抛物线的中点 Bresenham 算法

1 抛物线的特征

通常定义抛物线为到一条直线（准线）和直线外一点（焦点）距离相等的点的集合。这里只讨论顶点为原点，沿纵坐标轴对称且开口向上的情况。而对于其他情况可以通过图形的平移和旋转等线性变换得到。其描述方程如下：

$$F(x,y)=y\text{-}ax^2(a>0)$$

与椭圆不同，抛物线是无边界的非封闭图形，若要在屏幕上绘制，必须给定坐标范围，以绘制指定抛物线的一个片段。可以在函数中设置参数 x_border，则横坐标约束其范围为 $\left[-x\_border,\ x\_border\right]$。

抛物线关于纵坐标轴对称，故只需绘制其第一象限内的点，第二象限中的点可以通过对称得到。

为确定最大位移方向，考虑抛物线的斜率范围。在第一象限，其上一点(x，y)处的斜率为：

$$k(x)=\frac{\partial F(x,y)}{\partial x}=2ax\in[0,+\infty]$$

又由于斜率的变化率为：

$$\frac{dk(x)}{dx}=\frac{d(2ax)}{dx}=2a>0$$

所以在第一象限内，抛物线斜率从 0 开始随 x 递增至正无穷。用斜率为 1 的点对图形进行划分。容易解出，当斜率为 1 时，$\mathrm{x}=\frac{1}{2a}$。只需沿 x 轴绘制图形，其中 $0<x<\frac{1}{2a}$ 时，最大位移方向为 x 方向；$\frac{1}{2a}\le x<x\_border$ 时，最大位移方向为 y 方向。

2 算法推导过程

假定当前与抛物线距离最近者已确定为 $P(x_i,y_i)$，那么在抛物线前部分时，下一候选点是 $P_d(x_i+1,y_i)$ 和 $P_u(x_i+1,y_i+1)$；而在抛物线的后半部分时，下一候选点是 $P_l(x_i,y_i+1)$ 和 $P_r(x_i+1,y_i+1)$。如何选用候选点仍然使用中点进行判别。

2.1 推导前半部分的抛物线绘制公式

对于 $0<x<\frac{1}{2a}$，构造判别式

$$d_{1i}=F(x_i+1,y_i+0.5)=y_i+0.5-a(x_i+1)^2$$

若 $d_{li} \geq 0$，中点在抛物线上方，应取正右方候选点 $P_d(x_i+1, y_i)$；反之，中点在抛物线下方，应取右上方候选点 $P_u(x_i+1, y_i+1)$。

前半部分误差项的递推。

在 $d_{li} \geq 0$ 时，应计算：

$$\begin{aligned} d_{l(i+1)} &= F(x_i+2, y_i+0.5) \\ &= y_i + 0.5 - a(x_i+2)^2 \\ &= d_{li} + a[(x_i+1)^2 - (x_i+2)^2] \\ &= d_{li} - 2ax_i - 3a \end{aligned}$$

在 $d_{li} < 0$ 时，应计算：

$$\begin{aligned} d_{l(i+1)} &= F(x_i+2, y_i+1.5) \\ &= y_i + 1.5 - a(x_i+2)^2 \\ &= d_{li} - 2ax_i - 3a + 1 \end{aligned}$$

计算判别式的初始值。弧起点为 $(0,0)$，因此第一个中点为 $(1, 0.5)$，对应的判别式为：

$$d_{l0} = y_0 + 0.5 - a(x_0+1)^2 = 0.5 - a$$

2.2 推导后半部分的抛物线绘制公式

对于 $\frac{1}{2a} \leq x < x\_border$，构造判别式

$$d_{ri} = F(x_i+0.5, y_i+1) = y_i + 1 - a(x_i+0.5)^2$$

若 $d_{ri} \leq 0$，中点在抛物线（左）上方，应取正上方候选点 $P_l(x_i, y_i+1)$；反之，中点在抛物线（右）下方，应取右上方候选点 $P_r(x_i+1, y_i+1)$。

后半部分误差项的递推。

在 $d_{ri} \leq 0$ 时，应计算：

$$\begin{aligned} d_{r(i+1)} &= F(x_i+0.5, y_i+2) \\ &= y_i + 2 - a(x_i+0.5)^2 \\ &= d_{ri} + 1 \end{aligned}$$

在 $d_{ri} > 0$ 时，应计算：

$$
\begin{aligned}
d_{r(i+1)} &= F(x_i + 1.5, y_i + 2) \\
&= y_i + 2 - a(x_i + 1.5)^2 \\
&= d_{ri} + 1 + a[(x_i + 0.5)^2 - (x_i + 1.5)^2] \\
&= d_{ri} - 2ax_i - 2a + 1
\end{aligned}
$$

