口（见图 7），得到输入的数据后，首先清除画布中的所有图元，然后根据获得的数据重新设置画布的宽度和高度。

图 7: 输入窗口

保存画布

调用 QFileDialog.getSaveFileName 函数，显示出常见的文件保存界面（见图 8）。再调用 QGraphicsView.grab 函数截取画布，并将返回的数据保存为位图。

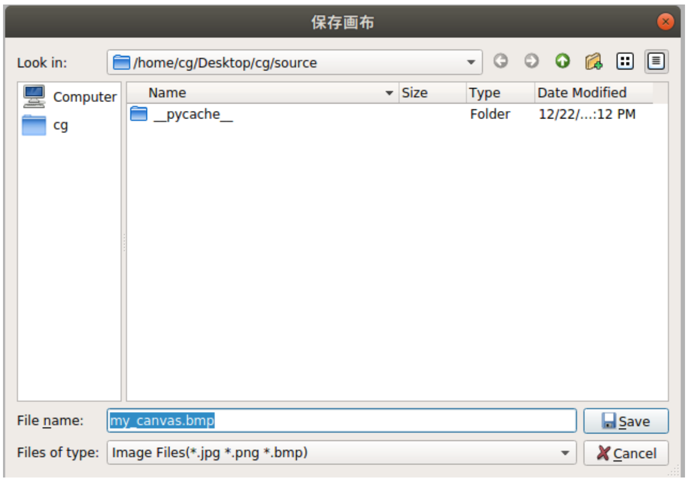


图 8: 文件保存窗口

设置画笔

调用 QColorDialog.getColor 函数，显示出常见的调板色界面（见图 9），并记录下函数返回值，作为系统当前的画笔颜色。为 MyItem 类添加成员变量 color，以记录图元颜色。新建 MyItem 实例时，令该图元的颜色为系统当前的画笔颜色，即可实现要求的功能。

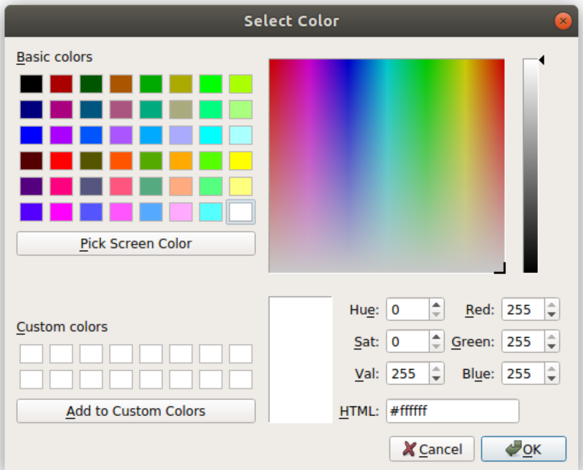


图 9: 颜色选择面板

绘制线段

点击鼠标时，将当前坐标作为线段的两个端点。鼠标移动时，将线段第二个端点更新为当前坐标。将两个端点的坐标作为参数传给 alg.draw_line 函数，得到线段。

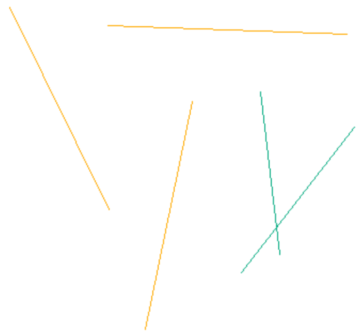


图 10: 绘制线段示例

绘制矩形

点击鼠标时，将当前坐标作为矩形的两个对角顶点。鼠标移动时，将第二个对角顶点更新为当前坐标。确定对角顶点后，计算出矩形的四个顶点，作为参数传给 alg.draw_polygon 函数，得到矩形。

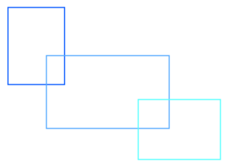


图 11: 绘制矩形示例

绘制多边形

点击鼠标时，如果当前的点在第一个顶点为圆心半径为 10 的圆内，且当前已确定的顶点数不小于 3，则认为已经绘制完毕；否则，把当前坐标加入多边形的顶点列表。鼠标移动时，将顶点列表的最后一个坐标更新为当前坐标。

