ปีการศึกษา 2533

POST ANTI-ALIASING

โดย

1. นายวีระยุทธ งามสม
2. นายสมปอง ศรีวิเชียร
3. นายสุรชัย สถาพรวัฒนายนต์

อาจารย์ที่ปรึกษา

อาจารย์ บวรวรรณ สาริกะภูติ

สารบัญ

บทที่ 1 บทนำ

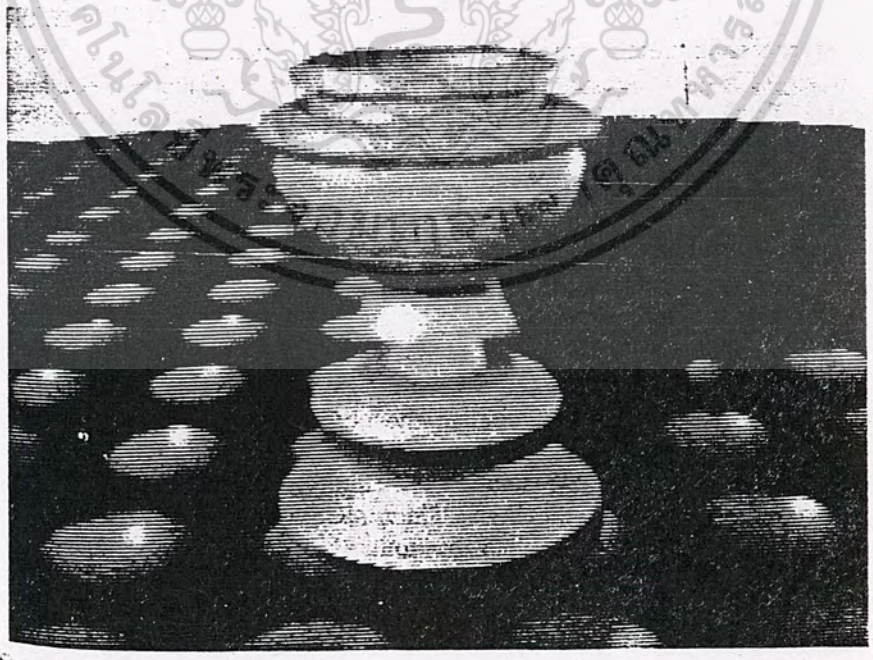
ในโลกของคอมพิวเตอร์กราฟิกการแสดงผล (Display) บนจอภาพเป็นอีกปัญหาหนึ่งที่เกิดขึ้นกับผู้ใช้คอมพิวเตอร์ หรือผู้เขียนโปรแกรม ที่มีความต้องการให้ภาพที่ screen มีความคมชัดเหมือนที่ตั้งใจไว้แต่พอโปรแกรมทำงานจริงๆภาพที่ปรากฏกลับไม่คมชัด หยาบและมีรอยหยัก(aliasel) ทำให้การออกแบบการใช้งานไม่ละเอียดเท่าที่ควร ดังนั้นจึงมีการพัฒนาการแสดงผลทางด้านกราฟิกให้ดีขึ้น

จุดเริ่มต้นของคอมพิวเตอร์กราฟิก เป็นการเริ่มต้นของการใช้ส่วนที่แสดงผล (Display Device) ที่เป็นจอภาพ CRT (Cathode Ray Tube) โดยแบ่งเทคนิคการแสดงผลแบ่งออกเป็น 2 วิธีคือเวกเตอร์สแกน (Vector Refreshed Monitors)เป็นการทำงานโดยใช้สัญญาณนาฬิกบังคับการส่งยิงแสงเป็นลำแสงพุ่งไปที่จอภาพ ใช้ในการสร้างภาพ อีกวิธีหนึ่งก็คือ ราสเตอร์สแกน (Raster Scaned Monitors)การทำงานถ้าบนจอโมโนโครมหรือจอสีเดียวจะกวาดไปบนจอภาพทีละเส้นสแกน แต่ในจอสีจะใช้ลำแสงอิเลคตรอน 3 สี โดยสแกนเส้นคู่หรือเส้นคี่สลับกันไป

จากปัญหาที่เกิดขึ้นสามารถแก้ปัญหาได้ 2 วิธีคือ

1. ทางฮาร์ดแวร์(Hardware) โดยการใช้จอภาพที่มีค่าความละเอียด (Resolution) สูงซึ่งได้แก่ จอภาพชนิดราสเตอร์สแกน CGA (Color Graphic Adapter) EGA (Enhanced Graphic Adapter) VGA (Vedio Graphic Array) และจอภาพชนิดเวกเตอร์สแกน เป็นต้น ภาพที่ได้จะให้ค่าความละเอียดที่แตกต่างกัน ดังภาพต่อไปนี้

รูป 1.1 ภาพของจอภาพที่มีความละเอียดต่ำ (Low resolution)

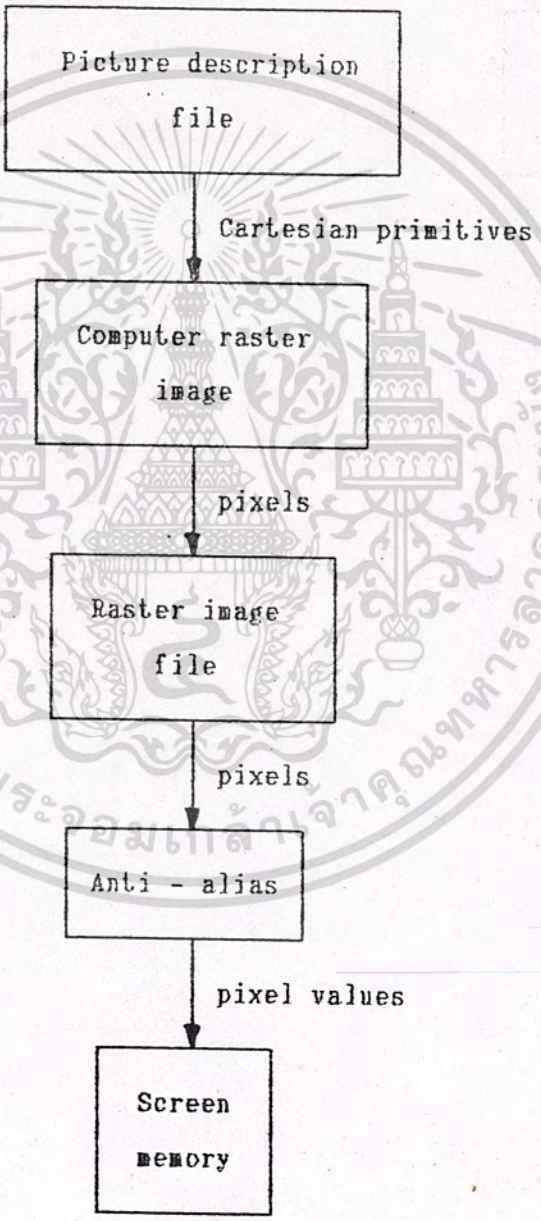


รูป 1.2 ภาพของจอภาพที่มีความละเอียดสูง (High resolution)

2. ทางซอฟท์แวร์ (software) ได้แก่ โปรแกรมที่ใช้ในการแก้ปัญหารอยหยักของภาพให้ดีขึ้น

วิธีการแก้ปัญหาที่ผู้จัดทำเลือกใช้คือ ทางซอฟท์แวร์ ทั้งนี้เพื่อลดต้นทุนในการให้ค่าความละเอียดของการแสดงผล เพื่อที่จะลดรอยหยัก(aliased) ให้น้อยลง ด้วยการทำ Antialiasing

เทคนิคการทำ Antialiasing นี้เป็นการแบ่ง pixels จำนวนมากที่เป็นตัวแปรในการวาดขอบภาพ เพื่อให้เป็นหยักเล็กน้อยลง สำหรับขั้นตอนและกระบวนการมีดังนี้



รูป 1.3 แสดงขั้นตอนการทำ Antialiasing

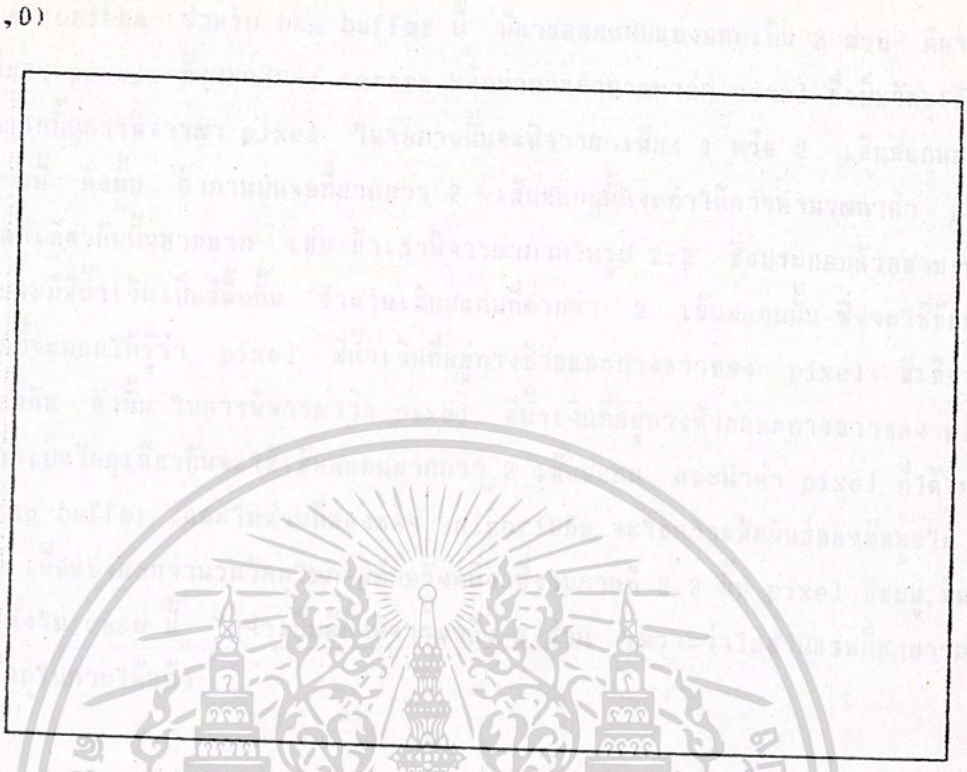
ซึ่งในส่วนของขั้นตอน (process) แม้ว่าไม่ได้อธิบายรายละเอียดไว้ แต่จะกล่าวรายละเอียดในบทต่อไป

บทที่ 2 จะอธิบายการกำหนดขอบเขตของวัตถุโดยใช้ tag buffer

บทที่ 3 จะอธิบายครอบคลุมแนวทางโครงสร้างของขอบเขตวัตถุ

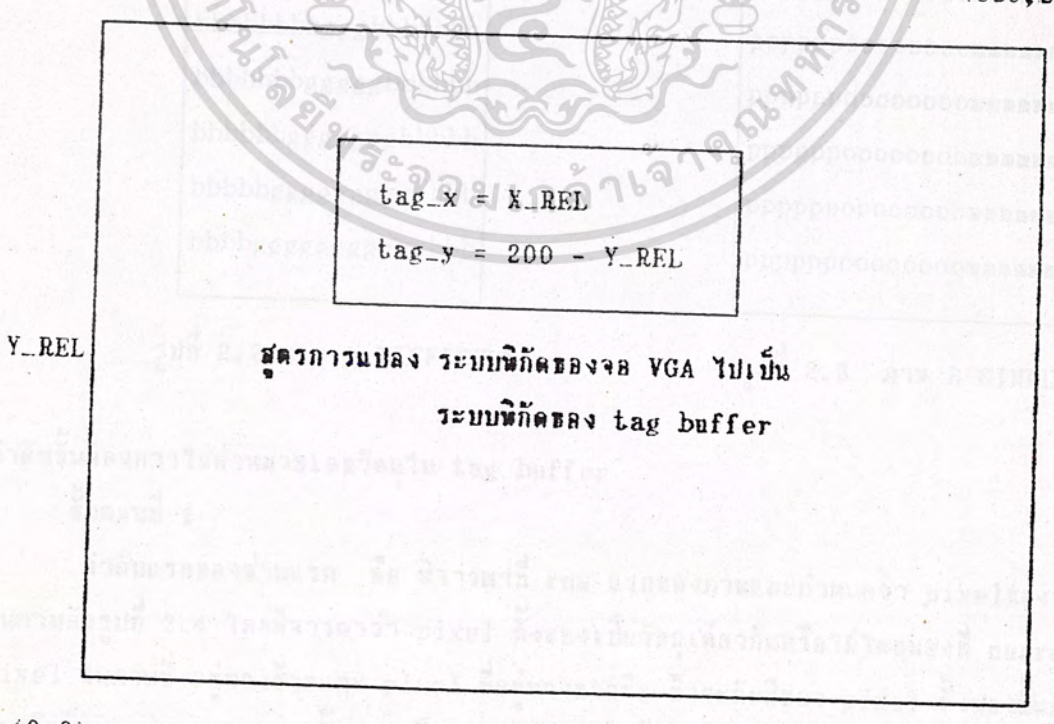
บทที่ 4 จะแนะนำเทคนิคที่นำมาใช้ในการ smooth รอยหยักตลอดขอบริมของวัตถุ

บทที่ 5 จะอธิบายถึงวิธีการแก้ปัญหาการ antialrasing ซ้ำกันเมื่อมีวัตถุสองชิ้นที่ overlab กัน



