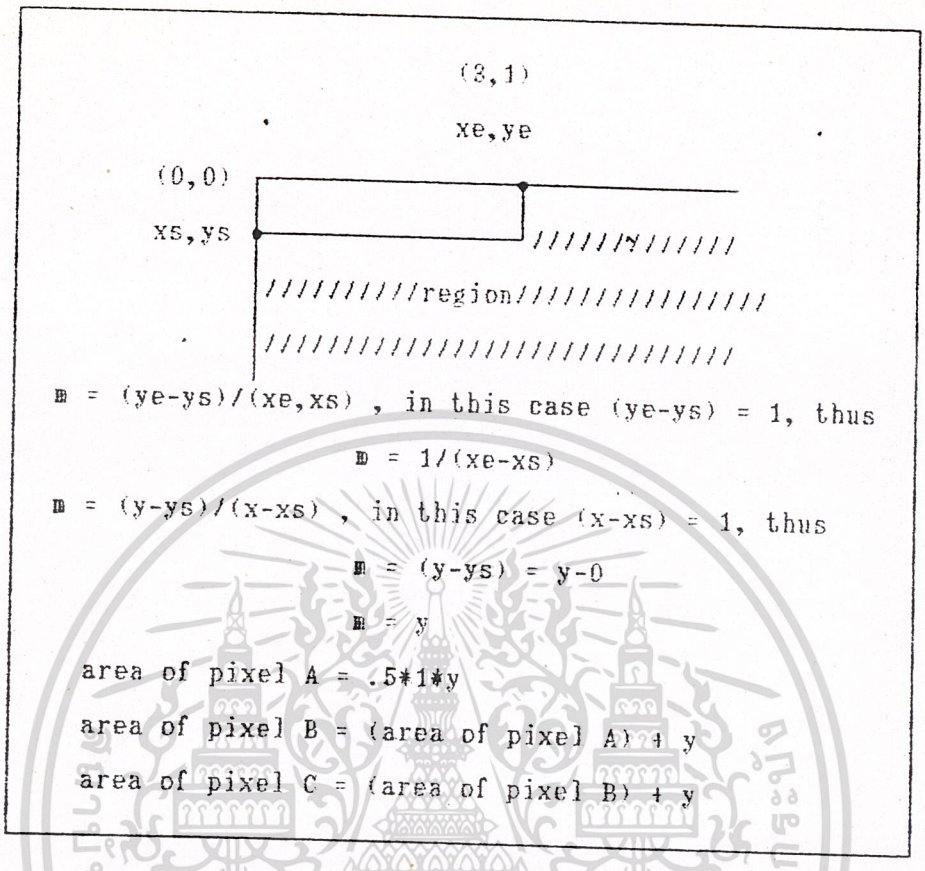


รูปที่ 22 REMOVING A JAGGIE BY RESHADING PIXELS

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 : วิธีการ 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 และการจัดลำดับสีสำหรับ 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 บนขอบนอกของ 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

