Fingerprint Recognition

โดย

นายธานินทร์ พินทอง

นายศรนรินทร์ ตพรัตน์

รายงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปีการศึกษา 2539

Fingerprint Recognition

โดย

นายธานินทร์ พินทอง

นายศรนรินทร์ ตพรัตน์

อาจารย์ที่ปรึกษา

รศ.ดร.ครรชิต ไมตรี

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง Fingerprint Recognition

ผู้จัดทำ

1. นายธานินทร์ พินทอง 37013296
2. นายศรนรินทร์ ตพรัตน์ 37013313

.....อาจารย์ที่ปรึกษา

(รศ.ดร.ครรชิต ไมตรี)

Fingerprint Recognition

นายธานินทร์ พินทอง
นายศรนรินทร์ ตพรัตน์
รศ.ดร.กรรชิต ไมตรี อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ในวิทยานิพนธ์ฉบับนี้เรียบเรียงขึ้นจากผลงานที่ได้ศึกษาและวิเคราะห์วิธีการ Preprocessing เริ่มจากการเก็บภาพลายนิ้วมือที่พิมพ์ด้วยหมึกจากเครื่องสแกนเนอร์แทนกล้องวีดีโอ ภาพที่ได้จากเครื่องสแกนเนอร์ถูกกำหนดให้มี 256 ระดับเทาเพื่อให้เหมือนกับที่ได้มาจากกล้องวีดีโอ หลังจากนั้นก็ทำการ Smooth กับภาพที่ได้เพื่อขจัดนอยส์ ภาพที่ผ่านการทำ Smooth แล้วก็ถูกนำมาหาค่าเทรสโฮลด์อัตโนมัติ โดยใช้สมการเอนโทรปีซึ่งสมการเอนโทรปีดังกล่าวมีการผสมผสานการแจกแจงปัวซอง เพื่อให้ค่าเทรสโฮลด์มีความถูกต้องมากขึ้น ค่าเทรสโฮลด์ที่ได้ดังกล่าวจะถูกนำมาพิจารณาทำไบนาไรซ์กับภาพ ผลที่ได้จะเป็นภาพไบนารี ภาพไบนารีนี้บางครั้งอาจต้องทำ Smooth ในลักษณะของภาพไบนารี และสุดท้ายของขั้นตอน Preprocessing ก็จะเป็นการทำ Thinning กับภาพไบนารี ภาพที่ได้จะเป็นภาพโครงร่าง (Skeleton or Median axis) เพื่อทำการ Postprocess ต่อไป

Fingerprint Recognition

THANIN PINTONG
SORNNARIN TOPPARAT
KANCHIT MAITREE ADVISOR
1996

Abstract

This thesis present a preprocessing method with fingerprint image in principle. Firstly, an acquisition of inky fingerprint image is from scanner instead of video camera. An image of scanner has 256 - grey levels like that of video camera. After that, the fingerprint image is smoothed for reducing noise. The smoothed image is then used to evaluate one threshold value as an automatic value by using entropy equation together with poisons distribution for more correct threshold value. Then binarizing image is determined by using the previous threshold value. A result of it provides binary image by using binary smoothing method. Finally, thinning image is used for obtaining minimal skeletons and for the following postprocessing methods.

สารบัญ

สารบัญภาพ

สารบัญตาราง

ตารางที่ หน้า

สารบัญสัญลักษณ์

Z	เซตของจำนวนเต็ม
Z^2	กริด 2 มิติของจุดต่าง ๆในแบบ ดิสครีต
A,B,C,X,Y, ...	สับเซตของ Z^2
$\varnothing$	เซตว่าง
$\Psi(X)$	การทรานฟอร์มเมชั่นของ X ด้วย Ψ
a,b,c,x,y, ...	สมาชิกใน Z^2 หรือ จุด ๆหนึ่งในกริด 2 มิติ
$x \in X$	x เป็นสมาชิกของ X
$x \notin X$	x ไม่เป็นสมาชิกของ X
$X \subset Y$	X เป็นสับเซตแท้ของ Y
$X \subseteq Y$	X ไม่เป็นสับเซตแท้ของ Y
$X \supset Y$	X เป็นซูปเปอร์เซตแท้ของ Y
$X \supseteq Y$	X ไม่เป็นซูปเปอร์เซตแท้ของ Y
$X \cap Y$	การอินเตอร์เซคชั่นของเซต
$X \cup Y$	การยูเนียนกันของเซต
$X \ominus Y$	การดำเนินการลบแบบ Minkowski ของ X ด้วย Y
$X \oplus Y$	การดำเนินการบวกแบบ Minkowski ของ X ด้วย Y
$X \ominus \breve{Y}$	อีโรชั่นของ X ด้วย Y
$X \oplus \breve{Y}$	ดิเลชั่นของ X ด้วย Y
$X \circledast Y$	hit/miss ทรานฟอร์มของ X ด้วย Y
$X \circ Y$	การดำเนินการ thinning ของ X ด้วย Y
X / Y	ผลต่างของเซตระหว่าง X และ Y
$\breve{X}$	การรีเฟลคชั่นเซต X ณ จุดกำเนิด หรือ เซตของจุดต่าง ๆที่ $-x \in X$
X^c	คอมพลีเมนต์ของ X ใน Z^2
$\{x : P\}$	เซตของจุดต่าง ๆ x ที่สอดคล้องกับเงื่อนไข P
X_b	การทรานสเลชั่นของเซต X ด้วยเวคเตอร์ b หรือ $\{x : x - b \in X\}$
$I_{b \in B} X_b$	การอินเตอร์เซคชั่นกันของการทรานสเลชั่น X_b ซึ่งมี $b \in B$
$Y_{b \in B} X_b$	การยูเนียนกันของการทรานสเลชั่น X_b ซึ่งมี $b \in B$

บทที่ 1

บทนำ

ในการจำแนกบุคคลว่าเราเป็นใครนั้น ปัจจุบันใช้ ภาพถ่าย รหัสลับประจำตัว บัตรประชาชน หรือกระทั่งบัตรแม่เหล็กรูปแบบต่างๆ กำลังจะถูกแทนที่ด้วย ระบบพิสูจน์ลักษณะจำเพาะของอวัยวะอย่างใดอย่างหนึ่งของตัวเรา ระบบรักษาความปลอดภัยนี้เรียกว่า " Biometrics Identification " ลักษณะการทำงานของระบบจะบันทึกลักษณะทางกายภาพของอวัยวะส่วนใดส่วนหนึ่งของร่างกายเอาไว้ในรูปแบบของรหัสดิจิตอล เก็บไว้ในฐานข้อมูลกลางของระบบ เมื่อมีการเข้าออกหรือกระทำอย่างใดกับสิ่งที่ป้องกันไว้ จะต้องผ่านการตรวจสอบกับข้อมูลกลางเพื่อที่จะตัดสินใจในการอนุญาติหรือไม่รู้จักกับบุคคลนั้น

ขณะนี้ระบบที่นำมาจำแนกบุคคลได้มีอยู่ 7 ระบบ คือ ระบบใช้เรตินา ระบบใช้ม่านตา ระบบใช้ภาพถ่ายใบหน้าแบบสามมิติ ระบบใช้ลายพิมพ์มือ ระบบใช้เสียง ระบบตรวจสอบด้วยลายมือชื่อ และ ระบบใช้ลายนิ้วมือ จะเห็นว่าทุกระบบใช้หลักการหาจุดเด่นหรือลักษณะเฉพาะของบุคคลในการเปรียบเทียบ แต่ไม่ว่าระบบใดก็ตามยังมีของจำกัดของระบบอยู่ทุกระบบ เนื่องจากการบาดเจ็บ การเปลี่ยนแปลงของอายุ แม้กระทั่งการเจ็บป่วย อาจทำให้ระบบเกิดการผิดพลาดได้ จากระบบที่มีอยู่การใช้ลายนิ้วมือน่าจะเป็นระบบที่สามารถรักษาความปลอดภัยได้ดีกว่าระบบอื่นๆ ปัจจุบันบันระบบใช้ลายนิ้วมือมีการตรวจสอบอยู่ 2 แบบ คือ

Fingerprint Identification เป็นระบบที่ทำการเปรียบเทียบลายนิ้วมือที่ป้อน เข้ามากับฐานข้อมูลทั้งหมด และหาลายนิ้วมือที่มีลักษณะใกล้เคียงกันที่สุดหรือตรงกัน ลักษณะการทำงานของระบบจะเป็น one - to many มักใช้ในกิจการกองพิสูจน์หลักฐานของกรมตำรวจ

Fingerprint Verification เป็นระบบที่ทำการเปรียบเทียบลายนิ้วมือที่ป้อน เข้ามากับฐานข้อมูลทั้งหมด และหาลายนิ้วมือที่มีลักษณะตรงกัน ลักษณะการทำงานของระบบจะเป็น real time มักใช้ในกิจการรักษาความปลอดภัยต่างๆ

เนื่องจากระบบเหล่านี้ที่ผลิตจากต่างประเทศมีราคาสูง เช่น ระบบของญี่ปุ่น AFVS - NEC, AFVS - HITACHI ระบบของอเมริกา PRINTRAX PIV100, RIDGE READER, IDX - 40 เป็นต้น หากประเทศไทยสามารถผลิตเทคโนโลยีเหล่านี้ได้ก็จะเป็นการดีต่อการพัฒนาเทคโนโลยีและลดต้นทุนในการซื้อเทคโนโลยีต่างๆได้

บทที่ 2

ความรู้เบื้องต้นของลายนิ้วมือ

2.1 ลักษณะลายนิ้วมือ

ผิวหนังบริเวณปลายนิ้วมือ ประกอบด้วยลายเส้นสองชนิด ชนิดหนึ่งเราเรียกว่า เส้นนูน (Ridges) อีกชนิดหนึ่งเรียกว่า รอยร่อง หรือ เส้นร่อง (Furrows) จะอยู่สลับกันตลอดไป โดยถ้าเราใช้หมึกสีดำทาบนนิ้วมือ และกดนิ้วมือลงบนกระดาษขาวจะได้ลายเส้นสีดำและสีขาวสลับกัน เราเรียกเส้นสีดำว่า เส้นนูน เราเรียกเส้นสีขาวว่า เส้นร่อง เราสามารถแยกลักษณะของลายนิ้วมือที่เป็นจุดสำคัญต่างๆ ดังนี้

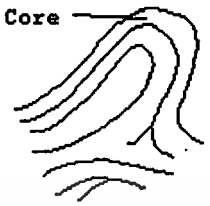
2.1.1. เส้นขอบ (Type Line) หมายถึง เส้นคู่ขนานคู่ในสุดซึ่งได้เดินขนานกันมาพอสมควรแล้วแยกตัวออกจากกัน เพื่อโอบล้อมหรือพยายามโอบล้อมบริเวณลายนิ้วมือที่อยู่ภายในและเส้นขอบไม่จำเป็นจะต้องเป็นเส้นยาวและราบเรียบติดต่อกันตลอด อาจจะเป็นเส้นที่ขาดกลางเส้น แต่เราก็ยังจัดเป็นเส้นขอบด้วย

รูปที่ 2.1 แสดงลักษณะเส้นขอบ

2.1.2. เส้นดอน (Delta) หมายถึง ลายเส้นในนิ้วมือ ซึ่งอยู่ตรงหน้าและใกล้ที่สุดกับกึ่งกลางของปากทางแยกเส้นขอบ

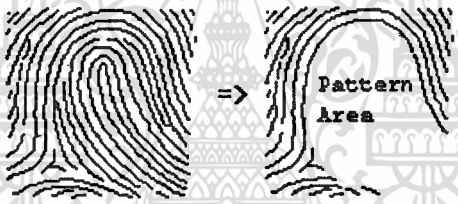
รูปที่ 2.2 แสดงลักษณะเส้นดอน

2.1.3. จุดใจกลาง (Core) หมายถึง จุดใดจุดหนึ่งบนปลายเส้น หรือ ไหล่ของเส้นวกกลับ ที่อยู่บนสุด



รูปที่ 2.3 แสดงลักษณะจุดใจกลาง

2.1.4. บริเวณลายนิ้วมือที่อยู่ภายใน (Pattern Area) หมายถึง พื้นที่บริเวณภายในของลายนิ้วมือที่ถูกเส้นขอบโอบล้อมไว้



รูปที่ 2.4 แสดงลักษณะบริเวณลายนิ้วมือที่อยู่ภายใน

2.2 ชนิดและรูปแบบลายนิ้วมือ

เราสามารถจัดรูปแบบลายนิ้วมือตามลักษณะต (Core) ออกเป็น 4 กลุ่ม คือ

2.2.1 กลุ่มเส้นโค้ง (Arch)

2.2.1.1 โค้งราบ (Plain Arch = PA) ลายเส้นวิ่งหรือไหลออกไปข้างหนึ่ง ไม่เกิดมุมแหลมหรือเส้นพุ่งสูงขึ้นตรงกลาง

รูปที่ 2.5 แสดงลักษณะโค้งราบ

2.2.1.2 โค้งกระโจม (Tented Arch = TA) ลายเส้นตรงกลางเกิดเป็นเส้นพุ่งขึ้นจากแนวนอน เป็นมุมแหลมหรือมุมฉาก

รูปที่ 2.6 แสดงลักษณะโค้งกระโจม

2.2.2 กลุ่มมัดหวาย(Loop)

2.2.2.1 มัดหวายปัดขวา (Right Slant Loop = RSL) มีสันดอนเพียงจุดเดียว มีเส้นวกหลักที่สมบูรณ์อย่างน้อย 1 เส้นมีทิศทางไปด้านขวา

รูปที่ 2.7 แสดงลักษณะมัดหวายปัดขวา

2.2.2.2 มัดหวายปัดซ้าย (Left Slant Loop = LSL) มีสันดอนเพียงจุดเดียว มีเส้นวกหลักที่สมบูรณ์อย่างน้อย 1 เส้นมีทิศทางไปด้านซ้าย

รูปที่ 2.8 แสดงลักษณะมัดหวายปัดซ้าย

2.2.2.3 **มัดหวายคู่** (Double = D) **มีลักษณะคล้ายกับลายนิ้วมือแบบมัดหวายข้างบน แต่มากอดหรือล้ำกันจนเกิดมีสันดอน 2 จุด โดยไม่จำเป็นต้องมีขนาดเท่ากัน ประกอบด้วย**

รูปที่ 2.9 แสดงลักษณะมัดหวายคู่ปัดขวา

รูปที่ 2.10 แสดงลักษณะมัดหวายคู่ปัดซ้าย

2.2.3 กลุ่มก้นหอย (Whorl)

ลายนิ้วมือที่มีเส้นเวียนรอบเป็นวงจร ลักษณะคล้ายรูปไข่ วงกลม หรือลักษณะอื่นๆ ประกอบด้วย

2.2.3.1 **ก้นหอยธรรมดา** (Plain Whorl = W)

รูปที่ 2.11 แสดงลักษณะก้นหอยธรรมดา

2.2.3.2 ก้นหอยกระเป๋ากลางปัดขวา (Right Central Pocket = RCP)

รูปที่ 2.12 แสดงลักษณะก้นหอยกระเป๋ากลางปัดขวา

2.2.3.3 ก้นหอยกระเป๋ากลางปัดซ้าย (Left Central Pocket = LCP)

รูปที่ 2.13 แสดงลักษณะก้นหอยกระเป๋ากลางปัดซ้าย

2.2.3.4 ก้นหอยกระเป๋าข้างปัดขวา (Right Lateral Pocket = RLP)

รูปที่ 2.14 แสดงลักษณะก้นหอยกระเป๋าข้างปัดขวา

2.2.3.5 ก้นหอยกระเป๋าข้างปัดซ้าย (Left Lateral Pocket = LLP)

รูปที่ 2.15 แสดงลักษณะก้นหอยกระเป๋าข้างปัดซ้าย

2.2.4 กลุ่ม ซับซ้อน (Accidental Whorl = AW)

