Lecture 7A: Filled-Area Primitives

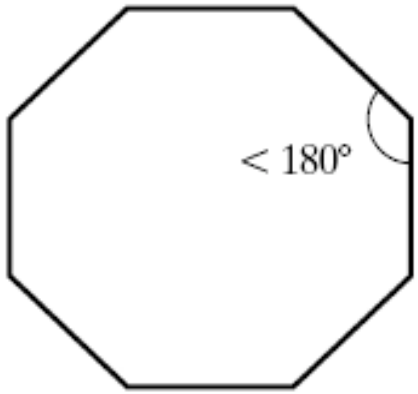
- Polygon Filling Algorithm
- Inside-Outside Test
- Boundary-Fill and Flood-Fill Algorithm

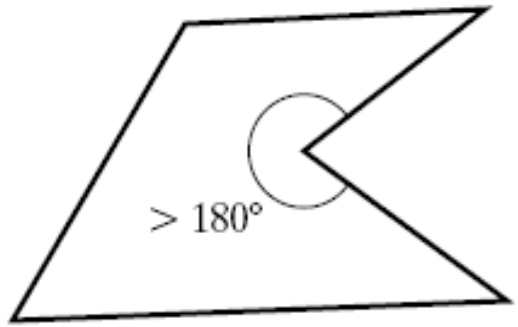
Filled-Area Primitives

Polygon (รูปหลายเหลี่ยม):

A cyclic list of points called vertices. Consecutive vertices in this list are connected by line segments called edges.

Convex & Concave Polygon:





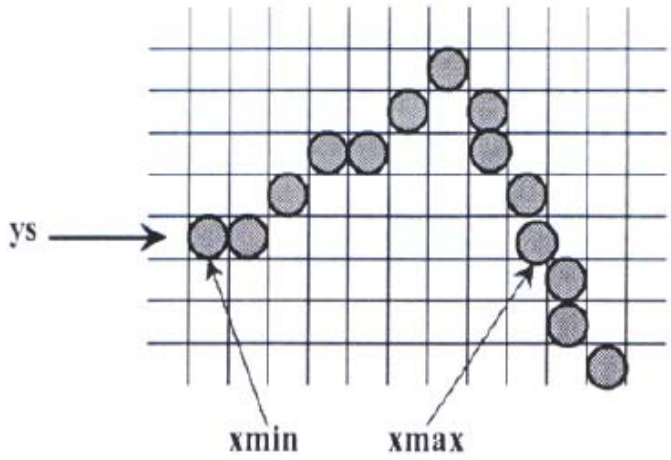
Convex: มุมภายในทุกมุม < 180

Concave: มีอย่างน้อย 1 มุมภายในที่ > 180 (มีด้านมากกว่า 4 ด้านเสมอ)

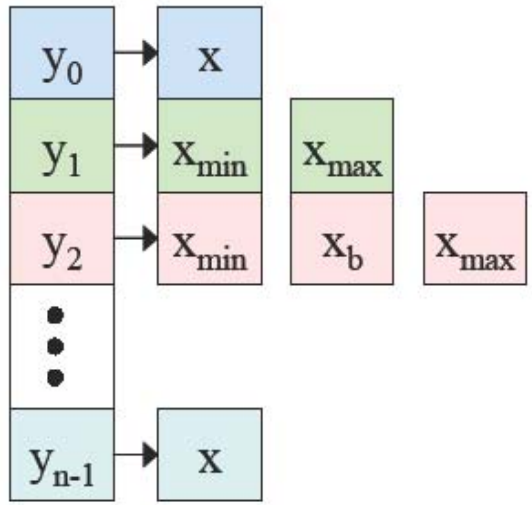
Polygon Filling Algorithm

Simple Polygon Filling Algorithm

วิธีการที่ง่ายที่สุด ของการระบายรูปหลายเหลี่ยม คือ Scan Line Algorithm



1. วาดรูป Polygon เชื่อมต่อจุดโดยใช้ เทคนิคการวาดเส้นตรง
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