รูปที่ 2.1 ระบบพิกัดของ จอ VGA 320x200

Algorithm เปลี่ยนพิกัด VGA เป็นพิกัด TARGA ระบบพิกัด TARGA แสดงในรูปที่ 2.2



สูตรการแปลง ระบบพิกัดของจอ VGA ไปเป็น
ระบบพิกัดของ tag buffer

รูปที่ 2.1A ระบบพิกัดของ tag buffer

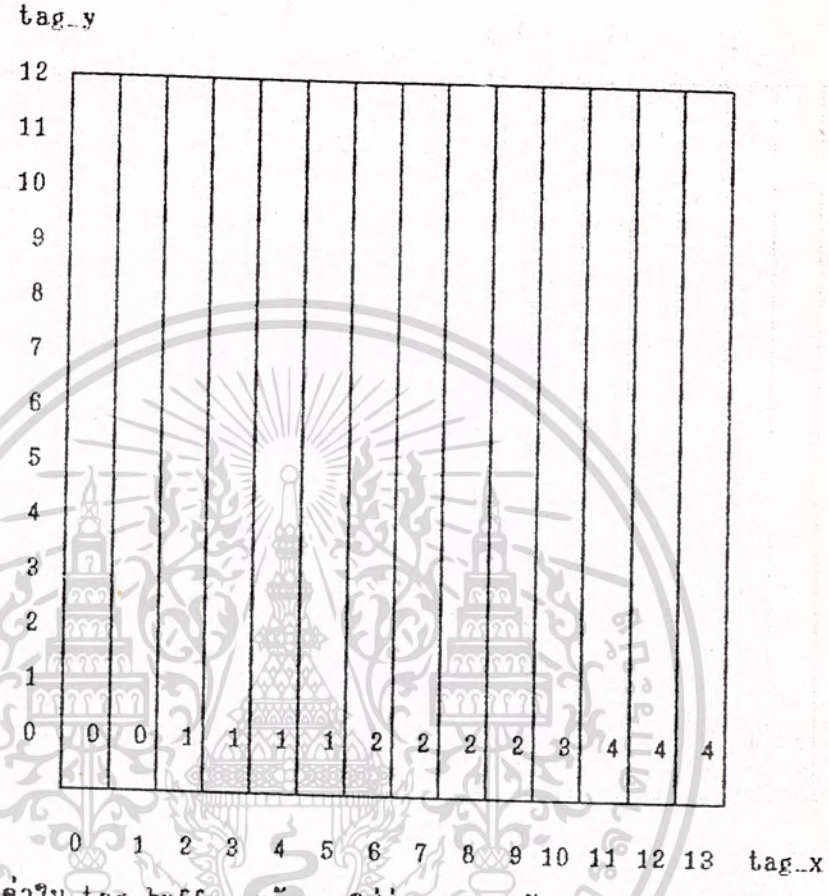
จากรูปที่ 2.5 เราจะทำการเริ่มที่ row ที่ 1โดยการเติมหมายเลขวัตถุลงใน tag buffer สำหรับ dixel แรกจะให้เท่ากับ "0" (tag[0][0]=0) ในขณะเดียวกันเราจะย้ายการมอง row มาอีกส่วนหนึ่งที่คู่กันของ pixelsในภาพ screen ของรูปที่ 2.5 เมื่อ current pixel อยู่ที่ tag-x=0 และ tag-y=0 (ตำแหน่ง(0,0))เราทำการเปรียบเทียบสีกับตำแหน่งที่ (1,0) จากรูปที่ 2.5 จะเห็นได้ว่าที่ตำแหน่ง (0,0)เป็นสีน้ำเงินและสีที่ตำแหน่ง (1,0) ก็เป็นสีน้ำเงินเหมือนกันซึ่งเป็นวัตถุเดียวกัน ดังนั้นหมายเลขวัตถุที่ใส่ในtag bufferคือให้ tag[1][0] =tag[0][0]=0 แต่เมื่อย้าย current pixel มาอยู่ที่จุด (1,0) แล้ว check ค่าสีที่จุด(2,0) เป็นสีแดง ในขณะที่ตำแหน่ง(1,0)เป็นสีน้ำเงิน ดังนั้นเราจะพบว่า pixel นี้เป็นของวัตถุใหม่ ดังนั้น tag[2][0]=objnum=1 (objnum=objnum+1) แล้วเราก็ทำตามขั้นตอนเหมือนกันนี้ จนกระทั่งถึง tag-x = x size - 1

Pseudocode สำหรับเติมค่า number วัตถุใน row แรกของ tag buffer เป็นดังนี้

```
/*=================================================================*/
    /* default : first pixel of row 1 belongs to object 0 */
    x = 0, y = 0, tag[0][0] = 0, objectnumber = 0
    /* move across row 1 */
       FOR (x = 0 TO x < (xsize - 1))
          BEGIN
          /* look at pixel pair */
          IF (color at position (x+1,y) + color at position (x,y)
             BEGIN
             /* same object, riht pixel gets same object number */
             tag[x+1][y] = tag[x][y]
             END
        ELSE
            BEGIN
           /*different object,right pixel gets new object number */
           objectnumber = objectnumber + 1
           tag[x+1][y] = objectnumber
           END
        x= x+1
        END
/*=================================================================*/
```

ALGORITHM : FILLING FIRST ROW OF TAG BUFFER

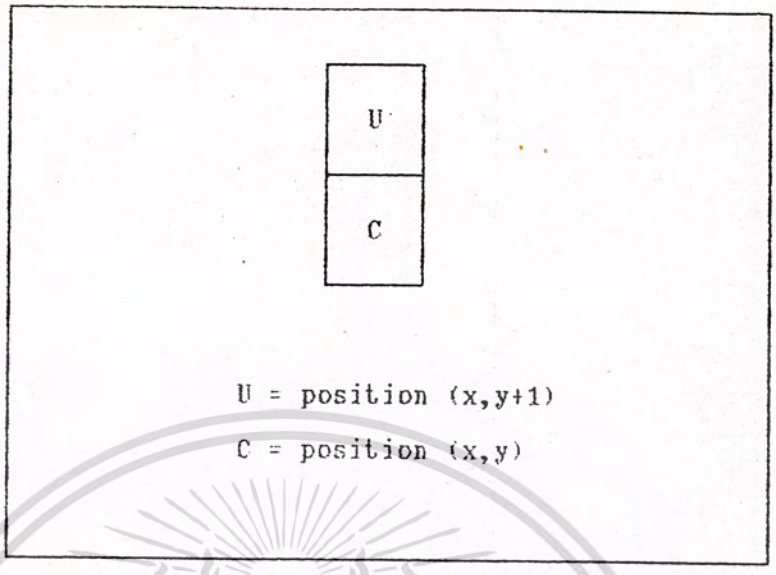
ดังนั้นผลที่ได้จากการเติมค่าหมายเลขวัตถุใน row แรกของ tag buffer จะได้ดังรูปที่ 2.6



รูปที่ 2.6 ค่าใน tag buffer หลังจากใส่ค่าหมายเลขวัตถุใน row แรกแล้ว

ขั้นตอนที่สอง

การใส่ค่าหมายเลขวัตถุในcolumnแรก ซึ่งใช้หลักการเดียวกันกับใน rowแรกจากรูปที่2.7 ใช้ในการอธิบายถึงหลักการใส่ค่าหมายเลขวัตถุใน column แรกใน tag buffer โดยพิจารณา ที่ current pixel (ที่ตำแหน่ง C)ใน screen และ pixel บน (ที่ตำแหน่ง U) ถ้า pixel ทั้งสองมีค่าสีเดียวกันแสดงว่า upper pixel มีหมายเลขวัตถุเดียวกัน กับ current pixel แต่ถ้า upper pixel มีสีที่ต่างกันก็จะให้ค่าหมายเลขของวัตถุใหม่ใน tag buffer แล้วก็ทำตามขั้นตอนดังที่กล่าวมาข้างต้นต่อเนื่องกันใน columnที่ 1 จนถึง tag-y = y size-1 จึงหยุด

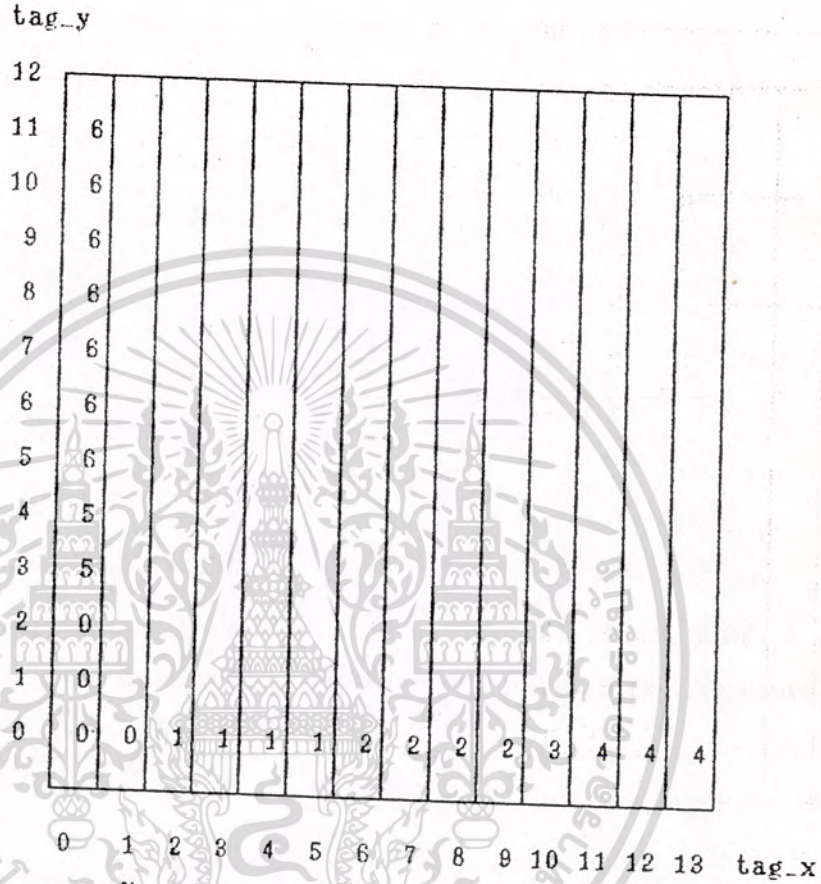


รูปที่ 2.7 แสดงถึง pixel 1 คู่ใน columnแรก

Pseudocode สำหรับการเติมค่าหมายเลขวัตถุใน column แรกของ tag buffer

```
/*=============================================================*/
    x = 0
    /*  move across column 1 */
    FOR (y=0 TO y < (ysize - 1))
       BEGIN
       /* look at pixel pairs */
       IF (color at position (x,y+1 = color at position (x,y))
          BEGIN
          /* same object, upper pixel gets same object number */
          tag[x][y+1]= tag[x][y]
          END
       ELSE
          BEGIN
         /* different object, upper pixel gets new object number */
          objectnumber= objectnumber+1
          tag[x][y+1]= objectnumber
          END
       y= y+1
    END
```

ดังนั้น ผลที่ได้จากการเติมค่าหมายเลขวัตถุใน column แรกใน tag buffer จะได้ดังรูปที่ 2.8



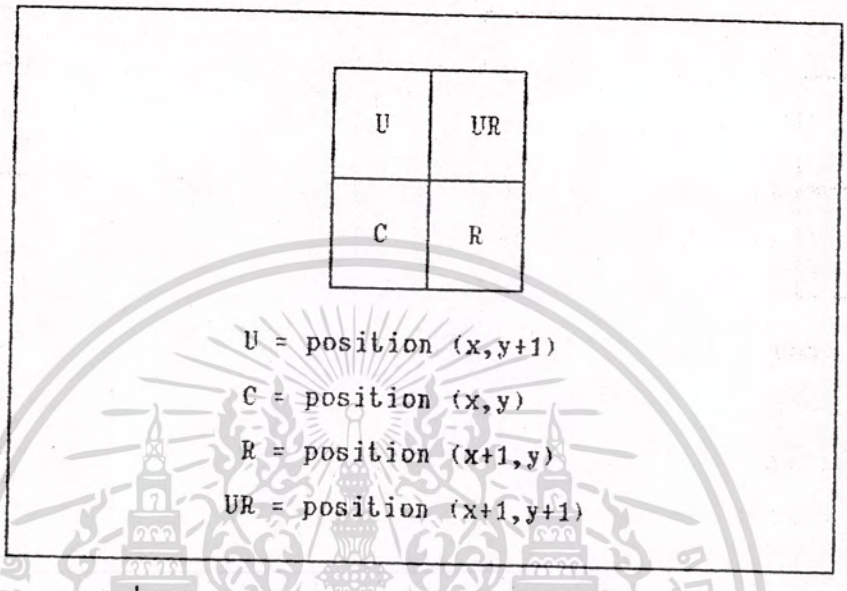
รูปที่ 2.8 ค่าหมายเลขวัตถุใน tag buffer ของ column แรกและ row แรก

ขั้นตอนที่ 3

เป็นการใส่ค่าหมายเลขวัตถุในส่วนที่เหลือของ tag buffer โดยการพิจารณาจาก 4 pixel ดังรูปที่ 2.9 ซึ่งเรารู้ค่าหมายของวัตถุใน row แรกและ column แรกแล้วเมื่อเรานำ 4 pixelไปแทนที่ โดยให้ current pixel(C) อยู่ที่ตำแหน่ง (0,0) ใน tag buffer ซึ่งจะทำให้เรารู้ค่าหมายเลขวัตถุถึง 3 pixelแล้ว คือ current pixel(C),upper pixel (U) และ rightpixel(R) ซึ่งเราจะต้องหาค่าสีของ upper right pixel(UR) เท่านั้น ถ้าสีของ pixel C,U,R และ pixel UR เป็นสีเดียวกันแสดงว่า pixel UR เป็นวัตถุเดียวกับ pixel ทั้งสามซึ่งก็จะมีหมายเลขวัตถุเดียวกัน แต่ถ้า pixel UR ไปเป็นขอบของ pixel C,U หรือ R เช่นนี้ pixel UR ก็คือวัตถุใหม่และมีหมายเลขวัตถุใหม่ใน tag buffer นั้นเอง

ตัวอย่างของการเติมส่วนนี้เหลือใน tag buffer โดยใช้ข้อมูลในรูปที่ 2.5 โดยเริ่มเลื่อน rowล่างที่ 4 pixels ที่เวลานั้นในข้อมูลภาพในรูปที่ 2.5โดยเราเริ่มต้นด้วย current pixelC ที่ตำแหน่ง tag-x=0 และ tag-y=0 ซึ่งเรารู้ค่าหมายเลขของวัตถุ pixel C,U, และ R แล้ว

ดังนั้นเราจะหาค่าสี pixel UR โดยการเปรียบเทียบสีที่ตำแหน่ง U กับ UR ซึ่งเป็นสีน้ำเงินและค่าที่เก็บใน tag buffer ก็คือ tag [1][1]=tag[0][1]=0 ส่วนในกรณีที่ pixel UR มีค่าแตกต่างจาก pixel อื่น จะแสดงให้เห็นถึง pseudocode ข้างล่าง



รูปที่ 2.9 แสดงถึง pixel 4 pixels ที่เติมลงใน tag buffer

Pseudocade ของ algorithm สำหรับเติมส่วนที่เหลือของ tag buffer เป็นดังนี้

```
/*=============================================================*/
    FOR(y = 0 TO y < (ysize-1))
       BEGIN
       /* move across a row */
       FOR(x = 0 TO x < (xsize-1))
          BEGIN
         IF (color at postition U = color at postion UR)
            BEGIN
        /* same object, upper right pixel gets same object number */
            tag[UR]= tag[u]
            END
        ELSE IF (color at position c = color at position UR)
            BEGIN
       /* same object, upper right pixel gets same object number */
            tag[UR]= tag [c]
            END
       ELSE IF (color at position R = color at position UR)
```

```
        BEGIN
    /* same object, upper right pixel gets same object number */
        tag[UR]= tag[R]
        END
    ELSE
        BEGIN
    /* differentobject, upper right pixel getsnewobject number */
        objectnumber= objectnumber+1
        tag[UR]= objectnumber
        END
    x= x+1
    END
y= y+1
END
/*=================================================================*/
```

