Suranaree University
of Technology

COMPUTER GRAPHICS
School of Computer Engineering

Lecture 1 Review

Paramate Horkaew

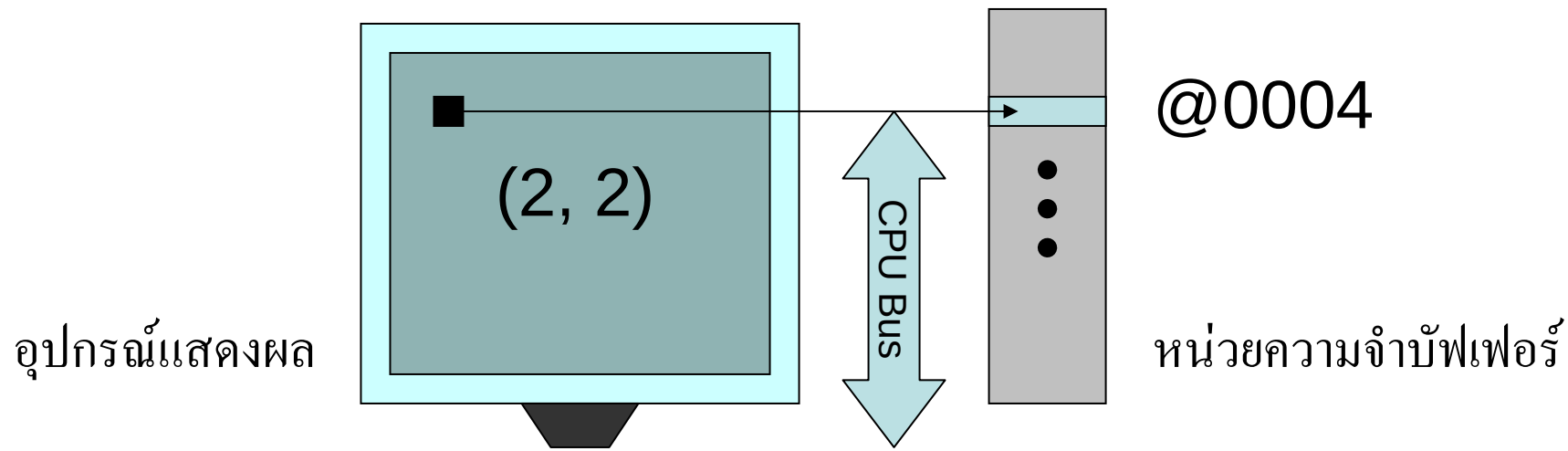
School of Computer Engineering, Institute of Engineering
Suranaree University of Technology

Raster Devices

Raster Devices

คืออุปกรณ์แสดงผล ที่พบได้โดยทั่วไป ได้แก่ จอภาพ และ เครื่องพิมพ์

ลักษณะจำเพาะคือ แต่ละจุด หรือ **Pic**ture **El**ement (Pixel) ซึ่งประกอบกันขึ้นเป็นภาพบนอุปกรณ์แสดงผล จะอ้างถึง (Map) หน่วยความจำ แบบ Random Access (RAM) ซึ่งสามารถเข้าถึงได้ โดย CPU หรือ Register ควบคุม

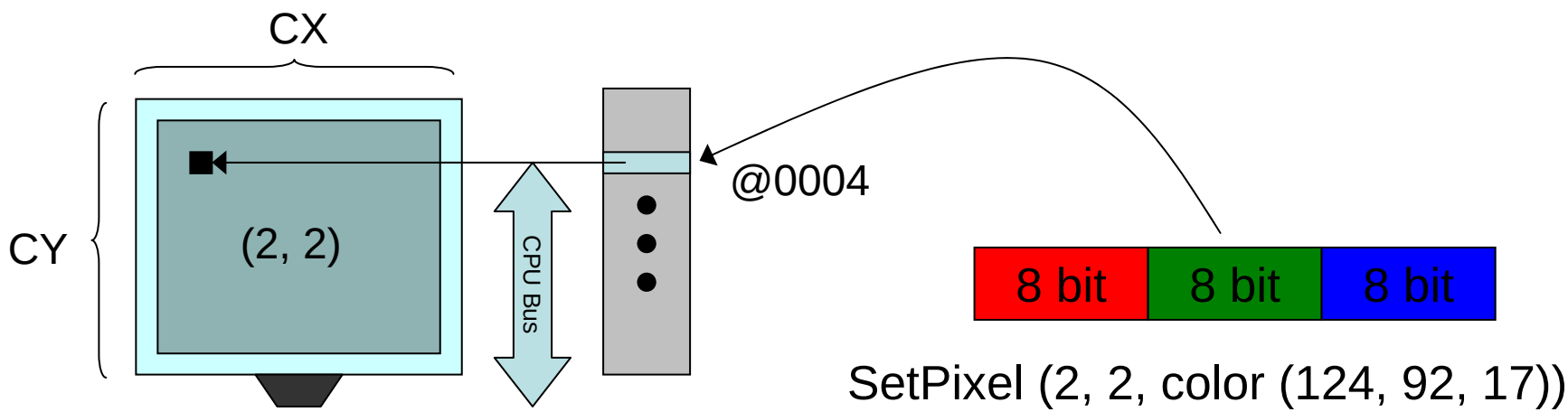


Raster Terminology (Review)

Pixel depth คือขนาดแต่ของหน่วยความจำแต่ละตำแหน่ง (บิต) ซึ่งกำหนดพิสัยของความเข้มของจุดภาพนั้น เช่น 1 บิต หมายถึงจุดภาพนั้น ดับ หรือ สว่าง 8 บิต หมายถึง จุดภาพมีความเข้มแตกต่างกัน 256 ระดับ

Color คือจำนวนสีพื้นฐาน ที่ประกอบขึ้นเป็นจุดภาพนั้นๆ เช่น จอภาพ CRT มี 3 สี ได้แก่ แดง (R) เขียว (G) น้ำเงิน (B) แต่ละสีมี Pixel depth 8 บิต จะแสดงจุดภาพที่มีสีที่สว่างแตกต่างกันได้ 16 ล้านสี หรือ Pixel depth รวม 24 บิต