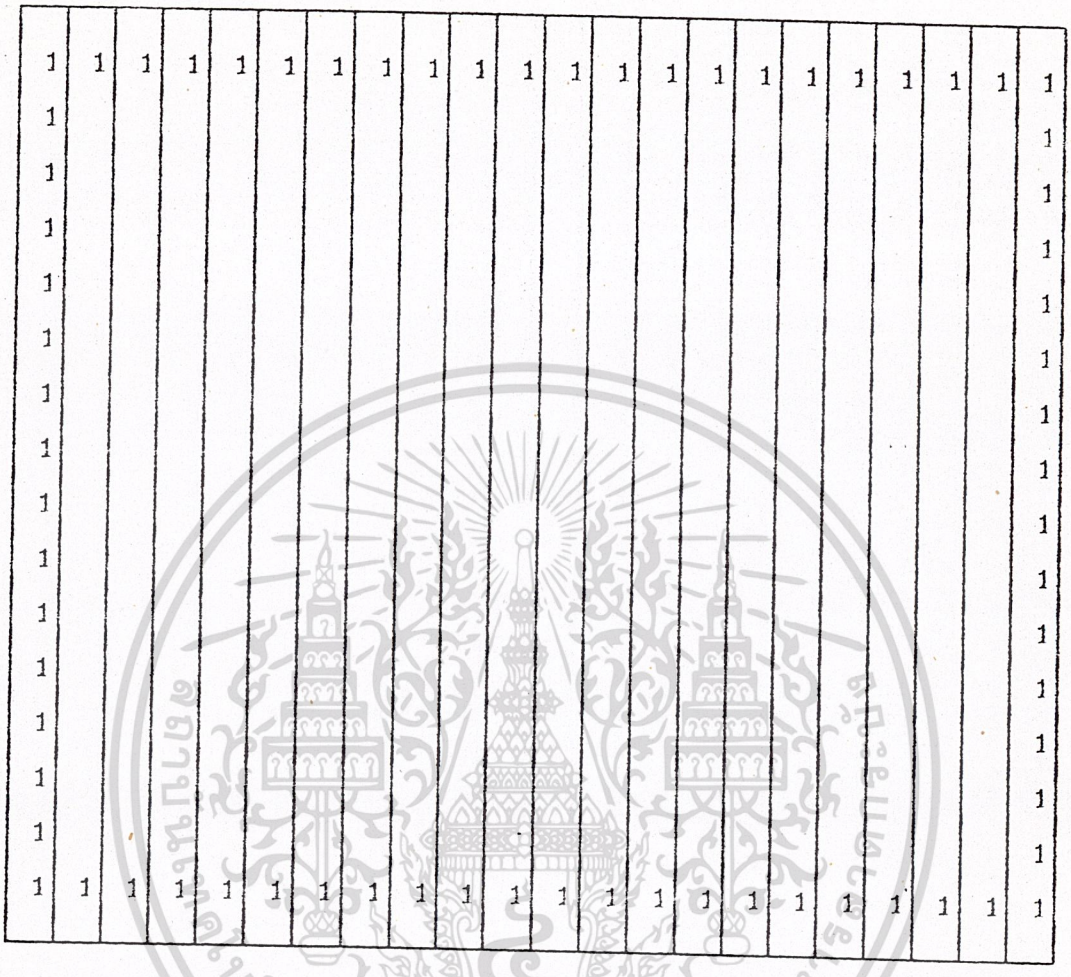
ต่อไปนี้การอธิบายการใช้ "fbv buffer" เราจะพิจารณาการจัดลำดับสีของภาพตามลำดับ

