Министерство образования и науки РФ
ГОУ ВПО Рыбинский государственный авиационно-технический университет имени П.А. Соловьева

Факультет радиоэлектроники и информатики
Кафедра математического и программного обеспечения электронных вычислительных средств

Специальность
«Програмное обеспечение вычислительной техники
и автоматизированных систем»

Лабораторная работа №1
по дисциплине
«Компьютерная графика»

Студенты гр. ИВП-09 ______________________ Кулаевский Д.Ю.
Руководитель ______________________ Малышев О.В.

Рыбинск 2012

1 Постановка задачи

Разобрать и реализовать растровые алгоритмы рисования линий и окружности с помощью алгоритма Брезенхэма для линий и кривых 2-го порядка.

2 Карткие теоретические сведения

Алгоритм Брезенхема (англ. Bresenham's line algorithm) — это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками. Это один из старейших алгоритмов в машинной графике — он был разработан Джеком Е. Брезенхэмом (Jack E. Bresenham) в компании IBM в 1962 году. Алгоритм широко используется, в частности, для рисования линий на экране компьютера. Существует обобщение алгоритма Брезенхэма для построения кривых 2-го порядка.

Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат — либо x, либо y (в зависиимости от углового коэффициента) — изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой. Алгоритм построен так, что требуется проверить лишь знак этой ошибки.

Общая формула линии между двумя точками:

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

.

Поскольку мы знаем колонку x, то строка y получается округлением к целому следующего значения:

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

.

Однако, вычислять точное значение этого выражения нет необходимости. Достаточно заметить, что y уменьшается от y_0 и за каждый шаг мы добавляем к x единицу и добавляем к y значение наклона (в нашем случае значение наклона будет отрицательным числом).

$$s = \frac{y_1 - y_0}{x_1 - x_0}$$

Это значение можно вычислить заранее. Более того, на каждом шаге мы делаем одно из двух: либо сохраняем тот же y, либо уменьшаем его на 1.

Что из этих двух выбрать — можно решить, отслеживая значение ошибки, которое означает — вертикальное расстояние между текущим значением y и точным значением y для текущего x. Всякий раз, когда мы увеличиваем x, мы увеличиваем значение ошибки на величину наклона s, приведённую выше. Если ошибка превысила 0,5, линия стала ближе к следующему y, поэтому мы уменьшаем y на единицу, одновременно уменьшая значение ошибки на 1.

Во многих областях приложений, таких как, например, системы автоматизированного проектирования машиностроительного направления, естественными графическими примитивами, кроме отрезков прямых и строк текстов, являются и конические сечения, т.е. окружности, эллипсы, параболы и гиперболы. Наиболее употребительным примитивом, естественно, является окружность. Одиним из наиболее простых и эффективных алгоритмов генерации окружности считается алгоритм Брезенхема для кривых второго порядка.

Для простоты и без ограничения общности рассмотрим генерацию 1/8 окружности, центр которой лежит в начале координат. Остальные части окружности могут быть получены последовательными отражениями (использованием симметрии точек на окружности относительно центра и осей координат).

Окружность с центром в начале координат описывается уравнением:

$$x^2 + y^2 = R^2$$

Алгоритм Брезенхема пошагово генерирует очередные точки окружности, выбирая на каждом шаге для занесения пиксела точку растра $P_i(x_i, y_i)$, ближайшую к истинной окружности, так чтобы ошибка:

$$E_i(P_i) = (x_i^2 + y_i^2) - R^2$$

была минимальной. Причем, как и в алгоритме Брезенхема для генерации отрезков, выбор ближайшей точки выполняется с помощью анализа значений управляющих переменных, для вычисления которых не требуется вещественной арифметики. Для выбора очередной точки достаточно проанализировать знаки.

Рассмотрим генерацию 1/8 окружности по часовой стрелке, начиная от точки $x = 0, y = R$.

Проанализируем возможные варианты занесения $i+1$-й точки, после занесения i-й.

