图形学大作业系统报告

陈劭源 (161240004)
sy_chen@smail.nju.edu.cn

May 25, 2019

1 综述

目前已完成所有算法的实现和命令行界面，图形界面尚未完成。

2 图元绘制算法介绍

绘制曲线 $f(x,y)=0$ 的基本原则是：当 $\left|\frac{\mathrm{d}y}{\mathrm{d}x}\big|_{(x_0,y_0)}\right| \leq 1$ 时，沿 x 轴递进采样画点；当 $\left|\frac{\mathrm{d}y}{\mathrm{d}x}\big|_{(x_0,y_0)}\right| > 1$ 时，沿 y 轴递进采样画点。这样可以保证相邻两个绘制点 $(x_i,y_i),(x_{i+1},y_{i+1})$ 之间满足 $\max\{|x_i-x_{i+1}|,|y_i-y_{i+1}|\}=1$。

2.1 直线绘制算法

2.1.1 DDA 算法

DDA 算法是利用对曲线微分方程积分的方法来绘制曲线的。DDA 算法通常用于绘制线段、多边形等，但也可用来绘制非线性曲线 [1]。对于直线 $y=kx+b$ ($|k|\leq 1$)，DDA 算法在每次递增 x 时，对 y 增加 k，并将取整后的值作为当前绘制点。利用 DDA 算法绘制线段的伪代码如下：

Algorithm 1 DDA 画线算法

Require: 线段的两个端点 $(x_1,y_1),(x_2,y_2)$。假定 $x_1<x_2,|x_2-x_1|\geq|y_2-y_1|$。

```
y = y_1, k = (y_2 - y_1)/(x_2 - x_1)
for x = x_1 to x_2 do
    绘制点 ([x], [y])
    y = y + k
end for
```

2.1.2 Bresenham 算法

Bresenham 算法的基本思想是，通过判断下两个绘制点的中点在直线的哪一侧来决定选取哪一个决策点。判断中点在直线哪一侧可以通过维护一个决策变量 Δ 来实现，而决策变量的维护通常可以利用整数的加减法实现 [2]，因此 Bresenham 算法比 DDA 算法更加高效。

对于以 $(x_1,y_1),(x_2,y_2)$（假设 $x_1<x_2,y_1\leq y_2,|x_1-x_2|\geq|y_1-y_2|$）为端点的线段，它的直线方程为

$$(y-y_1)(x_2-x_1)=(y_2-y_1)(x-x_1)$$

故可取决策变量 $\Delta(x,y)=2[(y_2-y_1)(x-x_1)-(y-y_1)(x_2-x_1)]$，并根据 $\Delta(x_i+1,y_i+0.5)$ 的符号决定绘制点。当决策变量为正时，递增 y，否则不递增 y。决策变量可用以下方式维护：

$$\Delta(x_i+1,y_i+0.5)=\Delta(x_i,y_i)+2\Delta y-\Delta x$$
$$\Delta(x_i+1,y_i+1)=\Delta(x_i+1,y_i+0.5)-\Delta x$$
$$\Delta(x_i+1,y_i)=\Delta(x_i+1,y_i+0.5)+\Delta x$$

Bresenham 算法的伪代码如下：

Algorithm 2 Bresenham 画线算法

Require: 线段的两个端点 $(x_1,y_1),(x_2,y_2)$。假定 $x_1<x_2,y_1\le y_2,|x_2-x_1|\ge|y_2-y_1|$。

```
y = y1, Δx = x2 − x1, Δy = y2 − y1
Δ = −Δx
for x = x1 to x2 do
    if Δ ≥ 0 then
        y = y + 1, Δ = Δ − Δx
    else
        Δ = Δ + Δx
    end if
    绘制点 (x, y)
    Δ = Δ + 2Δy − Δx
end for
```

其中，$[x]$ 表示 x 舍入至最近的整数。

2.2 多边形绘制算法

多边形的绘制算法可以很容易地由直线绘制算法导出。只需要用相应的直线绘制算法依次绘制多边形相邻两个顶点的连线段即可。

2.3 椭圆绘制算法（中点法）

中点法绘制椭圆的原理和 Bresenham 算法十分类似。在中点法中，仍然是通过计算相邻两个可能绘制点的中点在椭圆的哪一侧，从而决定在哪个点上绘制图形。但由于椭圆的特殊性，其绘制方法和直线的绘制方法存在以下差异：

- 由于椭圆有两条互相垂直且与坐标轴平行的对称轴，只需要绘制椭圆在任一象限中的图形，就可以通过对称的方式完成整个椭圆的绘制。
- 椭圆在任一象限中的图形均存在斜率的变化，因此必须以斜率为 1 的点作为分界点，对椭圆进行分段绘制。