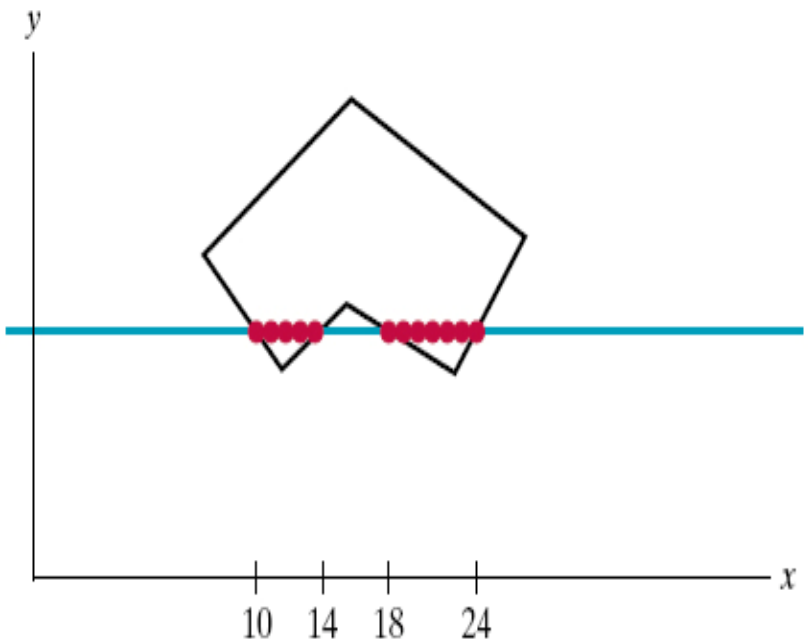


ใช้เมื่อ Polygon เป็นแบบ Convex

Line Scan Algorithm

เมื่อ Polygon เป็นแบบ Concave

ขั้นตอนวิธีการระบาย Concave Polygon ขยายผลจากการระบาย Convex Polygon ดังนี้:



ระบุจุดขอบ (edge pixels)
เรียง edge pixels ตามลำดับของ y

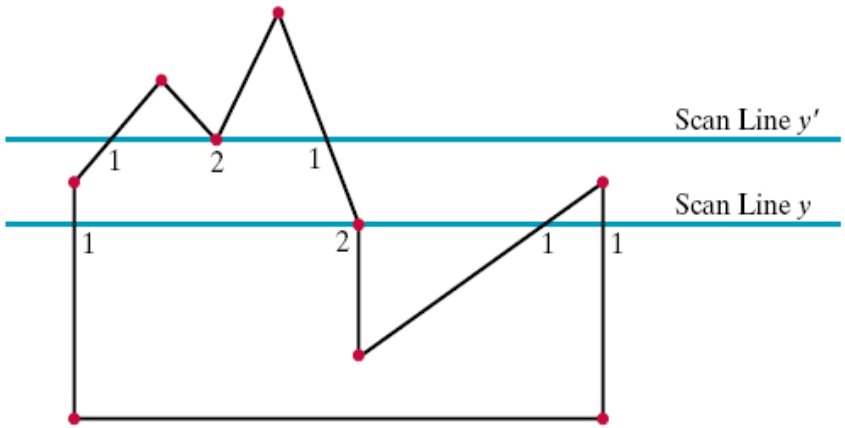
for (จุดที่มีค่า y เท่ากัน)do
    เรียงลำดับตามค่าของ x
    ระบายระยะ เว้นระยะ ตามแนวนอน
end for

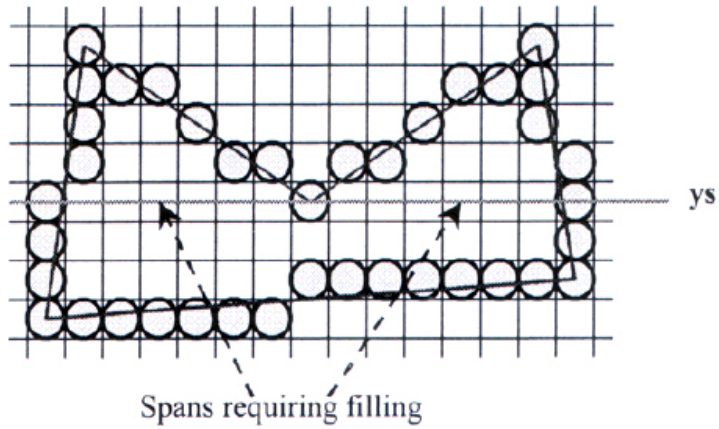
Exceptions

กรณีที่ scan line ตัดผานจุดยอดของ polygon **พอดี** จากรูป ที่จุดหมายเลข 2 มีความเป็นไปได้ อยู่ 2 รูปแบบ

กรณีที่ 1. จุดที่ scan line ตัดผานแบ่ง polygon ออกเป็นสองชวง (span) ซึ่งอยู่ภายใน polygon ทั้งสองชวง (scan line y')

กรณีที่ 2. จุดที่ scan line ตัดผานแบ่ง polygon ออกเป็นสองชวง แต่มี เพียงชวงเดียวที่ อยูภายใน polygon (scan line y)

