# MODUL 2

KOMPUTER GRAFIK 2D
PRIMITIF 2D DAN ATRIBUT

D4 TEKNIK INFORMATIKA
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI BANDUNG

MAHASISWA 000 | KOMPUTER GRAFIK | FEB, 19 2025

# CONTENTS

## RIVIEW SIGNAL

1. On – Off Grid
2. On – Off Kartesian

## LINE DRAWING ALGORITHM

A scan-conversion algorithm for lines computes the coordinates of the pixels that lie on or near an ideal, infinitely thin straight line imposed on a 2D raster grid. In principle, we would like the sequence of pixels to lie as close to the ideal line as possible and to be as straight as possible. Consider a 1-pixel-thick approximation to an ideal line; what properties should it have? For lines with slopes between $-1$ and 1 inclusive, exactly 1 pixel should be illuminated in each column; for lines with slopes outside this range, exactly 1 pixel should be illuminated in each row. All lines should be drawn with constant brightness, independent of length and orientation, and as rapidly as possible. There should also be provisions for drawing lines that are more than 1 pixel wide, centered on the ideal line, that are affected by line-style and pen-style attributes, and that create other effects needed for high-quality illustrations. For example, the shape of the endpoint regions should be under programmer control to allow beveled, rounded, and mitered corners. We would even like to be able to minimize the jaggies due to the discrete approximation of the ideal line by using antialiasing techniques exploiting the ability to set the intensity of individual pixels on $n$-bits-per-pixel displays.