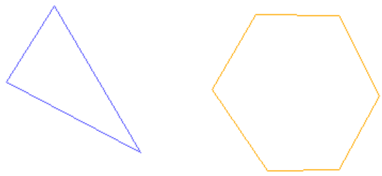


图 12: 绘制多边形示例

绘制椭圆

点击鼠标时，将当前坐标作为椭圆绑定矩形的对角顶点坐标。鼠标移动时，将第二个对角顶点坐标更新为当前坐标。将两个顶点的坐标作为参数传给 alg.draw_ellipse 函数，得到椭圆。

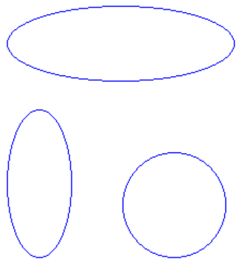


图 13: 绘制椭圆示例

绘制曲线

点击鼠标时，把当前坐标加入曲线的控制点列表。鼠标移动时，将控制点列表的最后一个坐标更新为当前坐标。鼠标点击其他绘制或编辑功能时，绘制结束。绘制曲线或曲线被选中时，会显示曲线控制点。

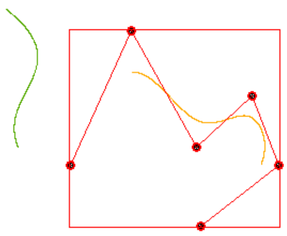
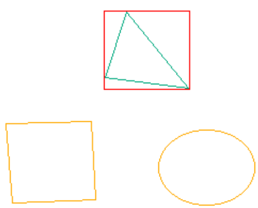


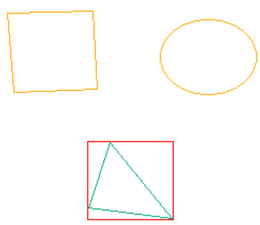
图 14: 绘制曲线示例

平移

点击鼠标时，若当前左边在选中图元的绑定矩形内，将当前坐标作为平移向量的起点和终点坐标。鼠标移动时，将平移向量的终点坐标更新为当前坐标。将平移向量的坐标作为参数传给 alg.translate 函数，得到平移后的图元。



(a) 平移前

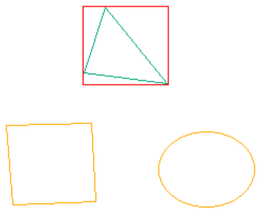


(b) 平移后

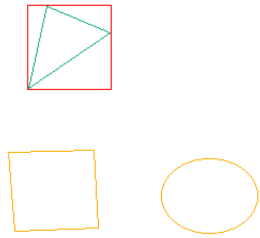
图 15: 平移示例

旋转

将图元绑定矩形的中心作为旋转中心，点击鼠标时将当前坐标作为旋转弧线的起点和终点，移动鼠标时将弧线的终点更新为当前坐标。将旋转中心坐标和根据弧线计算出的旋转角度作为参数传给 alg.rotate 函数，得到旋转后的图元。



(a) 旋转前

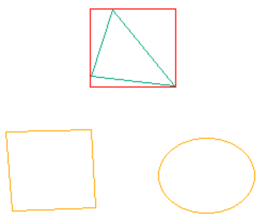


(b) 旋转后

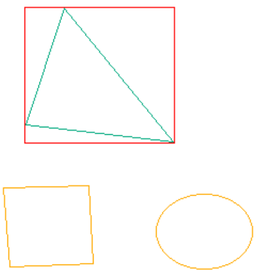
图 16: 旋转示例

缩放

将图元绑定矩形的中心作为缩放中心，点击鼠标时将当前坐标作为缩放线段的起点和终点，移动鼠标时将缩放线段的终点更新为当前坐标。令缩放倍数等于缩放线段长度与 80 的比值，将缩放中心坐标和缩放倍数作为参数传给 alg.scale 函数，得到缩放后的图元。



