计算机图形学大作业报告

作者：唐业

邮箱：1574588673@qq.com

1. 综述

本实验使用Python3语言编程，实现一个绘图系统。

根据讲义中系统实现要求中要实现的内容，完成了以下模块的内容：

- 核心算法模块（各种图元的生成、编辑算法）：cg_algorithms.py（**完成**）
- 命令行界面（CLI）程序：cg_cli.py（**完成**）
- 用户交互界面（GUI）程序：cg_gui.py（**完成**）

2. 算法介绍

2.1. 绘制线段

2.1.1. DDA算法

在已知线段的两个端点的情况下，即能得到$y = kx + b$，绘制线段的最简单方法是从直线起点开始令x或y每次增加1直到终点，每次根据直线方程计算对应的函数值之后四舍五入，即对应一个像素点，但这种方法显然需要每一步进行浮点数乘法运算，效率很低。

DDA算法是利用计算两个坐标方向的差分来确定现段显示的屏幕像素位置的线段扫描转换算法，它的**基本思想**是高数里的微分算法，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值。

$$
\begin{aligned}
y_i &= kx_i + b \\
y_{i+1} &= kx_{i+1} + b = kx_i + b + k \cdot dx = y_i + k \cdot dx
\end{aligned}
$$

当$\triangle x = 1$时，x每递增1，y就递增k，反之当$\triangle y = 1$时，y每递增1，x就递增$1/k$，所以只需要对x和y不断递增就可以得到下一点的函数值，这样避免了对每一个像素都使用直线方程来计算，消除了浮点数乘法运算。

DDA算法的实现一个迭代过程，设迭代过程中前一个点为(x_i, y_i)，后一个点为(x_{i+1}, y_{i+1})，则这两个点有以下的关系式：

$$
\begin{aligned}
x_{i+1} &= x_i + dx, \\
y_{i+1} &= y_i + dy,
\end{aligned}
$$

而dx, dy是通过选择$abs(\triangle x), abs(\triangle y)$中的较大值作为一个光栅单位，即$max(abs(\triangle x), abs(\triangle y))$来确定$dx$和$dy$中哪一个是1：

- 如果$\triangle x > \triangle y$，则说明$x$轴方向上的两个点的距离更远，此时设置$dx = 1, dy = k$；
- 如果$\triangle y > \triangle x$，则说明$y$轴方向上的两个点的距离更远，此时设置$dx = \frac{1}{k}, dy = 1$；

部分代码如下：

```
elif algorithm == 'DDA':
        x_, y_ = x1 - x0, y1 - y0
        length = max(abs(x_), abs(y_))
        if length == 0:
            result.append((x0, y0))
```

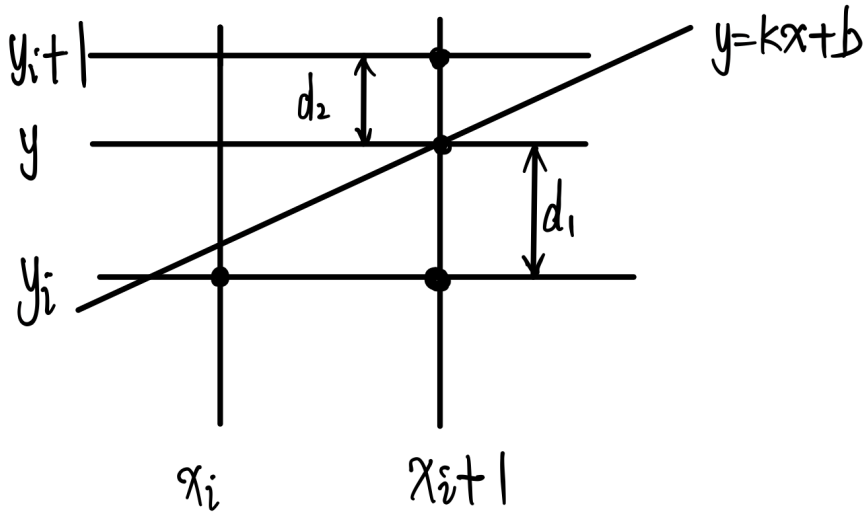
```
        else:
            dx, dy = x_ / length, y_ / length
            x, y = x0, y0
            i = 0
            while i <= length:
                result.append((int(x), int(y)))
                x += dx
                y += dy
                i += 1
```

2.1.2. Bresenham算法

DDA算法消除了浮点数乘法的弊端，但是仍然存在着浮点数加法和取整操作，Bresenham算法的好处在于只使用了整数增量计算，利用一个误差量的符号确定下一个像素点的位置。**推导**过程：需要决策的是找到更加接近于真实点的哪个最佳像素点。先考虑斜率在$(0,1)$的情况，从线段左端点开始处理，每当确定当前像素点(x_i,y_i)后，考虑下一个像素点x_{k+1}，固定$x_{i+1}=x_i+1$，那么下一个像素点在y_i和y_i+1。

对于直线$y=kx+b$，已知下一个点$x_{i+1}=x_i+1$，那么真实的$y=k(x_i+1)+b$，考虑真实的y与

- $d1=y-y_i=k(x_i+1)+b-y_i$
- $d2=y_i+1-y=y_i+1-k(x_i+1)-b$



定义一个决策变量p_i:

$$p_i=2dyx_i-2dxy_i+(2dy+2bdx-dx)=2dyx_i-2dxy_i+c$$

其中c是一个常数，因为dx大于0，所以p_i和$(d1-d2)$是同号的，可以得到如下的结论:

- 如果$p_i>0$，那么$d1-d2>0$，y_i+1距离真实的y更接近，所以选择(x_{i+1},y_i+1);
- 如果$p_i<0$，那么$d1-d2<0$，y_i距离真实的y更接近，所以选择(x_{i+1},y_i);

关于p_i的计算可以变成一个迭代过程:

假设斜率是$(0,1)$，即$x_{i+1}=x_i+1$，$y_{i+1}=y_i\ \ or\ \ y_i+1$: i

- 如果$p_i>0$，$y_{i+1}=y_i+1$，$p_{i+1}=p_i+2dy-2dx$
- 如果$p_i<0$，$y_{i+1}=y_i$，$p_{i+1}=p_i+2dy$

p_0的推导，$p_0=2dyx_0-2dxy_0+2dy+2dx(y0-\frac{dy}{dx}x0)-dx=2dy-dx$

综上，当直线斜率为0-1的时候$(x_{i+1}=x_i+1)$，Bresenham的算法流程处理如下:

