ปีการศึกษา 2533

POST ANTI-ALIASING

โดย

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

อาจารย์ที่ปรึกษา

อาจารย์ บวรวรรณ สาริกะภูติ

สารบัญ

บทที่ 1 บทนำ

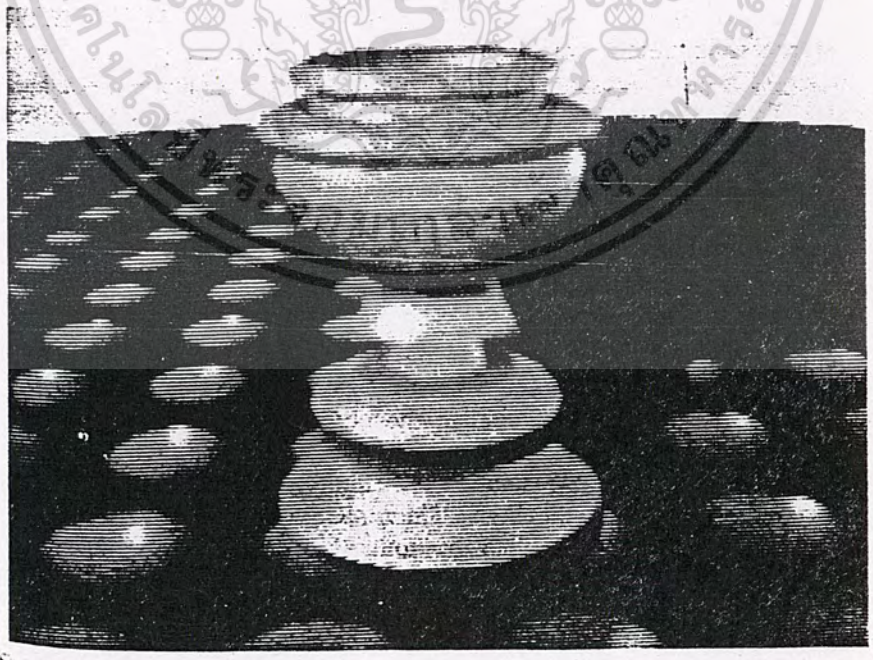
ในโลกของคอมพิวเตอร์กราฟิกการแสดงผล (Display) บนจอภาพเป็นอีกปัญหาหนึ่งที่เกิดขึ้นกับผู้ใช้คอมพิวเตอร์ หรือผู้เขียนโปรแกรม ที่มีความต้องการให้ภาพที่ screen มีความคมชัดเหมือนที่ตั้งใจไว้แต่พอโปรแกรมทำงานจริงๆภาพที่ปรากฏกลับไม่คมชัด หยาบและมีรอยหยัก(aliasel) ทำให้การออกแบบการใช้งานไม่ละเอียดเท่าที่ควร ดังนั้นจึงมีการพัฒนาการแสดงผลทางด้านกราฟิกให้ดีขึ้น

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)

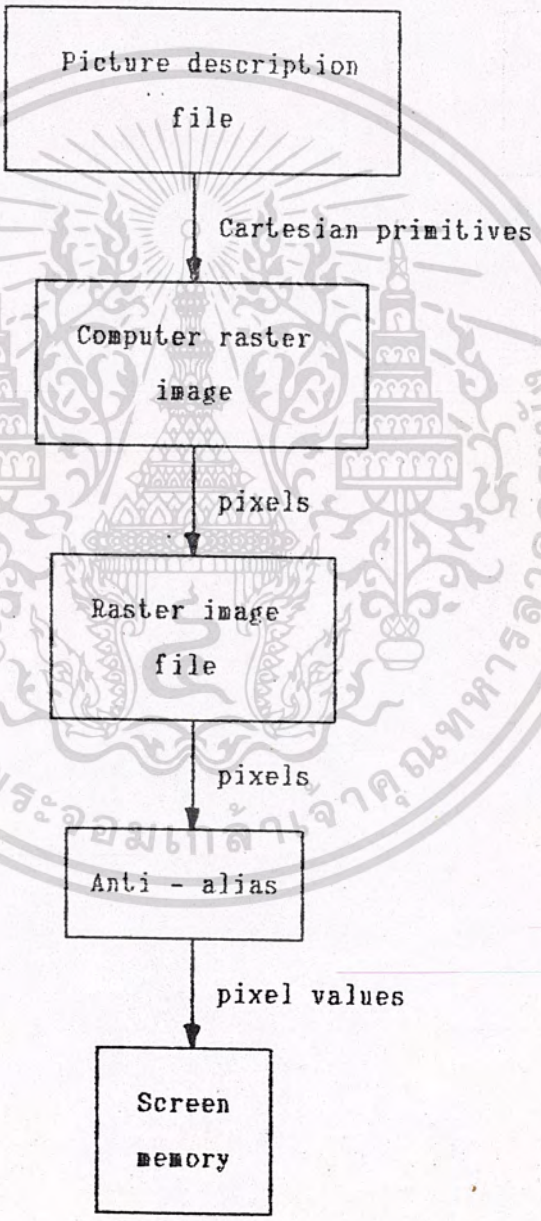


รูป 1.2 ภาพของจอภาพที่มีความละเอียดสูง (High resolution)

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

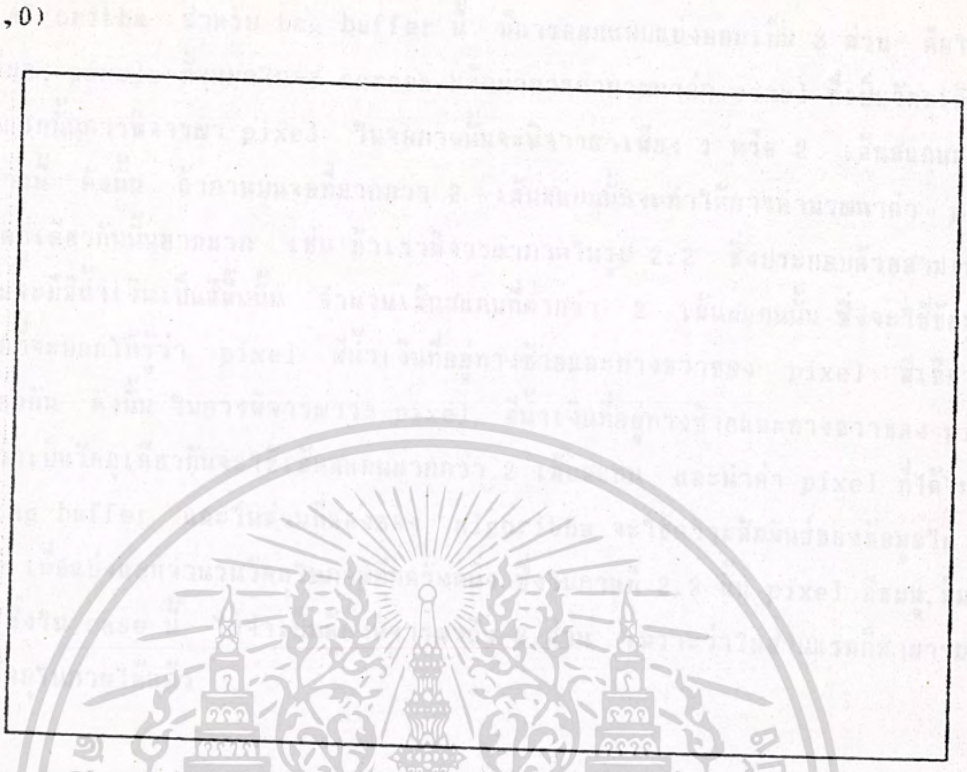
ซึ่งในส่วนของขั้นตอน (process) แม้ว่าไม่ได้อธิบายรายละเอียดไว้ แต่จะกล่าวรายละเอียดในบทต่อไป

บทที่ 2 จะอธิบายการกำหนดขอบเขตของวัตถุโดยใช้ tag buffer

บทที่ 3 จะอธิบายครอบคลุมแนวทางโครงสร้างของขอบเขตวัตถุ

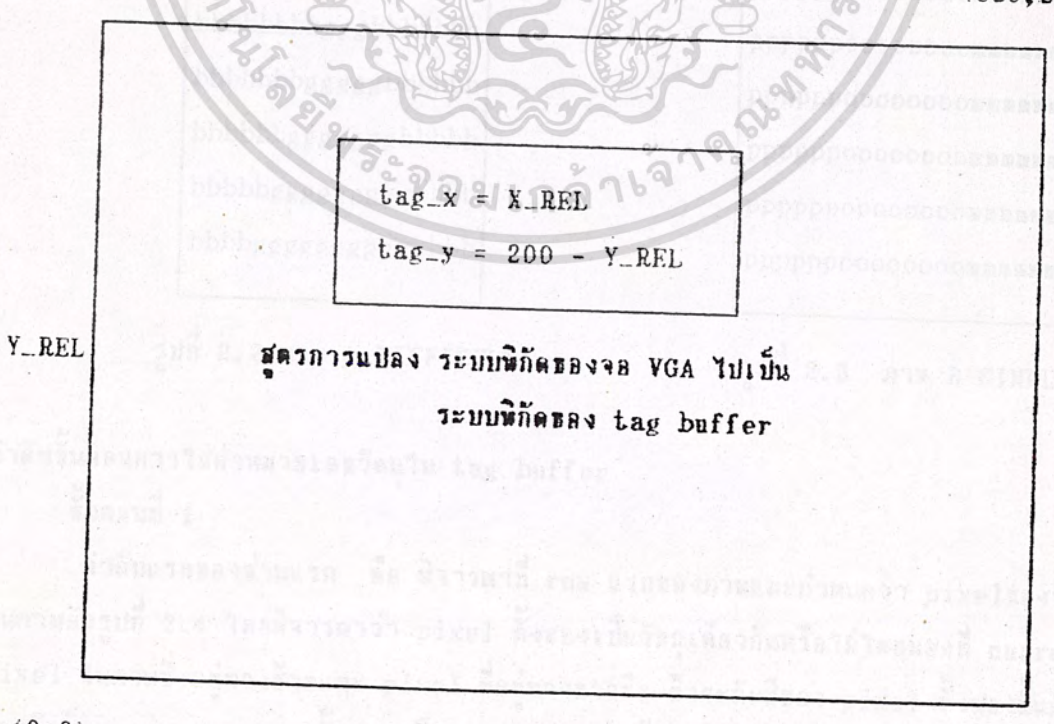
บทที่ 4 จะแนะนำเทคนิคที่นำมาใช้ในการ smooth รอยหยักตลอดขอบริมของวัตถุ

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

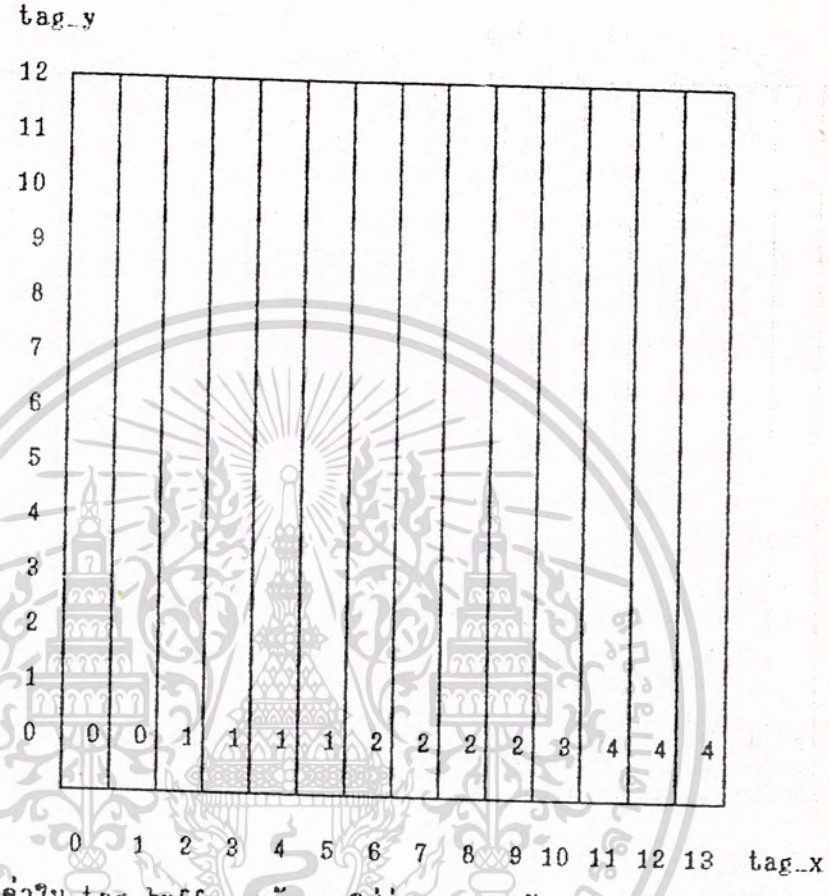
จากรูปที่ 2.5 เราจะทำการเริ่มที่ row ที่ 1โดยการเติมหมายเลขวัตถุลงใน tag buffer สำหรับ dixel แรกจะให้เท่ากับ "0" (tag[0][0]=0) ในขณะเดียวกันเราจะย้ายการมอง row มาอีกส่วนหนึ่งที่คู่กันของ pixelsในภาพ screen ของรูปที่ 2.5 เมื่อ current pixel อยู่ที่ tag-x=0 และ tag-y=0 (ตำแหน่ง(0,0))เราทำการเปรียบเทียบสีกับตำแหน่งที่ (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

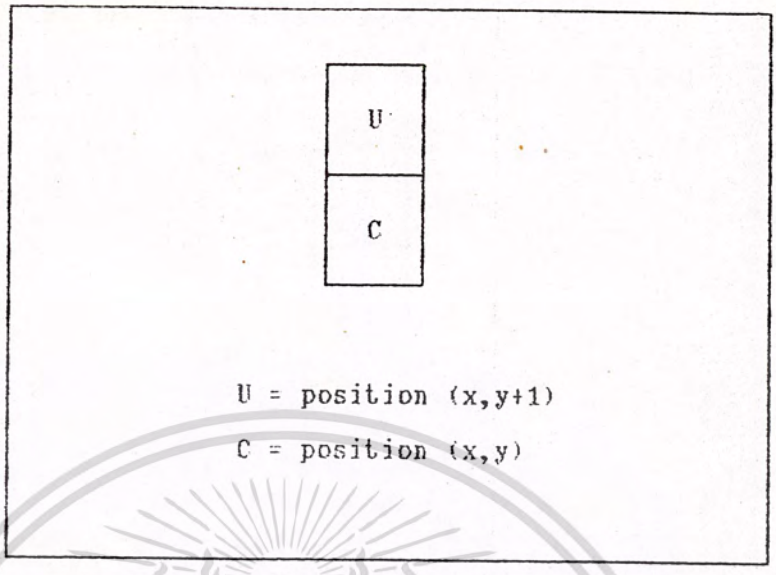
ดังนั้นผลที่ได้จากการเติมค่าหมายเลขวัตถุใน row แรกของ tag buffer จะได้ดังรูปที่ 2.6



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

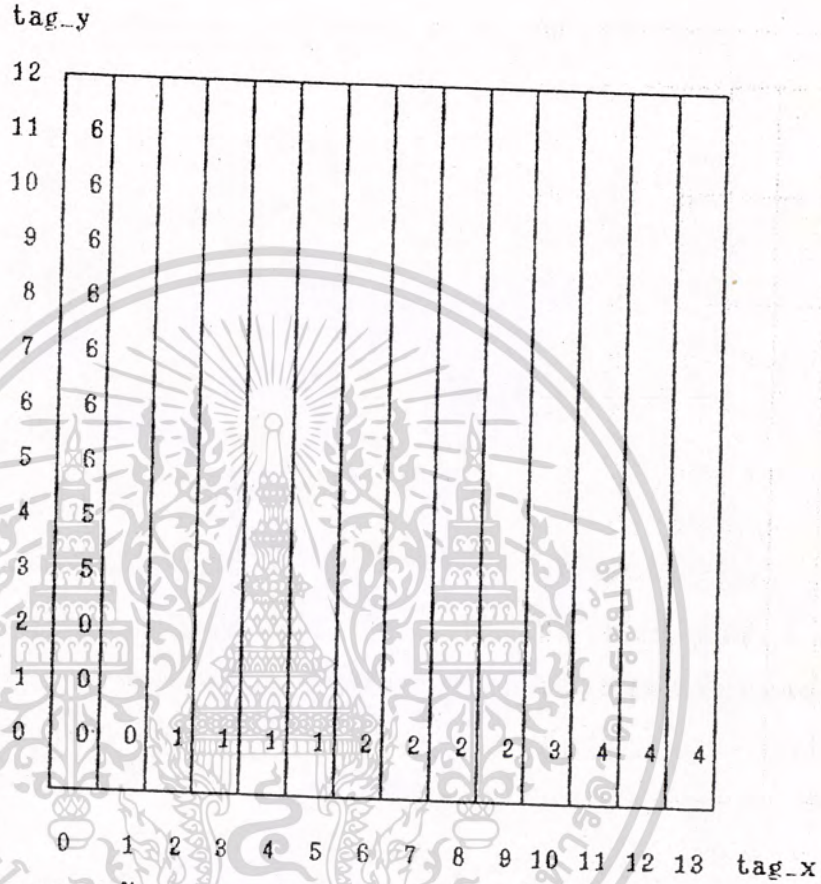


รูปที่ 2.7 แสดงถึง pixel 1 คู่ใน columnแรก

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



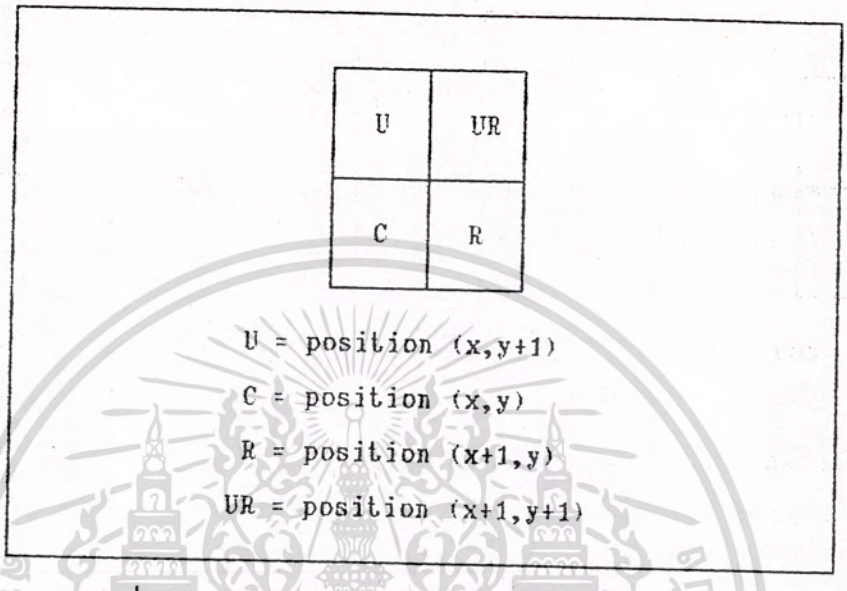
รูปที่ 2.8 ค่าหมายเลขวัตถุใน tag buffer ของ column แรกและ row แรก