ผลที่ได้จาก algorithm ในรูปที่ 2.5 ให้ค่าใน tag buffer ดังต่อไปนี้

tag_y														
12														
11	6	6	6	6	6	6	6	6	6	6	6	6	6	6
10	6	6	6	13	14	14	14	14	13	15	15	15	15	15
9	6	13	13	13	13	13	13	13	13	13	6	6	6	6
8	6	6	10	10	10	6	6	12	12	12	6	6	9	8
7	6	6	10	10	6	6	6	6	6	11	6	6	9	8
6	6	6	10	6	6	6	6	6	6	11	6	6	9	8
5	6	6	6	6	6	6	6	6	6	6	6	6	9	8
4	5	0	0	0	0	1	1	8	8	8	8	8	8	8
3	5	0	0	0	1	1	1	1	7	7	7	7	7	7
2	0	0	0	0	1	1	1	1	1	2	2	3	4	4
1	0	0	0	1	1	1	1	2	2	2	2	3	4	4
0	0	0	1	1	1	1	2	2	2	2	3	4	4	4
	0	1	2	3	4	5	6	7	8	9	10	11	12	13 tag_x

รูปที่ 2.10 ค่าใน tag buffer สำหรับรูปที่ 2.5

จากรูปที่ 2.10 จะสังเกตุเห็นได้ว่าภาพใน tag buffer นั้น แตกต่างไปจากภาพเดิมในรูปที่ 2.5 ซึ่งภาพในรูปที่ 2.5 จะถูกแบ่งแยกออกเป็นภาพเล็ก ๆ ใน tag buffer ซึ่งปัญหานี้เกิดจากกรรมวิธีใส่ค่าหมายเลขวัตถุใน tag bufferในส่วนแรกอธิบายถึงลำดับการบรรจุ first row, first column และส่วนที่เหลือของ tag buffer ในการพิจารณาเราพิจารณาเพียงส่วนเล็ก ๆ ของภาพ ที่ขณะนั้นซึ่งส่วนใหญ่จะใช้ 4 pixel และบางครั้งใช้ 2 pixel ดังเหมือนกับภาพ 2.2 ที่ได้กล่าวไว้ตอนต้นบทนี้ ส่วนรูปร่างของวัตถุในภาพที่ 2.5 มันยากต่อการแบ่ง pixel เป็นขอบเขตของวัตถุเมื่อใช้ช่องหน้าต่างเล็ก ๆ ในการมองภาพ แต่ว่าเราทราบปัญหาในส่วนที่ 1 และไม่ครอบคลุม pixel ที่แบ่งแยก ซึ่งเป็นของขอบเขต ความสัมพันธ์นี้ถูกจัดเก็บไว้ใน data structure ที่เรียกว่า"F-TAB" ตาราง F-TAB คือโครงสร้างของการบรรจุส่วนที่เหลือใน tag buffer โดยใช้ 4 pixel ของรูปที่ 12 หลังจากขอบเขตวัตถุซึ่ง pixel UR ที่กำหนดไว้ ทำให้เรามองเห็นสีของ pixel ใน window ถ้าสีของ pixel เหล่านี้มีสีเหมือนกันก็ยังคงกำหนดใน tag buffer เหมือนการแบ่งแยกของวัตถุ แต่เราทำอย่างนี้โดยเปรียบเทียบสีของ pixel C ด้วย pixel U ในภาพ screen และถ้า pixel ทั้งคู่เป็นสีเดียวกันและมีหมายเลขวัตถุต่างกัน ดังนั้นเราก็เก็บข้อมูลนี้ไว้ใน F-TAB table และทำการเปรียบเทียบอย่างเดียวกันกับ pixel U และ R แล้วเก็บข้อมูลไว้ใน F-TAB แต่เมื่อพบว่า pixel ทั้งสอง เป็นสีเดียวกันและยังคงเป็นวัตถุที่ต่างกันใน tag bufferเราทราบจุดนี้ที่พิจารณาว่า วัตถุทั้งสองเหมือนกันเป็นวัตถุเดียวกัน (actually on object)ตอนนั้นเราบันทึกหมายเลขวัตถุที่ต่ำสุดไว้ใน F-TAB สำหรับวัตถุทั้งสอง เมื่อเราพิจารณาวัตถุทั้งหมดใน F-TAB และให้วัตถุทั้งหมดที่มีความสัมพันธ์กันนี้เหมือนกับวัตถุที่ต่ำสุด ขั้นตอนที่ต่อเนื่องนี้เหมือนกับ 4 pixel window ถูกย้ายใน screen และบรรจุใน tag buffer เมื่อวิธีนี้เสร็จสมบูรณ์ในส่วนที่ 1 แล้ว และถ้าการเข้าไปทำใน F-TAB บ้างแล้ว ในส่วนที่สองจำเป็นที่จะต้องปรับปรุงขอบเขตทั้งหมดของวัตถุที่ต่ำสุดใน tag buffer

ตัวอย่างต่อไปนี้ใช้ข้อมูลในรูปที่ 2.5 และการโต้ตอบกันใน tag buffer ของรูปที่ 2.10 ถึงการอธิบายที่ใช้ตารางของ F-TAB

รูปที่ 2.10 คือ tag buffer สำหรับภาพของรูป 2.5 หลังจากทำตามขั้นตอนที่ 1 เสร็จเรียบร้อยแล้วก็เก็บค่าที่ได้ไว้ใน tag buffer จากการใช้ 2 และ 4 pixel window ในการแสดงผลมีการค่า error ขึ้นใน tag bufferเช่นเดียวกับวัตถุจำนวนมากกว่าที่แสดงผลใน tag buffer อยู่ในรูปที่ 2.5 ขอบเขตวัตถุน้ำเงินของรูปที่ 2.5 แสดงใน tag buffer ด้วยค่าตัวเลขของ 0,2,4,6,7,8,14 และ 15 ขอบของสีชมพู แสดงด้วยค่าตัวเลข 10,11,12,และ 13 เหมือนตอนต้น ๆ เทคนิคนี้สำหรับบรรจุใน tag buffer ข้อมูลที่ไม่ถูกต้องจะปรากฏให้เห็นใน tag buffer แต่algorithm จะทราบปัญหานี้ได้และจะเก็บการแก้ไขใน F-TAB tableเพื่อเหตุนี้ด้วย

Algorithm ต่อไปนี้คือ pseudocode ที่รวมการใช้ของ F-TAB เพื่อเติมส่วนที่เหลือใน tag buffer

```
/*=================================================================*/
     FOR(y = 0 TO y < (ysize-1))
        BEGIN
        FOR(x = 0 TO x < (xsize-1))
           BEGIN
           IF (color at position U = color at position c)
              AND (tag[U] != tag[c])
              in F_TAB we note that value stored in tag [U]
              and value stored in tag[c] are related
           IF (color at position U = color at position R)
              AND (tag[U] != tag [R]
              in F_TAB we note that value stored in tag [R]
              and value stored in tag[U] are related
          IF (color at position U = color at position UR)
              tag[UR]= tag[U]
          ELSE IF (color at position C = color at position UR)
              tag[UR]= tag[C]
          ELSE IF (color at position R = color at position UR)
              tag[UR] = tag[R]
          ELSE
              BEGIN
         /* differentobject,upper right pixed getsnewobject number*/
              objectnumber= objectnumber+1
              tag[UR]= objectnumber
              END
          x= x+1
          END
      y= y+1
      END
/*=================================================================*/
ALGORITHM : INCLUDING THE F_TAB STRUCTURE WHEN
            FILLING THE REST OF THE TAG BUFFER
```

การประยุกต์ใช้ของขั้นตอนข้างบนกับภาพในรูปที่ 2.5 จะทำการเติม ตาราง F_TAB ตามค่าต่อไปนี้

```
F_TAB[ 0] =  0
F_TAB[ 1] = -1
F_TAB[ 2] =  0
F_TAB[ 3] = -1
F_TAB[ 4] =  0
F_TAB[ 5] = -1
F_TAB[ 6] =  0
F_TAB[ 7] =  0
F_TAB[ 8] =  0
F_TAB[ 9] = -1
F_TAB[10] = 10
F_TAB[11] = 10
F_TAB[12] = 10
F_TAB[13] = 10
F_TAB[14] =  0
F_TAB[15] =  0
```

รูปที่ 2.11 ส่วนประกอบของ F_TAB สำหรับภาพในรูปที่ 2.5 และ Tag buffer ของรูปที่ 2.10

ทุกตำแหน่งในตาราง F_TAB จะถูกกำหนดค่าเริ่มแรกเป็น -1 และค่าใน F_TAB[X] =-1 หมายความว่าวัตถุหมายเลข X ไม่ได้มีความสัมพันธุ์กับวัตถุหมายเลขที่อื่นๆ และค่าของ F_TAB[0]=F_TAB[2]=F_TAB[4]=F_TAB[6]=F_TAB[7]=F_TAB[8]=F_TAB[14]=F_TAB[15] = min(0,2,4,6,7,8,14,15) แสดงให้เห็นว่าวัตถุเหล่านี้มีความสัมพันธุ์กันและเป็น region เดียวกัน ค่าของ F_TAB[10]=F_TAB[11]=F_TAB[12]=F_TAB[13]=min(10,11,12,13) ก็มีความสัมพันธุ์ในลักษณะเดียวกัน

ในส่วนที่2 จะทำให้เกิดการเติมค่าใน tag buffer ใหม่อีกครั้ง ในตำแหน่งของจุด ซึ่งเป็นของวัตถุเดียวกัน แต่มีหมายเลขวัตถุต่างกัน ให้มีหมายเลขวัตถุเดียวกัน ซึ่งจะทำให้ได้จำนวนวัตถุน้อยที่สุด ดังที่เราได้ทราบแล้วจากตัวอย่างข้างบนว่า วัตถุหมายเลข 0,2,4,6,7,8,14 และ 15

มีความสัมพันธ์กัน เพราะฉนั้นเมื่อไหร่ที่เราพบวัตถุหมายเลข 0,2,4,6,7,8,14 หรือ 15ใน tag buffer เราก็จะแทนที่ด้วยค่า 0 ซึ่งเป็นหมายที่ต่ำสุดและในทำนองเดียวกัน เมื่อไหร่ที่เราพบวัตถุหมายเลข 11,12 หรือ 13 เราก็แทนที่ด้วยค่า 10 ซึ่งเป็นหมายที่ต่ำสุด

ถ้าหมายเลขที่ถูกเก็บไว้ใน F_TAB[X] มีค่าเป็น -1 หมายความว่าวัตถุหมายเลข X ไม่มีความสัมพันธ์กับวัตถุอื่นๆเลย ตัวอย่างเช่น จากรูปที่ 2.5 F_TAB[1] = -1 มีความหมายว่าค่าทั้งหมดใน tag buffer ที่มีค่าหมายเลขวัตถุเป็น 1 เป็น region ที่มีสีแตกต่างจาก region ที่มีค่าหมายเลขวัตถุเป็น 0 หรือ region ที่มีค่าหมายเลขวัตถุเป็น 6 หรือ 8 หรือ 2

ขั้นตอนต่อไปนี้จะแสดงว่าเราจะหาจุดเริ่มต้นและจะทำการเติมค่าใน tag buffer ใหม่อีกครั้งหนึ่ง

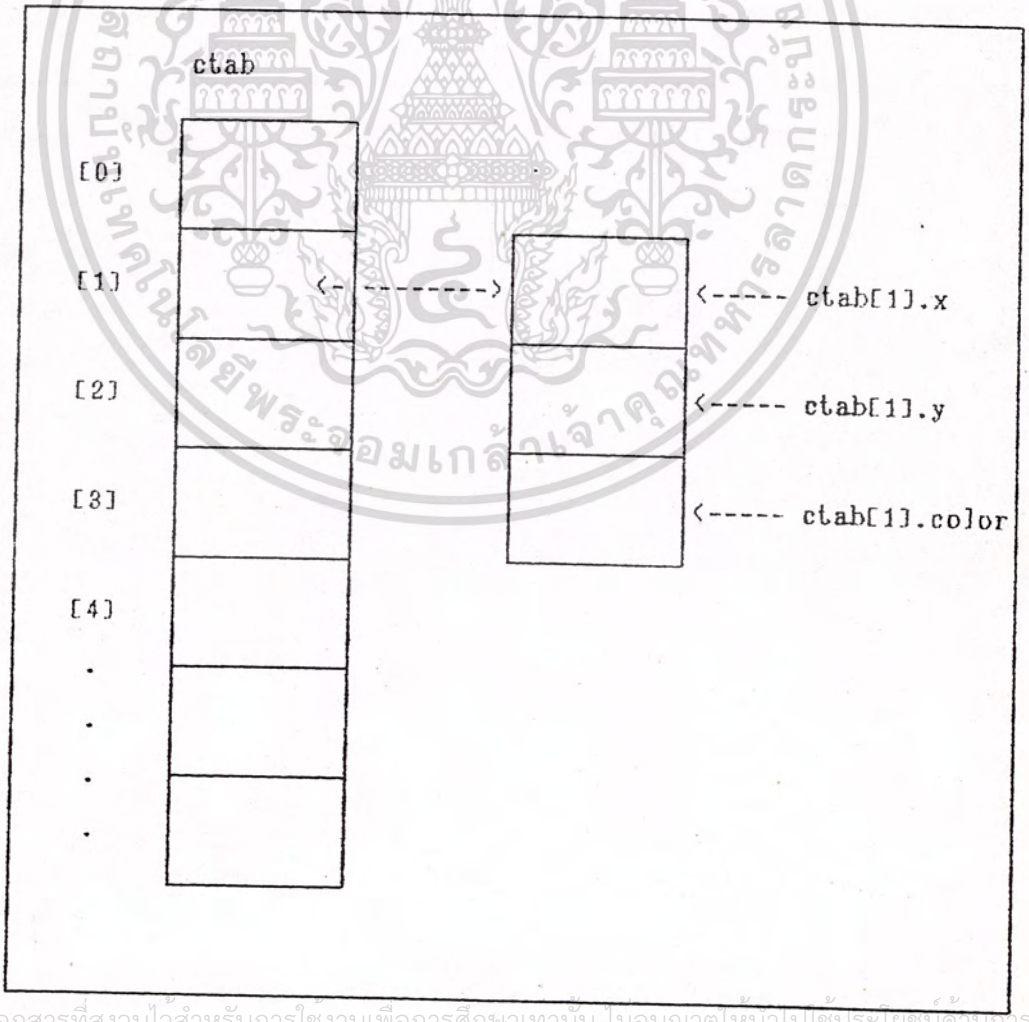
```
/*=================================================================*/
    j = -1
    FOR (y=0 to (y < ysize))
        BEGIN
        /* moving across a row */
        FOR (x=0 to (x < xsize))
            BEGIN
        /* check that this objnum is not related to anyother objects
           or it is the lowest leftmost pixel of the region */
        IF (F_TAB[tag[x][y]] = -1 OR (tag[x][y] = F_TAB[tag[x][y]])
             BEGIN
             j =j+1
             /* store a starting point in ctab structure */
             ctab[j].x=x
             ctab[j].y=y
             ctab[j].color= color at position(x,y)
             F_TAB[tag[x][y]] = -2
             END
        ELSE IF (F_TAB[tag[x][y]] > -1
             BEGIN
             /* refilling the tag buffer */
             tag[x][y] = F_TAB[tag[x][y]]
             END
        x=x+1
```

```
        END
    y=y+1
    END
/*=================================================================*/
```