1. 选择起始点(x_0, y_0),终止点(x_1, y_1)，保证$x1 > x0$，否则交换;计算
   $dx = x_1 - x_0, dy = y_1 - y_0, p_0 = 2dy - dx$
2. 从$i = 0$，已知$(x_i, y_i), p_i$，计算下一个像素点(x_{i+1}, y_{ii+1})以及决策变量p_{i+1}，其中
   $x_{k+1} = x_k + 1$;
   - 如果$p_i > 0, y_{i+1} = y_i + 1. p_{i+1} = p_i + 2dy - 2dx$i
   - 如果$p_i \leq 0, y_{i+1} = y_i, p_{i+1} = p_i + 2dy$
3. 循环直到$x_i = x_1$;

当斜率为其他的时候，其实改变的主要是决策变量的更新，以及附属的变量的更新，整理出来大概是一些正负号的问题，比如斜率在0-1之间意味着直线位于第一象限，如果要绘制第二象限的直线，只需要利用这两个象限关于直线$x = y$对称的性质，特别需要注意的是，当斜率为正无穷和0的时候需要像处理DDA算法一样进行特判。

2.2. 绘制多边形

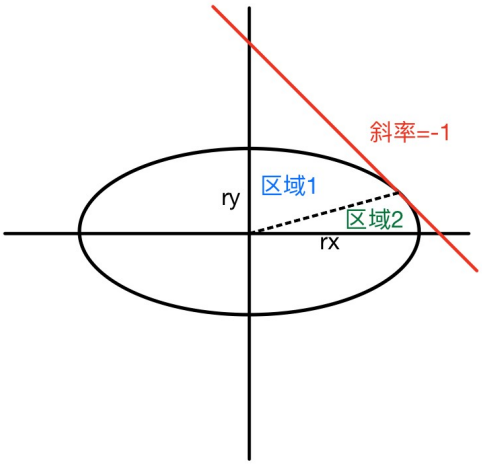
多边形只需要通过给定的一系列点，多次调用直线生成算法就能实现。

2.3. 绘制椭圆

2.3.1. 中点圆生成算法

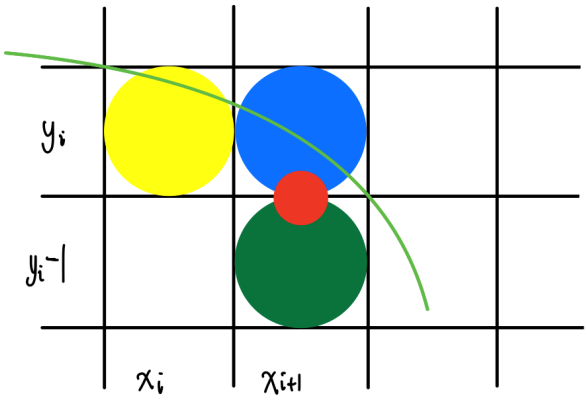
推导过程：首先考虑椭圆中心在原点时候的情况，计算出第一象限的像素点，然后通过平移和对称操作来得到整个椭圆的像素点。

已知椭圆函数为：$f(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$，得到椭圆的斜率为$\frac{dy}{dx} = -r_y^2 x / r_x^2 y$，根据斜率的绝对值是否小于1和大于1将椭圆分成两个部分，则两个区域交替条件为$r_y^2 x \geq r_x^2 y$



以$(0, ry)$为起点在整个第一象限中按顺时针方向生成椭圆，先生成区域1的点，在生成区域2的点；

- 区域1中，因为斜率的绝对值小于1，因此决策变量为x，设已经确定好(x_i, y_i)，则下一个取样点$x_{i+1} = x_i + 1$，$y_{i+1} = y_i \ or \ y_i - 1$，有两个选择。



  对于y_{i+1}的选择同样引入一个决策参数$p1_i$，这里取样位于两个候选像素点中间的点$(x_i + 1, y_i - \frac{1}{2})$，判断这个点与椭圆的位置，所以有
  $p1_i = f(x_{i+1}, y_i - \frac{1}{2}) = r_y^2(x_i + 1)^2 + r_x^2(y_i - \frac{1}{2})^2 - r_x^2 r_y^2$

  同样，$p1$也可以通过迭代过程来计算得到：

  - 若$p1_i < 0$，中点在椭圆内部，所以选择$(x_i + 1, y_i)$，$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$
  - 若$p1_i \geq 0$，中点在椭圆内部，所以选择$(x_i + 1, y_i - 1)$，
    $p1_{i+1} = p1_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_y^2$

  且$p1_0 = r_y^2 - r_x^2 r_y + r_x^2/4$，循环直到$2r_y^2 x \geq 2r_x^2 y$时进入区域2;

- 区域2中，切线的斜率绝对值小于1，因此决策变量是y，从(x_i, y_i)到下一个点(x_{i+1}, y_{i+1})的推导如下:

  $y_{i+1} = y_i - 1, \ p2_i = f(x_i + \frac{1}{2}, y_i - 1) = r_y^2(x_i + \frac{1}{2})^2 + r_x^2(y_i - 1)^2 - r_x^2 r_y^2$

  - 若$p2_i < 0$，中点在椭圆的内部，选择$(x_i + 1, y_i - 1)$，$p2_{i+1} = p2_i + 2r_y^2 x_{i+1} + r_x^2$
  - 若$p2_i \geq 0$，中点在椭圆的外部，选择$(x_i, y_i - 1)$，$p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$

这里$p2_0 = r_y^2(x_1 + \frac{1}{2}) + r_x^2(y_1 - 1)^2 - r_x^2 r_y^2$，循环到$(r_x, 0)$完成了第一象限椭圆的绘制了，最后通过对称、平移操作就可以得到整个椭圆。

在具体实现的过程中遇到的问题：我一开始使用了`round`函数进行浮点数的计算取整，但发现画出来的椭圆有断断续续的效果，最后改成直接用`int`进行类型转换解决了这个问题。

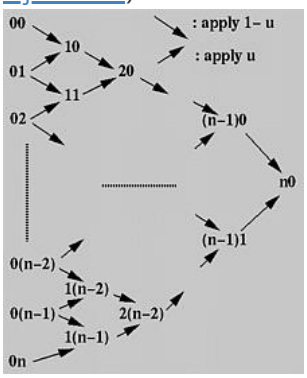
2.4. 绘制曲线

2.4.1. Bezier算法

Bezier曲线是在控制点连接的线段上，按照比例$t \in [0, 1]$，取到一组分割点，然后将所有的分割点连接起来再按照这个比例分割，直到取的点只有一个，这个点也是在曲线上的点，最后把所有的点就形成了Bezier曲线。