Resolution คือ จำนวนจุดภาพทั้งหมด ที่ปรากฏบนอุปกรณ์แสดงผล



Application Programmer's Interface

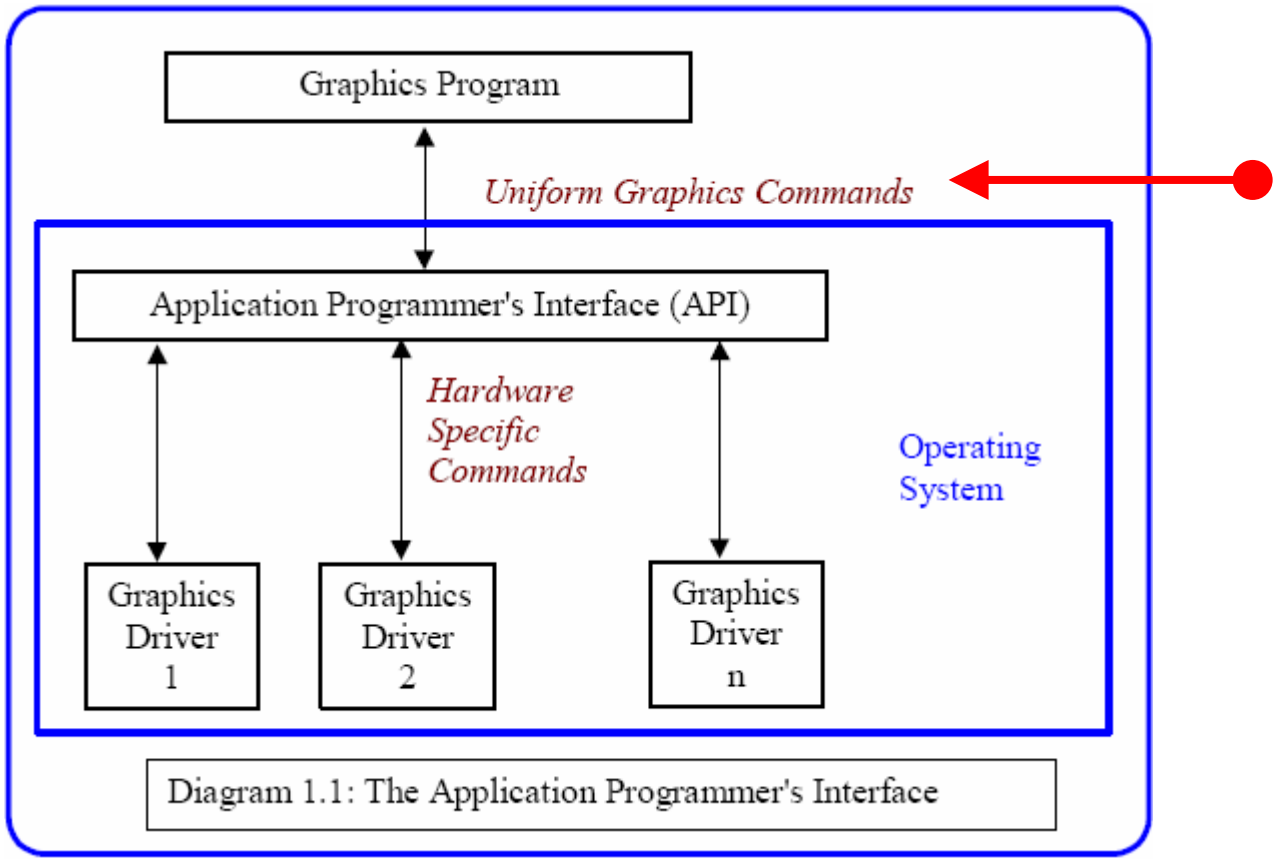


Diagram 1.1: The Application Programmer's Interface

บรรยายครั้งนี้กล่าวถึง Algorithm ในการแปลงคำสั่ง Uniform Graphics เป็น API พื้นฐาน ซึ่งมักอยู่ในรูปของ Library ที่มีพร้อมมากับ Compiler ทั่วไป

COMPUTER GRAPHICS

School of Computer Engineering

Suranaree University of Technology

Lecture 2 Raster Algorithms

Paramate Horkaew

School of Computer Engineering, Institute of Engineering
Suranaree University of Technology

Lecture Outline

- Frame Buffer Class
- Simple Graphic Primitive Algorithms
  - Line Drawing Algorithms (Direct v.s. Integer Arithmetic)
  - Circle Drawing Algorithms
  - Rasterization of Arbitrary Curves
  - Polynomial Curve and Spline Drawing Algorithms
- Filled-Area Primitives
  - Polygon Filling
  - Flood-Fill Algorithm
  - Inside-Outside Tests
- Picture Approximation using Halftone
- Text Generation

Device Independence

เทคนิคหนึ่งที่พยายามทำให้โปรแกรมกราฟิก ขึ้นอยู่กับความสามารถของอุปกรณ์ให้น้อยที่สุด คือ พยายามสร้างฟังก์ชันกราฟิกพื้นฐานขึ้นมาเอง

MFC Library Reference
CDC::Ellipse

Draws an ellipse.

```
BOOL Ellipse(
    int x1,
    int y1,
    int x2,
    int y2
);
BOOL Ellipse(
    LPCRECT lpRect
);
```

ถึงแม้ว่าจะมีผลทำให้ความเร็วของการทำงานลดลง แต่โปรแกรมจะมีความยืดหยุ่น สูง เทคนิคนี้ ใช้กันมาก โดย ผู้เขียนโปรแกรม Micro-controller สำหรับ แสดงผล และ ผู้ผลิตระบบพัฒนาโปรแกรม (Integrated Development Environment : IDE) เช่น ชุดคำสั่ง GDI ของ Microsoft Foundation Class เพื่อ อำนวยความสะดวก ให้กับ ผู้เขียนโปรแกรม ได้เรียกใช้