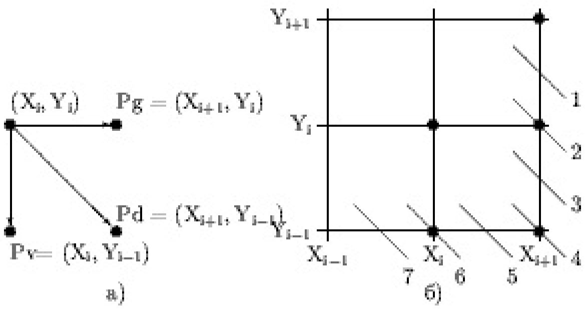

Рисунок 1 – Варианты расположения очередного пиксела окружности

При генерации окружности по часовой стрелке после занесения точки (x_i, y_i) следующая точка может быть (см. рис. 1 а)) либо $P_g = (x_{i+1}, y_i)$ — перемещение по горизонтали, либо $P_d = (x_{i+1}, y_{i-1})$ — перемещение по диагонали, либо $P_v = (x_i, y_{i-1})$ - перемещение по вертикали.

Для этих возможных точек вычислим и сравним абсолютные значения разностей квадратов расстояний от центра окружности до точки и окружности:

$$|D_g| = |(x+1)^2 + y^2 - R^2|$$
$$|D_d| = |(x+1)^2 + (y-1)^2 - R^2|$$
$$|D_v| = |x^2 + (y-1)^2 - R^2|$$

Выбирается и заносится та точка, для которой это значение минимально.

Выбор способа расчета определяется по значению D_d. Если $D_d < 0$, то диагональная точка внутри окружности. Это варианты 1-3 (см. рис. 1 б)). Если $D_d > 0$, то диагональная точка вне окружности. Это варианты 5-7. И, наконец, если $D_d = 0$, то диагональная точка лежит точно на окружности. Это вариант 4. Рассмотрим случаи различных значений D_d в только что приведенной последовательности.

Случай $D_d < 0$ Здесь в качестве следующего пиксела могут быть выбраны или горизонтальный — P_g или диагональный — P_d.

Для определения того, какой пиксел выбрать P_g или P_d составим разность:

$$d_i = |D_g| - |D_d| = |(x+1)^2 + yY^2 - R^2| - |(x+1)^2 + (y-1)^2 - R^2|$$

И будем выбирать точку P_g при $d_i < 0$, в противном случае выберем P_d.

Рассмотрим вычисление d_i для разных вариантов.

Варианты 2 и 3 $D_g > 0$ и $D_d < 0$, так как горизонтальный пиксел либо вне, либо на окружности, а диагональный внутри.

$$d_i = (x+1)^2 + y^2 - R^2 + (x+1)^2 + (y-1)^2 - R^2$$

Добавив и вычтя $(y-1)^2$ получим:

$$d_i = 2[(x+1)^2 + (y-1)^2 - R^2] + 2y - 1$$

В квадратных скобках стоит D_d, так что

$$d_i = 2(D_d + y) - 1$$

Вариант 1 Ясно, что должен быть выбран горизонтальный пиксел P_g. Проверка компонент d_i показывает, что $D_g < 0$ и $D_d < 0$, причем $d_i < 0$, так как диагональная точка больше удалена от окружности, т.е. по критерию $d_i < 0$ как и в предыдущих случаях следует выбрать горизонтальный пиксел P_g, что верно.

Случай $D_d > 0$ Здесь в качестве следующего пиксела могут быть выбраны или диагональный - P_d или вертикальный P_v. Для определения того, какую пиксел выбрать P_d или P_v составим разность:

$$s_i = |D_d| - |D_v| = |(x+1)^2 + (y-1)^2 - R^2| - |x^2 + (y-1)^2 - R^2|$$

Если $s_i < 0$, то расстояние до вертикальной точки больше и надо выбирать диагональный пиксел P_d, если же $s_i > 0$, то выбираем вертикальный пиксел P_v.

Рассмотрим вычисление s_i для разных вариантов.

Варианты 5 и 6 $D_d > 0$ и $D_v < 0$, так как диагональный пиксел вне, а вертикальный либо вне либо на окружности.

$$s_i = (x+1)^2 + (y-1)^2 - R^2 + x^2 + (y-1)^2 - R^2;$$

Добавив и вычтя $(x+1)^2$ получим:

$$s_i = 2[(x+1)^2 + (y-1)^2 - R^2] - 2x - 1$$

В квадратных скобках стоит D_d, так что

$$s_i = 2(D_d - x) - 1$$

Вариант 7 Ясно, что должен быть выбран вертикальный пиксел P_v. Проверка компонент s_i показывает, что $D_d > 0$ и $D_v > 0$, причем $s_i > 0$, так как диагональная точка больше удалена от окружности, т.е. по критерию $s_i > 0$ как и в предыдущих случаях следует выбрать вертикальный пиксел P_v, что соответствует выбору для вариантов 5 и 6.