使用 De Casteljau's Algorithm 迭代算法来求Bezier曲线，其思想是：为了计算n次贝塞尔曲线上的点$C(t), t \in [0, 1]$，首先将控制点连接形成一条折线$00 - 01 - 02$ .....$0(n-1) - 0n$。利用上述方法，计算出折线中每条线段$0j - 0(j+1)$上的一个点$1j$，使得点$1j$分该线段的比为$t : 1 - t$，然后在折线$10 - 11 -$.....$-1(n-1)$上递归调用该算法，最后求得的那个点$n0$。 De Casteljau证明点$n0$一定是曲线上的点。（图引用自http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-cast

eljau.html)



伪算法如下：

```
Input: P[0:n] 表示n+1个点，实数u表示分割的比例,t∈[0,1]
Output: 曲线上的点C[t]

Q[0:n]
for(int i=0;i<=n;i++)
{
    Q[i]=P[i]
}
for(int k=1;k<=n;k++)
{
    for(int i=0;i<=n-k;i++)
    {
        Q[i]=(1-t)*Q[i]+t*Q[i+1];
    }
}
return Q[0];
```

在具体实现的时候，由于只允许使用math库，所以无法使用numpy中的linspace函数，但是可以自己定义步长，然后对于每次新的t调用上面的伪算法。

2.4.2. B-spline算法

Beizer曲线之中,只要改动一个点就会牵扯到整个曲线,可谓牵一发而动全身,B样条曲线则具有很好的局部修改特性;

使用 de Boor递推算法完成3次B样条曲线的绘制，下面阐述具体的算法;

- B样条曲线的定义是: $P(t)=\sum_{i=0}^{n}P_iN_{i,k}(t)$，其中，$P_i(i=0,1,\ldots,n)$是控制多边形的顶点，$N_{i,k}(t)(i=0,1,\ldots,n)$是$k$阶B样条基函数。
- 基函数的递推公式，也称为de Boor-Cox公式:

$$\begin{cases} N_{i,1}(t)=\begin{cases}1, t_i\le t\le t_{i+1}\\ 0, t<t_i \text{ or } t\ge t_{i+k}\end{cases}\\ N_{i,k}(t)=\frac{t-t_i}{t_{i+k-1}-t_i}N_{i,k-1}(t)+\frac{t_{i+k}-t}{t_{i+k-t_{i+1}}}N_{i+1,k-1}(t). \quad k\ge 2\end{cases}$$

- 由de Boor-Cox公式有，

$$
\begin{aligned}
P(t) &= \sum_{i=0}^{n} P_i N_{i,k}(t) = \sum_{i=j-k+1}^{j} P_i N_{i,k}(t) \\
&= \sum_{i=j-k+1}^{j} P_i \left[\left(\frac{t-t_i}{t_{i+k-1}-t_i} \right) N_{i,k-1}(t) + \left(\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} \right) N_{i+1,k-1}(t) \right] \\
&= \sum_{i=j-k+1}^{j} \left[\left(\frac{t-t_i}{t_{i+k-1}-t_i} \right) P_i + \left(\frac{t_{i+k-1}-t}{t_{i+k-1}-t_{i+1}} \right) P_{i-1} \right] N_{i,k-1}(t)
\end{aligned}
$$

令

$$
\mathbf{P}_i^{(r)}(t) = \begin{cases} \left(1-\tau_i^j\right) \mathbf{P}_{i-1}^{(r-1)}(t) + \tau_i^j \mathbf{P}_i^{(r-1)}(t) & \text{if } r = 1, 2, \ldots, k-1 \\ \mathbf{P}_i & \text{if } r = 0 \end{cases}
$$

其中，

$$
\tau_i^r = \frac{t-t_i}{t_{i+k-r}-t_i}
$$

最终有，

$$
P(t) = P_j^{[k-1]}(t)
$$

- 不难看出B样条曲线和Bezier曲线很相似，主要区别是是t变成了τ_i^r;观察上面的式子，对于某一个3次B样条曲线的点P_j^3，与它关联的控制点实际上只有$P_j^0, P_{j-1}^0, P_{j-2}^0, P_{j-3}^0$，所以B样条曲线有局部控制能力;

2.5. 图元平移

图元的平移的实现较为简单，直接修改图元的参数`p_list`中存放点的坐标即可，修改的公式如下：

$$
\begin{aligned}
x_{new} &= x_{old} + dx \\
y_{new} &= y_{old} + dy
\end{aligned}
$$

之后在绘制的时候会根据已经平移的参数进行绘制。

CLI中调用的方法如下：

```
item_id = line[1]
dx = int(line[2])
dy = int(line[3])
item_dict[item_id][1] = alg.translate(item_dict[item_id][1], dx, dy)
```

先取出参数，之后对应图元的`p_list`修改成平移之后的参数，下面的旋转和缩放实现也是一样的。

2.6. 图元旋转

推导过程：

先考虑以$(0,0)$为中心旋转，设原来的点A为(x_0, y_0)
此时半径为$r = \sqrt{x_0^2 + y_0^2}$，则A的极坐标为$(r\cos\theta, r\sin\theta)$，
其中$\cos\theta = \frac{x_0}{r}, \sin\theta = \frac{y_0}{r}$，
逆时针旋转α（转换为弧度制）后，得到
$x' = rcos(\theta+\alpha) = rcos\theta\cos\alpha - rsin\theta\sin\alpha = x_0\cos\alpha - y_0\sin\alpha$，
$y' = rsin(\theta+\alpha) = rsin\theta\cos\alpha + rcos\theta\sin\alpha = y_0\cos\alpha + x_0\sin\alpha$.

通过上述的式子不难得到以(x,y)为中心的旋转公式：
$x' = x + (x_0 - x)\cos\alpha - (y_0 - y)\sin\alpha$，
$y' = y + (y_0 - y)\cos\alpha + (x_0 - x)\sin\alpha$.

在实现过程中，需要注意两点：

1. 由于要求是顺时针旋转，所以在传入旋转角度r时，可以直接传入$-r$;
2. Python中计算弧度制以及三角函数时，可以直接调用math库中的`radians`，`cos`，`sin`等函数.