http://msdn.microsoft.com/developercenters/

Frame Buffer Class

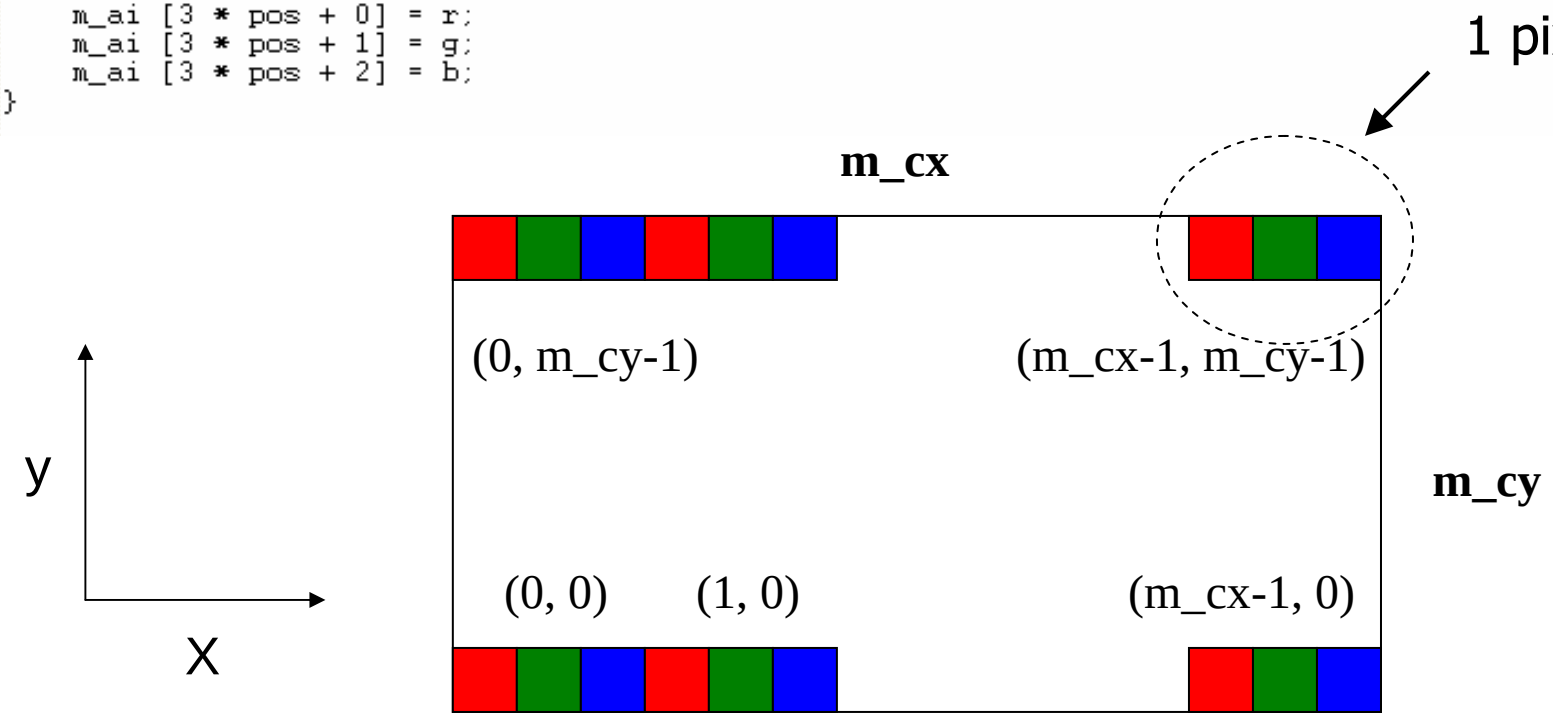
บทนี้เน้นแนวคิด โครงสร้างข้อมูล และขั้นตอนวิธี การสร้างชุดคำสั่งกราฟิกประเภท Device Independent โดยเฉพาะอย่างยิ่ง การนำ array ชนิด byte (unsigned character) มาสร้างหน่วยความจำแสดงผล (Frame Buffer) ซึ่งออกแบบในลักษณะ OOP ดังนี้

```
#include <windows.h>

class cfbuffer
{
public:
     cfbuffer ();
    ~cfbuffer ();

public:
    void    init (long cx, long cy);
    void    clearresource (void);

public:
    void    setpixel (long ix, long iy, unsigned char i);
    void    setpixel (long ix, long iy, unsigned char  r, unsigned char  g, unsigned char  b);

    void    getpixel (long ix, long iy, unsigned char *i);
    void    getpixel (long ix, long iy, unsigned char *r, unsigned char *g, unsigned char *b);

    void    clrscr (void);
    void    display (HDC hdc);

protected:
    unsigned char   *m_ai;      //  rgb components of size 3 * m_cx * m_cy

    long             m_cx;      //  frame buffer size
    long             m_cy;
};
```

Pixel Coordinates

การอ้างถึงตำแหน่งใน frame buffer โดยกำหนดค่าพิกัด (x, y) สามารถทำได้ด้วยการคำนวณ ตำแหน่งใน array ขนาด 1 มิติ ดังนี้

```
void cfbuffer::setpixel (long ix, long iy, unsigned char r, unsigned char g, unsigned char b)
{
    long    pos;

    pos = ix + (m_cy - iy - 1) * m_cx;

    m_ai [3 * pos + 0] = r;
    m_ai [3 * pos + 1] = g;
    m_ai [3 * pos + 2] = b;
}
```



Device Dependence Function

จากการประกาศ class จะเห็นว่ามีเพียงฟังก์ชันเดียว ที่อ้างอิงกับระบบปฏิบัติการ Windows นั้นคือ การเรียกใช้ pointer (handle) ไปยัง Device Context (DC)

```
void cfbuffer::display (HDC hdc)
{
    BITMAPINFO  bmi;

    ::memset (&bmi, 0, sizeof (bmi));

    bmi.bmiHeader.biSize            = sizeof (BITMAPINFOHEADER);
    bmi.bmiHeader.biWidth           = (int) m_cx;
    bmi.bmiHeader.biHeight          = (int) m_cy;
    bmi.bmiHeader.biPlanes          = 1;
    bmi.bmiHeader.biBitCount        = 24;
    bmi.bmiHeader.biCompression     = BI_RGB;
    bmi.bmiHeader.biSizeImage       = m_cx * m_cy * 3;
    bmi.bmiHeader.biXPelsPerMeter   = 0;
    bmi.bmiHeader.biYPelsPerMeter   = 0;
    bmi.bmiHeader.biClrUsed         = 0;
    bmi.bmiHeader.biClrImportant    = 0;

    ::SetDIBitsToDevice (hdc, 0, 0, m_cx, m_cy, 0, 0, 0, m_cy,
                         m_ai, &bmi, DIB_RGB_COLORS);
}
```

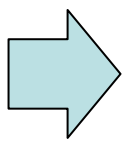
การส่งข้อมูลใน Frame Buffer ออกทางจอภาพทำได้โดยเรียกฟังก์ชันของ OS

