COMPUTER GRAPHICS

School of Computer Engineering

Suranaree University of Technology

Lecture 2 Review (Part I)

Paramate Horkaew

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

Frame Buffer Class

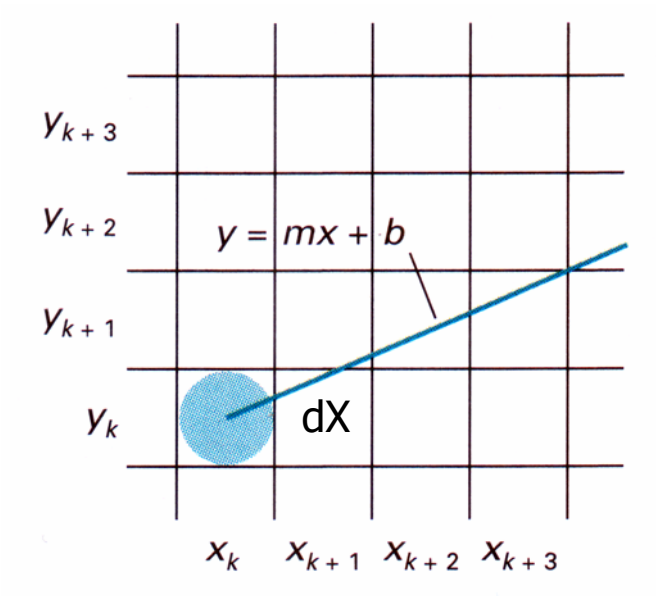
บทนี้เน้นแนวคิด โครงสร้างข้อมูล และขั้นตอนวิธี การสร้างชุดคำสั่งกราฟิกประเภท Device Independent โดยเฉพาะอย่างยิ่ง การนำ array ชนิด byte (unsigned character) มาสร้างหน่วยความจำแสดงผล (Frame Buffer) ซึ่งออกแบบในลักษณะ OOP ดังนี้

```
#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;
};
```

Line Drawing Algorithms

วิธีการวาดเส้นตรงแบบแรก คือ Digital Differential Analyzer (DDA)

กรณีที่ 1) m น้อยกว่าหรือเท่ากับ 1



เลื่อนไปตามแนวแกน x ครั้งละ 1 จุดภาพ (dX = 1) เนื่องจากว่า:

จากสมการเส้นตรงจะได้ว่า dY = m · dX

เมื่อ dX = 1 และ 0 <= m <= 1 จะได้ว่า

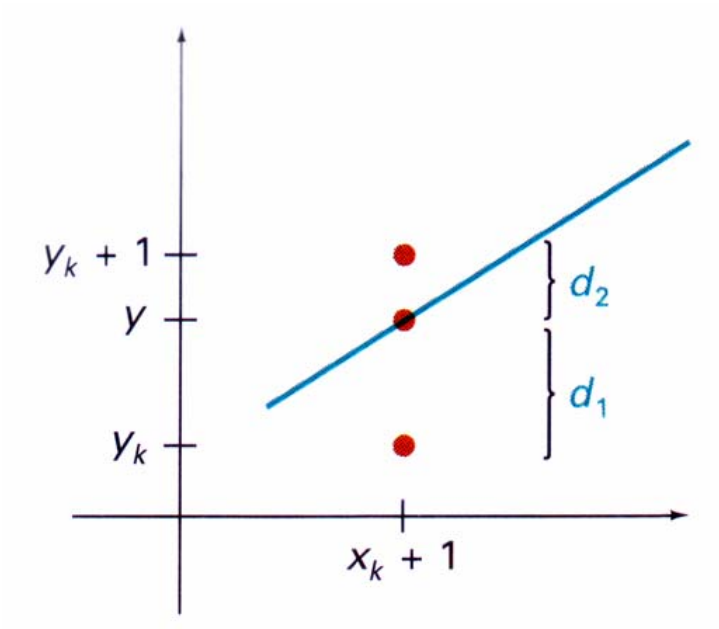
dY <= 1 หมายความว่า ความแตกต่างของระยะห่างระหว่างจุดตามแนวตั้งมีค่าไม่เกิน 1 จุดภาพ **(เส้นไม่ขาดตอน)**

กรณีอื่นๆ ของค่า m และ quadrant ของเส้นตรง ก็พิจารณาในทำนองเดียวกัน

Line Drawing Algorithms

วิธีที่สอง คือ Bresenham's Algorithm ซึ่งใช้หลักการสร้างตัวแปรตัดสินใจ

ตัวแปรตัดสินใจอยู่ในรูปของ $p_k = d_1 - d_2$



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

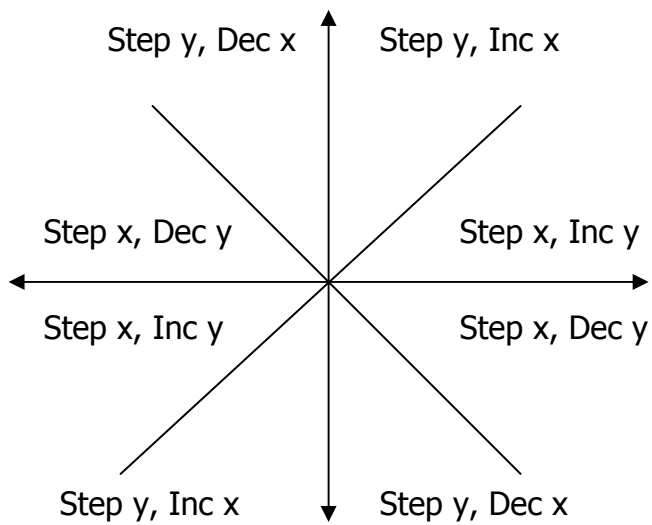
$$p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$
$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$
$$p_0 = 2\Delta y - 2\Delta x$$

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

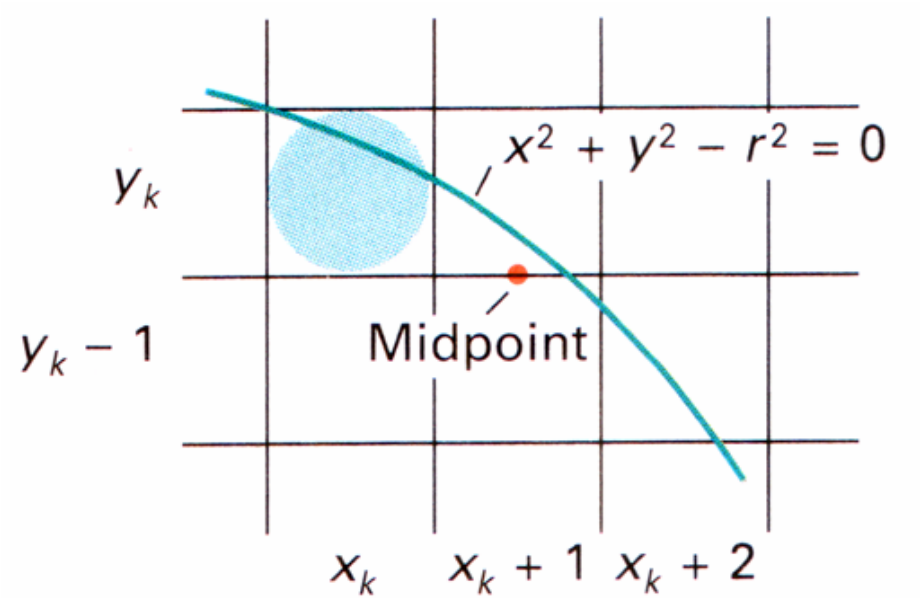
1) คำนวณความชันของเส้นตรง
2) หาสมการของเส้นตรง
3) ใช้ DDA Algorithm ในการหาพิกัดจุดลำดับที่ k
4) หาค่าตัวแปรตัดสินใจที่ตำแหน่งเริ่มต้น (p_0)
5) ใช้ Bresenham's Algorithm ในการหาค่าตัวแปรตัดสินใจ (p_k) และพิกัดจุดลำดับที่ k

