《计算机图形学》12 月报告

181860106，王梓博，181860106@smail.nju.edu.cn

2020 年 12 月 6 日

目录

1 综述

本实验要求跟随课程进度在项目中实现各种图形学算法，最终完成一个完整的图形学系统，进度自由安排。在 12 月，我的进度如下：

- 核心算法模块 cg_algorithm.py 已完成;
- 命令行界面（CLI）程序 cg_cli.py 已完成;
- 用户交互界面（GUI）程序 cg_gui.py 已完成;

2 算法介绍

2.1 绘制线段

2.1.1 DDA 算法

DDA 算法主要是利用了增量的思想，通过同时对 x 和 y 各增加一个小增量，计算下一步的 x 和 y 的值。由于已知线段的两个端点，因此我们很容易可以计算出直线方程 $y = kx + b$ 中 k 和 b 的值。当线段的斜率的绝对值小于 1 时，线段在 y 方向上的增长速度小于在 x 方向上的增长速度，因此令 $dx = 1$，可根据下式按顺序得到与每个 x 对应的 y 值。

$$y_i = kx_i + b \tag{1}$$

$$y_{i+1} = kx_{i+1} + b = kx_i + b + kdx = y_i + kdx = y_i + k \tag{2}$$

而当线段的斜率的绝对值大于 1 时，线段在 y 方向上的增长速度大于在 x 方向上的增长速度，因此令 $dy = 1$，可根据下式按顺序得到与每个 y 对应的 x 值。

$$x_i = \frac{y_i + b}{k} \tag{3}$$

$$x_{i+1} = \frac{y_{i+1} + b}{k} = \frac{y_i + dy + b}{k} = x_i + \frac{dy}{k} = x_i + \frac{1}{k} \tag{4}$$

通过上述两式，只需要对 x 和 y 不断递增就可以得到下一个像素点的坐标值，这样可以避免对每个像素点都调用直线方程计算带来的浮点乘法运算开销。

2.1.2 Bresenham 算法

DDA 算法尽管消除了浮点数乘法运算，但仍存在浮点数加法和取整操作，效率仍有待提高。而 Bresenham 提出的直线生成算法只用了整数的增量运算，借助一个误差量的符号确定下一个像素点的位置，大大提高了运行效率，成为时至今日图形学领域使用最广泛的直线生成算法。

设线段所在直线方程为

$$y = kx + b \tag{5}$$

先考虑斜率在 0 1 之间的情况，从线段的左端点开始，按步长 dx 为 1 进行处理，每确定当前的像素坐标 (x_i, y_i) 后，在下一个点 (x_{i+1}, y_{i+1})，由于斜率在 0 1 之间，y_{i+1} 的值要么为

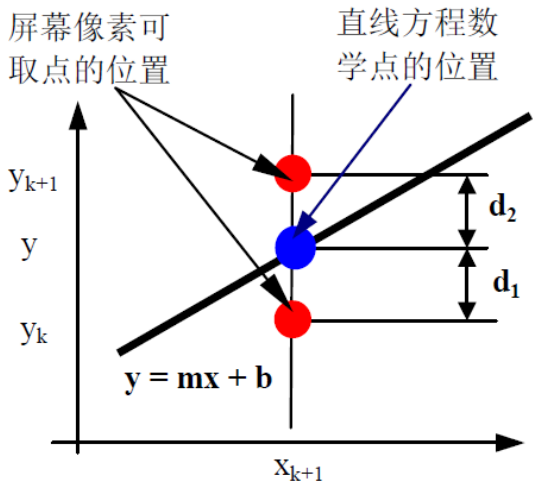


图 1: 取样位置 x_{i+1} 处侯选像素点与真实点的距离

y_i，要么为 y_i+1。因此，可能选择的两个点为 (x_{i+1}, y_i)、(x_{i+1}, y_i+1)。设这两个点与直线在 x_{i+1} 处的真实点的距离分别为 d_1、d_2，则有：

$$d_1 = y - y_i = kx_{i+1} + b - y_i = k(x_i + 1) + b - y_i \tag{6}$$

$$d_2 = y_i + 1 - y = y_i + 1 - (k(x_i + 1) + b) \tag{7}$$

$$d_1 - d_2 = 2(k(x_i + 1) + b) - 2y_i - 1 \tag{8}$$

$$= 2\frac{dy}{dx}(x_i + 1) - 2y_i + 2b - 1 \tag{9}$$

$$p_i = dx(d_1 - d_2) = 2dyx_i - 2dxy_i + c \tag{10}$$

其中，dx、dy 分别为起点到终点在 x 轴和 y 轴上的距离，$c = 2dy + dx(2b - 1)$ 为一个与位置无关的常量。由于斜率在 0 1 之间，所以 p_k 与 $d_1 - d_2$ 的符号相同。即当 $p_k > 0$ 时，$d_1 > d_2$，$y_i + 1$ 处的像素点更接近真实点，应选择 $y_i + 1$ 作为 x_{i+1} 处像素点的纵坐标；反之当 $p_k < 0$ 时，$d_1 < d_2$，y_i 处的像素点更接近真实点，应选择 y_i 作为 x_{i+1} 处像素点的纵坐标。对 p_i 求递推公式，有：