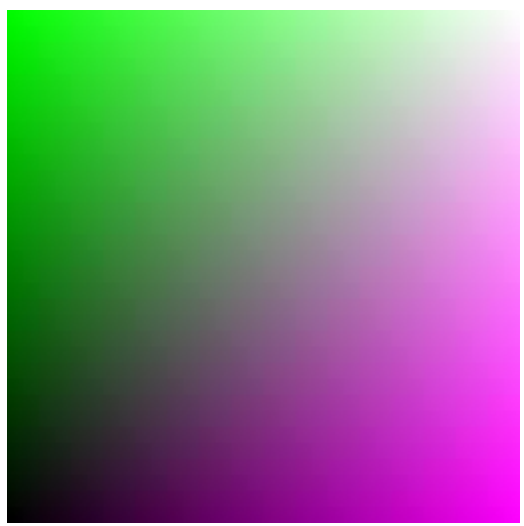
Implementation Example

การเรียกใช้ class cfbuffer สามารถทำได้ดังนี้

1) กำหนดขนาดของหน่วยความจำ (init)
2) ตั้งค่าปริยายให้กับหน่วยความจำเป็นศูนย์ (clrscr)
3) กำหนดค่าสีของแต่ละจุด (setpixel)
4) แสดงผลออกหน้าจอ (display)

```
void CMainFrame::TestFrameBuffer (HWND hwnd)
{
    HDC         hdc;
    cfbuffer    fbuffer;
    long        x, y;

    hdc     = ::GetDC (hwnd);

    fbuffer.init (256, 256);
    fbuffer.clrscr ();

    for (y = 0; y < 256; y ++)
    {
        for (x = 0; x < 256; x ++)
        {
            fbuffer.setpixel (x, y, x, 255-y, x);
        }
    }

    fbuffer.display (hdc);

    ::ReleaseDC (hwnd, hdc);
}
```





Line Drawing Algorithms

การวาดเส้นตรงบนจอภาพสามารถแสดงได้ โดยเริ่มจากการศึกษาสมการเส้นตรง

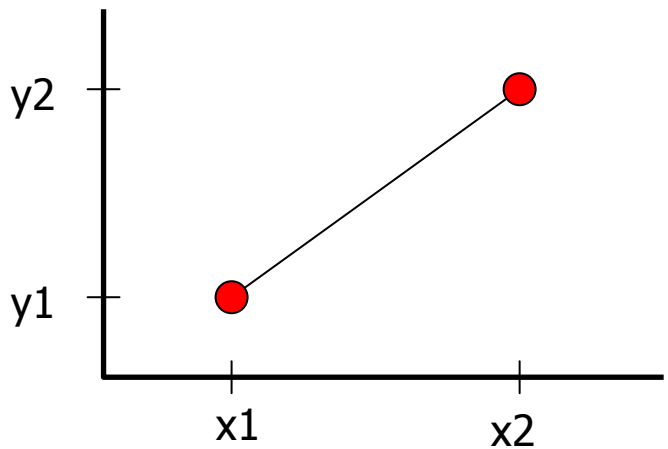
$$y = m \cdot x + b$$

โดยที่ m และ b คือ ความชัน และ จุดตัดแกน y ของเส้นตรง ตามลำดับ

ถ้ากำหนด จุดปลายสองจุด คือ (x_1, y_1) และ (x_2, y_2) ตัวแปรสมการเขียนได้เป็น

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - m \cdot x_1$$



Digital Differential Analyzer (DDA)

จากการสังเกตพบว่า ความแตกต่างตามแกน x ($\Delta x = x2 - x1$) และ แกน y ($\Delta y = y2 - y1$) จะสัมพันธ์ กับค่าความชัน m ตามสมการ

$$\Delta y = m \cdot \Delta x \text{ และ } \Delta x = \Delta y / m$$

เนื่องจากใน Computer Graphics เราจะพิจารณา Frame Buffer ซึ่งมีระบบปริภูมิแบบ Discrete นั่นคือ Δ จะมีค่าเป็นจำนวนเต็มหน่วยเท่านั้น (pixels) ดังนั้น แบ่งการพิจารณาเป็น 2 กรณี **เมื่อ m และ Δx มีค่าเป็นบวก**

กรณีที่ 1) m น้อยกว่าหรือเท่ากับ 1 **กรณีที่ 2)** m มากกว่า 1

$$y_{k+1} = y_k + m \qquad x_{k+1} = x_k + \frac{1}{m}$$

ถ้า Δx มีค่าเป็นลบก็เพียงแต่เปลี่ยนเครื่องหมายหน้า m จาก + เป็น -

Digital Differential Analyzer (DDA)

ในทำนองเดียวกัน กรณีที่ m มีค่าเป็นลบ จะพิจารณาจากค่าสัมบูรณ์ของ m

กรณีที่ 1) |m| น้อยกว่าหรือเท่ากับ 1 **กรณีที่ 2)** |m| มากกว่า 1

$$y_{k+1} = y_k + m \qquad x_{k+1} = x_k + \frac{1}{m}$$

DDA Algorithm จะเร็วกว่าทำการคำนวณจากสมการตรงๆ โดยการตัดกระบวนการคูณ (พิกัด x กับความชัน m) ทิ้ง แล้วแทนที่ด้วยการบวก จำนวนจริง

อย่างไรก็ดี ข้อควรระวังคือ หากเส้นตรงมีความยาวมาก ความผิดพลาดจะสะสม จนกระทั่งเบี่ยงเบนไปจาก เส้นตรงที่ต้องการอย่างเด่นชัด อาจแก้ไขโดยการบวกค่าสะสมเป็นจำนวนจริง แล้วปัดเศษเป็นจำนวนเต็ม ก่อนจะวาดจุดภาพ ทว่าวิธีนี้จะใช้เวลา CPU ค่อนข้างมาก

Digital Differential Analyzer (DDA)