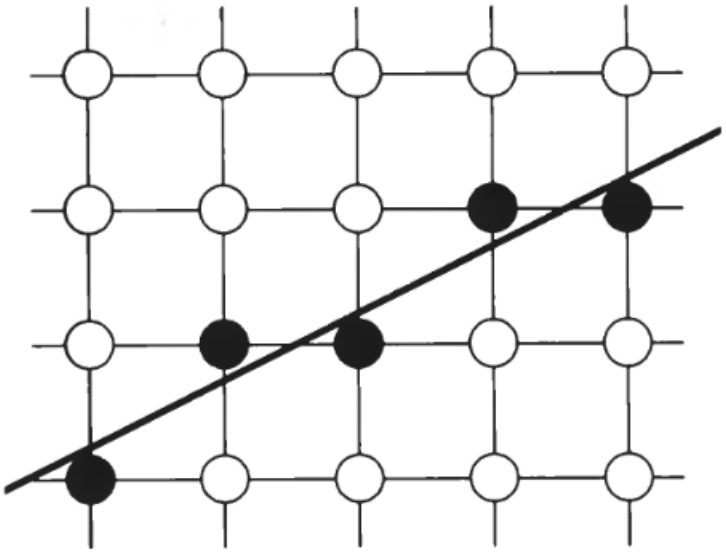

**Fig. 3.4** A scan-converted line showing intensified pixels as black circles.

## LINE DDA



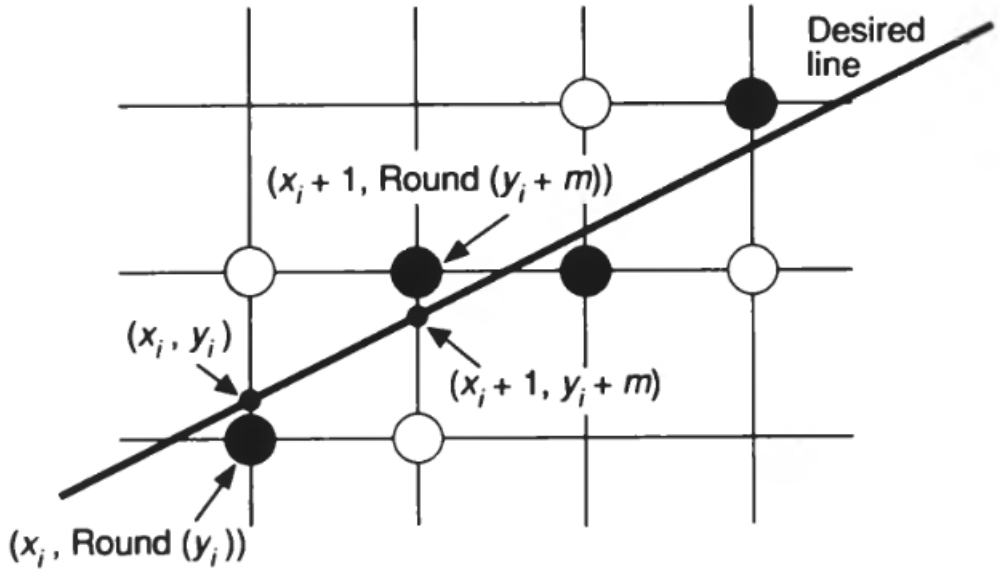


**Fig. 3.5** Incremental calculation of $(x_i, y_i)$.

This algorithm is often referred to as a *digital differential analyzer (DDA)* algorithm. The DDA is a mechanical device that solves differential equations by numerical methods: It traces out successive $(x, y)$ values by simultaneously incrementing $x$ and $y$ by small steps proportional to the first derivative of $x$ and $y$. In our case, the $x$ increment is 1, and the $y$ increment is $dy/dx = m$. Since real variables have limited precision, summing an inexact $m$ repetitively introduces cumulative error buildup and eventually a drift away from a true Round($y_i$); for most (short) lines, this will not present a problem.

```
void Line (                                  /* Assumes -1 ≤ m ≤ 1, x0 < x1 */
        int x0, int y0,                      /* Left endpoint */
        int x1, int y1,                      /* Right endpoint */
        int value)                           /* Value to place in line's pixels */
{
    int x;                                   /* x runs from x0 to x1 in unit increments. */

    double dy = y1 - y0;
    double dx = x1 - x0;
    double m = dy / dx;
    double y = y0;
    for (x = x0; x <= x1; x++) {
        WritePixel (x, Round (y), value);    /* Set pixel to value */
        y += m;                              /* Step y by slope m */
    }
}  /* Line */
```

**Fig. 3.6** The incremental line scan-conversion algorithm.

## BERSENHAM

The drawbacks of procedure Line are that rounding $y$ to an integer takes time, and that the variables $y$ and $m$ must be real or fractional binary because the slope is a fraction. Bresenham developed a classic algorithm [BRES65] that is attractive because it uses only

integer arithmetic, thus avoiding the Round function, and allows the calculation for $(x_{i+1}, y_{i+1})$ to be performed incrementally—that is, by using the calculation already done at $(x_i, y_i)$. A floating-point version of this algorithm can be applied to lines with arbitrary real-valued endpoint coordinates. Furthermore, Bresenham's incremental technique may be applied to the integer computation of circles as well, although it does not generalize easily to arbitrary conics. We therefore use a slightly different formulation, the *midpoint technique*, first published by Pitteway [PITT67] and adapted by Van Aken [VANA84] and