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

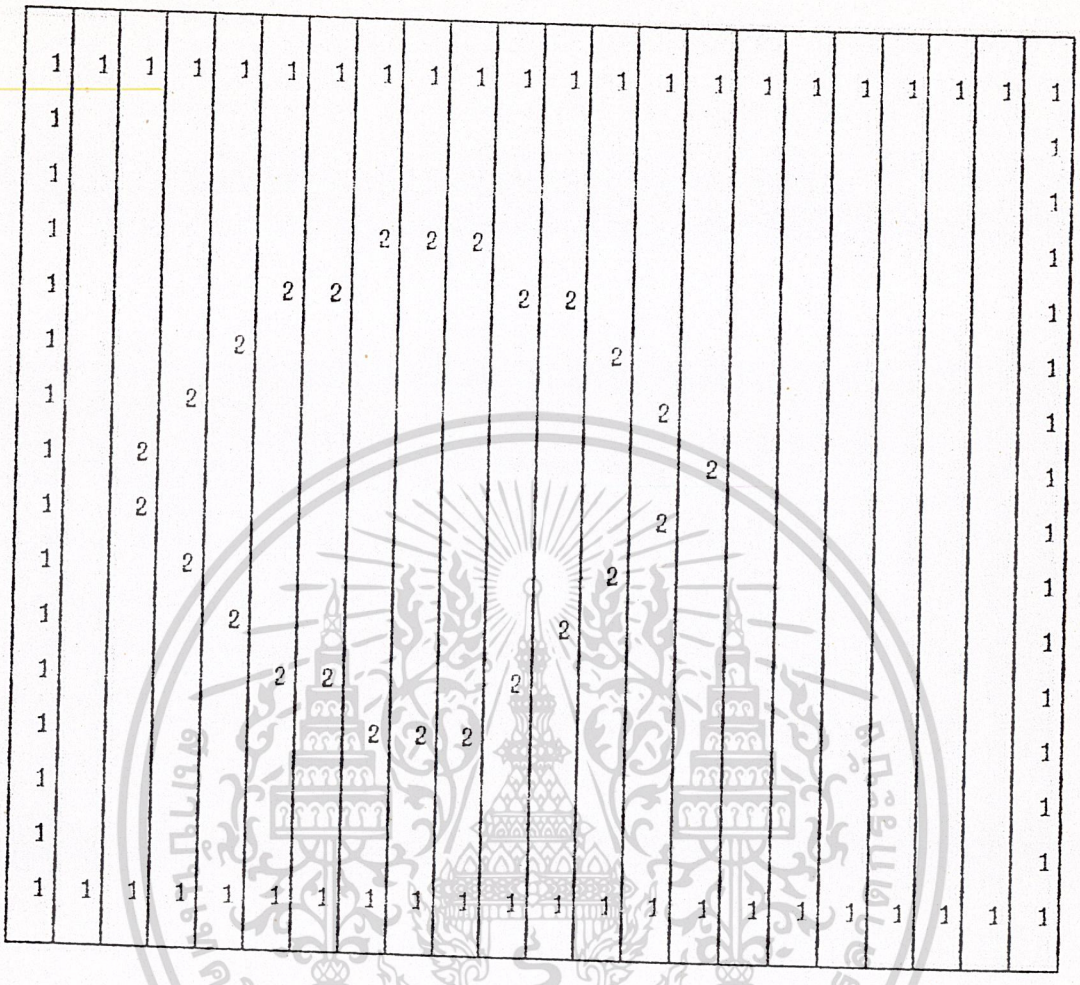
ในรูปที่ 23 จะเห็นได้ว่ามี tag buffer image 2 object คือวงกลมสีแดงและวงกลมสีเหลืองทับกันอยู่ และมี background สีน้ำเงิน ซึ่งจะมีอยู่จริง 3 objectในภาพ ดังนั้น back ground ของภาพ จะพิจารณาที่ blue pixels ก็เป็นวัตถุด้วย ก่อนที่จะเริ่มต้นการจัดลำดับสี "fbv buffer" ให้มีขนาดเท่ากับ tag bufferโดยกำหนดค่าเริ่มแรกเท่ากับ 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



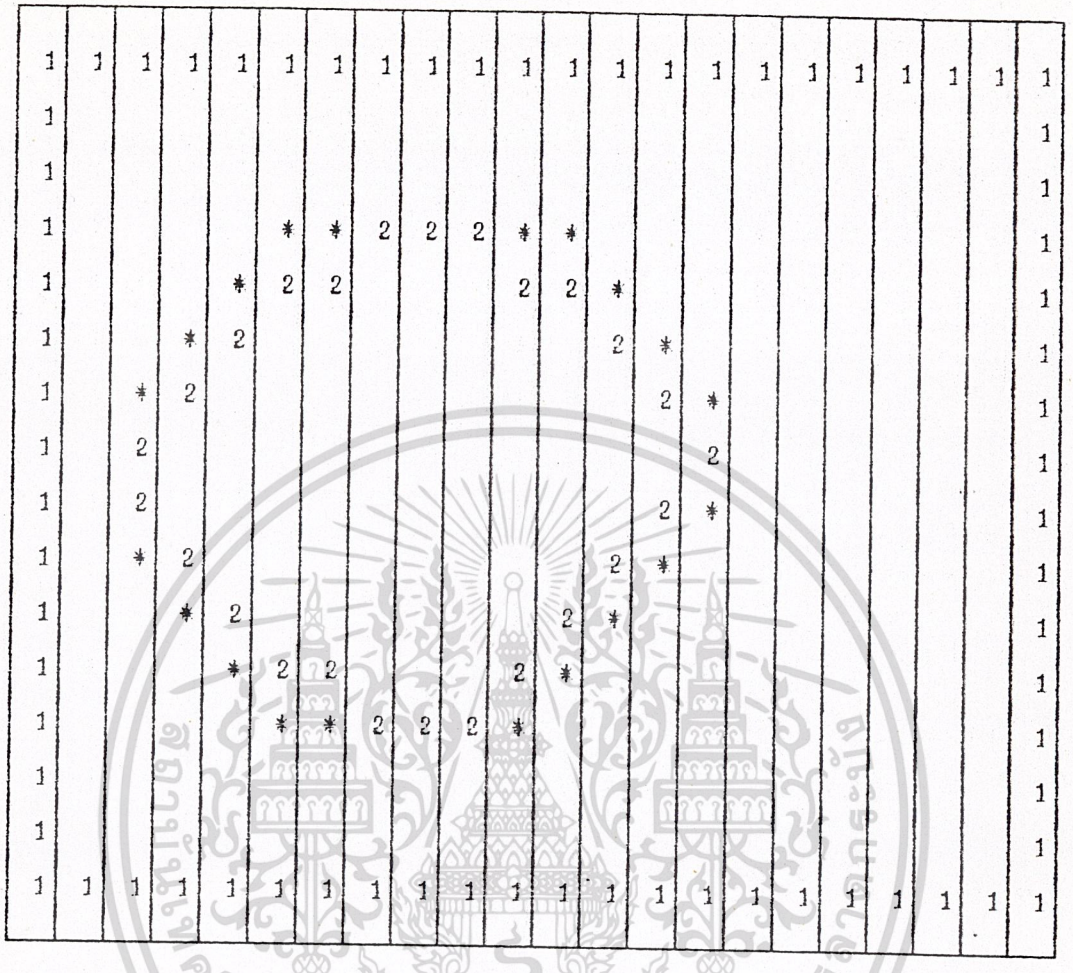
รูปที่ 5.2 ส่วนประกอบของ FBV BUFFER ของขอบ pixel ของ object 1