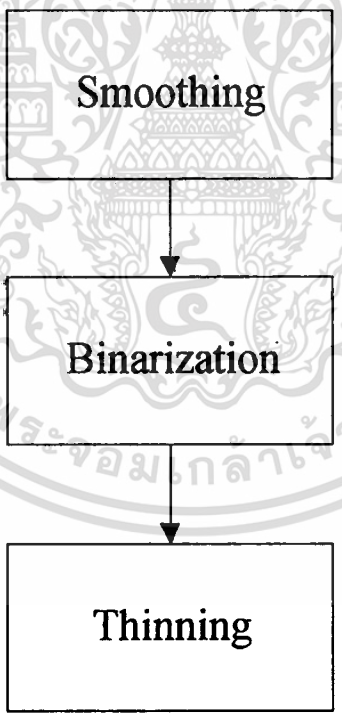
ลายนิ้วมือที่มีลักษณะพิเศษที่ไม่สามารถจัดเข้ากับกลุ่มใดกลุ่มหนึ่งได้มักประกอบจากลายนิ้วมือ 2 กลุ่ม มาผสมกัน

รูปที่ 2.16 แสดงลักษณะซับซ้อน

บทที่ 3

Image Processing

เนื่องจากปัจจุบันการทำงานของระบบจดจำลายนิ้วมือแบบอัตโนมัติ เป็นการประมวลผลภาพแบบดิจิตอล การนำภาพที่ได้จาก เครื่องสแกนเนอร์ กล้องวีดีโอ หรือเครื่องอ่านลายนิ้วมือ ภาพที่ได้มาจะมีการรบกวนจากซึ่งต่างๆเกิดขึ้น ทำให้ภาพที่ได้ผิดเพี้ยนจากความจริง ดังนั้นจึงต้องผ่านขบวนการ Preprocessing ซึ่งจะเป็นขบวนการที่ทำให้ภาพมีคุณสมบัติที่ดีขึ้นโดยขจัดสัญญาณรบกวนออกและขจัดข้อมูลที่เกินออกด้วย และขบวนการ Postprocessing เป็นขบวนการที่แก้ไขรูปแบบลักษณะสำคัญที่ผิดเนื่องจากการเลอะของหมึก หรือไขมันที่นิ้วมือก่อนพิมพ์หมึกจริงได้ภาพที่สมบูรณ์ขึ้น



รูปที่ 3.1 แสดงขั้นตอน Preprocessing

3.1 Smoothing

เป็นวิธีที่ใช้กำจัด noise ที่ปนมากับภาพลายนิ้วมือ ที่เป็นภาพ grey - levels วิธีการต่างๆ ที่มีใช้กันนั้นจะมีคุณสมบัติเป็น low pass filter ซึ่งในที่นี้จะใช้หลักการของ average filtering หลักการนี้จะใช้ 3x3 windows ภายใน window ดังกล่าวจะประกอบไปด้วยจุดต่าง ๆ ตั้งแต่ X0-X9 ดังรูปที่ 3.2 โดยมีจุด X0 เป็นจุดที่เราพิจารณาถึง จุด X0 นี้เปลี่ยนแบบวนรอบไปจนครบทุกจุดภาพ ขณะที่แต่ละจุดภาพมีตำแหน่งเป็น X0 ก็จะมีการหาค่า X0 นั้นใหม่ด้วยสมการ 3.1

$$X0 = \left(\frac{1}{9}\right) \sum_{i=0}^{8} \left(X_i\right) \qquad \text{..... (3.1)}$$

X4	X3	X2
X5	X0	X1
X6	X7	X8

รูปที่ 3.2 แสดงตำแหน่งของ 3 * 3 windows

3.2 Binarization

การ binarization เป็นขั้นตอนการทำให้ภาพ grey - levels เป็นภาพ ขาวดำ โดยอาศัยค่าจาก histogram เพื่อกำหนดจุด threshold ในการแปลงเป็นภาพ ขาวดำ ดังสมการ 3.2

$$g(x,y) = \begin{cases} 1 \; ; \; f(x,y) \; > \; T \\ 0 \; ; \; f(x,y) \; \leq \; T \end{cases} \qquad \text{..... (3.2)}$$

f(x,y) คือ ค่าของ grey levels ของ ตำแหน่ง x,y

T คือ ค่า threshold

g(x,y) คือ ค่าใหม่ที่ได้ของจุด (x,y)

3.2.1 การหาค่า Threshold

การหาค่า threshold อัตโนมัติด้วย minimum cross-entropy thresholding จากในอดีตการเลือกค่า threshold นั้นยังต้องใช้มนุษย์ในการทำ กล่าวคือมนุษย์จะเป็นผู้สังเกตดูภาพที่ได้จากการทำ binarization ว่าค่า threshold ค่าไหนที่ให้รายละเอียดต่างๆ ของภาพได้ชัดเจนที่สุด ก็จะเลือกค่านั้นไปใช้ เป็นที่สังเกตได้ว่ามนุษย์ในยุคต่อๆมาเริ่มตระหนักถึงวิธีการดังกล่าวนั้นใช้เวลามาก อันเนื่องมาจากข้อมูลที่เป็นลายนิ้วมือเริ่มมีจำนวนมากขึ้นเรื่อยๆ อีกทั้งยังความผิดพลาดจากการเลือกค่า threshold เอง เช่น ปัญหาจากภาพลายนิ้วมือหนึ่ง ที่คนสองคนต่างให้ค่า threshold คนละค่าแล้วบอกว่าของตนเป็นค่าที่ถูกต้อง หรือ คนคนเดียวกันที่ให้ค่า threshold มากกว่าหนึ่งค่า ซึ่งตัวเขาเองก็ไม่รู้ว่าจะเลือกค่าไหนดี ฉะนั้นจึงเป็นการดีถ้ามีการหาค่า threshold อย่างอัตโนมัติ ในที่นี้จะเสนอวิธีการหาด้วยวิธี minimum cross-entropy thresholding หลักการพื้นฐานที่ควรทราบมีดังนี้

$$D(P,Q) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} \quad \text{.....} \quad (3.3)$$

D(P,Q) คือ distance ระหว่าง การแจกแจง 2 เหตุการณ์ความน่าจะเป็น

P คือ เป็นเซตของความน่าจะเป็นของเหตุการณ์แรก

ซึ่ง $P = \{p_1, p_2, \ldots, p_n\}$

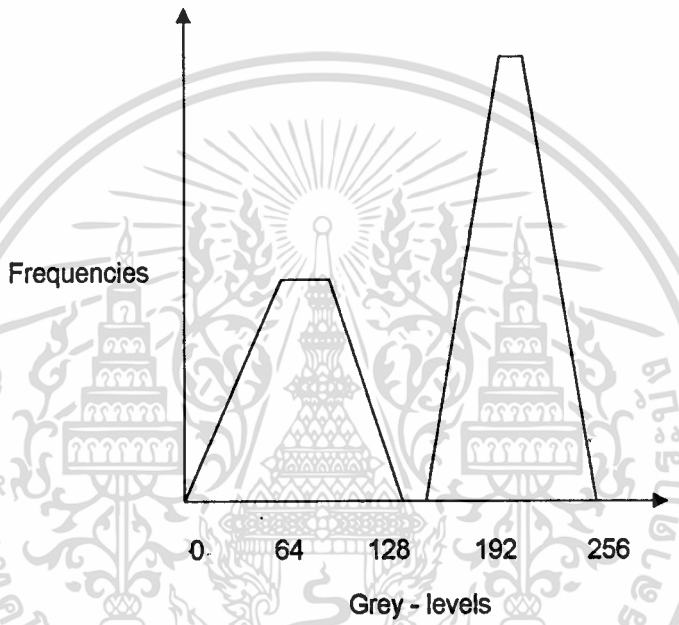
Q คือ เป็นเซตของความน่าจะเป็นของเหตุการณ์ที่สอง

ซึ่ง $Q = \{q_1, q_2, \ldots, q_n\}$

เนื่องจาก $\Sigma_{i=1}^{n} p_i = \Sigma_{i=1}^{n} q_i = 1$, จากอสมการของ Gibbs $(-\Sigma p_i \log p_i \leq -\Sigma p_i \log q_i)$ จะพิสูจน์ได้ว่า $D(P,Q) \geq 0$, ถ้า $p_i \neq q_i$, ทุกๆ ค่า i แล้ว D(P,Q) ก็จะไม่มีคุณสมบัติของความสมมาตร คือ $D(P,Q) \neq D(Q,P)$ ในงานประยุกต์บางอย่างจะมีการใช้รูปแบบของ symmetric version ดังนี้คือ

$$D_s(P,Q) = D(P,Q) + D(Q,P) \quad \text{.....} \quad (3.4)$$

มีส่วนที่ควรทราบต่อไปคือ กำหนดให้ $F = \{f(x,y)\}_{M \times N}$ เป็นภาพหนึ่งทาง digital ซึ่งมีระดับของ grey - levels L พร้อมด้วย $f(x,y) \in \{0,1,...,L\}$, กำหนดต่อไปให้ h_i เป็นความถี่ของ grey - levels ณ. ระดับที่ i และกำหนดให้ $p_i = \frac{h_i}{(M \times N)}$, ซึ่ง p_i คือ ความน่าจะเป็นของระดับ grey - levels ที่ i นั้นๆ MxN เป็นขนาดของภาพ



รูปที่ 3.3 แสดง Synthetic histogram

จากรูป Synthetic Histogram ข้างบนนี้ กำหนดให้ S เป็นค่า Threshold และมีข้อสันนิษฐานเบื้องต้นที่ควรทราบคือ จากช่วง [0-S] ใน grey - levels นั้นจะเป็นส่วนของ object และ ช่วง [(S+1)-L] จะเป็นส่วนของ background ซึ่งในที่ L = 255 (256 grey - levels) ในบางทฤษฎีจะใช้ช่วง [0-(S-1)] เป็น object และ [S-L] เป็น background แทนซึ่งก็ไม่ผิดเช่นกัน เมื่อทราบหลักการพื้นฐานแล้วเราก็มาเริ่มด้วยหลักการที่ใช้จริงซึ่งมีดังนี้ เริ่มจากการสมการ

$$D_S(P,Q) = \sum_i p_i \log \frac{p_i}{q_i} + \sum_i q_i \log \frac{q_i}{p_i} \quad \text{..... (3.5)}$$

กำหนดให้ S เป็นค่า threshold สำหรับการทำ segmentation ดังนั้นถ้าเรามีการแจกแจงความน่าจะเป็นนี้กับ object ใน grey - levels ก็จะกำหนดได้ดังนี้ $P_O = \{p_1^o, p_2^o, \ldots, p_s^o\}$ และการแจกแจงความน่าจะเป็นนี้กับ background เป็น $P_B = \{p_{s+1}^B, p_{s+2}^B, \ldots, p_L^B\}$ ที่ซึ่ง

$$p_i^O = \frac{h_i}{P_S}, \quad i = 1,2, \ldots, S; \quad P_S = \sum_{i=1}^{S} h_i ; \quad \ldots\ldots\ldots\ldots \quad (3.6)$$

$$p_i^B = \frac{h_i}{MN - P_S}, \quad i = S+1, S+2, \ldots, L \quad \ldots\ldots\ldots\ldots \quad (3.7)$$

ข้อสังเกต $\sum_{i=1}^{S} p_i^O = \sum_{i=S+1}^{L} p_i^B = 1$, ใน model Q ก็เช่นกันจะมี $Q_O = \{q_1^O, q_2^O, \ldots, q_S^O\}$ สำหรับ object และ $Q_B = \{q_{S+1}^B, q_{S+2}^B, \ldots, q_L^B\}$ สำหรับ background เมื่อถึงตอนนี้เราจะกำหนด cross entropy ของขอบเขต object ได้เป็น

$$D_O(s) = D_S(P_O, Q_O) \quad \ldots\ldots\ldots\ldots \quad (3.8)$$

และ cross entropy ของขอบเขต background ได้เป็น

$$D_B(s) = D_S(P_B, Q_B) \quad \ldots\ldots\ldots\ldots \quad (3.9)$$

สุดท้ายก็จะหา total cross entropy ดังสมการต่อไปนี้

$$D(s) = D_O(s) + D_B(s)$$

$$= \sum_{i=1}^{s} p_i^o \log\left(\frac{p_i^o}{q_i^o}\right) + \sum_{i=1}^{s} q_i^o \log\left(\frac{q_i^o}{p_i^o}\right) +$$

$$\sum_{i=s+1}^{L} p_i^B \log\left(\frac{p_i^B}{q_i^B}\right) + \sum_{i=s+1}^{L} q_i^B \log\left(\frac{q_i^B}{p_i^B}\right) \quad \ldots\ldots\ldots\ldots \quad (3.10)$$

สมการดังกล่าวนี้เราจะต้องหาค่าที่น้อยที่สุดของ D(s) แล้วพิจารณาค่า S ให้เป็นค่า threshold นั้น ปัญหาจากสมการที่ (3.4) นั้นจะเห็นว่าเรายังไม่ทราบวิธีการหา Q_B และ Q_O เลย

ดังนั้นขั้นตอนต่อไปก็จะศึกษาเกี่ยวกับวิธีการหาค่าดังกล่าวโดยใช้ model ของการแจกแจงความน่าจะเป็นของ Poisson

$$q_i^o = \frac{e^{-\lambda_o} \lambda_o^i}{i!}, \quad i = 1,2,\ldots,S; \qquad \ldots\ldots\ldots\ldots \quad (3.11)$$

$$q_i^B = \frac{e^{-\lambda_B} \lambda_B^i}{i!}, \quad i = S+1, S+2, \ldots, L; \qquad \ldots\ldots\ldots\ldots \quad (3.12)$$

ที่ซึ่งค่าของ λ_o และ λ_B หาได้ดังนี้

$$\lambda_o = \frac{\sum_{i=1}^{s} i h_i}{\sum_{i=1}^{s} h_i} \qquad \ldots\ldots\ldots\ldots \quad (3.13)$$

$$\lambda_B = \frac{\sum_{i=s+1}^{L} i h_i}{\sum_{i=s+1}^{L} h_i} \qquad \ldots\ldots\ldots\ldots \quad (3.14)$$

สรุปหลักการที่เขียนมาทั้งหมดตาม Algorithm 1 ได้ดังนี้

Algorithm 1

Begin

Mindiv = High-value; {กำหนดให้ Mindiv เป็นค่าที่มากที่สุดเท่าที่เป็นไปได้}

for $2 \leq S \leq L-1$ do

Begin

คำนวณหาค่า p_i^o , i = 1,2,...,S โดยใช้สมการที่ (3.6);

คำนวณหาค่า p_i^B , i = S+1,...,L โดยใช้สมการที่ (3.7);

คำนวณหาค่า λ_o , โดยการใช้สมการที่ (3.13);

คำนวณหาค่า λ_B , โดยการใช้สมการที่ (3.14);

คำนวณหาค่า q_i^o , i = 1,2,...,S โดยใช้สมการที่ (3.11);

คำนวณหาค่า q_i^B , i = S+1,...,L โดยใช้สมการที่ (3.12);

คำนวณหาค่า D(s) , โดยใช้สมการที่ (3.10);

if(Mindiv > D(s))

Begin

Mindiv = D(s);

threshold = S;

End

End

End.

เนื่องจาก Algorithm 1 นั้นถ้าวิเคราะห์ให้ดีจะเห็นว่า $\sum_{i=1}^{s} q_i^o \neq 1$, $\sum_{i=s+1}^{L} q_i^B \neq 1$ และจากอสมการของ Gibbs ส่งผลทำให้ค่า Q_O และ Q_B ไม่ถูกต้องด้วยเช่นกัน ดังนั้นเพื่อให้ตรงกับอสมการของ Gibbs เราจะทำการ normalize ค่า Q_O และ Q_B ด้วยดัง Algorithm 2

Algorithm 2

Begin

Algorithm 1 ที่ซึ่งใช้ค่าของ Q_O และ Q_B ที่ถูก normalize แล้ว

$(\sum_{i=1}^{s} q_i^o = 1 , \sum_{i=s+1}^{L} q_i^B = 1)$

End.