Случай $D_d = 0$ Для компонент d_i имеем: $D_g > 0$ и $Dd = 0$, следовательно по критерию $d_i > 0$ выбираем диагональный пиксел.

С другой стороны, для компонент s_i имеем: $D_d = 0$ и $D_v < 0$, так что по критерию $s_i < 0$ также выбираем диагональный пиксел.

Итак:

1) $D_d < 0$

    1) $d_i < 0$ — выбор горизонтального пиксела P_g
    2) $d_i > 0$ — выбор диагонального пиксела P_d

2) $D_d > 0$

    1) $s_i < 0$ — выбор диагонального пиксела P_d
    2) $s_i > 0$ — выбор вертикального пиксела P_v

3) $D_d = 0$ — выбор диагонального пиксела Pd.

Выведем рекуррентные соотношения для вычисления D_d для (i+1)-го шага, после выполнения i-го.

1) Для горизонтального шага к $x_i + 1$, y_i

$$\begin{aligned}
&x_{i+1} = x_i + 1\\
&y_{i+1} = y_i\\
&D_{d_i} + 1 = (x_{i+1} + 1)^2 + (y_{i+1} - 1)2 - R^2 =\\
&x_{i+1}^2 + 2X_{i+1} + 1 + (y_{i+1} - 1^)2 - R^2 =\\
&(x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2X_{i+1} + 1 =\\
&D_{d_i} + 2X_{i+1} + 1
\end{aligned}$$

2) Для диагонального шага к $x_i + 1$, $y_i - 1$

$$
\begin{aligned}
x_{i+1} &= x_i + 1 \\
y_{i+1} &= y_i - 1 \\
D_{d_{i+1}} &= D_{d_i} + 2x_{i+1} - 2y_{i+1} + 2
\end{aligned}
$$

3) Для вертикального шага к x_i, $y_i - 1$

$$
\begin{aligned}
x_{i+1} &= x_i \\
y_{i+1} &= y_i - 1 \\
D_{d_{i+1}} &= D_{d_i} - 2y_{i+1} + 1
\end{aligned}
$$

3 Исходный код основных процедур программы

3.1 Рисование линии методом Брезенхема

```
void drawLineBrezenhema(QPainter* painter, int x1,int y1, int x2, int y2)
{
    int f, x, y, dx, dy, incrE, incrNE, stepX, stepY, L, l;

    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    L = max2(dx, dy);
    l = min2(dx, dy);
    f  = 2*l - L;
    incrE = 2*l;
    incrNE = 2*l - 2*L;
    stepX = sign(x2 - x1);
    stepY = sign(y2 - y1);

    x = x1; y = y1;
    painter->drawPoint(x, y);

    for(int i=0; i<L; ++i)
    {
        if( f > 0 )
        {
            if(L == dx)
                y += stepY;
            else
                x += stepX;
            f = f + incrNE;
        }
        else
            f = f + incrE;

        if(L == dx)
            x += stepX;
        else
            y += stepY;

        painter->drawPoint(x, y);
    }
}
```

3.2 Рисование окружности методом Брезенхема

```
void drawCircBrezenhema(QPainter* painter, int x1,int y1, int x2, int y2)
{
    int dx = (x2-x1), dy = (y2-y1),
        radius = round(pow(dx*dx+dy*dy,0.5)),
        x =  0, y = radius,
        f = 1 - radius,
        incrE = 3, incrSE = 5 - 2*radius;

    putPixel4Circle(painter, x1, y1, radius);
    while(x <= y)
    {
        if(f > 0)
        {
            y -= 1;
            f += incrSE;
            incrSE += 4;
        }
        else
        {
            f += incrE;
            incrSE += 2;
        }
        incrE += 2;
        x += 1;
        putPixel8Circle(painter, x1, y1, x, y);
    }
}
```

4 Выводы

В ходе лабораторной работы были разобраны основные принципы растровой графики, требования к алгоритмам и сами алгоритмы апроксимации линий и кривых на дискретном графическом устройстве. Были изучены такие алгоритмы, как: DDA-алгоритм, оптимизированный DDA-алгоритм, алгоритм Брезенхема для построения отрезка, алгоритм Брезенхема для кривой 2-го порядка. Главным преимуществом алгоритма Брезенхема является использование целочисленной арифметики и только операций сложения и вычитания, что дает ему преимущество в скорости над алгоритмом DDA.