$$p_{i+1} - p_i = 2dy(x_{i+1} - x_i) - 2dx(y_{i+1} - y_i) \tag{11}$$

$$= 2dy - 2dx(y_{i+1} - y_i) \tag{12}$$

$$p_{i+1} = p_i + 2dy - 2dx(y_{i+1} - y_i) \tag{13}$$

而 $y_{i+1} - y_i$ 取决于 p_i 的符号，即：

$$y_{i+1} - y_i = \begin{cases} 1, p_i > 0 \\ 0, p_i < 0 \end{cases} \tag{14}$$

因此有

$$p_{i+1} = \begin{cases} p_i + 2dy - 2dx, & p_i > 0 \\ p_i + 2dy, & p_i < 0 \end{cases} \tag{15}$$

通过上式对 p 不断进行递推，确定每个 x 对应的 y 值即可。

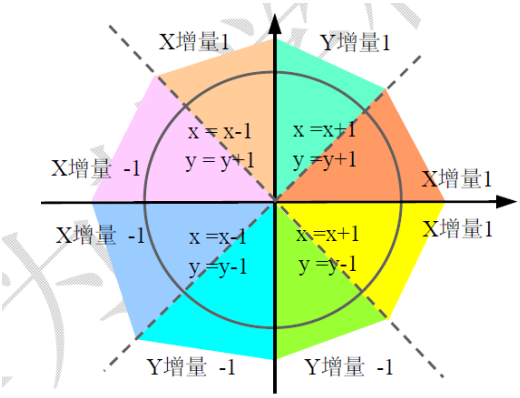


图 2: Bresenham 算法的通用性处理

要实现任意方向的 Bresenham 算法也很容易，在编程时，通过条件判断确保两个点中左边的点（即横坐标较小的点）为起始点，则图中 3、4、5、6 象限的情况被自动转化为其对角象限的情况。对于第二象限，则将上述推导中 x、y 互换，使 y 增量一直为 1，x 的增量根据 p 进行判断即可。再通过对两个点纵坐标的大小关系进行判断，使得当 dy 小于 0 时 y 的增量为-1，即可用处理 1、2 象限的代码对 7、8 象限的情况进行处理。

2.2 绘制多边形

绘制多边形，只需要将多边形每两个相邻的点作为线段的起始点和终点，调用上述实现的相应绘制线段算法进行绘制即可。

2.3 绘制椭圆

先考虑椭圆中心在原点的情况。在绘制椭圆时，利用了椭圆的对称性，只需绘制第一象限上的点，再通过将这些点分别对称到第二、第三和第四象限，即可得到完整的椭圆。绘制椭圆时，用到了利用 Brezenhem 算法绘制线段时的思路，即借助一个误差量的符号确定下一个像素点的位置。

已知椭圆函数可以表示为：

$$f_{ellipse}(x,y) = r_y^2x^2 + r_x^2y^2 - r_x^2r_y^2 \tag{16}$$

当 $f_{ellipse}(x,y) < 0$ 时，点 (x,y) 位于椭圆内部，当 $f_{ellipse}(x,y) > 0$ 时，点 (x,y) 位于椭圆外部，当 $f_{ellipse}(x,y) = 0$ 时，点 (x,y) 位于椭圆边界上。根据椭圆的公式，可以得到椭圆上点 (x,y) 处的切线斜率为

$$\frac{dy}{dx} = -2\frac{r_y^2x}{r_x^2y} \tag{17}$$

根据切线斜率与 1 的关系，可以将椭圆分为以上两个区域，区域 1 中椭圆切线斜率绝对值小于 1，在 x 方向上取步长；区域 2 中椭圆切线斜率绝对值大于 1，在 y 方向上取步长。从区域 1 移入区域 2 的条件是 $2r_y^2x$ $2r_x^2y$。

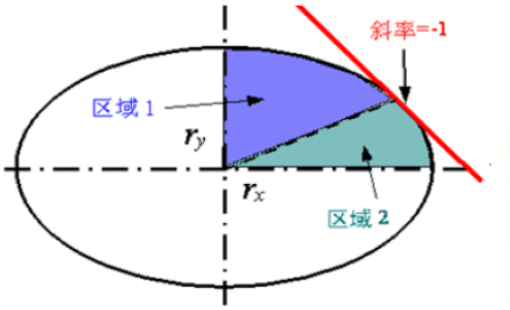


图 3: 第一象限椭圆区域划分

区域 1 中，切线斜率的绝对值小于 1，决策变量为 x。假设上一步中选择的像素点位置为 (x_k, y_k)，则在下一步中，可能选择的点为 (x_{k+1}, y_k) 和 $(x_{k+1}, y_k - 1)$。

