《计算机图形学》报告

181870290，周心瑜，181870290@smail.nju.edu.cn

2020 年 12 月 8 日

1 综述

本次实验完成了基础图形的绘制与基础图形变换。下文中算法介绍将说明 cg_algorithms.py 中代码的理论基础和部分伪代码结构；系统介绍部分将说明 cg_gui.py 中实现的图形界面逻辑。

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 值。

取样方向取决于直线斜率的绝对值大小。斜率的绝对值小于 1 时，在 X 方向取样，计算 Y 方向位置坐标；斜率的绝对值大于 1 时，在 Y 方向取样，计算 X 方向位置坐标。

增量 (Δx 或 Δy) 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.

Algorithm 1 DrawLineDDA

Input: [[x0, y0], [x1, y1]] 线段的起点和终点坐标
Output: [[x0, y0], [x1, y1], [x2, y2], ...] 绘制结果的像素点坐标列表

```
if 斜率不存在 then
    按 Δx = 1 计算 y，增量为 0;
else
    if (斜率绝对值 ≥ 1) then
        按 Δy = 1 计算 x，增量为斜率 m;
    else
        按 Δx = 1 计算 y，增量为斜率 1/m;
    end if
end if
```

2.1.2 Bresenham 算法

Bresenham 算法是一种采用整数增量运算，精确而有效的光栅设备线生成算法。

我们先只考虑斜率 $k \in [0,1]$ 的情况。从线段左端点开始处理，逐步处理每个后续列，每确定当前列的像素坐标 (x_i, y_i) 后，那么下一步需要在列 x_i+1 上确定 y 的值，此时 y 值不变或者增加 1，这是因为斜率 $k \in [0,1]$ 之间，x 增长更快，x 每增加 1，y 的增量小于 1。

对于左端点默认为其像素坐标，下一列选择的像素点为右方的像素或右上方的像素。设右上方像素到直线的距离为 d_2，右方像素到直线的距离为 d_1，只需要判断直线离哪个像素点更近，即判断 $d_1 - d_2$ 的符号即可找到最佳像素。

根据课本推导，得到下列公式：

$$p_k = \Delta x(d_1 - d_2)$$

$$p_{k+1} = \begin{cases} p_k + 2\Delta y & ,p_k < 0 \\ p_k + 2\Delta y - 2\Delta x & ,p_k \geq 0. \end{cases}$$

Algorithm 2 DrawLineBresenham

Input: [[x0, y0], [x1, y1]] 线段的起点和终点坐标

Output: [[x0, y0], [x1, y1], [x2, y2], ...] 绘制结果的像素点坐标列表

```
计算 dx, dy, p0
if 斜率不存在 then
    直接计算;
else
    if (斜率绝对值 ≥ 1) then
        计算增量方向符号，并求下一步 pk;
    else
        按 Y 方向计算，dx,dy 沿 y = x 对称后求 pk;
    end if
end if
```

2.2 绘制多边形

在 CG_demo.py 中已给出标准程序。将多边形中相邻两点作为 draw_line 的参数，从而在 result 中增加一条边。

2.3 绘制椭圆

程序实现通过定义决策参数，选择靠近圆周的像素来逼近椭圆方程中的曲线。下文将来介绍中点椭圆算法。

2.3.1 中点椭圆算法

首先, 我们来定义椭圆函数:

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

接着, 通过绘制椭圆在第一象限的曲线, 根据对称性, 可以得到四个象限曲线而绘制椭圆。其次, 根据第一象限中曲线切线斜率的不同, 将曲线图像分为两个部分。若斜率 $|m| < 1$, 则沿 x 方向离散取样, 若斜率 $|m| > 1$, 则沿 y 方向离散取样。

参考课本, 可知区域交替的条件为:

$$2r_y^2x \geq 2r_x^2y$$

最后, 在离散取样方向上, 取候选像素点中点来对椭圆函数求值。根据取样点相对于椭圆的位置, 决定选取哪一点。以区域 1(切线斜率 $|m| \geq 1$) 为例, $p_k^{(1)}$ 表示候选像素中点的椭圆函数。可得:

$$p_k^{(1)} = f_{ellipse}(x_{k+1}, y - \frac{1}{2}) = r_y^2(x_k+1)^2 + r_x^2(y_k - \frac{1}{2})^2 - rx^2r_y^2$$