2.7. 图元缩放

推导过程：

先考虑以$(0,0)$为中心缩放，设原来的点A为(x_0,y_0)，缩放倍数为s时，有
$x'=x_0\cdot s,y'=y_0\cdot s$

通过上述的式子不难得到以(x,y)为中心的缩放公式：
$x'=x+x_0\cdot s,y'=y+y_0\cdot s$

2.8. 线段裁剪

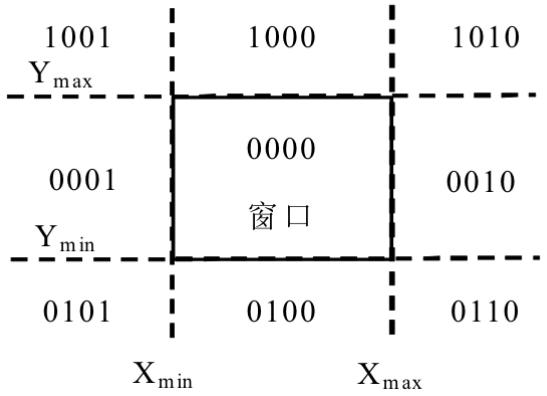
2.8.1. Cohen-Sutherland算法

线段的裁剪是根据裁剪窗口对线段的起点和终点进行修改，最简单的思路是：

- 如果直线完全在裁剪窗口内部，那么无需修改；
- 如果直线完全在裁剪窗口外部，那么直接舍弃；
- 如果直线有一部分在窗口内部，那么求得线段和窗口的交点作为新的起点和终点。

这里遇到的问题有两个：一是如何判断直线和窗口的位置关系；二是如何求线段和窗口的交点

Cohen-Sutherland算法采用区域检查的方法，通过位运算从而快速判断直线和窗口的位置关系，如下图（图引用自教材）所示，该算法以9个区域为基础，根据线段的端点所在的区域，每个端点均赋以四位二进制码。



区域编码的生成规则这里使用比较法：

- 如果 $x<x_{min}$，表示该点在裁剪窗口左边界的左边，则第 1 位置 1，否则置 0;
- 如果 $x>x_{max}$，表示该点在裁剪窗口右边界的右边，则第 2 位置 1，否则置 0;
- 如果 $y<y_{min}$，表示该点在裁剪窗口下边界的下边，则第 3 位置 1，否则置 0;
- 如果 $y>y_{max}$，表示该点在裁剪窗口上边界的上边，则第 4 位置 1，否则置 0。

具体代码如下：

```python
def encode(x, y, x_min, y_min, x_max, y_max):
    res = 0b0000
    inside, left, right, down, up = 0b0000, 0b0001, 0b0010, 0b0100, 0b1000
    if x < x_min:
        res |= left
    elif x > x_max:
        res |= right
    if y < y_min:
        res |= down
    elif y > y_max:
        res |= up
    return res
```

判断线段是否完全在裁剪窗口内或外可采用对两个端点的区域码进行逻辑与操作的方法，根据线段与裁剪窗口的关系可分三种情况处理:

1. **线段完全在裁剪窗口之内**

   两个端点的区域码均为 0000，则该线段完全在裁剪窗口之内。

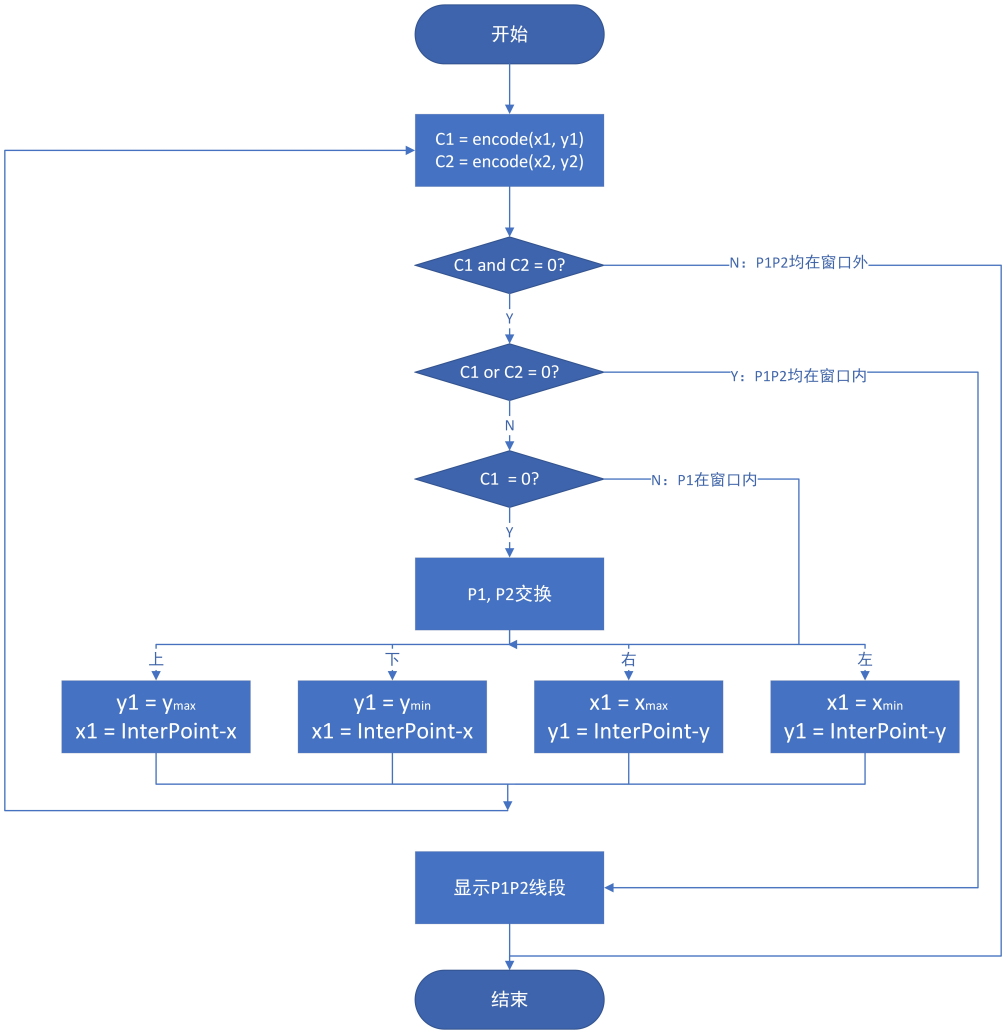
2. **线段完全在裁剪窗口之外**

   两个端点的区域码与操作的结果不为 0000，则该线段完全在裁剪窗口之外。

3. **其它**

   需进行求交计算，求交的过程为：首先，对一条线段的外端点与一条裁剪边界比较来确定应裁剪掉多少线段。然后，对线段的剩下部分与其他裁剪边界比较，直到该线段完全被舍弃或者找到位于窗口内的一段线段为止(即线段完全可见，则不需进一步判断)。可按上、下、 右、左的顺序用裁剪边界检查线段的端点。在算法实现时，不必把线段与每条窗口边界依次求交，只需按顺序检测到端点区域码的某位不为 0 时，才把线段与对应的窗口边界进行求交。