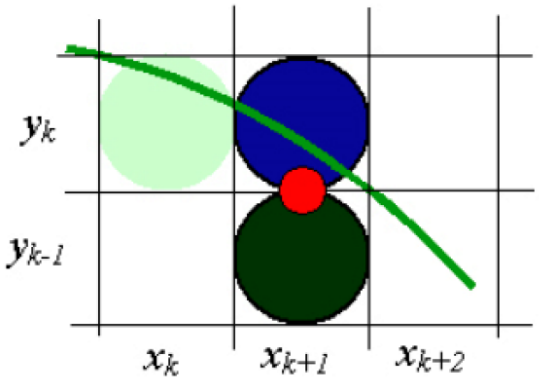


图 4: 中点圆算法第 k+1 步侯选点

将两个候选点的中点 $(x_{k+1}, y_k + \frac{1}{2})$ 作为决策参数，求值

$$p1_k = f_{ellipse}(x_{k+1}, y_k - \frac{1}{2}) = r_y^2(x_k + 1)^2 + r_x^2(y_k - \frac{1}{2})^2 - r_x^2 r_y^2 \tag{18}$$

则有：

- 当 $p1_k < 0$ 时，中点在椭圆的内部，选择点 (x_{k+1}, y_k)
- 当 $p1_k \geq 0$ 时，中点在椭圆的外部或边界上，选择点 $(x_{k+1}, y_k - 1)$

再寻找决策参数 $p1_k$ 的递推函数：

$$p1_{k+1} = \begin{cases} p1_k + 2r_y^2 x_k + 3r_y^2, & p_k < 0 \\ p1_k + 2r_y^2 x_k + 3r_y^2 - 2r_x^2 y_k + 2r_x^2, & p_k \geq 0 \end{cases} \tag{19}$$

在这个阶段，$p1_0$ 由式 $p1_0 = r_y^2 - r_x^2 r_y + \frac{r_x^2}{4}$ 得到

区域 2 中，切线斜率的绝对值大于 1，在 y 上取步长，其余思路与区域 1 中类似，在

此给出推导：

$$y_{k+1} = y_k + 1 - 1 \tag{20}$$

$$p2_k = f_{ellipse}(x_k + \frac{1}{2}, y_{k+1}) = r_y^2(x_k + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \tag{21}$$

则有：

- 当 $p2_k < 0$ 时，中点在椭圆的内部，选择点 $(x_k + 1, y_k - 1)$
- 当 $p2_k \geq 0$ 时，中点在椭圆的外部或边界上，选择点 $(x_k, y_k - 1)$

再寻找决策参数 $p1_k$ 的递推函数：

$$p2_{k+1} = \begin{cases} p2_k - 2r_x^2 y_k + 3r_x^2, & p_k < 0 \\ p2_k + 2r_x^2 y_k + 3r_x^2 + 2r_y^2 x_k + 2r_y^2, & p_k \geq 0 \end{cases} \tag{22}$$

在这个阶段使用区域 1 中最后的点作为区域 2 的起点 (x_0, y_0)，$p2_0$ 由式 $p2_0 = r_y^2(x_0 + \frac{1}{2}) + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$ 得到。

完成上述两步之后，就可以完成第一象限内的椭圆绘制了，再将其对称到其他三个象限，再将每个点平移，即可实现以任一点为椭圆中心的椭圆绘制了。

2.4 绘制曲线

2.4.1 Bezier 曲线

在学习了老师的 ppt 及中国大学 mooc 上的慕课之后，我使用 De Cadteljau's Algorithm 来绘制 Bezier 曲线。通过该方法来实现绘制 Bezier 曲线只需要在控制点连接得到的线段上按比例 u 进行划分，取到分割点，连接相邻分割点后再按该比例 u 取分割点，递归直到取到的点只有 1 个，就得到了参数 u 对应的曲线上的点。将参数 u 从 0 到 1 按一定步长遍历，即可得到所需的曲线。

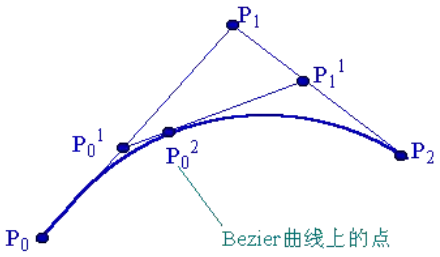


图 5: 二阶 Bezier 曲线

以图5为例，图中 P_0、P_1、P_2 即为三个控制点，第一次分割得到分割点 P_0^1、P_1^1，满足：

$$\frac{P_0P_0^1}{P_0^1P_1} = \frac{P_1P_1^1}{P_1^1P_2} = u \tag{23}$$

将这两个分割点相连，并在得到的线段上再次进行分割得到 P_0^2，满足：

$$\frac{P_0^1 P_0^2}{P_0^2 P_1^1} = u \tag{24}$$

由于此次分割只有该分割点，因此该分割点即为 Bezier 曲线上的点。该过程可以直观地表示成图6：通过以上的思路进行迭代，即可画出给定控制点对应的 Bezier 曲线。

