G01. Алгоритм Брезенхэма для прямой.

Ускорение алгоритма производится за счет рисования только половины отрезка.
Для этого, алгоритм необходимо модифицировать следующим образом:

пусть необходимо нарисовать точку

$$(x_{curr}, y_{curr}) \quad ,$$

тогда необходимо нарисовать и точку

$$(x_2-(x_{curr}-x_1), y_2-(y_{curr}-y_1)) \quad , \text{где}$$

x_1 , y_1 и x_2 , y_2 - начальные и конечные точки по соответствующей координате соответственно.

Данный способ «нахождения» зеркальной точки — теоретический, и, в зависимости от реализации, может быть выражен выражен более удобно, к примеру:

$$(deltaX-1-coordX, deltaY-1-i)$$

(см. код *Bresenham_line.c*).

Результаты замеров показали ускорение как минимум на 5% и, в зависимости от реализации и размера отрезка, может достигать как минимум 17%.

Примеры реализации:

```
void drawLine(int x1, int y1, int x2, int y2)
{
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;

    int error = deltaX - deltaY;

    setPixel(x2, y2);
    while(x1 != x2 || y1 != y2)
    {
        setPixel(x1, y1);
        const int error2 = error * 2;

        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2  deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }
}
```

А так же:

specnano/gcode/Bresenham_line.c
specnano/gcode/Bresenham_line_accelerated.c

См. так же:

G01: зависимость длины от угла наклона

G02-G03. Алгоритм Брезенхэма для окружности.

Помимо прямых линий, *алгоритм Брезенхема* может рисовать кривые второго порядка, т.е. существует возможность отрисовки окружности с помощью этого алгоритма. Точки отрисованной кривой, представляющей из себя дугу четверти окружности, зеркально отражаются относительно центра на остальные четверти окружности. Алгоритм требует задания координат центра и радиуса.
Пример реализации:

```
void drawCircle(int x0, int y0, int radius)
{
        int x = 0;
        int y = radius;
        int delta = 2 - 2 * radius;
        int error = 0;
        while(y >= 0)
        {
                setPixel(x0 + x, y0 + y);
                setPixel(x0 + x, y0 - y);
                setPixel(x0 - x, y0 + y);
                setPixel(x0 - x, y0 - y);
                error = 2 * (delta + y) - 1;
                if(delta < 0 && error <= 0)
                {
                        ++x;
                        delta += 2 * x + 1;
                        continue;
                }
                error = 2 * (delta - x) - 1;
                if(delta > 0 && error > 0)
                {
                        --y;
                        delta += 1 - 2 * y;
                        continue;
                }
                ++x;
                delta += 2 * (x - y);
                --y;
        }
}
```

А так же:

specnano/gcode/Bresenham_circle.c

Попытка ускорить алгоритм рисованием восьмой и меньших долей окружности приводит к алгоритму *Midpoint Circle Algorithm*.

G02-G03. Midpoint Circle Algorithm.

Данный алгоритм проходит только 1/8 часть окружности, остальные части отражаются симметрично.
Результаты замеров показывают ускорение как минимум на 2% и на 35% в среднем:

Радиус	Ускорение, %
100	7
500	15
1000	61
5000	52
10000	35
20000	30

Пример реализации:

```
void drawCircle(int x0, int y0, int radius)
{
  int x = radius, y = 0;
  int radiusError = 1-x;

  while(x >= y)
  {
    DrawPixel(x + x0, y + y0);
    DrawPixel(y + x0, x + y0);
    DrawPixel(-x + x0, y + y0);
    DrawPixel(-y + x0, x + y0);
    DrawPixel(-x + x0, -y + y0);
    DrawPixel(-y + x0, -x + y0);
    DrawPixel(x + x0, -y + y0);
    DrawPixel(y + x0, -x + y0);

    y++;
        if(radiusError<0)
                radiusError+=2*y+1;
        else
        {
                x--;
                radiusError+=2*(y-x+1);
        }
  }
}
```

А так же:

specnano/gcode/Midpoint.c

Модификация данного алгоритма на 1/16 и меньшие доли окружности требует дополнительных исследований и в ряде случаев не требуется.

См. так же:

G02, G03: зависимость деления от радиуса