流程图如下：



2.8.2. Liang-Barsky算法

Liang-Barsky算法的原理是：将待裁剪线段及裁剪矩形窗口均看作点集，那么裁剪结果即为两点集的交集。设P_1P_2所在直线(或其延长线)与裁剪窗口的两交点为Q_1Q_2，则Q_1Q_2为诱导窗口，它是一维的，即将二维裁剪问题化简为一维裁剪问题。

在P_1P_2上坐标的参数表达式为：$P = P_1 + u(P_2 - P_1)$，假设Q_1, Q_2的参数分别是u_1, u_2，且$u_1 < u_2$，则根据P_1P_2和Q_1Q_2的四种位置关系，不难得出可见部分的区间为$[\max(u_{P_1}, u1), \min(u_{P_2}, u2)]$。

因此，参数化形式的裁剪条件为：

$$\begin{cases} x_{min} \le x_1 + u \cdot \Delta x \le x_{max} \\ y_{min} \le y_1 + u \cdot \Delta y \le y_{max} \end{cases}$$

可统一表示为：$u \cdot p_k \le q_k$。其中，$k = 1, 2, 3, 4$，对应裁剪窗口的左、右、上、下边界；参数p、q 定义为：

$$\begin{cases} p_1 = -\Delta x, q_1 = x_1 - x_{\min} \\ p_2 = \Delta x, q_2 = x_{\max} - x_1 \\ p_3 = -\Delta y, q_3 = y_1 - y_{\min} \\ p_4 = \Delta y, q_4 = y_{\max} - y_1 \end{cases}$$

- 若$p_k = 0$，直线平行于裁剪边界之一。此时，如果同时满足$q_k < 0$，则线段完全在边界外；如果同时满足$q_k \ge 0$，线段平行于裁剪边界，并且在窗口内；
- 当$p_k < 0$, 线段从裁剪边界延长线的外部延伸到内部；
- 当$p_k > 0$，线段从裁剪边界延长线的内部延伸到外部。

当p_k非零时，可计算出线段与边界k 或延长线交点的u值：$u = q_k / p_k$；

具体算法的**实现过程**如下：

(1) 将线段交点的参数初始化为 $u1=0$，$u2=1$；
(2) 对于每个$k=1$，2，3，4，判断p_k

- 当$p<0$ 时，$u1=\max(q_k/p_k,u1)$；
- 当$p>0$ 时，$u2=\min(q_k/p_k,u2)$；
- 如果$u1>u2$，则舍弃该线段，否则，更新的u 值求出交点，缩短线段；
- 如果$p_k=0\wedge q_k<0$，舍弃该线段；