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



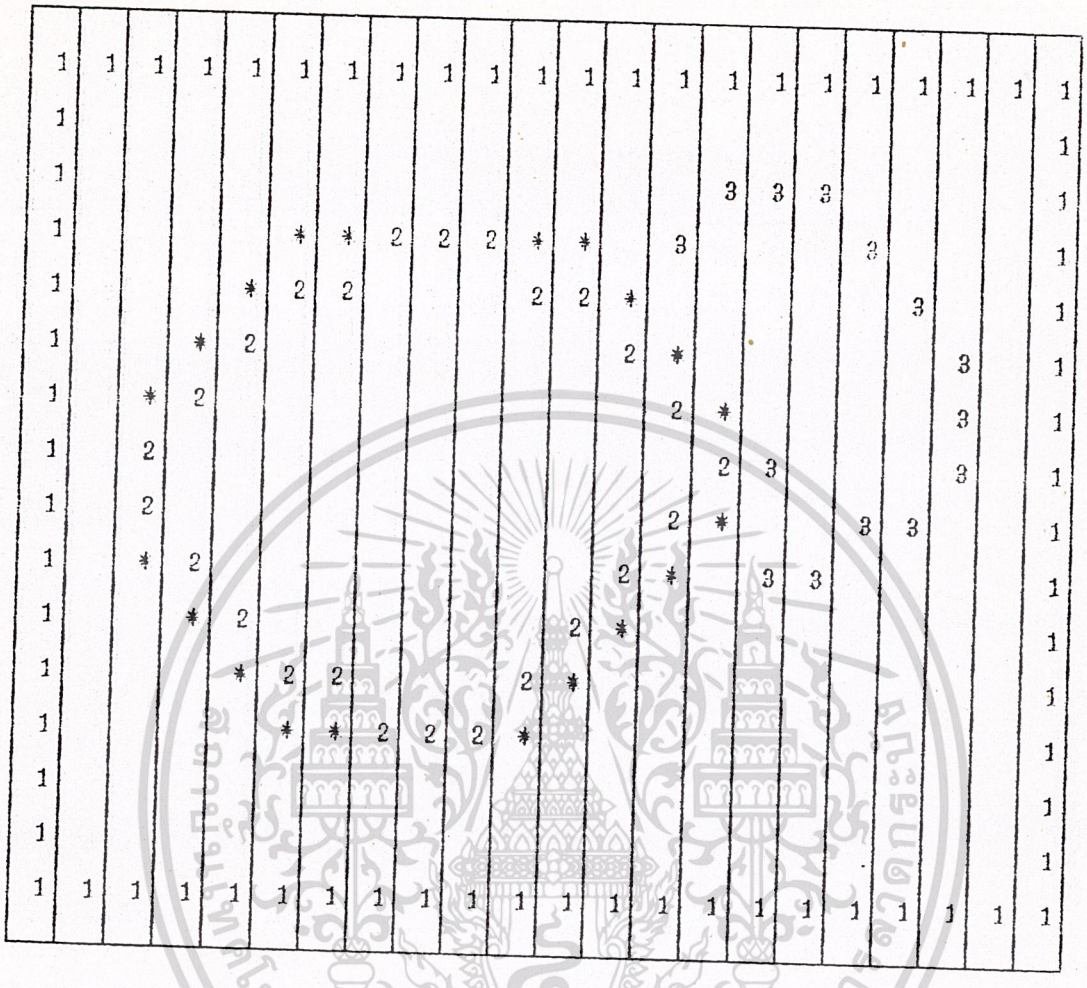
รูปที่ 5.3 ส่วนประกอบของ FBV BUFFER หลังจาก marked ขอบ pixel ของ object 2

