

《计算机图形学》11 月报告

学号:181860117, 姓名: 徐佳美

candyann77@163.com

2020 年 11 月 30 日

目录

1 综述	2
1.1 环境介绍	2
1.2 已完成功能	2
1.3 可扩展功能	2
2 算法介绍	2
2.1 draw line	2
2.1.1 DDA 算法	2
2.1.2 Bresenham 算法	2
2.2 draw polygon	4
2.3 draw ellipse	5
2.3.1 中点画圆法	5
2.3.2 中点画椭圆	6
2.4 draw curve	8
2.4.1 Bezier 算法	8
2.4.2 B-spline 曲线	9
2.5 translate	10
2.6 rotate	10
2.7 scale	10
2.8 clip	10
2.8.1 Cohen-Sutherland	10
2.8.2 Liang-Barsky	11
3 系统介绍	12
4 总结	12

摘要

该部分内容是放置摘要信息的。该部分内容是放置摘要信息的。该部分内容是放置摘要信息的。该部分内容是放置摘要信息的。该部分内容是放置摘要信息的。

1 综述

1.1 环境介绍

Ubuntu-18.04 64bit

Python-3.7.9

Numpy-1.19.2

Pillow-7.0.0

Pyqt-5.9.2

1.2 已完成功能

绘制直线，多边形，椭圆，曲线；剪裁

1.3 可扩展功能

...

2 算法介绍

2.1 draw line

对于水平线，垂直线，对角线，可以直接得出结果而无需进行画线算法处理；

2.1.1 DDA 算法

采用增量思想，若设置步长为 1，对于点 x_0, y_0, x_1, y_1 ，则有：

$$y_1 = kx_1 + b = k(x_0 + 1) + b = kx_0 + k + b = y_0 + k$$

这里的 k 即为图形的斜率，即 x 每增加 1， y 增加 k 。

但是当 k 比较大时，这样的画法会导致画布上的点比较稀疏。因此，当 k 的绝对值 >1 时，考虑将 x, y 反置，则有：

$$x_1 = ty_1 + b = t(y_0 + 1) + b = ty_0 + b + t = x_0 + t$$

2.1.2 Bresenham 算法

每移动一个步长，在两个像素点中选择距离更近的一个；对 midpoint 画线法进行改进，考虑到光栅上是取整型点， $y_{k+1} = y_k + 1$ ；因此可以不用计算两个像素点距离线段上点的具体

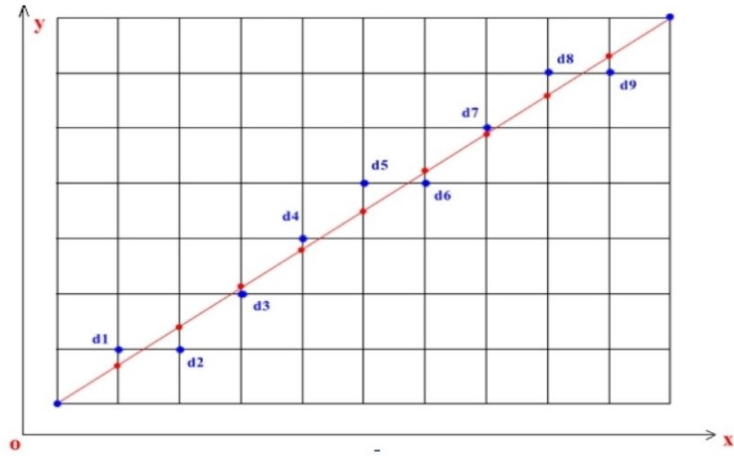


图 1: Bresenham 画直线图示

值，只要判断线段上的点是在格子中线的上方还是下方；这个算式包含很多小数计算，对决策参数进行变形，得到整数计算的 Bresenham 算法。

(1) 斜率在 0-1 之间的:

P_k 表示决策参数, P_k 每次加 k , 初始值为-0.5, 因此只要与 0 比较, 如果 > 0 , 移动, $y+1, P_k-1$;

$$dx = x_1 - x_0 \quad dy = y_1 - y_0 \quad k = dy/dx$$

$$P_k = -0.5 \quad y = y_0$$

$$P_k + = k$$

$$\text{if } P_k > 0 :$$

$$y + = 1 \quad P_k - = 1$$

在递推的等式中，两边同时乘以 $2 \cdot dx$; dx 为正数，不影响符号的判断

$$P_k = -0.5 \times 2dx = -dx$$

$$2 * dx * P_k = 2 * dx * P_k + dy/dx \times 2dx = 2 * dx * P_k + 2dy$$

$$P_k = P_k + 2dy$$

如果有移动，原来的 $P_k - = 1$ 变成 $P_k - = 2dx$;

(2) 斜率在-1 到 0 之间的按 x 递减方向看就和 0 到 1 的斜率一样;

合并 (1) (2)

$$\begin{aligned}
 & (y1 > y0) \\
 & dx = x1 - x0 \quad dy = y1 - y0 \quad k = dy/dx \\
 & \text{if } |k| < 1 : \\
 & \quad detay = dy << 1 \quad detax = dx << 1 \\
 & \quad Pk = -dx \quad y = y0 \\
 & \quad Pk+ = detay \\
 & \quad \text{if } Pk > 0 : \\
 & \quad \quad y+ = 1 \quad Pk- = detax
 \end{aligned}$$

下面将算法拓展到全象限，根据二维平面区域间的对称性 当斜率绝对值 >1 时，将 x

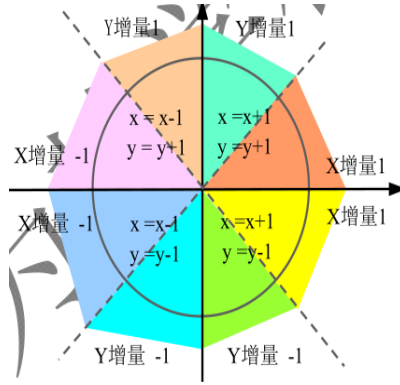


图 2: Bresenham 算法的通用性处理

和 y 的关系对调，新的算式如下

$$\begin{aligned}
 & (x1 > x0) \\
 & dx = x1 - x0 \quad dy = y1 - y0 \quad k = dy/dx \\
 & \text{if } |k| > 1 : \\
 & \quad detay = dy << 1 \quad detax = dx << 1 \\
 & \quad Pk = -dy \quad x = x0 \\
 & \quad Pk+ = detax \\
 & \quad \text{if } Pk > 0 : \\
 & \quad \quad x+ = 1 \quad Pk- = detay
 \end{aligned}$$