(a) 缩放前

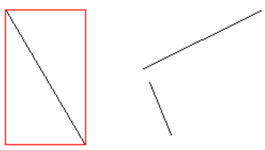


(b) 缩放后

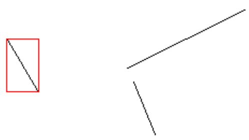
图 17: 缩放示例

裁剪

点击鼠标时，将当前坐标作为裁剪窗口的对角顶点坐标。鼠标移动时，将裁剪窗口的第二个对角顶点坐标更新为当前坐标。将两个顶点坐标作为参数传给 alg.clip 函数，得到裁剪后的线段。



(a) 裁剪前

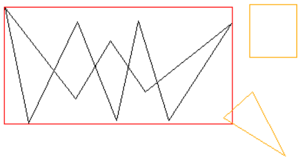


(b) 裁剪后

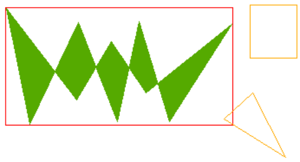
图 18: 线段裁剪示例

多边形填充

将选中多边形的 fill 标记置为 True。多边形的 paint 函数被调用时，如果 fill 标记为 True，会调用 alg.polygon_fill 函数，得到需要着色的内部像素点列表，并将其着色。



(a) 填充前



(b) 填充后

图 19: 多边形填充示例

多边形裁剪

点击鼠标时，将当前坐标作为裁剪窗口的对角顶点坐标。鼠标移动时，将裁剪窗口的第二个对角顶点坐标更新为当前坐标。将两个顶点坐标作为参数传给 alg.polygon_clip 函数，得到裁剪后的多边形。

(a) 裁剪前

(b) 裁剪后

图 20: 多边形裁剪示例

删除

将选中图元从 scene 中删除即可。

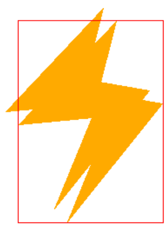
复制

将选中图元添加到画布剪切板，待后续粘贴。

粘贴

根据剪切板中的图元的信息（图元类型、顶点列表、颜色等），新建新的图元。新图元的位置较原图元的位置有一定偏差，用于区分两个图元。

(a) 粘贴前



(b) 粘贴后

图 21: 粘贴示例

3.2 设计思路

系统设计有两大主要方向：使系统功能更强大、使系统使用更便捷。

在拓展系统功能方面，本系统在实现基本功能的基础上，添加了绘制矩形、多边形填充、多边形裁剪、复制和粘贴等附加功能。

在优化用户体验方面，本系统在图形界面中进行了若干优化，如实现鼠标点选图元、显示曲线控制点等，优化细节参见 4.3 节。

4 补充内容

4.1 附加功能

系统在图元绘制、图元编辑和交互方式三个方面实现了以下附加功能：

图元绘制：绘制矩形

图元编辑：多边形填充、多边形裁剪、删除图元、复制、粘贴

交互方式：鼠标点选

其中，图元绘制和图元编辑中添加的功能已在 3.2 节中详细介绍交互逻辑，具体的交互操作参见系统说明书。此处将对鼠标点选功能进行简要介绍：当系统空闲时，即不在进行系统设置、图元绘制和图元编辑操作时，点击画布任意处，如果该处有图元，该图元被选中并显示绑定矩形。