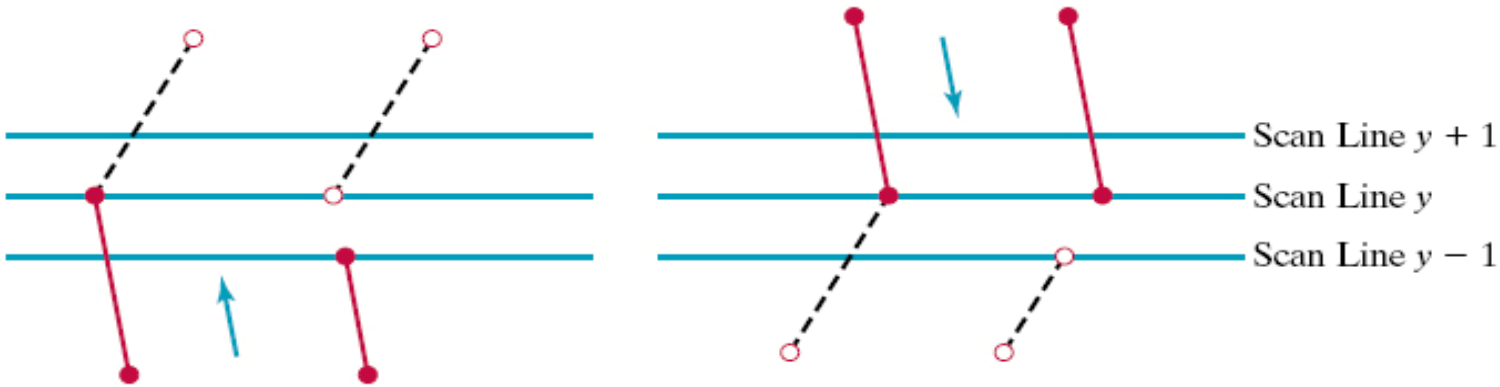




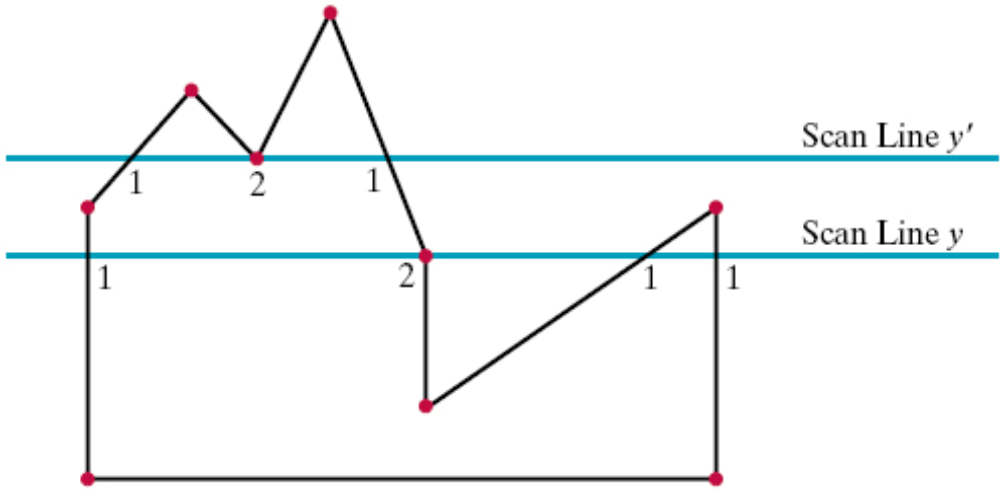
Identifying Sharing Vertices

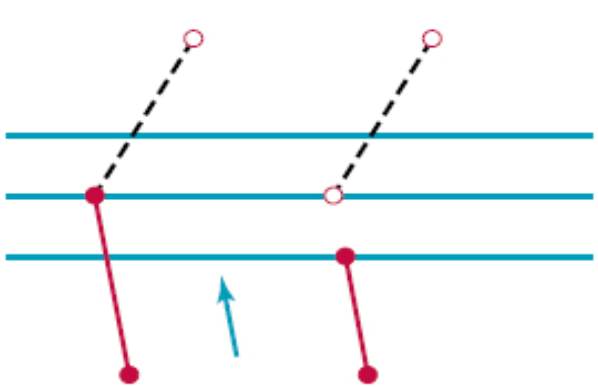
การระบุว่า scan line ของกรณี Exceptions นั้นจัดอยู่ในรูปแบบใด สามารถทำได้ดังต่อไปนี้