不妨设椭圆中心位于原点。对于两轴半径分别为 r_x,r_y 的椭圆，其方程为 $\frac{x^2}{r_x^2}+\frac{y^2}{r_y^2}=1$，故可取 $p(x,y)=x^2r_y^2+y^2r_x^2-r_x^2r_y^2$ 为决策变量，并根据 $p(x+1,y+0.5)$（或 $p(x+0.5,y+1)$）的符号决定绘制点。决策点可按以下方式维护：

$$p(x_i+1,y_i+0.5)=p(x_i,y_i)+2r_y^2+r_x^2$$

$$p(x_i+1, y_i+1) = p(x_i, y_i) + 2r_y^2 + 2r_x^2$$
$$p(x_i+1, y_i) = p(x_i, y_i) + 2r_y^2$$

算法的伪代码如下：

Algorithm 3 中点法绘制椭圆

Require: 椭圆的两轴半径 r_x, r_y。

$p = r_y^2 - r_x^2 r y + r x^2/4$
$p_x = 0, p_y = 2r_x^2 r_y$
$c_x = 0, c_y = r_y$
while $x = x_1$ to x_2 **do**
    $c_x = c_x + 1, p_x = p_x + 2r_y^2$
    if $p < 0$ **then**
        $p = p + r_y^2 + px$
    else
        $c_y = c_y - 1, p_y = p_y - 2r_x^2$
        $p = p + r_y^2 + px - py$
    end if
    绘制点 (x, y)
end while
交换 x, y 后重复上述步骤。

2.4 曲线绘制算法

2.4.1 Bezier 曲线

Bezier 曲线是对给定样本点逼近的常用方法。一般来说，对于 $n+1$ 个样本点（或称控制点），我们可以绘制出 n 阶 Bezier 曲线。n 阶 Bezier 曲线实际上是以 $(tx + (1-t))^n$ 的二项式展开的各项系数为权函数，对样本点进行加权混合的结果，其公式如下:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} P_i t^i (1-t)^{n-i}$$

其中，$P_0, P_1, \cdots, P_n$ 为样本点。我们可以直接根据公式计算出 Bezier 曲线在某一点处的坐标，但 Bezier 曲线还存在一种递推的计算方式：

$$B_{i,j}(t) = tB_{i-1,j}(t) + (1-t)B_{i-1,j+1}(t)$$

其中，$B_{0,i}(t) = P_i$，$B_{n,0}(t) = B(t)$。这种计算方式避免了代价较高的浮点数指数运算，虽然时间复杂度由 $O(n)$ 上升为了 $O(n^2)$，但在曲线阶数较低时仍有较好的效果。

可以看到，Bezier 曲线实际上是由样本点确定的参数曲线，难以表示成显函数的形式，因此之前用于直线、椭圆绘制的中点法等方法不在适用。一种简单的绘制参数曲线的方法是，确定一个步长 η，以 η 为间隔等距确定参数 t 的值，然后依次画出曲线上的每个点。然而，参数 η 的确定通常是较为困难的，如果 η 设置地过小，则会产生很多重复或是多余的点，如果 η 设置地过大，则绘制的曲线会产生“缺口”。解决此问题的常用方法是自适应调整 η 的值：如果发现相邻两个绘制点重合，则调大 η 的值；如果相邻两个绘制点不连续，则调小 η 的值。

这里，我们采用一种更加优雅的方法——分治法。注意到，曲线绘制的过程实际上是将曲线上的部分点舍入到平面网格上的整数点，并满足以下条件：

1. 对于相邻两个绘制点 $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}$，满足 $\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|_\infty = 1$，其中 $\|(x,y)\|_\infty = \max\{|x|,|y|\}$ 为 ∞-范数；
2. 在满足上述条件的前提下，绘制点的数目尽可能少。

使用分治法绘制曲线时，首先检查两端点之间的 ∞-范数是否大于 1，如果是，则计算中点的坐标并递归绘制左右两段曲线。上述递归算法的终止条件保证了相邻两个绘制点之间满足 $\|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|_\infty = 1$，又能保证绘制点的个数尽可能少。

利用分治法绘制参数曲线 $P(t)$ 的伪代码如下：

Algorithm 4 分治法绘制参数曲线

Require: 曲线的参数方程 $P(t)$ $(0 \le t \le 1)$。

function 递归绘制 $(P, l, r, \boldsymbol{x}_l, \boldsymbol{x}_r)$
  if $\|[\boldsymbol{x_l}] - [\boldsymbol{x_r}]\|_\infty \le 1$ **then**
    return
  else
    $m = \frac{l+r}{2}$, $\boldsymbol{x}_m = [P(m)]$
    递归绘制 $(P, l, m, \boldsymbol{x}_l, \boldsymbol{x}_m)$
    绘制点 $[\boldsymbol{x}_m]$
    递归绘制 $(P, m, r, \boldsymbol{x}_m, \boldsymbol{x}_r)$
  end if
end function
$l = 0, r = 1, \boldsymbol{x}_l = P(0), \boldsymbol{x}_r = P(1)$
绘制点 $[\boldsymbol{x}_l]$
递归绘制 $(P, l, r, \boldsymbol{x}_l, \boldsymbol{x}_r)$
绘制点 $[\boldsymbol{x}_r]$