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

(เฉลยการบ้าน)

Midpoint Circle Algorithm

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

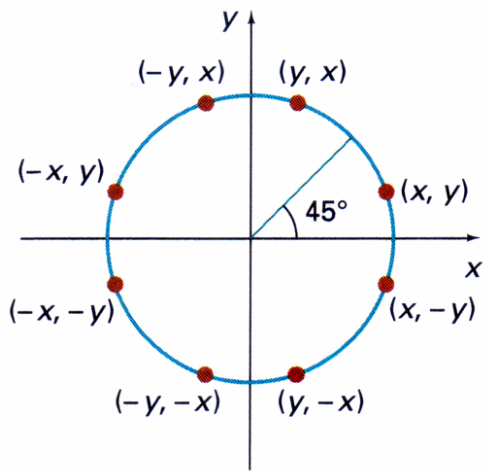
Recursive Decision Parameter

ตัวแปรตัดสินใจ 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

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

Suranaree University
of Technology

COMPUTER GRAPHICS
School of Computer Engineering

Lecture 2 Raster Algorithms (Part II)

Paramate Horkaew

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

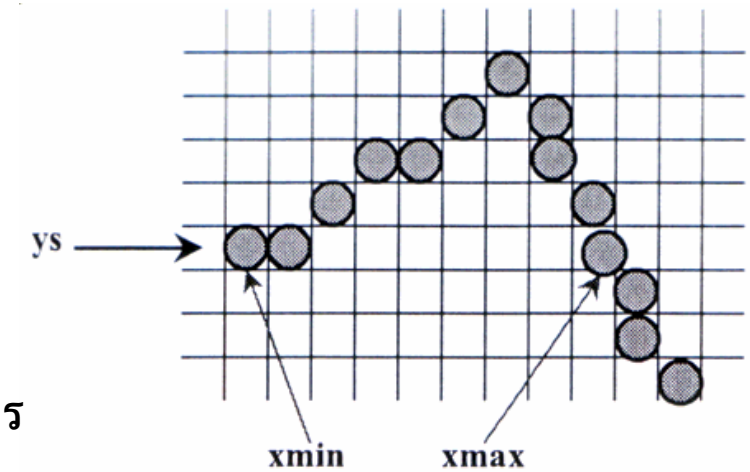
Lecture Outline

- 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
- Output Primitive Attributes
- Picture Approximation using Halftone

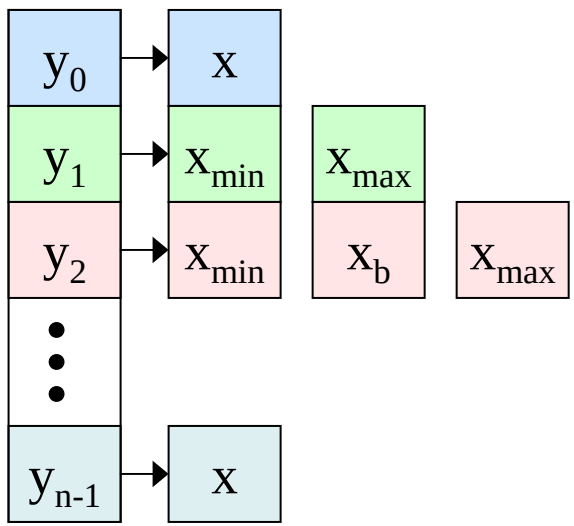
Filled Area Primitives

Simple Polygon Filling Algorithm

วิธีการที่ง่ายที่สุด ของการระบายรูปหลายเหลี่ยม คือ Line Scan Algorithm



1. วาดรูป Polygon เชื่อมต่อจุดโดยใช้ เทคนิค การวาดเส้นตรง
2. จัดเก็บจุดที่ประกอบเป็นขอบของ Polygon (หรือ Edge Pixels)
3. เรียงจุดดังกล่าว ใน Array ตามลำดับพิกัดในแนวตั้ง (y) จากน้อยไปมาก
4. สำหรับพิกัด y แต่ละค่า ถ้ามี จุดเดียวแสดงว่าเป็นจุดยอดบนสุด หรือ ล่างสุด ให้ระบายสีจุดนั้น
5. มิฉะนั้น ระบายสีระหว่างจุดที่มี x มากและ น้อยที่สุด ดังรูป

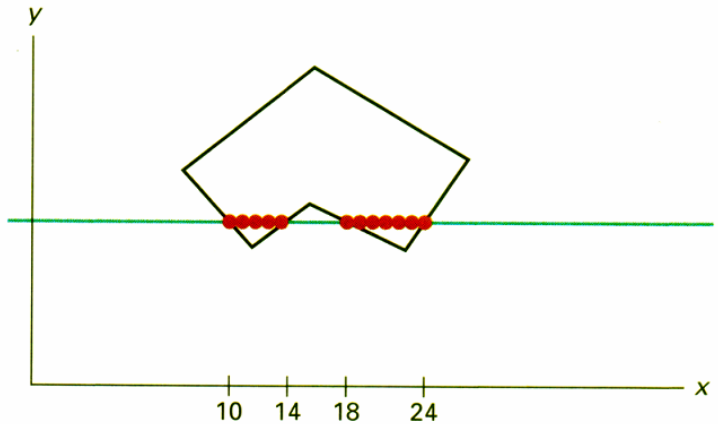


Concave Polygons

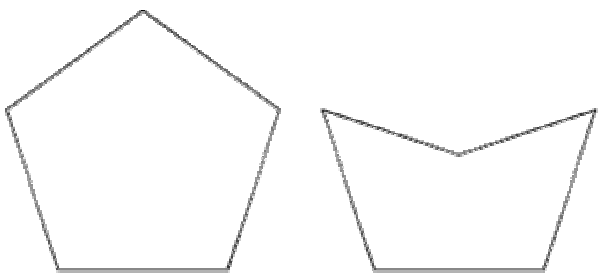
นิยาม

Polygon เรียกว่าเป็น concave ก็ต่อเมื่อ มุมภายใน อย่างน้อย 1 มุม มีขนาด มากกว่า 180 องศา (π เรเดียน)

Concave polygon จะต้องมีอย่างน้อย 4 ด้านเสมอ (ไม่มีรูปสามเหลี่ยมใดๆ เป็น concave)