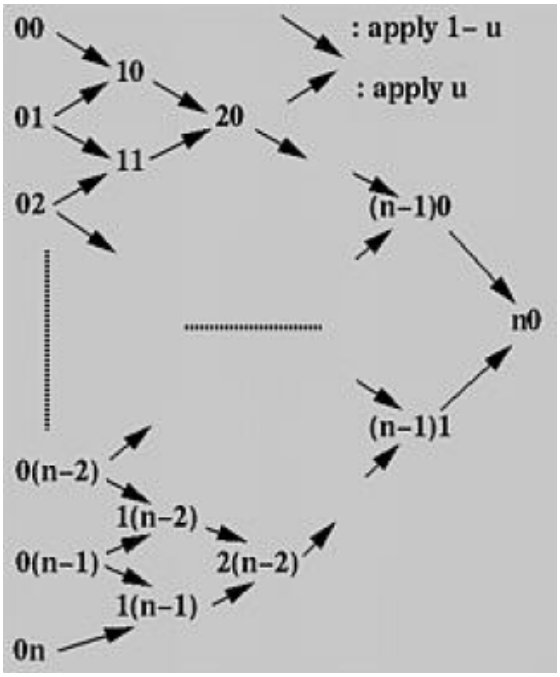


图 6: Bezier 曲线递归计算示例

2.4.2 B 样条曲线

Bezier 曲线的一个特点是主要改动一个控制点，整条曲线就会被改变，而 B 样条曲线则具有很好的局部性。同样，在学习了老师的 ppt 及中国大学 mooc 上的慕课之后，我使用 de Boor-Cox 算法来完成对于三次均匀 B 样条曲线的绘制。

B 样条曲线的定义是：给定 n+1 个控制点$P_0,\ldots,P_n$ 和 m+1 个节点$u_0,\ldots,u_m$，k 阶 B 样条曲线

$$C(u) = \sum_{i=0}^{n} B_{i,k}(u) P_i, u \in [u_{k-1}, u_{n+1}] \tag{25}$$

de Boor-Cox 算法中，对于 B 的递推定义如下：当 k=1 时，

$$B_{i,1(u)} = \begin{cases} 1, & u_i < u < u_{i+1} \\ 0, & Otherwise \end{cases} \tag{26}$$

k>1 时，

$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u) \tag{27}$$

通过上述递推公式，结合 B 样条曲线定义的公式，可以得到 B 样条离散点的递推计算公式如下：

$$P_i^r(u) = \begin{cases} P_i, & r = 0 \\ \lambda_i^r(u)P_i^{r-1}(u) + (1 - \lambda_i^r(u))P_{i-1}^{r-1}(u), & r > 0 \end{cases} \tag{28}$$

其中，$u \in [u_i, u_{i+1})$，

$$\lambda_i^r(u) = \frac{u - u_i}{u_{i+k-r} - u_i}(r = 0, 1, 2, \ldots, k-1) \tag{29}$$

观察上式不难发现，B 样条曲线和 Bezier 曲线是非常相似的, 主要的变化是 u 变成了 $\lambda_i^r(u)$; 观察上面的式子同样可以发现, 对于某一个三次 B 样条曲线的点 P_j^3, 与它关联的控制点实际上只有 P_j^0、P_{j-1}^0、P_{j-2}^0、P_{j-3}^0, 这也是 B 样条曲线局部控制能力的原因。通过以上的思路进行迭代，即可画出给定控制点对应的 B 样条曲线。

2.5 图元平移

图元的平移可以通过两种方法实现：

- 根据需要平移的图元 id 找到对应的图元，将已经画出的所有点逐个进行平移
- 在画图之前将对应图元的关键点进行平移，之后再进行画图操作

显然，第一种方法需要成倍的计算资源及处理时间，而第二种方法只需对图元的关键点进行处理，只需要常数时间，因此我才用第二种方法实现了对图元的平移。

2.6 图元旋转

先考虑以原点为基准点时的情况。当以原点为基准点，进行逆时针旋转时，二维旋转变换方程为：

$$\begin{cases} x_2 &= x_1 cos\theta - y_1 sin\theta \\ y_2 &= x_1 sin\theta + y_1 cos\theta \end{cases} \tag{30}$$

为了实现以任意点 (x_0, y_0) 为基准点的旋转，可先将选定为基准点的点作为新坐标系的原点，利用上式完成旋转后再将坐标变换回原来的坐标系中。与图元平移中相同，在实现图元旋转时，我通过对图元的关键点进行旋转来实现。

$$\begin{cases} x_2 &= x_0 + (x_1 - x_0)cos\theta - (y_1 - y_0)sin\theta \\ y_2 &= y_0 + (x_1 - x_0)sin\theta + (y_1 - y_0)cos\theta \end{cases} \tag{31}$$

2.7 图元缩放

依然先考虑以原点为缩放中心时的情况。以原点为缩放中心时，缩放倍数为 s 时的二维缩放变换方程为：

$$\begin{cases} x_2 &= sx_1 \\ y_2 &= sy_1 \end{cases} \tag{32}$$

为了实现以任意点 (x_0, y_0) 为缩放中心的缩放，可先将选定为缩放中心的点作为新坐标系的原点，利用上式完成缩放后再将坐标变换回原来的坐标系中。与图元平移中相同，在实现图元缩放时，我通过对图元的关键点进行缩放来实现。

$$\begin{cases} x_2 &= x_0 + s(x_1 - x_0) \\ y_2 &= y_1 + s(y_1 - y_0) \end{cases} \tag{33}$$