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

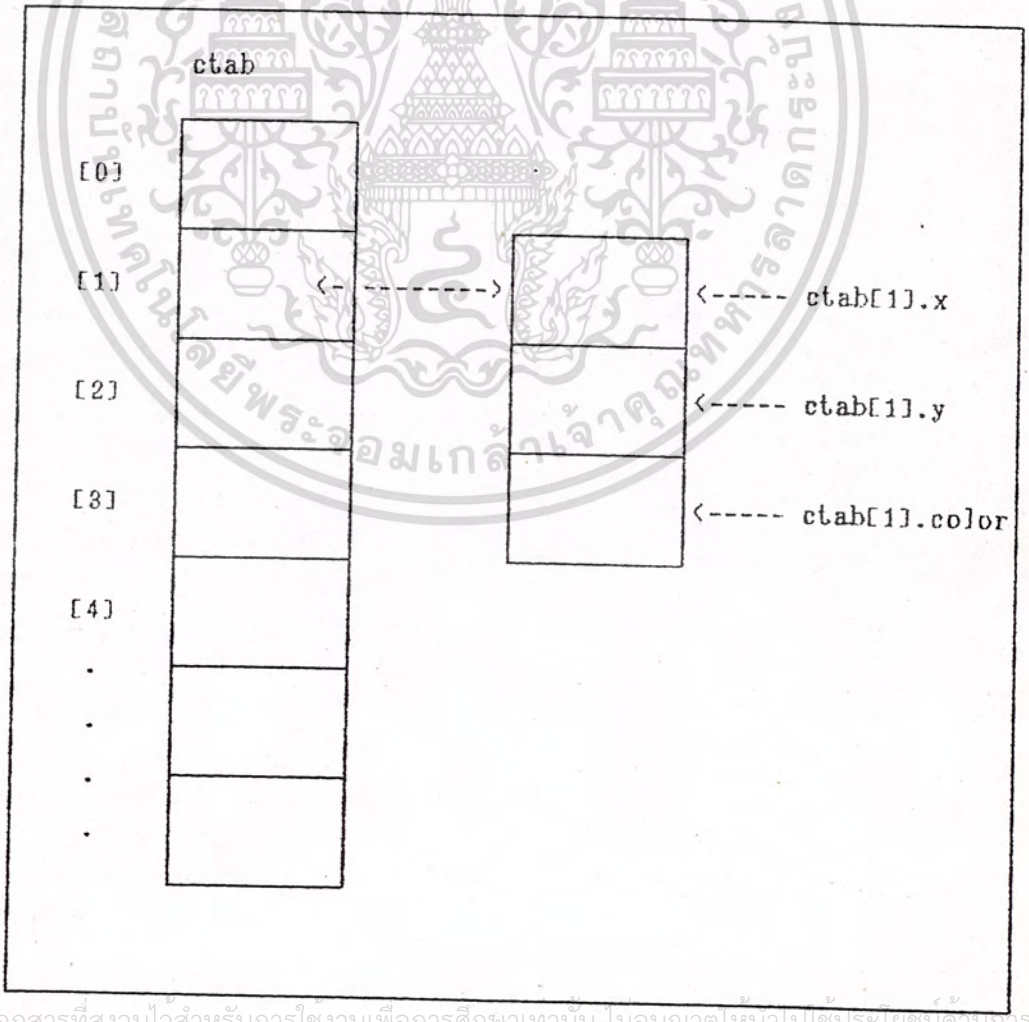
```
/*=================================================================*/
    j = -1
    FOR (y=0 to (y < ysize))
        BEGIN
        /* moving across a row */
        FOR (x=0 to (x < xsize))
            BEGIN
        /* check that this objnum is not related to anyother objects
           or it is 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

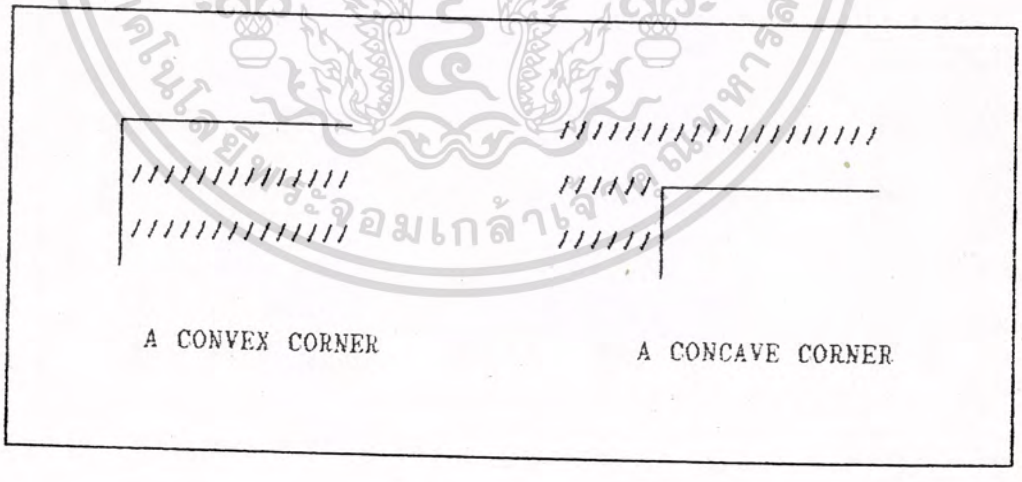
การประยุกต์ใช้ขั้นตอนส่วนที่ 1 และส่วนที่ 2 ถือเป็นการสำเร็จขั้นแรกของกระบวนการ antialiasing และในขณะนี้ภาพวัตถุ เริ่มแรกได้ถูกแทนค่าลงใน tag buffer ซึ่ง regions ของ pixels ได้ถูกกำหนดและใน ctag table ซึ่งจุดเริ่มต้นบนขอบของแต่ละวัตถุได้ถูกกำหนดเช่นเดียวกับสีของแต่ละ regions ซึ่งเมื่อมีมูลต่างๆ ได้ครบแล้ว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

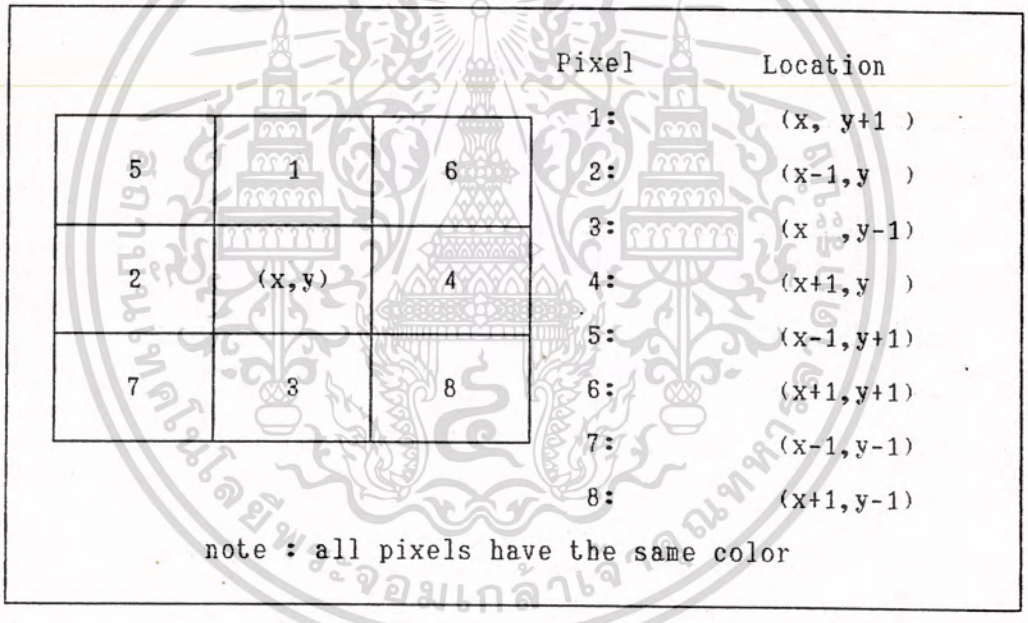
สำหรับในขั้นตอนที่ 3 นี้เราจะสนใจเกี่ยวกับการหาและเก็บตำแหน่งของจุดเริ่มต้นรอยหยัก และในขั้นตอนต่อไป เราจะทำการกำหนดค่าตำแหน่งของรอยหยักที่จะทำการ antialiaing มันตำแหน่งซึ่งเป็นจุดเริ่มต้นของรอยหยัก จะหาได้โดยทำการ trackingไปตามขอบของวัตถุ และไม่มี Pixel ณ ตำแหน่งที่ Pixelพบกันเป็นมุม ณ ตำแหน่งที่เกิดขึ้นตามขอบของวัตถุจะเรียกเป็น 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"

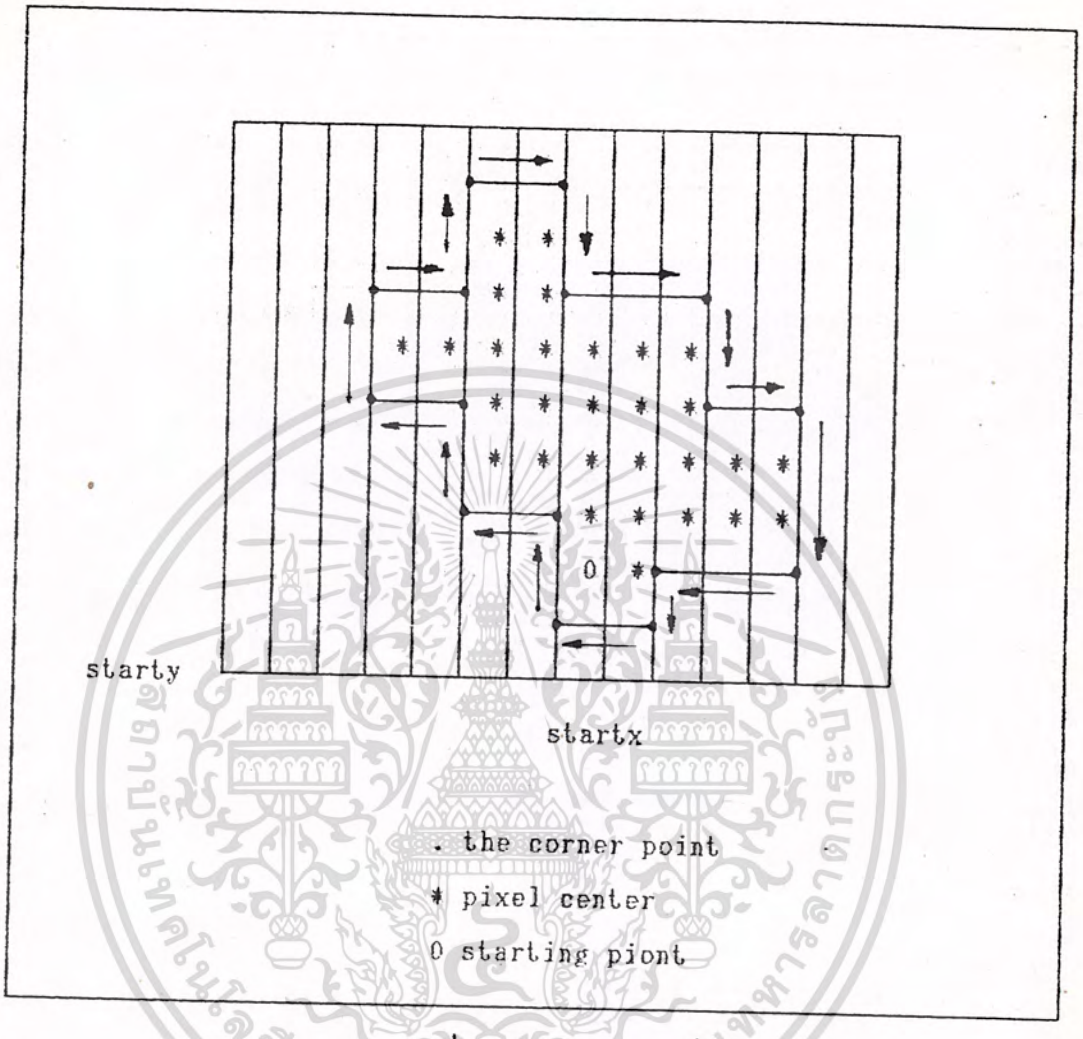
จากหนังสือ "Computer Graphics", Volume 17, Number 3 ต่อไปนี้จะการปรับปรุงทฤษฎี ของ Bloomenthal's algonithm, ซึ่งจะใช้ pixel 8 pixel ที่ติดกันในการพิจารณา ดังแสดงในรูปที่ 3.2 กำหนดให้ 8 pixel ที่ติดกับ Pixel (x,y) คือ Pixel 1,2,3,4,5,6, 7 และ 8 เป็น Pixel ที่อยู่ติดกันกับ Pixel ที่ตำแหน่ง (x,y) และเป็นสีเดียวกันทั้งหมดนั้น 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

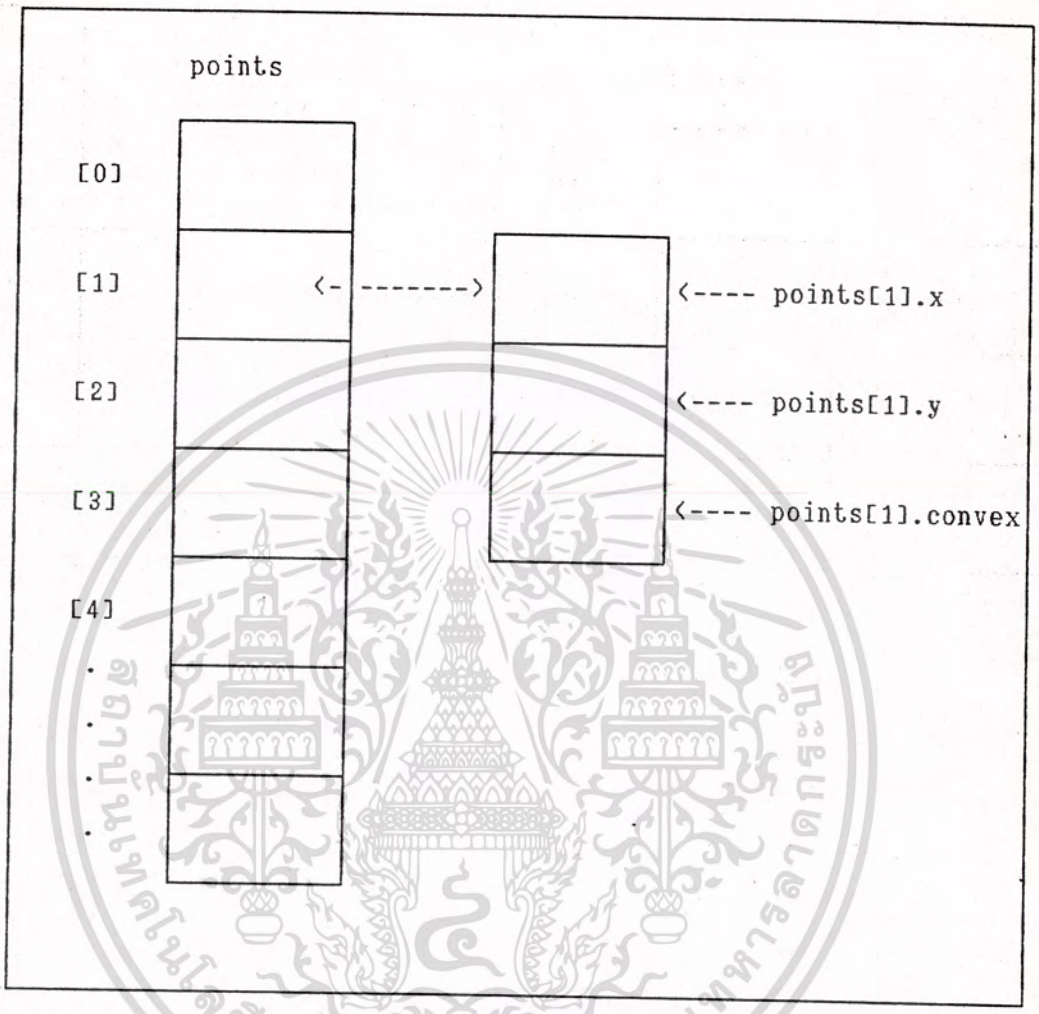


รูปที่ 3.3 : region ที่ติดกัน

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

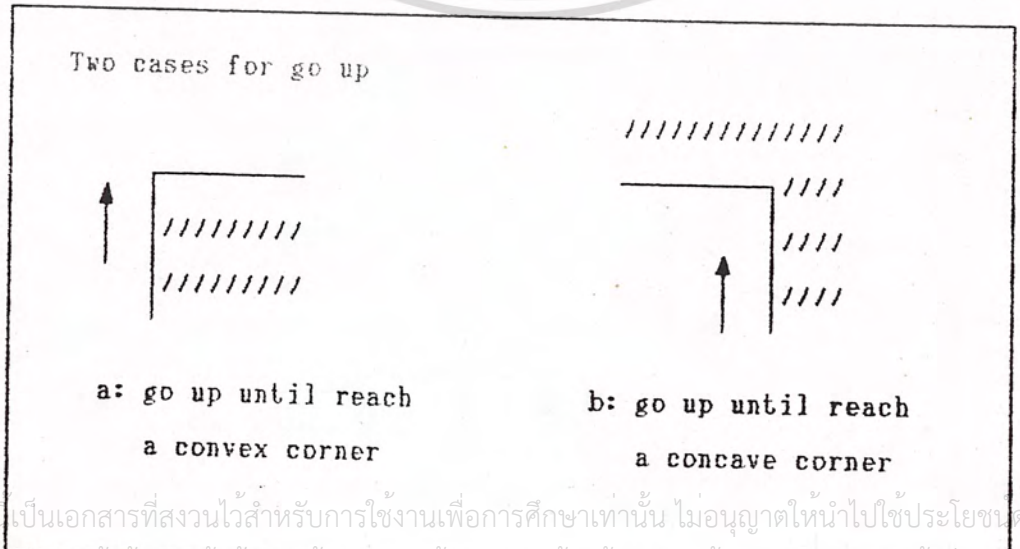
ในรูปที่ 3.4 เราแสดงว่าข้อมูลของจุมมุมถูกเก็บได้อย่างไร สำหรับจุดมุมที่ 1 ตำแหน่ง x และ y นโครงสร้างเก็บตำแหน่งของจุดมุม และในตำแหน่ง Convex ใช้แสดงความเว้าของจุดมุมซึ่งค่า 0 จะหมายถึง มุมโค้งออก (concave) และค่า 1 แสดงว่าเป็นมุมโค้งเข้า (Convex)

ในการที่จะแสดงว่าจะนำค่ามาเติมในโครงสร้าง point ได้อย่างไรนั้น เราจะแสดงด้วย pseudocodeของลำดับขั้นตอนในการ track ขอบภาพได้เตรียมไว้ในลักษณะของการช่วยในการ

พิจารณาการทำงานของขั้นตอน

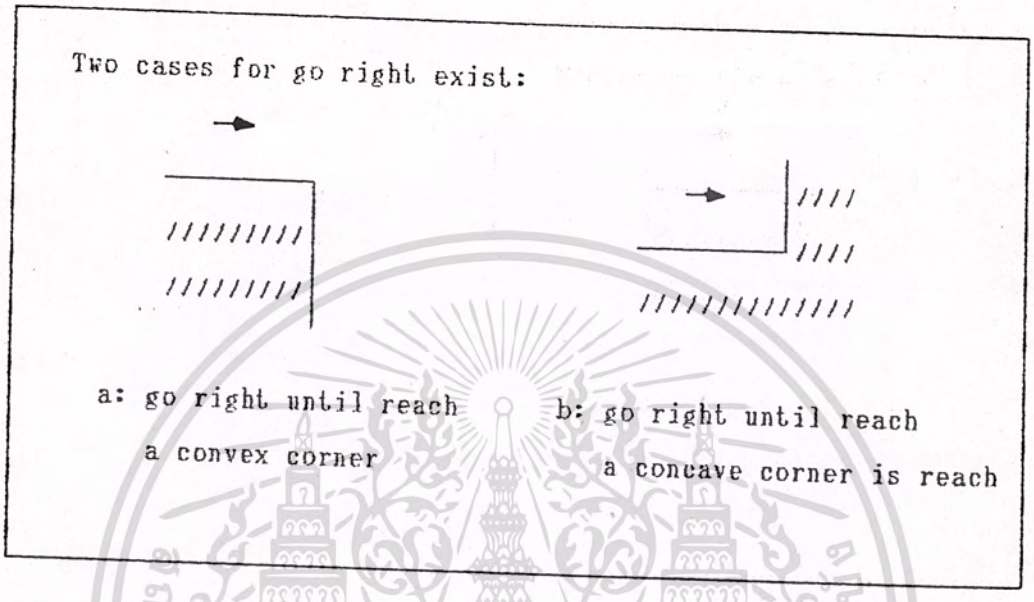
```
/*================== ALGORITHM EDGE TRRACKKING ======================*/
/*   grive a starting point startx,starty for an object region      */
     x = startx
     y = starty
/*  store first corner point  */
     save this corner point(startx,starty) and convex = 1 in point
     sturcture
/* objnum 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



