《计算机图形学》报告

姓名：王智坚
学号：191870202
邮箱：928937364@qq.com

1. 综述

9 月：配置了开发环境。学习了用到的主要算法，完成了部分代码。了解了大作业的程序逻辑和代码的总框架。掌握了常用的测试方法。
10 月：完成了所有算法部分。完成了控制台版本。
11 月：完成了可视化版本。

2. 算法介绍

2.1. 图元生成

2.1.0. 绘制直线

2.1.0.1.DDA 算法

算法原理：

直接利用直线 x 或 y 方向增量Δx 或Δy 的线段扫描算法。使用 x 或 y 方向单位增量间隔（Δx 或Δy=±1）。当斜率绝对值在 0 到 1 之间时，在 x 方向取样；当斜率绝对值大于 1 时，在 y 方向取样。增量(△x 或△y)的取值取决于线段生成方向与坐标轴方向的关系。线段生成方向与坐标轴方向相同取 + 1；线段生成方向与坐标轴方向相反取 − 1。

算法实现：

```
elif algorithm == 'DDA':
    dx = x1 - x0
    dy = y1 - y0
    if dx == 0:
        if y0 > y1:
            x0, y0, x1, y1 = x1, y1, x0, y0
        for y in range(y0, y1+1):
            result.append(x0, y)
    else:
        xk, yk = x0, y0
        m = dy / dx
        if abs(m) <= 1:
            s = 1 if x0 < x1 else -1
            for x in range(abs(dx) + 1):
                result.append((xk, round(yk)))
                xk += s
                yk += s * m
        else:
            s = 1 if y0 < y1 else -1
            for y in range(abs(dy) + 1):
                result.append((round(xk), yk))
                xk += s * m
                yk += s
```

2.1.0.2.Bresenham 算法

算法原理：

第 k 步决策参数为 $p_k=\Delta x(d1-d2)=2\Delta yx_k-2\Delta xy_k+c$

通过决策参数的符号，判定两候选像素与线段的偏移关系，从而确定第 k+1 步的像素。

算法实现：

```
elif algorithm == 'Bresenham':
    s = 1 if y0 < y1 else -1
    dx, dy = x1 - x0, y1 - y0
    xk, yk = x0, y0
    if dy < dx:
        p = 2 * dy - dx
        for x in range(dx + 1):
            result.append((xk, yk))
            if p >= 0:
                yk += s
                p -= 2 * dx
            xk += 1
            p += 2 * dy
    else:
        p = 2 * dx - dy
        for y in range(dy + 1):
            result.append((xk, yk))
            if p >= 0:
                xk += 1
                p -= 2 * dy
            yk += s
            p += 2 * dx
return result
```

2.1.1. 绘制多边形

算法原理：

调用画直线算法将每段直线加入结果中

算法实现：

```
result = []
for i in range(len(p_list)):
    line = draw_line([p_list[i - 1], p_list[i]], algorithm)
    result += line
return result
```

2.1.2. 绘制椭圆

2.1.2.1.中点圆生成算法

算法原理：

先将中心设置在原点。再分别于斜率绝对值小于 1 和绝对值大于 1 时，按决策参数判断选取的像素点，画出第一象限的图形。由于对称可得到其它三个象限的点坐标。最后将圆心平移至输入的坐标中心，即可得到要求的椭圆。

算法实现：

```
x0, y0 = p_list[0]
x1, y1 = p_list[1]
quarter = []
xc, yc = round((x0 + x1) / 2), round((y0+y1)/2)
rx, ry = round((abs(x1-x0))/2), round((abs(y1-y0))/2)
p = float(ry**2+rx**2*(0.25-ry))
xk, yk = 0, ry
quarter.append((xk, yk))
while(ry**2*xk<rx**2*yk):
    if(p<0):
        p+=float(ry**2*(2*xk+3))
    else:
        p+=float(ry**2*(2*xk+3)-rx**2*(2*yk-2))
        yk-=1
    xk+=1
    quarter.append((xk, yk))
p=float((ry*(xk+0.5))**2+(rx*(yk-1))**2-(rx*ry)**2)
while(yk>0):
    if(p<0):
        p+=float(ry**2*(2*xk+2)+rx**2*(-2*yk+3))
        xk+=1
    else:
        p+=float(rx**2*(-2*yk+3))
    yk-=1
    quarter.append((xk, yk))
result = []
```

```
for p in quarter:
    result.append(xc+p[0], yc+p[1])
    result.append(xc+p[0], yc-p[1])
    result.append(xc-p[0], yc+p[1])
    result.append(xc-p[0], yc-p[1])
return result
```

2.1.3. 绘制曲线

2.1.3.1.Bezier 算法

算法原理：

以参数方程表示直线，参数由 0 到 1 取若干值得到若干点。对于每个值，对每两个控

制点进行一次 de Casteljau 运算，得到新的一轮控制点，再进行相同操作，直到最后只有一个点，即为该参数值对应的所求点。

算法实现：

```
du = 0.001
result = []
if algorithm == 'Bezier':
    n = len(p_list) - 1
    result.append(p_list[0])
    u = du
    while u < 1:
        p = p_list.copy()
        for i in range(n):
            p_new = []
            for k in range(len(p) - 1):
                p_new.append([(1 - u) * p[k][0] + u * p[k+1][0], (1 - u) * p[k][1] + u * p[k+1][1]])
            p = p_new.copy()
        x, y = round(p[0][0]), round(p[0][1])
        result.append([x, y])
        u += du
    result.append(p_list[-1])
```