2.8 裁剪线段

线段裁剪要求根据给定的裁剪窗口左下点 (x_{min}, y_{ming}) 和右上点 (x_{max}, y_{max}) 来对指定线段进行裁剪。此过程的一般思路为求线段与裁剪窗口的交，并更新线段的两端点 (x_0, y_0)，(x_1, y_1) 进行修正。

2.8.1 Cohen-Sutherland 裁剪算法

Cohen-Sutherland 算法采用区域检查的方法，能够快速有效地判断一条线段与裁剪窗口地位置关系，如果线段在矩形外，则直接丢弃；若线段在矩形内，则直接保留；若线段一部分在矩形外，一部分在矩形内，则需要将该线段与矩形求交，并更新该线段的端点信息。由于完全接受或完全舍弃的线段无需求交，可以直接识别，大大减少求交的计算从而提高裁剪算法的效率。

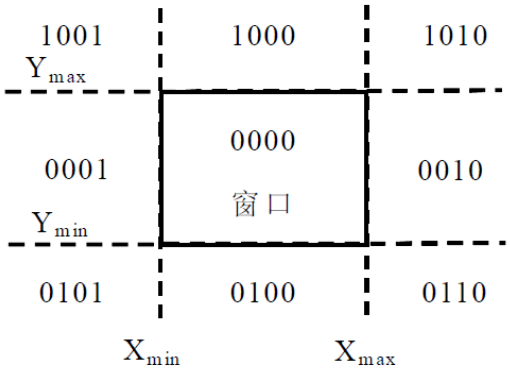


图 7: 区域编码

具体而言，Cohen-Sutherland 算法用位运算来确定线段端点如上图所示的 9 个状态，规则如下：

- 如果端点在裁剪窗口上方，即 $y > y_{max}$，则第 1 位置为 1，否则为 0
- 如果端点在裁剪窗口下方，即 $y < y_{min}$，则第 2 位置为 1，否则为 0
- 如果端点在裁剪窗口右边，即 $x > x_{max}$，则第 3 位置为 1，否则为 0
- 如果端点在裁剪窗口左边，即 $x < x_{min}$，则第 4 位置为 1，否则为 0

上述编码规则可由以下 Python 代码实现：

```
def ComputeOutCode(x_min, y_min, x_max, y_max, x, y):
        code = 0
        if x < x_min:
                code |= 1
        elif x > x_max:
                code |= 2
        if y < y_min:
                code |= 4
        elif y > y_max:
                code |= 8
        return code
```

根据上面的算法计算得到两个点的位置信息 (outcode1,outcode2)，可以分为以下三种情况：

- outcode1=outcode2=0，这种情况表示两个点都在裁剪窗口里面，无需裁剪；
- (outcode1&outcode2)!=0，这种情况表示两个点在裁剪窗口外的同一边，此时线段与不可能有在裁剪窗口内的部分，因此这种情况可以直接舍弃；
- 当不是上述两种情况的时候，线段一定会和裁剪窗口有一个或两个交点，则可以通过一个 while 循环每次取一个在窗口外的点，根据其编码显示的该点与裁剪窗口的位置信息，取直线与相应边界的交点作为该点的新的取值，知道两个点都到达裁剪窗口边界或在裁剪窗口内（编码为 0000）时，即可得到所需的裁剪结果。

2.8.2 Liang-Barsky 裁剪算法

Liang-Barsky 算法和 Cohen-Sutherland 算法相比，需要进行的状态判断更少，使算法效率得到了进一步的提升。该算法的基本思路是把待裁剪的线段和裁剪窗口都看成是一维的点集，那么裁剪结果即为两点击的交集。

我们知道了，一条两端点为 $P_1(x_1, y_1)$、$P_2(x_2, y_2)$ 的线段可以用参数方程的形式表示为

$$\begin{cases} x = x_1 + u(x_2 - x_1) \\ y = y_1 + u(y_2 - y_1) \end{cases} \tag{34}$$

式中参数 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 $P_1(x_1, y_1)$，当 u=1 时，该点为 $(P_2(x_2, y_2))$。如果点 $P(x, y)$ 位于由坐标 (x_{min}, y_{min}) 和 (x_{max}, y_{max}) 确定的窗口内，那么有下式成立：

$$\begin{cases} x_{min} \le x_1 + u(x_2 - x_1) \le x_{max} \\ y_{min} \le y_1 + u(y_2 - y_1) \le y_{max} \end{cases} \tag{35}$$

