

# 《计算机图形学》4 月报告

211240045, 杨镇源, [211240045@smail.nju.edu.cn](mailto:211240045@smail.nju.edu.cn)

2024 年 4 月 23 日

## 目录

<b>1 综述</b>	<b>3</b>
1.1 2024 年 3 月 . . . . .	3
1.2 2024 年 4 月 . . . . .	3
<b>2 图元绘制算法介绍</b>	<b>3</b>
2.1 绘制直线 . . . . .	3
2.1.1 DDA 算法 . . . . .	3
2.1.2 Bresenham 算法 . . . . .	3
2.1.3 方法对比 . . . . .	4
2.2 绘制多边形 . . . . .	4
2.3 绘制椭圆 . . . . .	4
2.3.1 中心椭圆生成算法 . . . . .	4
2.4 绘制曲线 . . . . .	4
2.4.1 Bezier 算法 . . . . .	4
2.4.2 B-spline 算法 . . . . .	5
2.4.3 方法对比 . . . . .	6
<b>3 图元变换算法介绍</b>	<b>7</b>
3.1 图元平移 . . . . .	7
3.1.1 算法原理 . . . . .	7
3.1.2 算法实现 . . . . .	7
3.2 图元旋转 . . . . .	7
3.2.1 算法原理 . . . . .	7
3.2.2 算法实现 . . . . .	7
3.3 图元缩放 . . . . .	8
3.3.1 算法原理 . . . . .	8
3.3.2 算法实现 . . . . .	8

<b>4</b>	<b>用户交互功能介绍</b>	<b>8</b>
4.1	gui 重置画布 . . . . .	8
4.2	gui 保存画布 . . . . .	8
	4.2.1 实现效果 . . . . .	9
4.3	gui 设置画笔颜色 . . . . .	9
	4.3.1 实现效果 . . . . .	10
<b>5</b>	<b>总结</b>	<b>11</b>

# 1 综述

## 1.1 2024 年 3 月

- 完成绘制直线 (DDA 算法)
- 完成绘制直线 (Bresenham 算法)
- 实现 gui 重置画布
- 实现 gui 保存画布

## 1.2 2024 年 4 月

- 完成绘制多边形 (DDA 算法)
- 完成绘制多边形 (Bresenham 算法)
- 完成绘制椭圆 (中点圆生成算法)
- 完成图元平移
- 完成图元旋转
- 完成图元缩放
- 实现 gui 设置画笔颜色

# 2 图元绘制算法介绍

## 2.1 绘制直线

### 2.1.1 DDA 算法

**算法原理:** DDA 算法的核心思想是通过计算斜率来确定每个像素点的位置,从而形成一条连续的直线。通过坐标轴上以单位间隔取样 ( $\Delta x = 1$  或  $\Delta y = 1$ ), 因为取单位间隔, 所以显见对应的  $\Delta y = k, -k$  或  $\Delta x = \frac{1}{k}, -\frac{1}{k}$  (记直线为  $y = kx + b$ )。当起始点在左侧取值, 起始点在右侧取负值。

**算法改进:** 可利用直线构成的连贯性, 通过将增量  $k$  和  $\frac{1}{k}$  分离成整数和小数部分从而使所有的计算都简化为整数操作来改善 DDA 算法的性能。

### 2.1.2 Bresenham 算法

**算法原理:** 通过坐标轴上以单位间隔取样 ( $\Delta x = 1$  或  $\Delta y = 1$ ), 不失一般性取  $0 \leq k < 1$ 。假设  $(x_k, y_k)$  为已经确定的像素坐标, 那么下一个像素坐标为  $(x_k + 1, y_k)$  或  $(x_k + 1, y_k + 1)$ , 确定  $y$  轴坐标选择哪一个的依据是判断直线和  $x = x_k + 1$  的交点的  $y$  轴坐标与  $y_k + 1, y_k$  哪个的绝对差值更小。若与  $y_k$  绝对差值更小, 则下一个点选择  $(x_k + 1, y_k)$ ; 若与  $y_k + 1$  绝对差值更小, 则下一个点选择  $(x_k + 1, y_k + 1)$ 。