(3) 将更新的u_1,u_2代入得到新的端点

3. 系统介绍

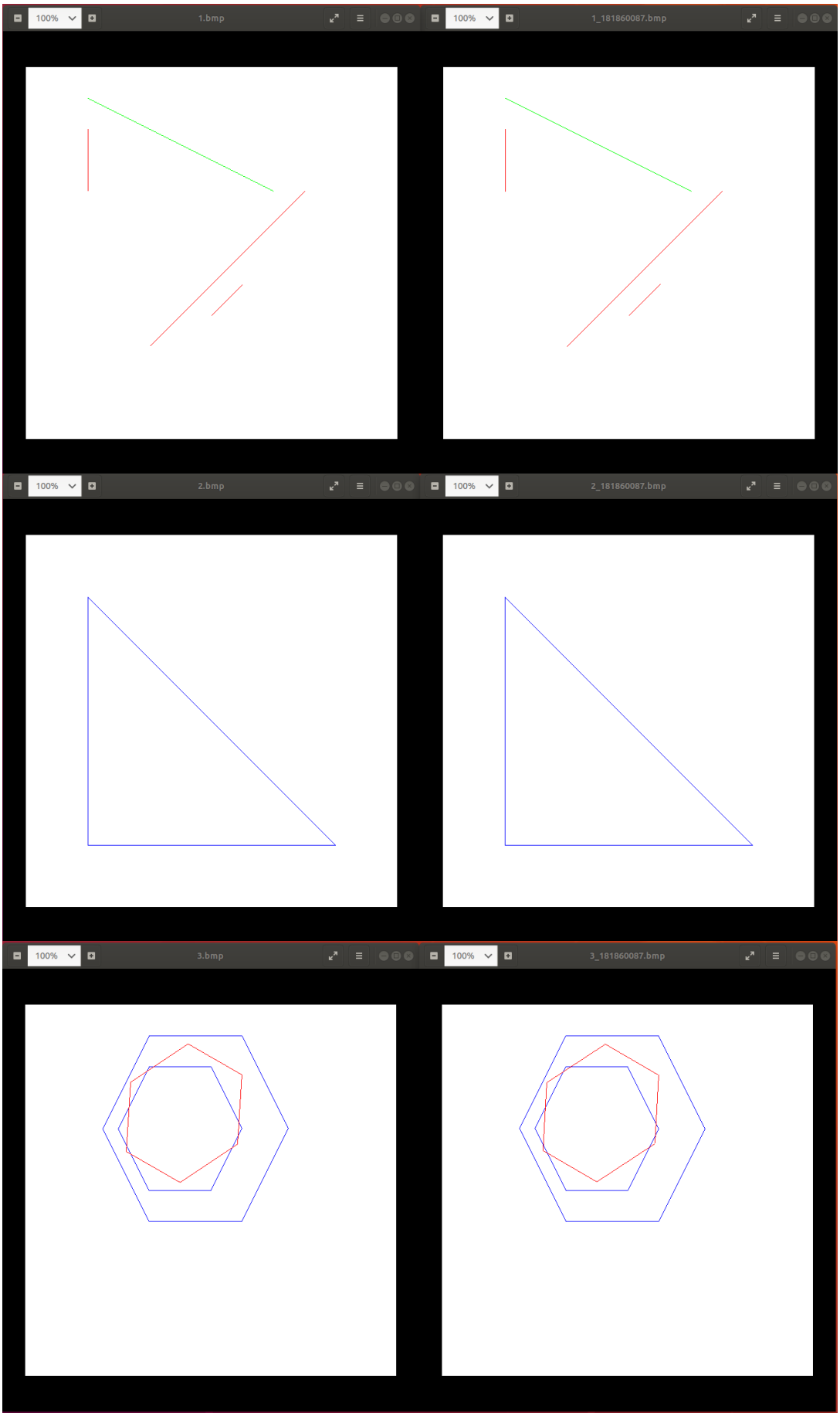
3.1.CLI

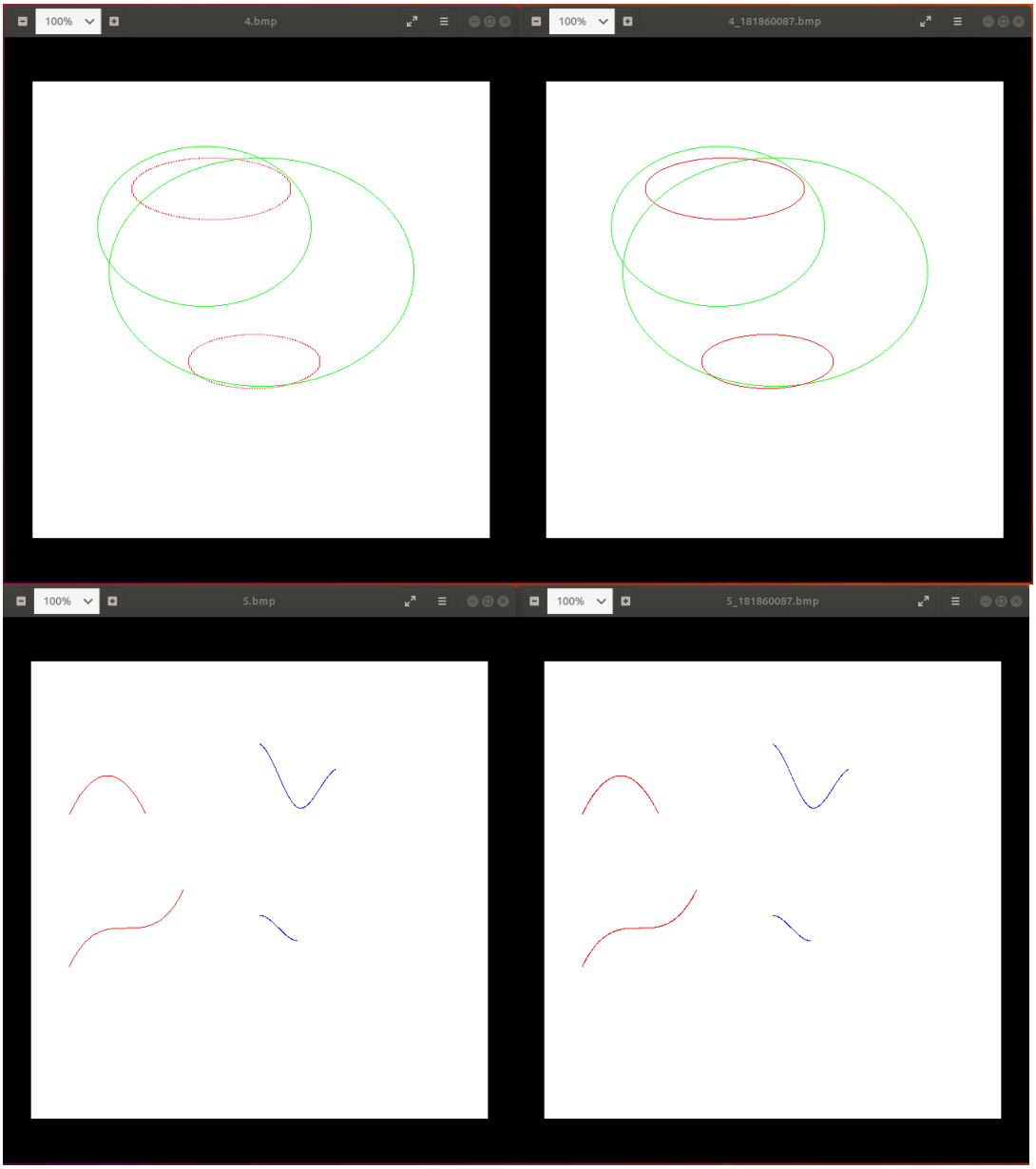
使用PIL中的Image类型进行图形的绘制；

绘制的过程如框架代码是通过扫描txt文件夹，逐句分析语义，然后调用algorithm文件中的算法得到相应的点存储到字典类型中，在savecanvas指令被扫描到的时候进行遍历绘制图片的p_list，最后调用Image类型进行输出。

（10月更新）

根据助教给的测试文件，生成的图像与可供参考的结果图片进行比对，基本一致，对比图如下（左侧为参考图片，右侧为生成图片）：





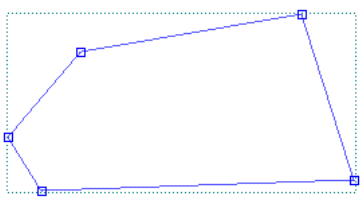
3.2. GUI

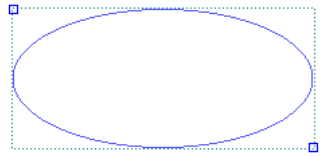
3.2.1.绘制部分

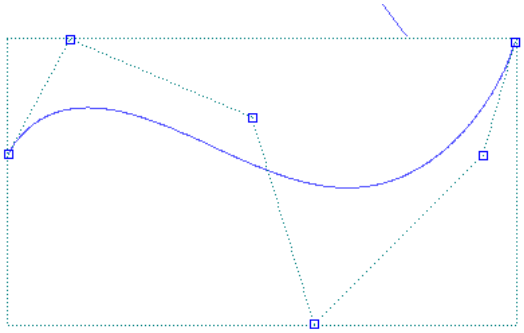
画布部分是在`MyCanvas`和`MyItem`两个类中实现，

`MyItem`继承自`QGraphicsItem`，下面简单阐述几个重要方法的实现方法和作用

- 初始化方法：初始化图元的ID、类型、参数、绘制算法、是否选中，并增加了一个`pencil`变量，类型是`QPen`，用来调整画笔的颜色和粗细；
- `paint`：这部分的重写主要参考cli中调用各个绘制算法的过程，这里增加了图元选中时标记参数点的代码，实现方法是调用`QPainter`中的`drawRect`方法，在参数点的附近画大小为`6*6`的矩形，并且将`boundingRect`的样式调整为虚线，具体效果如下：







- `boundingRect`：这部分主要是重写图元的边界矩形，对于直线和椭圆这类只有两个参数点的边界比较好找，对于曲线和多边形这类多参数点的边界，求出所有参数点中x坐标以及y坐标的最大值和最小值，最小值作为矩形的左上顶点，最大值和最小值的差值作为宽和高。
- `translate` / `rotate` / `scale_` / `clip`：这四个方法主要是调用四个编辑图元的算法，需要注意的是`scale_`方法不能命名为`scale`，原因是与python中的某个方法重名了。

`MyCanvas`继承自`QGraphicsView`，下面简单阐述几个重要方法的实现方法和作用：

- 初始化方法：初始化主窗口main_window、图元列表list_widget、图元字典`item_dict`、选中图元的编号，当前的状态，缓存的图元，这些都是在原有框架有的，
  增加的变量——