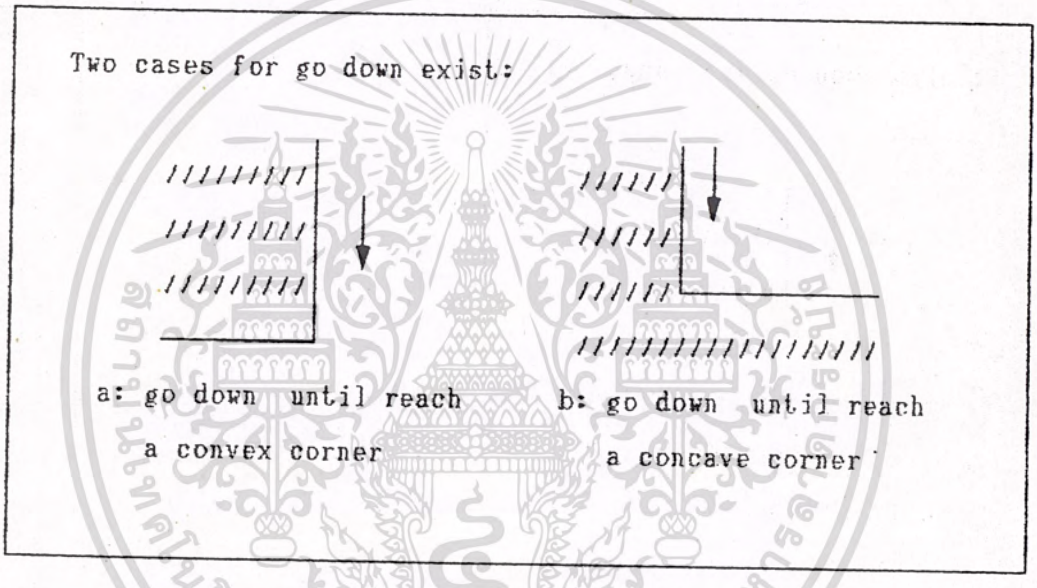
```
label1:  IF ( C = objnum )
            BEGIN
            y = y+1
         IF (L = objnum )
            BEGIN
            /* case b: */
            convex  = 0
            save this corner point(x,y) and convex condition
            in points sturcture

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



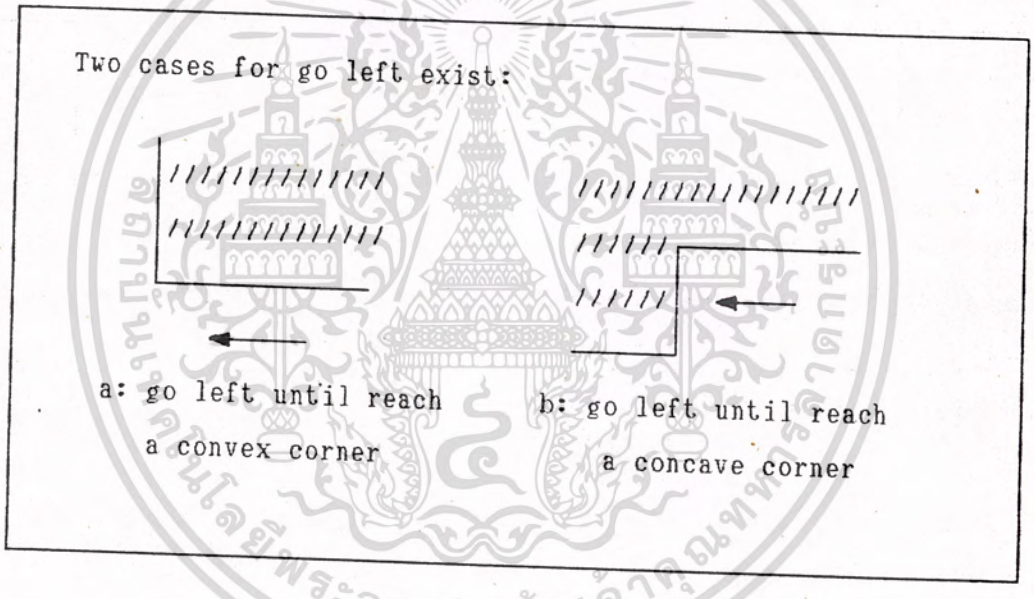

```
label2: IF ( C!= objnum)
          BEGIN
          x = x+1
/* add the condition C != objnum in to bander 8 connected region */
          IF ((D != objnum) AND ( C != objnum ))
             BEGIN
             /* case a: */
             convex = 1
             save this corner point(x,y) and convex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

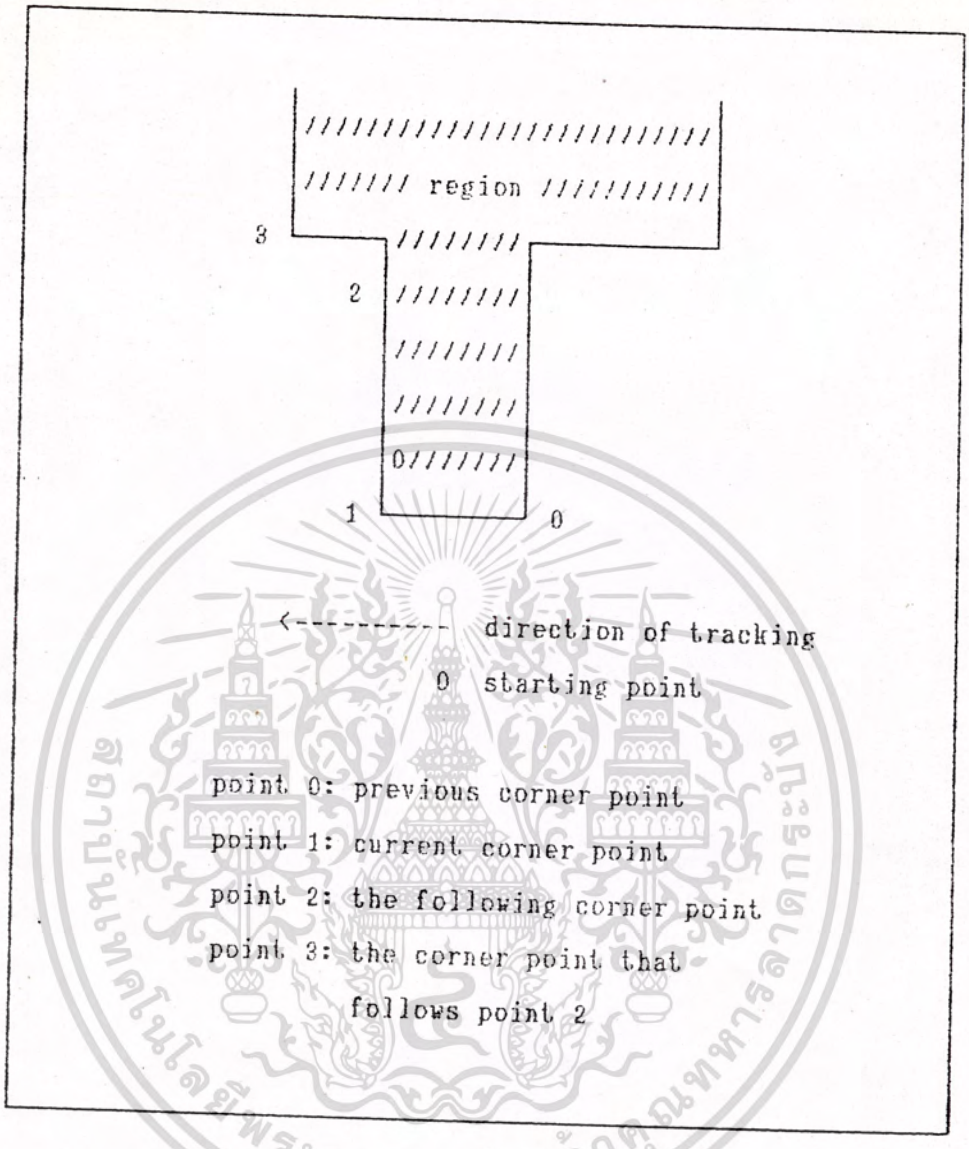
ขณะนี้เราอยู่ขั้นตอนที่ 3 ของปัญหา Antialiasing ในขั้นตอนแรก เราทำการเติมค่าใน tag buffer และกำหนด pixels ใน image ว่าอยู่ในวัตถุใดเราได้กำหนดจุดเริ่มต้นสำหรับแต่ละวัตถุตามด้วยสีของแต่ละวัตถุ ในโครงสร้างข้อมูลที่เรียกว่า ctab เมื่อทำขั้นตอนต่างๆในขั้นแรกเสร็จ ค่าใน tag buffer และ ctab จะเก็บข้อมูลทั้งหมดเกี่ยวกับ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

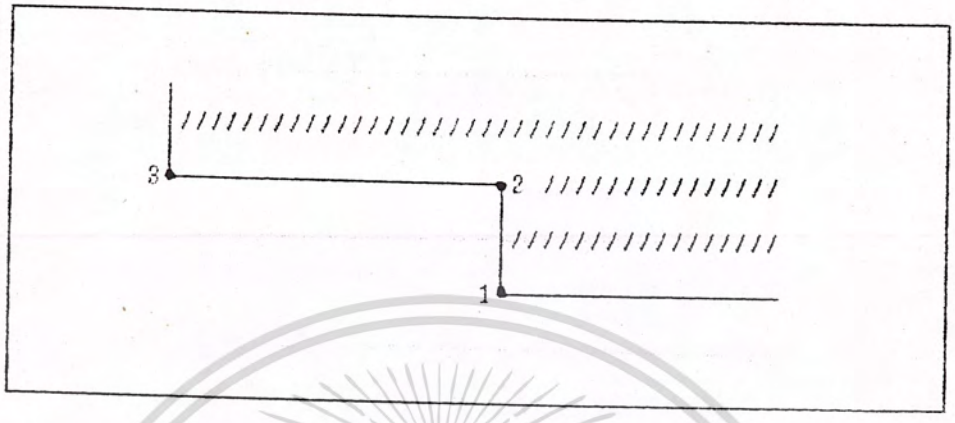
x indicates don't care condition
corner can be concave or convex

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

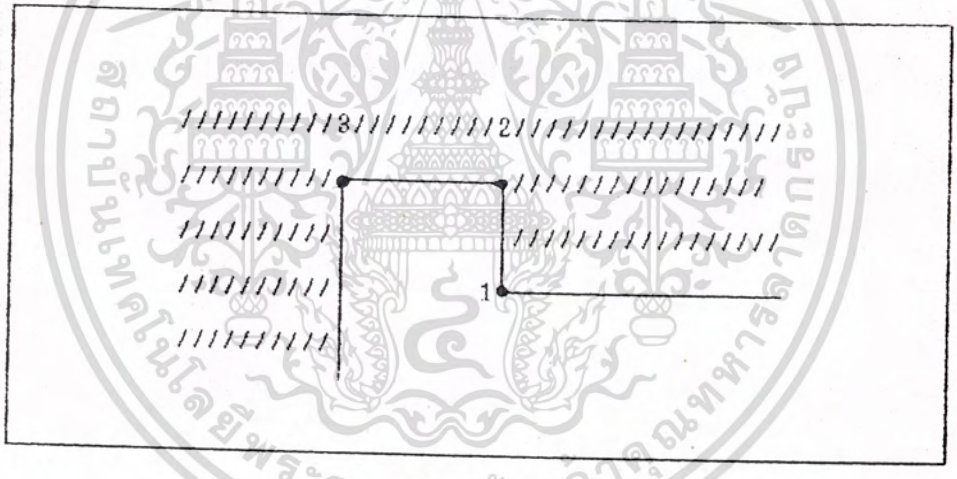
ตารางที่ 1 : การกำหนดของจุดเริ่มต้นและจุดสุดท้าย ขึ้นอยู่กับข้อมูลของจุดมุม

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

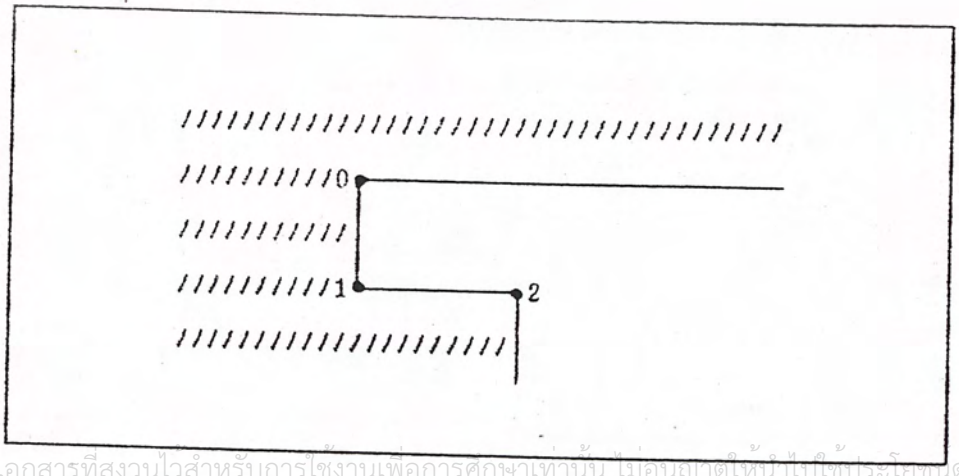
รูปต่อไปแสดงกรณีที่แตกต่างกันที่พบในตารางที่ 1 ทั้งหมด



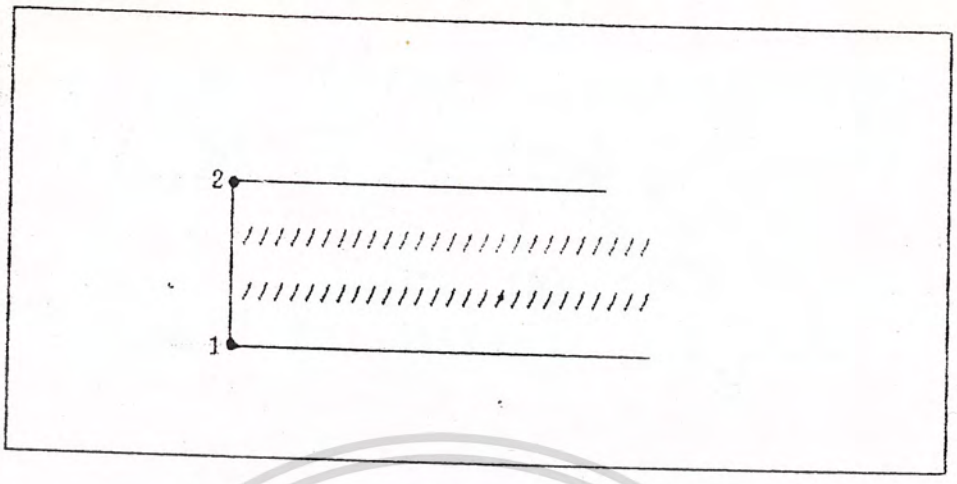
CASE C1



CASE C2

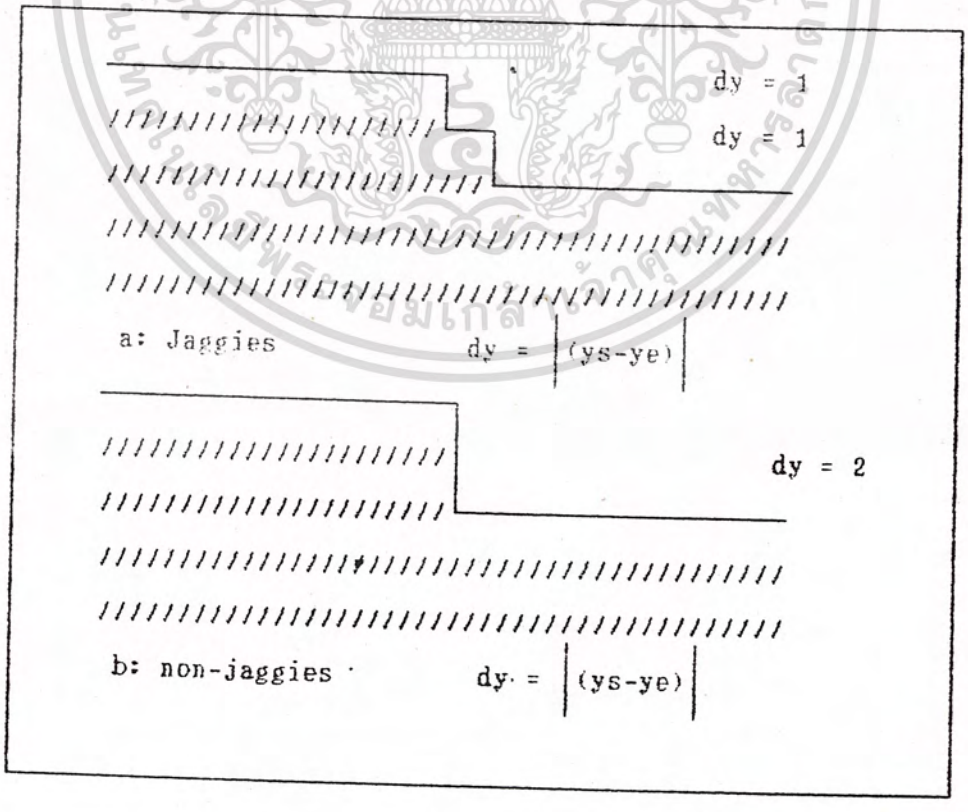


CASE C3



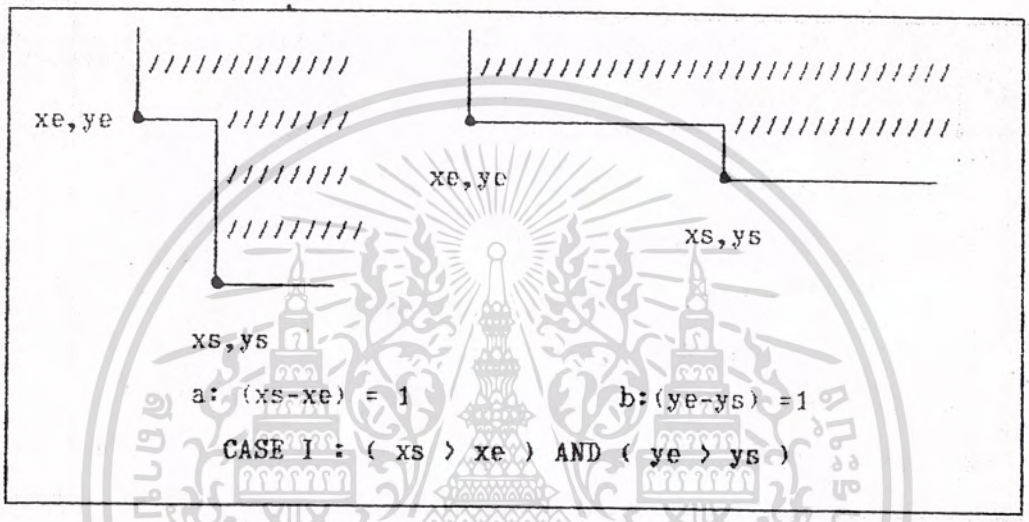
CASE C4

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

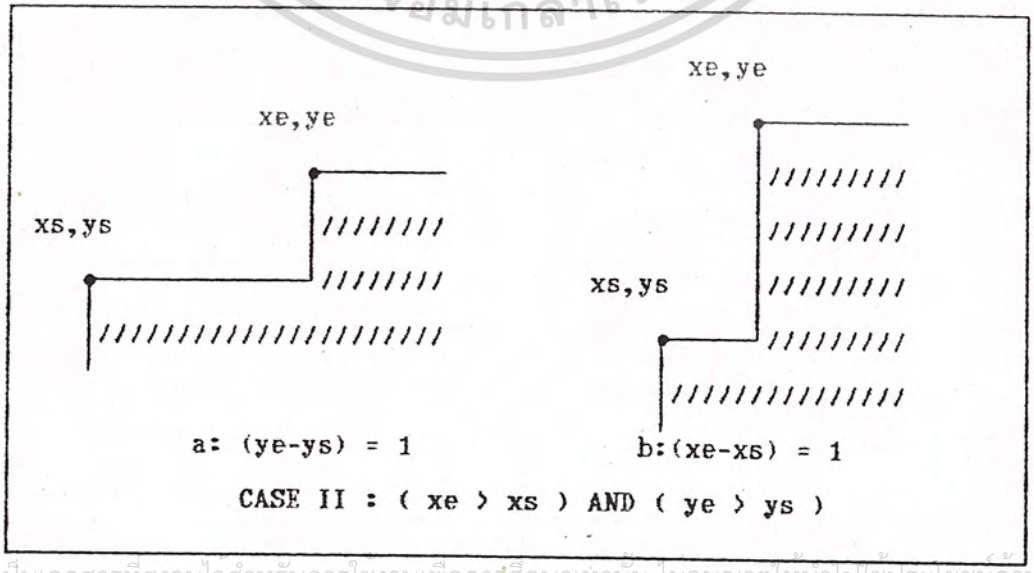


ตัวอย่าง ของ Jaggies และ Non-Jaggies

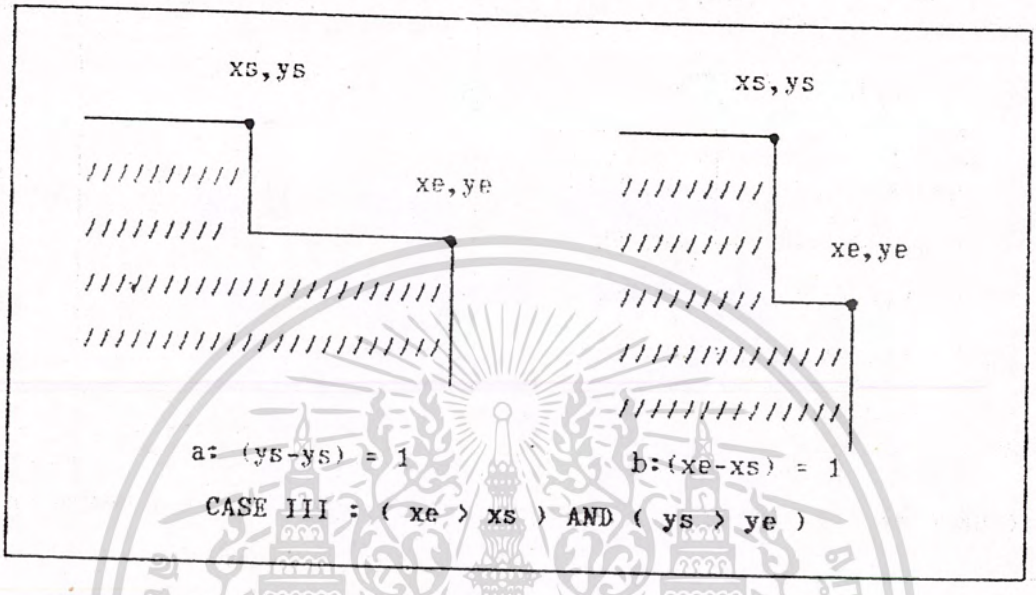
แต่ถ้ากรณีเป็นจริง แสดงว่าเป็นรอยหักและเราจะทำการ smooth ตามขอบที่กำหนด โดย (Xs,Ys) และ (Xe,Ye) ความสัมพันธ์ระหว่าง 2 จุดนี้ จะบอกเราได้ว่า pixel ไหนที่เราจะต้องเปลี่ยน shade ความสัมพันธ์ระหว่างจุด (Xs,Ys) และ (Xe,Ye) แยกออกเป็น 4 กรณี สำหรับแต่ละกรณี ทั้งจุดเริ่มต้น (Xs,Ys)และจุดสุดท้าย (Xe,Ye)เป็น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