这四个不等式可以表示为

$$up_k \le q_k, k = 1, 2, 3, 4 \tag{36}$$

其中，p、q 定义为

$$\begin{cases} p_1 = -(x_1 - x_2) \\ p_2 = x_1 - x_2 \\ p_3 = -(y_1 - y_2) \\ p_3 = y_1 - y_2 \end{cases} \quad \begin{cases} q_1 = x_1 - x_{min} \\ q_2 = x_{max} - x_1 \\ q_3 = y_1 - y_{min} \\ q_4 = y_{max} - y_1 \end{cases} \tag{37}$$

从上式可知：

- 任何平行于窗口某边界的直线，其 $p_k = 0$，k 值对应于相应的边界（k=1，2，3，4 对应于左、下、右、上边界），如果此时还满足 $q_k < 0$，则线段完全在边界外，应舍弃该线段。如果 $p_k = 0$ 且 $q_k \geq 0$，则线段平行于窗口某边界并且在窗口内。
- 当 $p_k < 0$ 时，线段从裁剪边界延长线的外部延申到内部
- 当 $p_k > 0$ 时，线段从裁剪边界延长线的内部延申到外部

因此，当 $p_k \neq 0$ 时，可计算出线段与边界 k 或其延长线的交点的 u 值 $u = \frac{q_k}{p_k}$；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：

- u1 的值由线段从外到内遇到的矩形边界所决定（$p_k < 0$），对这些边界计算 $r_k = q_k/p_k$，u1 取 0 和各个 r 值之中的最大值
- u2 的值由线段从内到外遇到的矩形边界所决定（$p_k > 0$），对这些边界计算 $r_k = q_k/p_k$，u2 取 1 和各个 r 值之中的最小值
- 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

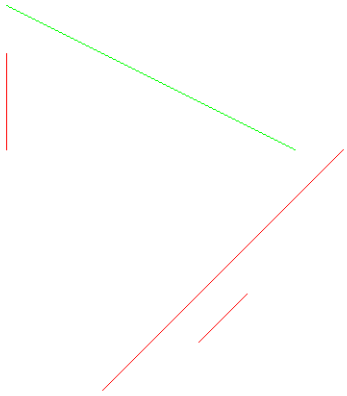
3 系统介绍

3.1 命令行界面 CLI

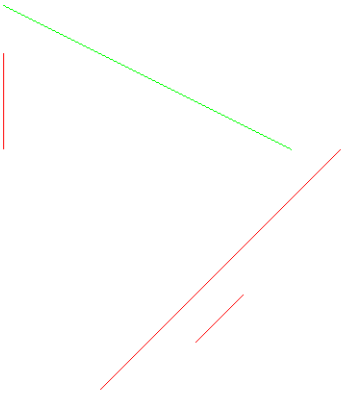
该部分实现于 cg_cli.py 中，功能为读取包含了图元绘制指令序列的文本文件，依据指令调用核心算法模块中的算法绘制图形以及保存图像。对于输入的文本，程序逐句进行语义分析，并通过以下操作实现图形的绘制及画布的导出：

- 对于图元绘制的指令，程序只保存绘制图元的类型、编号、关键点及所采用的算法；
- 对于图形变换的指令，程序调用核心算法模块中相应的函数对被选中图元的关键点进行变换；
- 当遇到保存画布的指令时，程序遍历已保存的图元库，对图元逐一调用核心算法模块中相应的绘制函数进行绘制，并将得到的像素点在画布上一一绘制出。