**算法改进:** 为进一步提高算法效率, 可以利用线段本身的对称性。用 Bresenham 算法产生起点一侧的半条线段, 至于终点一侧的半条线段, 可以看作以终点为起点线段的生成。起点一侧的线段像素坐标在  $x$  或  $y$  方向每前进一个坐标单位, 终点一侧的线段像素坐标就在  $x$  或  $y$  方向后退一个坐标单位。

### 2.1.3 方法对比

DDA 算法的优点在于适用面广，实现简单。但是它存在一个问题，在计算斜率时会产生精度损失，从而使得绘制出来的直线可能出现明显的锯齿状。因此，在对线条的精度有较高要求的情况下，可以采用 Bresenham 算法。

Bresenham 算法通过整数计算来绘制线条，避免了 DDA 算法中的精度损失问题。因此，Bresenham 算法具有更高的绘制速度和较好的像素级别的控制，可以在需要绘制直线的情况下带来更好的性能和画质。

总的来说，Bresenham 算法比 DDA 算法绘制直线更准确高效。

## 2.2 绘制多边形

绘制多边形的方法基于绘制直线的算法，对于多边形的每条边，获得其两端点后利用直线绘制算法进行绘制即可。

## 2.3 绘制椭圆

### 2.3.1 中心椭圆生成算法

**算法原理：**算法类似于 Bresenham 算法，不失一般性，考虑中心在原点处的椭圆，研究第一象限。将第一象限中椭圆切线绝对值等于 1 所对应的切点记作临界点  $P$ ， $P$  上方的点  $\frac{dy}{dx} < 1$ ， $P$  下方的点  $\frac{dy}{dx} > 1$ ，由此可利用 Bresenham 算法解决选点问题。根据参数方程定义椭圆函数为：

$$f_{ellipse}(x, y) = b^2x^2 + a^2y^2 - a^2b^2$$

在  $P$  上方的点，取  $x$  方向单位步长，再通过决策函数判断真实值与两候选像素之间哪个位置更近，更新对应的  $y$  值；在  $P$  下方的点，取  $y$  方向单位步长，再通过决策函数判断真实值与两候选像素之间哪个位置更近，更新对应的  $x$  值。又因为椭圆四个象限是互相对称的，可以通过改变对应的符号补全其余象限，最后结合中心点的实际坐标即可计算出待绘制椭圆的点坐标。

## 2.4 绘制曲线

### 2.4.1 Bezier 算法

**算法原理：**Bezier 算法是由多边形控制，逼近多边形的一种曲线。其做法是通过 Bezier 基函数来得到绘制点的坐标。Bezier 基函数的原型比较复杂，但我了解到 Bezier 曲线基函数已被证明可简化为伯恩斯坦基函数，其形式为：

$$C_n^i t^i (1-t)^{n-i}$$

有了基函数，自然可以用参数  $t$ ，取某一个步长，来近似绘制曲线。下面以 4 个点控制的曲线为例，介绍算法执行步骤，其中算法接受参数中的顶点序列应当是有序的：

1. 对两两相邻点求中点，得到三个中点
2. 对得到的三个中点再次求中点，得到两个中点

3. 对得到的两个中点再次求中点，得到一个中点
4. 将得到的最接近曲线的 7 个点根据最后计算得到的中点对半分为每方 4 个，即： $P_0^0, P_0^1, P_0^2, P_0^3$  和  $P_1^3, P_1^2, P_1^1, P_1^0$
5. 分别对这两组 4 个点调用上述算法，递归直到误差在可接受范围内

### 2.4.2 B-spline 算法

**算法原理：**由于 Bezier 曲线调整的时候有“牵一发而动全身”的缺点，因此引入 B-spline (B 样条) 曲线。B 样条曲线同样拥有自己的基函数，这些基函数在定义曲线时起到关键作用。与贝塞尔曲线不同，B 样条曲线的基函数具有局部影响的特点，即当调整曲线上的一个控制点时，只有曲线的一小部分会受到影响，而不是整个曲线。这种局部影响的特性是通过基函数的支撑区间 (support interval) 概念来实现的。