2.2 draw polygon

多边形是由多条直线围成的，对多边形的每一条边，调用绘制线段的算法即可

2.3 draw ellipse

2.3.1 中点画圆法

在画椭圆之前，先看椭圆的特例，圆的生成

首先圆具有很好的对称性，因此只要计算 $x=0$ 到 $x=y$ 之间的点。

用直线路径逼近圆路径，为了得到更连续的边界，将步长设置为 $\frac{1}{r}$ ，这样每个像素点大概为 1 个单位间隔；

对这个八分圆上的点进行离散化，沿 x 方向取单位步长，确定对应的 y 位置；每一步，我们首先得到候选的 y 位置；

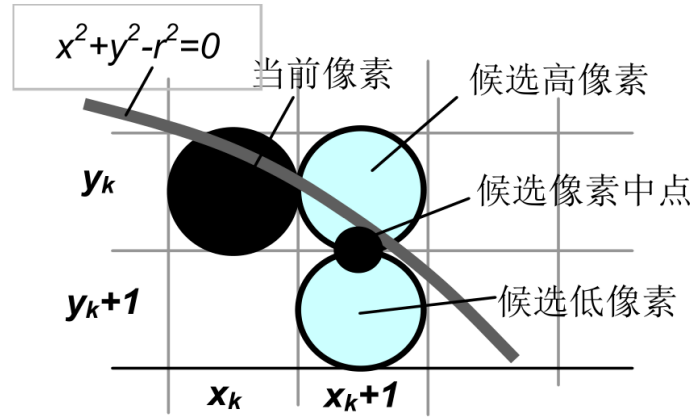


图 3: 中点圆算法的第 $k+1$ 步候选像素

为了应用中点法，定义圆函数为：

$$f_{circle}(x, y) = x^2 + y^2 - r^2;$$

则任一点 (x, y) 的相对位置可以由圆函数的符号决定：

$$\begin{cases} f_{circle}(x, y) < 0 & (x, y) \text{ 位于圆内;} \\ f_{circle}(x, y) = 0 & (x, y) \text{ 位于圆上;} \\ f_{circle}(x, y) > 0 & (x, y) \text{ 位于圆外;} \end{cases}$$

显然，在第一象限， y 是递减的，对于 (x_k, y_k) ，我们有两个候选点，即 (x_{k+1}, y_k) 和 $(x_{k+1}, y_k - 1)$ ，决策参数是圆函数在这两个候选点中间的值

$$p_k = f_{circle}(x_{k+1}, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \quad (1)$$

则不难得到，当 $p_k < 0$ 时，取高像素， $p_k \geq 0$ 时，取低像素。

则到第 $k+1$ 步时，如果是取高像素，中点为 $(x_{k+2}, y_k - \frac{1}{2})$ 如果是取低像素，则中点为 $(x_{k+2}, y_k - \frac{3}{2})$

化简决策参数为：

$$\begin{cases} p_{k+1} = P_k + 2x_k + 3 & p_k < 0 \\ p_{k+1} = P_k + 2x_k - 2y_k + 5 & p_k \geq 0 \end{cases} \quad (2)$$

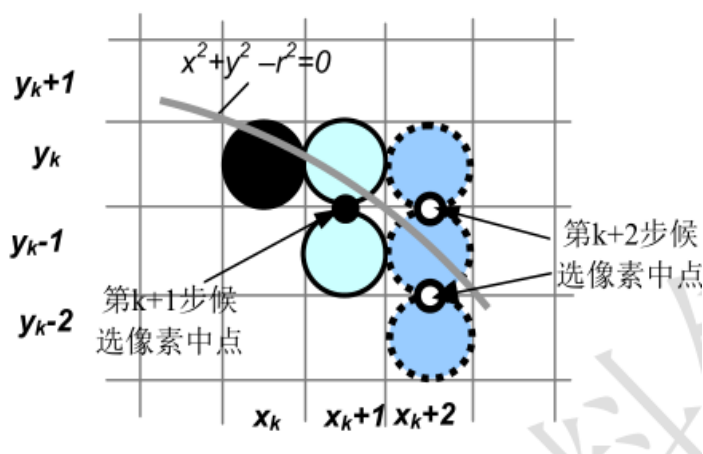


图 4: 中点圆算法的第 $k+2$ 步候选像素

决策参数初始值 $p_0 = \frac{5}{4} - r$

上述 p_{k+2} 决策参数和书本上不一样, p_0 又是怎么来的

对于点 (x, y) , 它在八分圆中的其他七个对称点为: (y, x) , $(-y, x)$, $(-x, y)$, $(-x, 2r - y)$, $(y - 2r, r - x)$, $(2r - y, r - x)$, $(x, 2r - y)$

整合画圆的部分为:

1. 得到圆心和半径 $(x_c, y_c), r$
2. 将圆心定为原点 $(0, r)$
 $dx = x_c, dy = y_c - r$
3. 确定决策参数初始值
4. 按上述公式完成增量计算
5. 确定其他八分圆中的对称点
6. 将点移动到圆心为 (x_c, y_c) 的圆路径上
7. 重复 4-6, 直到 $x \geq y$

2.3.2 中点画椭圆

接下来在画圆的基本思想上得到画椭圆的方法椭圆是四分对称, 将椭圆中心作为坐标中心, 在第一象限的一个四分区域, 根据中点椭圆法又可以分成两个区域:

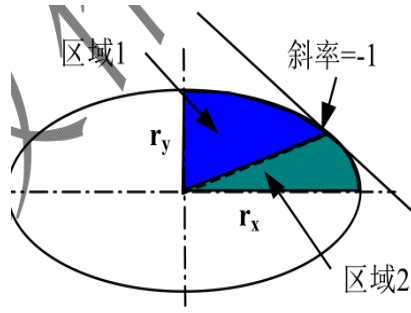
根据椭圆的定义化简点到椭圆中心的距离的平方, 则有:

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

则有:

$$\begin{cases} f_{ellipse}(x, y) < 0 & (x, y) \text{ 位于椭圆内;} \\ f_{ellipse}(x, y) = 0 & (x, y) \text{ 位于椭圆上;} \\ f_{ellipse}(x, y) > 0 & (x, y) \text{ 位于椭圆外;} \end{cases}$$

假设 $r_x > r_y$ (1) 在区域 1 以 x 为单位步长, 沿椭圆路径顺时针步进, 初始点为 $(0, r_y)$
 假设前一步的位置为 (x_k, y_k) , 则下一步两个候选像素点为 (x_{k+1}, y_k) , $(x_{k+1}, y_k - 1)$



第一象限椭圆两个区域划分：

区域1中，椭圆切线斜率绝对值小于1；

区域2中，椭圆切线斜率绝对值大于1。

图 5: 第一象限椭圆区域划分

对两个像素点中间计算决策参数为：

$$p1_k = f_{eclipse}(x_{k+1}, y_k - \frac{1}{2}) = r_y^2(x_k + 1)^2 + r_x^2(y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$

如果 $p1_k < 0$ ，中点位于椭圆内，则 y_k 更接近于椭圆边界；否则，取点 $y_k - 1$

$$\begin{cases} p1_{k+1} = p1_k + 2r_y^2 x_k + 3r_y^2 & \text{if } p_k < 0; \\ p1_{k+1} = p1_k + 2r_y^2 x_k - 2r_x^2 y_k + 3r_y^2 & \text{if } p_k \geq 0; \end{cases} \quad (3)$$

(2) 在区域 2 中，在 -y 方向以单位步长取样

则两个候选像素点为 $(x_k, y_k - 1), (x_k + 1, y_k - 1)$

对两个候选点的中间计算决策参数为：

$$p2_k = f_{eclipse}(x_k + \frac{1}{2}, y_k - 1) = r_y^2(x_k + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

如果 $p2_k < 0$ ，中点位于椭圆内，则 $x_k + 1$ 更接近于椭圆边界；否则，取点 x_k

$$\begin{cases} p1_{k+1} = p2_k - 2r_x^2 y_k + 3r_x^2 & \text{if } p_k \leq 0; \\ p2_{k+1} = p2_k + 2r_y^2 x_k - 2r_x^2 y_k + 2r_y^2 + 3r_x^2 & \text{if } p_k > 0; \end{cases} \quad (4)$$

$$p_0 = r_y^2 - r_x^2 r_y + \frac{r_x^2}{4} \quad (5)$$

(3) 假设求到的点为 (x, y) 根据对称关系，在其他三个四分椭圆的坐标为 $(-x, y), (-x, -y), (x, -y)$ 然后转回到以 (x_c, y_c) 为中心的椭圆

$$\begin{cases} x = x + x_c \\ y = y + y_c \end{cases} \quad (6)$$

综合 (1)(2)(3)，对于中心为 (x_c, y_c) ，长短轴分别为 $r_x, r_y (r_y < r_x)$ 的椭圆，生成过程如下：

① 输入中心 (x_c, y_c) r_x, r_y

② $(x_c, y_c) \Rightarrow (0, 0)$ ，第一个点为 $(0, r_y)$

③ 按公式计算 $p1_0$

④ 在区域 1 的每个 x_k 位置，从 $k=0$ 开始，按公式计算决策参数，循环至 $2r_y^2 x \geq 2r_x^2 y$;

⑤ 使用区域 1 的最后一个点作为区域 2 的起始点 (x_0, y_0) 计算区域 2 的参数初始值

$$p2_0 = r_y^2(x_0 + \frac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2$$