ALGORITHM : ส่วนที่ 2 เป็นการสร้าง CTAB และเติมค่าใน TAB BUFFER อีกครั้งด้วย หมายของวัตถุที่ถูกต้อง

หมายเหตุ : (ctab [J].X, ctab [J].Y) เป็น Pixel ที่อยู่มุมซ้ายล่างของ object number = tag [ctab[j].x] [ctab [j].y]

ขั้นตอนข้างบนใช้โครงสร้างข้อมูลที่เรียกว่า ctab ctab นี้ จะใช้เก็บจุดเริ่มต้นทั้งหมดของ regions ใน image และสีของแต่ละ region ใน image รูปที่ 2.12 แสดงโครงสร้างของ ctab และข้อมูลสำหรับหมายเลขวัตถุที่ 1



รูปที่ 2.12 แสดงรายละเอียดของโครงสร้าง ctab

ผลของการประยุกต์ขั้นตอนส่วนที่ 2 กับ tag buffer ในรูปที่ 2.10 จะได้ปรับปรุง tag buffer ดังนี้

tag_y																
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	0	0	0	10	0	0	0	0	10	0	0	0	0	0	0	
9	0	10	10	10	10	10	10	10	10	10	0	0	0	0	0	
8	0	0	10	10	10	0	0	10	10	10	0	0	9	0	0	
7	0	0	10	10	0	0	0	0	0	10	0	0	9	0	0	
6	0	0	10	0	0	0	0	0	0	10	0	0	9	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	
4	5	0	0	0	0	1	1	0	0	0	0	0	0	0	0	
3	5	0	0	0	1	1	1	1	0	0	0	0	0	0	0	
2	0	0	0	0	1	1	1	1	1	0	0	3	0	0	0	
1	0	0	0	1	1	1	1	0	0	0	0	3	0	0	0	
0	0	0	1	1	1	1	0	0	0	0	3	0	0	0	0	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	tag_x

รูปที่ 2.13 tag buffer ของรูปที่ 2.10

หลังจากผ่านขั้นตอนส่วนที่ 2 จะปรับปรุงข้อมูลที่ไม่ถูกต้อง

ขั้นตอนที่ 2 จะสร้างโครงสร้าง ctab สำหรับ image ในรูปที่ 2.5 และ tag buffer ในรูปที่ 2.13 ด้วย ดังรูปที่ 2.14

ctab[j]:

J	X	Y	COLOR
0	0	0	blue
1	2	0	red
2	10	0	green
3	0	3	orange
4	12	5	yellow
5	2	6	pink

รูปที่ 2.14 ค่าต่างๆ ใน ctab

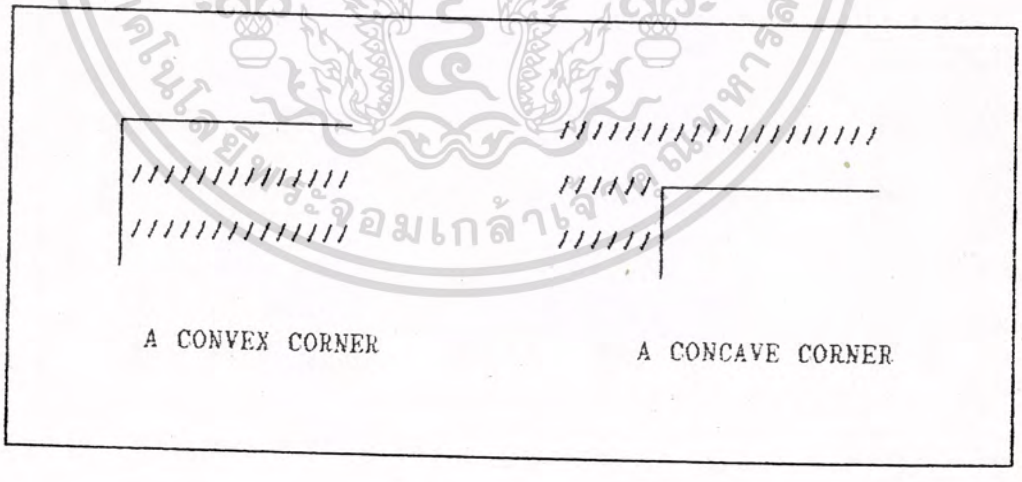
การประยุกต์ใช้ขั้นตอนส่วนที่ 1 และส่วนที่ 2 ถือเป็นการสำเร็จขั้นแรกของกระบวนการ antialiasing และในขณะนี้ภาพวัตถุ เริ่มแรกได้ถูกแทนค่าลงใน tag buffer ซึ่ง regions ของ pixels ได้ถูกกำหนดและใน ctag table ซึ่งจุดเริ่มต้นบนขอบของแต่ละวัตถุได้ถูกกำหนดเช่นเดียวกับสีของแต่ละ regions ซึ่งเมื่อมีมูลต่างๆ ได้ครบแล้ว กระบวนการ antialiasing สามารถดำเนินการได้

บทที่ 3 การติดตามขอบภาพ

การติดตามขอบภาพของวัตถุซึ่งเป็นขั้นตอนที่ 2 ของกระบวนการ antialiasing โดยใช้ tag buffer ที่ได้อธิบายไว้ในบทที่ 2 เราได้กำหนดให้ Pixels ใน image ซึ่งเป็นของแต่ละวัตถุ และได้กำหนด pixels ที่อยู่ทางด้านซ้ายล่างของแต่ละวัตถุเป็นจุดเริ่มต้นของวัตถุ

ลำดับขั้นต่อไปคือกำหนดตำแหน่งของรอยหยัก(jaggies) ตามขอบของแต่ละวัตถุ โดยการตรวจสอบ pixel ตามขอบของแต่ละวัตถุ และเก็บข้อมูลบางอย่างไว้ เช่น เกิดที่ไหน

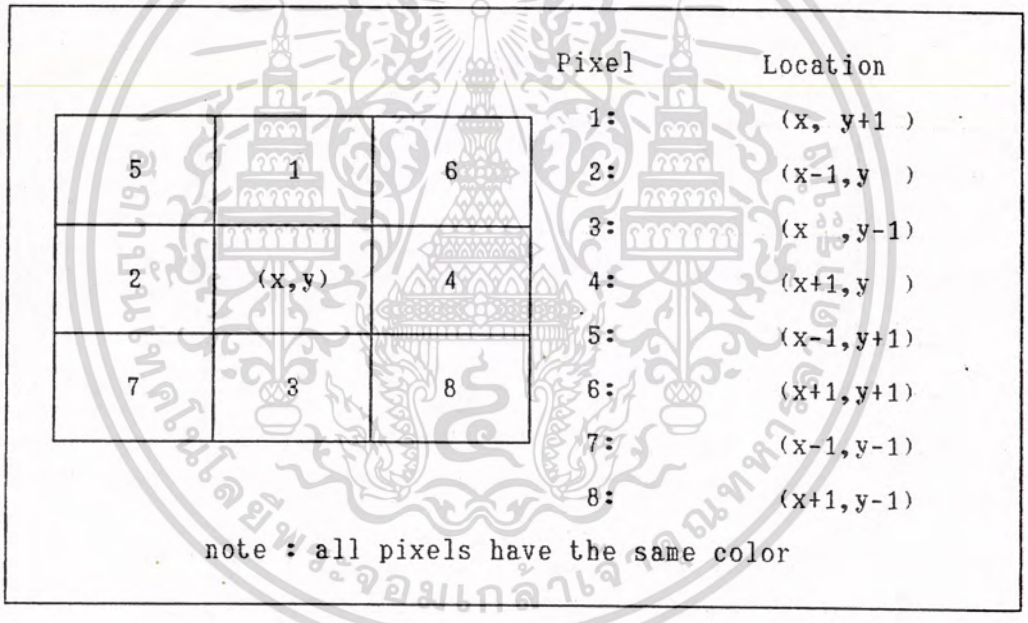
สำหรับในขั้นตอนที่ 3 นี้เราจะสนใจเกี่ยวกับการหาและเก็บตำแหน่งของจุดเริ่มต้นรอยหยัก และในขั้นตอนต่อไป เราจะทำการกำหนดค่าตำแหน่งของรอยหยักที่จะทำการ antialiaing มันตำแหน่งซึ่งเป็นจุดเริ่มต้นของรอยหยัก จะหาได้โดยทำการ trackingไปตามขอบของวัตถุ และไม่มี Pixel ณ ตำแหน่งที่ Pixelพบกันเป็นมุม ณ ตำแหน่งที่เกิดขึ้นตามขอบของวัตถุจะเรียกเป็น corner point ซึ่งจะประกอบด้วย2 ลักษณะคือ มุมโค้งเข้า(convex corner)และ มุมโค้งออก (concave corner)ดังแสดงในรูปที่ 3.1 เราจะเก็บข้อมูลของความเว้า (Concavity) ตามตำแหน่งของแต่ละ Corner point ตามขอบของวัตถุเพราะว่าข้อมูลเหล่านี้ จะทำให้เราสามารถกำหนดตำแหน่งของรอยหยัก ที่จะทำการ antialising ต่อไป.



รูปที่ 3.1 : มุมทั้ง 2 ชนิด

ขั้นตอนที่จะกล่าวต่อไปนี้จะอธิบายว่า เราจะทำการ track ตามขอบนอกของวัตถุ และทำการกำหนดจุด corner point ของแต่ละวัตถุอย่างไรนั้น ซึ่งขั้นตอนนี้มีพื้นฐานอยู่บนขั้นตอนของการ tracking ซึ่งการกำหนดขอบนั้น ขึ้นอยู่กับจำนวน 4 pixel ซึ่งอยู่ติดกัน และได้แสดงโดย Bloomenthal ใน " Edge Inference with Applications to Antialiasing"

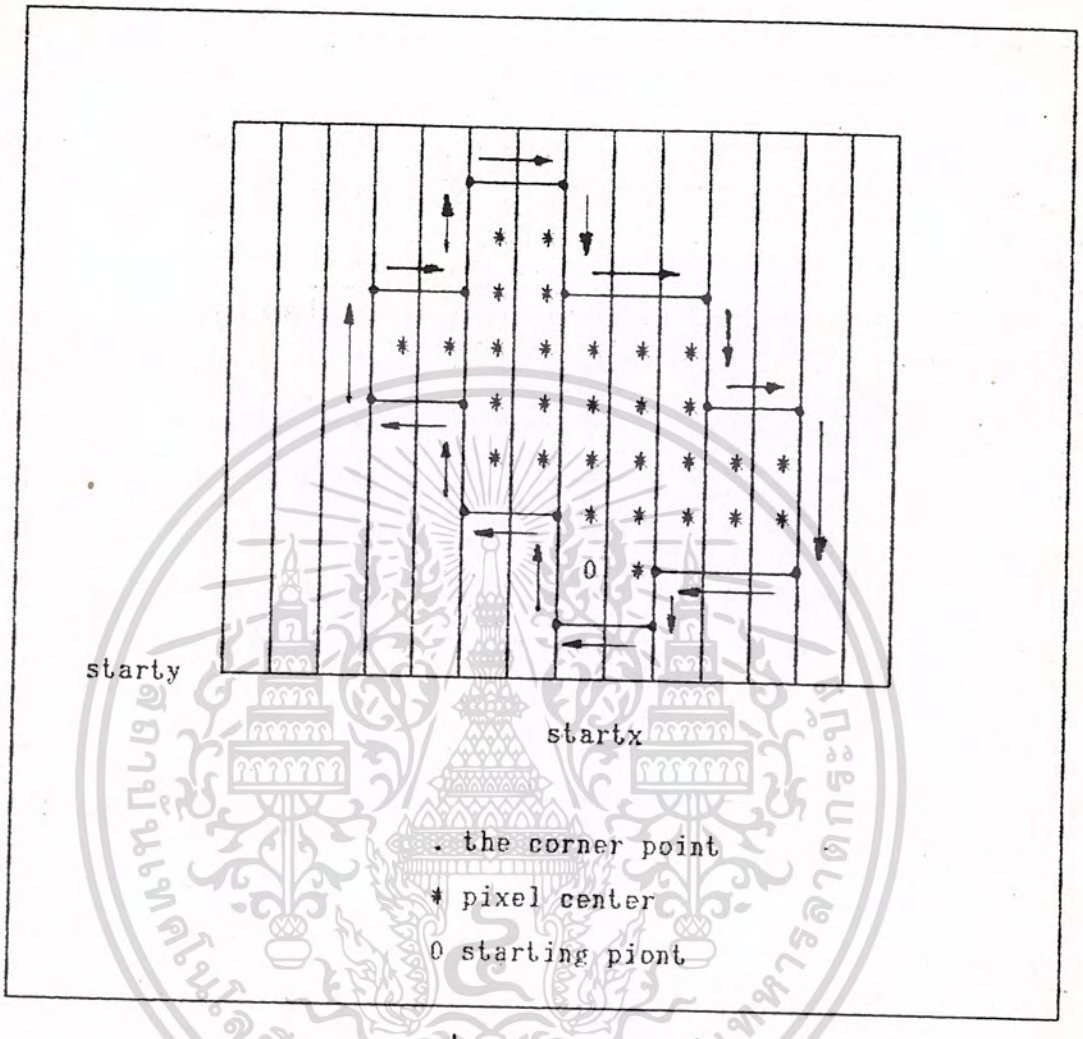
จากหนังสือ "Computer Graphics", Volume 17, Number 3 ต่อไปนี้จะการปรับปรุงทฤษฎี ของ Bloomenthal's algonithm, ซึ่งจะใช้ pixel 8 pixel ที่ติดกันในการพิจารณา ดังแสดงในรูปที่ 3.2 กำหนดให้ 8 pixel ที่ติดกับ Pixel (x,y) คือ Pixel 1,2,3,4,5,6, 7 และ 8 เป็น Pixel ที่อยู่ติดกันกับ Pixel ที่ตำแหน่ง (x,y) และเป็นสีเดียวกันทั้งหมดนั้น มีความสำคัญมากในการบันทึกค่าของ pixel ที่ 1 ถึง pixel 8 และ pixel(x,y)ไว้ แต่ถ้ามีเพียง 1 ใน 8 pixel เท่านั้นที่มีสีเดียวกันกับ pixel(x,y) นั้น ในวิธีการ tracking ขอบภาพของวัตถุเราจะทำการ tracing บนขอบซึ่งกำหนดโดยใช้ 8 pixel ซึ่งติดกัน และวิธีการนี้จะยอมให้เราทำการ antialiaing วัตถุที่แคบๆ โดยไม่จำเป็นต้องมีการกำหนดขอบของวัตถุโดย 4 regions ซึ่งติดกัน (ตัวอย่างเช่น เส้นที่มีขนาดความกว้างเท่ากับ 1)