3.2.2 Smooth binary

หลังจากที่มีการหาค่า threshold ได้แล้วเราก็จะนำค่า threshold ดังกล่าวมาทำภาพขาวดำดังสมการที่ (3.2) เนื่องจากภาพ binary image ที่ได้จากการ threshold หรือ edge detection จะได้ภาพที่มี 2 ระดับได้แก่ จุดขาว(0) กับจุดดำ(1) ซึ่งบางครั้งผลที่ได้อาจจะมี noise ปนเข้ามา noise ดังกล่าวอาจจะมีสาเหตุมาจาก algorithm ที่ใช้ทำหรือตัว hardware เอง noise ในภาพ binary image นั้นถูกรวบรวมได้ดังนี้ ขอบเขตที่ผิดปกติ (irregular boundaries) , โฮลด์ขนาดเล็ก (small holes) , มุมที่ผิดปกติ (missing corners) , จุดอิสระ (isolated points) แนวทางในการพิจารณานี้จะใช้สมการบูลลีนในการตรวจสอบ ซึ่งหลักการดังกล่าวจะใช้ window 3 x 3 ซึ่งมีจุด p เป็นจุดศูนย์กลางพร้อมกันนั้นก็มีจุดข้างเคียงเพื่อแทนค่าในสมการบูลลีนนั้นเช่นกันดูรูปที่ 3.4

a	b	c
d	p	e
f	g	h

รูปที่ 3.4 แสดงตำแหน่ง windows ของ smoothing binarization

หลักการทำ smooth ที่จะเสนอนี้มีจุดมุ่งหมายดังต่อไปนี้ (1) เติม hole ด้วยจุดดำ (2) เติมรอยบากของแนวตรงที่ขาดหายไปด้วยจุดดำ (3) กำจัดจุดขาวอิสระ (4) กำจัดส่วนเกินของแนวตรงที่เกินมาด้วยจุดขาว (5) เติมมุมที่ขาดหายไปหัวข้อทั้งหมดนี้นำมาสรุปเป็นสมการบูลลีนได้ 6 หัวข้อดังนี้

$$B_1 = p + b \cdot g \cdot (d + e) + d \cdot e \cdot (b + g) \quad \text{..... (3.15)}$$

จากเงื่อนไขจะเป็นการตรวจสอบรอยบากของแนวเส้นตรงที่ขาดหายไป ซึ่งถ้าเงื่อนไข B1 เป็นจริงจุด p จะถูกกำหนดให้มีค่าเป็น 1 อื่นๆจะกำหนดให้มีค่าเป็น 0

$$B_2 = p \cdot [(a + b + d) \cdot (e + g + h) + (b + c + e) \cdot (d + f + g)] \quad \text{... (3.16)}$$

เงื่อนไข จะเป็นการตรวจสอบโฮลด์ที่เป็นอิสระและโฮลด์ในลักษณะต่างๆที่เกือบจะเป็นอิสระ

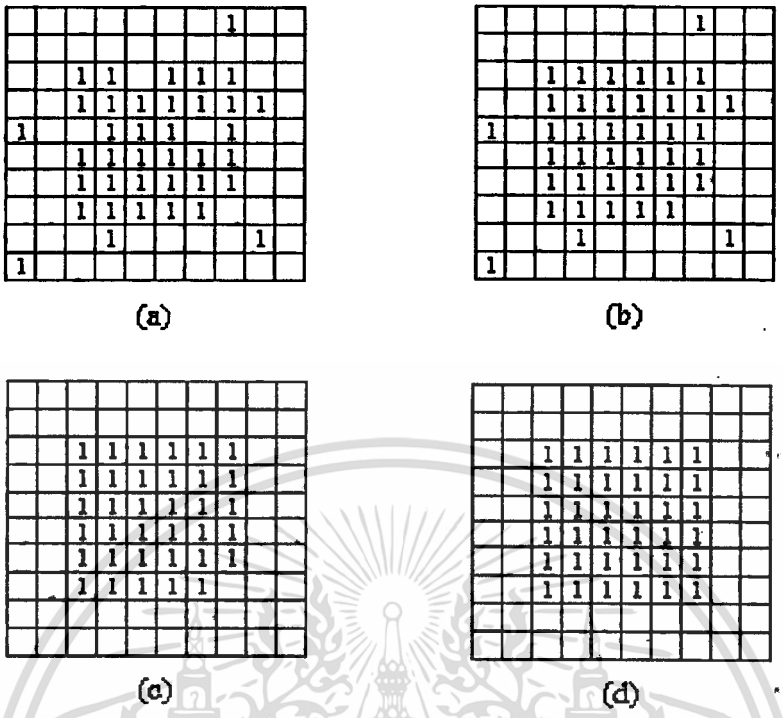
$$B_3 = \overline{p} \cdot (d \cdot f \cdot g) \cdot \overline{(a + b + c + e + h)} + p \quad \text{..... (3.17)}$$

$$B_4 = \overline{p} \cdot (a \cdot b \cdot d) \cdot \overline{(c + e + f + g + h)} + p \quad \text{..... (3.18)}$$

$$B_5 = \overline{p} \cdot (e \cdot g \cdot h) \cdot \overline{(a + b + c + d + f)} + p \quad \text{..... (3.19)}$$

$$B_6 = \overline{p} \cdot (b \cdot c \cdot e) \cdot \overline{(a + d + f + g + h)} + p \quad \text{..... (3.20)}$$

การนำสมการต่างๆไปใช้ บางครั้งไม่จำเป็นต้องใช้ทั้งหมด 6 สมการขึ้นอยู่กับขบวนการของ thinning ด้วย เช่น thinning จะลบมุมของภาพก่อนจึงจะเข้าทำขบวนการต่างๆต่อไป ดังนั้นจึงไม่จำเป็นต้องใช้สมการ 317 - 3.20 ตัวอย่างของการนำไปใช้กับ binary image ทั้ง 6 สมการดังรูปที่ 3.5



รูปที่ 3.5 (a) original image (b) ผลลัพธ์ของการใช้เงื่อนไข B_1 (c) ผลลัพธ์ของการใช้เงื่อนไข B_2 (d) ผลลัพธ์ของการใช้เงื่อนไข B_3 จนถึง B_6

3.3 Thinning

ภายหลังจากที่ได้ภาพ binary image แล้ว ก่อนที่จะทำการวิเคราะห์ด้วยวิธีการจดจำลายนิ้วมือต่างๆนั้นหลายๆระบบจะมีการทำ thinning เพื่อให้ภาพนั้นบางลงเป็นรูปเส้นโครงร่าง(median axis) ซึ่งลักษณะดังกล่าวจะทำให้วิเคราะห์ภาพได้ง่าย

3.3.1 ทฤษฎีและหลักการเบื้องต้น

คณิตศาสตร์แขนงหนึ่งที่ใช้ในการวิเคราะห์เพื่อการทำ thinning ส่วนใหญ่จะนิยมใช้ สัณฐานวิทยา(morphology) ในการอธิบาย เริ่มต้นด้วยการกำหนดให้ X เป็นเซตของจุดภาพ (เฉพาะเส้นนูน) ถ้าให้ a เป็นจุดภาพใดๆของเส้นนูนแล้วแสดงว่า $a \in X$ รวมทั้งถ้ากำหนดให้ T เป็น template ที่ใช้สำหรับการทำคอนโวลูชัน(convolution) กับภาพลายนิ้วมือเพื่อให้ได้มาซึ่ง

เส้นโครงร่าง จากหลักการของสัณฐานวิทยาจะมีการใช้ตัวดำเนิน การบวก และ ลบ กับเซตแบบ Minkowski ได้ดังนี้

$$X \oplus T = \{a + b : a \in X, b \in T\} = \bigcup_{b \in T} X_b \quad \text{..... (3.21)}$$

และ

$$X \ominus T = (X^c \oplus T)^c = \bigcap_{b \in T} X_b \quad \text{..... (3.22)}$$

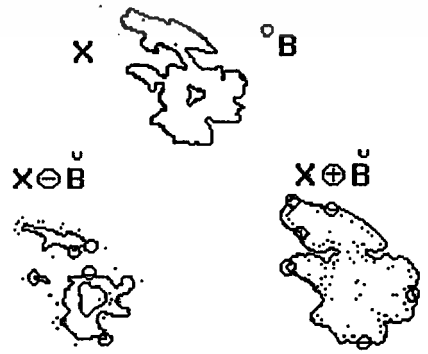
จากหลักการสัณฐานวิทยาจะมีการดำเนินการอีโรชั่นและดิเลชั่นกับ X ซึ่งสามารถแสดงด้วยการดำเนินการแบบ Minkowski ของการบวกและการลบได้ การดำเนินการอีโรชั่นของ X ด้วย T ถูกกำหนดด้วยเซตของจุดต่างๆ x เพื่อให้เกิดการทรานสเลชั่น T ไป x ซึ่งเขียนแทนด้วย T_x ลักษณะดังกล่าวยังต้องอยู่ภายในเซต X ซึ่งแสดงได้ดังนี้

$$X \ominus \check{T} = \{x : T_x \subseteq X\} = \bigcap_{b \in \check{T}} X_b \quad \text{..... (3.23)}$$

ในส่วนของดิเลชั่นนั้นกำหนดได้ว่า ดิเลชั่นของ X ด้วย T เป็นเซตของจุดต่างๆ x เพื่อให้ T_x นั้นอินเตอร์เซคกับ X ซึ่งแสดงได้ดังนี้

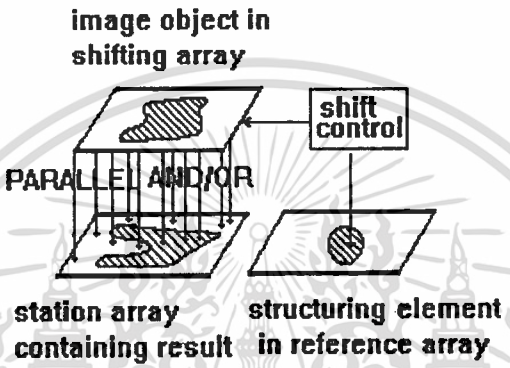
$$X \oplus \check{T} = \{x : T_x \cap X \neq \emptyset\} = \bigcup_{b \in \check{T}} X_b \quad \text{..... (3.24)}$$

จากลักษณะดังกล่าวข้างต้นนี้ดูรูปที่ 3.6 ประกอบ



รูปที่ 3.6 แสดงการอิเลชั่นและดิเลชั่น

จากรูปที่ 3.6 จะเห็นได้ว่า X เปรียบได้กับภาพๆ หนึ่ง พร้อมกับมีเทมเพลต B ซึ่งลักษณะเทมเพลต B นี้จะเป็นเทมเพลตในทางอะนาลอก(ดูได้จากลักษณะที่เป็นวงกลม) ฉะนั้นอีโรชั่นของ X ด้วย B และ ดิเลชั่นของ X ด้วย B นั้นดูได้จากรูปเบื้องล่างถัดมาซ้ายขวาตามลำดับ พร้อมกันนั้นถ้ามีการนำไปใช้งานในทางปฏิบัติของระบบขนานแล้ว การดำเนินการ อีโรชั่น และ ดิเลชั่นนั้นอาจแทนด้วยการดำเนินการ ออร์(OR) หรือ แอนด์(AND) ดังรูปที่ 3.7



รูปที่ 3.7 ลักษณะทางลอจิกในการทำ thinning

ยังมีตัวดำเนินการที่สำคัญในทางสัณฐานวิทยาก็คือ hit/miss ทรานฟอร์ม (transform)

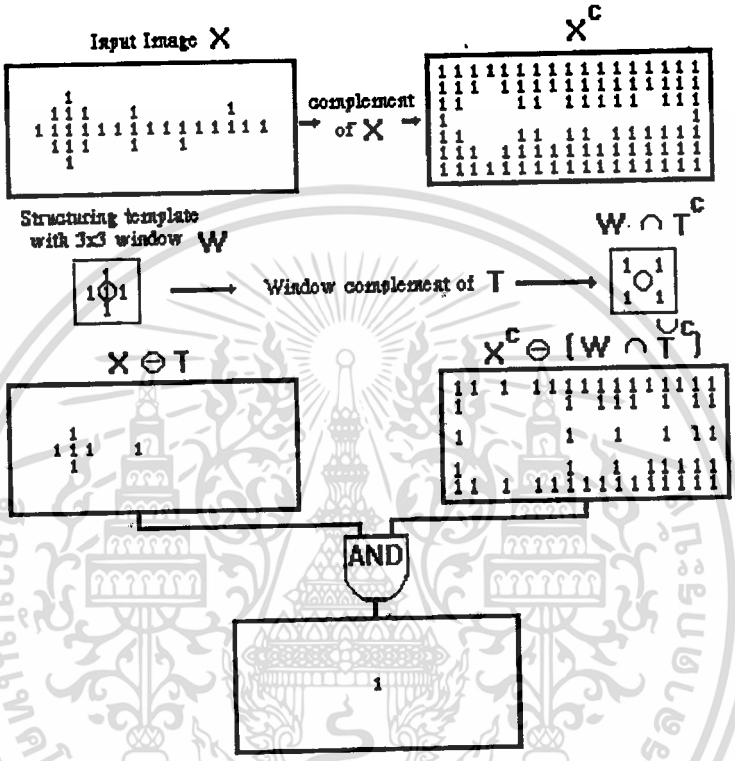
กำหนดด้วย T ซึ่งประกอบด้วย ซับเซต T^1 และ T^2 hit/miss ทรานฟอร์มของ X ด้วย T ถูกกำหนดอย่างเซตของจุดต่างๆที่ T_X^1 อยู่ใน X และ T_X^2 อยู่ใน X^C ที่ซึ่ง X^C เป็นคอมพลีเมนต์ของ X hit/miss โอเปอร์เรชั่นแสดงได้ด้วยนิพจน์ดังต่อไปนี้

$$X \circledast T = \{x : T_x^1 \subset X ; T_x^2 \subset X^c\}$$
$$= (X \ominus \breve{T}^1) \cap (X^c \ominus \breve{T}^2) \quad \text{..... (3.25)}$$

ถ้าเลือกเทมเพลต T^2 เป็นวินโดว์คอมพลีเมนต์ของ T^1 จากสมการสามารถเขียนใหม่ได้เป็น

$$X \circledast T = (X \ominus \breve{T}^1) \cap (X^c \ominus (W \cap \breve{T}^2)) \quad \text{.....} \quad (3.26)$$

ที่ซึ่ง W เป็น วินโดว์ที่มีขอบเขตจำกัดค่าหนึ่งจากรูปมีขนาด 3 x 3 ดูรูปที่ 3.8 ประกอบ



รูปที่ 3.8 แสดงการทำ hit and miss transform

เป็นที่สังเกตได้ว่า อิมเมจ X ถูกอิโรชั่นด้วยเซต T ซึ่งก็คือ $X \oplus \breve{T}$ เปรียบได้กับตำแหน่งโลกัส (locus) ทุกๆตำแหน่งที่มี T บรรจุอยู่ภายใน X สุดท้ายที่จะกล่าวต่อไปนั้นเป็นการดำเนินการ thinning ของ X ด้วย T ถูกกำหนดได้ดังนี้

$$X \otimes T = X / (X \circledast T) \quad \text{.....} \quad (3.27)$$

ตัวดำเนินการ / มีความหมายเช่นเดียวกันกับผลต่างของเซตนั้นเอง

จากสมการ 3.27 ได้เกิดแนวความคิดที่จะสร้างขั้นตอนที่จะดำเนินการซ้ำๆ ในลักษณะ ค้นหาและลบด้วยเซตๆ หนึ่งของสทรัคเจอร์เทมเพลต(structure template) ดังนั้นการดำเนินการ thinning ของ X ด้วย T จะเป็นการดำเนินการ thinning ของ X ด้วยซีเควนซ์ของ T $\{T^n\}$

$$X \otimes T = (...((X \otimes T^1) \otimes T^2)... \otimes T^n) \quad \text{..... (3.28)}$$

นอกจากนี้แล้วยังมีการดำเนินการซีเควนซ์ของสทรัคเจอร์เทมเพลตกับการทำอีโรชั่นและ hit/miss ทรานฟอร์มได้อีกด้วยตามลำดับ

$$X \ominus T = (X \ominus T^1) \cap (X \ominus T^2) \cap \text{.....} \quad \text{(3.29)}$$
$$(X \ominus T^n)$$

$$X \circledast T = (X \circledast T^1) \cup (X \circledast T^2) \cup \text{.....} \quad \text{(3.30)}$$
$$(X \circledast T^n)$$

จากสมการ 3.28 จะเห็นได้ว่าสมการดังกล่าว จะเป็นการดำเนินการแบบซีเควนเชียลส่วน สมการ 3.29 และ 3.30 จะเป็นการดำเนินการแบบขนาน จากหลักการเบื้องต้นที่กล่าวมาแล้วนั้นนำไปสู่การหา thinning