ขั้นตอนวิธีการระบาย Concave Polygon ขยายผลจากการระบาย Convex Polygon ดังนี้:



convex polygon *concave polygon*

ระบุจุดขอบ (edge pixels)
เรียง edge pixels ตามลำดับของ y

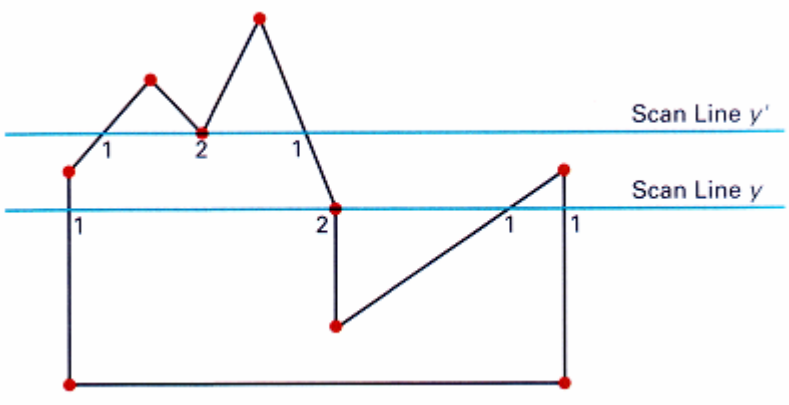
for (จุดที่มีค่า y เท่ากัน) do
  เรียงลำดับตามค่าของ x
  ระบายระยะ เว้นระยะ ตามแนวนอน
end for

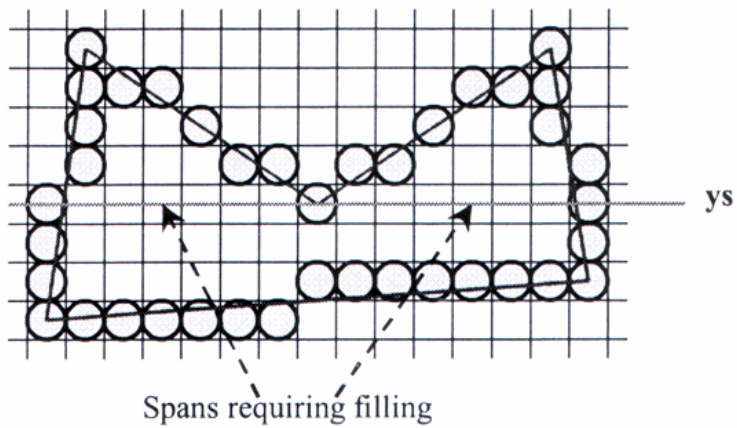
Exceptions (I)

กรณีที่ 1 scan line ตัดผ่านจุดยอดของ polygon **พอดี** ที่จุดหมายเลข 2 มีความเป็นไปได้ อยู่ 2 รูปแบบ

1.1 จุดที่ scan line ตัดผ่านแบ่ง polygon ออกเป็นสองช่วง (span) ซึ่งอยู่ภายใน polygon ทั้งสองช่วง (scan line y')

1.2 จุดที่ scan line ตัดผ่านแบ่ง polygon ออกเป็นสองช่วง แต่มีเพียงช่วงเดียวที่อยู่ภายใน polygon (scan line y)

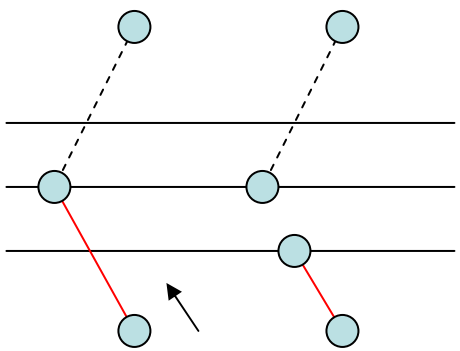


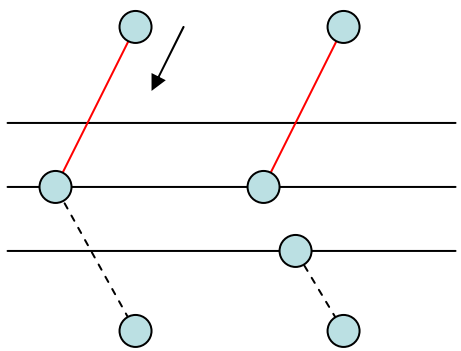


Identifying Sharing Vertices

การระบุว่า scan line นั้นจัดอยู่ในรูปแบบใด สามารถทำได้ ดังต่อไปนี้

- เดิน (trace) ตามเส้นรอบ polygon ในทิศทาง ทวนเข็ม หรือ ตามเข็ม นาฬิกา ก็ได้
- สำหรับ จุดยอด ของ polygon สังเกตลักษณะของ การเปลี่ยนแปลง ค่า y
  - ถ้าค่า y ลดลงตลอด (หรือเพิ่มขึ้นตลอด) เมื่อเทียบ จุดยอดเป็น จุดศูนย์กลาง (monotonic) แสดงว่า จุดตัดนั้น แบ่งออกเป็นส่วนคือที่อยู่ภายใน ก้บ ภายนอก polygon (กรณี 1.2) ให้ตัดเส้นขอบลง 1 scan line
  - ถ้าค่า y เปลี่ยนแปลง ไปในทิศทางตรงกันข้าม แสดงว่า จุดตัดนั้นแบ่ง scan line ออกเป็น 2 ส่วน ภายใน polygon (กรณี 1.1) เพิ่มจุดตัดซ้ำ 2 จุด





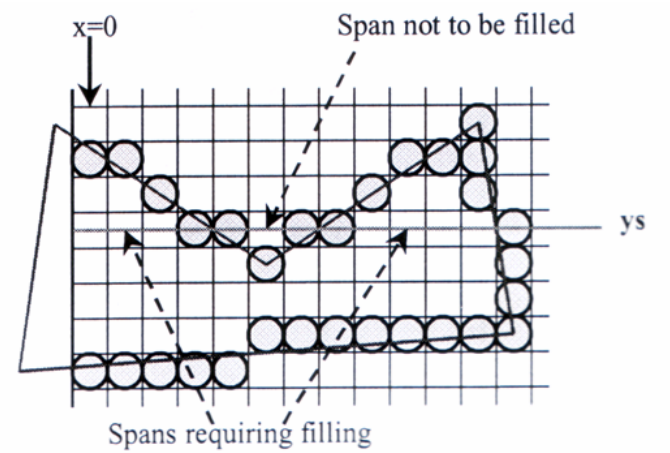
Scan line y+1
Scan line y
Scan line y-1

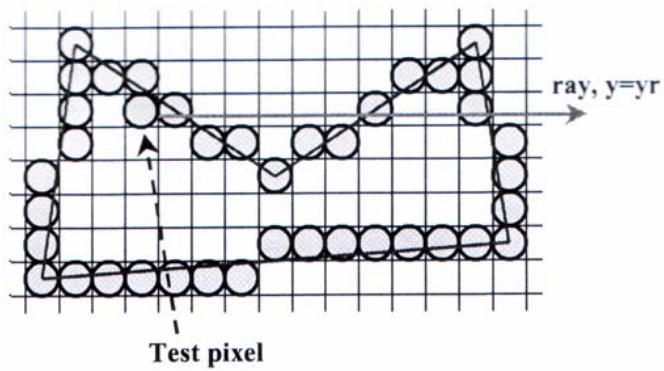
Exceptions (II)

กรณีที่ 2 บางส่วนของ Polygon ถูกตัดทิ้ง เนื่องจาก เกินมาจากเนื้อที่ แสดงผล ซึ่ง สามารถแก้ไขได้ 2 วิธี

1.1 คำนวณ algorithm สำหรับวาดเส้นตรง ทุกครั้งที่มีการทดสอบ Polygon Fill เพื่อให้สามารถ อ้างอิงส่วนที่(จะ)ถูกตัดทิ้งได้ แต่เมื่อทำการวาดผลลัพธ์ ให้ วาดเฉพาะส่วนที่อยู่ภายใน Raster (Frame) Buffer เท่านั้น (**ยุ่งยาก**)