รูปที่ 3.2 : แสดงภาพของ 8 pixels ที่ติดกัน

เมื่อ regions ของ pixels ได้ถูกกำหนดว่าอันไหนเป็นวัตถุ จะมีประโยชน์ในการกำหนดจุดเริ่มต้นของการ tracking ขอบวัตถุ สำหรับแต่ละวัตถุจะมีจุดเริ่มต้นหลายแห่ง แต่เราต้องการเพียงแห่งเดียวเท่านั้นในการ tracking ของขอบภาพ จุดเริ่มต้นที่ใช้คือ Pixel ที่อยู่มุมซ้ายล่างสุดของวัตถุ ซึ่งก็คือจุดเริ่มต้นที่กำหนดขึ้นขณะที่สร้าง tag buffer นั่นเองเมื่อเรารู้จุดเริ่มต้นของขอบวัตถุเราก็สามารถทำการ trackingได้และหยุดการ tracking เมื่อพบจุดเริ่มต้นอีกครั้งหนึ่ง

ขอบของวัตถุจะถูก tracing จากจุดเริ่มต้นในทิศทางตามเข็มนาฬิกา รูปที่ 3.3 ใช้แสดงภาพของการ tracking

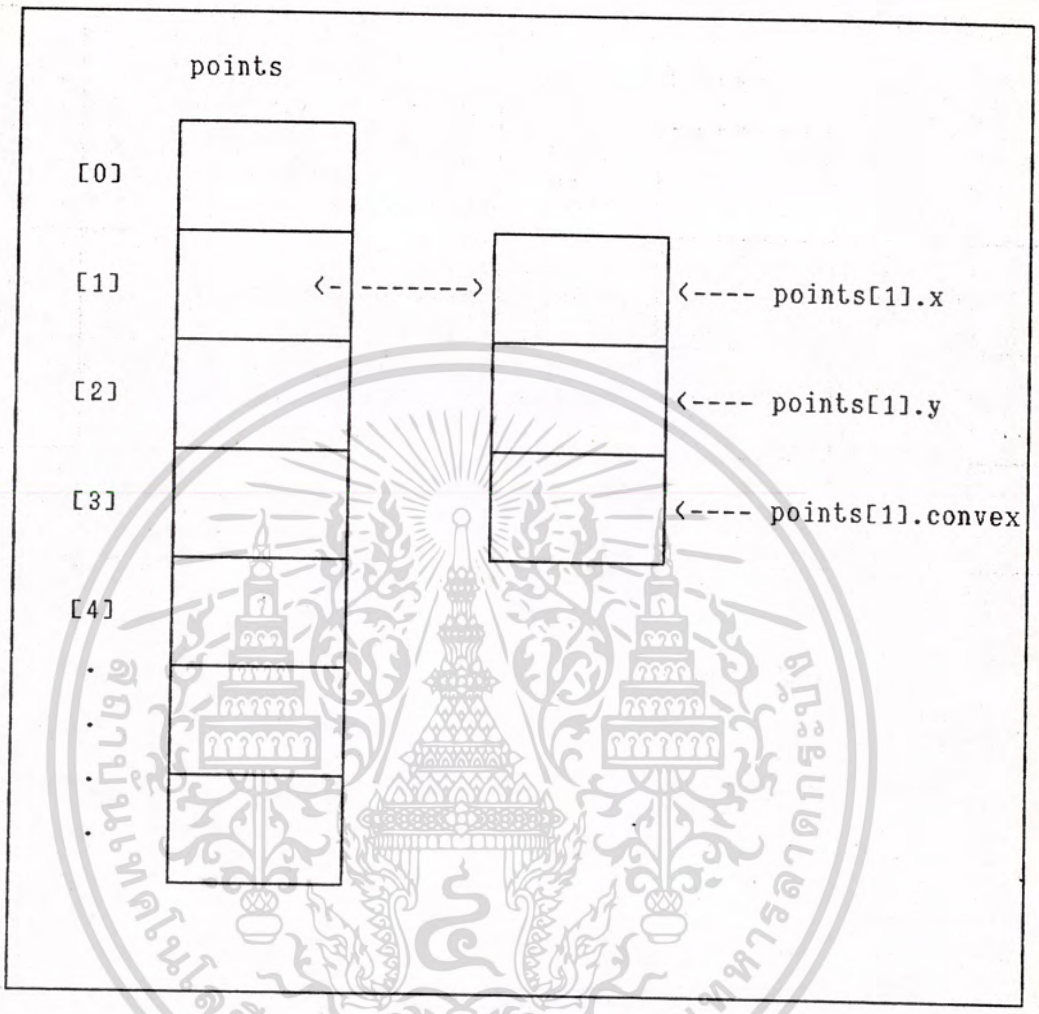


รูปที่ 3.3 : region ที่ติดกัน

จากรูปที่ 3.3 จะเห็นว่าจุดเริ่มต้น คือจุดที่อยู่มุมซ้ายล่างสุดของภาพ และการ track ขอบภาพจะกระทำตามเซ็มนาทิกาดังนี้

ขั้นแรกเราเริ่มที่จุดเริ่มต้น และเก็บตำแหน่งของจุดนี้เป็น corner pointตามด้วยความเว้าของมุม แล้วเราก็เคลื่อนย้ายไปตามขอบวัตถุที่ได้กำหนดไว้แล้วโดย 8 pixel ที่ติดกันจนกระทั่ง Pixel ที่ขอบชนกับ corner point อื่น, ข้อมูลของ corner point นี้จะถูกเก็บไว้ก่อนเรา สามารถเคลื่อนต่อไปตามเซ็มนาทิกา ตามขอบของวัตถุและเก็บข้อมูลของ corner point จนกระทั่งพบจุดเริ่มต้นอีกครั้งหนึ่งก็ถือว่าการ tracking ขอบวัตถุนั้นเสร็จสิ้นสมบูรณ์

ในขณะนี้ corner point ได้ถูกพบในขบวนการ track ขอบจะมีการเก็บข้อมูลร่วมกับแต่ละจุดมุม และเก็บไว้ในโครงสร้างที่เรียกว่า points โครงสร้าง points นี้จะใช้เก็บตำแหน่งของทุกๆ จุดมุมสำหรับวัตถุ ซึ่งได้ถูก track และความเว้า (concave หรือ convex)



รูปที่ 3.4 : รายละเอียดของโครงสร้าง points

รูปที่ 3.4 แสดงภาพว่าข้อมูลของจุดมุม ถูกจัดเก็บไว้อย่างไรในโครงสร้างข้อมูลแบบ points โครงสร้างข้อมูลแบบ points จริงๆแล้วสร้างขึ้นจากลำดับ (Array) ของโครงสร้างหลายๆ อัน ซึ่งแต่ละโครงสร้างจะเก็บตำแหน่งและความเว้าของจุดมุม ตำแหน่ง 0 เก็บข้อมูลจุดมุมของจุดเริ่มต้นของวัตถุ และตำแหน่งที่เหลือใช้สำหรับเก็บข้อมูลของแต่ละจุดมุมที่พบตามขอบวัตถุที่ถูก track ตามเข็มนาฬิกา

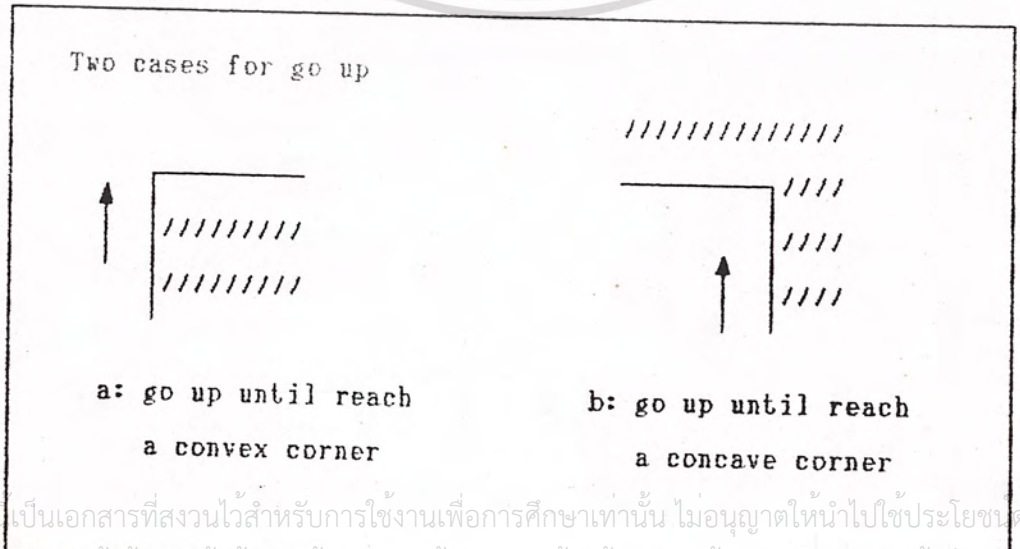
ในรูปที่ 3.4 เราแสดงว่าข้อมูลของจุมมุมถูกเก็บได้อย่างไร สำหรับจุดมุมที่ 1 ตำแหน่ง x และ y นโครงสร้างเก็บตำแหน่งของจุดมุม และในตำแหน่ง Convex ใช้แสดงความเว้าของจุดมุมซึ่งค่า 0 จะหมายถึง มุมโค้งออก (concave) และค่า 1 แสดงว่าเป็นมุมโค้งเข้า (Convex)

ในการที่จะแสดงว่าจะนำค่ามาเติมในโครงสร้าง point ได้อย่างไรนั้น เราจะแสดงด้วย pseudocodeของลำดับขั้นตอนในการ track ขอบภาพได้เตรียมไว้ในลักษณะของการช่วยในการ

พิจารณาการทำงานของขั้นตอน

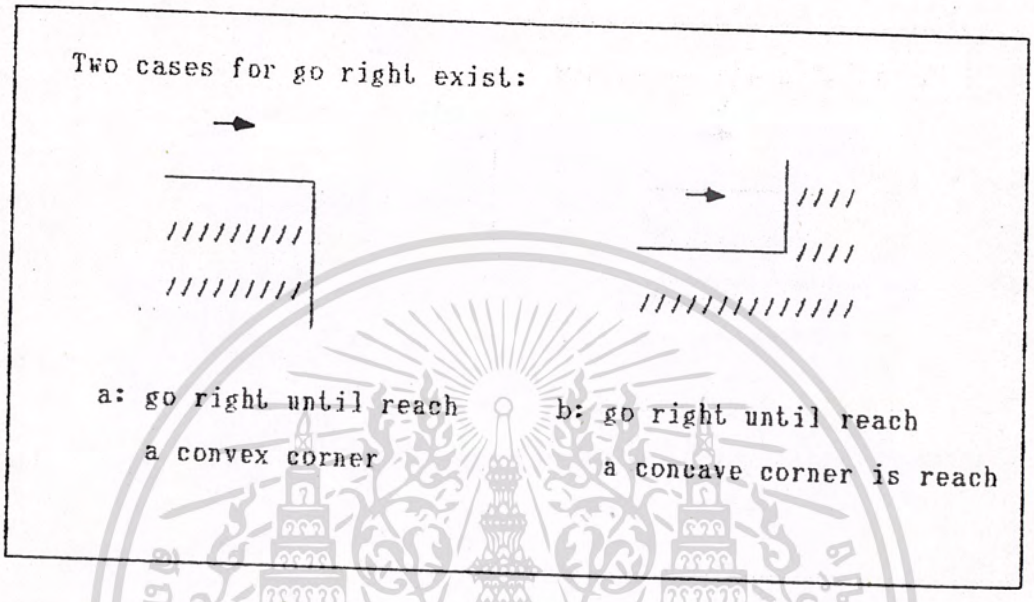
```
/*================== ALGORITHM EDGE TRRACKKING ======================*/
/*   grive a starting point startx,starty for an object region      */
     x = startx
     y = starty
/*  store first corner point  */
     save this corner point(startx,starty) and convex = 1 in point
     sturcture
/* objnum is the object number of a region that we are currently
   working with */
     objnum = tag[x][y]
     UL = tag[x-1][y+1]
      U = tag[x  ][y+1]
     UR = tag[x+1][y+1]
      L = tag[x-1][y  ]
      C = tag[x  ][y  ]
      R = tag[x+1][y  ]
     LL = tag[x-1][y-1]
      D = tag[x  ][y-1]
     LR = tag[x+1][y-1]
/*========================= GO UP ==================================*/
```

UL	U	UR
L	C	R
LL	D	LR



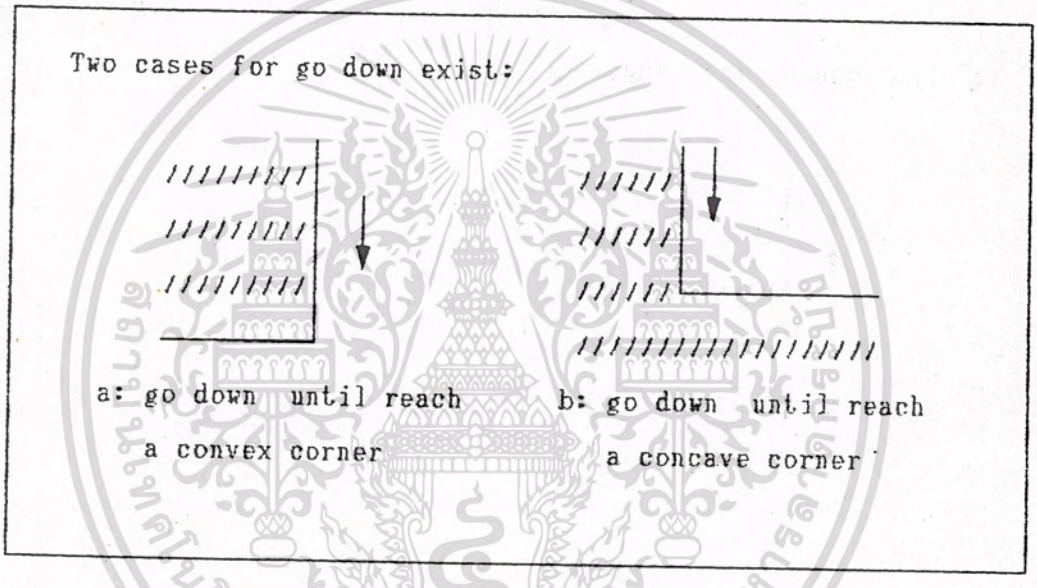
```
label1:  IF ( C = objnum )
            BEGIN
            y = y+1
         IF (L = objnum )
            BEGIN
            /* case b: */
            convex  = 0
            save this corner point(x,y) and convex condition
            in points sturcture

            GOTO label6
            END
         ELSE
            GOTO label1
            END
         ELSE
            BEGIN
           /* case a: */
           convex = 1
           save this corner point(x,y) and convex condition
           in points sturcture
           END
label6: IF (convex = 0) GOTO label4
```

```
/*============================  Rigth  ============================*/
```