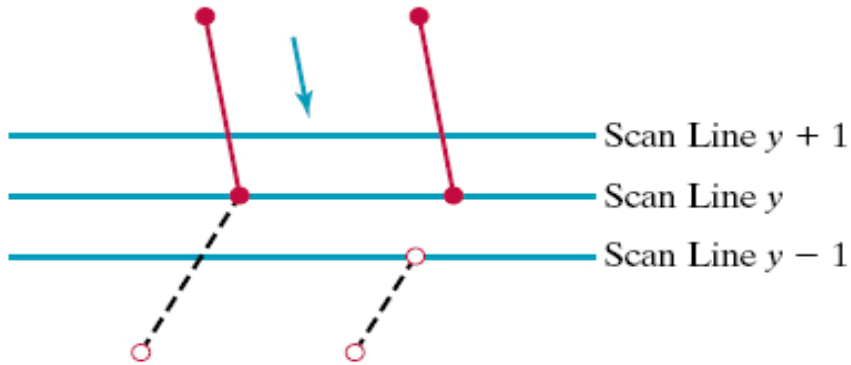
- เดิน (trace) ตามเส้นรอบ polygon ในทิศทาง ทวนเข็ม หรือ ตามเข็มนาฬิกาก็ได้
- สำหรับ จุดยอด ของ polygon สังเกตลักษณะของ การเปลี่ยนแปลง ค่า y
  - ถ้าค่า y เปลี่ยนแปลง ไปในทิศทางตรงกันข้าม แสดงว่า จุดตัดนั้นแบ่ง scan line ออกเป็น 2 ส่วน ภายใน polygon (กรณีที่ 1) **เพิ่มจุดตัดซ้ำ 2 จุด**
  - ถ้าค่า y ลดลงตลอด (หรือเพิ่มขึ้นตลอด) เมื่อเทียบ จุดยอดเป็นจุดศูนย์กลาง (monotonic) แสดงว่า จุดตัดนั้น แบ่งออกเป็นส่วนที่อยู่ภายในกับภายนอก polygon (กรณีที่ 2) **ให้ตัดเส้นขอบลง 1 scan line**



Identifying Sharing Vertices



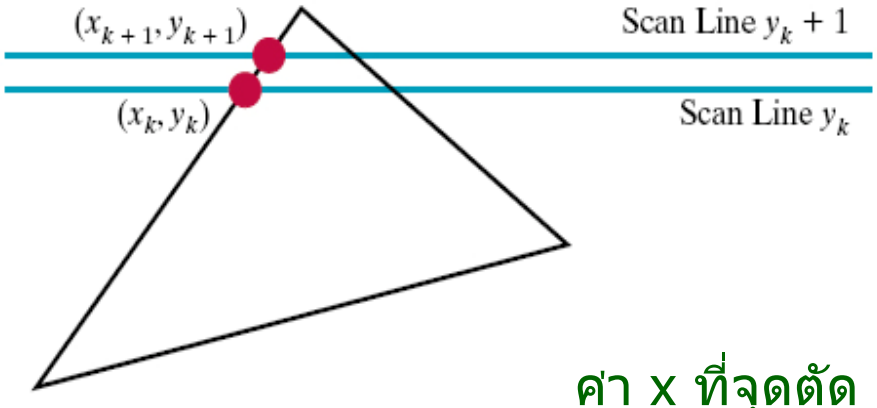




Coherent Polygon Processing

เราสามารถใช้ประโยชน์ ความเกี่ยวเนื่องกันขององค์ประกอบ polygon สำหรับ scan line ที่ติดกัน มาใช้เพิ่มประสิทธิภาพ ในการประมวลผลได้

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



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

เนื่องจาก $y_{k+1} - y_k = 1$

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

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

An Efficient Polygon Algorithm

จากสมการหาพิกัด x ของจุดตัดระหว่าง scan line ที่ k กับ polygon

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

เราสามารถหาคำตอบแบบวนซ้ำ โดยใช้ การคำนวณแบบ **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 พจน์เศษสวนข้างหลังมีค่า 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 พจน์เศษสวนข้างหลังมีค่า 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), ... ซึ่งจะได้ ว่าค่า x จะเป็น x_0+0, x_0+0, x_0+1, x_0+1, x_0+2,...