这个初始值咋来的

⑥ 在区域 2 的每个 y_k 位置，从 $k=0$ 开始，按公式计算决策参数，循环至 $(r_x, 0)$;

⑦ 确定其他对称点，平移到中心为 (x_c, y_c) 的椭圆轨迹上

2.4 draw curve

2.4.1 Bezier 算法

贝塞尔曲线是不规则曲线，由起始点，终止点，控制点组成，通过调整控制点，可以改变曲线的形状。

在实现时，需要在起点和终点之间构建插值多项式的混合函数，通常由 $n+1$ 个顶点定义一个 n 次多项式;

一阶贝塞尔曲线，也就是线段:

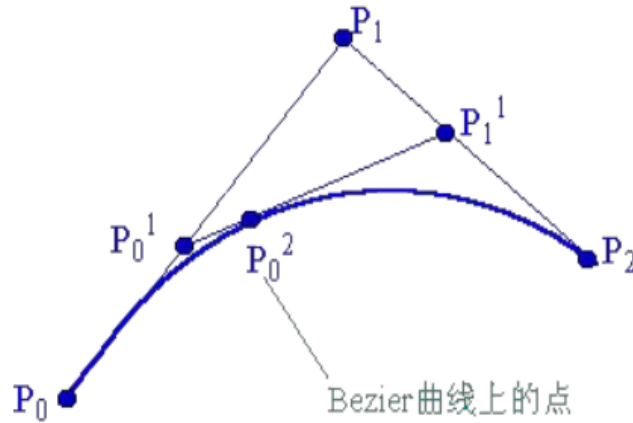
$$B(t) = P_0 + (P_1 - P_0)t = (1-t)P_0 + tP_1$$

二阶贝塞尔曲线（抛物线）:

设 $P_0 P_{02} P_2$ 是一条抛物线上顺序三个不同的点。过 P_0 和 P_2 点的两切线交于 P_1 点，在点 P_{02} 的切线交 P_0P_1 和 P_2P_1 于 P_{01} 和 P_{11} ，则如下比例成立

$$\frac{P_0P_{01}}{P_{01}P_1} = \frac{P_1P_{11}}{P_{11}P_2} = \frac{P_{01}P_{02}}{P_{02}P_{11}}$$

上述式子也称为抛物线的三切线定理



参考一阶的贝塞尔曲线，当 P_0, P_2 固定，引入参数 t ，令上述比值为 $t:(1-t)$ ，即有：

$$\begin{cases} P_{01} = (1-t)P_0 + tP_1 & (1) \\ P_{11} = (1-t)P_1 + tP_2 & (2) \\ P_{02} = (1-t)P_{01} + tP_{11} & (3) \end{cases}$$

将 (1) (2) 式带入 (3) 式， $B(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2$

不同阶的贝塞尔曲线生成动图如下：（注：部分 pdf 阅读器不支持动图展示）

(a) 二阶 bezier 生成

(b) 三阶 bezier 生成

(c) 四阶 bezier 生成

(d) 五阶 bezier 生成

图 6: bezire 曲线动态生成示意图

递推到高阶: 给定这 $n + 1$ 个点的位置, 则贝塞尔参数曲线上各点的插值公式为:

$$B(t) = \sum_{i=0}^n P_i b_{i,n}(t)$$

其中, $b_{i,n}(t) = C_n^i t^i (1-t)^{n-i}, t \in [0, 1]$

2.4.2 B-spline 曲线

啊啊啊贝塞尔曲线拼接复杂, 修改需要整体修改, 而 B 样条曲线是对贝塞尔曲线的补充; 给定 $n+1$ 个控制点, P_0, P_1, \dots, P_n 以及一个节点向量 $U = u_0, u_1, \dots, u_m$, p 次 B-样条曲线由这些控制点和节点向量 U 定义, 其公式为:

$$C(u) = \sum_{i=0}^n P_i N_{i,p}(u)$$

基函数定义如下:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (8)$$

上述公式通常称为 Cox-de Boor 递归公式

三次均匀 B 样条曲线, 取 $p=3$, 一共 $n+1$ 个控制点, 所以节点个数为 $n+1+p+1$, 节点沿参数轴均匀等距分布, 设节点范围为 $[0, 1, 2, 3, \dots, n+1+p]$ 从公式 (7) 可以看出, 需要计算 $N_{i,3}(u) (i = 0, 1, 2, \dots, n)$

2.5 translate

...