2.1.3.2.B-spline 算法

算法原理：

对于四阶均匀 B 样条曲线，可通过 deboox_cox 算法迭代直接推出参数的函数，将得到的函数作用在控制点上，即可得到所求点。

算法实现：

```
elif algorithm == 'B-spline':
    u = 0
    while u <= 1:
        coef = [-u ** 3 + 3 * u ** 2 - 3 * u + 1, 3 * u ** 3 - 6 * u ** 2 + 4,
                -3 * u ** 3 + 3 * u ** 2 + 3 * u + 1, u ** 3]
        for i in range(n - 3):
            x, y = 0.0, 0.0
            for j in range(4):
                x += coef[j] * p_list[i+j][0]
                y += coef[j] * p_list[i+j][1]
            result.append([round(x/6), round(y/6)])
        u += du
```

2.2. 图元变换

2.2.0. 图元平移

算法原理：

对每个点加上所需的平移量

算法实现：

```
return [[p[0] + dx, p[1] + dy] for p in p_list]
```

2.2.1. 图元旋转

算法原理：

首先得到弧度制。然后对每个点坐标乘上旋转的对应矩阵。

算法实现：

```
theta = r * math.pi / 180
for p in p_list:
    tx = round((p[0] - x) * math.cos(theta) - (p[1] - y) * math.sin(theta) + x)
    ty = round((p[0] - x) * math.sin(theta) + (p[1] - y) * math.cos(theta) + y)
    p[0], p[1] = tx, ty
return p_list
```

2.2.2. 图元缩放

算法原理：

对于每个点，将其到中心的距离乘以所缩放的倍率即可得到要求的点

算法实现：

```
return [[round((p[0] - x) * s + x), round((p[1] - y) * s + y)] for p in p_list]
```

2.2.3. 图元裁剪

2.2.3.1.Cohen-Sutherland 算法

算法原理：

延长裁剪窗口对整个平面分块，标记以不同二进制码，通过位运算确定直线所处位置。若在窗口内则保留，若在窗口外则去除。否则不停对直线进行裁剪，直到出现上述两种情况。

算法实现：

```
if code0 == 0:
    x0, x1 = x1, x0
    y0, y1 = y1, y0
    code0, code1 = code1, code0
if code0 & 1:
    y0 = round(y0 + ((x_min - x0) * (y0 - y1) / (x0 - x1)))
    x0 = x_min
if code0 & 2:
    y0 = round(y0 + ((x_max - x0) * (y0 - y1) / (x0 - x1)))
    x0 = x_max
if code0 & 4:
    x0 = round(x0 + ((y_min - y0) * (x0 - x1) / (y0 - y1)))
    y0 = y_min
if code0 & 8:
    x0 = round(x0 + ((y_max - y0) * (x0 - x1) / (y0 - y1)))
    y0 = y_max
```

2.2.3.2.Liang-Barsky 算法

算法原理：

以参数方程形式表示直线，求直线与裁剪窗口的上下左右边界交点，求出参数范围，即可得到直线的两个端点。

算法实现：

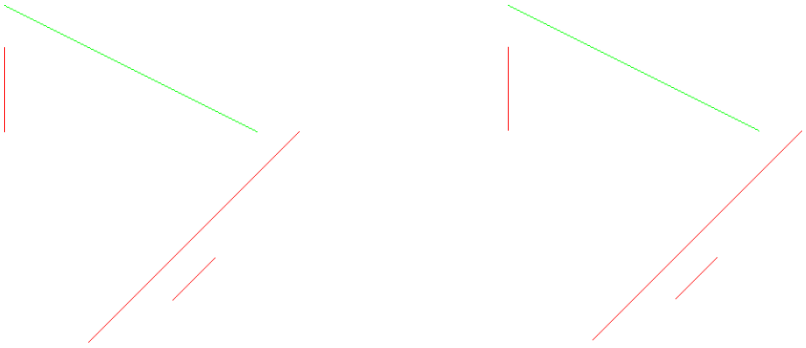
```
elif algorithm == 'Liang-Barsky':
    q = [x0 - x1, x1 - x0, y0 - y1, y1 - y0]
    d = [x0 - x_min, x_max - x0, y0 - y_min, y_max - y0]
    t0 = 0
    t1 = 1
    for i in range(4):
        if q[i] == 0:
            if d[i] < 0:
                return []
        elif q[i] < 0:
            t0 = max(t0, d[i] / q[i])
        else:
            t1 = min(t1, d[i] / q[i])
        if t0 > t1:
            return []
    return [[round(x0 + t0 * (x1 - x0)), round(y0 + t0 * (y1 - y0))],
            [round(x0 + t1 * (x1 - x0)), round(y0 + t1 * (y1 - y0))]]
```