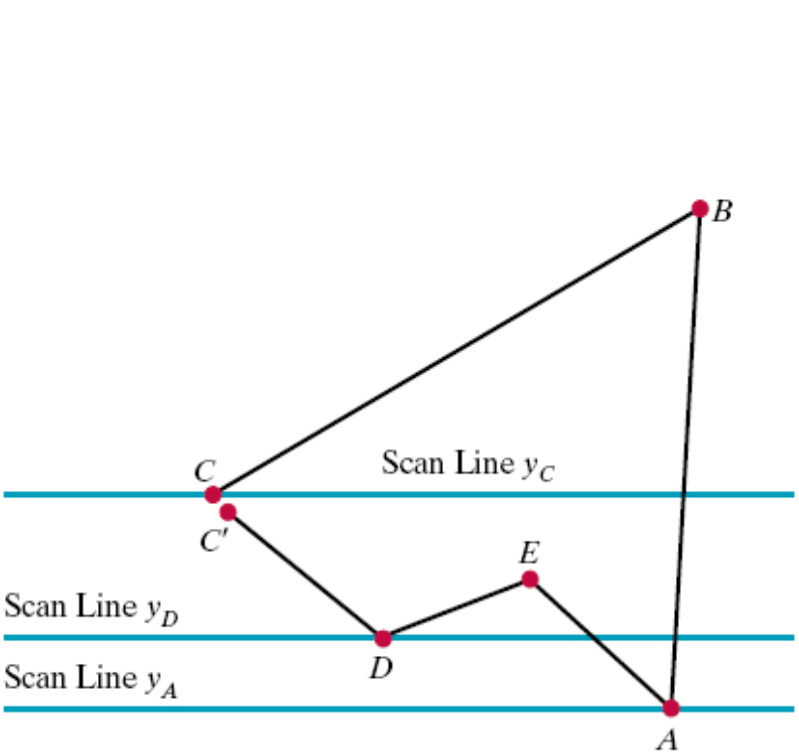
Implementation of the Algorithm

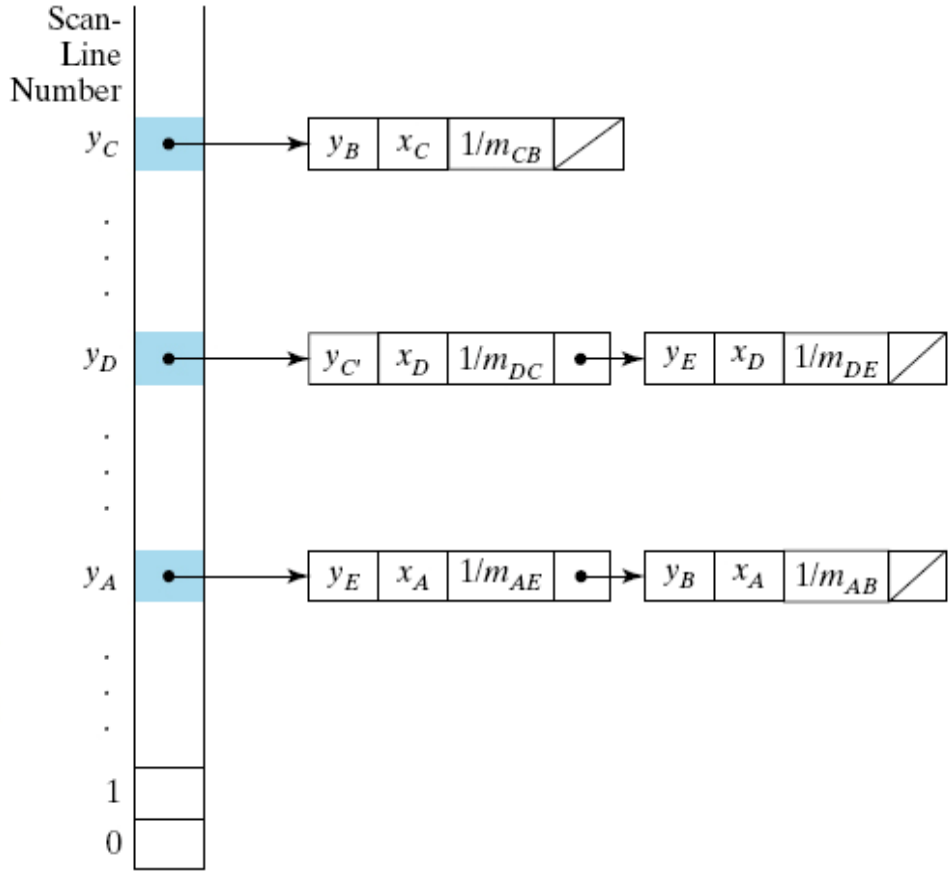
- เดินไปตามขอบ (edge) ของ polygon แต่ละเส้นในทิศทางทวนเข็ม (หรือตามเข็ม) นาฬิกา
- ทำการ shorten edge ในกรณีที่จุดยอด (vertex) ตัดผ่าน scan line พอดีเพื่อแยก แยะชนิดของจุดแบ่งเส้น scan line
- จัดเก็บแต่ละ edge ไว้ในตาราง edge โดยเรียงตามลำดับ พิกัด y ที่น้อยที่สุด (จุดยอดล่างสุดของ edge)
- ในแต่ละช่อง 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