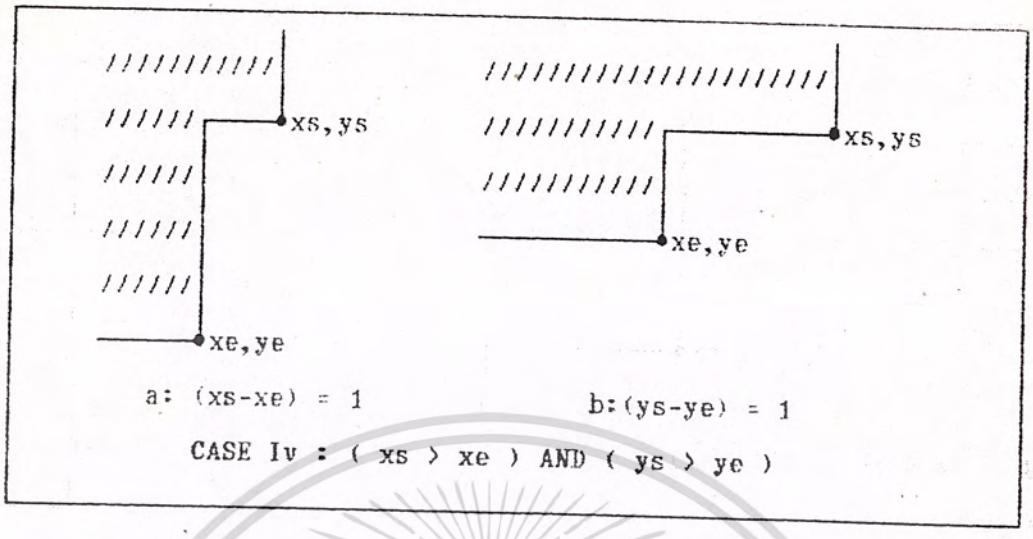
CASE I: แสดงจุดเริ่มต้น xs อยู่ทางขวามือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะต้องอยู่ข้างล่างของจุดสุดท้าย ye สำหรับ case Ia: จุด xs จะต้องอยู่ห่างจากจุด xe ไปทางขวามือ 1 pixel และ case Ib: จุด ys จะต้องอยู่ต่ำว่าจุด ye อยู่ 1 pixel



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

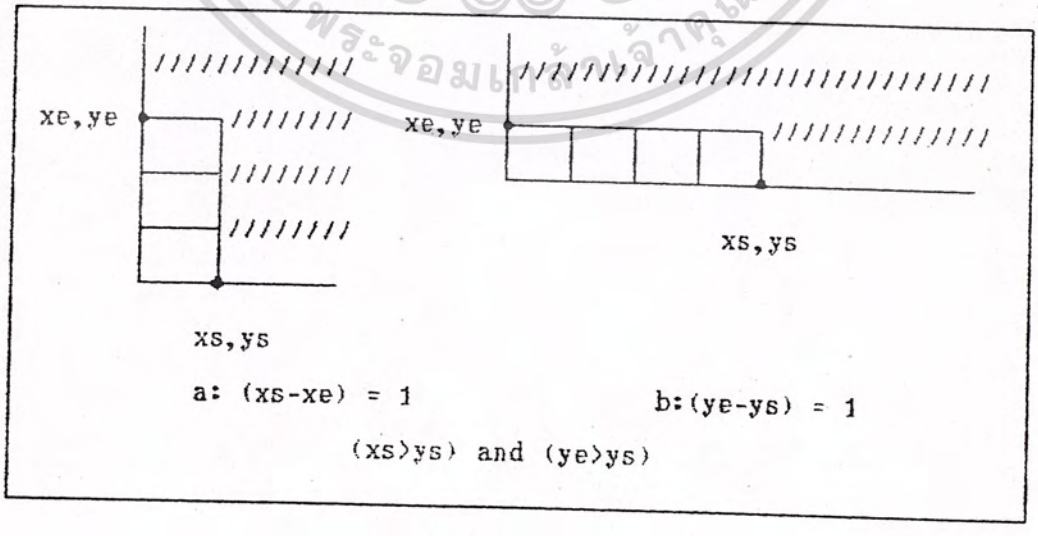


CASE III : แสดงถึงจุดเริ่มต้น xs อยู่ทางซ้ายมือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะอยู่ข้างบนจุดสุดท้าย ye สำหรับ case IIIb. จะแสดงถึงจุด xs จะต้องอยู่ทางซ้ายมือของจุด xe อยู่ 1 pixel และ case IIIa. จุด ys จะต้องอยู่เหนือจุด ye อยู่ 1 pixel

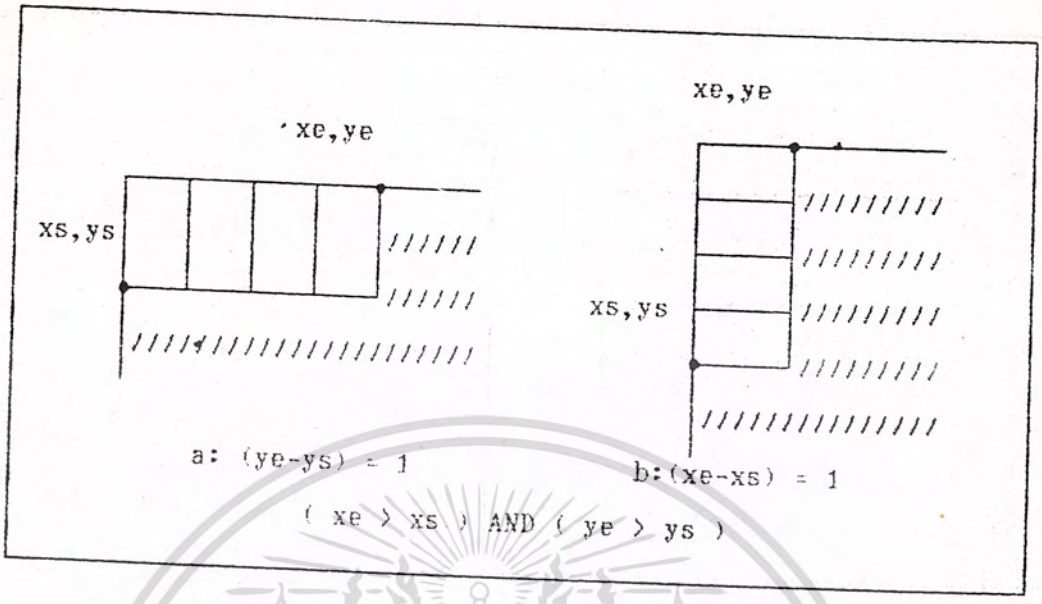


CASE IV : แสดงถึงจุดเริ่มต้น xs จะอยู่ทางขวามือของจุดสุดท้าย xe และจุดเริ่มต้น ys จะต้องอยู่ข้างบนจุด ye สำหรับ case IVa : จุดเริ่มต้น xs จะต้องอยู่ทางขวามือจุดสุดท้าย xe อยู่ 1 pixel และสำหรับ case IVb : จุด ys จะต้องอยู่บนจุด ye อยู่ 1 pixel

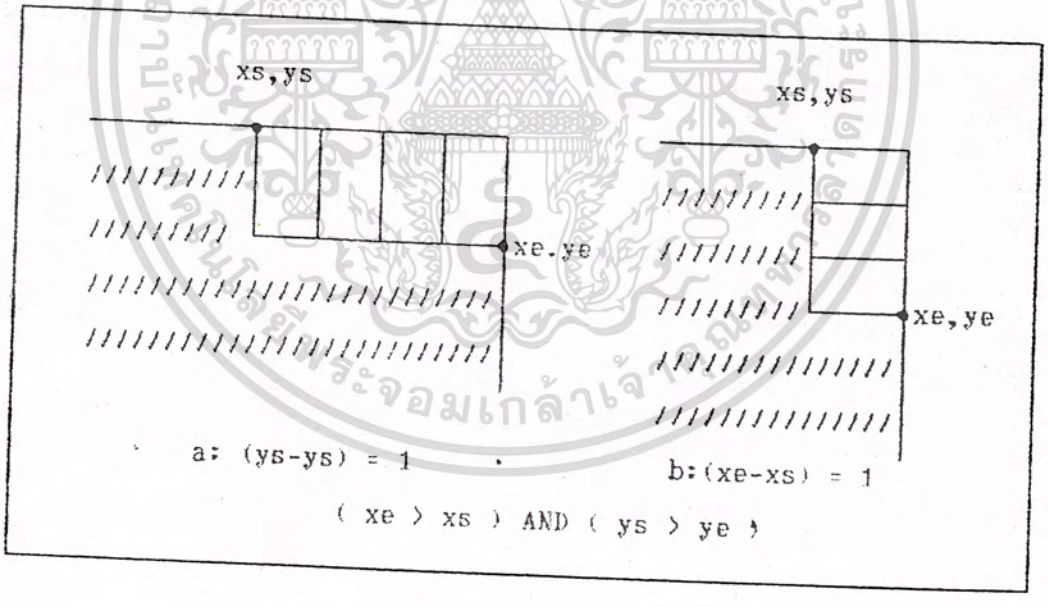
ต่อไปนี้เป็นรูปที่จะอธิบายเกี่ยวกับ pixels (yellow pixel) สำหรับ case ทั้ง 4 ที่ต้องการ smooth โดยวิธีการหาค่าตัวเลขมาคำนวณร่วมกับส่วนที่ต้องการ smooth เพื่อให้ได้ลำดับชั้นสีของ pixel ใหม่



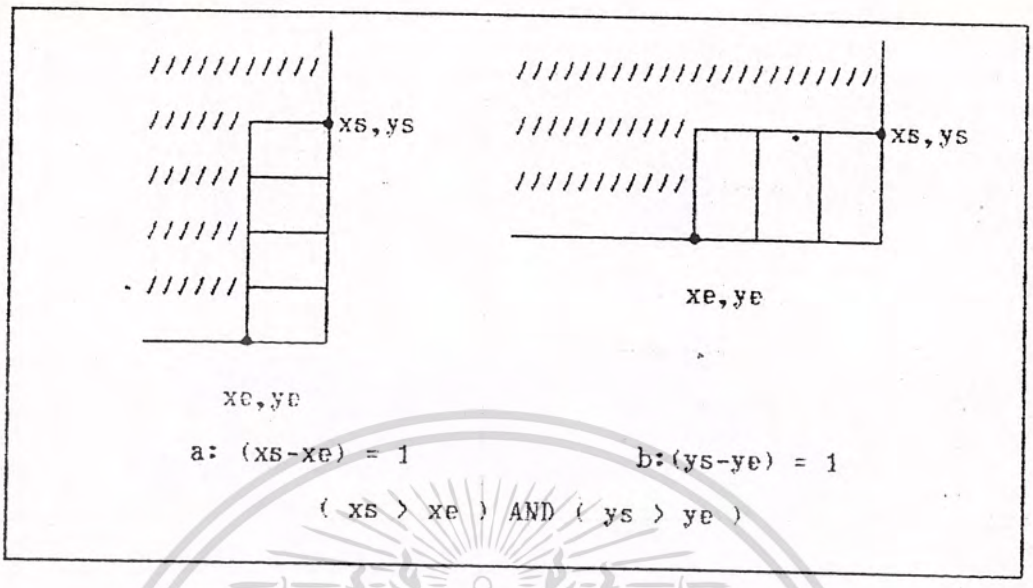
CASE I: PIXELS TO BE RESHADED



CASE II: PIXELS TO BE RESHSDED



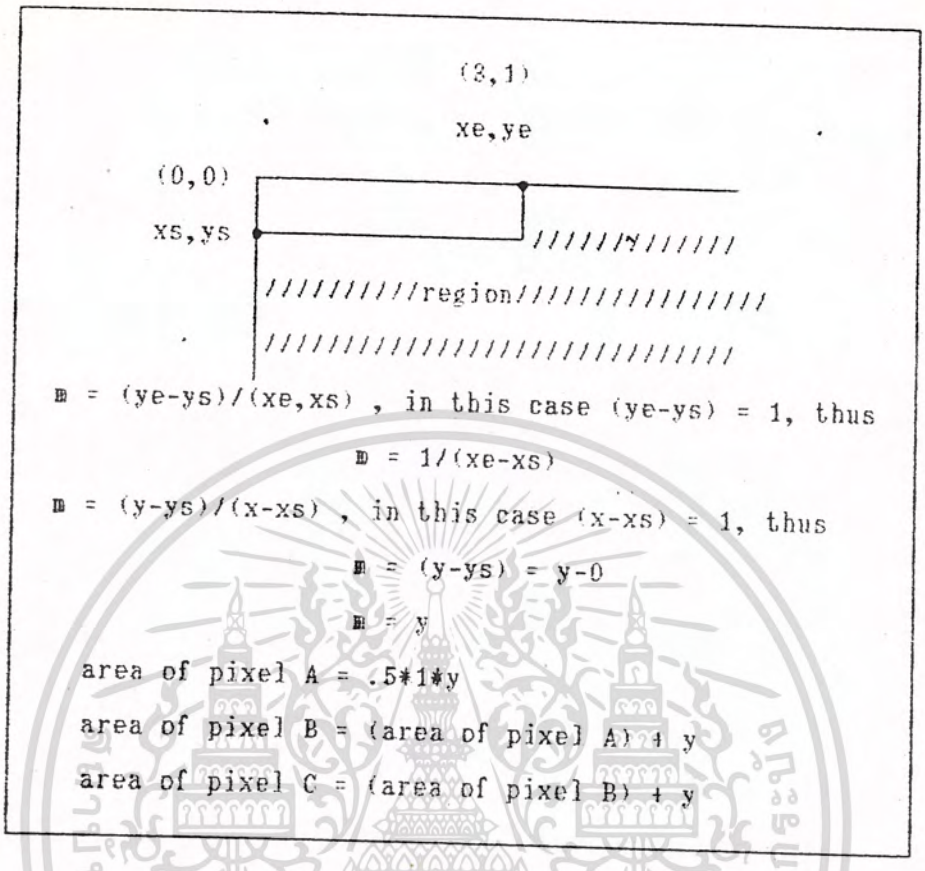
CASE III: PIXEL TO BE RESHADED



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



รูปที่ 22 REMOVING A JAGGIE BY RESHADING PIXELS

Pixel สีแดงระหว่างจุด (xs,ys) และจุด (xe,ye) ได้มาจากการคำนวณของรอยหยัก และไม่มีส่วนสีเหลืองของขอบของวัตถุ เราจะต้องจัดสีของ pixel เหล่านี้เพื่อให้แสดงออกมาเป็นส่วนของวัตถุจริง ๆ วิธีการคำนวณ ส.ป.ส สำหรับ pixel เหล่านี้จะอธิบายในรายละเอียด ส.ป.ส ของ pixel A ก็จะเท่ากับพื้นที่ของ pixel A และทำการเปลี่ยนค่า pixel A ด้วยค่าสีที่ได้จากการคำนวณโดย

(coefficient * region's color) + (1-coefficient) * pixel A's color)

ซึ่งผลลัพธ์ที่ได้มาก็คือสีแดงผสมกับสีเหลือง ดังนั้น pixel A จะไม่เป็นของวัตถุ ในทำนองเดียวกันรูปแบบการคำนวณ pixel B และ pixel C ก็หาเหมือน ๆกัน ซึ่ง pixel B จะมีสีเหลืองมากกว่า pixel A และมีสีคล้ายกับ pixel ของ object และ pixel C จะมีสีเหลืองมากที่สุดโดยจะมีสีแดงเล็กน้อย ดังนั้น pixel C ก็จะมีสีใกล้เคียง pixel object มากที่สุด

ต่อไปนี้เป็นรูปแบบการคำนวณค่า ส.ป.ส. และการจัดลำดับสีของ pixel

```
/*=================================================================*/
/*  FOR case II (xe > XS) AND (ye > ys)  */
     color = griven a color for a region  that you are working with
     IF ( ye-ys  = 1 )     /*** case IIa. ****/
        BEGIN
        m = 1.0 / ( xe-xs )
        coeff = 0.5 * m
     /* check  the pixel have already been marked as reshaded yet */
        IF ( fbv [xs][ys] = 0 )
           BEGIN
     /* draw at point xs,ys with the color = coeff*color */
        DRAW ( xs,ys, coeff*color + (1.0 - coeff)*background color)
        fbv [xs][ys] = 1   /* pixel have already been marked
                              as reshaded */
           END
       FOR ( i = xs + 1 to ( i < xe )
        BEGIN
        coeff = coeff + m
        IF ( fbv [i][ys] = 0 )   /* check  the pixel have already
                                    been marked as reshaded yet */
          BEGIN
        DRAW (i,ys, coeff*color + (1.0 - coeff)*background color)
        fbv [xs][ys] = 1    /* pixel have already been marked
                               as reshaded  */
          END
        i = i+1
        END
      END
/*=================================================================*/
```

ALGORITHM : ขั้นตอน smoothing ใน case IIa

Note : กล่นที่จะทำการเขียนอะไรไปที่จุด x,y จะต้องตรวจสอบว่า fbv[x][y] = 0 หรือไม่ ถ้า fbv[x][y]=1 แสดงว่าตำแหน่ง x,y ทำการเขียนเรียบร้อยแล้ว

การคำนวณหาค่า ส.ป.ส และการจัดลำดับสีสำหรั pixel ใน case Ia, Ib, IIb, IIIa IIIb, IVa และ IVb มีลักษณะการคิดเหมือน ๆ กัน ผลลัพธ์ของ algorithm ที่ผ่านมาเป็นวิธีที่ง่าย ๆ ในบทนี้ คือในภาพนั้นขอบของรอยหยัก จะมีการ antialiased pixel ขยบนอกขอบเดิม ดังนั้นวัตถุทั้งหมดใน picture มีขอบที่ติดกัน ซึ่งวิธีการแบบนี้ไม่ค่อยถูกต้องนัก จะทำให้ภาพที่ได้ blurred หรือมัวมาก ๆ ซึ่งเราไม่ต้องการภาพลักษณะนี้

ในบทต่อไปเป็นการหาค่า solution ของปัญหานี้

บทที่ 5 การแก้ปัญหาการ ANTIALIASING ซ้ำกัน

เราย้อนกลับไปที่ขั้นตอนที่ 3 ของขบวนการ antialiasing หน้าที่ของขั้นตอนที่เกี่ยวข้องกับตำแหน่งของรอยหยักบนขอบของวัตถุและการ smooth ขอบ ซึ่งรอยหยักที่พบได้จัดลำดับสี pixel ตามขอบเหล่านั้น รูปแบบการจัดลำดับสี pixel ตามขอบของวัตถุ เป็นเพียงขั้นตอนแรกเท่านั้น และไม่ได้ทำการพิจารณาส่วนขอบของวัตถุ 2 วัตถุ เพียงแต่จัดลำดับสีใหม่ครั้งเดียว

ในบทนี้จะเป็นการปรับปรุงรูปแบบการจัดลำดับสี อย่างง่าย ๆ ดังนั้นการแบ่งส่วนของขอบกระทำเพียงครั้งเดียว

วิธีที่จะกล่าวต่อไปนี้ใช้สำหรับการคำนวณค่า pixel ที่ผ่านการจัดลำดับสีเรียบร้อยแล้ว วิธีพื้นฐานที่ได้ถูกพัฒนาโดย JULES BLOOMINTHAL

BLOOMENTHALได้กล่าวไว้คือ data structure จะเรียก "fbv buffer" ซึ่งเก็บรายละเอียดที่เกี่ยวกับ pixels ไว้ในแบบจำลองที่ผ่านขบวนการ antialiasing แล้ว ซึ่งได้นำ data structure ของ BLOOMENTHAL มาใช้ใน algorithm เพื่อลบล้างส่วนเกินของลำดับสี algorithm ในบทก่อน ประกอบด้วยพื้นฐาน 2 step คือ

1. ณ ที่จุด object's corner points ทำการหารอยหยักตามขอบ
2. การผ่านรอยหยักการจัดลำดับสี pixel ตามขอบใหม่