支撑区间指的是基函数值非零的区间。对于三次贝塞尔曲线，其四个基函数的支撑区间都是整个定义域  $[0,1]$ ，这意味着任何一个控制点的调整都会对曲线的整体形状产生影响。而 B 样条曲线的基函数则具有更短的支撑区间，这允许控制点的局部调整，从而提高了曲线的可控性和灵活性。

B 样条曲线的基函数产生使用递推形式，即 de-BoorCox 递推定义：

$$\begin{cases} N_{i,1}(u) = \begin{cases} 1, & u_i < u < u_{i+1} \\ 0, & o.t. \end{cases} \\ N_{i,k}(u) = \frac{u-u_i}{u_{i+k-1}-u_i} N_{i,k-1}(u) + \frac{u_{i+k}-u}{u_{i+k}-u_{i-1}} N_{i+1,k-1}(u) \end{cases}$$

规定  $\frac{0}{0} = 0$

其中  $k$  是阶数， $u$  是参数

均匀 B 样条基函数是指当节点向量按照等间隔排列，即节点向量为  $0, 1, 2, \dots, n+k$  时，其中  $n$  是控制点的数量，而  $k$  是基函数的阶。这种均匀分布的节点向量保证了 B 样条曲线的基函数在整个定义域内具有相同的支撑区间，从而使得曲线在各个部分的局部控制能力相同。相对地，准均匀 B 样条基函数则是指节点向量的前  $k$  个和后  $k$  个元素相等，而中间的元素则均匀分布。这种分布方式允许基函数在曲线的某些部分具有更大的控制力，而在其他部分则相对较小。

针对实验要求使用均匀 B 样条函数的情况，为了优化性能，我们可以选择一个简化的节点向量，即仅包含从 0 到  $n+k$  的连续整数序列。在实际的编程实现中，代码可以按照如下形式编写：

```

Function DeBoorCox( $i, k, u$ ):
    if  $k == 1$  then
        if  $i \leq u \wedge u < i + 1$  then
            return 1
        end
    else
        return  $\frac{u-i}{k-1} \times \text{DeBoorCox}(i, k-1, u) + \frac{i+k-u}{k-1} \times \text{DeBoorCox}(i+1, k-1, u)$ 
    end
end

```

Algorithm 1: De Boor-Cox Algorithm

### 2.4.3 方法对比

- 控制点影响范围：
  - 贝塞尔曲线：任一控制点的移动都会影响整条曲线。
  - B 样条曲线：控制点主要影响曲线的局部区域。
- 曲线表达式：
  - 贝塞尔曲线： $\vec{P}(t) = \sum_{i=0}^{n-1} \vec{p}_i B_{i,n}(t)$ .
  - B 样条曲线： $\vec{P}(t) = \sum_{i=0}^{n-1} \vec{p}_i B_{i,d}(t)$ .
- 参数  $t$  的取值范围：
  - 贝塞尔曲线： $t$  的取值通常为  $[0, 1]$ .
  - B 样条曲线： $t$  的取值范围更广，由节点向量定义。
- 局部性与连续性：
  - 贝塞尔曲线：不具备局部性，但连续性容易满足。
  - B 样条曲线：具有局部性，可以构造出高阶连续的曲线。
- 几何属性：
  - 贝塞尔曲线：几何属性简单，易于理解。
  - B 样条曲线：具有变差缩减性、凸包性、仿射不变性等复杂几何属性。
- 应用场景：
  - 贝塞尔曲线：适用于简单图形设计，如字体设计。
  - B 样条曲线：适用于复杂曲面建模，如 CAD/CAM 系统。
- 计算复杂度：
  - 贝塞尔曲线：计算简单，适合实时应用。
  - B 样条曲线：计算复杂，提供更高的设计灵活性。

- 节点向量 (Knot Vector):