变量名	含义
`self.pen_color/self.pen_width`	更改画笔的粗细和颜色
`self.x_center/self.y_center`	图元的中心，记录旋转、缩放中心
`self.x_old/self.y_old`	旋转、缩放过程时上一次停留的坐标，从而通过这个坐标、新的坐标和图元中心来计算旋转角度和缩放倍数
`self.current_id`	暂存当前用鼠标选中的图元编号
`self.draw_item`	存储裁剪过程中裁剪窗口的数据，类型为MyItem
`self.changed_point_id`	存储变换参数点里面选中点的编号
`self.mode`	区分用滚轮是否有设定旋转中心

- 几个不同的鼠标操作的方法，分别为`mousePress`、`mouseMove`、`mouseRealease`三个事件，重写这几个方法并不是很难，主要过程是对当前处于的状态进行区分处理，根据鼠标的坐标新建图元加入到`scene()`队列和`item_dict`，或选中一个图元，更新正在绘制或编辑的图元，最后使用updatescene()函数可以提示框架进行图元的更新。

  需要注意的有几点：

  1. 计算鼠标拖动的旋转角度和缩放倍数。

     旋转角度参考了一篇博客中计算向量夹角的角度，原理是用余弦定理，主要使用了numpy中的点乘dot；

     缩放倍数的计算的原理是向量模的比，即计算两个向量之间的模，且相同向量的点乘就是它的模，所以直接返回两个向量分别点乘后的分数就能得到缩放倍数。

2. 一个关于鲁棒性的问题。在画曲线和多边形的时候，都存在一种状态，即还没有正常结束绘制曲线或多边形，此时有两种情况：一是切换了其他状态；二是点击了图元列表中的某一项。我的处理是完成那个没有结束绘制的图元，然后将其加入图元列表，所以在进行不同模式切换或点击图元列表的时候都需要检查缓存的图元是否为空，如果缓存的图元非空，表明此时还有正在绘制的图元，此时需要特殊处理，当将缓存的图元加入画布后再进行后面的处理。

- wheelEvent：重写滚轮事件，通过滚轮自带的角度来模拟旋转角度或缩放倍数，同时这样也可以实现随意设置旋转/缩放中心
- keyPressEvent：重写键盘事件，当按下Esc按钮时可以清空选中图元的状态，将其转换成非选中的状态，这样就能实现取消编辑的状态。

3.2.2. 界面布局

界面部分是在`MainWindow`这个类中实现，继承自`QMainWindow`，下面为实现的主要思路：

1. 整体布局使用QDockWidget停靠组件，共有三个Dock，分别为工具栏、绘制区域、图元列表，然后将`tool_widget`、`canvas_widget`、`list_widget`分别加进去，这样可以实现工具栏停靠的效果，且可以自由进行排版。
2. 工具栏`tool_widget`使用QGridLayout网格布局，可以将窗口分割成行和列的网格来进行排列，这样能实现工具按钮的排版；
3. 几个工具栏中的按钮使用QToolButton，原因是QToolButton提供了一个快速访问按钮，可以在工具栏内部使用，且工具按钮通常不显示文本标签，而是显示图标。如果遇到需要选择不同模式的工具，调用`setPopupMode(QToolButton.MenuButtonPopup)`方法即可；
4. 绘制区域`canvas_widget`即`Mycanvas`，可以调用`setFixedSize`方法来将画布的大小固定，这样画布就不会因为窗口的大小调整出现滚动条；
5. 菜单中的动作使用QAction，可以添加不同的图标图片，使用起来更直观；
6. 重写closeEvent，对关闭程序做确认操作，检查当前画布上是否有图元，若有图元，那么直接关闭窗口可能会误操作，所以关闭之前需要确认该操作；若没有图元，那么可以直接退出程序；

3.2.3.拓展功能

1. 左键点击选择图元

   在`Mycanvas`中增加一个`selection`的状态，当处于这个状态时，鼠标点击会调用`itemAt(x, y)`方法，来获取一个选中的item，然后再`item_dict`中查找这个图元，获取它的id，并将这个id保存至`current_id`，鼠标松开后调用`selection_changed`方法，即将选中图元编号设置为`current_id`。

2. 坐标变换

   实现这个功能的目的是能编辑绘制的图元，可以点击选中一个参数点，然后拖动进行这个参数点的变换。实现方法是在`Mycanvas`中增加一个`change`的状态，当处于这个状态且存在选中图元编号时，鼠标点击会计算当前点击的点和选中图元所有参数点的距离，当这个距离小于等于4时，就会将这个参数点的编号保存至`changed_point_id`，鼠标拖动时将`changed_point_id`的参数点的坐标改变成新的坐标即可。

3. 设置画笔粗细

   这个功能实现起来比较简单，只需要获取该边的画笔宽度，然后将这个参数最终传给`MyItem`中的`self.pen_width`，`MyItem`中会调用`QPen().setPenWidth(w)`来设置画笔宽度。

4. 导出画布命令

   实现这个功能的目的是能将GUI中绘制的图元，通过cli程序来生成相同的效果，所以可以将当前画布中的图元以正确的命令的形式导到外部文件中。实现的方法是遍历`item_dict`中的所有图元，然后将其参数保存为cli中正确的格式，需要注意的是在导出图元类型的时候，类型的首字母要大写，可以调用python中的`capitalize()`方法。

5. 主题切换

   实现这个功能，首先定义一个qss文件，然后编写各种控件（QLabel，QLIneEdit，QPushButton等）的样式，这里的语法参考css，最后使用QMainWindow加载样式，调用`setStyleSheet`，这样就可以让整个应用程序改变样式了。因此，在qss这个文件夹中编写了三个qss文件，区别是在maiwindow中的背景颜色等，由于是首次使用qss，所以这块参考了github中一些开源的样式。

6. 复制粘贴yItem

   首先在`MainWindow`中增加了一个`copy_item`变量，类型为`MyItem`，当有选中的图元时，将图元的信息复制给`copy_item`，这里不能直接使用等号的原因是存在浅拷贝的问题，所以这里实现的方法是直接创建一个新的Item，参数通过选中的图元复制过去，这样就实现了复制的功能；粘贴部分实现是，当存在复制图元时，复制按钮状态转变成可点击，然后将复制的图元的所有参数点加上$(30, 30)$这个向量，这样就能区分复制的图元和原来的图元，而不是重叠在一起。

7. 启动动画

   启动动画是在使用PyCharm等其他软件的时候的灵感，使用QSplashScreen类实现在程序启动过程中显示启动画面的功能，重写原来的QSplashScreen类，将开机启动画面设置为已有的图片。