1.2 ใช้วิธี Containment Test (นับจำนวนจุดที่เวคเตอร์ จากจุดที่กำหนดตัดกับ polygon: ODD = inside, EVEN = outside)

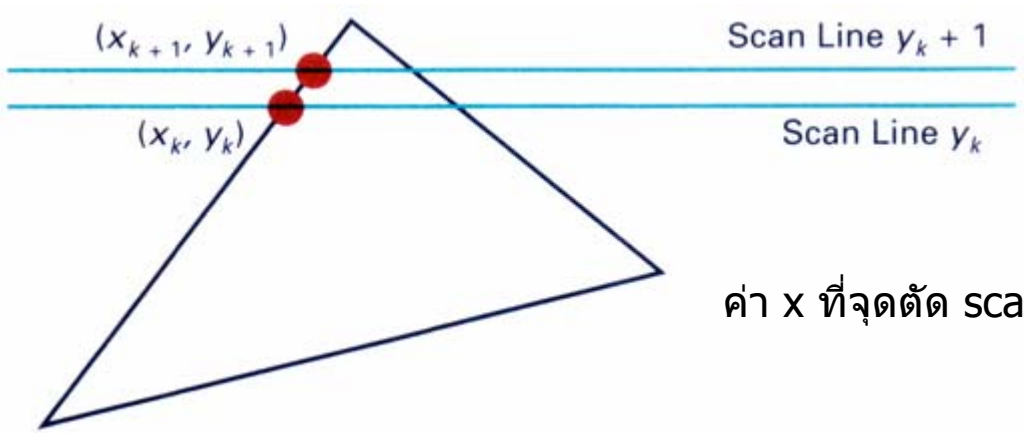




Coherent Polygon Processing

เราสามารถใช้ประโยชน์ ความเกี่ยวเนื่องกันขององค์ประกอบ polygon สำหรับ scan line ที่ติดกัน มาใช้เพิ่ม ประสิทธิภาพ ในการประมวลผลได้

ตัวอย่างเช่น ในกรณีนี้ สังเกตว่า ความชัน m ของเส้นขอบ ของ polygon (edge) มีค่าคงที่ ระหว่าง scan line ที่ติดกัน (ดังรูป) ซึ่งสามารถเขียนความสัมพันธ์ได้ว่า



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

since $y_{k+1} - y_k = 1$

ค่า x ที่จุดตัด scan line ถัดไป $x_{k+1} = x_k + \frac{1}{m}$

เขียนในรูปของสมการเส้นขอบ $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$

เราสามารถใช้ประโยชน์

An Efficient Polygon Algorithm

จากสมการหาพิกัด x ของจุดตัดระหว่าง scan line ที่ k กับ polygon

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

เราสามารถหาคำตอบแบบวนซ้ำ โดยใช้การคำนวณแบบ **จำนวนเต็ม** ได้โดย

- กำหนดให้ counter เริ่มต้นเป็น 0
- ทุกครั้งที่เลื่อนไปพิจารณา scan line ถัดไป k = k + 1 ให้เพิ่มค่า counter ไป Δx
- ถ้า counter ที่ได้มีค่ามากกว่า Δy เราจะเพิ่มค่าจุดตัด x ไป 1 (นั่นคือ พจน์เศษส่วนข้างหลังมีค่า หลังจาก **ตัดเศษ** เป็นจำนวนเต็มเท่ากับ 1) แล้วลดค่า counter ลง Δy
- วนซ้ำขั้นตอนที่ 2 และ 3 สำหรับ scan line ถัดไปสำหรับเส้นขอบ polygon ที่พิจารณา

Precise Integer Arithmetic

เราสามารถหาคำตอบที่แม่นตรงยิ่งขึ้นโดยใช้การ **ปัดเศษ** แทนการ **ตัดเศษ**

- กำหนดให้ counter เริ่มต้นเป็น 0
- ทุกครั้งที่เลื่อนไปพิจารณา scan line ถัดไป k = k + 1 ให้เพิ่มค่า counter ไป $2\Delta x = (\Delta x + \Delta x)$
- ถ้า counter ที่ได้มีค่ามากกว่า Δy เราจะเพิ่มค่าจุดตัด x ไป 1 (นั่นคือ พจน์เศษส่วนข้างหลังมีค่า หลังจากตัดเศษ เป็นจำนวนเต็มเท่ากับ 1) แล้วลดค่า counter ลง $2\Delta y = (\Delta y + \Delta y)$
- วนซ้ำขั้นตอนที่ 2 และ 3 สำหรับ scan line ถัดไปสำหรับเส้นขอบ polygon ที่พิจารณา

ขั้นตอนวิธีดังกล่าว เทียบได้กับ การเปรียบเทียบค่าที่เพิ่มขึ้น Δx กับ $\Delta y/2$ (มากกว่า 0.5 ปัดขึ้น) ดังนั้นถ้า m=7/3 ค่า counter สำหรับ k แรกๆ จะเป็นดังนี้ 0, 6, 12 (ลดลงเป็น 12-2*7 = -2), 4, 10 (ลดลงเป็น -4), ... ซึ่งจะได้ว่าค่า x จะเป็น $x_0 + 0$, $x_0 + 0$, $x_0 + 1$, $x_0 + 1$, $x_0 + 2$, ...