- 贝塞尔曲线: 不使用节点向量。
- B 样条曲线: 使用节点向量来定义曲线的局部控制范围。

## 3 图元变换算法介绍

### 3.1 图元平移

#### 3.1.1 算法原理

不妨设  $dx$  为水平方向平移量,  $dy$  为垂直方向平移量。

$x, y$  为原始坐标,  $x', y'$  为平移后坐标, 则:

$$\begin{cases} x' = x + dx \\ y' = y + dy \end{cases}$$

#### 3.1.2 算法实现

需要实现或改动的函数有:

- start\_translate 函数
- mousePressEvent 函数
- mouseMoveEvent 函数
- mouseReleaseEvent 函数
- MainWindow 类中连接槽函数, 实现 translate\_action 函数

在 start\_translate 函数将当前系统状态调整为 translate; 在 mousePressEvent 函数中确定平移对象并记录初始位置; 在 mouseMoveEvent 函数中随着鼠标指针移动, 调用 alg.translate 函数更新平移对象的点集坐标, 然后刷新界面。由此, 即可实现平移功能。

### 3.2 图元旋转

#### 3.2.1 算法原理

不妨设  $x_r, y_r$  为旋转中心点坐标,  $\theta$  为旋转角度。

$x, y$  为原始坐标,  $x', y'$  为旋转后坐标, 则:

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

#### 3.2.2 算法实现

需要实现或改动的函数有:

- start\_rotate 函数
- mousePressEvent 函数
- mouseMoveEvent 函数

- mousePressEvent 函数
- MainWindow 类中连接槽函数，实现 rotate\_action 函数

在 start\_rotate 函数将当前系统状态调整为 rotate；在 mousePressEvent 函数中第一次点击确定旋转中心，第二次点击确定旋转起始位置；在 mouseMoveEvent 函数中随着鼠标指针移动，计算顺时针旋转角度参数 r，调用 alg.rotate 函数更新旋转对象的点集坐标，然后刷新界面。由此，即可实现旋转功能。

### 3.3 图元缩放

#### 3.3.1 算法原理

不妨设  $x_f, y_f$  为缩放中心点坐标， $s$  为缩放倍数。

$x, y$  为原始坐标， $x', y'$  为缩放后坐标，则：

$$\begin{cases} x' = x \cdot s + x_f \cdot (1 - s) \\ y' = y \cdot s + y_f \cdot (1 - s) \end{cases}$$

#### 3.3.2 算法实现

需要实现或改动的函数有：

- start\_scale 函数
- mousePressEvent 函数
- mouseMoveEvent 函数
- mouseReleaseEvent 函数
- MainWindow 类中连接槽函数，实现 scale\_action 函数

在 start\_scale 函数将当前系统状态调整为 scale；在 mousePressEvent 函数中第一次点击确定缩放中心，第二次点击确定缩放起始位置；在 mouseMoveEvent 函数中随着鼠标指针移动，计算缩放倍率参数 s，调用 alg.scale 函数更新缩放对象的点集坐标，然后刷新界面。由此，即可实现缩放功能。