使用项目中给定的测试样例进行测试，并与标准结果进行对比，结果见图 8 ∼ 12，测试样例与标准结果肉眼上一致。

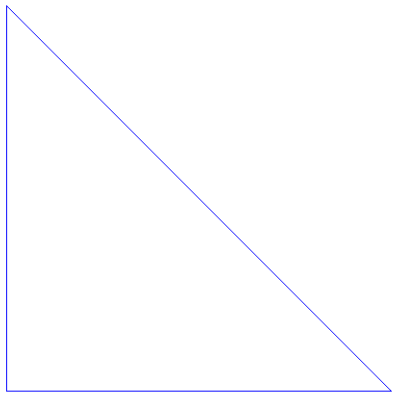


(a) 1-测试

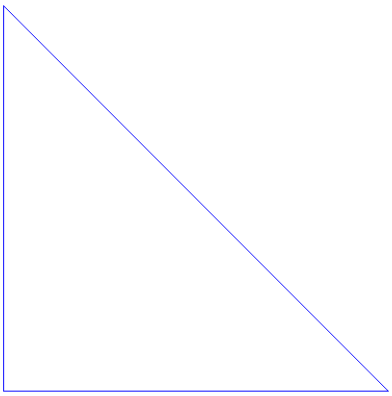


(b) 1-参考

图 8: 测试样例一

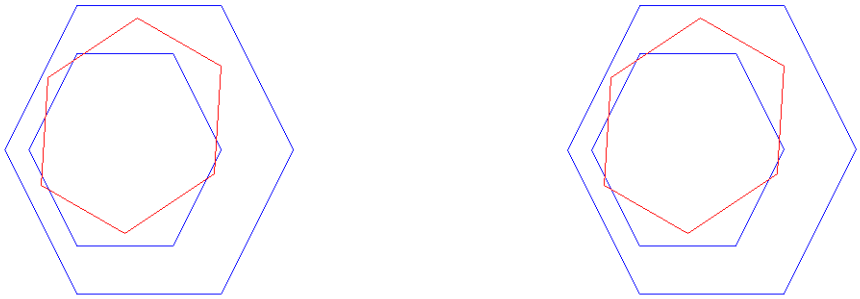


(a) 2-测试



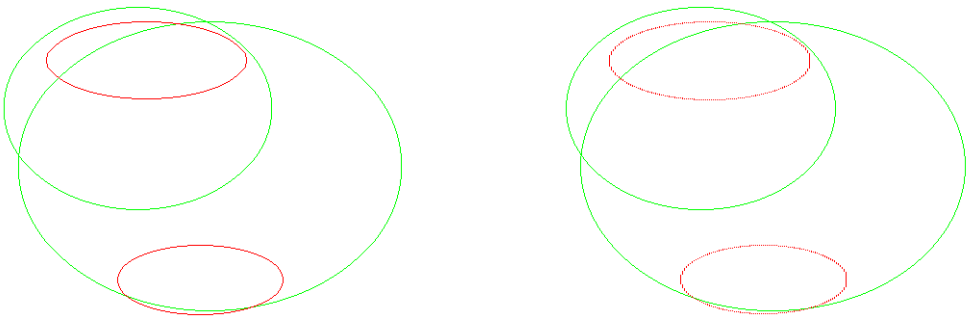
(b) 2-参考

图 9: 测试样例二



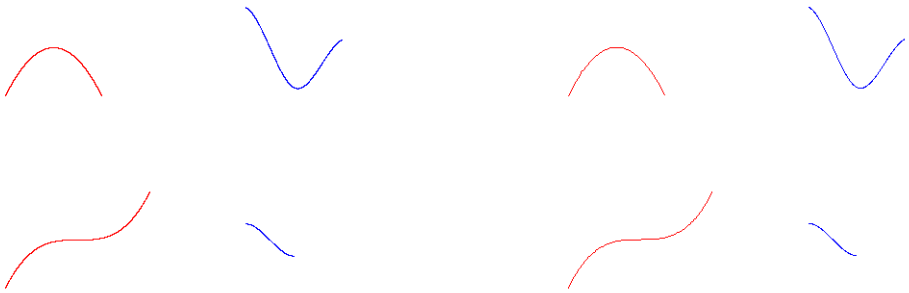
(a) 3-测试　　(b) 3-参考

图 10:　测试样例三



(a) 4-测试　　(b) 4-参考

图 11:　测试样例四



(a) 5-测试　　(b) 5-参考

图 12: 测试样例五

3.2 用户交互界面 GUI

3.2.1 实验环境

- 系统环境：

  Windows 10 专业版

  Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00GHz

  16.0 GB RAM

- 开发环境：

  Python 3.7.6

  numpy==1.18.1

  Pillow==8.0.1

  PyQt5==5.15.2

3.2.2 代码结构

- MainWindow.py 中实现与主窗口相关的各种工作，包括设置主窗口的布局、连接信号和槽函数、设置菜单栏、槽函数的实现等
- MyCanvas.py 中实现画布的各种核心功能，包括提供供槽函数调用的各种接口、对鼠标事件的处理等
- MyItem.py 中实现自定义的图元类，在该类中记录图元的各种信息，并通过对核心算法模块的调用实现图元的绘制、变换等

- cg_gui.py 为 GUI 程序的入口

3.2.3 实现思路

用户完成一个绘图或变换操作的过程中，程序需进行的处理如下：

- 用户在菜单栏选取一种操作，与该操作相关的按键会向程序发送出一个信号，由于该信号在 MainWindow 模块中被与一个槽函数绑定，因此当系统接收到该信号时，会调用相应的槽函数，由其对该点击事件进行处理
- 当某个槽函数被调用时，意味着用户接下来会进行相应的操作，因此在槽函数中，我们需要告知画布做相应的准备，即调用 MyCanvas 模块给出的各个相应的接口，并传入必须的参数（如绘制直线、多边形、曲线时需要传入所选择的算法）；此外，槽函数中还需要实现对用户进行相应的操作提示，以方便用户的下一步操作；
- 在 MyCanvas 模块的被调用的准备函数中，程序会为下一个操作进行准备，如结束上一次操作，并将模块中用以标识当前操作的变量置为相关的值等；完成这些操作之后，对收到的信号的处理就已经完成了。
- 之后，用户会在操作提示下，利用鼠标在画布上进行相关的操作。鼠标的点击、移动结合可以实现多种操作，因此我们在程序中必须处理鼠标的点击、拖动、松开时间，结合合适的操作逻辑，实现图元的绘制、变换等；
- 在这个部分，程序根据鼠标点击时记录下的坐标点及鼠标拖动时不断变化的坐标点，结合不同操作的不同实现，更新当前正在操作的图元的相关信息，并对画布上所有的图元进行反复的刷新重绘，以达到用户眼中的实时操作。