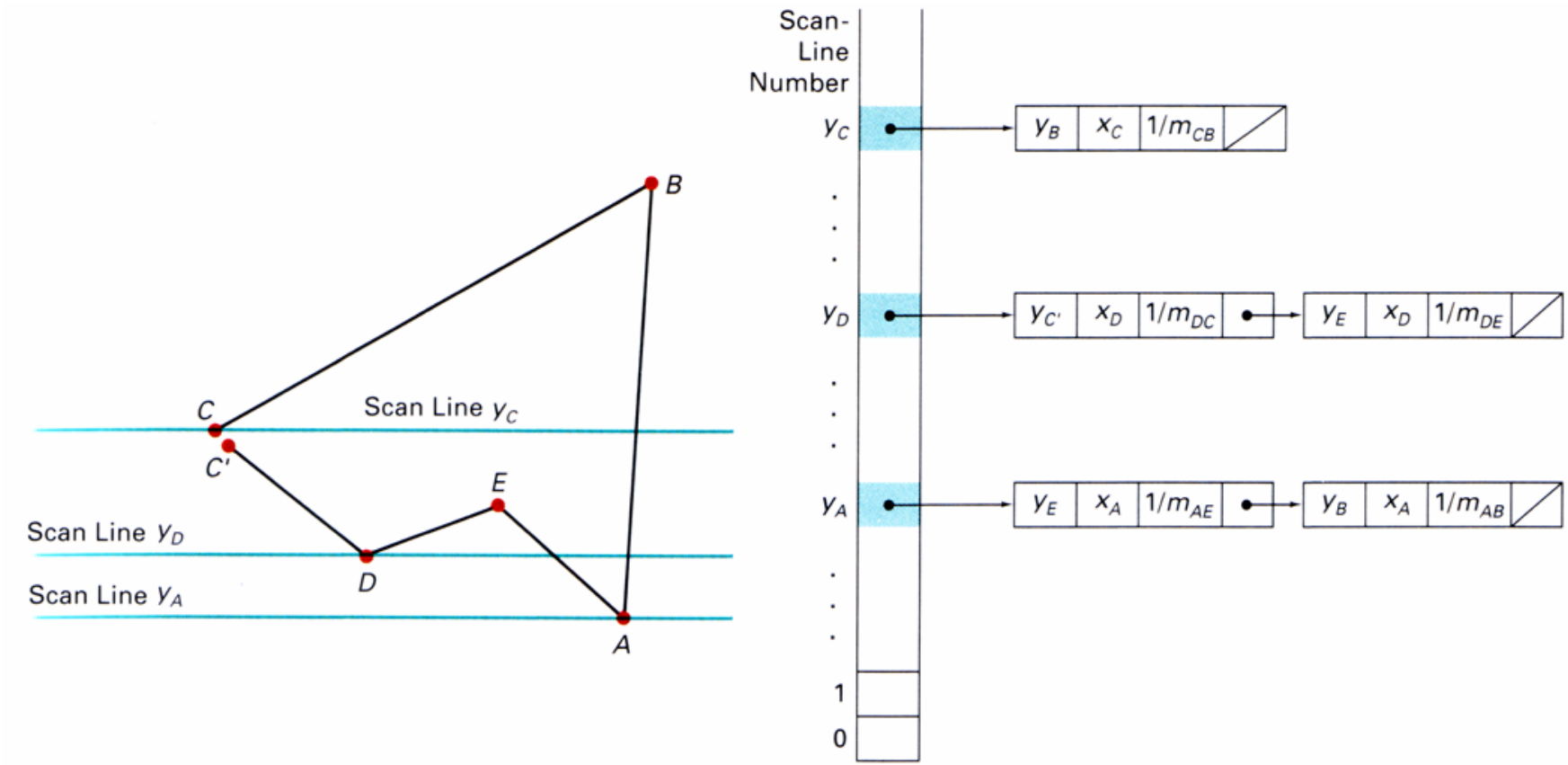
Implementation of the Algorithm

- เดินไปตามขอบ (edge) ของ polygon แต่ละเส้นในทิศทางทวนเข็ม (หรือตามเข็ม) นาฬิกา
- ทำการ shorten edge ในกรณีที่จุดยอด (vertex) ตัดผ่าน scan line พอดีเพื่อแยก แยะชนิดของจุดแบ่งเส้น scan line
- จัดเก็บแต่ละ edge ไว้ในตาราง edge โดยเรียงตามลำดับ พิกัด y ที่น้อยที่สุด (จุดยอดล่างสุดของ edge)
- ในแต่ละช่อง ของตาราง ใส่สมการของ edge ได้แก่ ค่า y ที่มากที่สุด (จุดบนสุด), จุดตัด x ของจุดล่างสุด และค่า 1/m
- สำหรับแต่ละ scan line เรียงสมการตาม ลำดับ ของค่าจุดตัด x จากซ้ายไปขวา

จากโครงสร้างข้อมูลดังกล่าว ไล่ลำดับ scan line จากจุดล่างสุดของ polygon ไปจุดบนสุด เพื่อสร้าง active edge list ซึ่งนำไปหาค่า x ของจุดตัด เพื่อนำไปสู่ขั้นตอน scan line polygon filling algorithm ต่อไป

Implementation Diagram

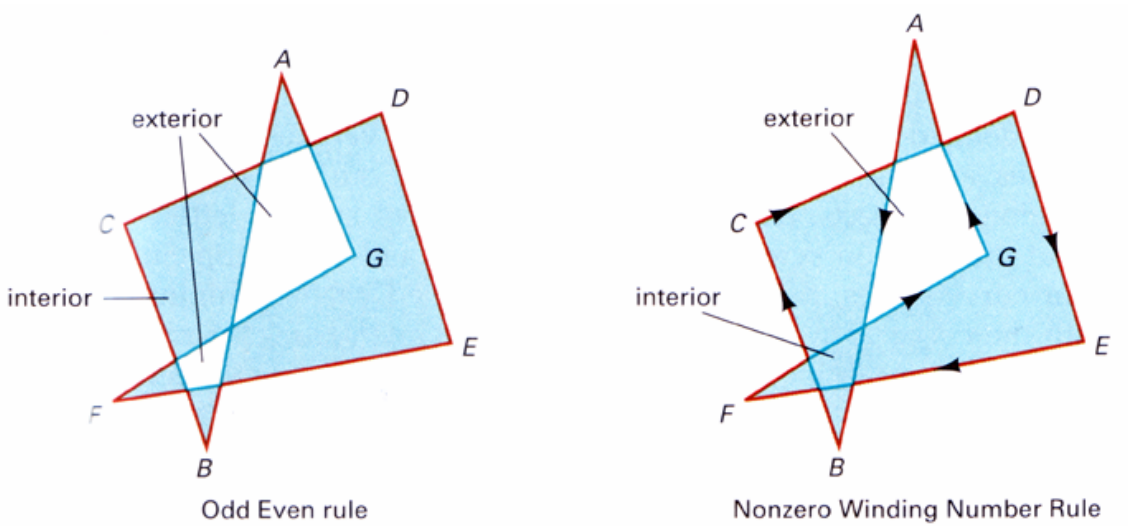
แผนผังแสดง โครงสร้างข้อมูล ของ integer arithmetic ของ scan line algorithm ซึ่งประกอบด้วย sorted edge table และ active edge list



Inside and Outside Tests

ที่ผ่านมา เราพิจารณา การประมวลผล เฉพาะ Polygon เดี่ยวอย่างง่าย ซึ่งแต่ละ edge จะไม่ตัดกัน ทว่า สำหรับกรณีทั่วไปใน Computer Graphics เราสามารถนิยามจุดยอดของ polygon (vertices) ได้อิสระ ซึ่งอาจจะทำให้เกิดการซ้อนทับกันของ edges ได้ (self-intersection) ดังรูป

Algorithm ที่ใช้สำหรับพิจารณาว่า จุดที่กำหนด อยู่ด้านใน (interior pixel) หรือด้านนอก (exterior pixel) polygon ที่สำคัญมีอยู่สองวิธี



Nonzero Winding Number Rules

วิธีนี้จะทำการนับจำนวนที่ edge ของ polygon วนรอบจุด ที่ต้องการทดสอบ ในทิศทาง ทวนเข็มนาฬิกา ซึ่งสามารถทำได้ดังต่อไปนี้

Algorithm

- จากจุดที่ต้องการทดสอบ ลาก vector **u** ให้ตัดผ่าน polygon แต่ไม่ผ่านจุดยอดมุมใดๆ ไปยังระยะอนันต์ กำหนดค่า winding counter เป็น 0
- หาผลคูณ cross product (ในระนาบ x, y) ระหว่าง vector **u** กับ edge **E** ที่ตัดผ่าน
- ผลลัพธ์ที่ได้จะเป็น องค์ประกอบ z ตั้งฉากกับระนาบ x, y พิจารณาได้ 2 กรณี
  - มีค่าเป็นบวก ให้นับค่า counter **เพิ่มขึ้น** 1
  - มีค่าเป็นลบ ให้นับค่า counter **ลดลง** 1
- ถ้าผลลัพธ์สุดท้ายหลังจากพิจารณาทุก edge แล้วจำนวน counter ไม่เท่ากับ 0 จุดที่ทดสอบเป็นจุดภายใน มิฉะนั้น จุดดังกล่าวเป็นจุดภายนอก

Fill of Curved Boundary Areas

Algorithm ประเภท scan line สำหรับ บริเวณที่มีขอบเขตเป็นเส้นโค้ง จะมีความซับซ้อน มากกว่า บริเวณที่เป็น polygon (ยกเว้นในกรณีพิเศษที่ บริเวณเป็นส่วนหนึ่ง ของภาคตัดกรวย เช่น วงกลม หรือ วงรี)