2.6 rotate

...

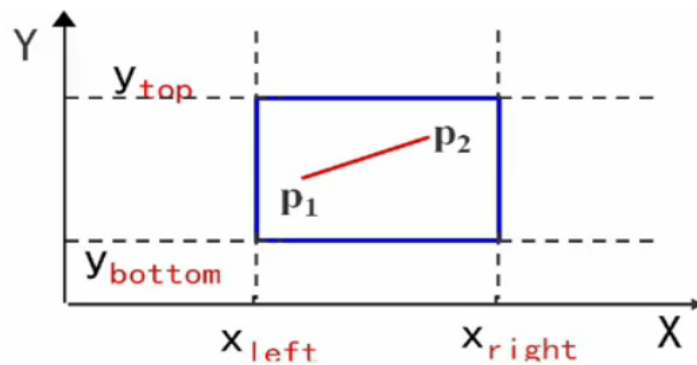
2.7 scale

...

2.8 clip

2.8.1 Cohen-Sutherland

Cohen-Sutherland 算法也称编码裁剪算法，对裁剪的窗口和直线段进行编码，通过二进制的位运算确定线段和窗口的位置关系对于每条直线段，分三种情况处理：（1）端点都在窗口内，保留



(2) 端点都在窗口外，且满足以下四个条件之一的，舍弃

$$\begin{cases} x_1 < x_{left}, x_2 < x_{left}; \\ x_1 > x_{right}, x_2 > x_{right}; \\ y_1 < y_{bottom}, y_2 < y_{bottom}; \\ y_1 > y_{top}, y_2 > y_{top}; \end{cases}$$

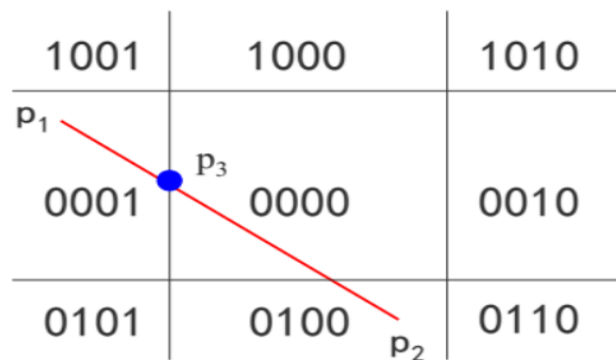
(3) 对于不满足上述两种情况的直线段：

step1: 对线段的端点进行编码 $D_3D_2D_1D_0$ ，编码规则如下：

$$\begin{cases} x < x_{left}, D_0 = 1, & D_0 = 0; \\ x > x_{right}, D_1 = 1, & D_1 = 0; \\ y < y_{bottom}, D_2 = 1, & D_2 = 0; \\ y > y_{top}, D_3 = 1, & D_3 = 0; \end{cases}$$

step2: 假设两个端点编码分别为 code1,code2; 分以下三种情况:

1. code1|code2 = 0, 保留
2. code1 & code2 ≠ 0, 舍弃
3. 两种情况之外, 求出与窗口边界的交点, 将线段分成两部分, 分别调用本算法