3. 系统介绍

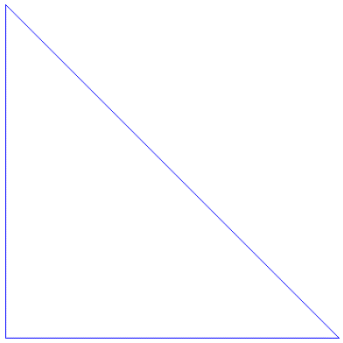
3.1. CLI 程序

通过按行读取指令实现。调用 cg_algorithms.py 中函数计算图形，对生成的像素点进行着色。将生成的图片输出为.bmp 文件。

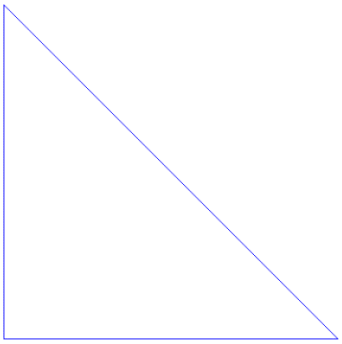
输出图形与样例图形见下图。



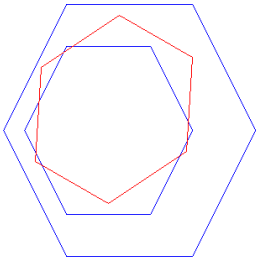
输出　　样例



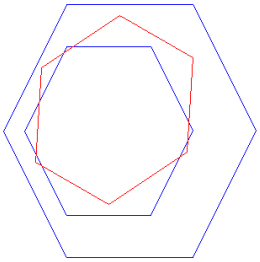
输出



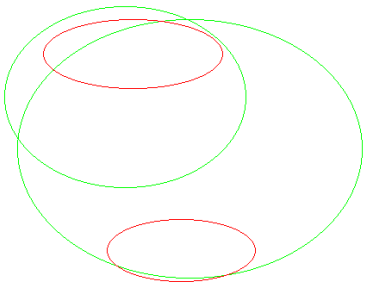
样例



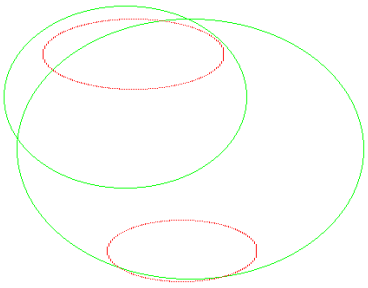
输出



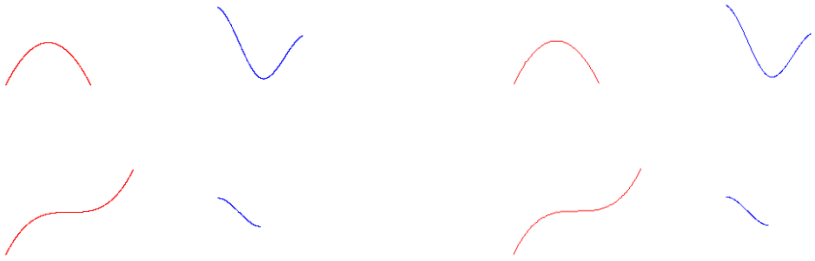
样例



输出



样例



输出　　样例

3.2.　GUI 程序

3.2.0. 总述

通过 MainWindow 启动程序并且连接槽函数。
通过 MyCanvas 响应鼠标事件。
通过 MyItem 画出图元。

3.2.1. 重置画布

MyCanvas:
　　clearItem: 清除所有目前的 Item 和状态
MainWindow:
　　reset_canvas_action: 调用 QInputDialog 输入新画布尺寸

3.2.2. 保存画布

MyCanvas:
　　save_canvas: 调用 QPixmap 生成保存图片
MainWindow:
　　save_canvas_action: 调用 QFileDialog 保存函数
　　__init__: 添加 width 和 height 属性

3.2.3. 设置画笔颜色

MyCanvas:
　　mousePressEvent: 初始化每个 MyItem 时传递 main_window.color 参数
MyItem:
　　__init__: 增加 color 参数

paint: 调用 setPen 设置颜色

MainWindow:

Set_pen_action: 调用 QColorDialog 得到颜色

3.2.4. 绘制线段

MainWindow:

line_dda_action: 仿照 line_naive_action 调用 dda 算法

bresenham_dda_action: 仿照 line_naive_action 调用 bresenham 算法

3.2.5. 绘制多边形

仿照绘制线段，增加 polygonCompleted 判断多边形绘制是否完成

3.2.6. 绘制椭圆

仿照绘制线段

3.2.7. 绘制曲线

仿照绘制多边形

3.2.8. 图元平移

按下鼠标时记录初始点，移动鼠标时记录位移，松开鼠标时完成平移

3.2.9. 图元旋转

输入中心点和旋转角度，完成旋转

3.2.10. 图元缩放

仿照图元旋转

3.2.11. 对线段裁剪

仿照图元平移，得到选取框，调用 clip 函数得到框中所剩的图元

4. 参考资料

孙正兴，计算机图形学 PPT，2017