8. 丰富图形化界面

   所有按钮的图标主要从图标下载，ICON(SVG/PNG/ICO/ICNS)图标搜索下载 - Easyicon这个网站上下载得到，然后将所有控件进行重新的设置和排版，这块代码主要是现在`MainWindow`中的初始化方法中。

3.2.4. 遇到的问题以及解决方法

- **list_widget编号异常**：在10月的时候绘制一个图元有时编号会增加1或2，且在切换模式的时候编号也会增加。

  解决方法：通过追踪GUI框架中`item_cnt`，发现`item_cnt`的增加是在`add_id`这个method中，再追踪这个method的调用情况，发现在槽函数中调用`start_draw_line`这些绘制或编辑图元的方法时，会将`item_cnt`加1，而在`Mycanvas`中的`finish_draw`也会调用这个方法，导致绘制一次图元的时候`item_cnt`也会加1，编辑图元的时候`item_cnt`不会增加。由此增加一个新的函数`get_id`，当槽函数中想要获取当前`item_cnt`时调用`get_id`，而在完成一个新图元的绘制之后，`finish_draw`中调用`add_id`。

  ```
  def add_id(self):
          _id = str(self.item_cnt + 1)
          self.item_cnt += 1
          return _id
  def get_id(self):
          _id = str(self.item_cnt)
          return _id
  ```

- **画笔设置界面设置**：在11月增加的画笔的设置

  解决方法：使用QSpinBox作为宽度设置，使用QColorDialog作为颜色设置。

- **双击导致异常**：在11月发现在绘制直线或椭圆的时候快速点击会导致出现如下bug，且编号会异常地增加2：

  ```
  Traceback (most recent call last):
    File "cg_gui.py", line 288, in mouseReleaseEvent
      self.temp_item.p_list[1] = (x, y)
  AttributeError: 'NoneType' object has no attribute 'p_list'
  ```

解决方法：出现上述bug的原因是连续点击会将第一次点击作为一个新的`MyItem`，不会出现问题，而第二次点击`temp_item`此时是None，因此在相应的位置加上`temp_item`非None的限制判断即可。另外，编号的原因是在`finish_draw`中调用`add_id`的时候也没有判断`temp_item`是否非None，所以需要加上该判断。

- **多边形/曲线未完成绘制时的情况**：会出现图元列表中编号异常的问题

  解决方法：这块关于鲁棒性的问题主要是通过检测当前是否有缓存的图元来解决的，编号的问题进行多次调试后基本没有问题。

4. 总结

- 9月总结：
  1. 9月主要完成了直线两种算法、椭圆的中心圆生成算法、图元的平移、旋转、缩放算法的实现，以及CLI程序的实现；
  2. 初步掌握了几种常见图元的生成算法和变换算法；
  3. 由于时间紧张，GUI的进度还未开始，争取在10月完成大部分的功能。
- 10月总结：
  1. 解决bug，绘制椭圆的时候如果两个点重合或在一条直线上返回这个点或一条直线；
  2. 解决CLI中的两个小问题：一是由于一开始规定图片的左下角为坐标原点，但是经过助教确认右上角为原点，所以在进行点的绘制时，将点的y坐标进行翻转，现已修改为：

     ```
     for x, y in pixels:
         canvas[y, x] = color
     ```

     二是在旋转的时候，一开始传入的是`-r`这个参数，但是根据上面的变动，直接传入`r`。
  3. 完成了GUI绘制图元的部分，通过模仿naive画直线的逻辑，初步了解整个GUI的框架。
- 11月总结：
  1. 解决bug：GUI中右侧list_widget编号异常；
  2. 增加鼠标双击事件：绘制多边形以及曲线的时候能通过双击结束绘制；
  3. 完成GUI中编辑图元的部分，修改了原有框架中存在的部分bug。
- 12月总结：
  1. 解决了多边形和曲线未正常完成绘制时点击其他按钮或选中其他图元的鲁棒性，且解决了编号异常的bug；
  2. 丰富图形界面：通过重写MainWndow的初始化函数，由于是第一次使用PyQt来设计UI，所以在真正实现的时候还是经过了多次的调试来确定最后的呈现效果，没有在Qt Creator中来的直观。
  3. 本次大作业基本功能还是比较容易的，但是在调试list_widget中编号以及一些鲁棒性问题还是花了不少时间，另外图形界面的设计没有很多的经验，所以参考了部分开源的样式。最大的收获是：不同的图形学算法以及PyQt开发的经验

5. 参考文献

[1] 孙正兴.计算机图形学教程[M].机械工业出版社:北京,2006:56-.

[2] 青空哲也.图形学入门(1)——直线生成算法（DDA和Bresenham）[EB/OL].https://www.cnblogs.com/LiveForGame/p/11706904.html,2019-10-21.

[3] fengye2two.图像旋转算法[EB/OL].https://blog.csdn.net/fengye2two/article/details/83148607,2018-10-18.

[4] Abhishek Kataria.Ellipse Algorithm | Computer Graphics[EB/OL].https://www.includehelp.com/basics/ellipse-algorithm.aspx,2018-8-25.

[5] MTU.Finding a Point on a Bézier Curve: De Casteljau's Algorithm[EB/OL].http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-casteljau.html

[6] HachiLin.B样条曲线——de Boor递推算法实现[EB/OL].https://blog.csdn.net/Hachi_Lin/article/details/89812126,2019-05-04.

[7] python 线性代数：计算向量夹角.http://blog.sina.com.cn/s/blog_15382c6c80102whvh.html

[8] jia666666.PyQt5布局管理之QGridLayout(三).https://blog.csdn.net/jia666666/article/details/81701176

[9] jia666666.PyQt5高级界面控件之QDockWidget(八).https://blog.csdn.net/jia666666/article/details/81669995

[10] Python3 capitalize()方法.https://www.runoob.com/python3/python3-string-capitalize.html

[11] 青阳不会被占用.PyQt5之QPen画笔类学习_青阳的博客-CSDN博客https://blog.csdn.net/qq_16668303/article/details/96995288)

[12] jia666666.PyQt5基本控件详解之QSpinBox(十).https://blog.csdn.net/jia666666/article/details/81534431

[13] 李济雄.第八节 PyQt5之QToolButton对象（快速访问按钮）.https://blog.csdn.net/weixin_43496130/article/details/104237740

[14] jia666666.PyQt5图形和特效之加载QSS（十）.https://blog.csdn.net/jia666666/article/details/81875622

[15] Lawrence_121.利用QSplashScreen类实现在程序启动过程中显示启动画面的功能 https://blog.csdn.net/m0_37806112/article/details/80472661