根据中点椭圆函数取值, 可确定 (x_k, y_k) 的下一步选择

$$Coord_{(k+1)} = \begin{cases} (x_{k+1}, y_k) & p_k^{(1)} < 0; \\ (x_{k+1}, y_{k+1}) & p_k^{(1)} \geq 0. \end{cases}$$

Algorithm 3 DrawEllipse

Input: [[x0, y0], [x1, y1]] 椭圆的矩形包围框左上角和右下角顶点坐标

Output: [[x0, y0], [x1, y1], [x2, y2], ...] 绘制结果的像素点坐标列表

计算 $r_x, r_y, Coord_{(0)} = (0, r_y)$
计算 $p_0^{(1)}$ 初始值
while $(2r_y^2x \geq 2r_x^2y)$ **do**
  if $p_k^{(1)} < 0$ **then**
    $Coord_{(k+1)} = (x_{k+1}, y_k)$
    更新 $p_{k+1}^{(1)} = p_k^{(1)} + 2r_y^2x_{k+1} + r_y^2$
  else
    $Coord_{(k+1)} = (x_{k+1}, y_k - 1)$
    更新 $p_{k+1}^{(1)} = p_k^{(1)} + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_y^2$
  end if
end while
计算 $p_0^{(2)}$ 初始值
while $(y_k > 0)$ **do**
  if $p_k^{(2)} > 0$ **then**
    $Coord_{(k+1)} = (x_k, y_k - 1)$
    更新 $p_{k+1}^{(2)} = p_k^{(2)} - 2r_x^2y_{k+1} + r_x^2$
  else

$Coord_{(k+1)} = (x_{k+1}, y_k - 1)$

更新 $p_{k+1}^{(2)} = p_k^{(2)} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

end if

end while

根据 half_result 中的点确定其他三个象限对称点

同时，将 result 中的点 (x, y) 平移至 $(x + x_c, y + y_c)$ **return** result

---

2.4 绘制曲线

2.4.1 Brezier 算法

Brezier 曲线是通过一组多边折线的各个顶点唯一定义出来的，曲线的形状趋向于多边折线的形状。Bezier 曲线可以通过给定边界条件、特征矩阵或混合函数来决定，对一般 Bezier 曲线而言，最方便的是混合函数形式。

假设给出 $n+1$ 个控制定点位置 P_i。这些坐标点混合产生下列位置向量 $P(u)$：

$$P(u) = \sum_{i=0}^{n} P_i BEZ_{i,n}(u) \quad (0 \le u \le 1)$$

其中，n 次 Bernstein 基函数的多项式形式 $BEZ_{i,n}(u)$ 可以表达为：

$$BEZ_{i,n}(u) = C(n,i)u^i(1-u)^{n-i}$$

算法分析

第一步：选择精度（precision），在曲线段上选取 precision 个点作折线近似曲线

第二步：计算对于每一 u 的 $P(u)$（calculate_pu）

第三步：将 precision 个点使用 DDA 算法作折线

对比分析

Bezier 曲线有许多优点，如凸包性、保型性，但它也有很多不足之处，如特征多边形顶点的数量决定了 Bezier 曲线的阶次。同时，Bezier 曲线缺少局部性，修改某一控制点将影响整条曲线；控制多边形与曲线的逼近程度较差，次数越高，逼近程度越差。

2.4.2 B-spline 算法

B 样条基函数的定义：给定参数 u 轴上的节点分割 $U_{n,k} = u_i (i = 0, 1, 2, ..., n+k)$，称由下列 deBoox-Cox 递推关系所确定的 $B_{i,k}$ 为 $U_{n,k}$ 上的 k 阶（或 k-1 次）B 样条基函数。

deBoox-Cox 的递推公式为：

$$B_{i,k}(u) = [\frac{u - u_i}{u_{i+k-1} - u_i}]B_{i,k-1}(u) + [\frac{u_{i+k} - u}{u_{i+k} - u_{i+1}}]B_{i+1,k-1}(u) (i = 0, 1, ..., n)$$

当 $k = 1$ 时有：

$$\begin{cases} B_{i,1}(u) = 1 & u \in [u_i, u_{i+1}]; \\ B_{i,1}(u) = 0 & u \notin [u_i, u_{i+1}]. \end{cases}$$

称如下形式的参数曲线 $P(u)$ 为 k 阶（$k-1$ 次）B 样条曲线：

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

算法分析

第一步：选择精度（precision），在曲线每一个 $[u_i, u_{i+1}]$ 上选取 precision 个点作折线曲线

第二步：递归的计算对于每一 u 的 $P(u)$（calculate_p)