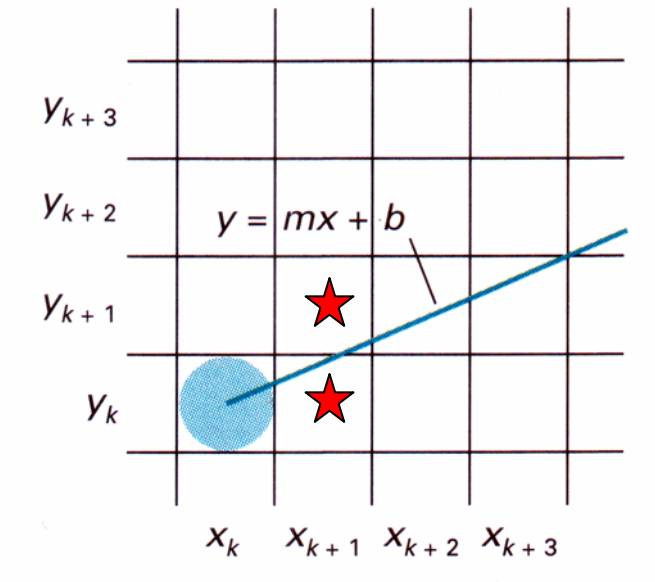
DDA Algorithm สามารถเขียนด้วย C++ บน class cfbuffer ได้ดังนี้

```
void cfbuffer::lineDDA (int x1, int y1, int x2, int y2)
{
    int     dx, dy, steps, k;
    double  xincrement, yincrement;
    double  x, y;

    dx  = x2 - x1;
    dy  = y2 - y1;

    steps = (abs (dx) > abs (dy)) ? abs (dx) : abs (dy);

    xincrement  = (double) dx / steps;
    yincrement  = (double) dy / steps;

    x   = x1;
    y   = y1;

    setpixel ((int) (x + 0.5), (int) (y + 0.5), 0, 0, 0);

    for (k = 0; k <= steps; k ++)
    {
        x   = x + xincrement;
        y   = y + xincrement;

        setpixel ((int) (x + 0.5), (int) (y + 0.5), 0, 0, 0);
    }
}
```

Bresenham's Line Algorithm

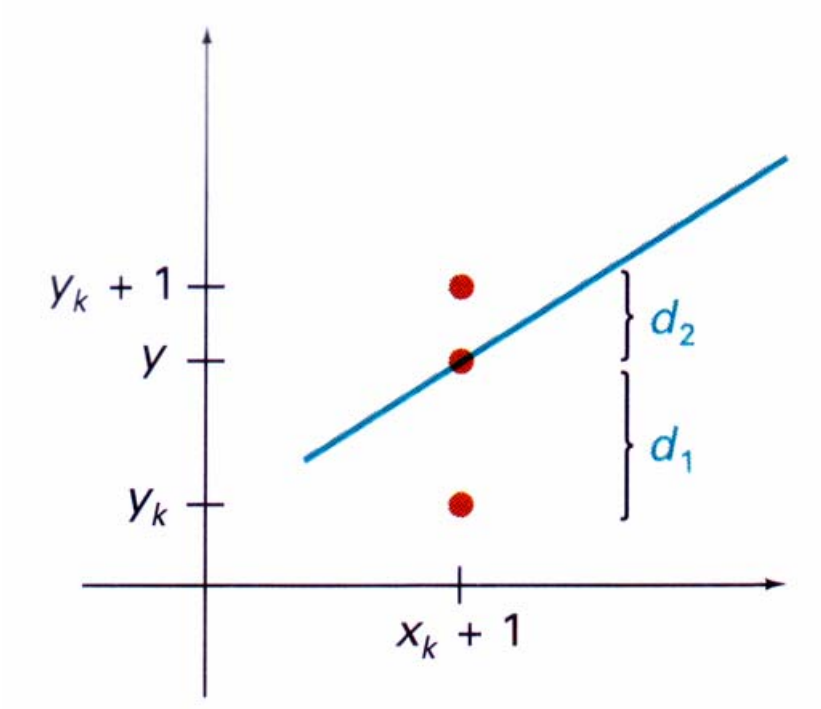
เป็น algorithm เพื่อใช้สำหรับวาดเส้นตรง ที่มีประสิทธิภาพ และเที่ยงตรง คิดค้น โดย Bresenham ในปี 1961 ซึ่งคำนวณ โดยใช้เพียงเลขจำนวนเต็ม อีกทั้งยัง สามารถนำไปประยุกต์ ใช้สำหรับวาดเส้นโค้ง ต่างๆได้อีกด้วย

เพื่อความสะดวก ในที่นี้จะสมมติให้ ความชัน มีค่าเป็นจำนวนบวก ($m<1$) โดยเริ่ม พิจารณาจากจุดปลายทางซ้ายมือ จากรูปจะเห็นว่าเมื่อวาดจุดภาพ ที่ตำแหน่งที่ (x_k, y_k) แล้ว ในการวนรอบครั้งต่อไปต้องการ หา ตำแหน่งของจุดที่ k+1 ซึ่ง algorithm จะตัดสินเลือก ระหว่างจุด (x_k+1, y_k) และ (x_k+1, y_{k+1})



Discrete and Exact Lines

ที่ตำแหน่งพิกัดที่ x_k+1 กำหนดให้ d1 และ d2 เป็นระยะห่างในพิกัดแนวดิ่งจากเส้นตรงจริง $y = m(x_k+1) + b$ ทางด้านล่าง และ ด้านบน ตามลำดับ



$$d_1 = y - y_k$$
$$= m(x_k + 1) + b - y_k$$
$$d_2 = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

ความแตกต่างของระยะทั้งสองคือ

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

Decision Parameter p_k

กำหนดให้ m = $\Delta y/\Delta x$ และ นิยามตัวแปลสำหรับเลือกตำแหน่ง y ดังนี้

$$p_k = \Delta x(d_1 - d_2)$$
$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$
$$c = 2\Delta y + \Delta x(2b - 1)$$

ในที่นี้เนื่องจาก Δx มีค่าเป็นบวก p_k จะมีเครื่องหมายเดียวกับ d_1-d_2 นั่นคือ

- ถ้า p_k เป็นลบแสดงว่า $d_1 < d_2$ หรือ y_k ใกล้เส้นจริงมากกว่า y_k+1 *เลือก* y_k
- ถ้า p_k เป็นบวกแสดงว่า $d_1 > d_2$ หรือ y_k+1 ใกล้เส้นจริงมากกว่า y_k *เลือก* y_k+1