Boundary-Fill Algorithm

เป็นขั้นตอนวิธีที่สามารถสร้างได้ง่าย และใช้กันมาก ในระบบกราฟิกแบบโต้ตอบ (Interactive Graphics) ซึ่งต้องการ input จากผู้ใช้ คือจุดเริ่มต้นภายในบริเวณ แล้วระบายสี เริ่มจากจุดที่กำหนด แล้วแผ่กระจายออกไป จนกระทั่งไปสิ้นสุดที่ **ขอบ** ของบริเวณ

Flood-Fill Algorithm

คล้ายกับวิธี Boundary Fill แต่ว่าเงื่อนไขคือ เปลี่ยนสี จุดภาพที่ กำหนดว่าเป็น สี ภายในบริเวณ ให้เป็นสีที่ต้องการระบาย

Boundary-Fill Algorithm

Boundary Fill ต้องการ input จำนวน 3 ตัวได้แก่ 1) พิกัดของจุดเริ่มต้น (x, y) ภายใน บริเวณที่ต้องการระบาย, 2) สีที่ต้องการจะระบาย และ 3) สีของขอบของบริเวณ

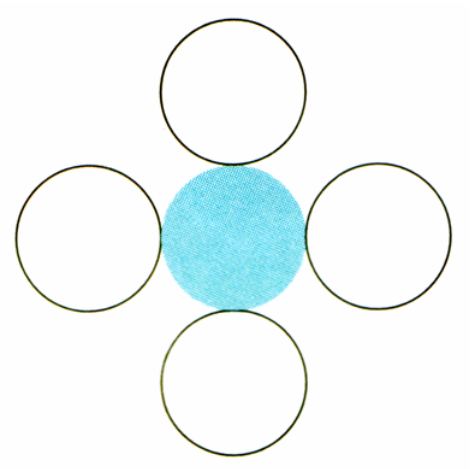
Steps (ความสัมพันธ์เวียนบังเกิด : A Recursion)

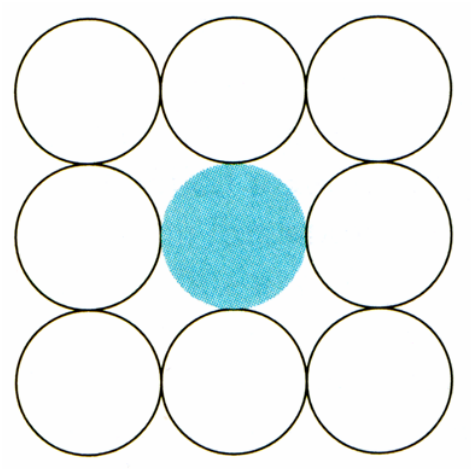
เริ่มจากจุดที่ผู้ใช้กำหนด (x, y), algorithm จะทำการตรวจสอบจุดรอบข้าง เพื่อพิจารณาว่า จุดนั้นเป็นจุดที่อยู่ **บนขอบ** หรือว่า ***ภายใน*** บริเวณ

กรณีที่จุดอยู่บนขอบ (สีของจุดปัจจุบันเป็นสีเดียวกับสีของขอบของบริเวณ)
หยุดการค้นหา (program terminates)

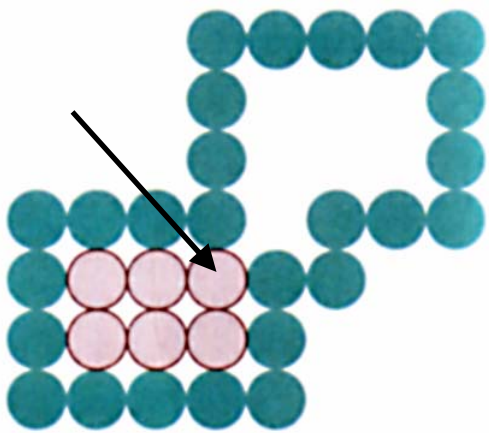
กรณีที่จุดอยู่ภายในบริเวณ
ระบายสีจุดนั้นด้วย สีที่ต้องการระบาย แล้วพิจารณาจุดรอบข้าง (4 จุด หรือ 8 จุด) สำหรับ recursion รอบถัดไป

Testing Neighboring Pixels





ภาพด้านซ้ายคือ แผนผังแบบ 4-connected test ในขณะที่แผนผังด้านขวามือ คือรูปแบบ 8-connected test ซึ่งรวมเอาจุดทแยงมุมทั้งสี่ไว้



ถ้าจุดปัจจุบันที่ทำการทดสอบคือจุดที่ศรชี้ การทดสอบแบบ 4-connect จะหยุดที่กรอบด้านล่างในขณะที่การทดสอบแบบ 8-connect จะแผ่ไปถึงกรอบด้านบนด้วย

Recursive Filling Code

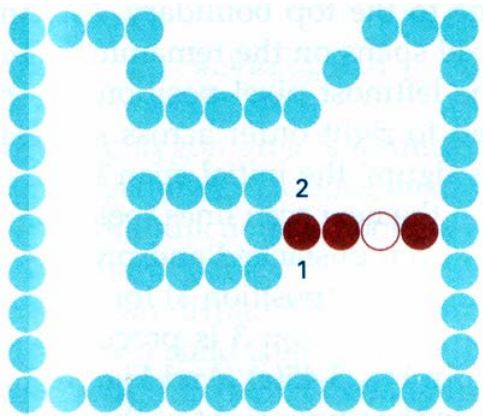
โปรแกรมด้านล่างแสดง recursive function ของ Boundary Fill Algorithm ชนิดที่ เป็นการทดสอบจุดรอบข้าง 4 จุด

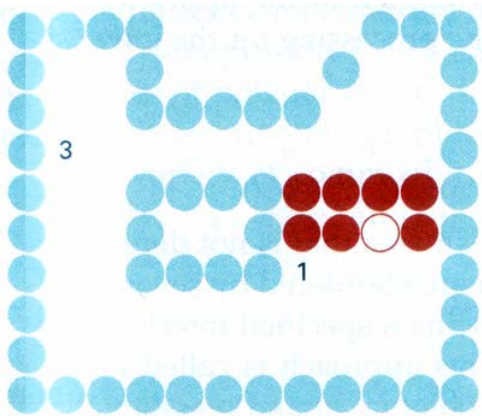
```
void cbuffer::boundaryfill4c (int x, int y, unsigned char fcolor, unsigned char bcolor)
//  fcolor  -> fill color
//  bcolor  -> boundary color
{
    unsigned char   ccolor; //  current color

    getpixel (x, y, &ccolor);

    if (ccolor != fcolor && ccolor != bcolor)
    {
        setpixel (x, y, fcolor);

        boundaryfill4c (x+1, y, fcolor, bcolor);
        boundaryfill4c (x-1, y, fcolor, bcolor);
        boundaryfill4c (x, y+1, fcolor, bcolor);
        boundaryfill4c (x, y-1, fcolor, bcolor);
    }
}
```