ทั้งสองอย่างนี้ได้ถูกนำมาปรับปรุงใช้"fbv buffer"ของ process นี้

ขณะนี้ object corner points เริ่มต้นพิจารณาที่ ขอบของ pixels บนขอบของวัตถุก็ทำการคำนวณจาก corner point information และเก็บไว้ใน "fbv buffer" ดังนั้นเมื่อ pixels ด้านนอกของขอบ object location ของ pixel เหล่านี้คือ ค่าใน "fbv buffer" ในวิธีนี้ object records มันจะมีการ antialiased pixel บนรอบนอกของ region และมีการป้องกัน pixels ด้านในด้วยเหมือนกัน จากการเริ่มต้น antialiasing ส่วนแบ่งของขอบ objects อื่น ๆ โดย algorithm ของบทก่อนคือการ antialiasing โดยการประมาณ ดังนั้นแต่ละขอบของวัตถุต้อง antialiasing ก่อนผ่านขบวนการวัตถุต่อไป และเหล่านี้คือ การรวมวิธีการปรับปรุงให้ดีขึ้นโดยใช้ "fbv buffer" ในขบวนการนี้ ซึ่งขณะนี้รอยหยัก ถูกหาตามขอบของวัตถุต่อไป และก็ยังคงคำนวณกับ pixel เพื่อจัดลำดับสีตามขอบของรอยหยัก ก่อนที่จะลำดับสี pixel จะทำการตรวจสอบไปที่ "fbv buffer" ถ้า pixel ใน fbv buffer นี้ผ่านการ antialiasing แล้วจะข้ามขอบนี้ไป กรณีอื่น ๆ การ antialiasing และการจัดลำดับสี pixel ที่เรียบร้อยแล้วจะถูก markไว้ใน fbv buffer ซึ่งใน case อื่น ๆ ก็จะ marked pixel เสมอจากวัตถุอื่น ๆเสมอ ซึ่งการปรับปรุงนี้จะเป็นการป้องกันการจัดลำดับสีซ้ำขอบเดิมใน algorithm ก่อน

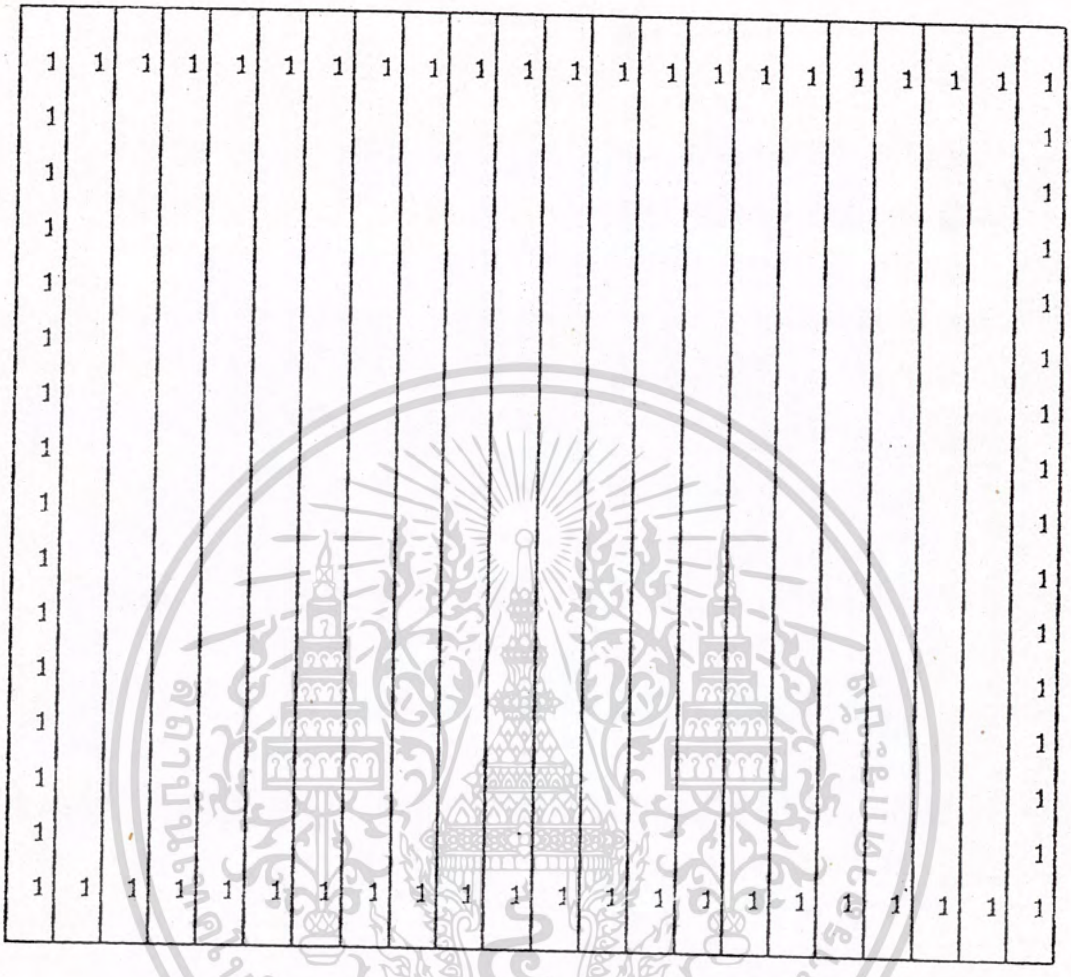
ต่อไปนี้การอธิบายการใช้ "fbv buffer" เราจะพิจารณาการจัดลำดับสีของภาพตามลำดับ

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	1	1	1	1	1
1	1	1	1	1	1	1	2	2	2	1	1	1	3	3	3	3	3	1	1	1	1
1	1	1	1	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	1	1
1	1	1	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	1	1
1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	1	1
1	1	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	1	1	1
1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	3	3	1	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

รูปที่ 5.1 ภาพของ tag buffer ซึ่งเป็นวงกลม 2 วงกลม overlap กัน

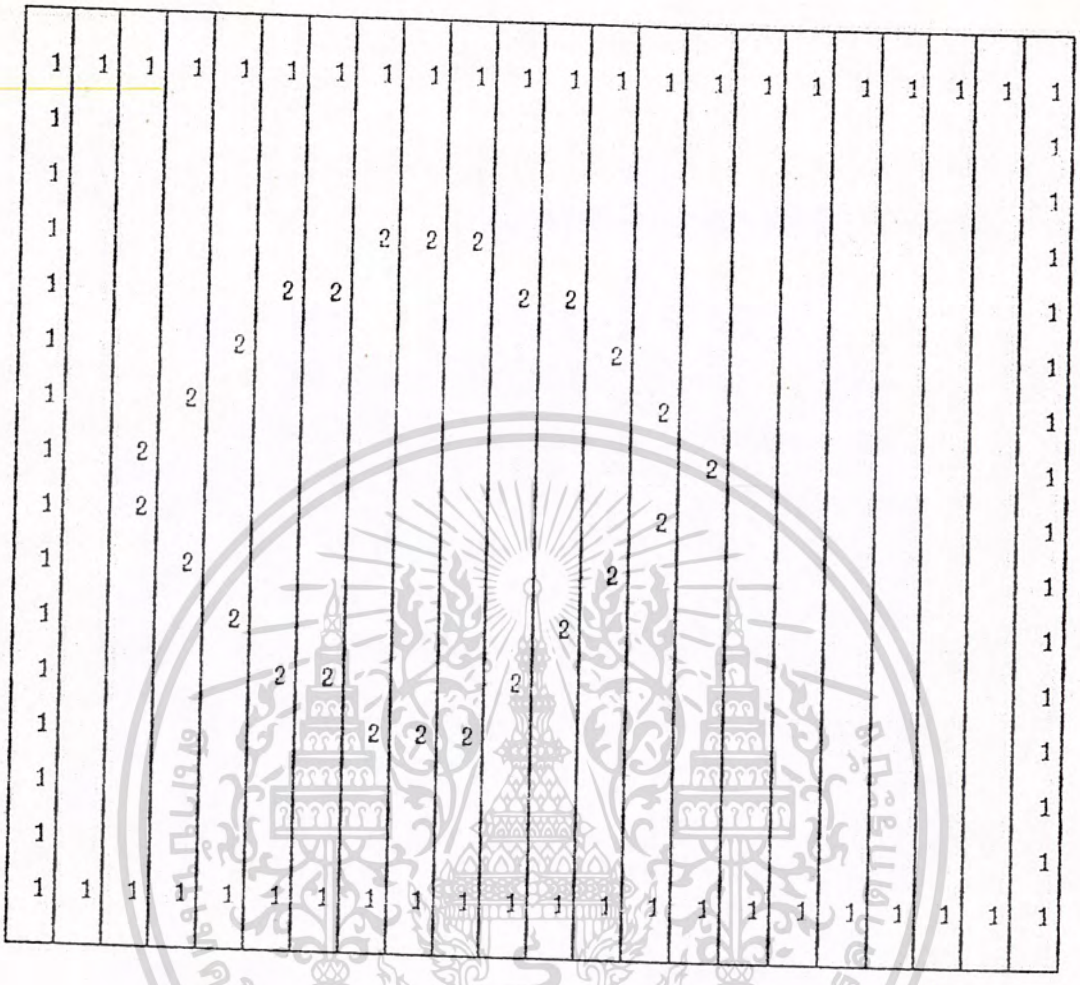
ในรูปที่ 23 จะเห็นได้ว่ามี tag buffer image 2 object คือวงกลมสีแดงและวงกลมสีเหลืองทับกันอยู่ และมี background สีน้ำเงิน ซึ่งจะมีอยู่จริง 3 objectในภาพ ดังนั้น back ground ของภาพ จะพิจารณาที่ blue pixels ก็เป็นวัตถุด้วย ก่อนที่จะเริ่มต้นการจัดลำดับสี "fbv buffer" ให้มีขนาดเท่ากับ tag bufferโดยกำหนดค่าเริ่มแรกเท่ากับ 0 ทั้งหมด ซึ่งแสดงให้เห็นว่าไม่มี pixel ใด ๆ ผ่านการจัดลำดับสีมาแล้ว ในการเริ่มต้นขบวนการจัดลำดับสี โดยจะพิจารณาที่ corner point information ของ วัตถุแรก,วัตถุแรก ทำการหารอยหยักตามขอบของ pixel บนรอบ ๆ ของวัตถุที่ 1 เก็บไว้ใน fbv buffer แสดงในรูป 5.2

Note : ใน "fbv buffer" จะประกอบด้วยตัวเลขสองตัวคือ 0 และ 1 ซึ่ง 0 หมายถึง pixel ที่ยังไม่ผ่านการจัดลำดับสี 1 หมายถึง pixel ที่ผ่านการจัดลำดับสีเรียบร้อยแล้วในตัวอย่างนี้ใช้ 0 เป็น blank และ 1เป็น non-blank เรามี label fbv ในระบบนี้ช่วยในการ clarity diagrams ตามมา



รูปที่ 5.2 ส่วนประกอบของ FBV BUFFER ของขอบ pixel ของ object 1

โครงสร้างของ corner point สำหรับวัตถุแรก ยอมให้ไม่มีรอยหยักตามรอบขอบของวัตถุ ดังนั้นจึงไม่มี pixel ใดได้รับการจัดลำดับสี ทำให้ fbv bufferไม่มีการเปลี่ยนแปลงไปจากรูปที่ 5.2 เมื่อเราทำขั้นตอนนี้ซ้ำกับวัตถุที่ 2 คือ yellow circle จุดแรก corner point คือแนวทางที่จะทำการหารอยหยัก และ pixelรอบ ๆ จะถูก marked ใน fbv bufferดังรูปที่ 5.3

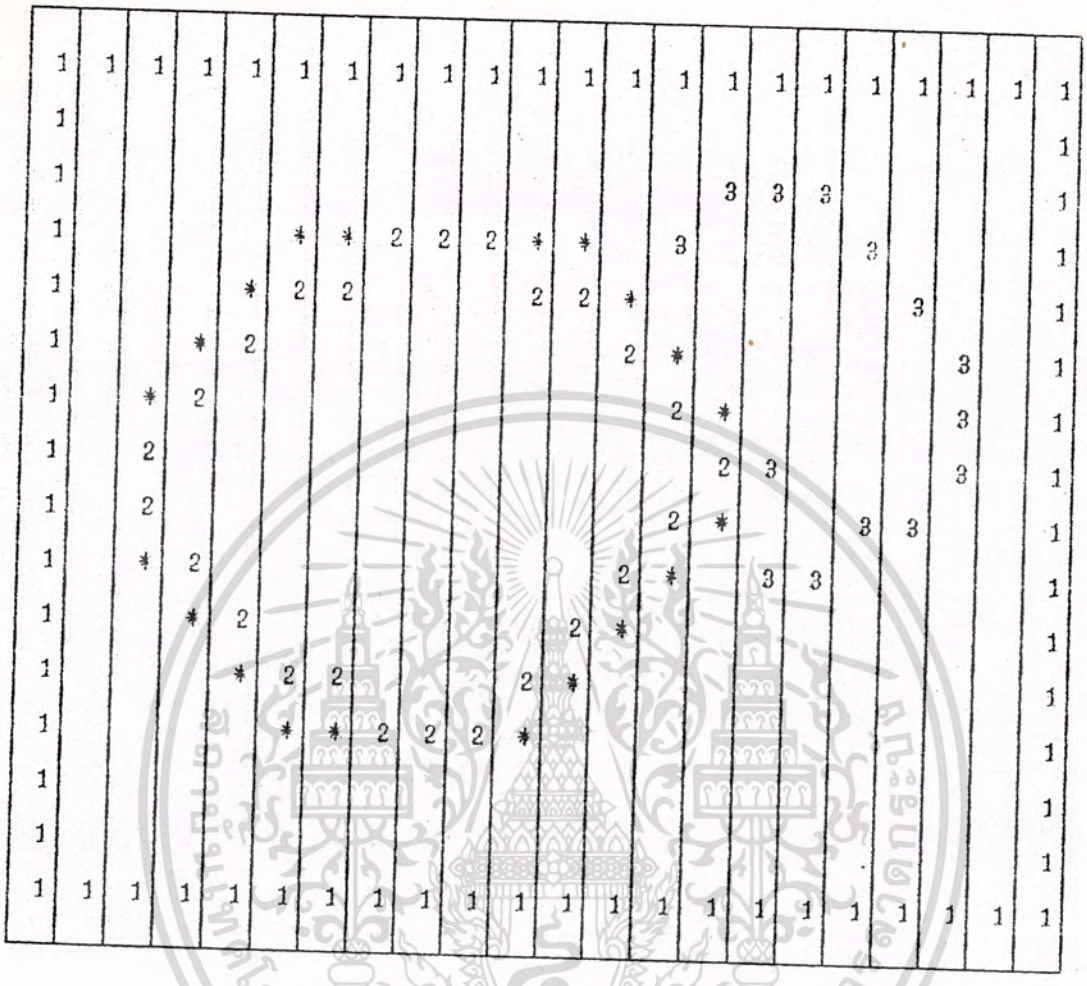


รูปที่ 5.3 ส่วนประกอบของ FBV BUFFER หลังจาก marked ขอบ pixel ของ object 2

โครงสร้างภายนอกของจุดมุมจะมีรอยหยัก ตามขอบของวัตถุ 2 และ pixel เหล่านั้นต้องการที่จะจัดลำดับสีใหม่ การตอบสนอง location ใน fbv buffer แล้วจะทำการตรวจสอบ pixel ก่อนก้า pixel เหล่านั้นผ่านการจำลำดับสีเรียบร้อยแล้ว(การจัดลำดับสี pixel แล้วจะ mark ค่า 1 ไว้) ที่สภาวะนี้ ไม่มี pixel ยกเว้นรอบขอบเขตของวัตถุ 1 และ 2 ผ่านการ mark ใน fbv buffer แสดงในรูปที่ 5.3 มีขอบไม่ธรรมดาสำหรับขอบเขตของ 2 วัตถุ อย่างไรก็ตาม pixel ทั้งหมดด้านนอกขอบเขตของวัตถุ 2 จะมีการเปลี่ยนแปลงการจัดลำดับสีใหม่ และ pixels ที่จัดลำดับสีใหม่เพื่อ smooth รอยหยักจะ mark ไว้ใน fbv buffer เป็น 1 ดังรูปที่ 5.4

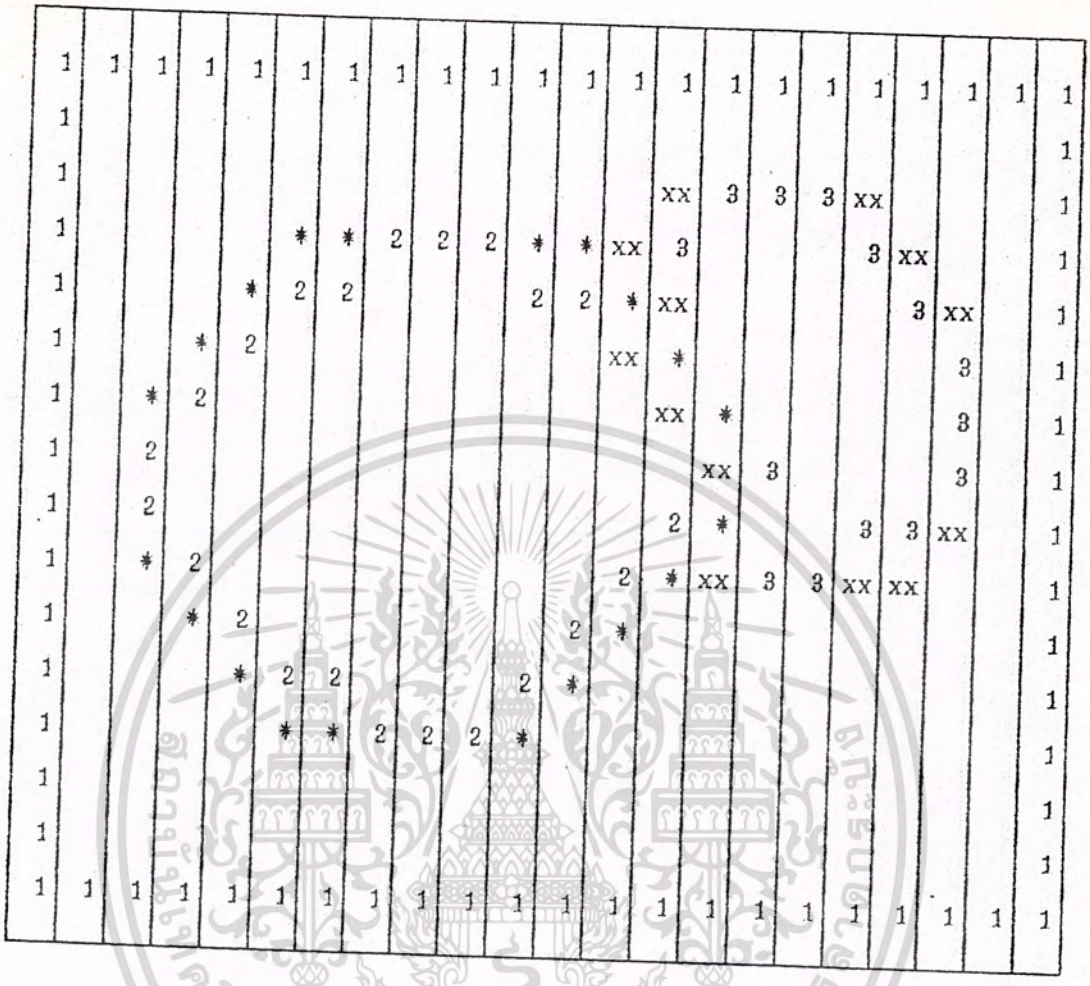
รูปที่ 5.4 ส่วนประกอบของ FBV BUFFER หลังจาก รอยหยักของ object 2 หลังจากผ่านการ smooth แล้ว

เราจะกล่าวถึงวัตถุที่ 3 ต่อไป ซึ่งเราพิจารณา corner points information และการ mark ขอบ pixels บนขอบเขตของวัตถุที่ 3 แต่ว่าส่วนแบ่งขอบระหว่างวัตถุ 2 และ 3 จะมีบาง pixel ที่ผ่านการจัดลำดับสีใหม่เรียบร้อยแล้ว ดังนั้น pixels ที่มี label 3 เท่านั้นที่ทำการ mark ขอบ pixels สำหรับวัตถุที่ 3 ดังแสดงในรูป 5.5



