《计算机图形学》3 月报告

学号：171860578，姓名：张梓悦，2846728884@qq.com

2020 年 3 月 30 日

1 综述

已于 3 月完成所有的核心算法模块以及 CLI 程序。预计 4 月份可以将 gui 部分完成，5 月份能够拓展 gui 界面，使得该界面更加美观

2 算法介绍

2.1 直线算法

2.1.1 DDA 算法

算法原理：

考虑 $y = kx + b, 0 < k < 1$ 时，x 每增加 1，y 增加 k，可令 $\Delta x = 1, \Delta y = k$。则可以从 x 的起始点 x_0 每次增加 Δx 直到 x_1，同时每次对 y 也增加 Δy，对于小数情况进行四舍五入即可，最终能够得到直线 $y = kx + b$。

算法实现：

```
if(abs(x1-x0)>=abs(y1-y0)):
    length=abs(x1-x0)
else:
    length=abs(y1-y0)
    if (length==0):
        result.append((x0,y0))
        return result
    dx=(float)(x1-x0)/length
    dy=(float)(y1-y0)/length
    i=1
    x=x0
    y=y0
    while(i<=length):
        result.append((int(x+0.5),int(y+0.5)))
        x=x+dx
        y=y+dy
        i+=1
```

算法效果：图 1

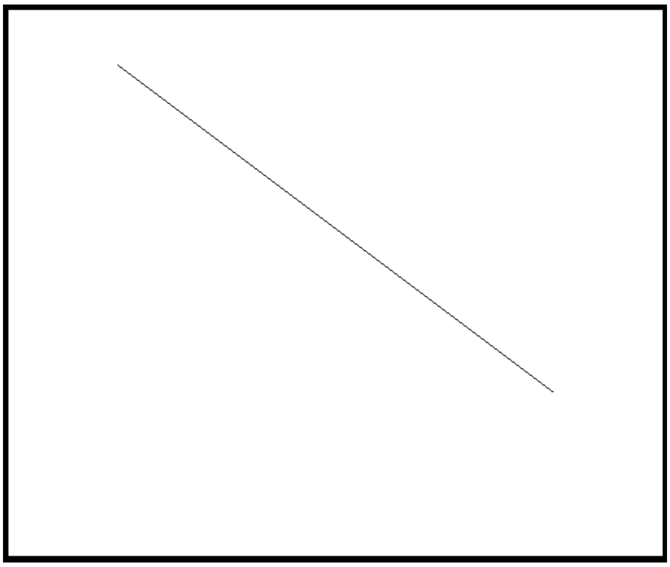

图 1: DDA 算法

2.1.2 Bresenham 算法

算法原理：

该算法避免了浮点运算，相当于 DDA 算法的一种改进算法。

当 x 方向是每一步的移动方向时（斜率小于 1），以每一小格的中点为界，如果当前的 yi 在中点下方，则 y 取 y_i; 如果当前的 yi 在中点上方，则 y 取 y_i+1。这样我们只需仅维护一个判别变量 p 即可。如果 p 大于等于 0 则下一个 y 的值 +1，同时 p 加上 2*dy-2*dx。如果 p 小于 0 则下一个 y 的值不变，同时 p 加上 2*dy。

算法实现：

```
dx=abs(x1-x0)
dy=abs(y1-y0)
if(dx==0 and dy==0):
    result.append((x0,y0))
    return result
gradient_flag=0
if(dx<dy):
    gradient_flag=1
if(gradient_flag==1):
    x0,y0=y0,x0
    x1,y1=y1,x1
    dx,dy=dy,dx
xx=1
if(x1-x0<0):
    xx=-1
yy=1
if(y1-y0<0):
    yy=-1
p=2*dy-dx
x=x0
y=y0
result.append((x,y))
while(x!=x1):
```

```
if(p>=0):
    p+=2*dy-2*dx
    y+=yy
else:
    p+=2*dy
x+=xx
if(gradient_flag):
    result.append((y,x))
else:
    result.append((x,y))
```