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



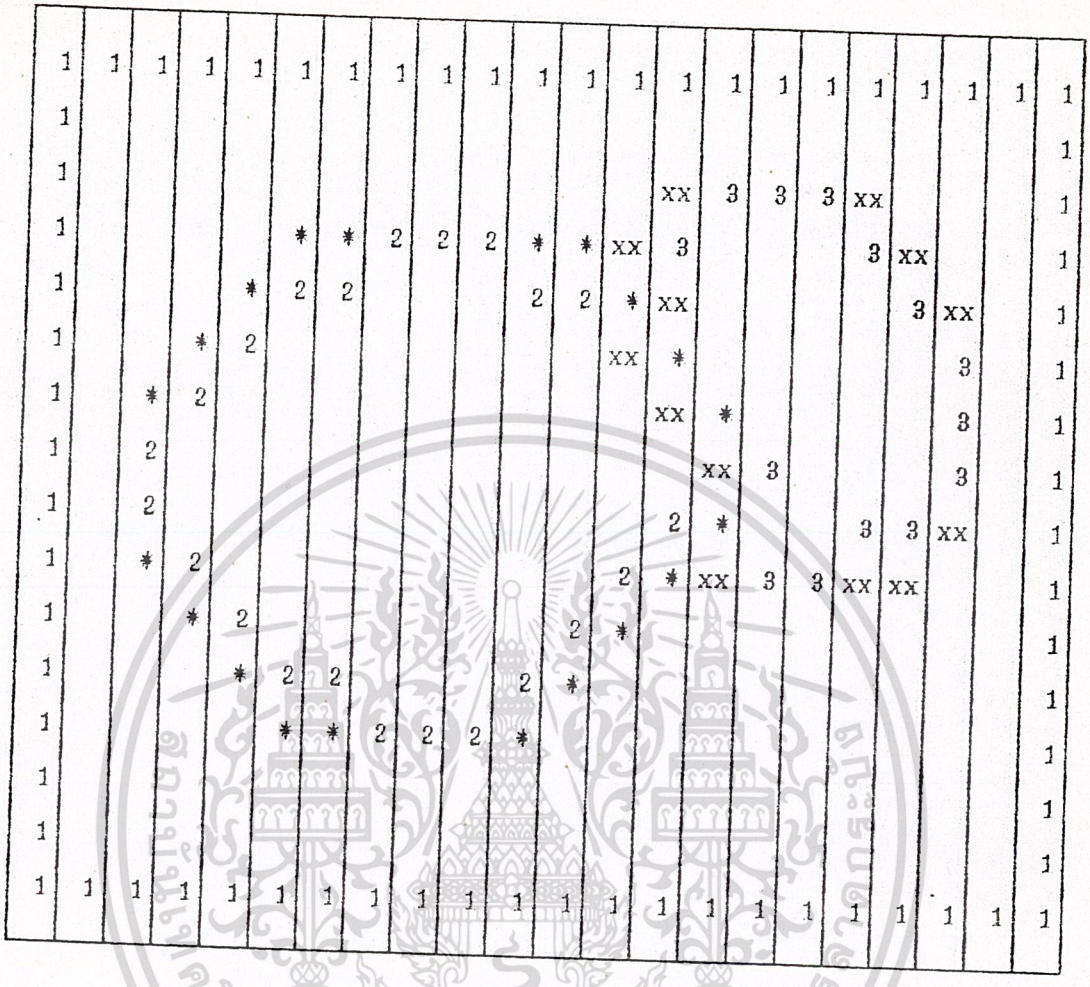
รูปที่ 5.4 ส่วนประกอบของ FBV BUFFER หลังจาก รอยหยักของ object 2 หลังจากผ่านการ smooth แล้ว