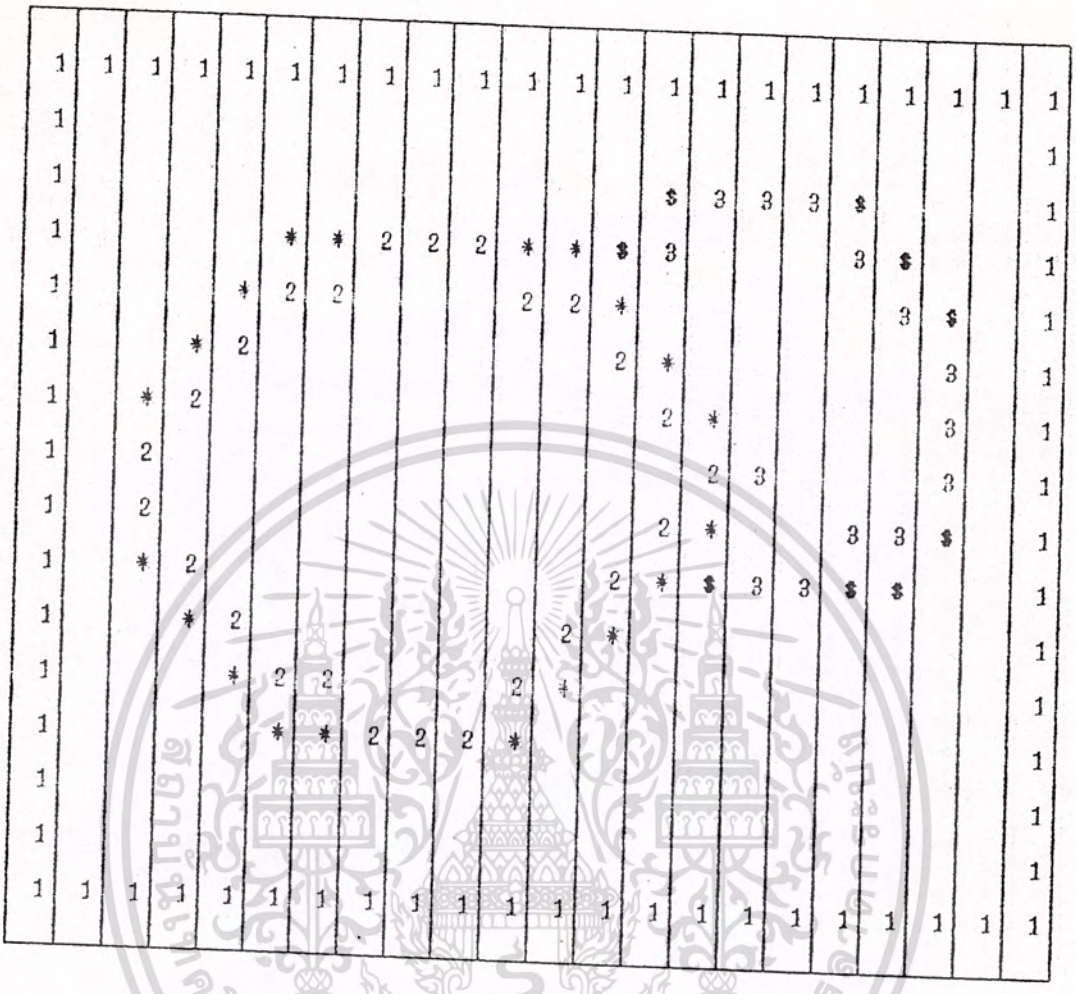
รูปที่ 5.5 ส่วนประกอบของ FBV BUFFER หลังจาก marked pixel ของขอบ object 3

โครงสร้างภายนอกของจุดมุมจะมีรอยหยัก ตามขอบของวัตถุ 3 และ pixel ซึ่งควรจะจัดลำดับสีใหม่เพื่อย้ายรอยหยักตามขอบของวัตถุที่ 3 ดังที่แสดงด้วยสีแดง ดังรูปที่ 5.6



รูปที่ 5.7 pixel xx เป็นส่วน ที่จะทำการ smooth ของ object 3

แต่มันสามารถ cleared pixel บางตัวที่ผ่านการ mark เรียบร้อยแล้วใน fbv buffer pixel ของวัตถุที่ 2 ผ่านการจัดลำดับสีใหม่ตามขอบ ดังนั้น pixel ในสีแดงไม่จำเป็นต้องจัดลำดับสีใหม่อีก ดังค่าใน fbv buffer ได้แสดงไว้ และ red pixls อื่น ๆ ที่ไม่ใช้ส่วนของขอบก็ไม่ต้องจัดลำดับสีใหม่ ส่วนที่ทำการจัดลำดับสีใหม่จะ mark $ ใน fbv buffer ดังแสดงในรูปที่ 5.8



รูปที่ 5.8 ส่วนประกอบของ FBV BUFFER หลังจาก smooth รอยหยัก ของ object 3

ดังนั้น จึงไม่มีวัตถุใดที่จะมาทำการ proces อีก ดังนั้น การ antialiasing ของขอบสำหรับภาพนี้จึงจบสมบูรณ์ นั่นคือการ clear ให้สมำ่เสมอใน fbv buffer ภาพของรูปที่ 5.8 ดังนั้นขอบของวัตถุที่ 2 และวัตถุที่ 3 จึงดูเหมือนว่ามีรอยขรุขระลดน้อยลง รูปที่ 5.9 แสดงถึงการ antialiasing ภาพเรียบร้อยแล้ว ซึ่งเป็นผลที่ได้มาจากภาพ tag buffer รูปที่ 5.1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	1	1	1	1	1
1	1	1	1	1	1	1	2	2	2	1	1	1	3	3	3	3	3	1	1	1	1
1	1	1	1	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	1	1
1	1	1	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	1	1
1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	1	1
1	1	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	1	1	1
1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	3	3	1	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

รูปที่ 5.9 ผลที่ได้จากการ antialiased ของรูปที่ 5.1

PSEUDOCODE สำหรับการปรับปรุง algorithm ที่ใช้สำหรับ smooth ของ ของวัตถุทั้งหมด ดังต่อไปนี้

```
-initialize all the values of the fbv buffer to 0
-FOR i=0 to (i <= objectnumber)
    BEBIN
    Fill the contents of the fbv along the inside edge of an
    object j
    Call the smoothing procedure to smooth the outside edge
    of an object j
    END
```

ALGORITHM: EDGE SMOOTHING FOR ENTIRE IMAGE,
MODIFIED TO USE THE FBV BUFFER WHEN RESHADING

จะยังไม่กำค่าทั้งหมดใน fbv buffer เป็น 0 หรือ 1 ก็ได้ แต่ถ้าเรา set ค่า initial ทั้งหมดใน fbv bufferให้เป็น 0 เมื่อ pixel บน image buffer มีการจัดลำดับสีใหม่ ก็จะมีผลให้ fbv buffer ที่สามารถจัดลำดับสีของภาพใหม่ได้ ดังนั้น วิธีนี้จึงเป็นวิธีที่ป้องกันไม่ให้ pixel ของภาพ สามารถจัดลำดับสีใหม่อีกครั้ง ตลอดทั้งหมดในการใช้ fbv buffer ก็สิ้นสุดสภาวะที่ 3 และเป็นสภาวะสุดท้ายของการแก้ปัญหา antialiasing ส่วนของขอบภาพให้ดีขึ้น

สรุปผล

Algorithm ที่ได้แสดงในบทก่อนๆจะยอมให้ image ที่ถูกสร้างขึ้นก่อนหน้า สามารถที่จะนำมาทำการ antialiasing ได้

โดยทั่วไปอุปกรณ์แสดงผลแบบ raster scan จะจำกัดขีดความสามารถในการวาดภาพ (image) ที่ต่อเนื่อง ลงในช่องตารางย่อยๆ ของอุปกรณ์แสดงผลแบบ raster scan

ดังนั้น algorithm นี้จะทำงานบน image หลังจากที่ได้รับการกำหนดเรียบร้อยแล้วซึ่งข้อมูลของความต่อเนื่องดั่งเดิมของ image ได้สูญหายไปแล้ว และขอบหลายส่วนของ image ที่ต่อเนื่องได้เลื่อนตำแหน่งไป

Algorithm นี้จะทำการ antialiasing บริเวณขอบซึ่งขึ้นอยู่กับ ตำแหน่งใน discrete image grid Bloomenthal's algorithm ทำการแก้ปัญหานี้โดยการพิจารณาตำแหน่งดั้งเดิมของขอบ จากข้อมูลที่ได้รับจาก ขอบของ image แบบ discrete ผลกระทบต่อขนาดของ antialiasing โดยการเพิ่ม pixels ด้านนอกขอบของขอบวัตถุ ดังนั้น ภาพที่ได้จะใหญ่กว่าภาพเดิม ซึ่ง bloomenthal's algorithm จะไม่สนใจในปัญหานี้

ใน algorithmนี้ กระทำโดยการแสดง coordinate ของขอบใหม่ ซึ่งไม่จำเป็นเสมอไปว่าจะต้องเพิ่ม pixel รอบนอกของวัตถุ และขอบ antialiased โดยการ วาดซ้ำที่ขอบของวัตถุจาก pixel ใหม่ระหว่าง coordinate เหล่านี้ ดังนั้นการ antealiasing บนขอบเขตของวัตถุจะถูกแทนที่ด้านนอกขอบเขตเสมอ

การเปรียบเทียบข้อดีข้อเสียของ Bloomenthal's algorithm กับ algorithm ของผม algorithm ของผมนี้ มีประสิทธิภาพดีกว่า สามารถยืดหยุ่นใช้จอภาพ VGA ที่ให้ค่าสีได้หลายระดับ และการ antialiasing ของวัตถุ ของ Bloomenthal ทำให้เกิดการเพิ่มขนาดของภาพโดยเพิ่มจำนวน pixels การเพิ่มนี้เพิ่มเฉพาะขอบของ object ใน images ทั่วไปทำให้ขอบของภาพหนาขึ้นได้