Integer Arithmetic

ปรับสมการให้อยู่ในรูปของตัวแปล p เป็นฟังก์ชันของ k เท่านั้น

$$p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x\left(y_{k+1} - y_k\right)$$

$$p_0 = 2\Delta y - 2\Delta x$$

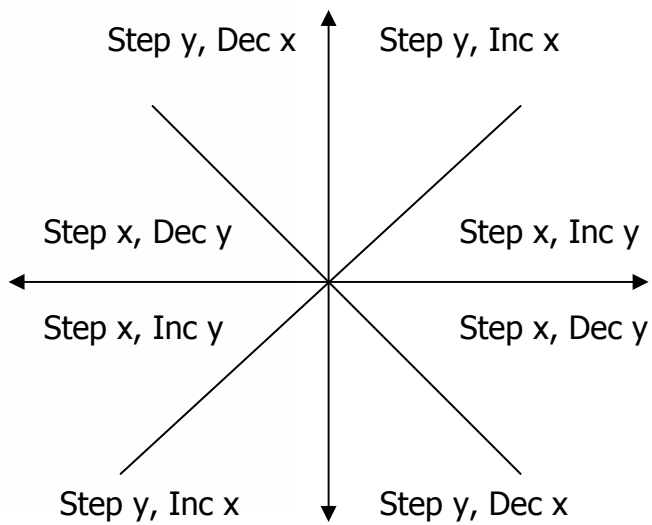
สังเกตว่า $y_{k+1} - y_k$ จะมีค่าเป็น 0 หรือ 1 ขึ้นอยู่กับเครื่องหมายของ pk

ค่า Δy, Δx และ $\Delta y - \Delta x$ เป็นจำนวนเต็ม ค่าคงที่ซึ่งคำนวณ เพียงครั้งเดียว

Bresenham's Implementation

ในกรณีที่ 0 < m < 1 algorithm สามารถเขียนด้วย C++ ได้ดังนี้

```
void cfbuffer::lineBres (int x1, int y1, int x2, int y2)
{
    int     dx, dy, x, y, xEnd, p;

    dx   = abs (x2 - x1);
    dy   = abs (y2 - y1);
    p    = 2*dy - dx;

    //  determine which point to use as start, which as end
    x       = (x1 > x2) ? x2 : x1;
    y       = (x1 > x2) ? y2 : y1;
    xEnd    = (x1 > x2) ? x1 : x2;

    setpixel (x, y, 0, 0, 0);

    while (x < xEnd)
    {
        x    = x + 1;
        if (p < 0)
            p = p + 2*dy;
        else
        {
            y    = y + 1;
            p    = p + 2*(dy - dx);
        }
        setpixel (x, y, 0, 0, 0);
    }
}
```



Homework

กำหนดให้จุดปลายทั้งสองจุดของเส้นตรงเป็น (20, 10) และ (30, 16) จง

1) คำนวณความชันของเส้นตรง
2) หาสมการของเส้นตรง
3) ใช้ DDA Algorithm ในการหาพิกัดจุดลำดับที่ k
4) หาค่าตัวแปรตัดสินใจที่ตำแหน่งเริ่มต้น (p_0)
5) ใช้ Bresenham's Algorithm ในการหาค่าตัวแปรตัดสินใจ (p_k) และพิกัดจุดลำดับที่ k

กำหนดให้ k = 0 ถึง 9 แจกจางค่าผลลัพธ์ที่ได้ในตาราง โดยใช้ตารางของผลจาก ข้อ 3 และ ข้อ 5 แยกกัน

(ส่งวันพฤหัสหน้า)

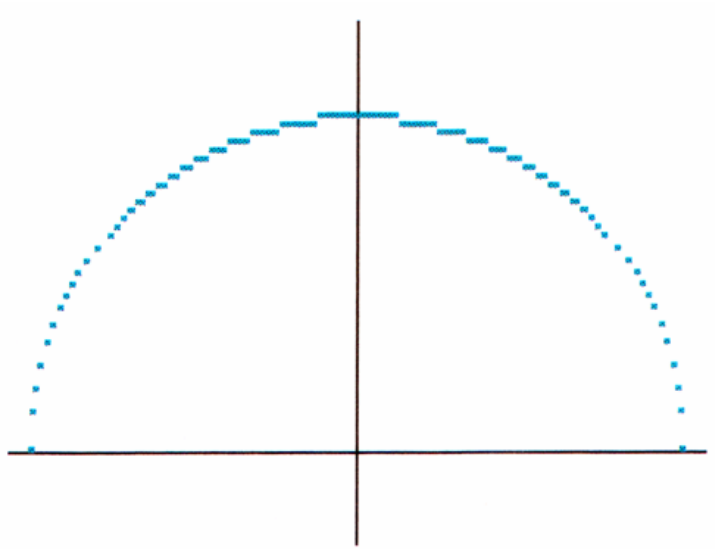
Drawing A Circle

วงกลมที่มีจุดศูนย์กลางที่ (x_c, y_c) และรัศมี r มีสมการ ดังนี้

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

วิธีรวาดวงกลมที่กำหนดอย่างง่ายที่สุดคือ กำหนดค่า x แล้ววาดจุดที่ตำแหน่ง y

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

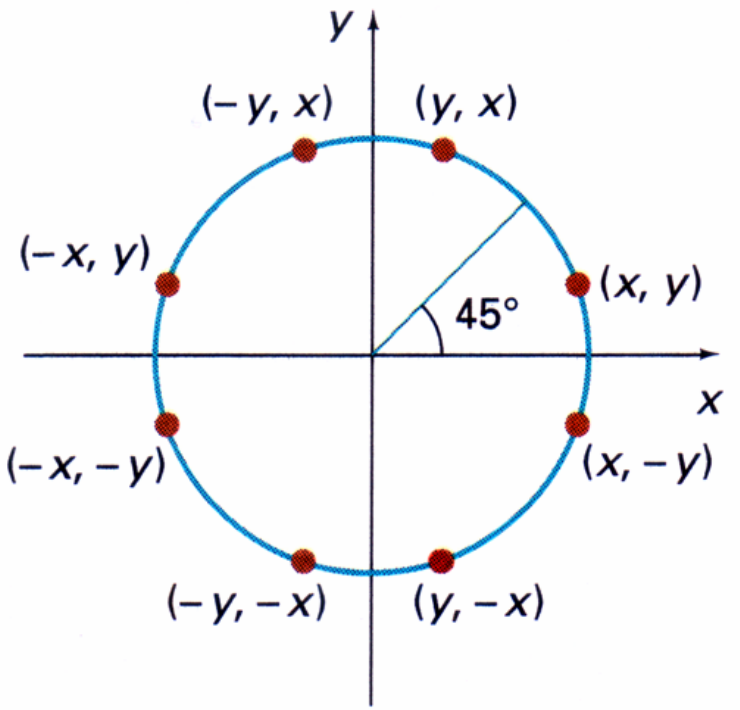


ข้อเสียวิธีนี้คือ

- การคำนวณที่ใช้เวลาของ CPU มาก
- ระยะห่างระหว่างจุดไม่สม่ำเสมอ การเลื่อนค่า x ที่ละน้อย ไม่แก้ปัญหา เพราะว่าทำให้ใช้เวลามากขึ้นไปอีก