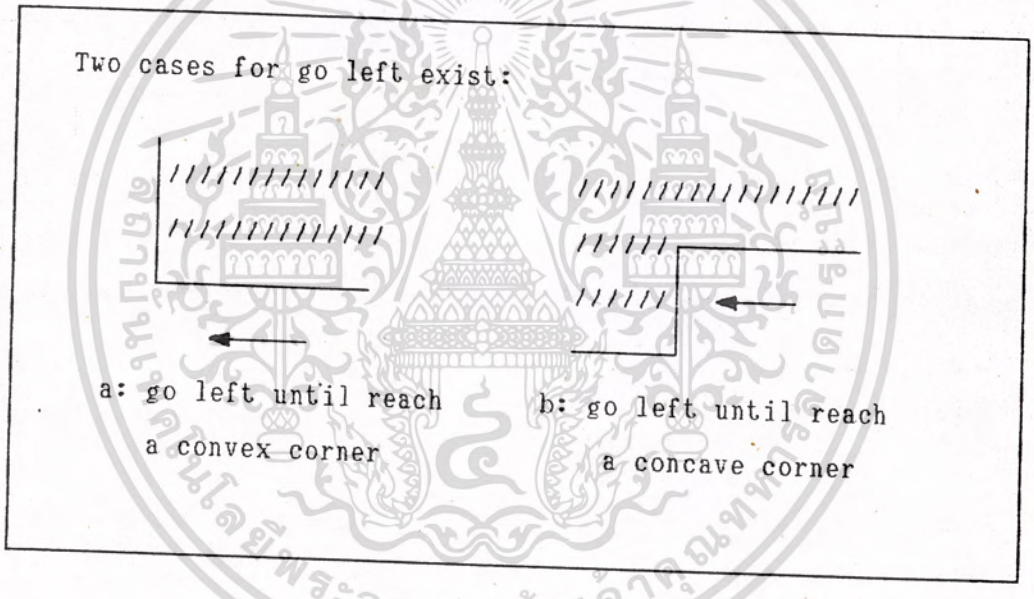

```
label2: IF ( C!= objnum)
          BEGIN
          x = x+1
/* add the condition C != objnum in to bander 8 connected region */
          IF ((D != objnum) AND ( C != objnum ))
             BEGIN
             /* case a: */
             convex = 1
             save this corner point(x,y) and convex condition
             in points sturcture
             GOTO label7
             END
          ELSE
             GOTO label2
             END
          ELSE
             BEGIN
             /* case b: */
```

```
                convex = 0
                save this corner point(x,y) and convex condition
                in points sturcture
                END
label7: IF (convex = 0) GOTO label5

/*============================= DOWN ================================*/
```




```
label3:  IF (LL = objnum)
            BEGIN
            y = y-1
         IF (D = objnum)
            BEGIN
            /*  case b: */
            convex = 0
            save this corner point(x,y) and convex condition
            in points sturcture
            GOTO label8
            END
         ELSE
            GOTO label3
            END
```

```
        ELSE
            BEGIN
            /* case a: */
            convex = 1
            save this corner point(x,y) and convex condition
            in points sturcture
            END
label8:  IF (convex = 0) GOTO label2

/*============================ Left ============================*/
```




```
label4 : IF (L = objnum)
            BEGIN
            x = x-1
            IF (LL = objnum)
               BEGIN
               /* case b:  */
               convex  = 0
               save this corner point(x,y) and convex condition
               in points sturcture
               GOTO label9
               END
```

```
                ELSE
                  GOTO label4
                  END
                ELSE
                  BEGIN
                  /* case a: */
                  convex = 1
                  save this corner point(x,y) and convex condition
                  in points sturcture
                  END
label9:   IF (convex = 0) GOTO label3
label5:   IF the starting point is again reached GOTO done
                ELSE  GOTO label1
done:
/*=================================================================*/
```

โดยการประลุกต์ ขั้นตอนนี้กับวัตถุ เราก้ำหนดต้ำแหน่งและความเว้าของจุดมุมตามขอบเขตของวัตถุ ข้อมูลของจุดมุมนี้เตรียมไว้เพื่อบอกเราว่าตรงไหนน่าจะมีรอยหยัก(jaggies) ปรากฏอยู่ดังที่ได้อธิบายไว้ในตอนต้นบท ซึ่งขณะนี้ข้อมูลต่างๆ ได้ถูกก้ำหนดเรียบร้อยแล้ว ขณะนี้ก็เป็นเวลาที่ จะต้องตรวจสอบว่า potential jaggies อันไหนที่เป็น jaggies จริงๆ ที่ต้องท้าการ smooth

บทที่ 4 การ SMOOTH ขอบ

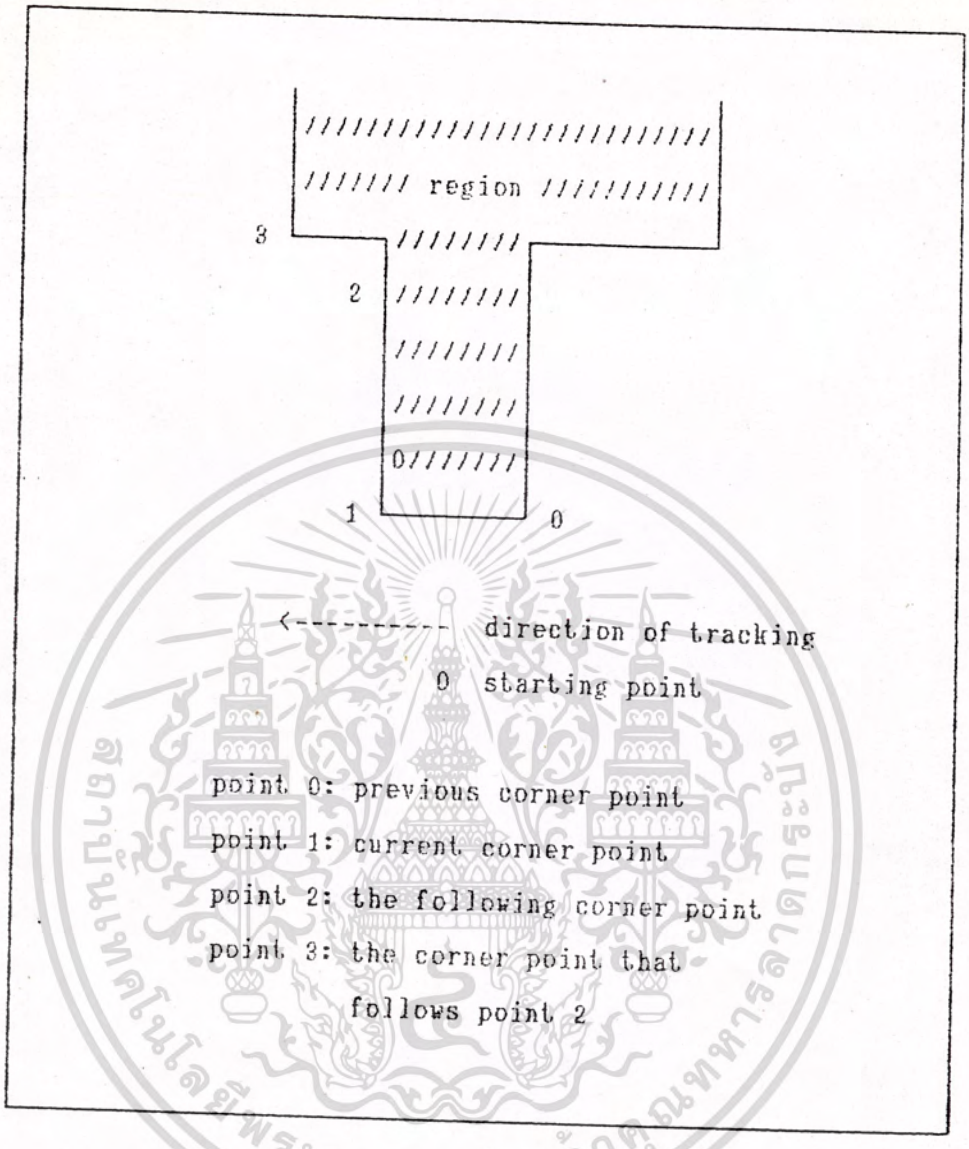
ขณะนี้เราอยู่ขั้นตอนที่ 3 ของปัญหา Antialiasing ในขั้นตอนแรก เราทำการเติมค่าใน tag buffer และกำหนด pixels ใน image ว่าอยู่ในวัตถุใดเราได้กำหนดจุดเริ่มต้นสำหรับแต่ละวัตถุตามด้วยสีของแต่ละวัตถุ ในโครงสร้างข้อมูลที่เรียกว่า ctab เมื่อทำขั้นตอนต่างๆในขั้นแรกเสร็จ ค่าใน tag buffer และ ctab จะเก็บข้อมูลทั้งหมดเกี่ยวกับ image บนจอภาพ ซึ่งจำเป็นสำหรับขั้นตอนที่จะตามมาของการ Antialiasing จอภาพจะไม่ถูกอ้างถึงอีกจนกระทั่งถึงการเขียนข้อมูลกลับลงบนจอภาพขณะทำการ Antialising

ในขั้นตอนที่ 2 เราเริ่มต้นที่จุดเริ่มต้นของแต่ละวัตถุและ track ไปตามขอบของวัตถุพร้อมกับเก็บข้อมูลของจุดมุมตามทาง
ต่อไปนี้เราจะใช้ข้อมูลเหล่านี้ในขั้นตอนที่ 3 ของขบวนการ Antialiasing

จากโครงร่างของจุดมุมตามขอบของวัตถุเราจะพิจารณาตำแหน่งของรอยหยัก ตามขอบของวัตถุ ในขณะที่พบ "รอยหยัก" ตามขอบของวัตถุ รอยหยักสามารถที่จะถูก Antialising ทำให้เกิดขอบที่ราบเรียบกว่าได้

ที่แต่ละจุดมุมที่ทำการตรวจสอบ เราจะพิจารณาจุดที่อยู่ก่อนหน้าและจุดที่อยู่หลังจุดปัจจุบันด้วย จากโครงร่างของจุดมุมนี้ เราจะต้องพิจารณาว่าเป็นรอยหยักหรือไม่ ถ้าเป็นเราต้องเลือก 2 จุด ซึ่งกำหนดขอบที่จะถูก Antialising และทำการ smooth ขอบนั้นตามด้านนอกของขอบของวัตถุ ถ้าเราพิจารณาว่าไม่มีรอยหยัก ปรากฏอยู่ ก็ไม่ต้อง smooth และเราก็ทำต่อไปตามขอบของวัตถุ เพื่อหารอยหยัก อื่น ๆ ต่อไป โดยการดูจากจุดมุม 4 จุด สามารถพิจารณา pixel เริ่มต้นและ pixel สุดท้ายที่เราต้องการเปลี่ยน shade ด้วยสีใหม่

ถ้าจุดมุมอยู่ในจุดที่เหมาะสม เราจะพบรอยหยัก ตามขอบของวัตถุ ถ้ารอยหยัก ถูกพบเราจะเลือก pixel เริ่มต้น (Xs,Ys) และ pixel สุดท้าย (Xe,Ye) จาก 4 จุดมุม ดังนั้นเราสามารถเปลี่ยน shade ของ pixel จาก (Xs,Ys) ถึง (Xe,Ye) ทำให้ขอบที่เป็นรอยหยัก ถูก smooth รูปที่ 4.1 และตารางที่ 1 ใช้แสดงให้เห็นว่าโครงร่างของจุดมุมที่ถูกกำหนดเป็นรอยหยัก ซึ่งจะต้องทำการ smooth นั้นอยู่ในลักษณะอย่างไร



รูปที่ 4.1 : โครงร่างของมุม

CONSECUTIVE CORNER POINTS: EDGE SMOOTH FROM:

case	point 0	point 1	point 2	point 3	(xs,ys)	(xe,ye)
C1	x	convex	concave	convex	point 1	point 3
c2	x	convex	concave	connave	point 1	point 3
c3	concave	concave	convex	x	point 0	point 2
c4	x	convex	convex	x	*	point 2

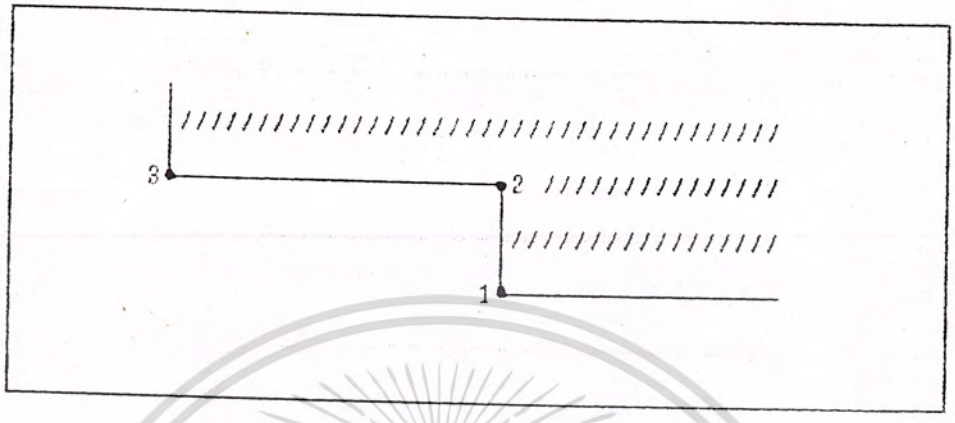
x indicates don't care condition
corner can be concave or convex

* indicates no smoothing done, therefore
no starting point (xs,ys)