2.4.2 B-Spline 曲线

B-Spline 曲线是对 Bezier 曲线的一种改进。Bezier 曲线有计算量大，无法局部修改等问题，而 B-Spline 采用分段方法，可以较好地改善这些问题。

B-Spline 也是利用权函数对样本点混合得到。$n+1$ 个样本点上的 k 次 B-Spline 曲线的公式如下：

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

由于 B-Spline 需要分段绘制曲线，我们需要将参数取值分为若干**节**：

$$0 = u_0 \le u_1 \le u_2 \le \cdots \le u_m = 1$$

其中，$m = n + k + 1$。

B-Spline 的权函数定义如下：

$$B_{i,1}(u) = \begin{cases} 1, & u_i \le u \le u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$B_{i,k+1}(u) = \frac{u - u_i}{u_{i+k} - u_i} B_{i,k}(u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} B_{i+1,k}(u) \tag{2}$$

我们可以等距的划分节点，但这样的 B-Spline 曲线不经过起点和终点。为了确保 B-Spline 曲线经过起点和终点，我们一般将前 $k+1$ 个节点置为 0，后 $k+1$ 个节点置为 1，中间节点则等距选取：

$$u_i = \begin{cases} 0, & i \le k \\ \frac{i-k}{m-2k}, & k < i < m-k \\ 1, & i \ge m-k \end{cases}$$

在绘制 B-Spline 时，仍然采用和 Bezier 曲线相同的算法。

3 图元变换算法介绍

3.1 图元平移、旋转、缩放

图元的平移、旋转、缩放只需要对图元的关键点进行平移、旋转、缩放即可。假设旋转、缩放的中心都是原点，它们的公式分别为

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x + \Delta x \\ y + \Delta y \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} s_x x \\ s_y y \end{bmatrix} \tag{5}$$

对于旋转、缩放中心不是原点的情况，先将缩放点连同图元平移至原点，然后进行相应的变换操作，最后平移回去即可。

3.2 直线裁剪

给定线段 S 和矩形 R，求出 S 完全位于 R 内的部分，就是直线裁剪问题。本系统实现了 Cohen-Sutherland 和 Liang-Barsky 两种直线裁剪算法。

3.3 Cohen-Sutherland 算法

Cohen-Sutherland 算法是利用区位码进行裁剪的算法。将裁剪矩形窗的边界延长，可将平面分为 9 个区域，每个区域赋予一个 4 位的二进制编码。编码的前 2 位表示横坐标的位置，后 2 位表示纵坐标的位置。区位码的具体编码方式见图1。

1001	1000	1010
0001	0000	0010
0101	0100	0110

Figure 1: Cohen-Sutherland 算法的区位码

我们可以计算出被裁剪线段两个端点 P_1, P_2 的区位码 c_1, c_2，然后根据区位码得到线段和裁剪框的关系：

1. $c_1 = c_2 = 0$：此时线段完全位于裁剪框内，裁剪完成。
2. $c_1 \& c_2 \neq 0$（其中 & 表示按位与运算）：此时线段完全位于裁剪框的外侧，裁剪完成。
3. 否则，无法确定线段的位置。假设 $c_1 \neq 0$，则根据 c_1 中哪一位为 1，可以知道用该线段在矩形框外的哪一侧；将该侧部分裁剪掉后，重复整个裁剪过程。

3.4 Liang-Barsky 算法

Liang-Barsky 是利用直线参数方程裁剪的算法。利用参数方程，线段可以表示成 $\boldsymbol{b}t + \boldsymbol{a}$ $(t_0 \leq t \leq t_1)$ 的形式，只需要确定位于裁剪框内线段的起始点和终点的参数 t_0, t_1，就可以完成裁剪。

如图2所示，直线箭头方向为单位向量 $\boldsymbol{b}$ 的方向。直线和裁剪边框一共形成 4 个交点，其中两个是与水平线的交点，另外两个是和垂直线的交点。图中，每个方向上的第一个交点用红点表示，第二个交点用蓝点表示。取红点的参数和原始起点参数三者较大值作为 t_0，蓝点参数和原始终点参数三者较小值作为 t_1，即可得到被裁剪线段起点和终点的参数。需要注意的是，当 $t_0 > t_1$ 时，被裁剪线段完全位于裁剪框外，直接舍弃即可。