3.3.2 One - Pass Parallel Thinning

เป็นวิธีของ Jang และ Chin มาใช้วิธีดังกล่าวมีชื่อว่า One-Pass Parallel Thinning วิธีดังกล่าวจะเริ่มด้วยการใช้เทมเพลต A ซึ่งมีด้วยกัน 30 เทมเพลต เทมเพลตดังกล่าวจะใช้เพื่อการคอนโวลูชั่น โดยเทมเพลต A^1 - A^{20} มีไว้เพื่อตรวจสอบจุดภาพที่เป็นขอบที่สามารถลบได้ ส่วนเทมเพลต A^{21} - A^{30} มีไว้เพื่อตรวจสอบการคงไว้ซึ่งจุดดังกล่าวดูรูปที่ 3.9

ถ้ากำหนดให้เซต S แทนเซตของจุดภาพ และ เซต D แทนเซตของจุดภาพที่สามารถลบได้ ซึ่งจะนิยามเซต D ได้ดังนี้

$$D = \bigcap_{1 \le k \le 20} (S \circledast A^k) \cup \bigcap_{21 \le k \le 30} (S \circledast A^k)$$

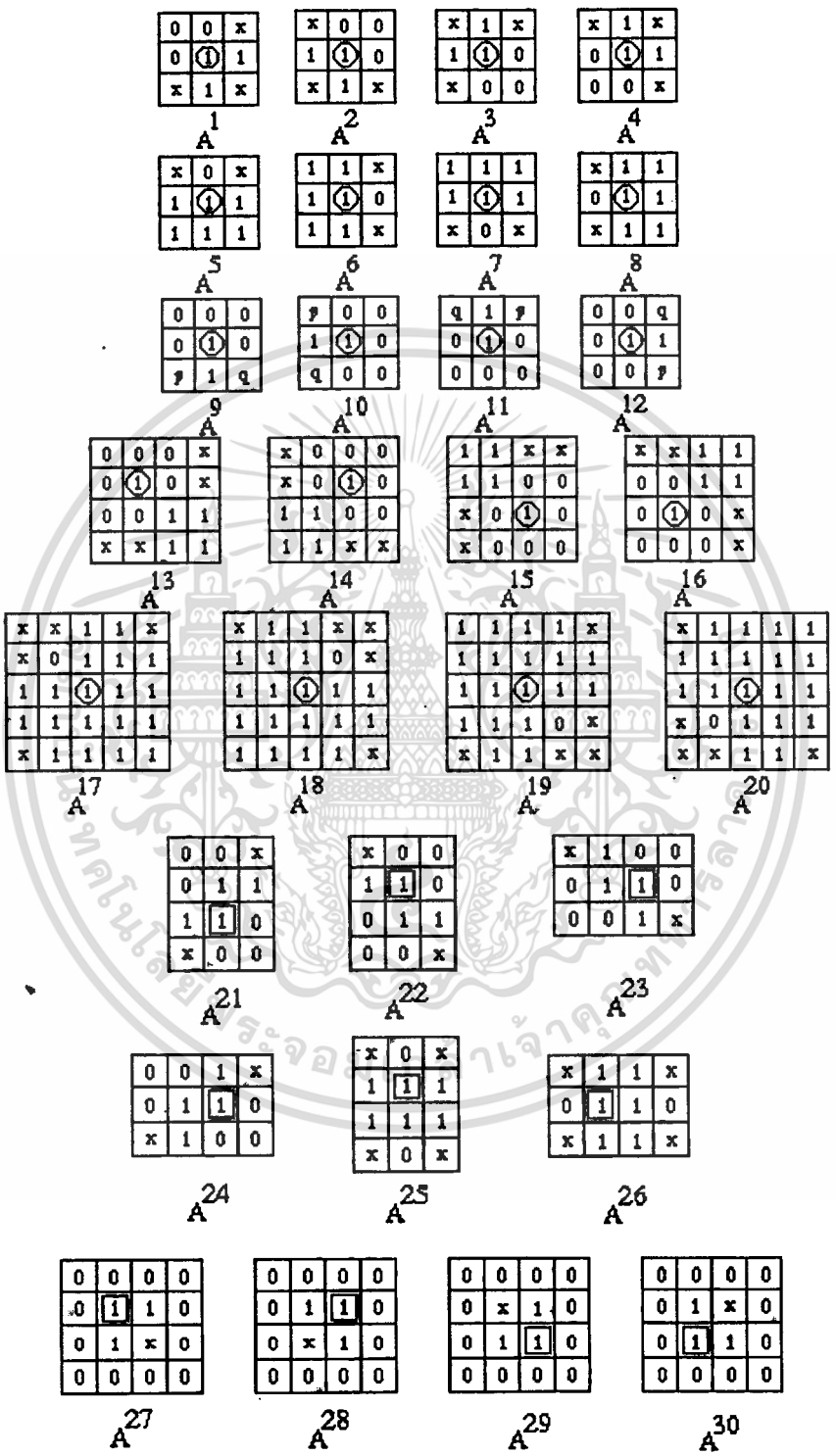
สรุปเป็นอัลกอริทึมได้ดังนี้

1. กำหนดเซต S เป็นเซตของจุดภาพ
2. กำหนดเซต D เป็นเซตของจุดภาพที่สามารถลบได้

$$D = \bigcap_{1 \le k \le 20} (S \circledast A^k) \cup \bigcap_{21 \le k \le 30} (S \circledast A^k)$$

3. ถ้า $D = \varnothing$ หยุด อื่น ๆ ทำข้อ 4
4. $S = S / D$

5. ย้อนกลับไปทำข้อ 2.



รูปที่ 3.9 สทรัคเจอร์เทมเพลตของ One-Pass Parallel Thinning

x = don't care และ p + q = 1

3.3.3 Thinning using Zhang and Suen's method

ในวิธีของ Zhang และ Suen จะเริ่มด้วยการใช้เทมเพลตดังรูปที่ 3.10 เทมเพลตดังกล่าวจะใช้เพื่อการคอนโวลูชั่น เราจะพิจารณาตำแหน่ง P0 และ P0 เป็นจุดภาพสีขาว (1) ในการตรวจสอบจุด P0 ว่าจะเปลี่ยนเป็นสีดำ (0) หรือไม่ แบ่งออกเป็น 2 ส่วน คือ

P1	P2	P3
P8	P0	P4
P7	P6	P5

รูปที่ 3.10 แสดงตำแหน่งของเทมเพลตของ Zhang และ Suen

ส่วนที่ 1

1. A(P0) = 1
2. 1 < B(P0) < 7
3. P2 * P4 * P6 = 0
4. P4 * P6 * P8 = 0

ส่วนที่ 2

1. A(P0) = 1
2. 1 < B(P0) < 7
3. P2 * P4 * P8 = 0
4. P2 * P6 * P8 = 0

A(P0) = นับจุดภาพที่มีคู่ 01 ในทิศทางเดียวกัน

B(P0) = จำนวนจุดภาพที่เป็น 1

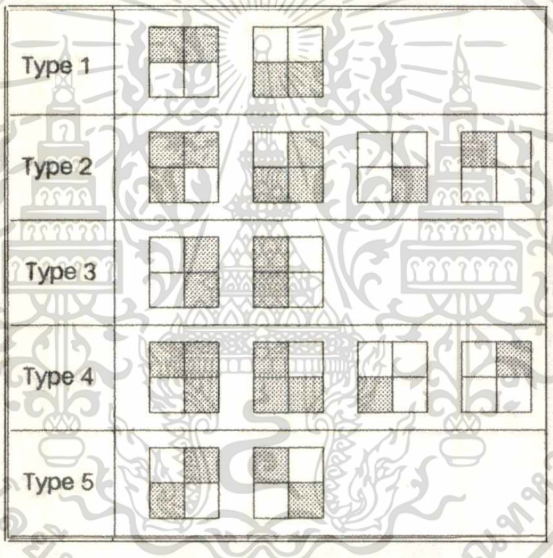
ถ้าเงื่อนไขทั้ง 4 ข้อ ในแต่ละส่วนเป็นจริงทั้งหมด จุดภาพนั้นจะถูกเปลี่ยนจาก ภาพสีขาว (1) เป็นภาพสีดำ (0) โดยจะทำการตรวจสอบจนไม่มีการเปลี่ยนแปลง เป็นการสิ้นสุดการทำ thinning ด้วยวิธีของ Zhang และ Suen

การทำ thinning ดังกล่าวนั้นจะใช้เวลาน้อยเพราะเนื่องมาจากการมี สตรัคเจอร์ เทมเพลต เพียงเทมเพลตเดียว เมื่อเทียบกับวิธีการทำ One-Pass Parallel Thinning ที่มี สตรัคเจอร์ เทมเพลต

30 เทมเพลต วิธีการดังกล่าวก็ให้ผลเป็นที่น่าพอในระดับหนึ่ง และพอที่จะนำไปใช้กับระบบที่เป็นเรียลไทม์ได้ และระบบที่ต้องการเส้นที่เป็นเส้นเดียวจริงๆ

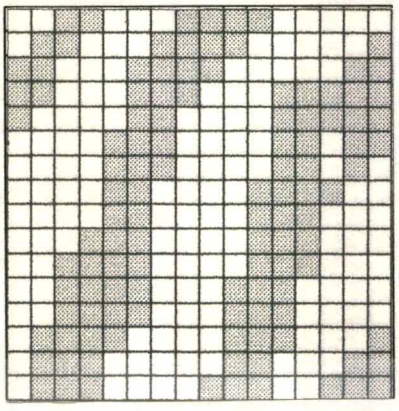
3.4 Direction Pattern

ในการหาทิศทางของ ridge line นั้นเราจะใช้ภาพ binary ที่ยังไม่ผ่านการทำ Thinning และแบ่งภาพออกเป็น subregion ขนาด16*16 pixel ดังนั้นภาพขนาด 256 * 256 จะมี subregion 256 subregion การคิดหาทิศทางใน subregion จะใช้ micropattern 5 แบบเป็นการพิจารณาทิศทางที่ได้



รูปที่ 3.11 แสดงรูปแบบของ micropattern

เราใช้ micropattern ทั้งหมดเทียบกับ subregion ที่จะหาทิศทางจะได้ว่า Type 0, Type 1, Type 2, Type 3 และ Type 4



รูปที่ 3.12 ตัวอย่างของ subregion

จากรูปที่ 3.12 สามารถหาทิศทางของ subregion ได้ คือ Type 1 = 2 , Type 2 = 61 , Type 3 = 34 , Type 4 = 2 และ Type 5 = 1 เนื่องจาก Type 5 $\subset$ Type 2 และ Type 5 $\subset$ Type 4 เพราะฉะนั้น

$$\text{Type 2} = \frac{\text{Type 2} + \text{Type 5}}{\sqrt{2}} \qquad \text{..... (3.31)}$$

$$\text{Type 4} = \frac{\text{Type 4} + \text{Type 5}}{\sqrt{2}} \qquad \text{..... (3.32)}$$

ดังนั้น Type 2 = 43.8 , Type 4 = 2.1 เราจะเลือกค่าที่สูงที่สุดเป็นทิศทางของ subregion นั้นๆ โดย Type 1 = 0° , Type 2 = 45° , Type 3 = 90° , Type 4 = 135° และค่า probability ของ subregion หาได้โดยคิดแต่ละ Type

$$P_i = \frac{\text{Type i}}{\sum_{j=1}^{4} \text{Type j}} \qquad \text{..... (3.33)}$$

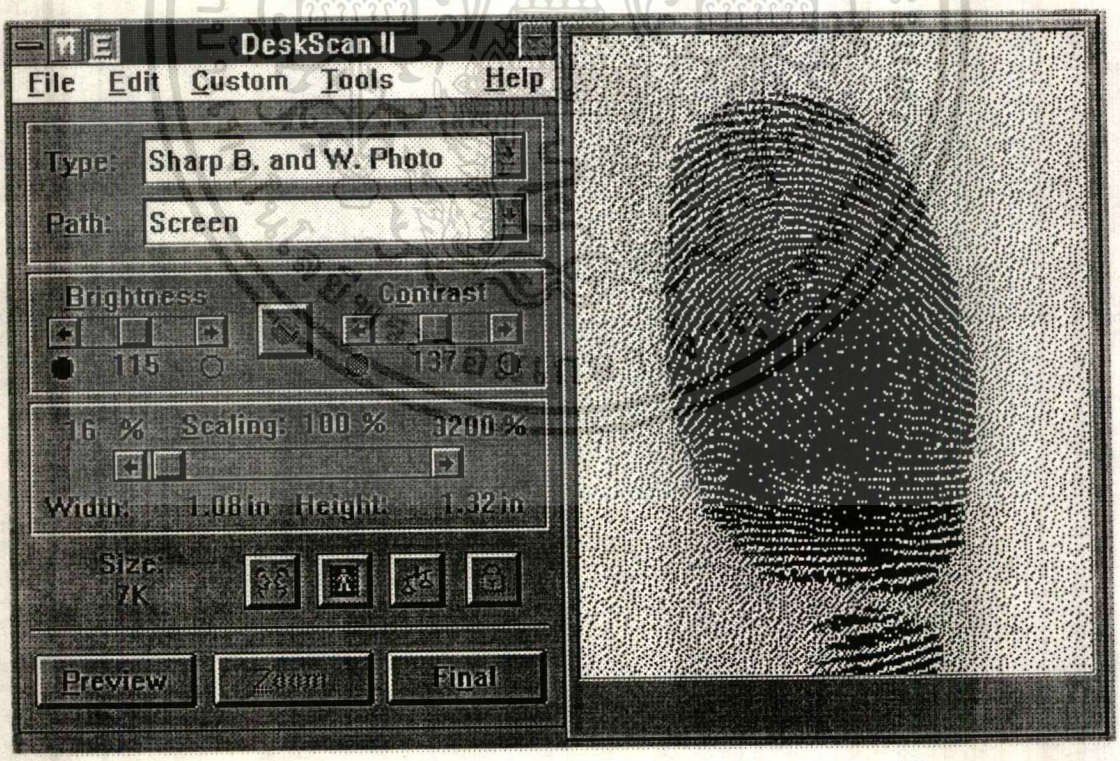
จากรูปที่ 3.12 เราสามารถสรุปได้ว่า (type 1 = 2 , 0.02) (type 2 = 43.8 , 0.53) (type 3 = 34 , 0.41) (type 4 = 2.1 , 0.02) และมาทิศทางเท่ากับ 45° จากภาพทิศทางที่เราได้มานั้นสามารถนำไปใช้หาจุดสำคัญของลายนิ้วมือได้ เช่น จุด Delta และ จุด Core ได้จนทิ้งสามารถแยกชนิดของลายนิ้วมือได้ โดยที่ต้องผ่านขั้นตอนต่างๆ อีก

บทที่ 4

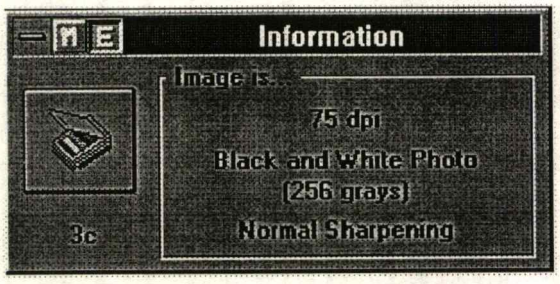
การคำนวณและการสร้าง

4.1 การเก็บภาพลายนิ้วมือ

เราจะได้ภาพจากการพิมพ์ลายนิ้วมือจากหมึกสีดำ พิมพ์ลงบนกระดาษขาว จากการสังเกตลักษณะการพิมพ์ลายนิ้วมือของผู้ทดลอง พบว่าส่วนมากจะใช้บริเวณปลายนิ้วด้านบนทาบลงบนกระดาษ จะทำให้ได้เฉพาะบริเวณ Core เป็นส่วนมาก น้อยมากที่จะพบจุด Delta ด้วย เมื่อได้ภาพลายนิ้วมือมาแล้วก็นำไปสแกนเพื่อที่จะได้ภาพลายนิ้วมือที่เป็น bitmap ในการใช้ สแกนเนอร์จะต้องปรับ แสงสว่าง และ ความคมชัดช่วยเพื่อให้ได้ภาพที่ดีขึ้น