ภาคผนวก

```
/*=====================================================================
MODULE  : Antialising.
PURPOSE : Filter the graphic image with the finess algorithm.
===================================================================== */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define VGA256  0x13
#define MAXOBJECT 16
#define MAXFTAB     MAXOBJECT*8
#define XMAX 160
#define YMAX160
#define CPOINTMAX1000 /*10922*/ /* (64*1024)/(2*3) */

/*=====================================================================*/
/*================== Function prototype ===========================*/
/*=====================================================================*/
int get_video_mode(void);                   /* Get current video mode .*/
int set_video_mode(int mode);               /* Set video mode to 'mode' .*/
int init_gray_scale(void);  /* Fill palette table with wighted gray .*/
int init_RGB(void);  /* Fill palette table with wighted R,G,B .*/
int putpel(int color, int x, int y);/* Put pixel 'color' to (x,y) use BIOS.*/
int getpel(int x, int y);          /* Get pixel 'color' from (x,y) use BIOS .*/
int dputpel(int color, int x, int y);/* Put pixel 'color' to (x,y) direct .*/
int dgetpel(int x, int y);           /* Get pixel 'color' from (x,y) direct .*/
int fill_block_slide_vga256();
   /* Fill VGA256 screen with 16*16 rectangular block .*/
int direct_fill_vga256(int color);
   /* Fill VGA256 screen with 'color' direct .*/
int blue_x(void); /* Display big blue X .*/
```

```
int demonstrate_antialiasing(void);  /* Demonstration routine .*/
int display_illustrate_image(void);  /* Fill screen with illustrate image .*/
int antialias(int xs, int ys, int xe, int ye); /* Antialias routine .*/
int descript_objects(int xs, int ys, int xe, int ye);   /* (1) .*/
int extract_objects(int jj);   /* (2) .*/
intdesire_and_antialias(unsigned cpoint_count);    /* (3) .*/

/*=====================================================================*/
/*================ GLOBAL VARIABLE DECLARATION ====================*/
/*=====================================================================*/

struct start_point{/* Coordinate information. */
int x;
int y;
int color;
};

struct corner_info{/* Concavity information. */
int x;
int y;
int convex;
};

intj, ii, cpoint_count, ftab[MAXFTAB];
unsigned intobjnum;
unsigned char tag[XMAX][YMAX];
unsigned charfbv[XMAX][YMAX];
struct start_point ctab[MAXOBJECT];
struct corner_info cpoint[CPOINTMAX];

/*=====================================================================*/
/*=========================== main program ============================*/
/*=====================================================================*/
```

```
main()
{

int xx, yy, original_mode;
   /* Variable which preserve the original display mode. */
original_mode = get_video_mode();/* Preserve original display mode before
  running demonstration program. */
set_video_mode(VGA256);/* Set video display mode to VGA 256 COLORs which
resolution is 320 * 200. */
/* Initiailize fbv with -1 . */
for(xx=0;xx<XMAX;xx++){for(yy=0;yy<YMAX;yy++){fbv[xx][yy]=0;}}
display_illustrate_image();
antialias(0, 0, XMAX, YMAX);
getch();
set_video_mode(original_mode);/* Restore the original video display mode
   which active before the demosteration program is run. */
getch();
/*printout();*//* Printout tag, ftab, ctab in numeric form. */
}
/*=============================================================*/
/*=======  antialias(int xs, int ys, int xe, int ye) ==========*/
/*=============================================================*/
int antialias(int xs, int ys, int xe, int ye)     /* Antialias routine. */
{
int temp, i;

 /* Prepare suitable start/end coordinate. */
if(xe<xs){ temp = xs; xs = xe; xe = temp; };
if(ye<ys){ temp = ys; ys = ye; ye = temp; };

 /* Body of antialiasing routine. */
  objnum = descript_objects(xs, ys, xe, ye);/* (1) Fill tag, ftab & ctab. */
for(i=1;i<=objnum;i++)
```

```
{ii = cpoint_count =  extract_objects(i);/* (2) Fill cpoint. */
desire_and_antialias(cpoint_count);/* (3) Antialias jaggies. */
}
 /* End body of antialiasing routine. */
}
/*=====================================================================*/
/*======(1)descript_objects(int xs, int ys, int xe, int ye)=========*/
/*=====================================================================*/
descript_objects(int xs, int ys, int xe, int ye)

#definetagC tag[x-xs][y-ys]
#definetagR    tag[x+1-xs][y-ys]
#definetagD    tag[x-xs][y+1-ys]
#definetagDR   tag[x+1-xs][y+1-ys]
#define tagXY   tag[x-xs][y-ys]
#defineftabC ftab[c]
#defineftabR  ftab[r]
#defineftabD  ftab[d]
#defineftabDR  ftab[dr]
{
register int x, y, c, d, r, dr, a, b;

tag[0][0] = objnum = 0;

y = ys;                              /* First row fill algorithm. */
for(x=xs; x<xe-1; x++)
{   a = dgetpel(x+1, y);
b = dgetpel(x, y);
if(a==b)
{tag[x+1-xs][0] = tag[x-xs][0];}
else
{  tag[x+1-xs][0] = ++objnum;  };
}                                    /* End first row fill algorithm. */
```

```
    x = xs;                                  /* First column fill algorithm. */
    for(y=ys; y<ye-1; y++)
    {if( dgetpel(x, y+1) == dgetpel(x, y) )
    {tag[0][y+1-ys] = tag[0][y-ys]; }
    else
    {  tag[0][y+1-ys] = ++objnum;  };
    }
                                  /* End first column fill algorithm. */

    for(c=0;c<MAXFTAB; c++) ftab[c] = -1; /* Initialize ftab with -1. */

    for(y=ys; y<ye-1; y++) /* Fill rest of tag buffer & ftab.  */
    {for(x=xs; x<xe-1; x++)
    {     c =  dgetpel(x, y);
      r =  dgetpel(x+1, y);
      d =  dgetpel(x, y+1);
     dr =  dgetpel(x+1, y+1);

    if((d == c)&&(tagD != tagC))
    {if(tagD<tagC) {ftab[tagC] = tagD;}
    else {ftab[tagD] = tagC;}
    }
    else if((d == r)&&(tagD != tagR))
    {if(tagD<tagR) {ftab[tagR] = tagD;}
    else {ftab[tagD] = tagR;}
    }
    if(d == dr)      tagDR = tagD;
   else if(c == dr)tagDR = tagC;
   else if(r == dr)tagDR = tagR;
   else tagDR = ++objnum;
   }
   }                  /* End fill rest of tag buffer & ftab.  */
```

```
    /* Preview redundance relative object number ftab. */
 for(a=0; a<(MAXFTAB); a++)
 {if( (ftab[a] != -1)&&(ftab[ftab[a]] != -1) ) { ftab[a] = ftab[ftab[a]];}
 }

 j = -1;                /* Fill the ctab & update tag.  */
 for(y=ys; y<ye; y++)
 {for(x=xs; x<xe; x++)
 {   c = tagXY;
 d = ftab[tagXY];
 if((d == -1) || (c == d))
 {j++; ctab[j].x = x; ctab[j].y = y;
 ctab[j].color = dgetpel(x,y);
 ftab[tagXY] = -2;    /* Only one start point per object.  */
 }
 if(d > -1)/* If tagXY not unique fill it with relative objnum. */
 { tagXY = d;
 }
 }
 }   /* End fill the ctab & update tag.  */

 b = -1;
 for(a=0;a<(MAXFTAB)+1;a++){if(ftab[a] == -2) b++; }
 return(b); /* Actual object number( 0,1,2,.. ). */
 }
 /*=================================================================*/
 /*==========(2)  int extract_objects(int jj)=======================*/
 /*=================================================================*/
 int extract_objects(int jj)/* (2) */
 {
 #define  CONVEX   1
 #define CONCAVE   0
 #define NE (cpoint[i].x!=x)||(cpoint[i].y!=y)||(cpoint[i].convex!=convex)
```

```
#define save_cp  if(NE){cpoint[++i].x=x;cpoint[i].y=y;cpoint[i].convex=convex
#define save_c1  convex=CONVEX;save_cp
#define save_c0  convex=CONCAVE;save_cp

 int i, object, convex;
register int    x, y, xs, ys;

i =  0;
y = ys = ctab[jj].y;
x = xs = ctab[jj].x;
   cpoint[0].x = x;
   cpoint[0].y = y;
  cpoint[0].convex = CONVEX;
   object =  tag[x][y];

/* Edge tracking anti-clockwise direction from top-left start point. */
#define  C   tag[x][y]
#define  R   tag[x+1][y]
#define  L   tag[x-1][y]
#define  U   tag[x][y-1]
#define  D   tag[x][y+1]
#define     UR   tag[x+1][y-1]
#define     UL   tag[x-1][y-1]
#define     LR   tag[x+1][y+1]
#define     LL   tag[x-1][y+1]

DOWN:if(D == object)
{y = y+1;
if(L == object)
{ save_c0; goto DOWN_1;}/*Case up-b:*/
else goto DOWN ;
}
else{save_c1; }/* Case up-a: */
```

```
DOWN_1:if(convex == 0) goto LEFT;

RIGHT:          if(i >=CPOINTMAX) {return(i);};/* Overflow check. */
if(R == object)
{x = x+1;
if(D == object)
{ save_c0; goto RIGHT_1;}/* Case right-a: */
else {goto RIGHT;}
}
else{save_c1; }/* Case right-b: */
RIGHT_1:        if(convex == 0) goto DOWN;

UP:if(U == object)
{y = y-1;
if(R == object)
{ save_c0; goto UP_1;}/*Casedown-b:*/
else{goto UP;}
}
else{ save_c1;}/*Case down-a:*/
UP_1:if(convex == 0) goto RIGHT;

LEFT:if(L == object)
{x = x-1;
if(U == object)
{ save_c0; goto LEFT_1;}/*Case left-b:*/
else{goto LEFT;}
}
else{ save_c1; }/*Case left-a:*/
LEFT_1:if(convex == 0) goto UP;

END:if( (x==xs)&&(y==ys) ) {goto DONE;}
else {goto DOWN;}
```

```
DONE: ;

return(i);
}
/*=================================================================*/
/*======(3)  int desire_and_antialias(unsigned cpoint_count);=======*/
/*=================================================================*/
int desire_and_antialias(unsigned cpoint_count)/* (3) */

#define p00  (cpoint[k-1].convex==0)
#define p01  (cpoint[k-1].convex==1)
#define p10  (cpoint[k].convex==0)
#define p11  (cpoint[k].convex==1)
#define p20  (cpoint[k+1].convex==0)
#define p21  (cpoint[k+1].convex==1)
#define p30  (cpoint[k+2].convex==0)
#define p31  (cpoint[k+2].convex==1)
/*
#define C1        p11&&p20&&p31  /* Original */
#define C2        p11&&p20&&p30
#define C3   p00&&p10&&p21
#define C4        p11&&p21

#define C1        p11&&p20&&p31
#define C2        p10&&p20&&p31
#define C3   p01&&p10&&p20
#define C4        p11&&p21
*/
#define C1   p01&&p10&&p21
#define C2   p00&&p10&&p21
#define C3        p11&&p20&&p30
#define C4        p11&&p21
#define S1 xs=cpoint[k-1].x;ys=cpoint[k-1].y;xe=cpoint[k+1].x;ye=cpoint[k+1].y
```

```
#define S2xs=cpoint[k].x;ys=cpoint[k].y;xe=cpoint[k+2].x;ye=cpoint[k+2].y
#define MAX_I 63
#define RESHADE(x,y) if(fbv[x][y]==0){dputpel(coeff,x,y);fbv[x][y]=1;}

{
 int s, k, xs, ys, xe, ye, color, bg_color;
unsigned int m, i, coeff;

for(k=0;k<=cpoint_count-1;k++)
{
s=0;        /* Reset status .*/
if(C1)
{ S1; s=1;};     /* Jaggies exist. */
if(C2)
{ S1; s=2;};     /* Jaggies exist. */
if(C3)
{ S2; s=3;};  /* Jaggies exist too but differrent case. */
if((s!=1)&&(s!=2)&&(s!=3))
{ continue;};
/* Jaggies does not exist do not do anything after this. */

if((xe>xs)&&(ye>ys)) /* Case 1. */
{if( (xe-xs)==1 ) /* Case 1a:. */
{m = MAX_I/(ye-ys);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xs,ye);
for(i=ye-1; i>ys; i--)
{coeff = coeff + m;
RESHADE(xs,i);
}
}                /* End case 1a:. */
else
if( (ye-ys)==1 ) /* Case 1b:. */
```

```
{m = MAX_I/(xe-xs);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xs,ye);
for(i=xs+1; i<xe; i++)
{coeff = coeff + m;
RESHADE(i,ye);
}
}                    /* End case Ib:. */
}/* End case I. */

if((xs>xe)&&(ye>ys)) /* Case II. */
{if( (ye-ys)==1 ) /* Case IIa:. */
{m = MAX_I/(xs-xe);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xe,ys);
for(i=xe+1; i<xs; i++)
{coeff = coeff + m;
RESHADE(i,ys);
}
}                    /* End case IIa:. */
else
if( (xs-xe)==1 ) /* Case IIb:. */
{m = MAX_I/(ye-ys);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xe,ys);
for(i=ys+1; i<ye; i++)
{coeff = coeff + m;
RESHADE(xe,i);
}
}                    /* End case IIb:. */
}/* End case II. */
```



```
if((xs>xe)&&(ys>ye)) /* Case III. */
```

```
{if( (ys-ye)==1 ) /* Case IIIa:. */
{m = MAX_I/(xs-xe);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xs,ye);
for(i=xs-1; i>xe; i--)
{coeff = coeff + m;
RESHADE(i,ye);
}
}                /* End case IIIa:. */
else
if( (xs-xe)==1 ) /* Case IIIb:. */
{m = MAX_I/(ys-ye);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xs,ye);
for(i=ye+1; i<ys; i++)
{coeff = coeff + m;
RESHADE(xs,i);
}
}                /* End case IIIb:. */
} /* End case III. */

if((xe>xs)&&(ys>ye)) /* Case IV. */
{if( (xe-xs)==1 ) /* Case IVa:. */
{m = MAX_I/(ys-ye);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xe,ys);
for(i=ys-1; i>ye; i--)
{coeff = coeff + m;
RESHADE(xe,i);
}
}                /* End case IVa:. */
else
if( (ys-ye)==1 ) /* Case IVb:. */
```

```
{m = MAX_I/(xe-xs);
coeff = m >> 1; /* coeff = m/2 ; */
RESHADE(xe,ys);
for(i=xe-1; i>xs; i--)
{coeff = coeff + m;
RESHADE(i,ys);
}
}                    /* End case IVb:. */
}/* End case IV. */
if(s==1)
{ k = k;}
if(s==2)
{ k = k;}
if(s==3)
{ k = k;}
}
}


printout()
{
int x,y;
clrscr();
printf("\ntag\n");
for(y=0;y<16;y++)
for(x=0;x<16;x++) printf(" %3d ",tag[x][y]);;;
getch();
printf("\nfbv\n");
for(y=0;y<16;y++)
for(x=0;x<16;x++) printf(" %3d ",fbv[x][y]);;;
getch();
printf("\nftab\n");
```

```
for(x=0;x<=4/*objnum*/;x++) printf(" %3d ",ftab[x]);;
getch();
printf("\nCtab.color\n");
for(x=0;x<=4/*objnum*/;x++)
{/*if(ftab[x] > -1)*/
printf("ctab[%3d].%3d.%3d.%3d\n",x,ctab[x].x,ctab[x].y,ctab[x].color);}
getch();
printf("\ncpoint\n");
for(x=0;x<=cpoint_count;x++)
printf("cpoint[%0.2d].%0.3d.%0.3d.%0.3d\n",x,cpoint[x].x,cpoint[x].y,cpoint[
getch();
printf("\nobjnum = %d.\n", objnum);
getch();
printf("\ncpoint_count = %d.\n", cpoint_count);
getch();
}
/*=========================================================================*/
/*=============   display_illustrate_image(void) ====================*/
/*=========================================================================*/
int display_illustrate_image(void) /* Fill screen with illustrate image. */
{

set_video_mode(VGA256);
direct_fill_vga256(0x00);/* Clear video screen. */
/*polygon();
getch();
*/
fill_block_slide_vga256();/* Fill the screen with 256 blocks of color. */
getch();
init_gray_scale();/* Load pallete table with weighted gray level
using BIOS. */
getch();
init_RGB();
```

```
    direct_fill_vga256(0x00);/* Clear video screen. */
  /*blue_x();*/
  fill_triag();
  getch();
  /*drawn_blue_bar(); */

  /*
        getch();
  direct_fill_vga256(0x08); /* Clear screen again. */
  getch();
    diago_line();/* Drawn diagonal line from top-left to bottom-right. */
  getch();/* Wait for keyboard hit. */
  */
  }
  /*=============================================================*/
  /*==================  polygon(void);  =========================*/
  /*=============================================================*/
  /* Display small polygon. */
  #define Y_START 10
  #define X_START 10
  #define Y_END   3
  #define X_OFFSET (Y_END - Y_START)
  #define THICK 3
  polygon()
  {
  register unsigned int  x, y,i;

  for(i=0;i<THICK;i++)
  {   x=i+X_START; y=Y_START;
  while(y < Y_END+Y_START){ dputpel(63, x--, y++);}
  }
  }
```

```
int init_RGB(void)
{
#definepalette_index 0x3C8
#define palette_color 0x3C9
unsigned int i;
outportb(palette_index, 0);
for(i=0;i<64;i++)
{outportb(palette_color, 0);/* Read */
outportb(palette_color, 0);/* Green */
outportb(palette_color, i);/* Blue */
}
    for(i=0;i<64;i++)
{outportb(palette_color, 0);/* Read */
outportb(palette_color, i);/* Green */
outportb(palette_color, 0);/* Blue */
}
    for(i=0;i<64;i++)
{outportb(palette_color, i);/* Read */
outportb(palette_color, 0);/* Green */
outportb(palette_color, 0);/* Blue */
}
}
/*drawn_blue_bar()
{
int x,y,i;
for(i=0;i<20;i++)
{line(x+i, y, x+i+100, y+100);
}

}   */
/*=========================================================================*/
/*=========================== blue_x(void); ===============================*/
/*=========================================================================*/
```

```
/* Display big blue X. */
#define Y_START 0
#define Y_END   25
#define X_OFFSET (Y_END - Y_START)
#define THICK 5
#define OFFSET 2
blue_x()
{
register unsigned int  x, y,i;

for(i=0;i<THICK;i++)
{   x=i+OFFSET; y=Y_START+OFFSET;
while(y < Y_END+OFFSET){ dputpel(63, x++, y++);}
}
for(i=0;i<THICK;i++)
{   x=i+X_OFFSET+OFFSET-1; y=Y_START+OFFSET;
while(y < Y_END+OFFSET){ dputpel(63, x--, y++);}
}
}
scanline(start,end,y,color)/* Fill next section    */
{
int i;
for(i=start;i<=end;i++){dputpel(color,i,y);}
}
/*****************************************************************************/
/* Fill a base triangle using scan-line function                             */
/* and then fill a general triangle (using triangle fill routine)            */
/*****************************************************************************/
/* Page 372 */
fill_triag()
        {
        /* Fill a base line triangle                                         */
```

```
int     x0 = 10, x1 = 1, x2 = 30, y0 = 10, y1 = 25, y2 = 25;
int     y, color = 63, start, end;
        for (y = y0; y <= y1; y++)      /* Loop over raster lines      */
                {                       /* Compute intercepts          */
                start = x0 + ((y - y0)*(x1 - x0))/(y1 - y0);
                end   = x0 + ((y - y0)*(x2 - x0))/(y2 - y0);
                scanline(start,end,y,color);    /* Fill next section   */
                }

        /* Fill a general triangle                                      */

        triangle(400,100, 500,0, 630,200, color);
        }

/*****************************************************************************/
/*                                                                           */
/*  triangle(x0,y0,x1,y1,x2,y2,color) - Fill a general triangle with         */
/*           color 'color'.  This routine will call the scanline             */
/*           procedure to fill scanlines between two edges of the            */
/*           triangle.                                                       */
/*                                                                           */
/*           First vertices are ordered so that vertex 'a' is the            */
/*           left most top vertex, and 'b' is the left most bottom           */
/*           vertex.  Triangle is than split into two 'baseline'             */
/*           triangles, and each triangle is filled separately.              */
/*           One triangle is an upward pointing arrow and the second         */
/*           Triangle is a downward pointing arrow.                          */
/*                                                                           */
/*****************************************************************************/

triangle(x0,y0, x1,y1, x2,y2, color)
int     x0,y0, x1,y1, x2,y2,color;
```

```
{
int     x[4],y[4],a,b,c,temp;
x[0] = x0;
x[1] = x1;
x[2] = x2;
y[0] = y0;
y[1] = y1;
y[2] = y2;

/* Order the vertices */
a = 0;                                  /* Get lowest left in a */
if (y[0] > y[1]) a = 1;
if (y[a] > y[2]) a = 2;
b = (a+1)%3;
c = (b+1)%3;
if (y[a] == y[b] && x[a] > x[b]) a = b;
if (y[a] == y[c] && x[a] > x[c]) a = c;
b = (a+1)%3;
c = (b+1)%3;
if (y[b] < y[c])                        /* Get highest left in b*/
        {
        temp = b;
        b = c;
        c = temp;
        }
if (y[b] == y[c] && x[b] > x[c])
        {
        temp = b;
        b = c;
        c = temp;
        }
/* fill 'base' triangles */
if (y[a] == y[b])                       /* Fill degenerates     */
```

```
        {
        scanline(x[a],x[b],y[b],color);
        scanline(x[a],x[c],y[c],color);
        }
    else if (y[a] == y[c])                  /* Fill down arrow      */
        fill_dw(x[a],y[a],x[b],y[b],x[c],y[c],x[b],y[b],y[c],color);
    else if (y[b] == y[c])                  /* Fill up arrow        */
        fill_up(x[a],y[a],x[b],y[b],x[a],y[a],x[c],y[c],y[c],color);
    else
        {                                   /* Split into two bases */
        y[3] = y[c];                        /* one up and one down  */
        x[3] = x[a] + ((y[3] - y[a])*(long int)(x[b] - x[a]))/(y[b] -
        if (x[3] < x[c])                    /* 'c' is to the right  */
          {
          fill_up(x[a],y[a],x[b],y[b],x[a],y[a],x[c],y[c],y[c],color
          fill_dw(x[a],y[a],x[b],y[b],x[c],y[c],x[b],y[b],y[c],color
          }
        else                                /* 'c' is to the left   */
          {
          fill_up(x[a],y[a],x[c],y[c],x[a],y[a],x[b],y[b],y[c],color)
          fill_dw(x[c],y[c],x[b],y[b],x[a],y[a],x[b],y[b],y[c],color)
          }
        }
    }

fill_up(ax,ay, bx,by, cx,cy, dx,dy,y0,color)    /* Fill up arrow        */
int     ax,ay,bx,by,cx,cy,dx,dy,y0,color;
        {
        int     dyab,dycd,y,d1,d2;
        dycd = dy - cy;
        dyab = by - ay;
        for (y = ay; y <= y0; y++)
            {
```

```
                d1 = ((y - ay)*(long)(bx - ax))/dyab;
                d2 = ((y - cy)*(long)(dx - cx))/dycd;
                scanline(ax+d1, cx+d2, y,color);
                }
        }
fill_dw(ax,ay, bx,by, cx,cy, dx,dy,y0,color)    /* Fill down arrow      */
int     ax,ay,bx,by,cx,cy,dx,dy,y0,color;
        {
        int     dyab,dycd,y,d1,d2;
        dycd = dy - cy;
        dyab = by - ay;
        for (y = dy; y >= y0; y--)
                {
                d1 = ((y - ay)*(long)(bx - ax))/dyab;
                d2 = ((y - cy)*(long)(dx - cx))/dycd;
                scanline(ax+d1, cx+d2, y, color);
                }
        }

/*=============================================================*/
/*============   direct_fill_vga256(int color)   ==============*/
/*=============================================================*/
/*
  Subroutine for fill screen in VGA 256 COLORs mode with 'color',
with direct video memory access.
*/
int direct_fill_vga256(int color)
{
register char far *vga256 = (char far *) 0xA0000000;
register unsigned i;
for(i=0; i<64000; i++) { *(vga256++) = (char) color; }
}
```

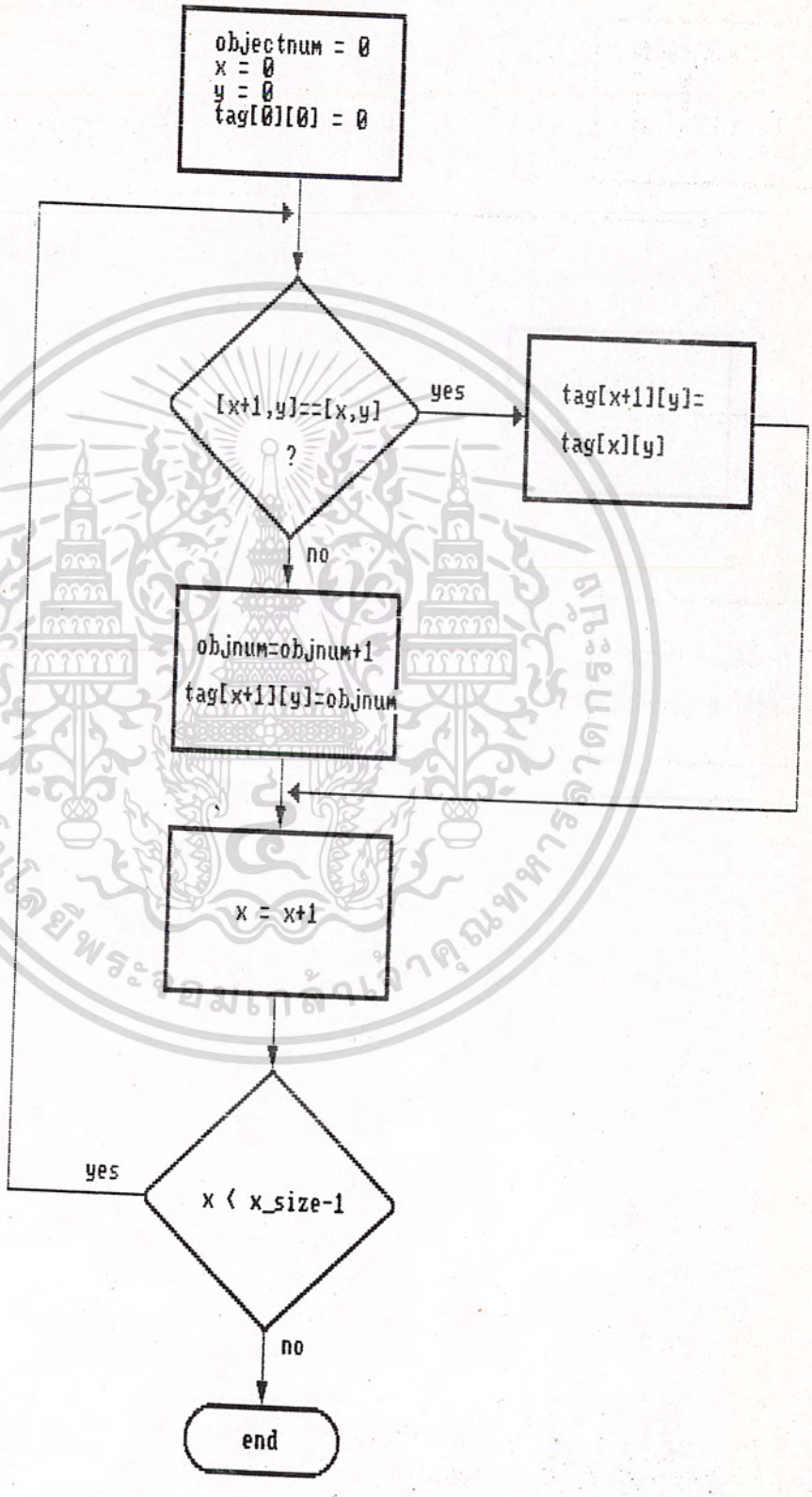
```
/*=============================================================================*/
/*=====================  get_video_mode()  ====================================*/
/*=============================================================================*/
int get_video_mode()
{
union REGS r;
r.h.ah = 0x0f; /* Get current video mode */
return( int86(0x10, &r, &r) ) & 0x00ff;
}
/*=============================================================================*/
/*=================  set_video_mode(int mode)  ================================*/
/*=============================================================================*/
int set_video_mode(int mode)
{
union REGS r;
r.h.ah = 0x00;  /* Set video mode */
r.h.al = (char) mode ;
int86(0x10, &r, &r);
}
/*=============================================================================*/
/*=================putpel(color, x, y)=========================================*/
/*=============================================================================*/
int putpel(int color, int x, int y)
{
union REGS r;

r.h.ah = 0x0C;  /* Write graphics pixel */
r.h.al = (char) color;
r.x.cx = x;
r.x.dx = y;
int86(0x10, &r, &r);
}
```

```
/*=========================================================================*/
/**=======================getpel(int x, int y)=============================*/
/*=========================================================================*/
int getpel(int x, int y)
{
union REGS r;

r.h.ah = 0x0D; /* Read graphics pixel */
r.x.cx = x;
r.x.dx = y;
int86(0x10, &r, &r);
}
/*=========================================================================*/
/*=======================dputpel(color, x, y)==============================*/
/*=========================================================================*/
/* direct  put pixel  in VGA256 colors mode */
int dputpel(int color, int  x, int  y)
{
register char far *vga256 = (char far *) 0xA0000000;

*(vga256+(y<<8)+(y<<6)+x) = (char) color;
}
/*=========================================================================*/
/*=======================dgetpel(int x, int y)=============================*/
/*=========================================================================*/
int dgetpel(int x, int y)
{
register char far *vga256 = (char far *) 0xA0000000;
int i;
i = *(vga256+(y<<8)+(y<<6)+x) ;
return((char) i);
}
```

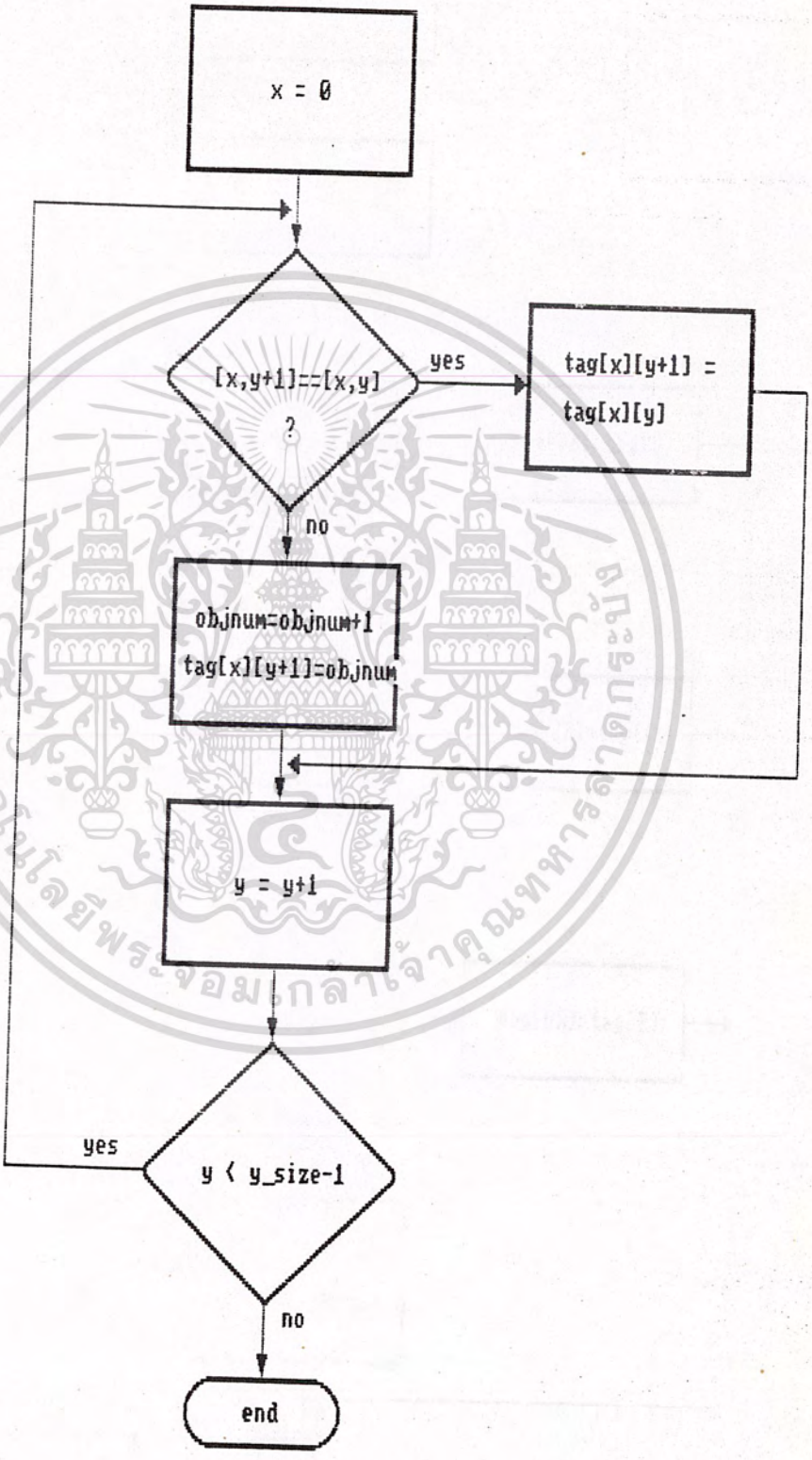
```
/*=============================================================================*/
/*======================== init_gray_scale(void) ==============================*/
/*=============================================================================*/
/*
  Fill palette with gray scale weighting.
*/
int init_gray_scale(void)
{
union REGS r;

r.h.ah = 0x10;
r.h.al = 0x1B;
r.x.bx = 0x0000;
r.x.cx = 0x0100;
int86(0x10, &r, &r);
}

/*=============================================================================*/
/*================ fill_block_slide_vga256()==================================*/
/*=============================================================================*/
/*
  Fill VGA 256 COLORs mode screen with partial block of color
which have value from 0 (0x00) to 255 (0xFF), from left to right &
top to bottom.
*/
int fill_block_slide_vga256()
{
 int  ii, xx, yy;
register int  i=0, x, y;

for(ii=0; ii<=255; )
{   y=0;
for(yy=12; yy<=192; yy = yy+12)
```

```
{for( ; y<yy; y++)
{   i = ii;
x=0;
for(xx = 20; xx <= 320; xx = xx+20)
{
  for( ; x < xx; x++) { dputpel( i, x, y); }
  i++;
}
}
ii = ii+16;
}
}
}
/*=============================================================*/
```

