《计算机图形学》3 月报告

171860013，刘恩萌，171860013@smail.nju.edu.cn

2020 年 3 月 30 日

1 综述

- 完成内容：本月以熟悉框架为主，实现了 cg_algorithms.py 中的大部分算法，以及 cg_cli.py 中对应的操作来测试算法的实现
- 开发环境：Ubuntu 18.04 Python 3.7.0
- 内容为 ... 的部分表示尚未实现

2 算法介绍

2.1 直线绘制

按照实验要求实现了 DDA 与 Bresenham 直线绘制算法。

2.1.1 DDA 算法

- 原理
  - 使用 x 或 y 方向单位增量间隔，逐步计算沿线路径上各像素点的位置
  - 在增量绝对值更大的方向上按照单位增量对线段离散取样，利用增量的比例计算出取样点在另一个方向上的坐标（此处需要取整操作）
- 实现 [1]
  1. 对于输入起始点（`x0,y0`），（`x1,y1`），先计算出各自的增量 `dx`，`dy`
  2. 选取增量中绝对值较大的一者为 `step=max(abs(dx), abs(dy))`
  3. x, y 方向每次的增量分别为 `dx/step`，`dy/step`
  4. 两个方向上每次递增增量并取整 (`round()`)，计算出 `step+1` 个点组成直线
- 分析
  - 优点：符合数学直观，易于理解和记忆；消除了直线方程中的乘法
  - 缺点：每个点的像素位置都需要面临取整，可能会产生累积误差（长线段所计算的像素位置可能会偏离实际线段）；涉及大量浮点运算和取整操作，比较耗时

2.1.2 Bresenham 算法

- 原理
  - 利用直线点的连续性，从起点开始，（斜率 $0 < m < 1$ 的直线）每次选取的下一个点要么是当前点右侧相邻的点，要么是当前点右上角相邻的点
  - 分别计算两个候选点与直线方程计算得到的实际点的偏移 d_1（右侧点的偏移）和 d_2（右上角点的偏移），总是选择离实际点较近的那个点
  - 对于偏移量不进行真正的浮点计算，而是使用作差法并将计算公式变形并化简为仅包含整数运算的决策参数 $p_k = \Delta x(d_1 - d_2) = 2\Delta y x_k - 2\Delta x y_k + c(c$ 为常数)。根据第 k 步决策参数的符号，判定两候选像素与线段的偏移关系。
  - 从第 k 步到第 $k+1$ 步只需要计算决策参数的增量就可以更新决策参数

    $$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) = \begin{cases} p_k + 2\Delta y - 2\Delta x & p_k > 0\ (y_{k+1} = y_k + 1) \\ p_k + 2\Delta y & p_k < 0\ (y_{k+1} = y_k) \end{cases}$$

    决策参数初始值为 $p = 2\Delta y - \Delta x$。

- 实现 [2]
  1. 输入直线的起始点（`x0,y0`），（`x1,y1`），如果发现是水平或垂直线（`x0 == x1 or y0 == y1`）直接输出所有端点。
  2. 计算常量：`dx`，`dy`，由于实现基于 Python 而非硬件环境，且对效率的要求并不苛刻，每次计算 $2\Delta y$ 和 $2\Delta y - 2\Delta x$ 的时间可以忽略不计，故此处不作存储。
  3. 判断直线生成方向是否与坐标轴方向一致（`(m > 0 and x0 < x1) or (m < 0 and y0 < y1)`），不一致时交换起始点坐标
  4. 按照斜率绝对值和 1 的关系（`dx`，`dy` 的相对大小）分情况处理。按照原理部分所述的步骤计算每步的决策参数，决定当前加入集合的点。

- 实现时遇到的困难及解决
  - 按照 ppt 给出算法实现的时候，ppt 上只给出了斜率 $m > 1$ 时的做法，我也只按照斜率的绝对值与 1 的大小关系区分了实现。测试时才发现 $m < -1$ 时决策参数的公式是不一样的。正确的做法应该是 1)Δx 和 Δy 需要取绝对值，2)$p_k >$ 时应有 $y_{k+1} = y_k - 1$，因为候选点是当前点右侧或右下角的相邻点。

- 分析
  - 优点：全都是整数计算，硬件实现非常容易，也不会出现累计误差
  - （不算缺点的）缺点：计算有些琐碎，有 8 种不一样的直线情况需要分类讨论（有些情况可以合并），需要一定时间理解和消化

2.2 椭圆绘制

按照要求，实现的是 Bresenham 中点椭圆生成算法。

- 原理 [2]
  - 利用**平移**和**对称性**，只需要生成第一象限部分的曲线。而第一象限被斜率为-1 的切线的切点分为了两部分。前半部分 Δx 较大，以 x 轴为基准计算；后半部分 Δy 较大，反之。下讨论前半部分的绘制思路。
  - 与 Bresenham 直线绘制算法类似，根据椭圆线段的连续性，下一个像素必然是当前点 (x_k, y_k) 的右侧像素 (x_k+1, y_k) 或右下侧像素 (x_k+1, y_k-1)。考察这两个候选像素的中点 $(x_k+1, y_k-1/2)$ 与椭圆 $f(x,y)$ 的位置关系可以得知哪个点与实际曲线上的点离得更近。