第三步：将 precision 个点使用 DDA 算法作折线

对比分析

B 样条曲线和 Bezier 曲线的区别表现在以下四个方面：

- 基函数的次数：Bezier 曲线次数为控制点数减去 1；B 样条曲线次数与控制点数无关。
- 基函数性质：Bezier 曲线基函数为多项式函数；B 样条曲线基函数是多项式样条。
- 曲线性质：Bezier 曲线是一种特殊标示形式的参数多项式曲线；B 样条曲线则是参数样条曲线。
- 局部控制能力：Bezier 曲线缺乏局部性质；B 样条曲线具有局部性质。

2.5 基本几何变换

2.5.1 平移

平移是指将物体沿直线路径从一个坐标位置到另一个坐标位置重定位。原始位置 $P(x, y)$ 按平移向量 (t_x, t_y) 到新位置 $P_1(x_1, y_1)$ 的移动：

$$x_1 = x + t_x$$

$$y_1 = y + t_y$$

2.5.2 旋转

旋转是指将物体沿 xy 平面内圆弧路径重定位。指定基准点 (x_r, y_r) 和旋转角 θ（顺时针为正方向）。那么，原始位置 $P(x, y)$ 按旋转角度 θ 到新位置 $P_1(x_1, y_1)$ 的移动：

$$x_1 = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y_1 = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

2.5.3 缩放

缩放变换改变图形的尺寸。原始位置 $P(x, y)$ 按缩放倍数 s 到新位置 $P_1(x_1, y_1)$ 的移动：

$$x_1 = xs + x_f(1 - s)$$

$$y_1 = ys + y_f(1 - s)$$

2.6 直线裁剪

2.6.1 Cohen-Sutherland 算法

Cohen-Sutherland 算法通过编码测试来计算交点。首先，线段端点将按区域赋以四位二进制码。将端点坐标值与裁剪窗口边界坐标比较：

$x < xw_{min}$ 则在位置 1 置为 1，否则置为 0；

$x > xw_{max}$ 则在位置 2 置为 1，否则置为 0；

$y < yw_{min}$ 则在位置 3 置为 1，否则置为 0；

$y > yw_{max}$ 则在位置 4 置为 1，否则置为 0.

根据区域码可以快速判断是否完全在窗口边界内（均为 0000）、是否完全在裁剪矩形外（逻辑与操作结果不为零）。再按照“左右上下”的顺序在边界求交，确定要裁剪掉的部分，直到线段完全被舍弃或者完全位于窗口内的线段。

Algorithm 4 ClipItemCohen-Sutherland

Input: [[x0, y0], [x1, y1]] 线段的起点和终点坐标

Output: [[x0, y0], [x1, y1]] 裁剪后线段的起点和终点坐标

计算 $(x0, y0)$, $(x1, y1)$ 的区域码

if (code0 == 0 and code1 == 0) **then return** $(x0, y0)$, $(x1, y1)$

end if

if (code0 & code1) != 0 **then return** $(0, 0)$, $(0, 0)$

end if

if 平行于任意一条边界 **then**

判断上下（左右）是否在裁剪区间外，若在外则纵（横）坐标定为边界 **return** $(x0, y0)$, $(x1, y1)$

end if

对 $(x0, y0)$, $(x1, y1)$ 在四个方向上求交，若有交点则将点更新 **return** $(x0, y0)$, $(x1, y1)$

2.6.2 Liang-Barsky 算法

Liang-Barsky 算法将二维裁剪问题简化为了一维裁剪问题。根据参数化形式的裁剪条件：

$$xw_{min} \leq x_1 + u\Delta x \leq xw_{max}$$

$$yw_{min} \leq y_1 + u\Delta x \leq yw_{max}$$

得到关于 u 的不等式 $up_k \leq q_k$. 当 $k = 1, 2, 3, 4$ 时，分别对应裁剪窗口的四个边界。当 $p_k = 0$ 时，线段平行与裁剪边界，进行特殊判断；当 $p_k < 0$ 时，更新 u 的下界 u_1；当 $p_k > 0$ 时，更新 u 的上界 u_2.