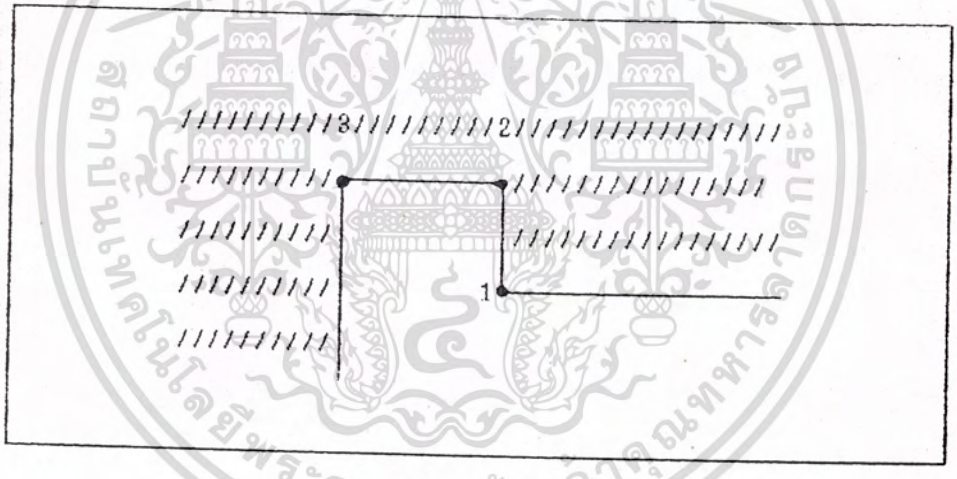
ตารางที่ 1 : การกำหนดของจุดเริ่มต้นและจุดสุดท้าย ขึ้นอยู่กับข้อมูลของจุดมุม

เราเริ่มทำการหาจอยหยัก โดยตรวจสอบจุดมุม ในโครงสร้าง point ดังรูปที่ 4.1 เราพิจารณาจุดมุมที่เรียงกันตามขอบของวัตถุในลักษณะกลุ่มละ 4 จุดมุม ซึ่งจุดมุมต่างๆเรียกว่าจุดปัจจุบัน จุดก่อนหน้า และอีก 2 จุดเป็นจุดที่ตามหลังมา ที่จุดปัจจุบัน เราดูที่จุด 0 จุด 1 จุด 2 และจุด 3 ถ้ากรณี c1, c2 หรือ c3 ถูกพบเราจะกำหนดค่าของ (Xs,Ys) และ (Xe,Ye)และทำการ smooth ระหว่าง 2 จุดนั้น และ (Xe,Ye) จะกลายเป็นจุดปัจจุบัน เราจะพิจารณาโครงร่างของจุด 0 จุด 1 จุด 2 และจุด 3 จากจุดปัจจุบันใหม่นี้ ถ้ากรณี c1,c2 หรือ c3 เกิดขึ้นเราจะกำหนด (Xs,Ys) และ (Xe,Ye) อีก และทำการ smooth ระหว่างขอบของจุดใหม่นี้ กระบวนการนี้จะดำเนินต่อไปจนกระทั่งพบจุดเริ่มต้น

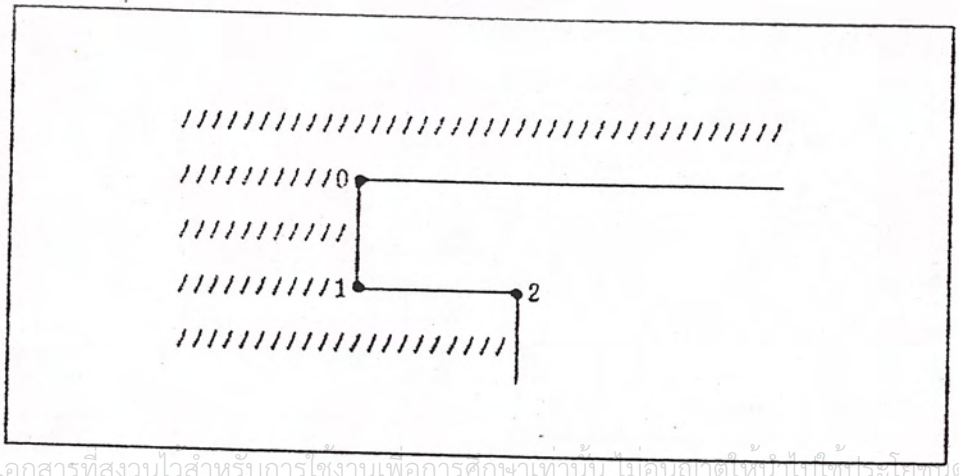
รูปต่อไปแสดงกรณีที่แตกต่างกันที่พบในตารางที่ 1 ทั้งหมด



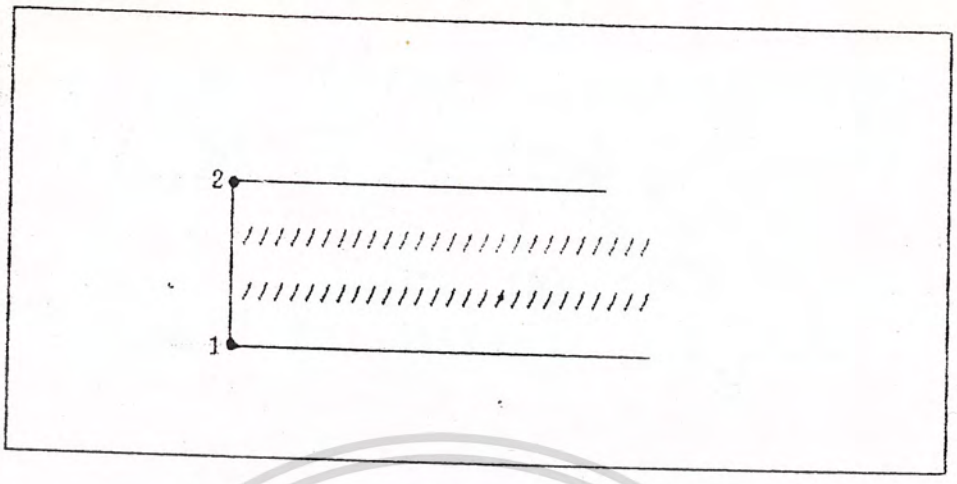
CASE C1



CASE C2

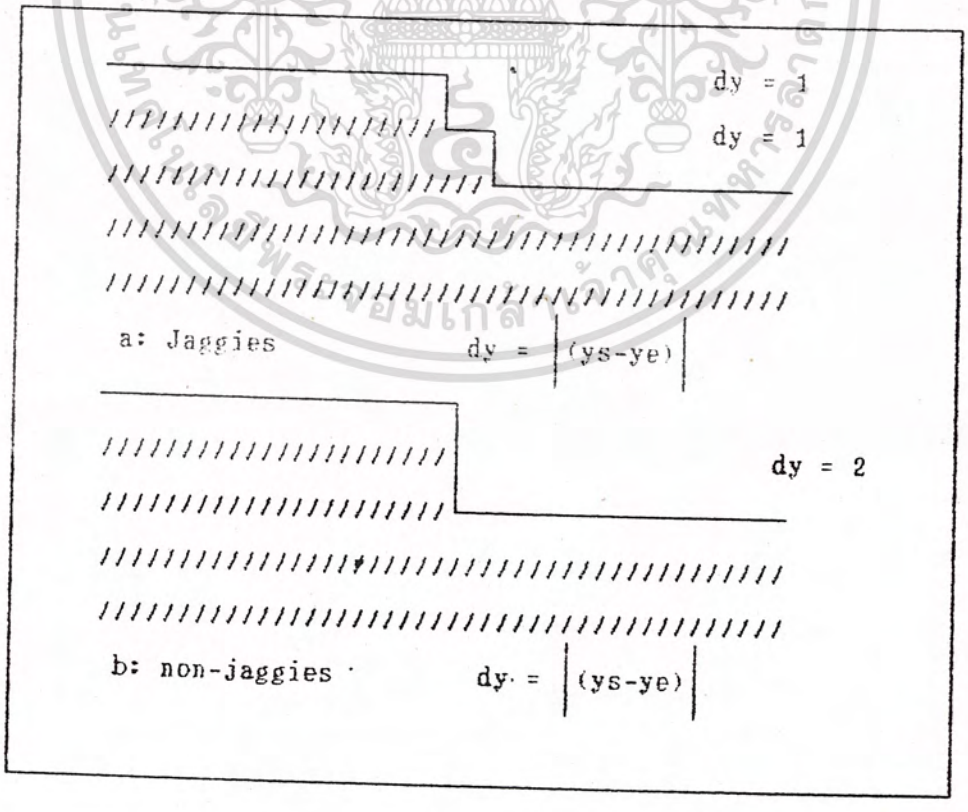


CASE C3



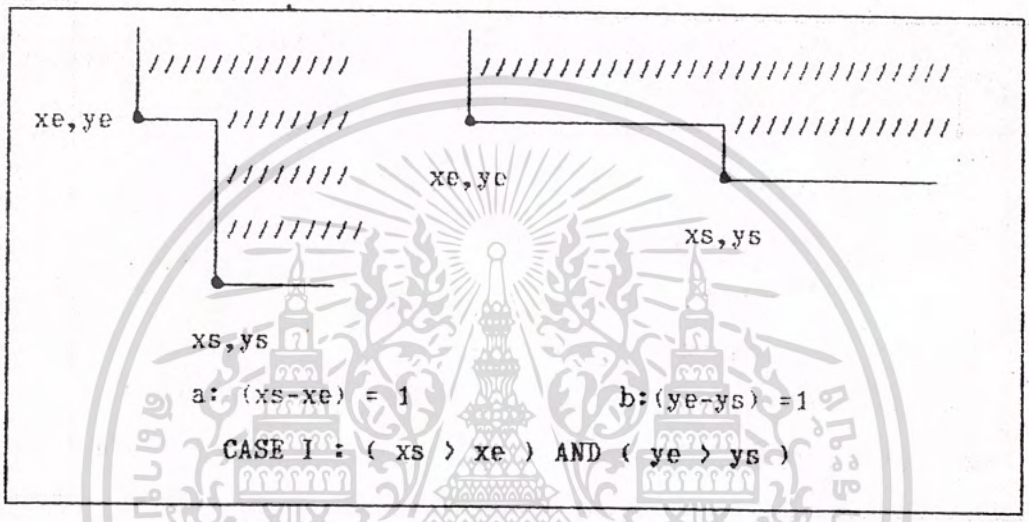
CASE C4

แต่ก่อนที่การ smooth ขอบจะสามารถกระทำจาก (Xs,Ys) ถึง (Xe,Ye) มีกรณีที่ต้องพิจารณาเพิ่มเติมคือ ค่าสัมบูรณ์ (Absolyte) ของ (Xs-Xe) จะต้องเท่ากับ 1 หรือค่าสัมบูรณ์ของ(Ys-Ye)ต้องเท่ากับ 1 ถ้าไม่พบ กรณีอันนี้ ถือว่าขอบนี้ไม่เป็นรอยหยัก จึงไม่ต้อง smooth

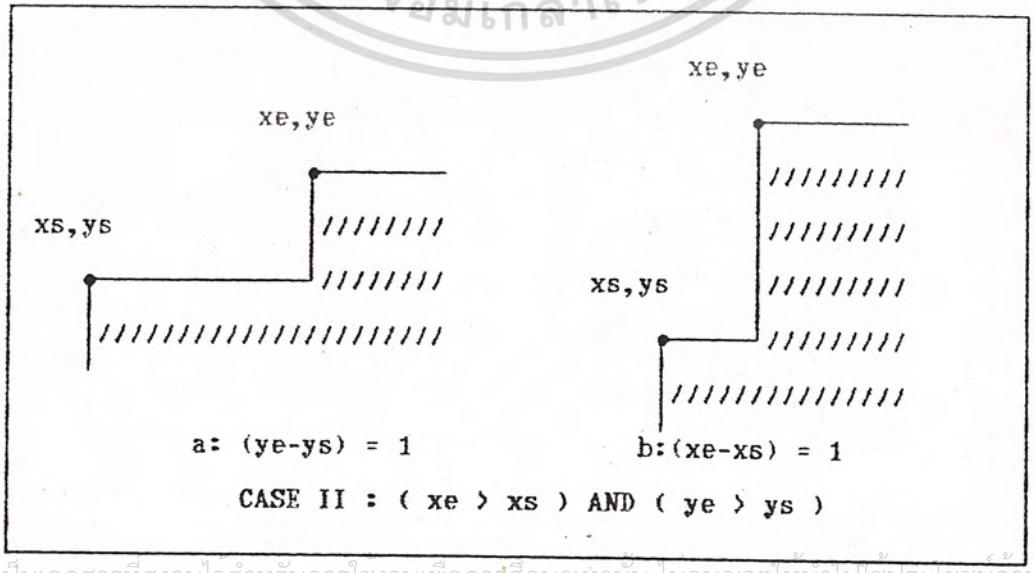


ตัวอย่าง ของ Jaggies และ Non-Jaggies

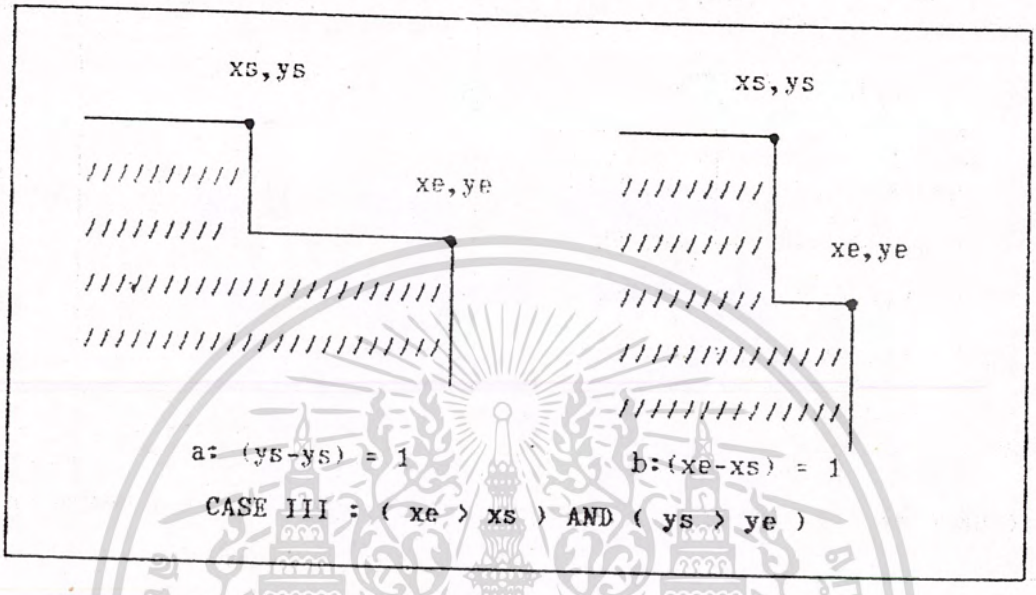
แต่ถ้ากรณีเป็นจริง แสดงว่าเป็นรอยหักและเราจะทำการ smooth ตามขอบที่กำหนด โดย (Xs,Ys) และ (Xe,Ye) ความสัมพันธ์ระหว่าง 2 จุดนี้ จะบอกเราได้ว่า pixel ไหนที่เราจะต้องเปลี่ยน shade ความสัมพันธ์ระหว่างจุด (Xs,Ys) และ (Xe,Ye) แยกออกเป็น 4 กรณี สำหรับแต่ละกรณี ทั้งจุดเริ่มต้น (Xs,Ys)และจุดสุดท้าย (Xe,Ye)เป็น Convex Corners และจุดมุมระหว่างจุดเหล่านี้เป็น Concave Corners