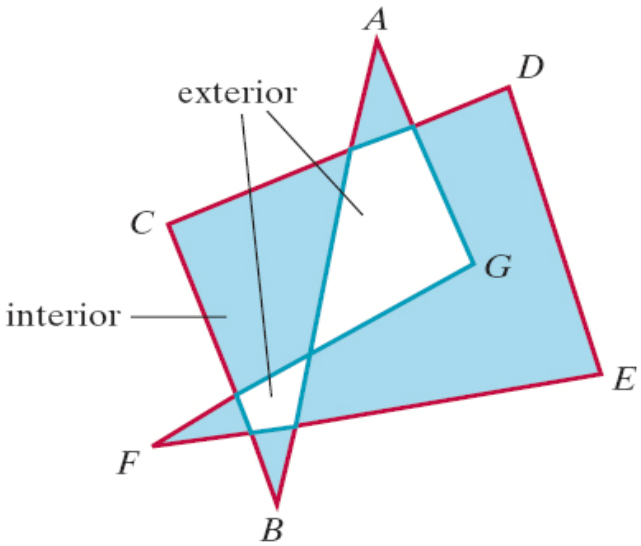




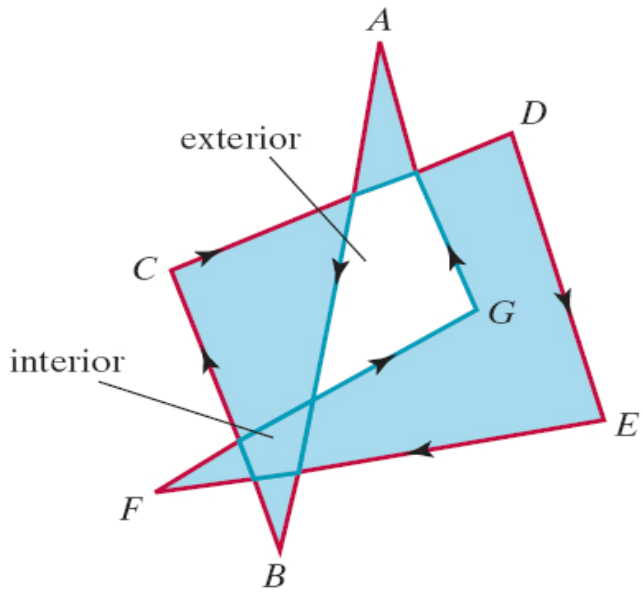
Inside and Outside Test

ที่ผานมา เราพิจารณา การประมวลผล เฉพาะ Polygon เดี่ยวอยางงาย ซึ่งแตละ edge จะไมตัดกัน ทวา สำหรับกรณีทั่วไปใน Computer Graphics เราสามารถนิยามจุดยอดของ polygon (vertices) ไดอิสระ ซึ่งอาจจะทำใหเกิดการซอนทับกันของ edges ได (self-intersection) ดังรูป

Algorithm ที่ใชสำหรับพิจารณาวา จุดที่กำหนด อยูดานใน (interior pixel) หรือดานนอก (exterior pixel) polygon ที่สำคัญมีอยูสองวิธี



Odd-Even Rule



Nonzero Winding-Number Rule

Nonzero Winding Number Rules

วิธีนี้จะทำการนับจำนวนที่ edge ของ polygon วนรอบจุด ที่ต้องการทดสอบ ในทิศทาง ทวนเข็มนาฬิกา ซึ่งสามารถทำได้ดังต่อไปนี

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 องศ์ประกอบ 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 กำหนดว่าเป็นสีภายในบริเวณ ให้เป็นสีที่ต้องการระบาย

Boundary-Fill Algorithm

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

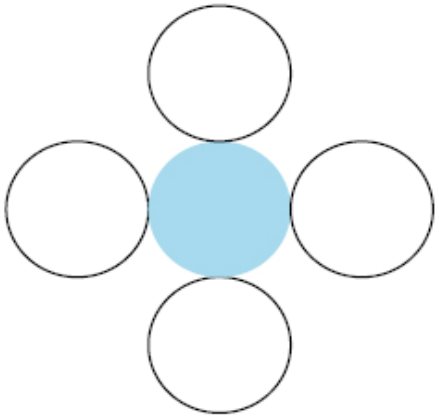
Steps (ความสัมพันธ์เวียนบังเกิด : A Recursion)

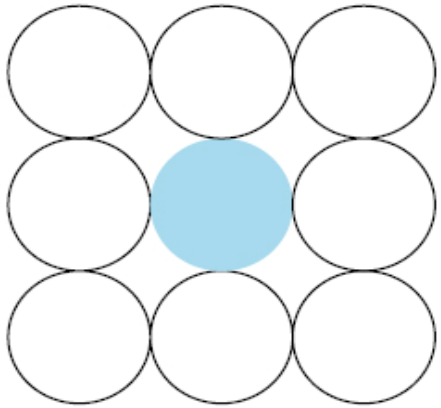
เริ่มจากจุดที่ผู้ใช้กำหนด (x,y),algorithm จะทำการตรวจสอบจุดรอบข้าง เพื่อพิจารณาว่า จุดนั้นเป็นจุดที่อยู่ **บนขอบ** หรือว่า ***ภายใน*** บริเวณ

กรณีที่จุดอยู่บนขอบ (สีของจุดปัจจุบันเป็นสีเดียวกับสีของขอบของบริเวณ)
หยุดการค้นหา (program terminates)