Midpoint Circle Algorithm

เนื่องจากสมการวงกลมแบบเต็มรูป มีความซับซ้อน เราอาจจะเลื่อนวงกลมมาที่จุดกำเนิดก่อน $(x_c, y_c) = (0, 0)$ แล้วหลังจากนั้นจึงเลื่อนไปยังจุดที่ถูกต้อง โดยการบวกจุดภาพที่ได้ด้วยค่าพิกัด (x_c, y_c) นอกจากนี้วงกลมยังมีความสมมาตรเทียบกับแกน $x = \pm y$ และแกน x และ y ดังรูป



ดังนั้นเราจึงสามารถคำนวณเพียงแค่ในส่วนที่ $x = 0$ ถึง $x = y$ แล้ว สะท้อนพิกัดจุดไปยัง octant ต่างๆ

ด้วยคุณสมบัติของวงกลม ค่าความชันใน octant นี้มีค่าตั้งแต่ 0 ถึง -1 เราจึงสามารถใช้ Bresenham's Algorithm ที่เพิ่มค่าในแกน x ทีละ 1 จุดได้

Implicit Circle Function

นิยามฟังก์ชัน (implicit) ของวงกลมดังนี้

$$f_c(x, y) = x^2 + y^2 - r^2$$

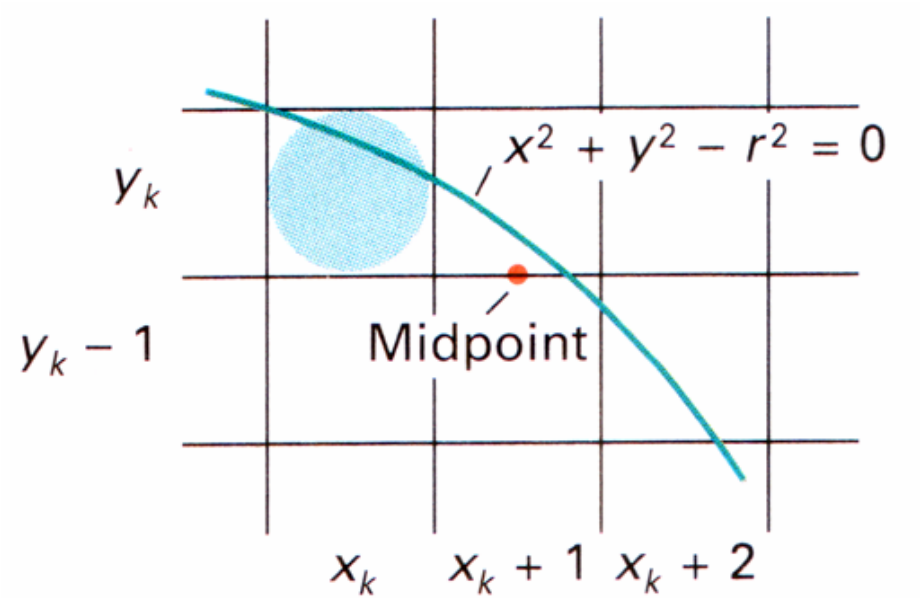
ค่าของฟังก์ชันที่ (x, y) สามารถแยกพิจารณาได้ 3 กรณี

- ค่าเป็นลบ หมายถึง (x, y) อยู่ภายในวงกลม
- ค่าเป็นศูนย์ หมายถึง (x, y) อยู่บนเส้นรอบวงของวงกลม
- ค่าเป็นบวก หมายถึง (x, y) อยู่ภายนอกวงกลม

ซึ่งสามารถใช้แทนตัวแปรตัดสินใจในทำนองเดียวกับ Bresenham's Algorithm

Deriving Decision Parameter

สมมติว่าเราได้วาดจุด (x_k, y_k) ไปแล้ว ขั้นต่อไปคือพิจารณาเลือกระหว่างจุด y_k และ y_k-1 สำหรับพิกัด x_k+1 ซึ่งทำได้โดยคำนวณค่าตัวแปรตัดสินใจที่ midpoint



$$p_k = f_c\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

- ค่าเป็นลบ หมายถึง midpoint อยู่ภายในวงกลม
- ค่าเป็นบวก หมายถึง midpoint อยู่ภายนอกวงกลม

Successive Decision Parameter

ตัวแปลตัดสินใจ ณ รอบที่ k+1 สามารถคำนวณได้ในทำนองเดียวกัน

$$p_{k+1} = f_c\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$

$$= \left(x_k+1+1\right)^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = p_k + 2\left(x_k+1\right) + \left(y_{k+1}^2 - y_k^2\right) - \left(y_{k+1}-y_k\right) + 1$$

สังเกตว่า y_{k+1} จะมีค่าเป็น y_k หรือ y_k-1 ขึ้นอยู่กับเครื่องหมายของ p_k

$$p_{k+1} = \begin{cases} p_k + 2x_{k+1} + 1 & p_k < 0 \\ p_k + 2x_{k+1} + 1 - 2y_{k+1} & p_k \geq 0 \end{cases}$$

Algorithm Summary

1) เลื่อนวงกลมไปที่จุดกำเนิด คำนวณตัวแปรตัดสินใจเริ่มต้น p_0

2) คำนวณพิกัดที่ (x_{k+1}, y_{k+1}) โดยใช้ตัวแปรตัดสินใจที่ p_k โดยที่ $x_{k+1} = x_k+1$ และ $y_{k+1} = y_k$ หรือ y_k-1

3) สะท้อนจุดที่คำนวณได้ไปยังอีก 7 octants ที่เหลือ

4) เลื่อนวงกลมไปที่จุดศูนย์กลางที่ต้องการ ทำซ้ำขั้นตอนที่ 2-4 ขณะที่ x < y

$$(x_0, y_0) = (0, r)$$

$$p_0 = f_c\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2 = \frac{5}{4} - r$$