算法效果：图 2

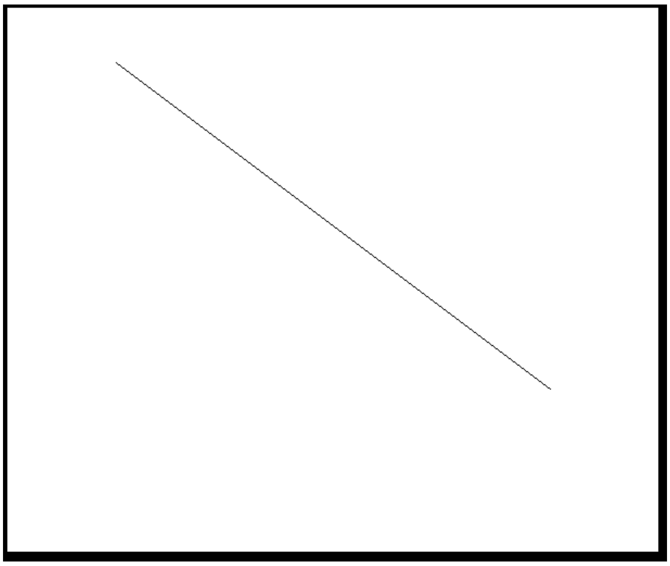

图 2: Bresenham 算法

2.2 椭圆算法（中点圆生成算法）

算法原理：

类似于 Bresenham 直线算法的思想，利用判别式选择像素，只需要做简单的整数运算。即根据判别式是否大于 0，来选择下一个 y 的值是不变还是减去 1，同时根据条件修改判别式的值。同时可以利用对称性，只需遍历椭圆的四分之一即右上方部分就能画出整个椭圆。

算法实现：

```
x0, y0 = p_list[0]
x1, y1 = p_list[1]
result = []
xx=int((x0+x1)/2)
yy=int((y0+y1)/2)
a=int((abs(x1-x0))/2)
b=int((abs(y1-y0))/2)
p=float(b**2+a**2*(0.25-b))
x=0
y=b
result.append((xx+x,yy+y))
```

```
    result.append((xx-x,yy+y))
    result.append((xx+x,yy-y))
    result.append((xx-x,yy-y))
    while(b**2*x<a**2*y):
        if(p<0):
            p+=float(b**2*(2*x+3))
        else:
            p+=float(b**2*(2*x+3)-a**2*(2*y-2))
            y-=1
        x+=1
        result.append((xx+x,yy+y))
        result.append((xx-x,yy+y))
        result.append((xx+x,yy-y))
        result.append((xx-x,yy-y))
    p=float((b*(x+0.5))**2+(a*(y-1))**2-(a*b)**2)
    while(y>0):
        if(p<0):
            p+=float(b**2*(2*x+2)+a**2*(-2*y+3))
            x+=1
        else:
            p+=float(a**2*(-2*y+3))
        y-=1
        result.append((xx+x,yy+y))
        result.append((xx-x,yy+y))
        result.append((xx+x,yy-y))
        result.append((xx-x,yy-y))
    return result
```