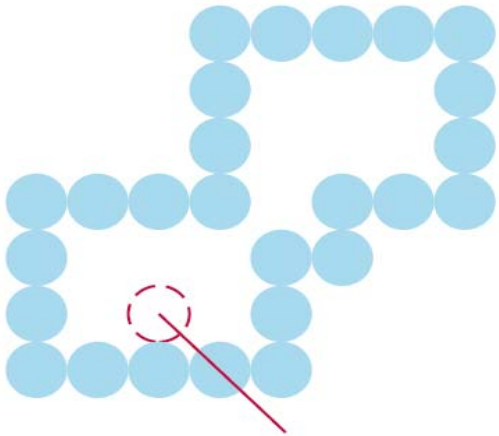
กรณีที่จุดอยู่ภายในบริเวณ
ระบายสีจุดนั้นด้วย สีที่ต้องการระบาย แล้วพิจารณาจุดรอบข้าง (4 จุด หรือ 8 จุด) สำหรับ recursion รอบถัดไป

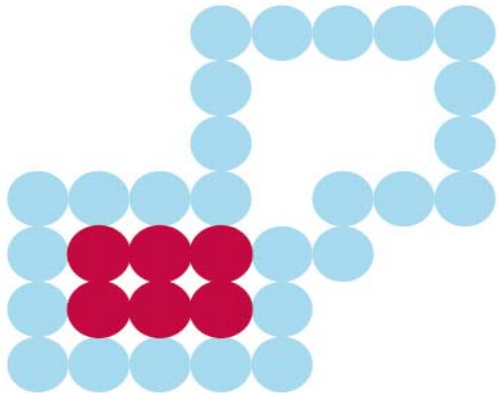
Testing Neighboring Pixels





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





ถ้าจุดปัจจุบันที่ทำการทดสอบคือจุดที่ศรชี้ การทดสอบแบบ 4-connect จะหยุดที่กรอบด้านล่างในขณะที่การทดสอบแบบ 8-connect จะแผ่ไปถึงกรอบด้านบนด้วย

Recursive Filling Code

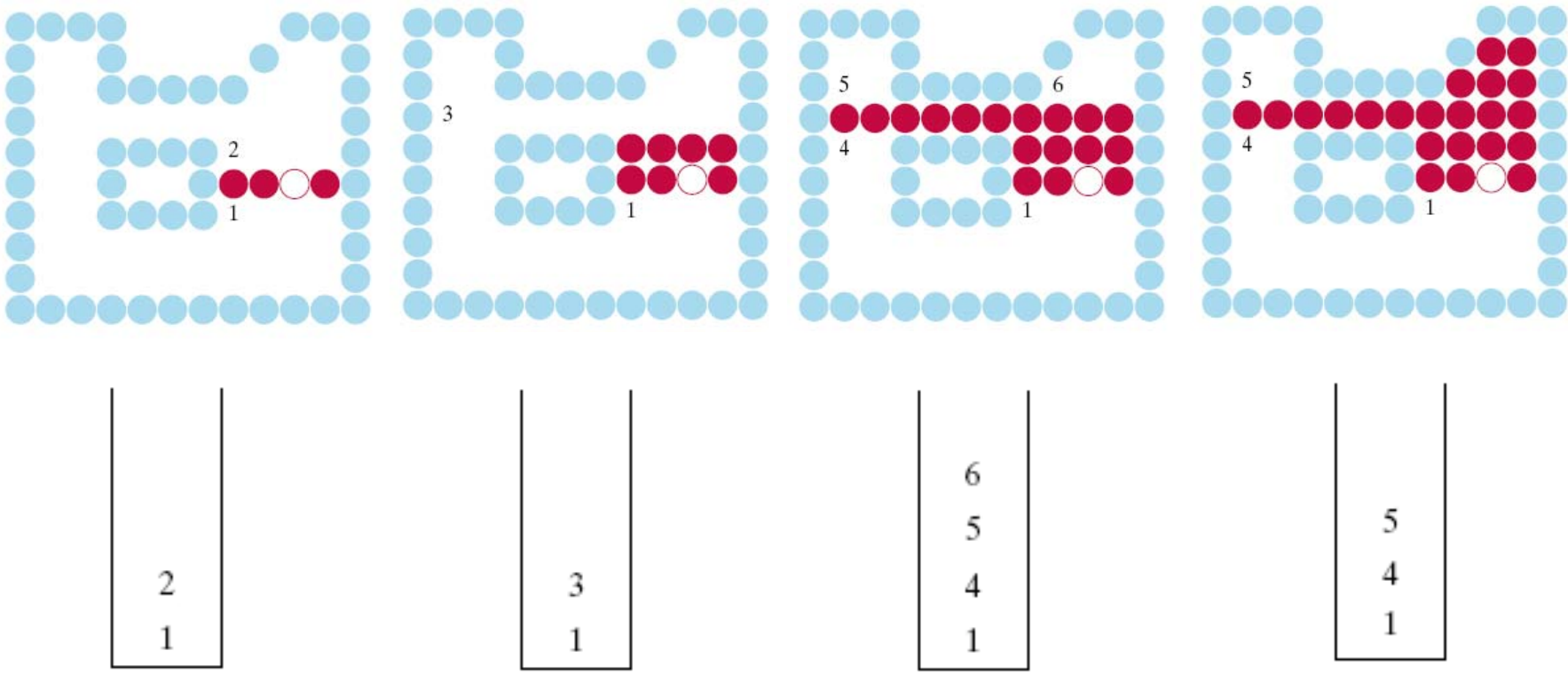
โปรแกรมด้านล่างแสดง recursive function ของ Boundary Fill Algorithm ชนิดที่ เป็นการทดสอบจุดรอบข้าง 4 จุด

```
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)



Flood-Fill Algorithm

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

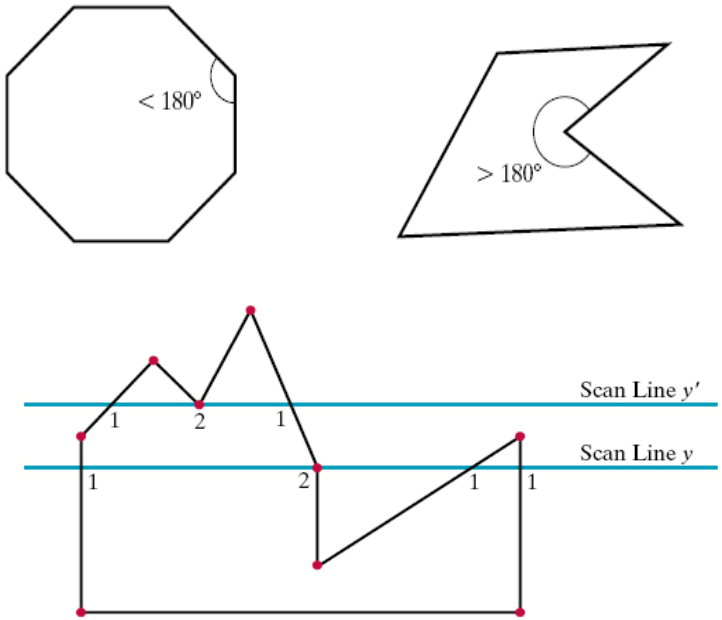
```
void cbuffer::floodfill4C (int x, int y, unsigned char fcolor, unsigned char ocolor)
//  fcolor  -> fill color
//  ocolor  -> old color
{
    unsigned char   ccolor; //  current color