## 4 用户交互功能介绍

### 4.1 gui 重置画布

实现 reset\_canvas\_action 函数即可：

- 将所有画好的图形都删掉，并且将各参数重置为初始值
- 通过 QDialog 获取并记录新设置的宽和高
- 将画布的宽和高设置为 Dialog 中得到的宽和高

### 4.2 gui 保存画布

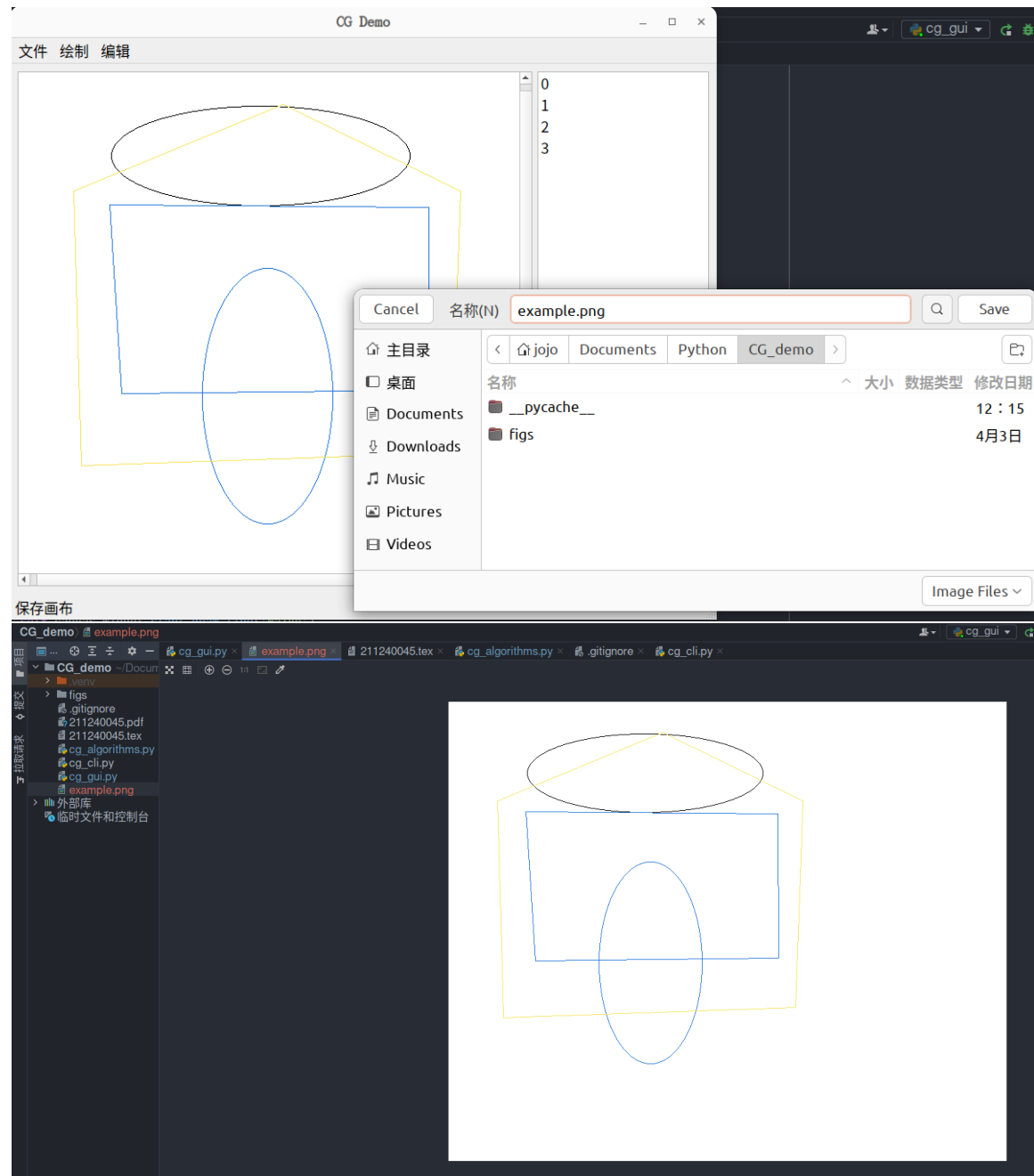
添加 save\_canvas\_act 信号，实现 save\_canvas\_action 槽函数，并将其连接即可：

- 通过 QFileDialog 创建一个文件对话框对象，用于选择保存文件的路径和设置文件名



- 通过 `getSaveFileName` 获取并记录设置的文件名，并且限定文件类型为 jpg、png 和 bmp 格式
- 使用 `Pixmap` 对象，存储绘制的图像内容