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



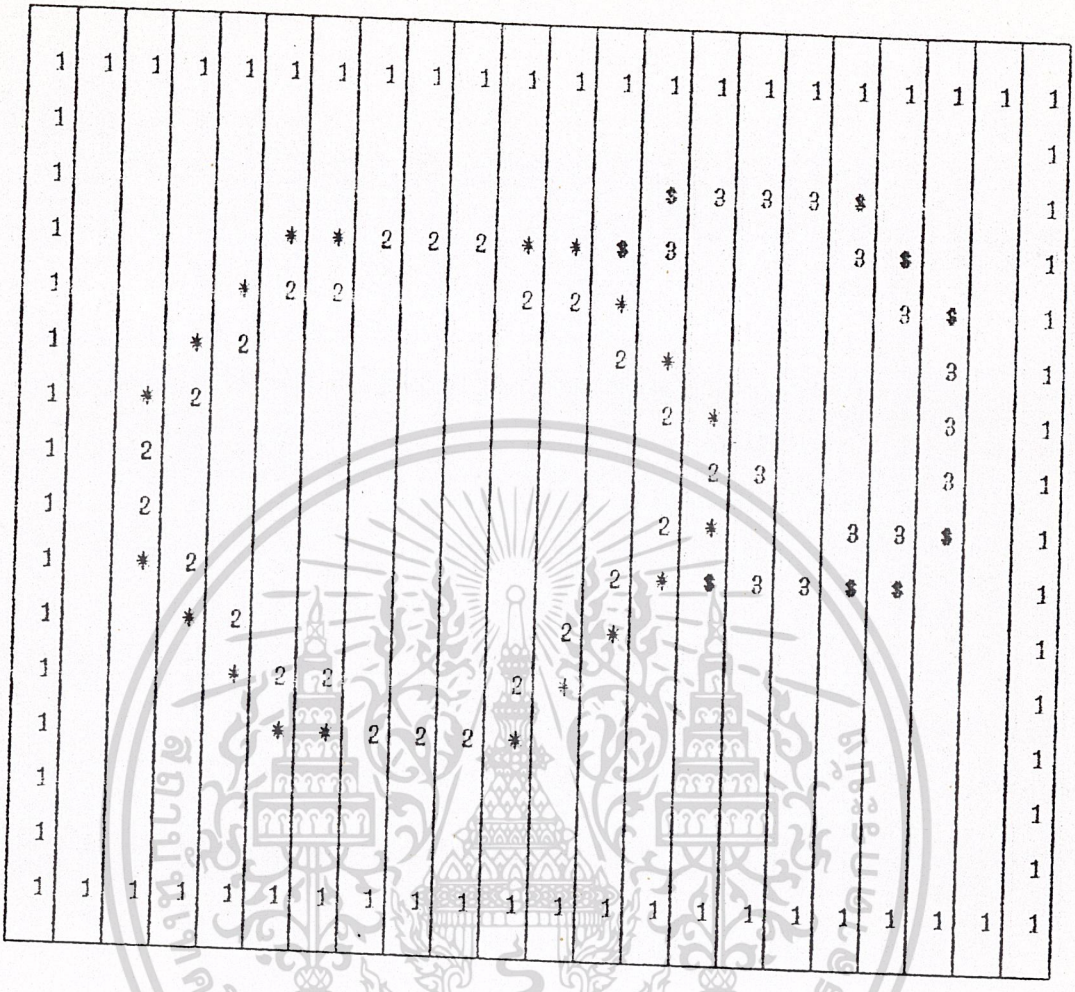
รูปที่ 5.5 ส่วนประกอบของ FBV BUFFER หลังจาก marked pixel ของขอบ object 3

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 นี่คือการ clear ให้สม่ำเสมอใน 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 i
    Call the smoothing procedure to smooth the outside edge
    of an object i
    END
```

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

จะมันทึกค่าทั้งหมดใน 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*1024L)/(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[x
    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     dycb,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