算法效果：图 3

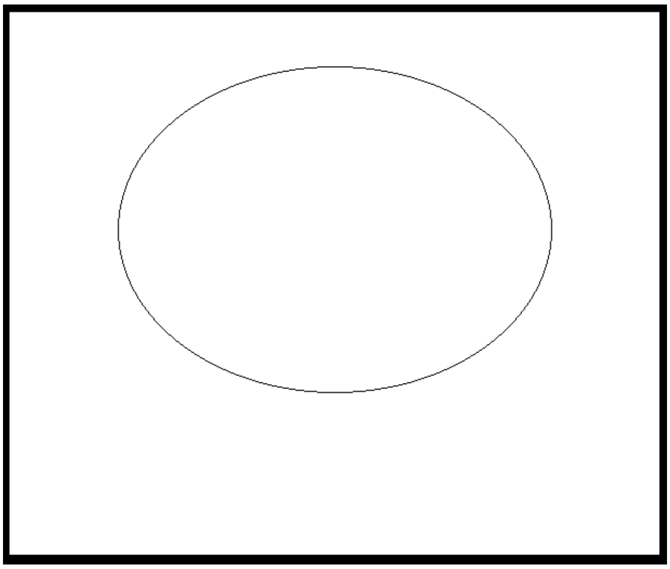

图 3: 中点圆生成算法

2.3 曲线算法

2.3.1 Bezier 算法

算法原理：
把 t 从 0 到 1 的移动过程中，在各个控制点连线的相应位置取点，并对相邻两条线上的点再次连线，重复以该过程使得没有可连接的两个点，即得到终的一个点。因此 t 从 0 到 1 的移动过程中计算出来的点的集合将构成目标曲线。
算法实现：

```
def Bezier_point(n, t, control_point):
while(n!=1):
    for i in range(0, n-1):
        x0,y0=control_point[i]
        x1,y1=control_point[i+1]
        x=float(x0*(1-t))+float(x1*t)
        y=float(y0*(1-t))+float(y1*t)
        control_point[i]=x,y
    n-=1
return control_point[0]

m=76800
for i in range(0, m):
    control_point=p_list
    t = float(i/m)
    x,y=Bezier_point(len(p_list), t, control_point)
    result.append((int(x+0.5),int(y+0.5)))
```

算法效果：图 4

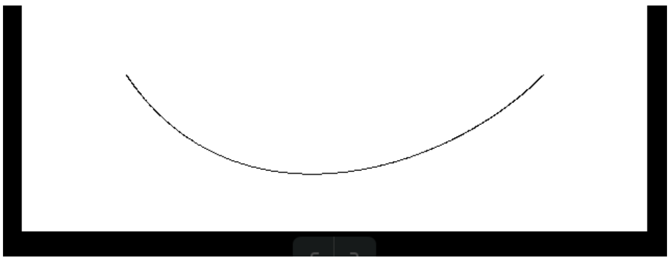

图 4: 中点圆生成算法

2.3.2 B-spline 算法

算法原理：
四个平面离散点可确定一条三次 B 样条曲线，可以先计算出三次 B 样条曲线的参数表达式，再由该参数表达式计算曲线上的每一个点即可。
算法实现：

```
n=len(p_list)
if(n<4):
    print('请至少输入4个点')
for i in range(0, n-3):
```

```
        x0,y0 = p_list[i]
        x1,y1 = p_list[i+1]
        x2,y2 = p_list[i+2]
        x3,y3 = p_list[i+3]
        p0 = float(-x0/6+x1/2-x2/2+x3/6)
        p1 = float(x0/2-x1+x2/2)
        p2 = float(-x0/2+x2/2)
        p3 = float(x0/6+2*x1/3+x2/6)
        q0 = float(-y0/6+y1/2-y2/2+y3/6)
        q1 = float(y0/2-y1+y2/2)
        q2 = float(-y0/2+y2/2)
        q3 = float(y0/6+2*y1/3+y2/6)
        m=50000
        for i in range(0, m):
            t = float(i/m)
            x = p0*t**3+p1*t**2+p2*t+p3
            y = q0*t**3+q1*t**2+q2*t+q3
            result.append((int(x+0.5),int(y+0.5)))
```

算法效果：图 5

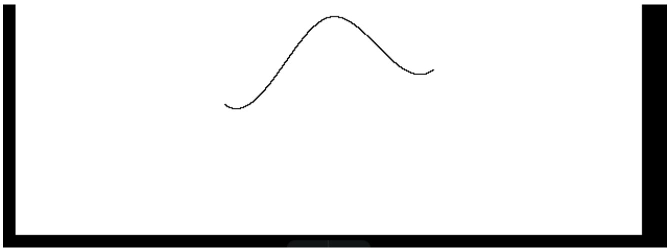

图 5: 中点圆生成算法

2.4 平移算法

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：

$$\begin{cases} x' = x + dx \\ y' = y + dy \end{cases} \tag{1}$$

算法实现：

```
    result = []
    for x, y in p_list:
        result.append((x+dx,y+dy))
    return result
```

2.5 旋转算法

算法原理：

不妨设 x_r,y_r 为旋转中心点坐标，θ 为顺时针旋转角度。

x,y 为原始坐标，x',y' 为旋转后坐标，则：

$$\begin{cases} x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{cases} \tag{2}$$

算法实现：

```
angel=float(r*math.pi/180)
cos=math.cos(angel)
sin=math.sin(angel)
result = []
for x0, y0 in p_list:
    x1=int(float(x)+float((x0-x)*cos)-float((y0-y)*sin)+0.5)
    y1=int(float(y)+float((x0-x)*sin)+float((y0-y)*cos)+0.5)
    result.append((x1,y1))
return result
```