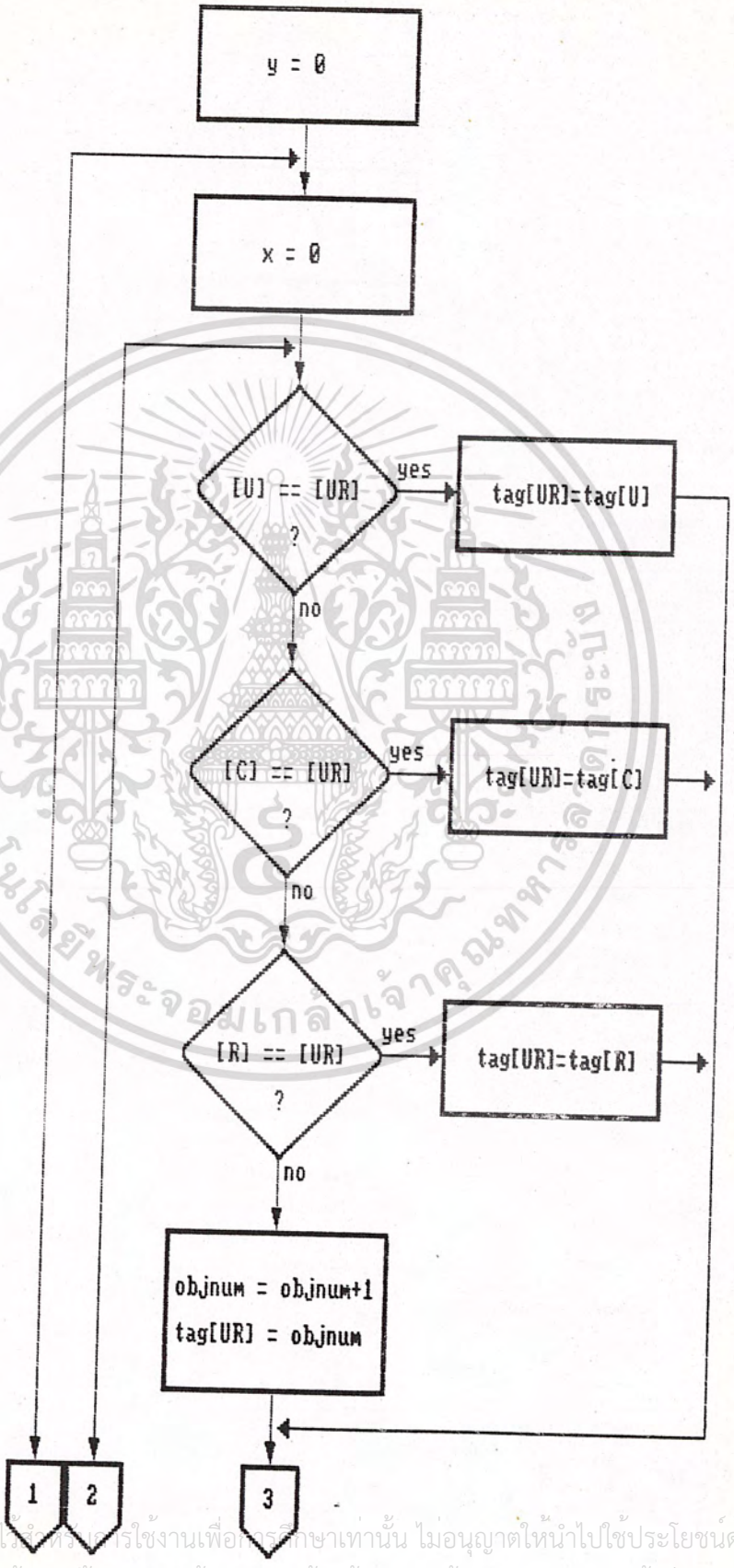
FIRST ROW FILL ALGORITHM

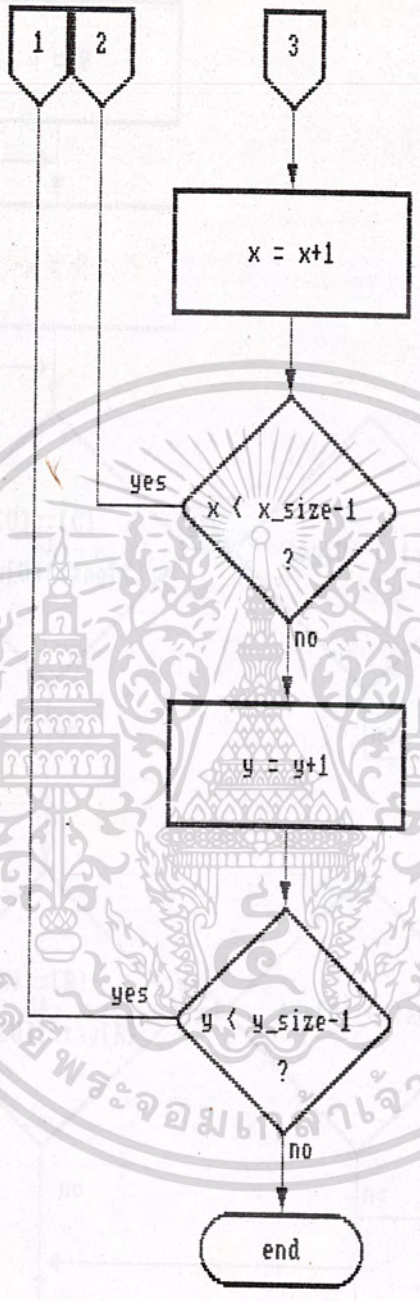


FIRST COLUMN FILL ALGORITHM

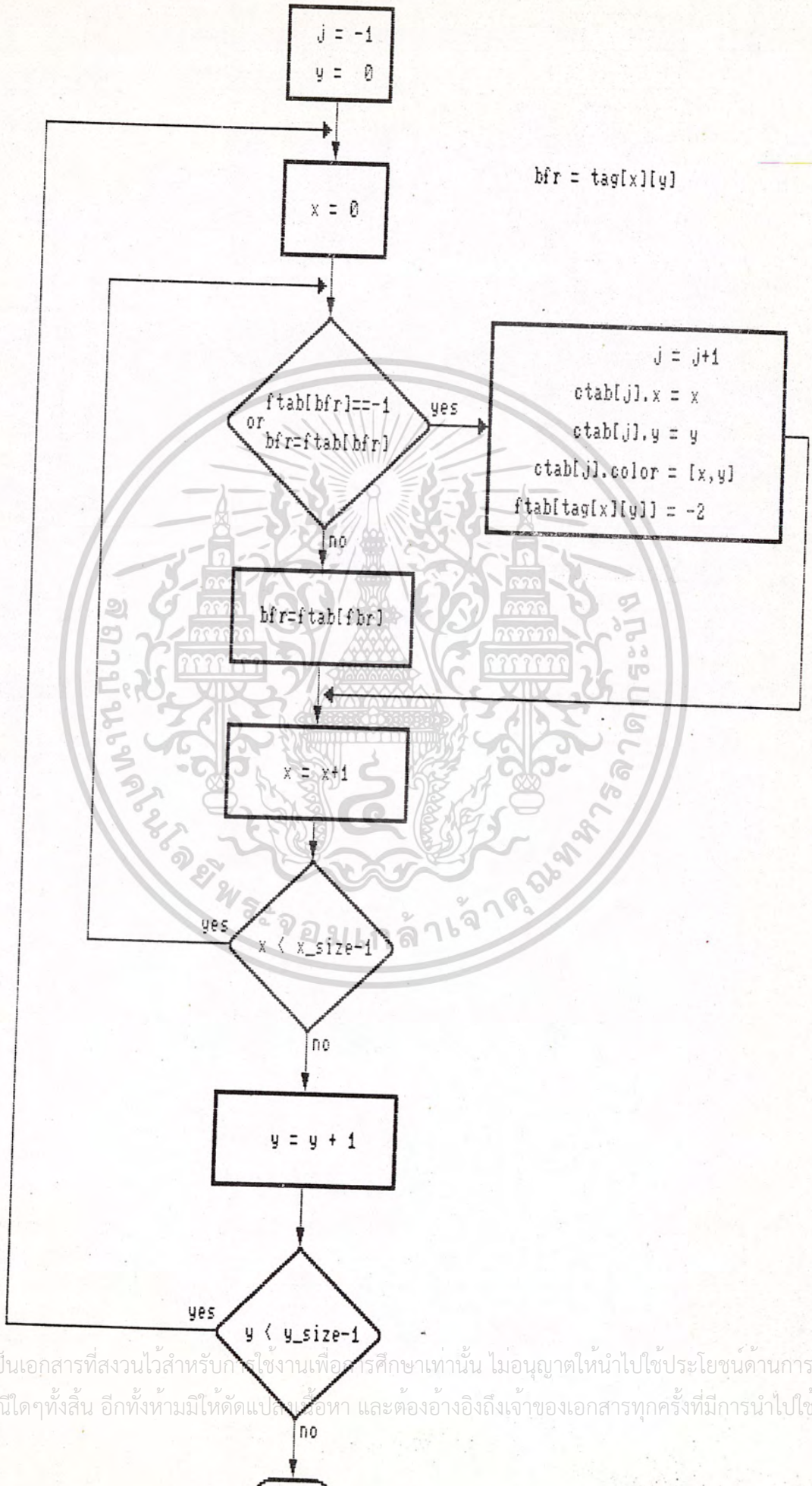


REST OF THE TAG FILLING ALGORITHM



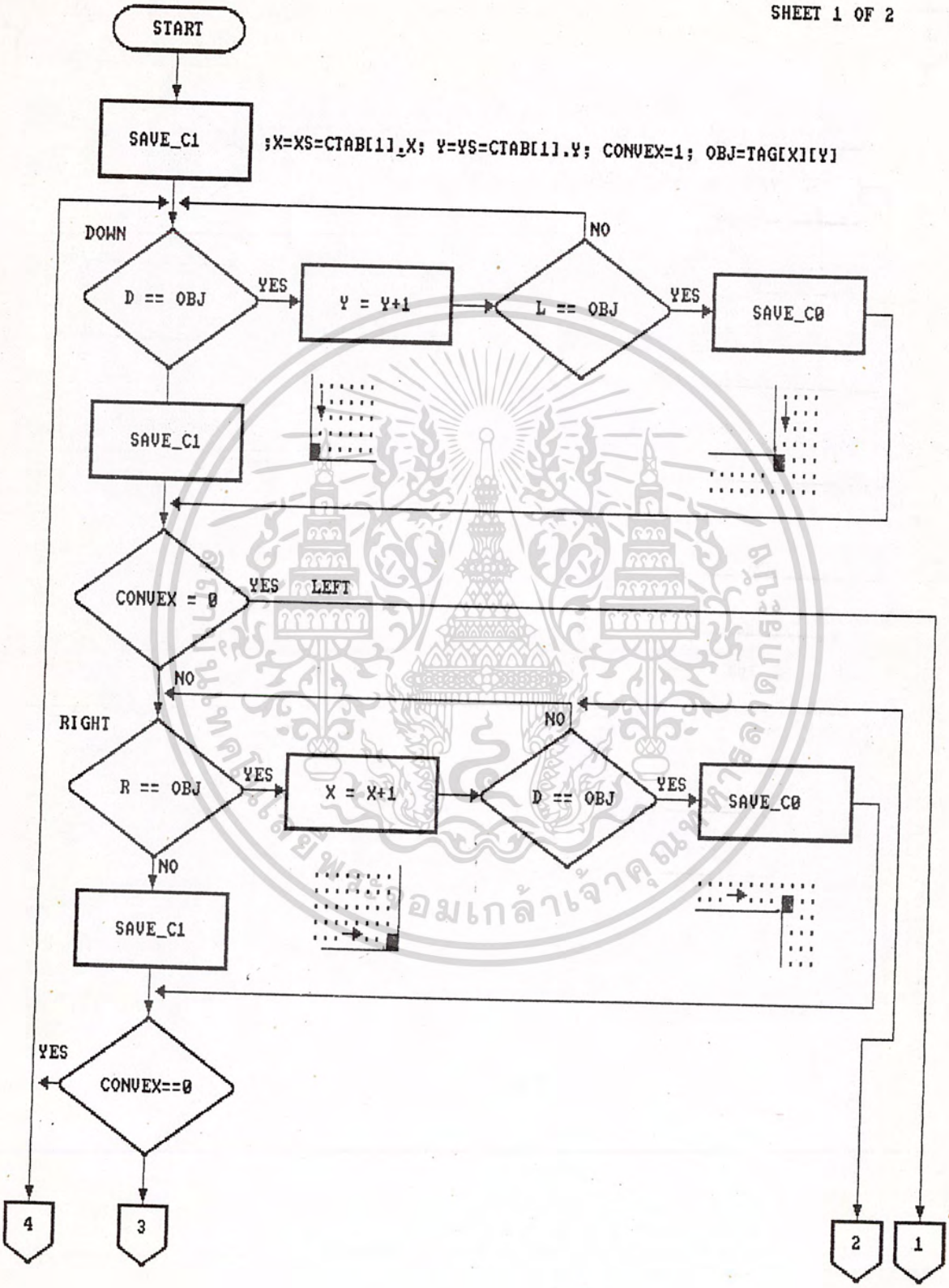


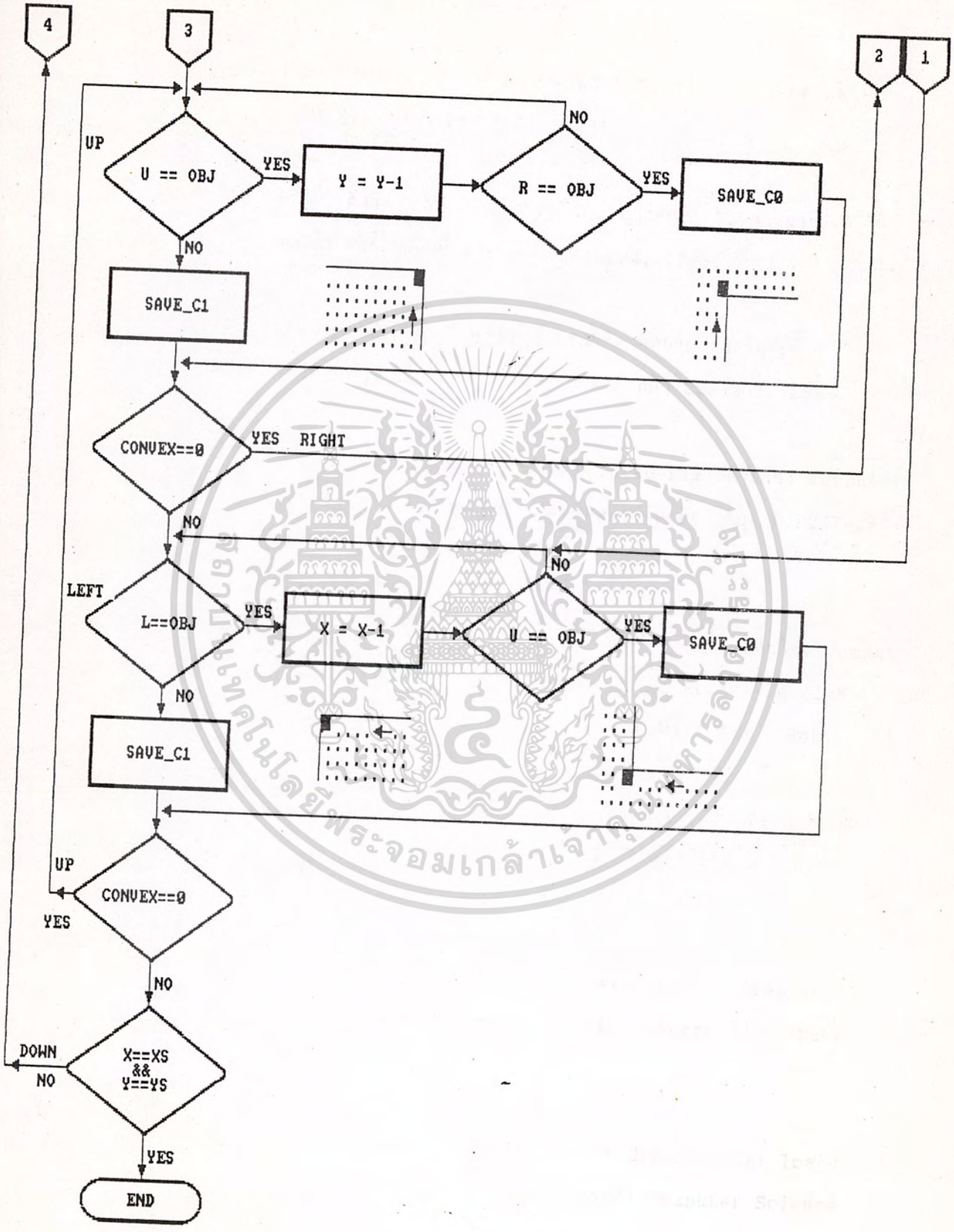
section 1 pass 2



EDGE TRACKING COUNTER CLOCK WISE

SHEET 1 OF 2





หนังสืออ้างอิง

1. กองบรรณาธิการ, " กราฟิกเทคโนโลยี ", วารสารไมโครคอมพิวเตอร์, ฉบับที่ 43, หน้า 182-195, 2531

2. ถวัลย์ ตั้งลิตานนท์, " เปิดโฉม VGA ฮาร์ดแวร์ ", วารสารไมโครคอมพิวเตอร์, ฉบับที่ 55, หน้า 284-294, 2533.

3. ชจร เลิศสุดวิชัย, " การโปรแกรมใช้งานการ์ด VGA ตอนที่ 1 ", วารสารไมโครคอมพิวเตอร์, ฉบับที่ 55, หน้า 295-305, 2533.

4. Alan Watt, " Fundamental of Three-Dimlnsional Computer Graphics ", Addison-Wesley Publishers Ltd, P.P267-284 1989.

5. Perter Burger and Duncan Gillies, " Interactive Computer Graphics Functional, Procedural and Eevice-Level Methods" Addison-Wesley Publishers Ltd., P.P111-143, 1989

6. Rod Salmon and Mel slater, " Cpmputer Graphics Systems Concepts ", Addison-Westey Publishers Ltd., P.P380, 420,426,428, 1989.

7. Levine, stephen R.," Interactive Computer Graphics " Course 365, Santa Monica, Carif.:Integrated Computer System, 1980.

8. Pavlidis, theo, " Algorithms for Graphics anf Image Processing ", Rockville, Maryland: Computer Science Press, 1982.

9. Rogers, David F.," Procedural Elements For Computer Graphics ", New York: McGraw-Hill Book., 1985.

10. Gupta, Satish, and sproull,Robert F.," Filtering Edges for Gray-Scale Displays ", Computer Graphics 15(3), August 1981,pp 1- 5.

11. Bloomenthal, Jules, " Edge Inferance with Apprications to Antialiasing ", Computer Graphics 17(3), July 1983, pp 157 - 162.

12. Crow, Franklin c.," A Comparision of Antialiasing Techniques ", IEEE Computer Graphics and Applications, Vol. 1, 1978.

13. George Sutty & Stew Blair, "Programmer's Guide to the EGA/VGA", Brody New York, 1988

14. Benezzell, "Graphics Programming in Turbo C 2.0", Knud Gellert, 1988

15. David F. Rogers, "Procedural elements for Computer Graphics", McGRAW HILL, P.P 93-108, 1986