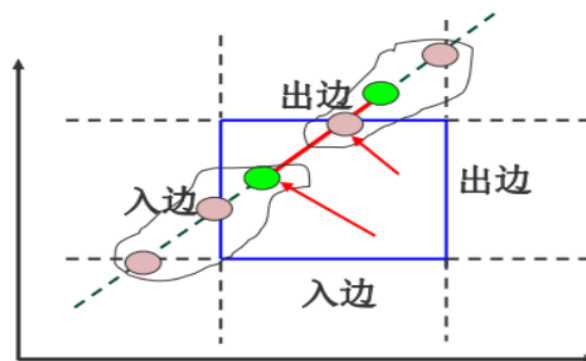
2.8.2 Liang-Barsky

梁友栋算法的基本出发点是直线的参数方程

(1) 用参数方程表示一条直线

$$\begin{cases} x = x_1 + u(x_2 - x_1) = x_1 + u\Delta x & 0 \leq u \leq 1 \\ y = y_1 + u(y_2 - y_1) = y_1 + u\Delta y & 0 \leq u \leq 1 \end{cases}$$

(2) 把被裁减的直线段看成是一条有方向的线段, 窗口的四条边分成两类, 入边和出边 裁剪



线段的起点是直线和两条入边的交点以及始端点三个点里最前面的一个点, 即参数 u 最大

的那个点；

裁剪线段的终点是直线和两条出边的交点以及端点最后面的一个点，即参数 u 最小的那个点；

用 u_1, u_2 分别表示线段 ($u_1 \leq u_2$) 可见部分的开始和结束，则有：

$$\begin{cases} u_1 = \max(0, u_l, u_b) \\ u_2 = \min(1, u_t, u_r) \end{cases}$$

(3) 若 $u_{max} > u_{min}$ ，则直线段在窗口外，删除该直线；

否则，将 u_{max}, u_{min} 代入参数方程，求得直线与窗口的两实交点坐标；

下面是 u_1, u_2 的计算方法

由参数方程可知，对于直线内的点：

$$\begin{cases} x_{left} \leq x_1 + u\Delta x \leq x_{right} \\ y_{bottom} \leq y_1 + u\Delta y \leq y_{top} \end{cases}$$

令 $p_1 = -\Delta x, q_1 = x_1 - x_{left}; \quad p_2 = \Delta x, q_2 = x_{right} - x_1;$

$p_3 = -\Delta y, q_3 = y_1 - y_{bottom}; \quad p_4 = \Delta y, q_4 = y_{top} - y_1;$

$u_k = \frac{q_k}{p_k} (p_k \neq 0, k = 1, 2, 3, 4)$

$p_k < 0$, 代表入边, $p_k > 0$ 代表出边

$$\begin{cases} u_{max} = \max(0, u_l | p_k < 0, u_b | p_k < 0) \\ u_{min} = \min(1, u_t | p_k > 0, u_r | p_k > 0) \end{cases}$$

特殊情况：

1. 若 $\Delta x = 0$, 判断是否有 $q_1 < 0$ 或 $q_2 < 0$, 若有，线段不在窗口内，丢弃；否则，继续计算 u_{max}, u_{min} , k 取 3, 4;
2. 若 $\Delta y = 0$, 判断是否有 $q_3 < 0$ 或 $q_4 < 0$, 若有，线段不在窗口内，丢弃；否则，继续计算 u_{max}, u_{min} , k 取 1, 2;

3 系统介绍

...

4 总结

...

参考文献

- [1] Bresenham 画直线算法 _ 陈嘉怡的博客: https://blog.csdn.net/chenjiayi_yun/article/details/38601439

- [2] latex 公式编辑器: <https://www.latexlive.com>
- [3] 中点椭圆算法 <https://www.cnblogs.com/clairvoyant/p/5540023.html>
- [4] 贝塞尔曲线总结 <https://blog.csdn.net/tianhai110/article/details/2203572>
- [5] 贝塞尔曲线原理 <https://www.jianshu.com/p/8f82db9556d2>
- [6] B 样条算法 _ 矢月: https://blog.csdn.net/weixin_44397852/article/details/108836781
- [7] <https://www.cnblogs.com/icmzn/p/5100761.html>
- [8] https://blog.csdn.net/qq_40597317/article/details/81155571
- [9] 裁剪算法 <https://www.cnblogs.com/Penglime/p/9690372.html>
- [10] 裁剪算法 <https://www.cnblogs.com/Penglime/p/9690407.html>