2.6 缩放算法

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：

$$\begin{cases} x' = x \cdot s + x_f \cdot (1 - s) \\ y' = y \cdot s + y_f \cdot (1 - s) \end{cases} \tag{3}$$

算法实现：

```
result = []
for x0, y0 in p_list:
    x1=int(float(x0*s)+float(x*(1-s))+0.5)
    y1=int(float(y0*s)+float(y*(1-s))+0.5)
    result.append((x1,y1))
return result
```

2.7 线段裁剪算法

2.7.1 Cohen-Sutherland 算法

算法原理：
先将平面由裁剪框分成 9 个部分，对于任一端点 (x, y)，根据其坐标所在的区域，赋予一个 4 位的二进制码，判断图形元素是否落在裁剪窗口之内，如果有一部分在窗口之外再通过求交运算找出其位于内部的部分。

(1) 若 $x < x_{min}$，对应 code+1

(2) 若 $x > x_{max}$，对应 code+2

(3) 若 $y < y_{min}$，对应 code+4

(4) 若 $y > y_{max}$，对应 code+8

裁剪一条线段时，先求出端点 a 和 b 的编码 code1 和 code2，接下来进行如下算法：

(1) 如果 code1 和 code2 按位或的结果为 0，则说明 a 和 b 均在窗口内，即线段完全位于窗口之内，直接返回原来的端点值即可。

(2) 如果 code1 和 code2 按位与的结果不等于 0。说明 a 和 b 同时在窗口的上、下、左或右方，即线段完全位于窗口的外部，返回空或者一对相同的端点即可。

(3) 求出线段与窗口边界的交点，并用该交点的坐标值替换端点的坐标值。即在该交点处将线段分为两部分。

(4) 再次计算 code1 和 code2 的值，如果 code1 和 code2 按位或的结果不为 0，则裁剪失败，返回空或者一对相同的端点。

(5) 保存修改后的端点坐标

(6) 算法结束。

算法实现：

```
x0,y0 = p_list[0]
x1,y1 = p_list[1]
code1=0
code2=0
if(x0<x_min):
    code1+=1
if(x0>x_max):
    code1+=2
if(y0<y_min):
    code1+=4
if(y0>y_max):
    code1+=8
if(x1<x_min):
    code2+=1
if(x1>x_max):
    code2+=2
if(y1<y_min):
    code2+=4
if(y1>y_max):
    code2+=8

if((code1|code2)==0):
    result=p_list
elif((code1&code2)!=0):
    result=[[0,0],[0,0]]
else:
    code=code1|code2
    if(code&1):
        yy=int(float((x_min-x1)*(y0-y1)/(x0-x1))+float(y1)+0.5)
        if(x0<x_min):
            x0=x_min
            y0=yy
        elif(x1<x_min):
```

```
                x1=x_min
                y1=yy
        if(code&2):
            yy=int(float((x_max-x1)*(y0-y1)/(x0-x1))+float(y1)+0.5)
            if(x0>x_max):
                x0=x_max
                y0=yy
            elif(x1>x_max):
                x1=x_max
                y1=yy
        if(code&4):
            xx=int(float((y_min-y1)*(x0-x1)/(y0-y1))+float(x1)+0.5)
            if(y0<y_min):
                x0=xx
                y0=y_min
            elif(y1<y_min):
                x1=xx
                y1=y_min
        if(code&8):
            xx=int(float((y_max-y1)*(x0-x1)/(y0-y1))+float(x1)+0.5)
            if(y0>y_max):
                x0=xx
                y0=y_max
            elif(y1>y_max):
                x1=xx
                y1=y_max
        code1=0
        code2=0
        if(x0<x_min):
            code1+=1
        if(x0>x_max):
            code1+=2
        if(y0<y_min):
            code1+=4
        if(y0>y_max):
            code1+=8
        if(x1<x_min):
            code2+=1
        if(x1>x_max):
            code2+=2
        if(y1<y_min):
            code2+=4
        if(y1>y_max):
            code2+=8
        if((code1|code2)==0):
            result=[[x0,y0], [x1,y1]]
        else:
            result=[[0,0],[0,0]]
```