    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);
    }
}
```

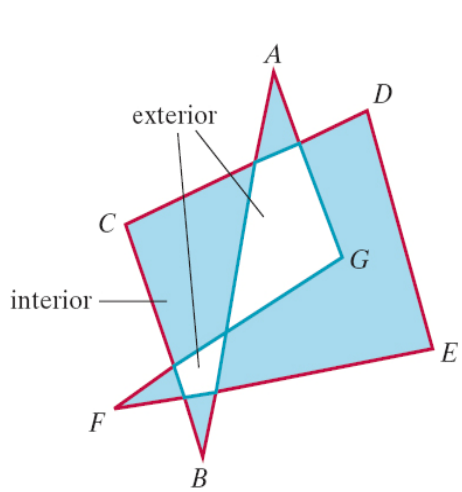
การเพิ่มประสิทธิภาพ ของโปรแกรม โดยลดปริมาณ stack สามารถทำได้ในทำนอง เดียวกันกับกรณี Boundary Fill Algorithm

Homework #2

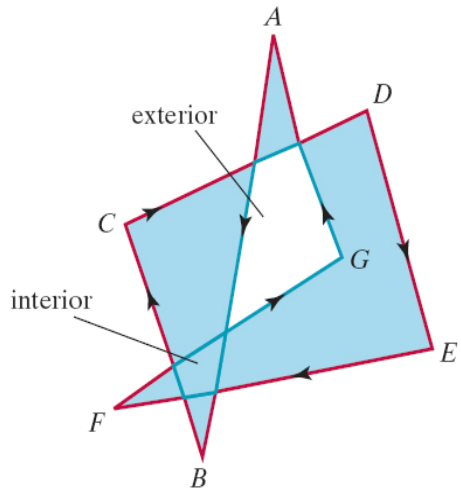
เขียนโปรแกรมวาดรูปขอบ (edges) ของ Polygon เชื่อมต่อจุด (vertices) โดยใช้เทคนิคการวาดเส้นตรงแบบ DDA ให้โปรแกรมสามารถรับลำดับของจุด (vertices) ใดๆ ได้จากผู้ใช้

ตัวอย่าง Polygons





Odd-Even Rule



Nonzero Winding-Number Rule