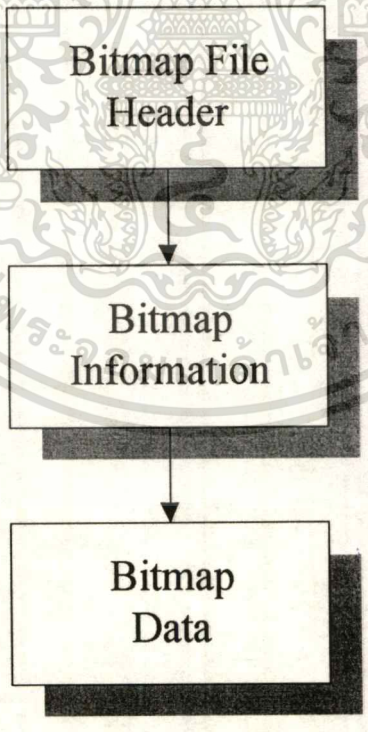
รูปที่ 4.1 แสดงการปรับแต่งภาพ



รูปที่ 4.2 แสดงข้อมูลของ สแกนเนอร์

4.2 การเปิดภาพ bitmap

ภาพ bitmap จะการจัดเก็บตามขนาดของสี เป็นตัวระบุจำนวน bit ต่อหนึ่งสี ดังนั้นถ้าสีมากจะทำให้ภาพที่จัดเก็บมีขนาดใหญ่ตามไปด้วย ภาพ bitmap จะประกอบไปด้วยส่วนระบุชนิดของภาพ , ส่วนข้อมูลของภาพ ขนาด สี อื่นๆ และส่วนสุดท้ายจะเป็นจุดภาพที่จะนำมาแสดงผล



รูปที่ 4.3 แสดงโครงสร้างของภาพ bitmap

Bitmap File Header

Byte #	Data	Details
1 - 2	File type	Must be ASCII text " BM "
3 - 6	Size of the file	In double word (32 bit integer)
7 - 10	Reserved for future use	Must be zero
11 - 14	Byte offset to bitmap data	Offset from the Bitmap File Header (i.e., the start of the file)

ตารางที่ 4.1 แสดง Bitmap File Header

Bitmap Information

Byte #	Data	Details
1 - 4	Number of byte in header	ขนาดของส่วนตัว 40 byte
5 - 8	Width of Bitmap	ความกว้างของภาพ (pixel)
9 - 12	Height of bitmap	ความสูงของภาพ (pixel)
13 - 14	Number of color planes	ต้องเท่ากับ 1
15 - 16	Number of bit per pixel	ส่วนที่บอกสีของภาพ 1 = 2 สี , 4 = 16 สี, 8 = 256 สี , 24 = 16M สี
17 - 20	Type of compression	0 = ไม่มีการอัดภาพ 1 = 8 bit per pixel 2 = 4 bit per pixel
21 - 24	Size of image	ไม่ได้ใช้งาน
25 - 28	Horizontal resolution	ไม่ได้ใช้งาน
29 - 32	Vertical resolution	ไม่ได้ใช้งาน
33 - 36	Number of color indexes used by the bitmap	ไม่ได้ใช้งาน
37 - 40	Number of color indexes important for displaying bitmap	ไม่ได้ใช้งาน

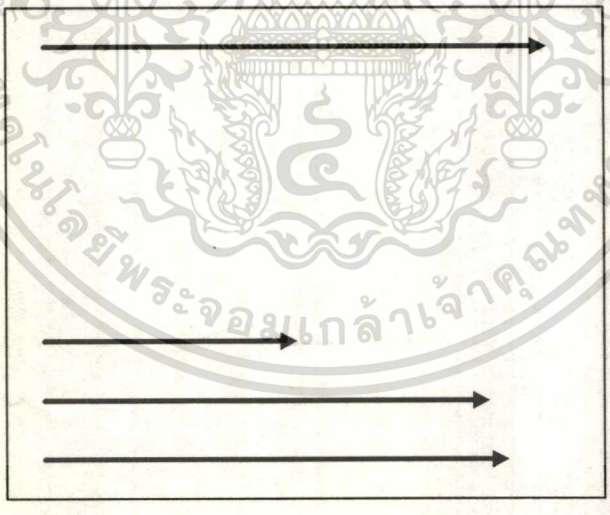
ตารางที่ 4.2 แสดง Bitmap Information

Bitmap Data

Byte #	Data	Details
41	Blue color value	ค่าสีน้ำเงินของภาพ
42	Green color value	ค่าสีเขียวของภาพ
43	Red color value	ค่าสีแดงของภาพ
44	Reserved for future use	ต้องเป็น 0
...	..... Remaining color palette entries	บอกค่าสีต่างๆ 4 bit เหมือนกันไปจนหมดภาพ

ตารางที่ 4.3 แสดง Bitmap Data

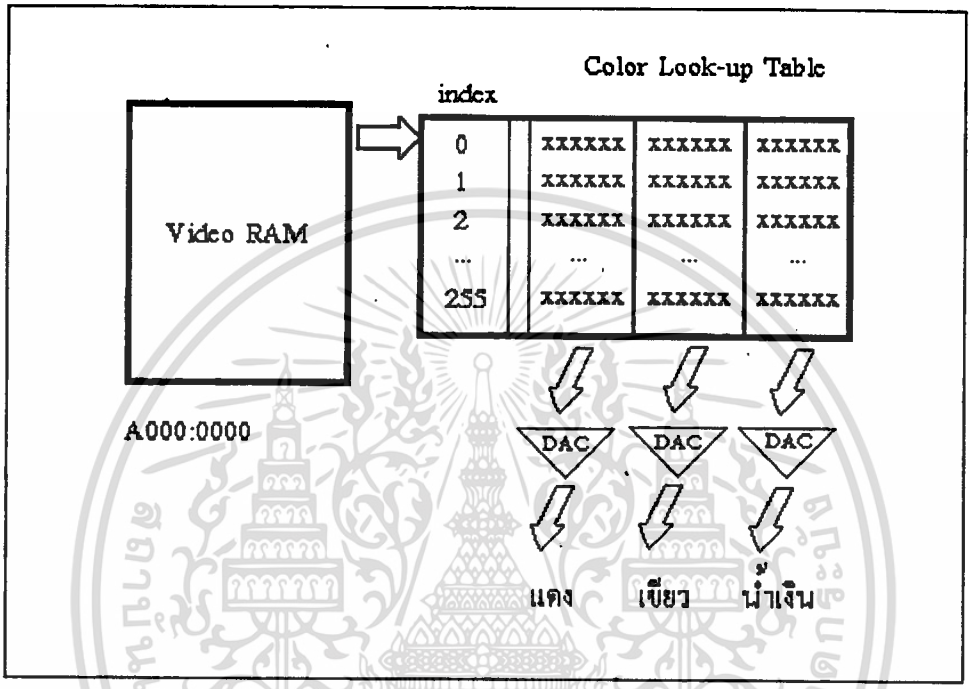
ภาพที่ใช้ในการวิเคราะห์ลายนิ้วมือเป็นภาพ 256 greys ดังนั้นจึงต้องมีการปรับค่าสีให้ตรงกัน และสิ่งที่ต้องระวังคือการเก็บภาพ bmp จะเก็บมุมล่างซ้ายไปขวา และไล่จนถึงบนสุด



รูปที่ 4.4 แสดงการเก็บภาพของ bmp

4.3 การ Set Mode Display

ในโหมดภาพ VGA มีโครงสร้างการจัดการหน่วยความจำและการแสดงสีดังนี้



รูปที่ 4.5 โครงสร้างของการแสดงสีบนจอภาพในโหมด VGA

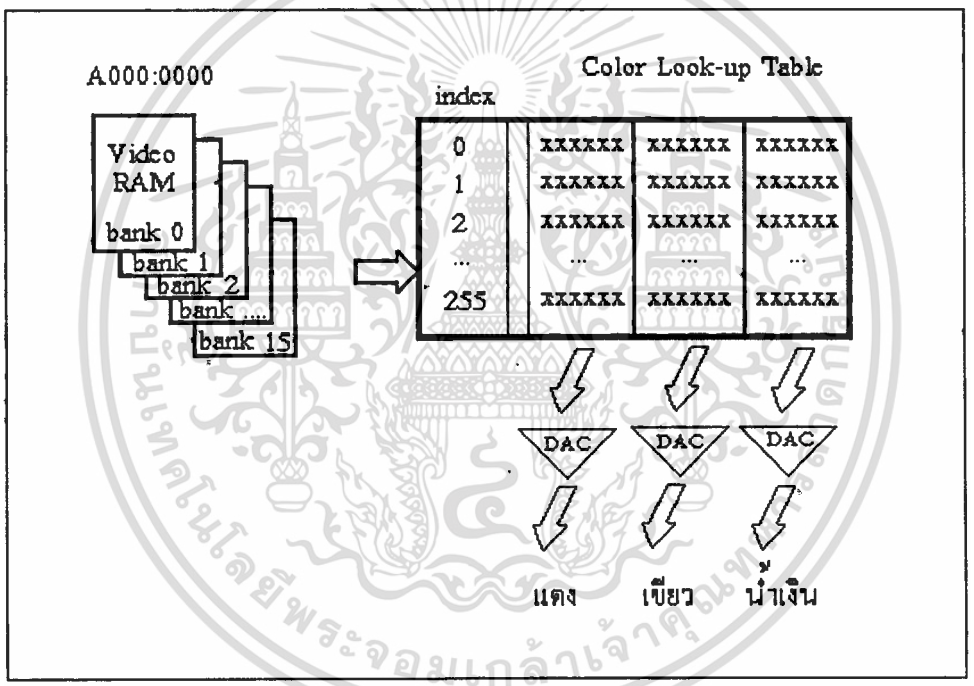
ในวิดีโอแรมบนการ์ดแสดงผล จะเก็บข้อมูลที่ใช้แสดงอยู่ในรูปของค่าหมายเลขแม่สีของตารางเทียบสี (Coor Look - up Table) หน่วยความจำในวิดีโอแรมบนการ์ด VGA มาตราฐานจะมีอยู่ 256 กิโลไบต์ แต่จะใช้ในโหมดนี้เพียง 64 กิโลไบต์ มีจุดเริ่มต้นที่ตำแหน่ง A000:0000 ใช้ข้อมูลหนึ่งไบต์แทนจุดหนึ่งจุด สูตรหาตำแหน่งหน่วยความจำที่ใช้เก็บค่าหมายเลขสีของจุด (X,Y) คือ

ค่าออฟเซตของตำแหน่งจุดภาพ = X + (Y * 320)

ในการแสดงภาพของการ์ดแสดงผล จะใช้ข้อมูลจากวิดีโอแรมมาเปิดตารางเทียบสีซึ่งมีอยู่ 256 เรคคอร์ด ในแต่ละเรคคอร์ดจะประกอบไปด้วยค่าแม่สีแดง สีเขียว และสีน้ำเงิน มีขนาดค่าละ 6 บิต ดังนั้นการ์ดจึงสามารถสร้างสีได้ทั้งหมด 262,144 สี แต่เนื่องจากมีตารางเพียง 256 เรคคอร์ด และในหนึ่งจุดภาพจะใช้พื้นที่เก็บหนึ่งไบต์ จึงทำให้การ์ดรุ่นนี้สามารถแสดงผลได้คราวละ 256 สี

เท่านั้น แม่สีที่ได้จากตารางเทียบสีจะถูกแปลงให้เป็นสัญญาณอนาลอก โดยวงจรแปลงสัญญาณจากดิจิตอลเป็นอนาลอก (Digital to Analog Converter - DAC) เพื่อส่งให้จอภาพแสดงต่อไป

ในการแสดงของการ์ด SVGA มีความซับซ้อนขึ้น โดยซีพียูจะเห็นหน่วยความจำวิดีโอแรม มีหน่วยความจำซ้อนอยู่ในตำแหน่ง A000:0000 ถึง A000:FFFF อยู่ตั้งแต่ 4 แบงค์ (bank) ถึง 16 แบงค์ ในแต่ละแบงค์ของหน่วยความจำจะมีขนาด 64 กิโลไบต์ ซีพียูจะติดต่อกับหน่วยความจำในแบงค์ใดๆ ได้โดยการใส่ค่าแบงค์ที่ต้องการติดต่อลงในรีจิสเตอร์สำหรับเลือกแบงค์บนการ์ดแสดงผล ซึ่งซีพียูจะติดต่อผ่านทางพอร์ตดังนั้น ในกรณีที่มีหน่วยความจำอยู่บนการ์ด 16 แบงค์ นั่นหมายถึง วิดีโอแรมจะมีขนาด 1 เมกะไบต์



รูปที่ 4.6 โครงสร้างของการทำงานของการ์ด SuperVGA

การเข้าถึงตำแหน่งข้อมูลของข้อมูลจอในจุด (X,Y) บนแบงค์ที่กำหนด มีสูตรดังนี้

หมายเลขแบงค์ = (X + (Y * จำนวนจุดที่แสดงในแกน X)) / 0x10000L

ค่าอ็อฟเซตของหน่วยความจำ = (X + (Y * จำนวนจุดที่แสดงในแกน X)) AND 0xffff

โหมดภาพมาตราฐานของจอ SVGA แบ่งได้เป็นสองชุดคือ โหมด 16 สีและโหมด 256 สี ในโหมด 16 สีจะมีหลักการแสดงเช่นเดียวกันกับการแสดงในโหมดของ EGA ส่วนในโหมด 256 สีมีโหมดดังต่อไปนี้คือ 640 คูณ 480 จุด 800 คูณ 600 จุด และ 1024 คูณ 768 จุด

การ์ดแสดงผลอีกประเภทหนึ่งที่ควรรู้จักกันก็คือ ExtendedVGA หรือ XGA การ์ดชนิดนี้มีวิธีแสดงผลที่แตกต่างออกไปจาก VGA และ SVGA ข้อมูลที่อยู่ภายในวิดีโอแรมจะหมายถึงค่าแม่สีโดยตรง สามารถแสดงสีได้สูงถึง 4 พันล้านสี แต่การ์ดแสดงผล SVGA หลายการ์ดในปัจจุบันก็ได้ถูกสร้างให้มีความสามารถแสดงผลได้ใกล้เคียงกัน และใช้กรรมวิธีในการเก็บข้อมูลในรูปแม่สีโดยตรงเช่นกัน การ์ด SVGA ชนิดนี้จะมีโหมดเพิ่มเติมมาอีกสองชนิดคือ Hi-color และ True-color ในโหมด Hi-color ยังมีโหมดย่อยออกไปอีกสองโหมดคือ โหมด 32,768 สี และ 655,36 สี ซึ่งให้ภาพที่ใกล้เคียงธรรมชาติ ส่วนโหมด True-color สามารถแสดงผลได้ถึง 16.7 ล้านสี ภาพที่ได้จึงสมจริงกว่าโหมด Hi-color ทั้งโหมด Hi-color และ True-color ในสองโหมดย่อย มีโหมดการแสดงมาตราฐานดังนี้คือ 320 คูณ 200 จุด 640 คูณ 480 จุด 800 คูณ 600 จุด และ 1024 คูณ 468 จุด

การเข้าถึงตำแหน่งข้อมูลบนวิดีโอแรม ยังคงใช้หลักการเข้าถึงแบบแบงค์เช่นเดียวกับการ์ด SVGA แต่ข้อมูลจะมีขนาด 2 ไบต์ และ 3 ไบต์ ตามลำดับ แต่บางโหมดหรือบางการ์ด ข้อมูลในตำแหน่ง X แรกของแนวแกน Y ถัดไป อาจจะไม่ได้อยู่ต่อท้ายตำแหน่ง X ท้ายสุดของแนว Y แถวก่อนก็ได้ การทำเช่นนี้ก็เพื่อให้การเข้าถึงหน่วยความจำ ใช้คำสั่งที่ง่ายและรวดเร็วยิ่งขึ้น โดยระยะห่างระหว่างจุด X เดียวกันในแนวแกน Y ที่ติดกันจะมีค่าเท่ากับจำนวนไบต์ของเส้นสแกน ซึ่งมักจะมีค่าเป็นจำนวนยกกำลังของสอง เช่น 1024 หรือ 2048 จุดเป็นต้น สูตรการเข้าถึงข้อมูลสีในตำแหน่งข้อมูล และตำแหน่งข้อมูลในตำแหน่งหน่วยความจำมีดังนี้

โครงสร้างข้อมูลสีในตำแหน่งข้อมูลที่ต้องการ

โหมด 32,768 สี

```
struct_32K
{
        unsigned blue       : 5;
        unsigned green      : 5;
        unsigned red        : 5;
        unsigned unused             : 1;
};
```

โหมด 65,536 สี

```
struct_64K
{
        unsigned blue       : 5;
```

```
        unsigned green      : 6;
        unsigned red        : 5;
    };
```