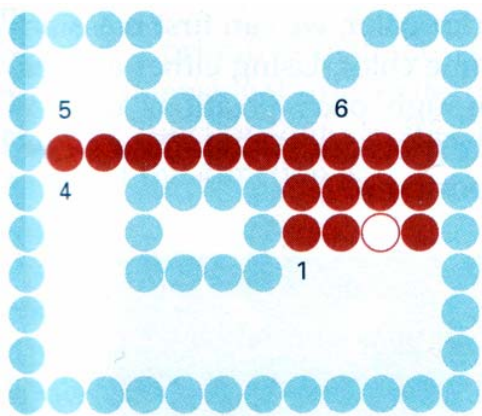
สังเกตว่า ฟังก์ชันแบบ recursive ที่เรียกตัวเอง 4(8) ครั้ง เมื่อพบจุดที่ยังไม่ได้ระบาย 1 จุด จะใช้หน่วยความจำ stack ปริมาณมาก ดังนั้นจึงต้องมีการปรับปรุงการทำงานให้มีประสิทธิภาพดีขึ้น

Improved Recursive Filling Code

- ระบายเฉพาะ span ในแนวนอน
- Push จุดเริ่มต้นของ span แถวบน และ ล่าง ของเส้นปัจจุบันใน stack
- Pop จุดเริ่มต้นจาก stack แล้วระบายเส้นนั้นในแนวนอน (ทำซ้ำ)







2
1

3
1

6
5
4
1

Flood-Fill Algorithm

บางครั้งเราอาจต้องการระบายบริเวณที่ล้อมรอบด้วยเส้นขอบหลายๆ สี ซึ่งสามารถทำได้โดย **แทนที่** สีปัจจุบัน ด้วยสีที่ต้องการระบาย ซึ่งแสดง ด้วยฟังก์ชัน ภาษา C++ บน class frame buffer ได้ดังต่อไปนี้

```
void cbuffer::floodfill4c (int x, int y, unsigned char fcolor, unsigned char ocolor)
//   fcolor  -> fill color
//   ocolor  -> old color
{
    unsigned char   ccolor; //  current color

    getpixel (x, y, &ccolor);

    if (ccolor == ocolor)
    {
        setpixel (x, y, fcolor);

        boundaryfill4c (x+1, y, fcolor, ocolor);
        boundaryfill4c (x-1, y, fcolor, ocolor);
        boundaryfill4c (x, y+1, fcolor, ocolor);
        boundaryfill4c (x, y-1, fcolor, ocolor);
    }
}
```

การเพิ่มประสิทธิภาพ ของโปรแกรม โดยลดปริมาณ stack สามารถทำได้ในทำนอง เดียวกันกับกรณี Boundary Fill Algorithm

Attributes of Output Primitives

นิยามได้ว่าคือ รูปแบบที่กำหนด สำหรับการที่จะแสดงผล output primitives ใดๆ ซึ่งแบ่งออกได้เป็น 2 ประเภท

1. ประเภท ที่ส่งผลต่อการวาดและแสดงผล primitive โดยตรง เช่น สี และขนาด และ รูปแบบของลายเส้น

2. ประเภท ที่ส่งผลต่อการวาด สำหรับเงื่อนไข พิเศษ ตัวอย่างเช่น ข้อมูลเกี่ยวกับความลึก (อ้างอิงกับระนาบจอภาพ) เพื่อแสดงวัตถุซ้อนกัน หรือ อนุญาตให้ผู้ใช้ สามารถ หยิบจับวัตถุ (ด้วยอุปกรณ์แบบโต้ตอบ mouse) ได้

การกำหนดค่า attribute สามารถทำได้โดย กำหนดเป็นตัวแปร สำหรับ ฟังก์ชันที่ทำหน้าที่วาดวัตถุ (local) หรือ กำหนดเป็นตัวแปรระบบ (global) ซึ่งฟังก์ชันจะ ต้องไปเรียกดูค่าปัจจุบัน เพื่อให้แสดงผลได้สอดคล้องกัน

Line Attributes: Types

Attributes พื้นฐานของเส้น (ตรง/โค้ง) ได้แก่ ชนิด (ลายเส้น) ความหนา และ สี ในที่นี้ จะกล่าวถึง วิธีการดัดแปลง ฟังก์ชันแสดงผลให้สัมพันธ์กับ ค่า attributes

——— solid line ใช้ algorithm การวาดเส้นพื้นฐาน

– – – – – dashed line ใช้ algorithm พื้นฐาน แต่วาด โดยเว้นช่องไฟซึ่งมีความกว้างเท่ากับ ความหนาของเส้น

..... dotted line ใช้ algorithm พื้นฐาน แต่วาดเป็นช่วงสั้นๆ โดยเว้นช่องไฟมากกว่าหรือเท่ากับความยาวของแต่ละช่วง

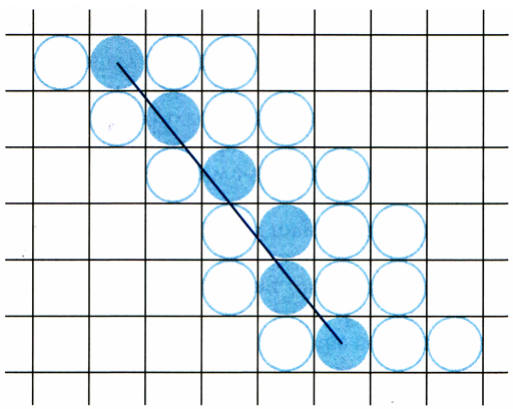
ถ้าใช้ Raster Algorithm การวาดเส้นเป็นช่วงๆ สามารถทำได้โดยการใช้ digital mask เพื่อ เปิด/ปิด จุดติดกันที่ต้องการวาดเช่น 00001111 แทน mask สำหรับ dashed line ซึ่งมีระยะ 4 จุด (สังเกตว่า เส้นแนวนอน กับ ทแยง มีระยะต่างกัน) ถ้าต้องการวาดให้ระยะเท่าๆ กันทุกๆ ค่าความชัน ต้องเปลี่ยน algorithm ให้วาดเส้นทีละช่วงแทน โดยแต่ละช่วงจะกำหนดได้ โดยจุดปลาย 2 จุด

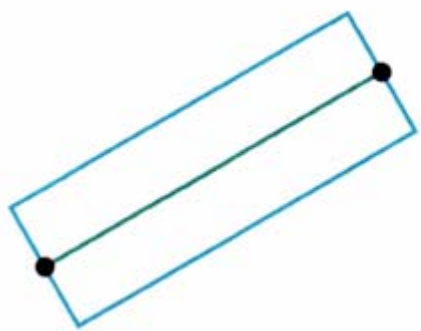
Line Attributes: Sizes

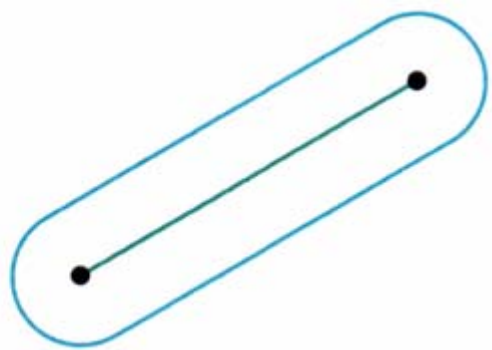
การเปลี่ยนขนาด (ความหนา) ของเส้น สามารถทำได้โดย วาดจำนวนจุดเพิ่มไปเป็นจำนวนเท่าตามความหนาที่ต้องการ

- ถ้าความชันของเส้นตรง น้อยกว่า 1 ให้เพิ่มจุดตามแนวตั้ง (ดังรูป)
- ถ้าความชันของเส้นตรง มากกว่า 1 ให้เพิ่มจุดตามแนวนอน ตั้งฉากกับทิศทาง