2.7.2 Liang-Barsky 算法

算法原理：

先计算 p 数组：p=$[x0-x1, x1-x0, y0-y1, y1-y0]$

再计算 q 数组：q=$[x0-x\_min, x\_max-x0, y0-y\_min, y\_max-y0]$

接下来可分情况讨论，考虑 p 和 q 数组中每一对 p、q：

(1) 若 $p = 0, q < 0$，表明直线与裁剪框平行，但是在裁剪框外面。因此需要直接舍弃该直线，即返回空或者一对相同的端点。

(2) 若 $p = 0, q \geq 0$，表明直线与裁剪框平行，且其延长线必然经过裁剪框的内部，接下来需要计算该线是在框的内部、外部还是与其相交

(3) 若 $p < 0$，表明直线从裁剪边界的外部延伸到内部

(4) 若 $p > 0$，表明直线从裁剪边界的内部延伸到外部

(5) 对于 (3) 和 (4) 的情况，找到直线与边界的交点，并计算出在该窗口内线段对应的参数 u_1 以及 u_2。u_1 由使直线是从外部延伸到窗口内部的边界决定。u_2 由使直线是从内部延伸到窗口外部的边界决定。先计算 $r = q/p$，则 $u_1 = max(r, 0)$，$u_2 = min(r, 1)$，如果 $u_1 > u_2$，则这条直线完全在窗口外面，需要舍弃。否则，可根据 u_1 以及 u_2 这两个值，计算出裁剪后的线段的端点，最后返回这两个新端点即可。

算法实现：

```
x0,y0 = p_list[0]
x1,y1 = p_list[1]
p=[x0-x1,x1-x0,y0-y1,y1-y0]
q=[x0-x_min,x_max-x0,y0-y_min,y_max-y0]
u1=float(0)
u2=float(1)
flag=False
for i in range(0,4):
    if(p[i]==0 and q[i]<0):
        flag=True
    else:
        if(p[i]==0):
            continue
        r=float(q[i]/p[i])
        if(p[i]<0):
            u1=max(float(0),r)
        else:
            u2=min(float(1),r)
        if(u1>u2):
            flag=True
if(flag==False):
    xx0=int(x0+u1*(x1-x0)+0.5)
    yy0=int(y0+u1*(y1-y0)+0.5)
    xx1=int(x0+u2*(x1-x0)+0.5)
    yy1=int(y0+u2*(y1-y0)+0.5)
    result=[[xx0,yy0],[xx1,yy1]]
```

3 系统介绍

3.1 采用的系统框架

本系统采用了助教所提供的 cg_cli.py 框架，支持所有合法的指令：
重置画布 resetCanvas width height
保存画布 saveCanvas name
设置画笔颜色 setColor R G B
绘制线段 drawLine id x0 y0 x1 y1 algorithm
绘制多边形 drawPolygon id x0 y0 x1 y1 x2 y2 ... algorithm
绘制椭圆 drawEllipse id x0 y0 x1 x1
绘制曲线 drawCurve id x0 y0 x1 y1 x2 y2 ... algorithm
图元平移 translate id dx dy
图元旋转 rotate id x y r
图元缩放 scale id x y s
对线段裁剪 clip id x0 y0 x1 y1 algorithm

3.2 设计思路

3.2.1 指令识别

每次读取一整行指令，由 split 函数分解该行指令，通过每一条指令的第一个单词，区别每一条指令的操作。同时解析剩下的单词，构造出函数调用时的参数，从而正确执行出每条指令对应的操作。

3.2.2 信息存储

对于直线、曲线、多边形、椭圆，将其关键点信息存到以 id 为键值得 hash 表中，每当需要修改图像时即可通过 hash 表快速访问图像信息从而进行修改操作。绘制图像时可遍历 hash 表中的每一个图像，分别调用相应的绘制函数进行绘制即可。

4 总结

通过这次图形学大作业，不仅让我们实现了一个功能全面的绘图工具，更大大加深了我们对课内所学的图形学算法理解，使我们受益匪浅。

5 引用

[1] 孙正兴, 计算机图形学课程 PPT, 2020