根据以上思路，对给定的框架代码进行补充，即可实现 GUI 程序的基本功能。基本功能实现如图 13。

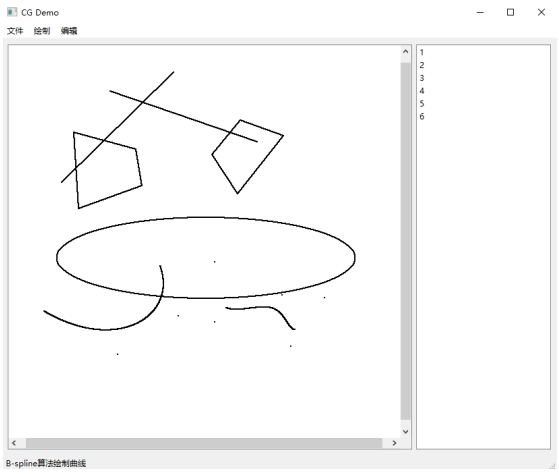


图 13: GUI 界面

而设置画笔、重置画布则需要进行单独的处理，如下：

设置画笔分为两个功能：设置画笔颜色和设置画笔粗细。设置画笔颜色功能可通过在对应槽函数中调用 QColorDialog 模块中的 getColor 函数，此时用户可通过 select color 对话框选取想要的画笔颜色，而后程序会得到一个 QColor 变量；而设置画笔粗细则是通过调用 QInputDialog 模块中的 getInt 函数，弹出一个对话框并得到用户输入的画笔粗细值。以上两个值会被槽函数先通过接口传入画布，画布在生成新图元时传入此两个参数，生成对应的 QPen，并通过对绘制函数中 painter 参数的 pen 进行替换实现设置画笔的功能

重置画布则是通过调用自己在 MyCanvas 中实现的接口，将存储的图元清空，并调用 self.list_widget.clear() 函数和 self.scene.clear() 函数清空列表和画布上的图元来实现。

此外，我重写了 get_id 函数，将其置于 MyCanvas 中，并根据存储的图元数量来返回新的图元应分配的编号，以解除原有的编号不准确的 bug。

GUI 的最终实现见图 13 ～ 15，操作逻辑见说明书及演示视频。

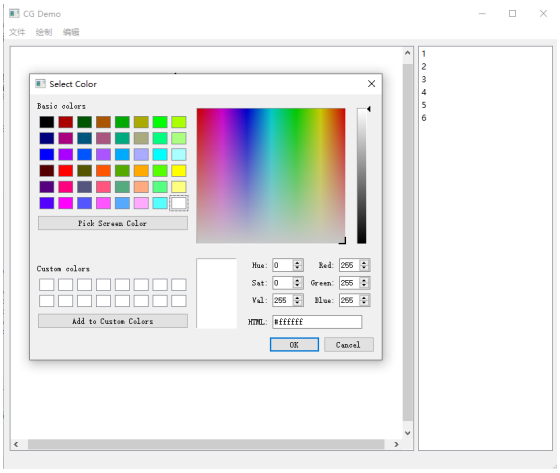


图 14: 设置画笔颜色

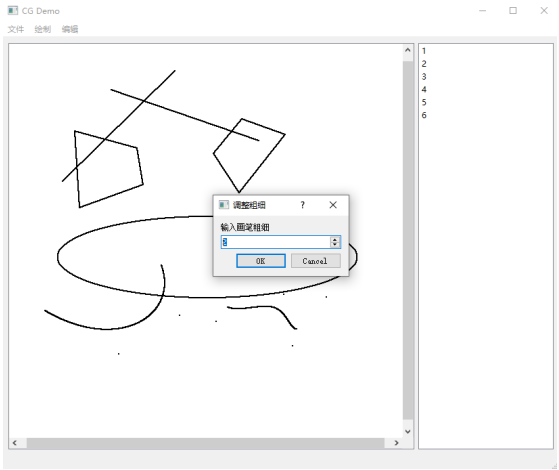


图 15: 设置画笔粗细

4 总结

在 12 月，我完成了图形学大作业的所有内容，实现了核心算法模块中的直线绘制、多边形绘制、椭圆绘制、曲线绘制、图元平移、图元旋转、图元缩放、线段裁剪算法，实现了命令行界面及用户交互界面。完成该大作业之后，我对计算机图形学的算法有了更深刻的理解和掌握。

参考文献

[1]《计算机图形学教程》孙正兴 2006

[2] 图形学入门 (1)——直线生成算法（DDA 和 Bresenham）

[3] 梁友栋-Barsky 裁剪算法

[4] B 样条曲线

[5] 贝塞尔 (Bézier) 曲线-之一

[6] 贝塞尔曲线原理 (实现图真漂亮)

[7] B 样条曲线（B-spline Curves）

[8] PyQt5 中文手册

[9] pyqt5 对话框——QInputDialog、QColorDialog、QFontDialog、QMessageBox、QFileDialog

[10] 对 python 中 math 模块下 atan 和 atan2 的区别详解

[11] C++ 根据三个点坐标计算夹角