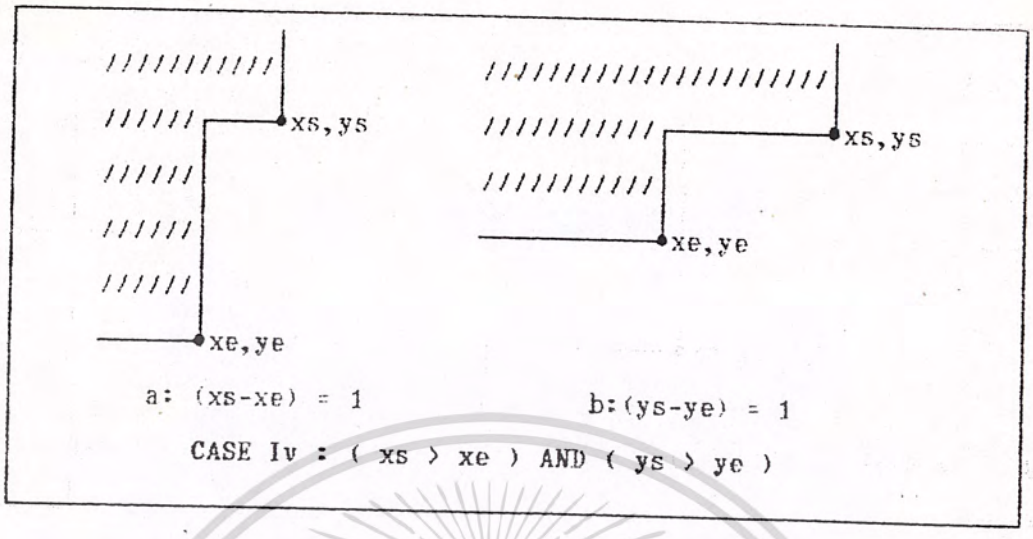
CASE I: แสดงจุดเริ่มต้น xs อยู่ทางขวามือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะต้องอยู่ข้างล่างของจุดสุดท้าย ye สำหรับ case Ia: จุด xs จะต้องอยู่ห่างจากจุด xe ไปทางขวามือ 1 pixel และ case Ib: จุด ys จะต้องอยู่ต่ำว่าจุด ye อยู่ 1 pixel



CASE II: แสดงถึงจุด xs อยู่ทางซ้ายมือของจุด xe และจุด ys ก็ต้องอยู่ต่ำกว่าจุด ye สำหรับ case IIb. จุด xs จะต้องอยู่ทางซ้ายมือจุด xe อยู่ 1pixel และ case IIa : จุด ys จะต้องอยู่ต่ำกว่า ye อยู่ 1 pixel

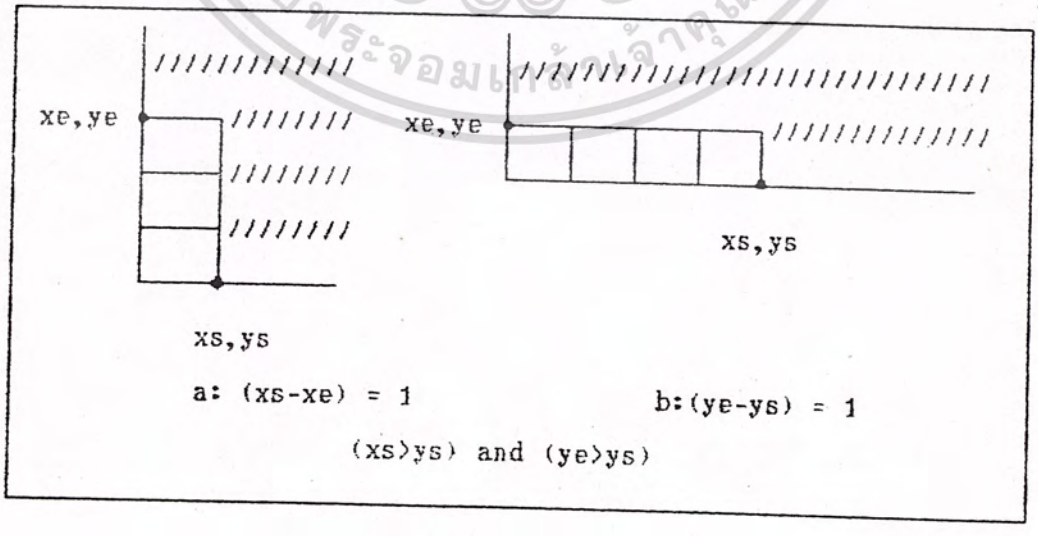


CASE III : แสดงถึงจุดเริ่มต้น xs อยู่ทางซ้ายมือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะอยู่ข้างบนจุดสุดท้าย ye สำหรับ case IIIb. จะแสดงถึงจุด xs จะต้องอยู่ทางซ้ายมือของจุด xe อยู่ 1 pixel และ case IIIa. จุด ys จะต้องอยู่เหนือจุด ye อยู่ 1 pixel

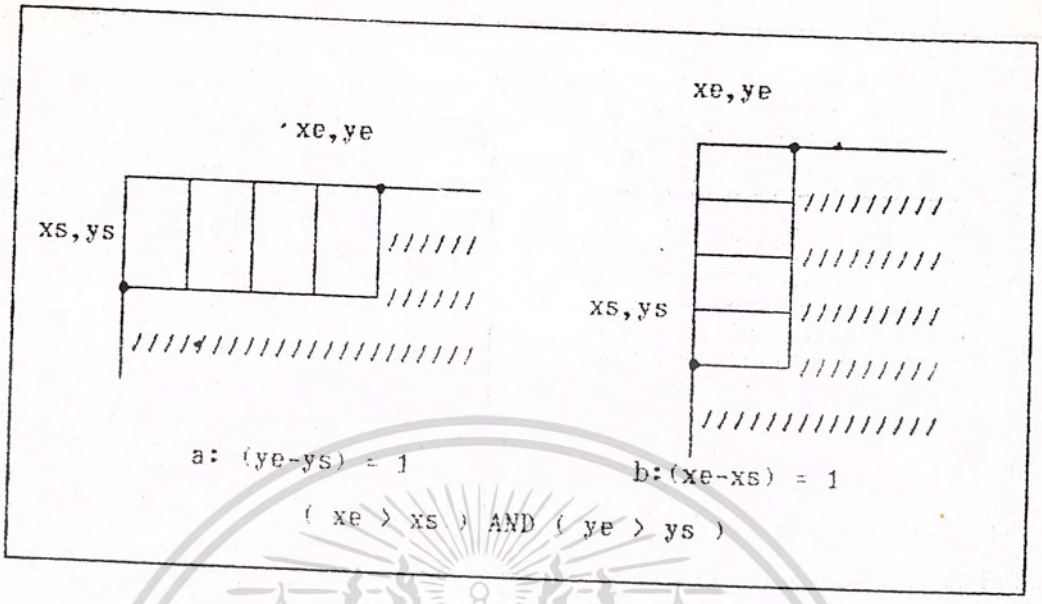


CASE IV : แสดงถึงจุดเริ่มต้น xs จะอยู่ทางขวามือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะต้องอยู่ข้างบนจุด ye สำหรับ case IVa : จุดเริ่มต้น xs จะต้องอยู่ทางขวามือจุดสุดท้าย xe อยู่ 1 pixel และสำหรับ case IVb : จุด ys จะต้องอยู่บนจุด ye อยู่ 1 pixel

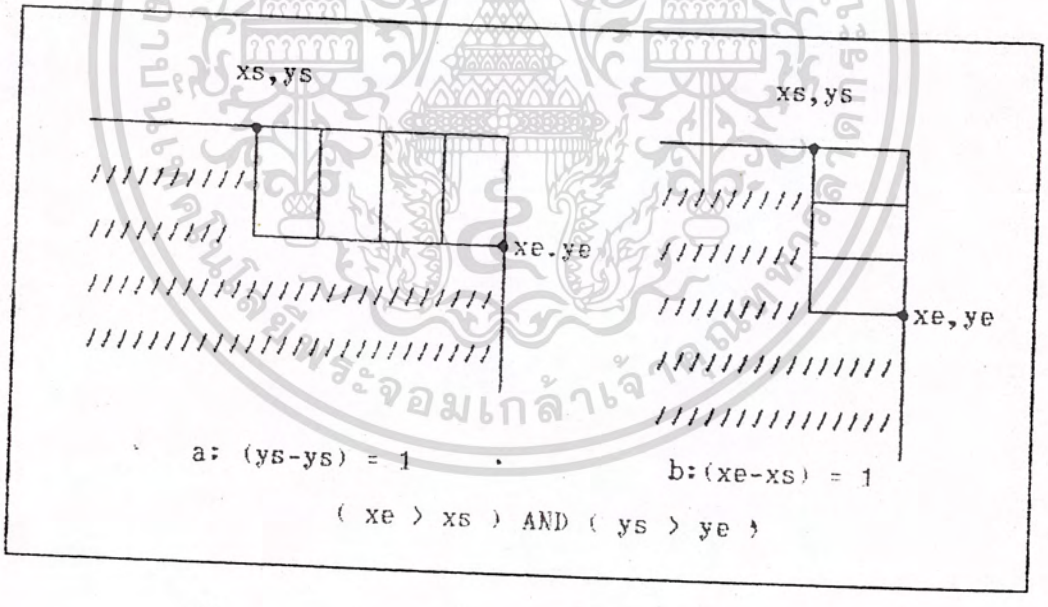
ต่อไปนี้เป็นรูปที่จะอธิบายเกี่ยวกับ pixels (yellow pixel) สำหรับ case ทั้ง 4 ที่ต้องการ smooth โดยวิธีการหาค่าตัวเลขมาคำนวณร่วมกับส่วนที่ต้องการ smooth เพื่อให้ได้ลำดับชั้นสีของ pixel ใหม่



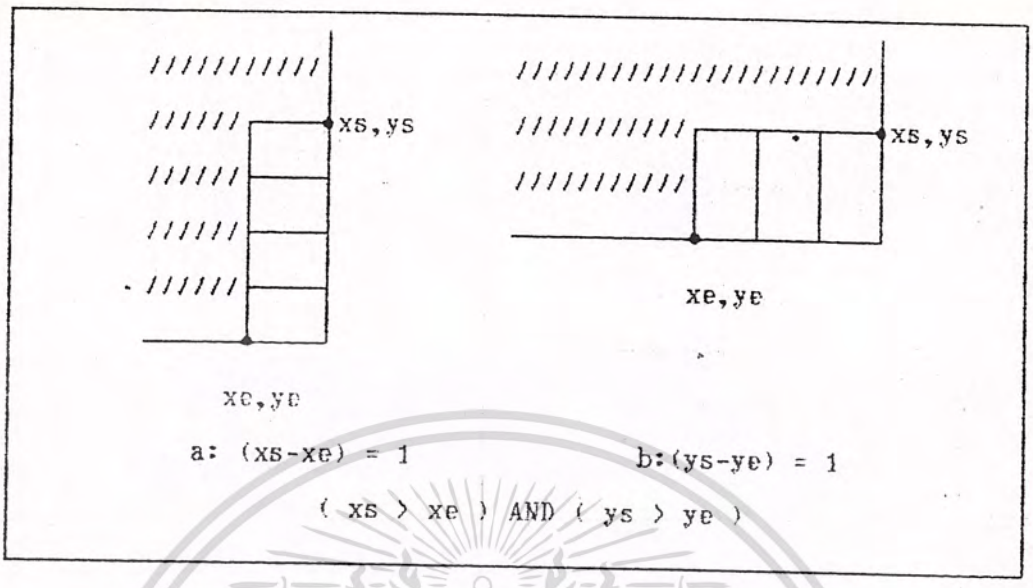
CASE I: PIXELS TO BE RESHADED



CASE II: PIXELS TO BE RESHSDED



CASE III: PIXEL TO BE RESHADED



CASE IV: PIXEL TO BE RESHADED

ดังนั้น เราจะต้องทำการคำนวณค่า pixel จากจุด (xs,ys) ถึงจุด (xe,ye) ในขณะนี้เรามีรอยหยักตามขอบเดิมของวัตถุอยู่แล้ว ดังนั้น pixel จาก (xs,ys) ถึง (xe,ye) บริเวณรอบนอกของรอยหยักของวัตถุนั้นจะมีสีที่แตกต่างจากวัตถุ ถ้าเราสามารถลากเส้นอย่างต่อเนื่องจากจุด (xs,ys) ถึงจุด (xe,ye) และ shade สีจากขอบถึงเส้นนี้ ทำให้รอยหยัก เปลี่ยนไปด้วยแต่ความแตกต่างระหว่างจุดเริ่มต้น (xs,ys) ถึงจุดสุดท้าย (xe,ye) ของเส้นนี้อีก 1 pixel (ในทิศทางของ x or y) มีรายละเอียดของ pixel ไม่ต่อยละเอียดนัก เราจึงกำหนด pixelใหม่โดยให้ตัดกับเส้นสีใหม่ เพื่อต้องการให้แสดงส่วนของแต่ละ pixel ที่เป็นของขอบวัตถุ สำหรับแต่ละ pixel ที่ถูกตัดโดยเส้นของภาพ เราสามารถคำนวณหาค่าเศษส่วนของ pixel ของขอบภาพถ้าเราสามารถจัดลำดับสีใหม่จากขอบภาพไปยังเส้นนี้ เราเรียกเศษส่วนนี้ว่า "สัมประสิทธิ์สำหรับ pixel" เราสามารถคำนวณหาค่าสีใหม่ของ pixel ได้ซึ่งเสมอเหมือนเป็นส่วนหนึ่งของวัตถุ สีใหม่ของ pixel นี้ ได้จากวิธี การคำนวณแล้วบวกเพิ่มส่วนของ (coefficient*object color) ด้วยซึ่งมันเป็นส่วนหนึ่งของวัตถุ ส่วนสี back ground หาได้จาก
((1 - coefficient) * original color of pixel) และจะทำการคำนวณลักษณะนี้ทุก ๆ pixel ซึ่งถูกตัดโดยเส้นจากจุด (xs,ys) ถึง (xe,ye) จะทำให้ภาพ smooth ซึ่งเป็นการลบรอยหยักระหว่างจุด 2 จุดนี้

ตัวอย่างนี้จะช่วยอธิบายการทำงานของกระบวนการนี้ให้เข้าใจง่ายขึ้น, ความสัมพันธ์ระหว่างจุด (xs,ys) และ (xe,ye) ใน case IIa: วิธีการคำนวณหาค่า ส.ป.ส ของแต่ละ region มีดังนี้