$f(x_k+1, y_k-1/2) < 0$	中点位于椭圆内	右侧像素与实际点更近
$f(x_k+1, y_k-1/2) = 0$	中点位于椭圆上	右侧像素和右下侧像素与实际点距离相同
$f(x_k+1, y_k-1/2) > 0$	中点位于椭圆外	右下侧像素与实际点更近

  - 经过化简，可以将 $p1_k = f(x_k+1, y_k-1/2) = r_y^2(x_k+1)^2 + r_x^2(y_k-1/2) - r_x r) y^2$ 作为决策参数，根据其符号来决定下一个像素的选择。初值为 $p1_0 = r_y^2 - r_x^2 * r_y + r_x^2/4$。
  - 同 Bresenham 直线绘制算法，可以使用增量来更新决策参数，即

$$p1_{k+1} = \begin{cases} p1_k + 2r_y^2 x_{k+1} + r_y^2 & p1_k < 0 \\ p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2 & p1_k > 0 \end{cases}$$

  - 第一部分计算到 $2r_y^2 x \geq 2r_x^2 y$ 为止（斜率为-1 的切线与椭圆的交点）。计算椭圆第二部分的原理类似。
  - 第二部分更换 y 轴为递增单位。决策参数初值为 $p2_0 = r_y^2(x_1+1/2)^2 + r_x^2(y_1-1)^2 - r_x^2 r_y^2$。候选像素点为下方 (x_k, y_k-1) 和右下方 (x_k+1, y_k-1) 的像素。增量更新公式为

$$p2_{k+1} = \begin{cases} p2_k - 2r_x^2 y_{k+1} + r_x^2 & p2_k > 0 \\ p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2 & p2_k < 0 \end{cases}$$

    循环至 $(r_x, 0)$ 处。

- 实现
  - 输入椭圆的外接矩形的左上角和右下角坐标（`x0,y0`），（`x1,y1`），计算出椭圆的长轴、短轴和圆心位置。
  - 计算出几个常用数值 `rx2=rx*rx`，`ry2=ry*ry`
  - 按照上述原理，递增计算每一步的决策参数选择像素点，分为两部分绘制椭圆的第一象限中的曲线。

- 将第一象限中的部分对称到其他三个象限中，并整体平移到圆心指示的位置上。
- 实现中遇到的问题及解决
  - 起初按照 ppt 上给出的过程实现的，会出现“烈焰红唇”一样的形状，一步一步推导后才发现 ppt 中第二部分的初始值公式中少打了一个平方符号。深刻体会到了自己理解算法的重要性，不然就被坑了呀。
- 分析
  - 优缺点基本同 Bresenham 直线绘制算法

2.3 多边形绘制

- 直接使用了框架提供的代码：在输入点两两之间绘制直线相连

2.4 曲线绘制

2.4.1 Brezier 曲线

...

2.4.2 三次 B 样条曲线

...

2.5 平移变换

- 原理
  - 将待平移图形的每个坐标加上需要平移的量
- 实现
  - 实际实现中，只需平移每种图形的“定位点”（直线的端点、多边形的所有顶点、椭圆的外接矩形顶点、曲线的定位点等），返回平移后的点并调用绘制算法重新绘制这些曲线即可

2.6 旋转变换

- 原理
  - 先考虑旋转中心为坐标原点的情况：利用极坐标表示可以得到平移变换后点的计算公式

$$\begin{cases} x_1 = x\cos\theta - y\sin\theta \\ y_1 = x\sin\theta + y\cos\theta \end{cases}$$

    即变换矩阵为

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

– 当旋转中心非坐标原点时，可以暂时以旋转中心 (x_r, y_r) 为原点建立临时坐标系，再讲旋转后的图形坐标变换回到原坐标系中即可，即

$$\begin{cases} x_1 = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ y_1 = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{cases}$$

2.7 缩放变换

...

2.8 线段裁剪

2.8.1 Cohen-Sutherland 算法

...

2.8.2 Liang-Barsky 算法

...

3 系统介绍

3.1 CLI 部分

- 基本采用原框架代码的命令解释器架构：分类讨论各个输入命令，提取命令中的参数，调用相应的绘制算法，或创建保存的动作
- 新增对支持注释，输入文件中以 # 开头的行全部视为注释不予解释

3.2 GUI 部分

...

4 总结

实验尚未成功，同志任需努力

参考文献

[1] https://en.wikipedia.org/wiki/Digital_differential_analyzer_(graphics_algorithm), 2020. [Online; accessed 28-February-2020].

[2] David F. Rogers, 石教英, and 彭群生. 计算机图形学的算法基础. 机械工业出版社, 2002.