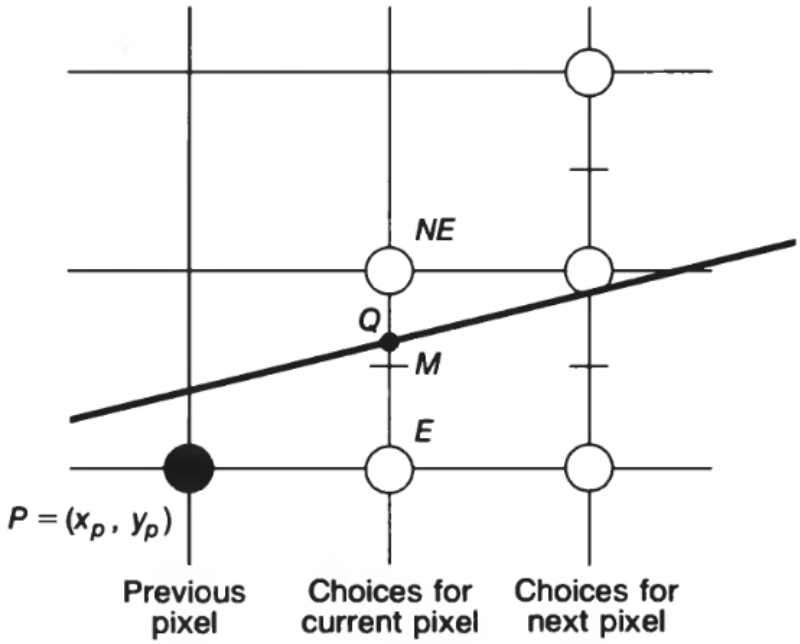


**Fig. 3.7** The pixel grid for the midpoint line algorithm, showing the midpoint *M*, and the *E* and *NE* pixels to choose between.

```
void MidpointLine (int x0, int y0, int x1, int y1, int value)
{
    int dx = x1 − x0;
    int dy = y1 − y0;
    int d = 2 ∗ dy − dx;              /∗ Initial value of d ∗/
    int incrE = 2 ∗ dy;               /∗ Increment used for move to E ∗/
    int incrNE = 2 ∗ (dy − dx);       /∗ Increment used for move to NE ∗/
    int x = x0;
    int y = y0;
    WritePixel (x, y, value);         /∗ The start pixel ∗/

    while (x < x1) {
        if (d <= 0) {                 /∗ Choose E ∗/
            d += incrE;
            x++;
        } else {                      /∗ Choose NE ∗/
            d += incrNE;
            x++;
            y++;
        }
        WritePixel (x, y, value);     /∗ The selected pixel closest to the line ∗/
    }  /∗ while ∗/

}  /∗ MidpointLine ∗/
```

**Fig. 3.8** The midpoint line scan-conversion algorithm.

## FUNGSI PERUBAHAN KARTESIAN KOORDINAT

```
# pixel coordinat to kartesian koordinat untuk fungsi
func convert_to_kartesian(xa, ya, xb, yb, margin_left, margin_top):
    var axis = ceil(float(OS.window_size.x / 2))
    var ordinat = ceil(float(OS.window_size.y / 2))
    xa = axis + xa
    xb = axis + xb
    ya = ordinat - ya
    yb = ordinat - yb

    return [xa,ya,xb,yb]
```

# FUNGSI WORLD KOORDINAT

```
# world coordinat to pixel coordinat untuk bentuk dasar
func convert_to_pixel(xa, ya, xb, yb, margin_left, margin_top):
    xa = margin_left+xa
    xb = margin_left+xb
    ya = OS.window_size.y -margin_top - ya
    yb = OS.window_size.y -margin_top - yb
    return [xa,ya,xb,yb]
```

## TASK 1: MANIPULASI GARIS

**Tugas Task 1**

1. Pelajari implementasi LineDDA dan Line Bresenham
2. Konversi fungsi Kartesian Koordinat dan World Koordinat
3. Tambahkan sebuah fungsi menggunakan rumus lain seperti absolute, exponensial, limit dll pada Karya 1 pada kuadran 1
4. Tuliskan Lesson Learnt Task 1

| Lesson Learnt |
| --- |
| |

## TASK 2: BENTUK DASAR SEDERHANA

**Tugas Task 2**

1. Perhatikan Contoh Bentuk Sederhana Persegi dan Persegi Panjang
2. Lengkapi Sheet eksperimen untuk bentuk dasar dan buatlah persamaan / parameter apa saja yang digunakan untuk membangun bentuk dasar tersebut.
3. Lengkapi fungsi pembentuk bentuk dasar, segi tiga siku-siku, trapesium siku, segitiga sama kaki, trapesium sama kaki.
4. Konversi fungsi Kartesian Koordinat dan World Koordinat, buat 4 di masing kuadran bentuk dasar itu.

| Lesson Learnt |
| --- |
| |

## PENGUMPULAN

Ikuti Format yang diberikan di Google Classroom.