Algorithm 5 ClipItemLiang-Barsky

Input: [[x0, y0], [x1, y1]] 线段的起点和终点坐标

Output: [[x0, y0], [x1, y1]] 裁剪后线段的起点和终点坐标

u_0,u_1 赋初值 0,1

计算对四个边界于线段所在直线交点的参数 p_i,q_i

for i in range(4) **do**

if $p_k == 0$ **then**

if $q_k < 0$ **then return** $(0, 0)$, $(0, 0)$

elsereturn (x_0, y_0), (x_1, y_1)

end if

end if
$u = \frac{q_k}{p_k}$
if $p_k < 0$ **then**
更新 $u_0 = \max(u_0, u)$
else
更新 $u_1 = \min(u_1, u)$
end if
if $u_0 > u_1$ **then return** $(0,0)$, $(0,0)$
end if
end for
根据 u_0,u_1 计算 (x_0', y_0'), (x_1', y_1') **return** (x_0', y_0'), (x_1', y_1')

3 系统介绍

3.1 选取两点绘制图形

适用图形：线段、椭圆

在 mousePressEvent 时，将 temp_item 的 p_list 中的起点终点都设定为当前点；在 mouseMoveEvent 时，将 p_list 终点设置为鼠标所在位置；在 mouseReleaseEvent 时，在 item_dict 中加入当前 item。

3.2 * 选取两点选择图形

实现函数：def showItemSelected(self) in class MyCanvas.

计算 item_dict 中与图形关联的矩形 A_i 与选择矩形 B 的变量式：

$$proportion = \frac{S(A_i B)}{S(A_i \cup B)}$$

其中，计算重叠部分面积的推导方式: 令 $(x_0, y_0)(x_1, y_1)(a_0, b_0)(a_1, b_1)$ 为两矩形的对角线点，x_{max} 与 x_{min} 分别记作四个坐标点中横坐标最大值与横坐标最小值。则重叠部分 x 轴方向长度计算为 $l' = max\{0, l_0 + l_1 - (x_{max} - x_{min}),\}$，$l_0, l_1$ 分别为两矩形在 x 轴方向长度。同理，计算获得 y 轴方向长度从而得到重叠矩形面积（不相交时面积为 0）。

最后，选择所占比例最大的图形作为选定图形。

3.3 选取多点绘制图形

适用图形：多边形、曲线

引入 temp_index 记录控制点个数，并在右击时设为 0 表示结束输入，在松开鼠标时结束图形输入。

3.4 选取向量进行图形变换

适用变换：平移

将鼠标点击设为起点，鼠标移动后位置设为终点记录平移向量。每次通过调用 translate 函数将图形平移，并将起点设为上一次平移的终点，知道鼠标释放操作后结束对选定图形的变换。

3.5 选取多个向量进行图形变换

适用变换：旋转、缩放

通过鼠标点击活动获得三个点 $A(x_c, y_c), B(x_0, y_0), C(x_1, y_1)$。其中 A 为第一次鼠标点击，B 为第一次鼠标释放，C 为第二次鼠标释放。A 为变换活动的中心点，$\vec{v_0} = \vec{AB}$ 为参考向量，$\vec{v_1} = \vec{AC}$ 为目标向量。对于旋转变换，旋转角度为 $\vec{v_0}$ 与 $\vec{v_1}$ 的夹角；对于缩放变换，缩放倍数为 $|\vec{v_1}|$ 与 $|\vec{v_0}|$ 的倍数。

计算角度的函数: def calculate_degree(self, x0, y0, x1, y1) in class MyCanvas.

通过向量的点乘得到旋转弧度 α 的余弦值，并叉乘判断顺逆时针方向。注：当向量垂直时无法判断方向。在表示时将向量横坐标进行一定的近似（$x_0 = x_0 + 1$），使得向量不再垂直。

3.6 裁剪线段删除

在裁剪操作时，如线段在裁剪框之外，则返回 None. 在释放鼠标时，如当前参数的 p_list 为 None. 则在 scene 与 list_widget 中删除。

3.7 设置画笔

将颜色参数设为 item.p_list 的最后一个参数。在调用 paint 函数时，首先将 p_list 的最后一个参数弹出，作为画笔的颜色设置。

3.8 重置画布

实现函数：def clear_canvas(self) in class MyCanvas.

将类 MyCanvas 中的参数初始化。

4 总结

在本次大实验中，通过阅读课本、资料查询、理解 demo 代码结构，学会了如何分步解决任务，并完成项目，受益良多。感谢所有提供帮助的老师助教与同学!

5 参考

直线参考

https://www.cnblogs.com/LiveForGame/p/11706904.html

曲线参考

https://blog.csdn.net/qq_32583189/article/details/53018981

https://zhuanlan.zhihu.com/p/50626506

图形界面参考
https://doc.qt.io/qtforpython/modules.html