Liang-Barsky 算法的一个缺点是，当被裁剪线段与坐标轴平行和垂直时，将会出现退化的情况，该情况需要特殊处理。

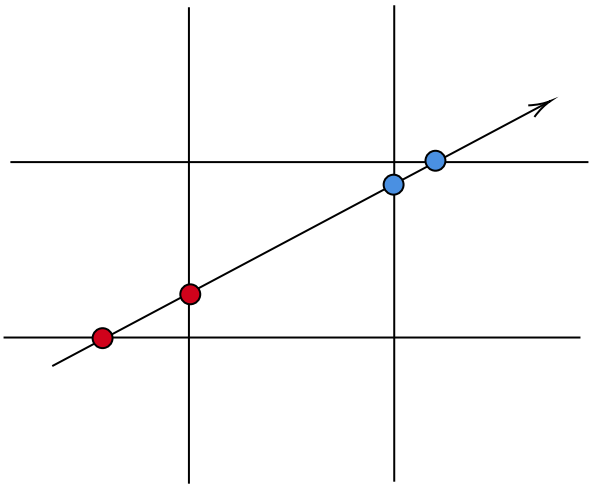

Figure 2: Liang-Barsky 算法示例

4 系统介绍

5 总结

…

参考文献

[1] Wikipedia. Digital differential analyzer (graphics algorithm) — Wikipedia, the free encyclopedia, 2019. [Online; accessed 12-April-2019].

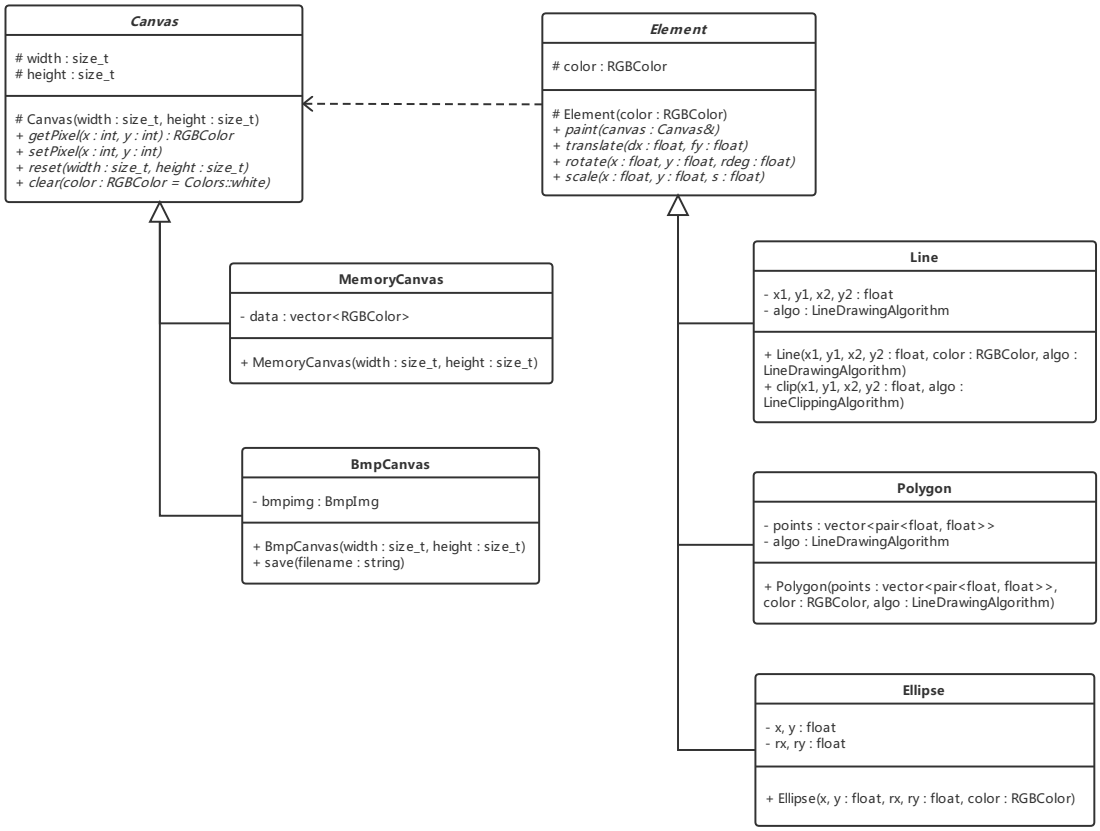

Figure 3: 系统的 UML 类图

[2] Wikipedia. Bresenham's line algorithm — Wikipedia, the free encyclopedia, 2019. [Online; accessed 12-April-2019].