ถ้าต้องการวาดเส้นที่มีความหนามากๆ ให้ออกมาสมมาตร อาจจะใช้การเพิ่ม line cap ปิดท้ายเส้นก็ได้ ตัวอย่างเช่น butt cap (ดังรูป) สร้างโดยวาดเส้นตรงหลายเส้นขนานกัน โดยที่จุดปลายเปลี่ยนไป และความชันของจุดปลายที่เรียงกัน ในส่วนที่เป็น butt มีค่าเท่ากับ -1/m เมื่อ เส้นตรงมีความชัน m





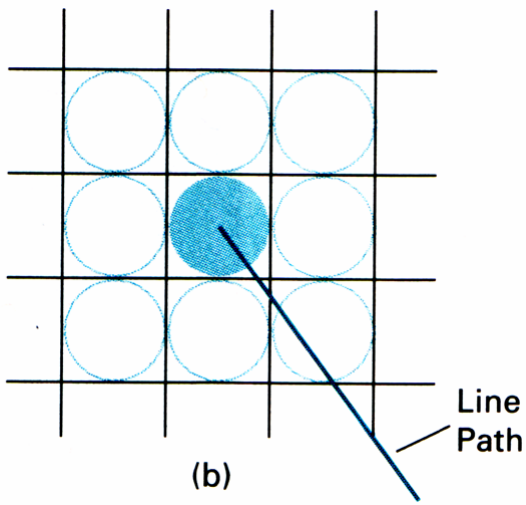


Line Attributes: Patterns

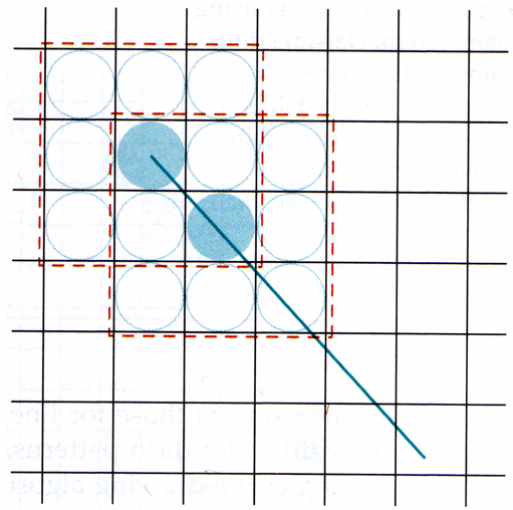
ลวดลายของเส้น สามารถทำได้โดยการทำ digital convolution ระหว่าง รูปแบบที่กำหนด (pixel mask) กับเส้นตรงนั้นๆ (ดังรูป)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(a)



(b)



นั่นคือ แทนที่จะวาดจุดภาพ ณ ตำแหน่งบนเส้นตรง ตามวิธีแบบ Raster Line ก็ ให้สะสมค่าใน Frame Buffer ด้วยค่าที่ปรากฏใน pixel mask แทน ในการใช้วิธีนี้ ควรพิจารณาว่าหาก mask ซ้อนทับกันที่ตำแหน่งใดๆ จะให้มีการ 1) บวกสะสมแล้วเฉลี่ย หรือว่า 2) เลือกค่าตาม mask ก่อน หรือ mask หลัง หรือว่า 3) ข้ามจุดที่ซ้อนทับกันไป วิธีนี้สามารถนำไปประยุกต์ใช้กับ Raster Polygon Filling Algorithm ได้อีกด้วย

Curve Attributes

สำหรับเส้นโค้ง เราสามารถนำวิธีกำหนด รูปแบบ และ ความหนาของเส้นตรงมาใช้ได้ แต่มีข้อควรระวังคือ

สำหรับ รูปแบบ (dashed, dotted) จะกำหนดโดยวิธี วาดเส้นโค้งที่ละช่วงย่อยๆ เช่น กรณีวงกลม จะลากเป็นช่วงที่มีองศา เท่าๆ กัน เพื่อที่จะสร้าง ระยะห่างที่เท่ากันตลอดเส้น

สำหรับความหนาเนื่องจากการวาดวงกลม (หรือเส้นโค้งอื่นๆ) จะใช้วิธีการสะท้อนจุดสมมาตร ที่ octant ต่างๆ ดังนั้นการทำให้เส้นหนา โดยเพิ่มจุดก็ควรจะใช้วิธีการสะท้อนในทำนองเดียวกัน

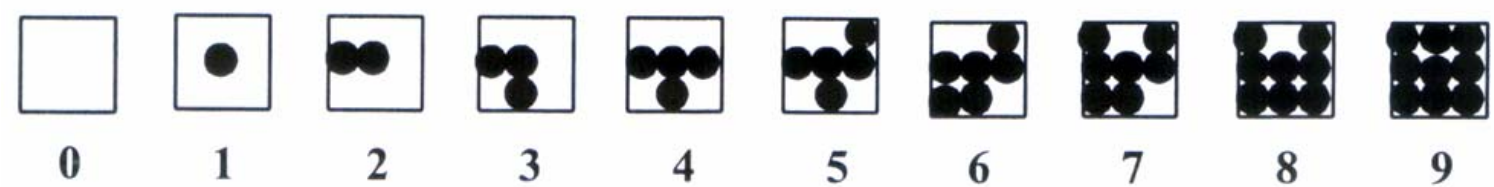
อีกวิธีหนึ่งที่ง่ายกว่า แต่ใช้ทรัพยากร CPU มากกว่า คือ แทนที่จะวาดวงกลมที่มีความหนา 3 จุดภาพ ก็วาดวงกลม ที่มีรัศมีต่างๆ 2 จุดภาพ แล้วระบายสีบริเวณระหว่างกลางแทน ซึ่งวิธีนี้สามารถใช้ได้กับเส้นโค้งที่มีสมการรูปแบบอื่นๆ ได้ด้วย

Picture Approximation

คือขั้นตอนวิธีที่ใช้สำหรับแสดงผลรูปที่มีระดับสีหลายระดับ บนอุปกรณ์แสดงผลที่รองรับแค่ สีขาว หรือ สีดำ เท่านั้น หรือ ระดับสีที่มีจำนวนน้อยกว่า การประยุกต์ใช้งานทั่วไป พบได้ใน Dot Matrix/Laser Printer และจอแสดงผล แบบ LED ซึ่งมีจำนวนสีจำกัด

ขั้นตอนวิธีนี้เรียกว่า การทำ Halftone แบบที่ง่ายที่สุดเรียกว่า ordered dithering โดยที่จัดสรรพื้นที่ในหน่วยความจำ array จัตุรัส สำหรับแต่ละจุดภาพ ตัวอย่าง เช่น กำหนด binary array ขนาด 3x3 ซึ่งใช้แทนระดับความเข้มของ 1 จุดภาพ

$$\begin{pmatrix} 7 & 9 & 5 \\ 2 & 1 & 4 \\ 6 & 3 & 8 \end{pmatrix}$$



Ordered Dithering

ด้านล่างแสดงตัวอย่าง ของการทำ ordered dithering สำหรับภาพระดับความเทา 256 ระดับ แต่เนื่องจาก dither matrix ใช้ขนาด 3x3 จึงรองรับได้เพียง 10 ระดับ ดังนั้นจึงต้องมีการทำ quantization บนภาพก่อน

```
index = 9 - (int) (9.0f * value/255.0f)
```

Original

Dithered Image

Conclusions

- 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
- Output Primitive Attributes
- Picture Approximation using Halftone