โหมด 16.7 ล้านสี

```
struct_16_7M
{
    unsigned char blue;
    unsigned char green;
    unsigned char red;
};
```

ตำแหน่งหน่วยความจำเริ่มต้นของโครงสร้างข้อมูลสี

หมายเลข bank = ((X*2(หรือ 3)) + (Y*ค่าจำนวนไบต์ของหนึ่งเส้นสแกน) / 0x10000L

ค่าอ็อฟเซตของหน่วยความจำ = ((X*2(หรือ3)) + (Y*ค่าจำนวนไบต์ของหนึ่งเส้นสแกน)) AND 0xffff

ค่า X*2 มาจากค่าขนาดของพื้นที่ที่ใช้เก็บข้อมูลหนึ่งจุดของโหมด 32,768 สีและ 65,536 สี แต่สำหรับโหมด 16.7 ล้านสี ค่านี้จะเป็น X*3

มาตราฐาน VESA เนื่องจากการ์ดแสดงผล SVGA และ XGA ไม่ได้ถูกกำหนดให้เป็นมาตราฐานของอุปกรณ์ในเครื่องคอมพิวเตอร์พีซีเช่นเดียวกับการ์ด VGA ซึ่งถูกกำหนดมาจาก IBM ในขณะที่ผู้ผลิตการ์ดแสดงผลอื่นผลิตการ์ด SVGA และ XGA ขึ้นมา บริษัท IBM ก็ได้ผลิตการ์ดแสดงผลของตนสำหรับเครื่อง PS/2 ซึ่งมีประสิทธิภาพในระดับเดียวกับการ์ด SVGA แต่เนื่องตลาดคอมพิวเตอร์ในยุค PC/XT และ PC/AT ซึ่งผู้ผลิตเครื่องเลียนแบบจะต้องคงมาตราฐานเช่นเดียวกันกับเครื่องของ IBM ผู้ผลิตเครื่องเลียนแบบแต่ละรายจึงกำหนดมาตราฐานของอุปกรณ์บางตัวเช่นการ์ดแสดงผลบนเครื่องของตนขึ้นใหม่ โดยไม่ยึดตาม IBM อีกต่อไป การ์ดแสดงผลบนเครื่องคอมพิวเตอร์ในยุคปัจจุบันจึงมีโครงสร้างทางฮาร์ดแวร์ที่แตกต่างกันออกไปตามชนิดของชิพที่นำมาผลิตหรือตามจำนวนของวิดีโอแรมที่ใส่ในการ์ด ซึ่งมีตั้งแต่ 256 กิโลไบต์ จนถึง 4 เมกะไบต์หรือมากกว่า

ในที่สุด กลุ่มบริษัทผู้ผลิตจอภาพและการ์ดแสดงผล ที่รวมตัวกันเป็นองค์กรที่ เรียกว่า VESA (Video Electronic Standards Association) จึงได้กำหนดมาคราฐานของการ์ดแสดงผล

SVGA ขึ้นในปี 1989 โดยบริษัทในกลุ่มได้ตกลงที่จะจัดทำการ์ดแสดงผลของตน ให้มีความเข้ากันได้กับมาตราฐานที่กำหนดขึ้น

VESA กำหนดมาตราฐานของการ์ดแสดงผลในรูปของ VESA-VGA BIOS เป็นฟังก์ชันเพิ่มเติมในอินเตอร์รัพท์หมายเลข 10h อันเป็นอินเตอร์รัพท์สำหรับการแสดงผลเพิ่มเติมของไมโครคอมพิวเตอร์ โหมดภาพมาตราฐานที่ VESA กำหนดขึ้นที่ควรทราบมีดังนี้คือ

โหมด	ความละเอียด	จำนวนสี	ขนาดวิดีโอแรมที่ต้องการ
100h	640x400	256	256k
101h	640x480	256	512k
102h/6Ah	800x600	16	256k
103h	800x600	256	512k
104h	1024x768	16	512k
105h	1024x768	256	1M
106h	1280x1024	16	1M
107h	1280x1024	256	2M(1.25M)
10dh	320x200	32,767	128k
10eh	320x200	65,536	128k
110h	640x480	32,768	1M
111h	640x480	65,536	1M
112h	640x480	16,777,216	1M
113h	800x600	32,768	1M
114h	800x600	65,536	1M

ตารางที่ 4.4 แสดงโหมดสี

ส่วนของ VESA-VGA BIOS ที่กำหนดขึ้น จะถูกเรียกใช้โดยผ่านอินเตอร์ รัพท์หมายเลข 10h ฟังก์ชัน 4Fh มีฟังก์ชันย่อย 8 ฟังก์ชันแต่มีฟังก์ชันที่น่าสนใจ 4 ฟังก์ชันคือ

ฟังก์ชัน 00h รายงานรายละเอียดของ SVGA

ค่าที่ส่งไป AF = 4Fh

AL = 00h

ES:DI ค่าการชี้ข้อมูลสตรัคเจอร์(ผู้เขียนโปรแกรมกำหนดเอง)

```
struct VGAinfo{
    char          VESAsignature[4];   ค่าคงที่ "VESA"
    char          MajorVersion;       เลขเวอร์ชันของการ์ด
    char          MinorVersion;       ทศนิยมเลขเวอร์ชัน
    void far      *OEMstr;            ชื่อรุ่นของการ์ด
    long          reserved;           สงวนไว้
    unsigned far  *VideoModeList;     หมายเลขโหมดที่มี
    unsigned      Bangkok;            จำนวนแบงค์ที่มี
    char          reserved2[242];     สงวนไว้
};
```

หมายเลขโหมด เป็นอาร์เรย์ของค่าคงที่จำนวนเต็ม ลงท้ายด้วย 0xffff เช่น unsigned mode[] = {0x100,0x101,0x102,0x112,0xffff};

ฟังก์ชัน 01h ตรวจสอบหมายเลขโหมด

ค่าที่ส่งไป AH = 4Fh

AL = 01h

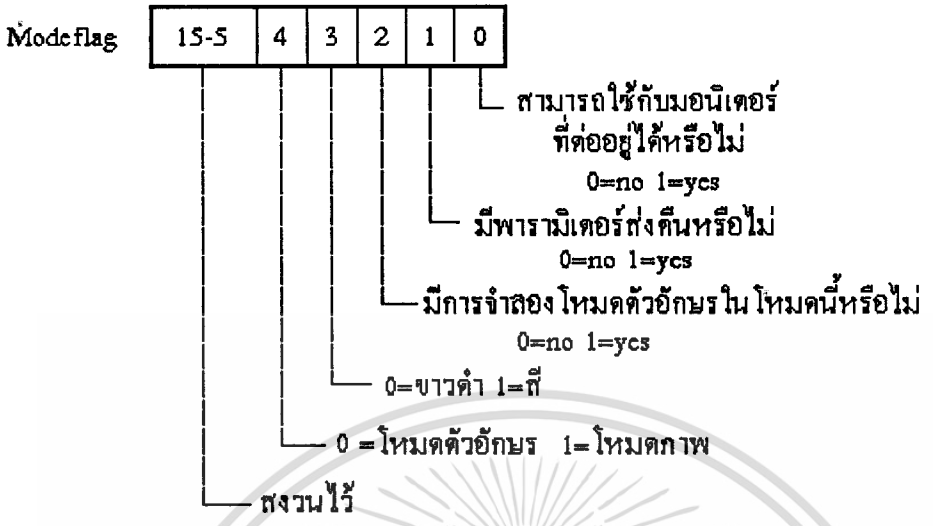
CX = หมายเลขโหมดที่ต้องการตรวจสอบ

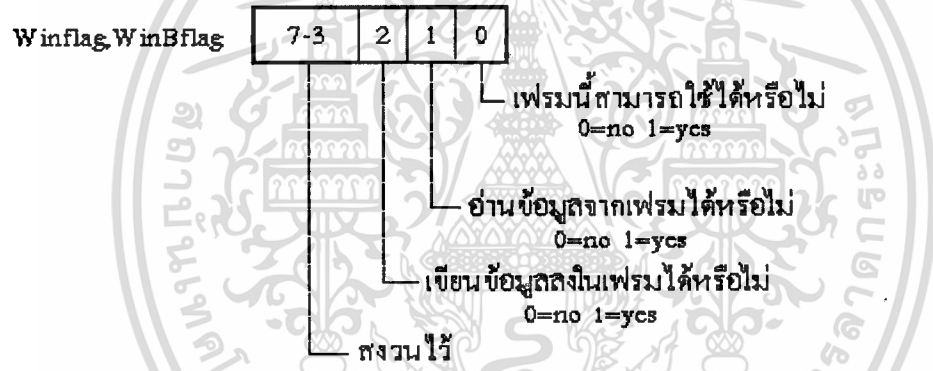
ES:DI ค่าการชี้ข้อมูลสตรัคเจอร์

```
struct MODEinfo{
    unsigned      ModeFlag;           รายละเอียดของโหมด
    char          WinAflag;           รายละเอียดของเฟรม A
    char          WinBflag;           รายละเอียดของเฟรม B
    unsigned      WinGranularity;     ค่าระยะห่างระหว่างข้อมูลแรกในแบงค์กับข้อมูลแรกในแบงค์ถัดไป (มีหน่วยเป็นกิโลไบต์)
```

```
unsigned    Winsize;        ขนาดของข้อมูลที่สามารถถ่ายอย่างต่อ
                            เนื่องได้สูงสุด(หน่วยเป็นกิโลไบต์)
unsigned    WinASegment;    เซกเมนต์ของเฟรม A
unsigned    WinBSegment;    เซคเมนต์ของเฟรม B
void far    (*WinFuncPtr)(); ค่าการชี้ฟังก์ชันซึ่งชี้ไปยังฟังก์ชัน
                            หมายเลข 5 สำหรับใช้เรียกอย่างตัวชี้ฟังก์ชันทั่วไป
unsigned    BPL;            จำนวนไบต์ต่อเส้นสแกน
unsigned    xres;           ความละเอียดในแกน x
unsigned    yres;           ความละเอียดในแกน y
char        Xcharsize;      ความกว้างของฟอนต์ตัวอักษร
char        Ycharsize;      ความกว้างของฟอนต์ตัวอักษร
char        Bitplane;       จำนวนเพลนของหน่วยความจำ (ใน
                            โหมด EGA ทั่วไปจะมีสีเพลน)
char        Bitperpixel;    จำนวนบิตที่ใช้ในการเก็บจุดภาพ
char        Memblock;       จำนวนบัฟเฟอร์สำหรับใช้ในการ
                            แสดงในโหมด EGA
char        Memmodel;       ชนิดของข้อมูลภาพ
char        Blocksize;      ขนาดของบัฟเฟอร์สำหรับใช้ในการ
                            แสดงในโหมด EGA
}VESA_info;
```

หมายเหตุ มาตราฐาน VESA กำหนดให้มีเฟรมรองรับการทำงาน 2 เฟรมคือ A และ B แต่การ์ดอาจจะมีเพียงเฟรมเดียวก็ได้





MemModel มีค่าที่กำหนดดังนี้คือ

00h โหมดตัวอักษร

01h ข้อมูลภาพชนิด CGA 4 สี

02h ข้อมูลภาพชนิด Hercules

03h ข้อมูนมูลภาพชนิด EGA/VGA 16 สี

04h หนึ่งข้อมูนภาพใช้ 4 ไบต์

05h หนึ่งข้อมูลภาพใช้ 8 ไบต์

06-0Fh ไม่ใช้

10h-FFh สงวนไว้สำหรับผู้ผลิต ปกติจะไม่ใช้

ฟังก์ชัน 02h เปลี่ยนโหมดจอภาพ

ค่าที่ส่งไป AH = 4Fh

AL = 02h

BX = หมายเลขโหมดจอภาพ บิต 15 กำหนดว่าจะให้ลบข้อมูลในจอหรือไม่

ฟังก์ชัน 03h สอบถามโหมดปัจจุบัน

ค่าที่ส่งไป AH = 4Fh

AL = 03h

ค่าที่ส่งกลับ BX = หมายเลขโหมดจอภาพ

ฟังก์ชัน 05h, ฟังก์ชันย่อย 00h เปลี่ยน bank ที่ต้องการติดต่อ

ค่าที่ส่งไป AH = 4Fh

AL = 05h

BH = 00h

BL = 0 เฟรม A

= 1 เฟรม B

DX = หมายเลข bank ใหม่

ฟังก์ชัน 05h, ฟังก์ชันย่อย 01h สอบถาม bank ปัจจุบัน

ค่าที่ส่งกลับ AH = 4Fh

AL = 05h

BH = 01h

BL = 0 เฟรม A

= 1 เฟรม B

ค่าที่ส่งกลับ DX = หมายเลข bank ปัจจุบันของเฟรมที่ต้องการทราบ

หมายเหตุ ค่าที่ส่งกลับมาตราฐานของทุกฟังก์ชันและฟังก์ชันย่อยคือ

AL = 4Fh มี VESA-BIOS อยู่

ค่าอื่นๆ ไม่มี VESA-BIOS ฟังก์ชันที่เรียกใช้ไม่มาตราฐาน

AH = 00h ทำงานได้สำเร็จ

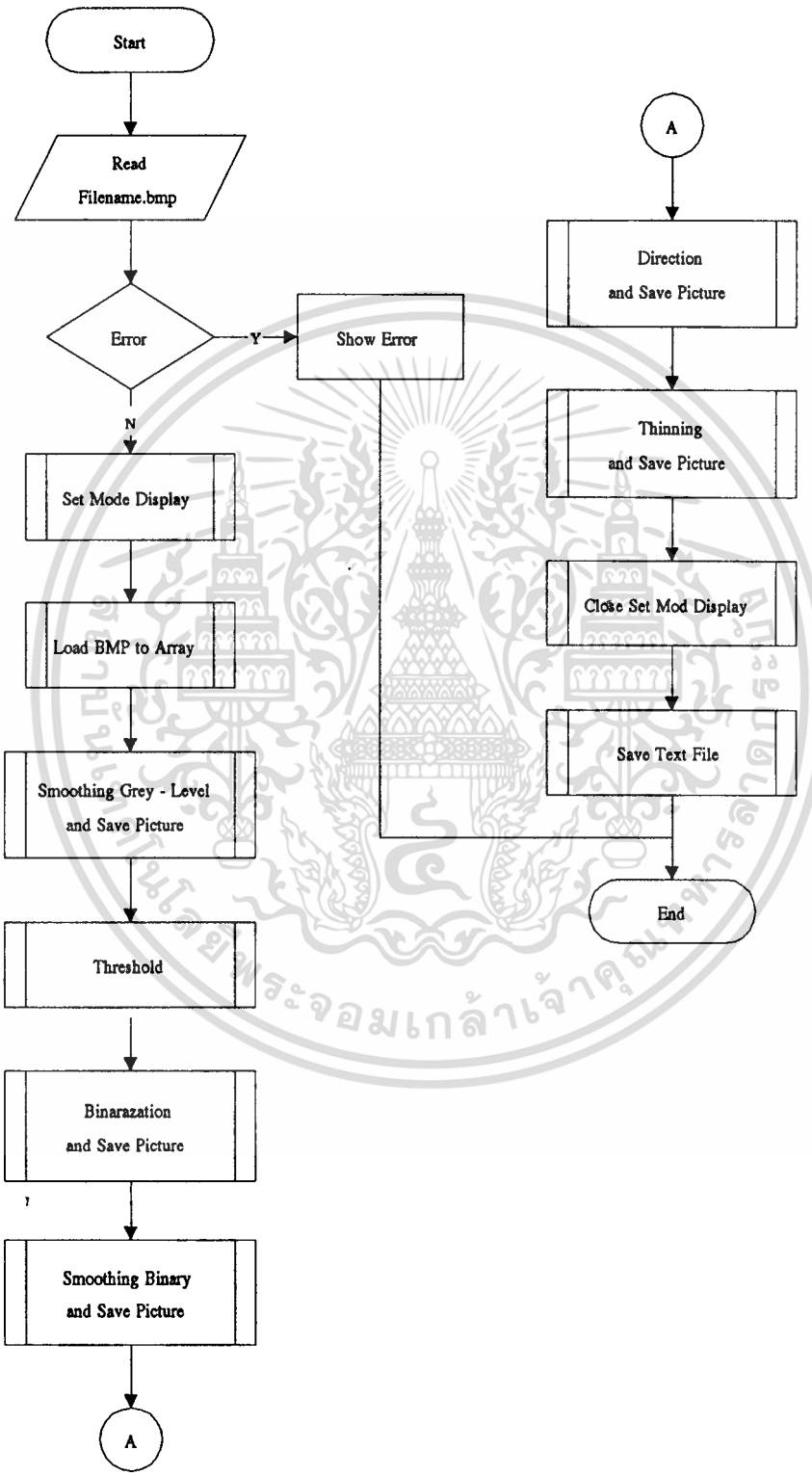
ค่าอื่นๆ มีความผิดพลาดในการทำงาน

ในการใช้งาน VESA-VGA BIOS เราจะต้องใช้ฟังก์ชันหมายเลข 00h เพื่อตรวจสอบว่าการ์ดแสดงผลที่ใช้งานอยู่นี้มี VESA-VESA BIOS อยู่ด้วยหรือไม่ หรือมีการติดตั้ง BIOS เพิ่มเติม(เป็นโปรแกรมเรซิเดนต์) หรือไม่ หากมี ฟังก์ชันจะส่งค่า 4Fh กลับมาทางรีจิสเตอร์ AL และส่งค่า 0 กลับมาทางรีจิสเตอร์ AH