#### 4.2.1 实现效果

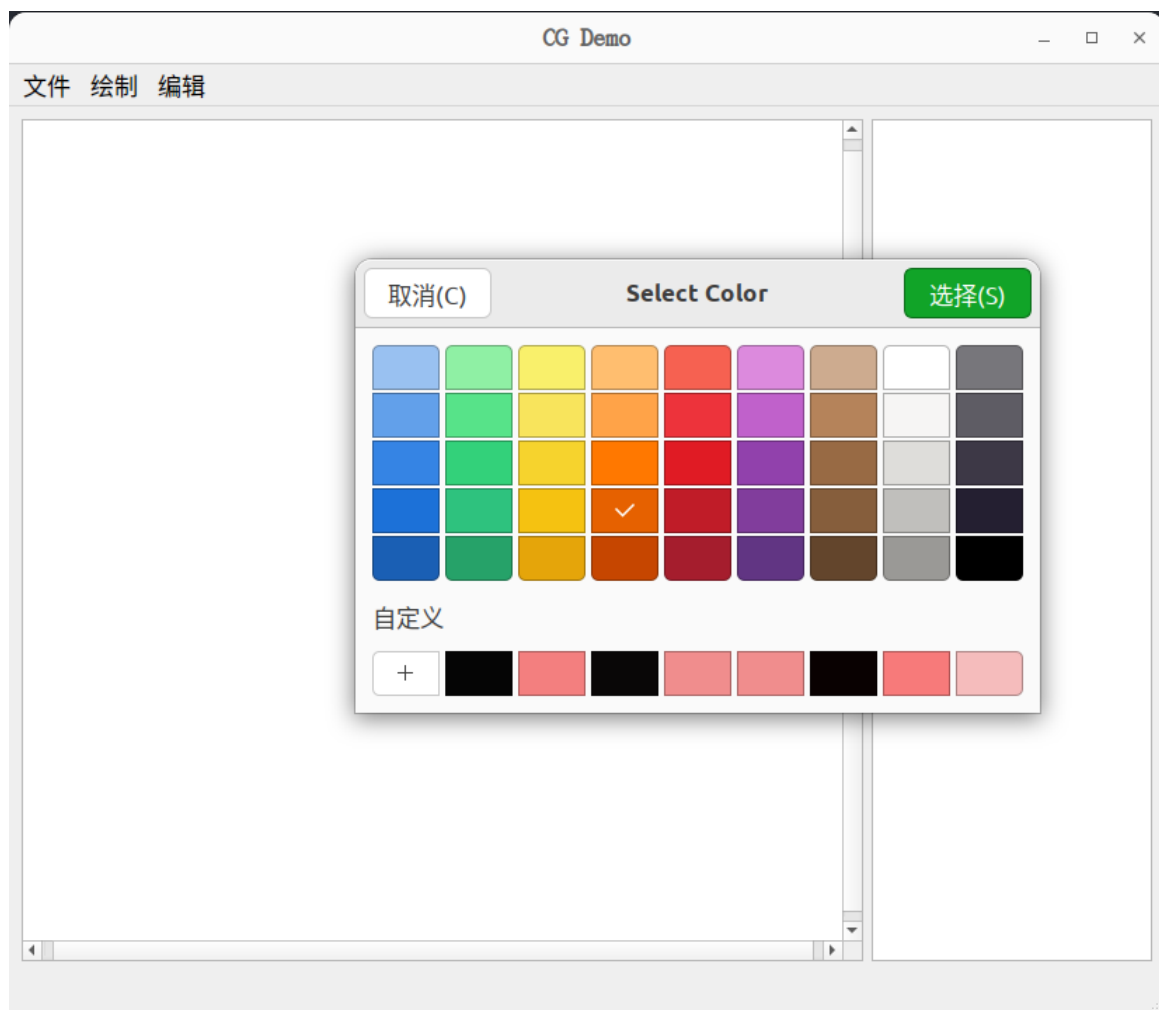


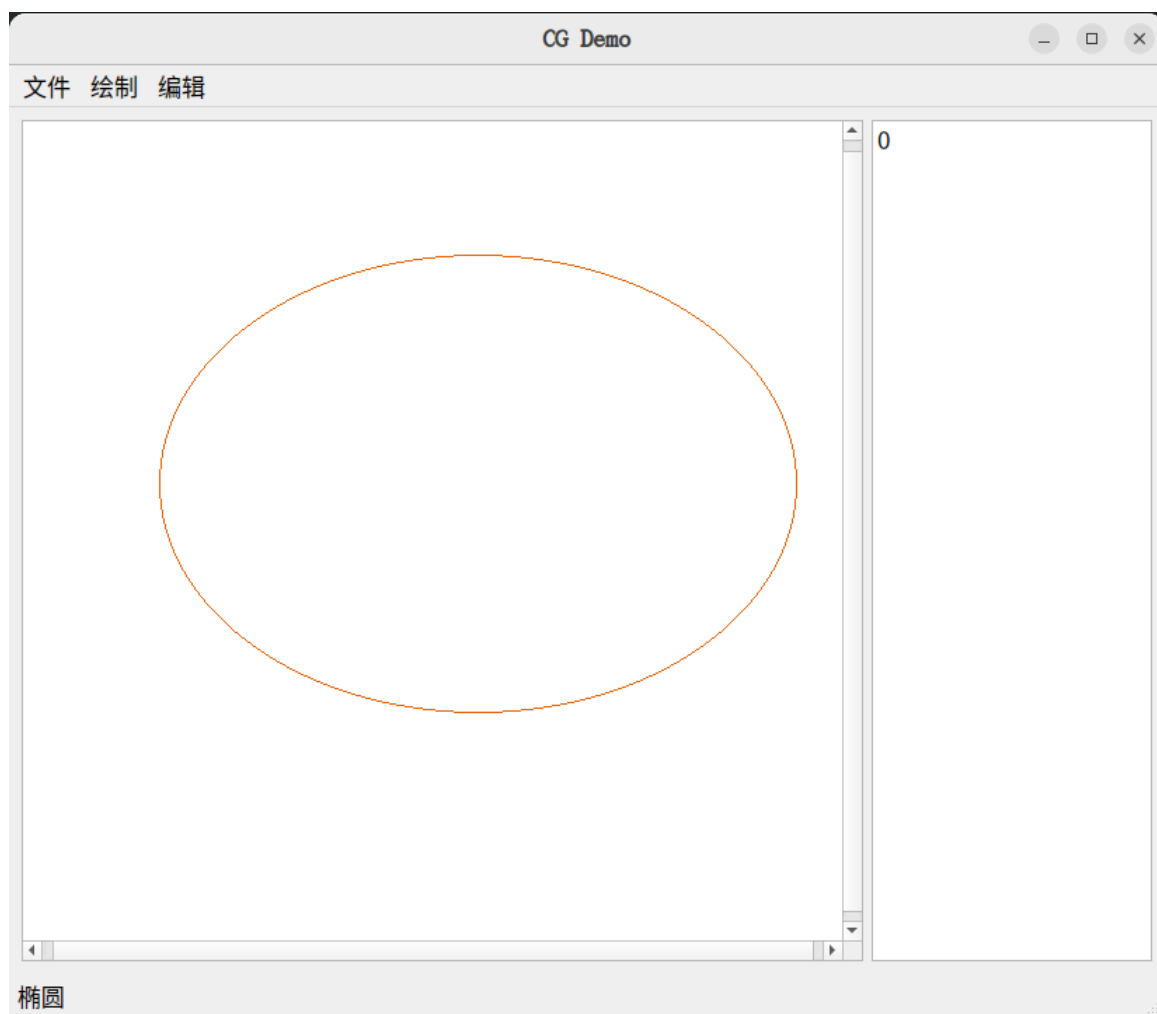
### 4.3 gui 设置画笔颜色

1. 首先实现 `set_pen_action` 函数：

- 调用 QColorDialog 类中的 getColor 函数获得新的颜色值
  - 将颜色值存入画布的 temp\_color 成员中
2. 后在 MyCanvas 类中创建新的图形 Item 时，将 temp\_color 作为参数传入构造函数，初始化图形 Item 的 color 成员
  3. 调用 painter.drawPoint 函数画图前，使用 setPen 函数设置画笔颜色，保证图形颜色符合预期

#### 4.3.1 实现效果





## 5 总结

...

## 参考文献