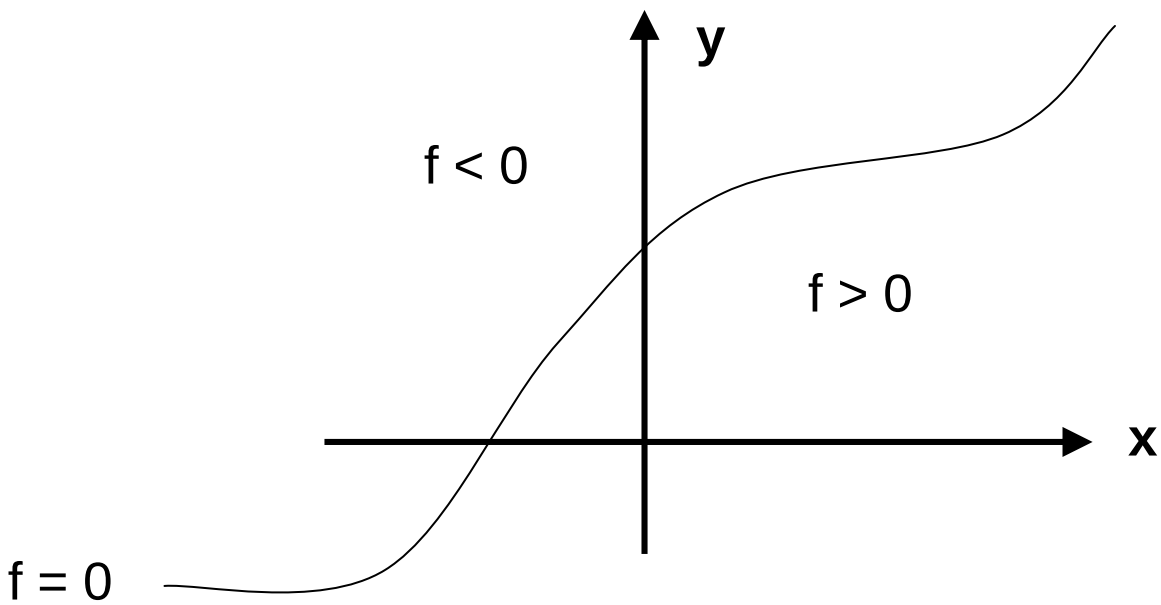
$$\approx 1 - r \quad , r \in N$$

แบบฝึกหัด จงเขียนฟังก์ชัน C++ จากคำอธิบายขั้นตอนวิธี midpoint circle drawing algorithm

Rasterization of Curves

จากตัวอย่างการใช้ตัวแปรตัดสินใจของ Bresenham สำหรับเส้นตรง และขั้นตอนวิธี midpoint ของ วงกลม เราสามารถขยายผล เพื่อนำมาใช้สร้าง เส้นโค้งใดๆ ได้

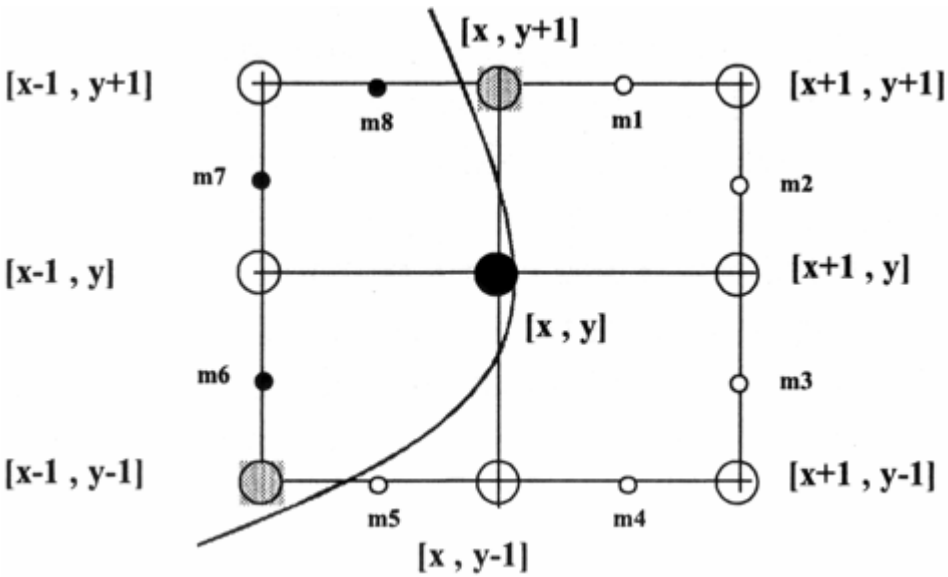
นิยามเส้นโค้งในรูปของ implicit function กล่าวคือ f (x, y) = 0 จะเป็นการแบ่งปริภูมิ Cartesian ออกเป็นสองระนาบ ดังรูป



Rasterization of Curves

การหาขั้นตอนวิธีสร้างเส้นโค้งใดๆ สามารถทำได้ดังนี้

- หาสมการ ค่าตัวแปรตัดสินใจในรูปของพิกัด x และ y (p_k)
- หาสมการ recursive $p_{k+1} = p_k + g(x_k, y_k, x_k \pm 1, y_k \pm 1)$
- ทั้งนี้ต้องระวังในเรื่องของ เครื่องหมาย เนื่องค่าความชันของเส้นโค้งเป็นไปได้ทั้ง + และ - และค่าสัมบูรณ์ อาจจะมีค่ามากกว่าหรือน้อยกว่า 1 ก็ได้



อีกวิธีหนึ่งที่ง่ายกว่าคือ การหาตำแหน่งจุด ที่ฟังก์ชันของเส้นโค้งเปลี่ยนเครื่องหมาย แต่ว่า การทำงานจะช้ากว่ามาก ดังรูป

Conclusion

- Frame Buffer Class
- Simple Graphic Primitive Algorithms
  - Line Drawing Algorithms (Direct v.s. Integer Arithmetic)
  - Circle Drawing Algorithms
  - Rasterization of Arbitrary Curves
  - Polynomial Curve and Spline Drawing Algorithms
- Filled-Area Primitives
  - Polygon Filling
  - Flood-Fill Algorithm
  - Inside-Outside Tests
- Picture Approximation using Halftone
- Text Generation