เมื่อตรวจสอบ VESA-VGA BIOS เรียบร้อยแล้ว เราจะต้องตรวจสอบรายละเอียดของโหมดที่เราต้องการติดต่อ โดยการเรียกฟังก์ชัน 01h ข้อมูลต่างๆ ที่ได้มาจะนำไปใช้ในการเข้าถึงการ์ดแสดงผล หลังจากนั้นเราก็สามารถใช้ฟังก์ชันอื่นๆ สำหรับการจัดการการ์ดแสดงผลได้ตามต้องการ

ข้อสังเกตที่น่าสนใจของ VESA-VGA BIOS ก็คือ ฟังก์ชันหมายเลข 05h ที่ใช้ในการสอบถามแบงค์หรือใช้ในการเปลี่ยนแบงค์นั้น นอกจากเราจะอาศัยการอินเตอร์รัพต์เพื่อเรียกใช้ฟังก์ชันดังกล่าวแล้ว เรายังอาจเรียกใช้ในรูปของตัวชี้ฟังก์ชันได้ จากข้อมูลที่ได้ในฟังก์ชัน 01h จะมีค่าการชี้ฟังก์ชันหมายเลข 05h ให้มาด้วย ซึ่งเราอาจนำค่านี้ไปกำหนดให้กับตัวชี้ฟังก์ชัน ไว้เรียกใช้ฟังก์ชัน 05h แทนการร้องขออินเตอร์รัพต์ เพื่อความรวดเร็วในการทำงาน

4.4 การทำงานของ Main Program



รูปที่ 4.5 แสดงการทำงาน Main Program

การทำงานของโปรแกรมจะรับ ชื่อของภาพต่อท้ายจากการเรียกใช้โปรแกรม แต่ถ้าไม่ได้ใช่ชื่อตอนเรียกใช้ จะสามารถใช่ได้อีกครั้ง เมื่อเรียกใช้โปรแกรมแล้วดังนี้

```
c: > Finger _f1.bmp

        or

c : > Finger
  ... Fingerprint Recognition ...
Usage : FINGER  file.bmp

File picture [ filename.bmp ] == _
```

เมื่อโปรแกรมถูกเรียกใช้จะแสดงขั้นตอนการทำงานของ Image processing ขั้นตอนต่างๆ พร้อมทั้งจัดเก็บรูปภาพที่ผ่านขั้นตอนแต่ๆไว้ด้วย และค่า Threshold ต่างๆ

Smg.bmp	ภาพที่ผ่าน Smoothing Grey - Level
Bi.bmp	ภาพที่ผ่าน Binarization
Smb.bmp	ภาพที่ผ่าน Smoothing Binary
Dir.bmp	ภาพที่ผ่าน Direction
Tt.bmp	ภาพที่ผ่าน Thinning
Histogra.txt	ค่าของ Threshold และ Histogram ของ Image
Binary.txt	ภาพ Binary
Directio.txt	ค่า Direction ในแต่ละ subregon
Minutia.txt	ค่า Minutia ของ ภาพ Binary