计算判别式的初始值。弧起点的横坐标为$\left\lceil \frac{1}{2a} \right\rceil$，对应的判别式为：

$$
\begin{aligned}
d_{r0} &= y_0 + 1 - a(x_0 + 0.5)^2 \\
&= y_0 + 1 - ax_0^2 - ax_0 - 0.25a \\
&= 1 - ax_0 - 0.25a \\
&= 1 - a\left\lceil \frac{1}{2a} \right\rceil - 0.25a
\end{aligned}
$$

3 算法的伪代码

```
中点 Bresenham 算法绘制抛物线函数(a 值，边界值 x_border):
    计算分界点 div=0.5/a;
    初始化:
        前半部分判别式初始值: d_pre = 0.5 - a;
        后半部分判别式初始值: d_post = 1 - a * ceil(div) - 0.25 * a
        x=x0, y=y0;
    当 x<x_border 循环执行:
        绘制点(x, y)和(-x, y)
        若 x<div:
            计算增量 tmp = -2 * a * x - 3 * a;
            x ++;
            如果 d_pre < 0:
                y ++;
                d_pre += tmp + 1;
            否则:
                d_pre += tmp;
        否则:
            计算增量 tmp = -2 * a * x - 2 * a + 1;
            y ++;
            如果 d_post >= 0:
                x ++;
                d_post += tmp;
            否则:
                d_post += 1;
```

4 一些注意事项

（1）为了使运行结果更利于观察，可以将画布网格化，产生离散的可选点集。设 grid_size 为网格的间距，将可选点的横纵坐标限制为 grid_size 的整数

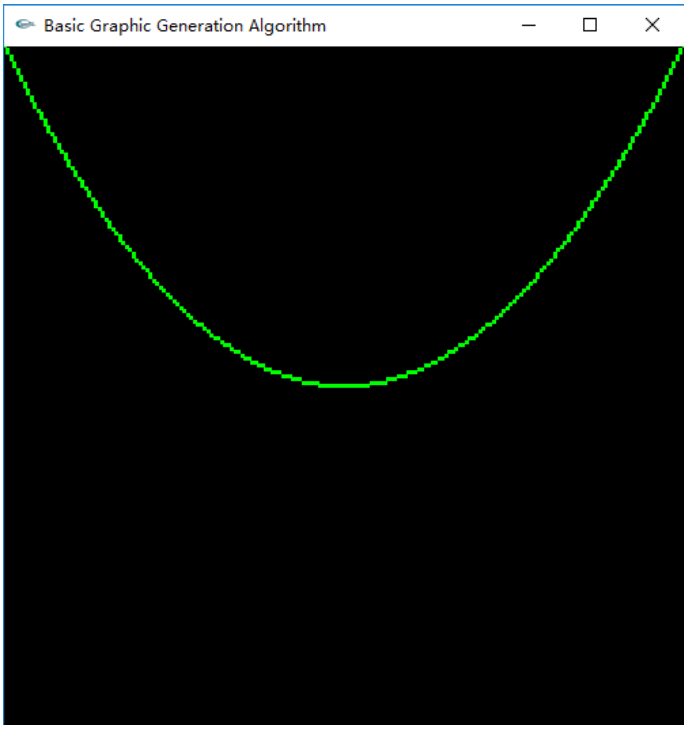
倍。

（2）注意在算法中计算增量 tmp 的步骤一定要在 x++与 y++之前。因为 tmp 的计算是基于上一步骤中的点的。

（3)此算法涉及到了很多浮点数运算与取整运算,可能导致算法效率不高。在实际应用时，可以用 $4ad_i$ 替换 d_i，以消除初始化时的复杂运算。

5 运行结果示例

以下是基于 C++和 OpenGL 编写的,使用参数值为 grid_size=0.01,a=0.01,x_border=100 的实例的运行结果。



6 附录：源代码

```
#define GLUT_DISABLE_ATEXIT_HACK
#include <windows.h>
#include <GL/glut.h>
#include <cmath>
using namespace std;

const float grid_size = 0.01f;

void paracurve_midpoint_bresenham(float a, int x_border) {
    float div = 0.5 / a;
    int x = 0, y = 0;
    float d_pre = 0.5 - a;
    float d_post = 1 - a * ceil(div) - 0.25 * a;
    glPointSize(3.0f);
```

```
    glBegin(GL_POINTS);
    while (x <= x_border) {
        glVertex2f(x * grid_size, y * grid_size);
        glVertex2f(-x * grid_size, y * grid_size);
        if (x < div) {
            float tmp = -2 * a * x - 3 * a;
            x ++;
            if (d_pre < 0) {
                y ++;
                d_pre += tmp + 1;
            } else {
                d_pre += tmp;
            }
        } else {
            float tmp = -2 * a * x - 2 * a + 1;
            y ++;
            if (d_post >= 0) {
                x ++;
                d_post += tmp;
            } else {
                d_post += 1;
            }
        }
    }
    glEnd();
}

void reDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0f, 1.0f, 0.0f);
    paracurve_midpoint_bresenham(0.01, 100);
    glFlush();
}

int main(int argc, char**argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Basic Graphic Generation Algorithm");
    glutDisplayFunc(&reDisplay);
    glutMainLoop();
}
```