4.2 界面设计

本系统保留了 demo 中的菜单栏，并添加了工具栏（见图 22）。工具栏提供了本系统所有功能的接口，以图标呈现，易于理解且便于用户操作。

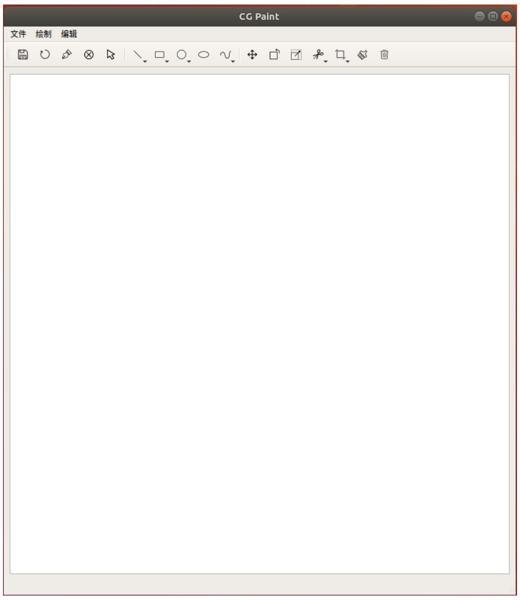


图 22: 图形界面

4.3 交互设计

本节将介绍系统图形界面若干易用或有特色的交互。

鼠标点选：本系统取消了 demo 中的图元列表，支持鼠标点选图元。

连续绘制、编辑：参考现有的商用画图系统，本系统支持用户连续绘制、编辑图元（除线段裁剪、多边形裁剪外）。

工具栏：demo 中以菜单栏实现对系统功能的选择，但操作不够简便。因此，本系统添加了工具栏，以图标形式展示系统功能。其中，系统设置和图元绘制功能基本沿用常见画图系统的图标，方便用户识别、选择。需要特别提及的是鼠标点击图标，即工具栏左起第五个图标。本系统支持用户连续绘制、编辑图元，点击这一图标用户即可使系统进入空闲状态，若多边形、曲线等图元未绘制完成，系统将自动处理。系统进入空闲状态后，用户可通过鼠标点击选择图元。

矩形绘制：通过多边形绘制也可绘制出矩形，但水平、垂直线段不易通过鼠标绘制出，因此系统添加了绘制矩形的功能。矩形绘制没有沿用多边形绘制时逐个点击顶点的交互方式，而是直接拖拽出矩形对角线即可。然而，矩形未绘制结束时，系统并不以多边形的形式保存它，因为矩形形状和位置的频繁变动与多边形绘制算法不适配，会导致画布显示的矩形有偏差。因此，系统先将该图元以矩形类型暂存，只保存对角顶点坐标，待绘制完毕再修改为多边形类型。

多边形绘制：多边形绘制时，仅当用户再次点击多边形第一个顶点（允许一定误差）时，系统才认为多边形绘制结束。尽管此交互需要用户额外点击一次，且存在点击误差较大导致绘制未按预期结束的情况，但此交互便于用户观察多边形绘制进度，以及各顶点间的位置关系。

曲线绘制：曲线绘制或曲线被选中时，系统会显示曲线的控制点，便于用户观察控制点位置，以及控制点对曲线的影响。

平移：鼠标拖出的平移向量的起点在图元绑定矩形内，平移操作才会成功。

旋转、缩放：起初，在图形界面旋转或缩放图元时，用户需要先指定一个旋转或缩放中心，在拖拽鼠标进行旋转或缩放。尽管上述交互符合实验要求，但操作不简易，且与常见画图系统差异较大，不易于用户操作。因此，最后选择以图元绑定矩形的中心为旋转、缩放中心。

5 总结

在系统开发过程中，运用了目前已讲到图形学基础知识，并自学了解了若干绘制图元和编辑图元的算法，以及图形界面开发的知识。通过实现所有指令的算法和图形界面，加深了对图形显示原理的理解。同时，通过反复测试找出了之前提交代码中的若干错误，如公式错误、功能异常等。在纠正代码错误的过程中，加深了对算法公式推导过程和 Python 语言的理解程度。

此外，为了使得系统界面更加美观、交互更加易用，开发过程中也做出了较多测试和修改，使得系统更接近现有的常见商用画图系统。这一过程并不因图形界面的简易而变得轻松，加深了我对图形界面开发的了解。

参考文献

[1] 孙正兴等. 计算机图形学教程 [M]. 机械工业出版社, 2006.

[2] Qt 5.15 手册
https://doc.qt.io/qt-5/index.html

[3] 计算向量夹角
https://blog.csdn.net/qq_42423940/article/details/83757427

[4] QListWidget 右键菜单
https://blog.csdn.net/qq_42365249/article/details/106651354

[5] PyQt 5 菜单栏、工具栏和状态栏使用
https://www.cnblogs.com/ygzhaof/p/10070523.html

[6] QToolButtton 简介
https://blog.csdn.net/qq_17351161/article/details/102917153

[7] 多边形填充算法
https://blog.csdn.net/keneyr/article/details/83747501

[8] 多边形裁剪 Weiler-Atherton 算法
https://blog.csdn.net/yangxi_pekin/article/details/37738219