ในการเรียกใช้โปรแกรมต้องการ หน่วยความจำมากพอสมควร

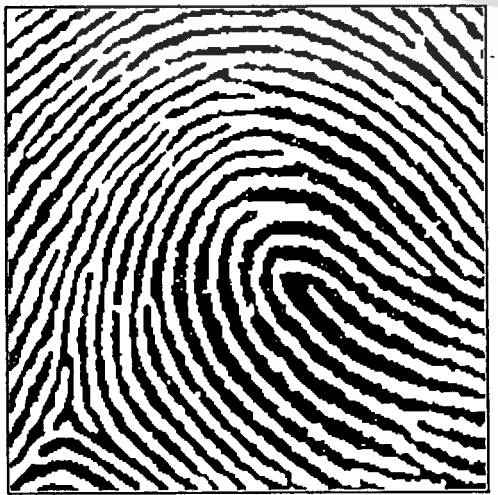
บทที่ 5

ผลการทดลอง

ภาพต้นแบบ F0.bmp

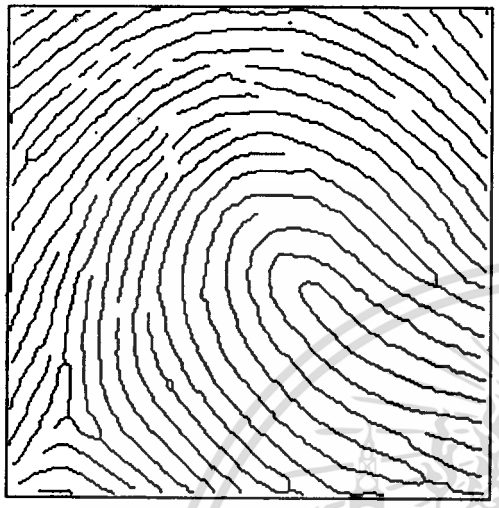
ภาพ Smooth grey - level

ภาพ Binary

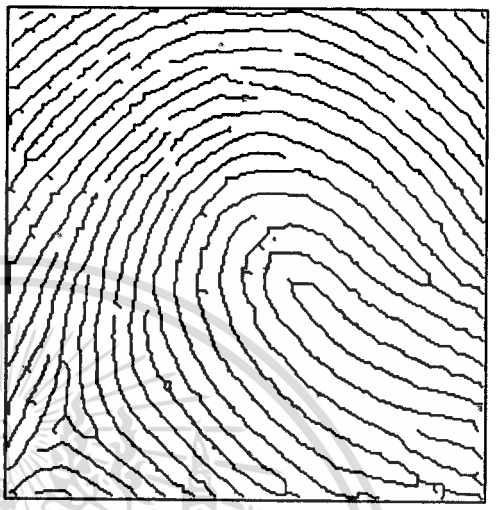


ภาพ Smooth binary

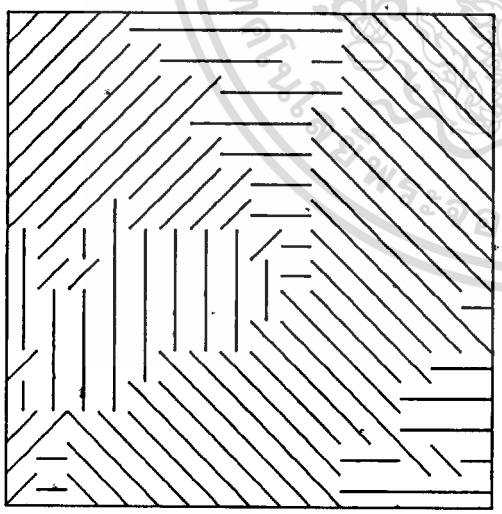
ภาพ Thinning Zhang & Suen Parallel



ภาพ Thinning One-Pass



ภาพ Direction



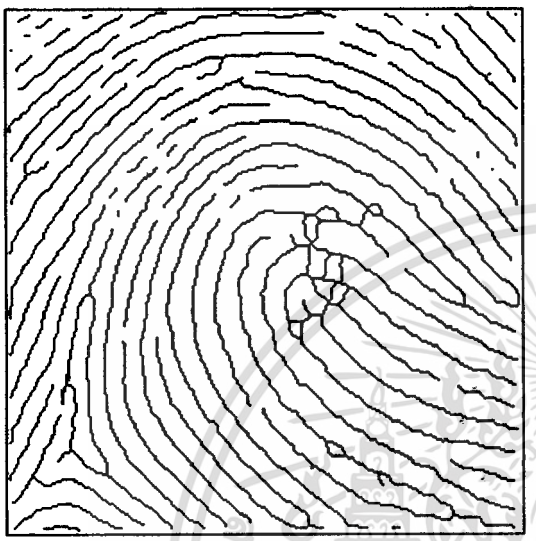
ภาพต้นแบบ F1.BMP

ภาพ Smooth grey - level

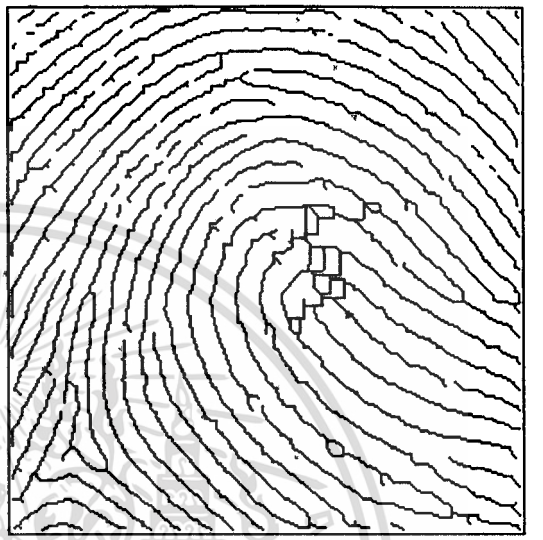
ภาพ Binary

ภาพ Smooth binary

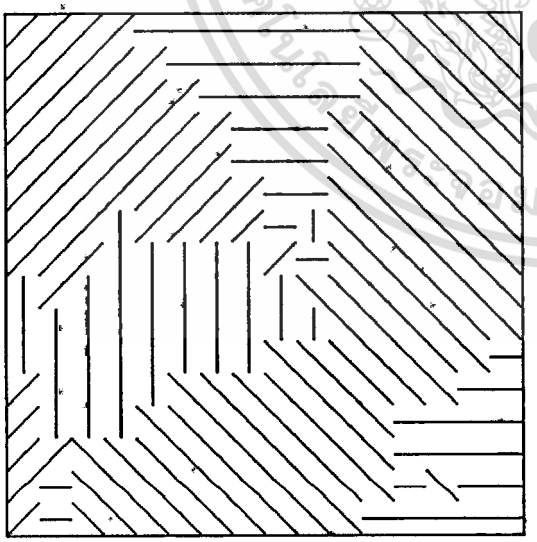
ภาพ Thinning Zhang & Suen Parallel



ภาพ Thinning One-Pass



ภาพ Direction



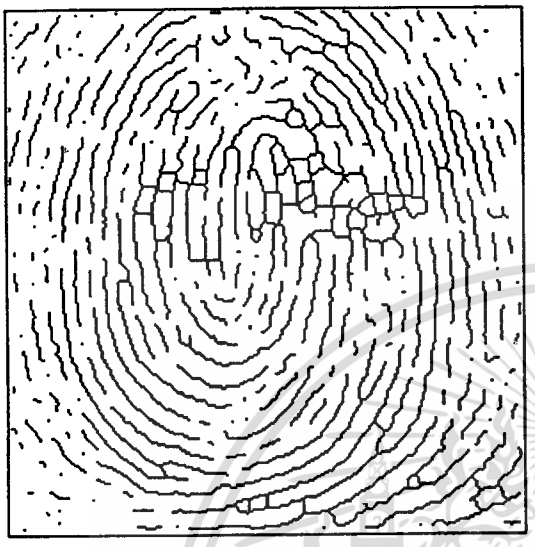
ภาพต้นแบบ JO.BMP

ภาพ Smooth grey - level

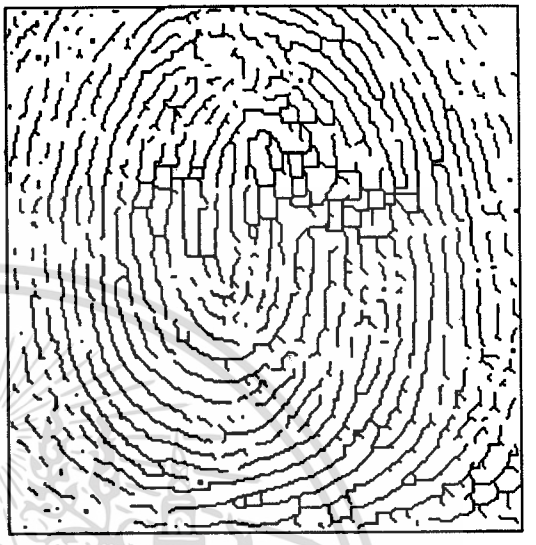
ภาพ Binary

ภาพ Smooth binary

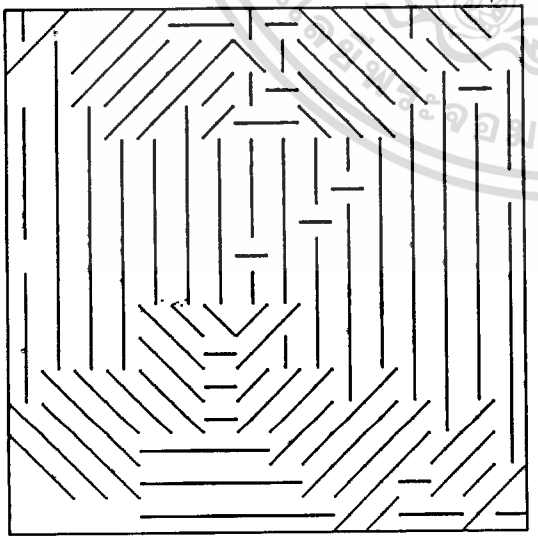
ภาพ Thinning Zhang & Suen



ภาพ Thinning One-Pass Parallel



ภาพ Direction



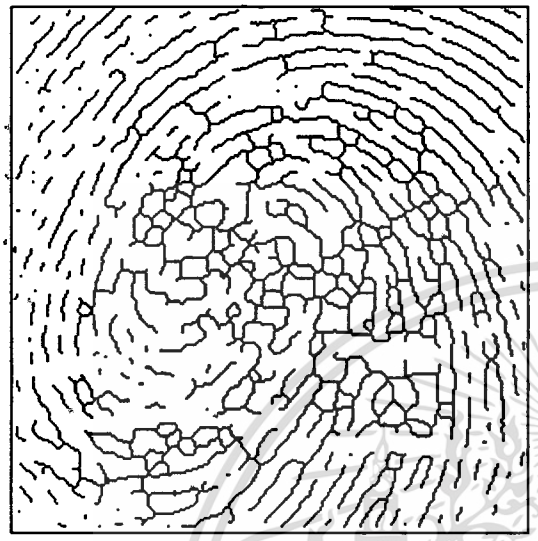
ภาพต้นแบบ NU.BMP

ภาพ Smooth grey - level

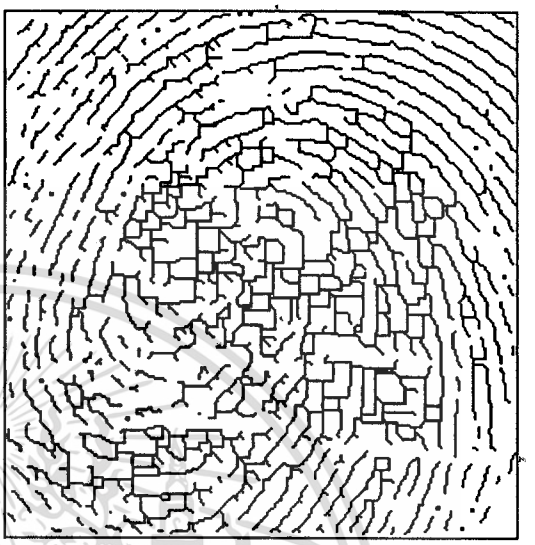
ภาพ Binary

ภาพ Smooth binary

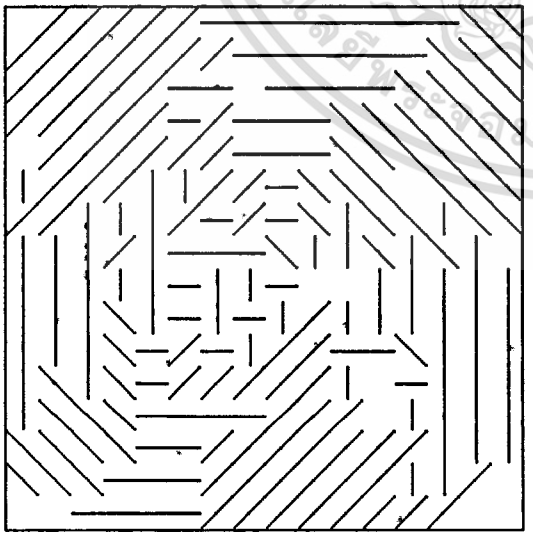
ภาพ Thinning Zhang & Suen



ภาพ Thinning One-Pass Parallel

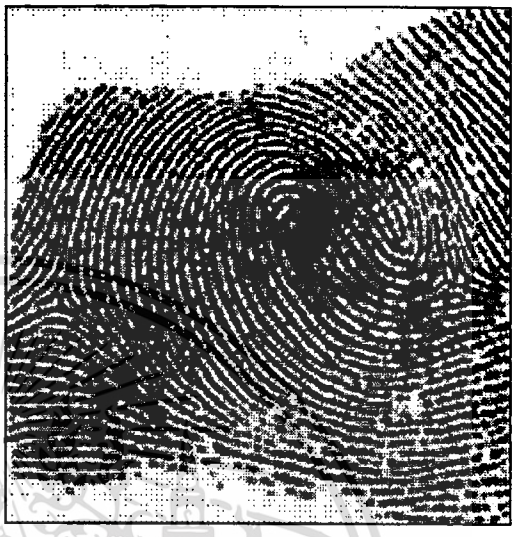


ภาพ Direction



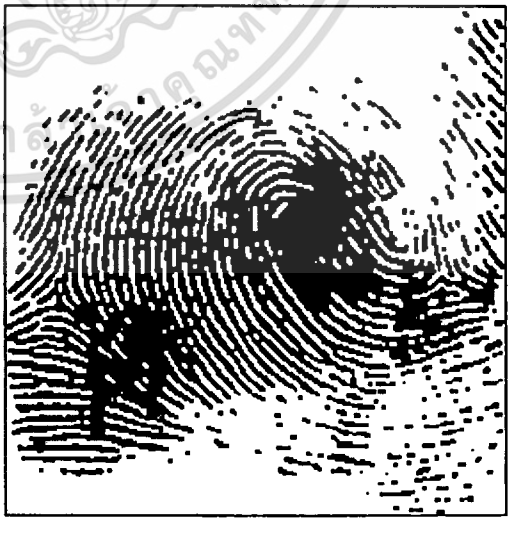
ภาพต้นแบบ PO.BMP

ภาพ Smooth grey - level

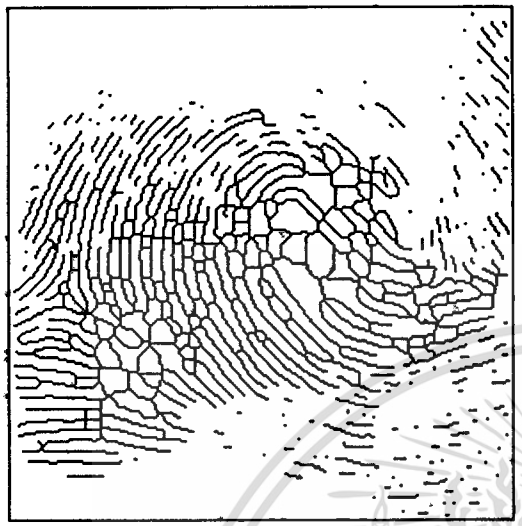


ภาพ Binary

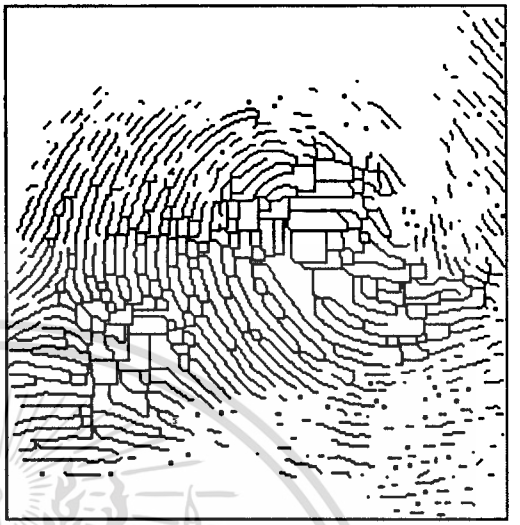
ภาพ Smooth binary



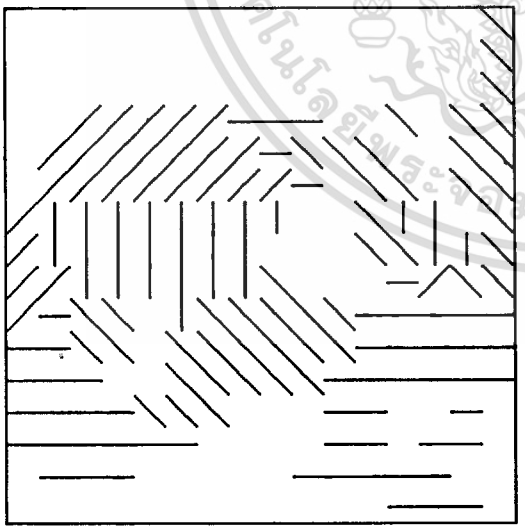
ภาพ Thinning Zhang & Suen



ภาพ Thinning One-Pass Parallel



ภาพ Direction



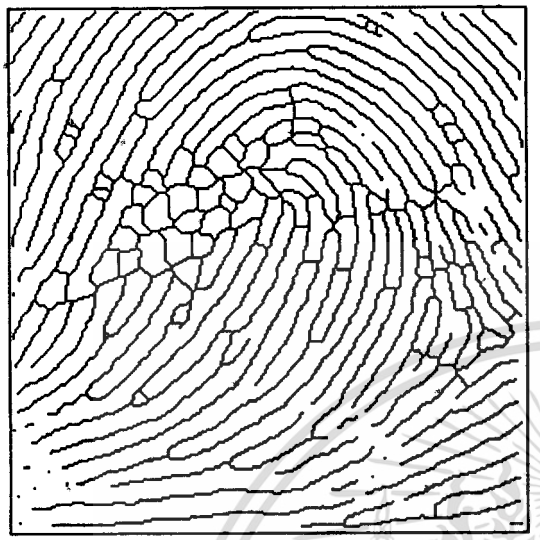
ภาพต้นแบบ RONG.BMP

ภาพ Smooth grey - level

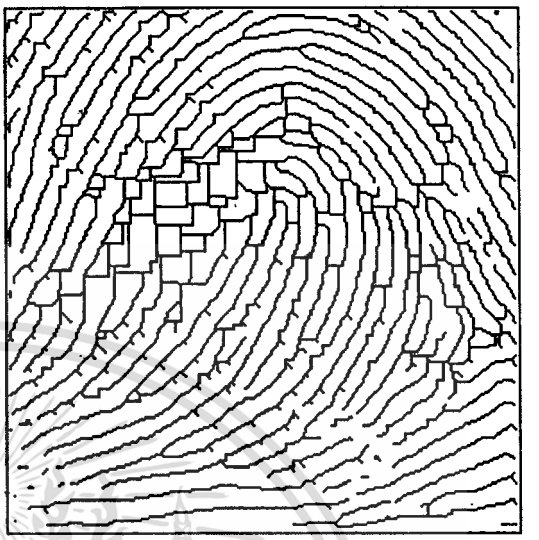
ภาพ Binary

ภาพ Smooth binary

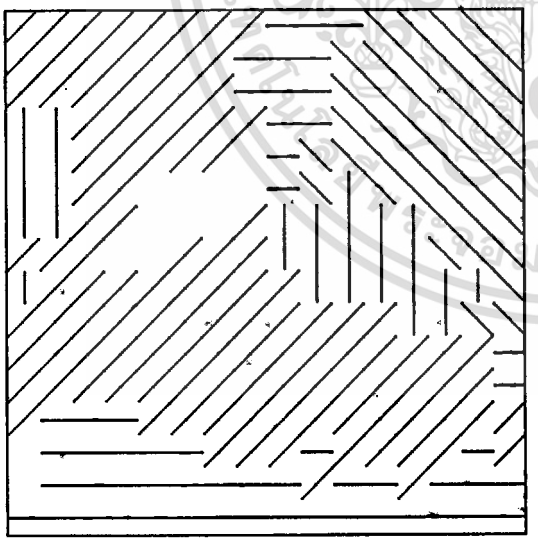
ภาพ Thinning Zhang & Suen



ภาพ Thinning One-Pass Parallel



ภาพ Direction



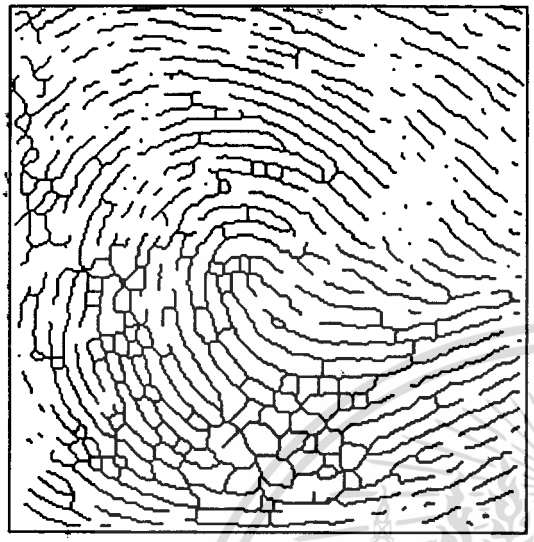
ภาพต้นแบบ TU.BMP

ภาพ Smooth grey - level

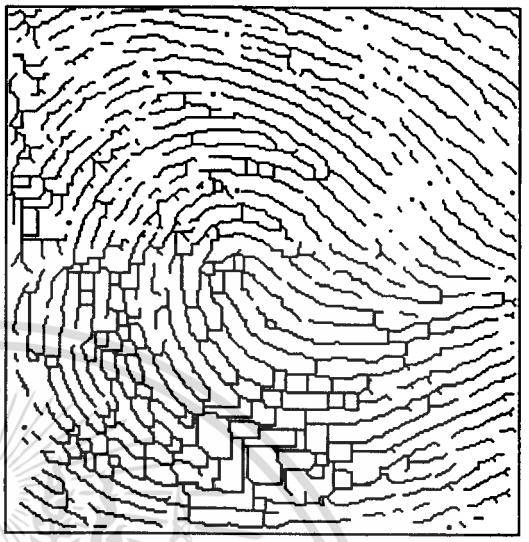
ภาพ Binary

ภาพ Smooth binary

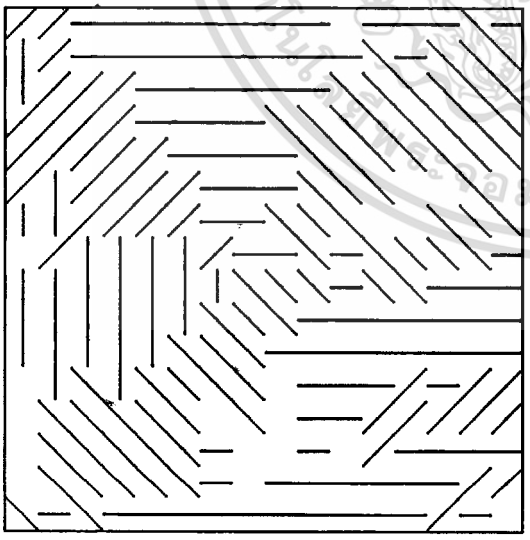
ภาพ Thinning Zhang & Suen



ภาพ Thinning One-Pass Parallel



ภาพ Direction



บทที่ 6

บทวิจารณ์และสรุป

การวิเคราะห์ลายนิ้วมือและจดจำ ครั้งนี้ได้จัดทำขั้นตอนของ Preprocessing ในขั้นตอนต่างๆ ได้แก่ Smoothing Grey - Level, Threshold, Binarization, Smoothing Binary, Direction, Thinning และ Minutia จนได้ผลเป็นที่พอใจ และสามารถนำไปพัฒนาต่อในส่วน Postprocessing และ Recognition ได้

แนวทางในการทำ Postprocessing นั้นไม่ได้นำเสนอพร้อมกับปริญญานิพนธ์ฉบับนี้ อันเนื่องจากเอกสารอ้างอิงของวิธีการทำนั้น หาไม่ได้ในประเทศไทย อีกทั้งเนื้อหาของการทำมีความซับซ้อน แนวทางการค้นหาเอกสาร ถ้าเป็นของ IEEE ตั้งแต่ปี 1988 จนถึงปัจจุบันสามารถค้นหาได้จาก CD-ROM ที่สำนักหอสมุดกลางชั้น 3 วารสาร Pattern Recognition ในปีที่ไม่เก่ามากสามารถค้นหาได้ที่ห้องหนังสืออ้างอิงตึก A ชั้น 3 คณะวิศวะฯ แต่ถ้าเก่ามากแต่ไม่เกินปี 1983 สามารถค้นหาได้ที่ AIT โดยติดต่องานเจ้าหน้าที่ห้องอ้างอิงได้ ส่วนเอกสารจากต่างประเทศนั้นให้ติดต่อที่ TIAC ของกระทรวงวิทยาศาสตร์ หรือจะติดต่อผ่านเจ้าหน้าที่หอสมุดกลางที่ชั้น 3 ก็ได้เช่นกัน การของเอกสารของต่างประเทศต้องใช้เวลานานมาก

ปัญหาอีกประการจะเป็นการเขียนโปรแกรมในโหมด 256 สีเพราะ Driver โหมดนี้ในตัวภาษา C ไม่มีการจัดเตรียมไว้ให้จึงต้องหาจาก Internet ตัว Driver นี้ไม่สามารถใช้ได้กับทุกเครื่องและยังมีปัญหาเรื่องของการใช้ Mouse ด้วยจึงของแนะนำให้ใช้โปรแกรมที่อยู่บน Windows จะลดปัญหาต่างๆได้มาก และสดวกในการทำ User Interface ด้วย

ภาคผนวก

```
//*********************************************************************
//
// Fingerprint Recognition
//
// by :   Thanin Pintong            37013296
//        Somnarin Topparat         37013313
//
// Adisor : Assoc. Prof. Kanchit Maitree (D.Eng)
//
//*********************************************************************

#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <class_1.h>
#include <btree.h>

#define X_RANGE   256l
#define Y_RANGE   256l
#define LEVEL_MAX 256
#define BINARY     2
#define x_org      50
#define y_org      10
#define x_new      350
#define y_new      10

#define P0 image[x][y]              // P1(x-1,y-1)  P2(x,y-1)  P3(x+1,y-1)
#define P1 image[x-1][y-1]
#define P2 image[x][y-1]            // P8(x-1,y)    P0(x,y)    P4(x+1,y)
#define P3 image[x+1][y-1]
```

```
#define P4 image[x+1][y]            // P7(x-1,y+1)  P6(x,y+1)  P5(x+1,y+1)
#define P5 image[x+1][y+1]
#define P6 image[x][y+1]            //  0  x  ( Picture )
#define P7 image[x-1][y+1]          //  y
#define P8 image[x-1][y]

unsigned char huge      grey_image[X_RANGE][Y_RANGE];
unsigned char huge    binary_image[X_RANGE][Y_RANGE];
unsigned char huge   minutia_image[X_RANGE][Y_RANGE];
unsigned char huge   tbinary_image[X_RANGE][Y_RANGE];
unsigned char huge   dbinary_image[X_RANGE][Y_RANGE];
unsigned char huge original_image[X_RANGE][Y_RANGE];
unsigned char huge      temp_image[X_RANGE][Y_RANGE];

#include "vesamode.cpp"
#include "bmp.cpp"
#include "histogra.cpp"
#include "smooth.cpp"
#include "threshol.cpp"
#include "binary.cpp"
#include "thin_zh.cpp"
#include "post.cpp"
#include "single.cpp"
#include "save.cpp"

void main(int argc,char *args[])
{
  char st[50];
  int threshold;
  const int L=LEVEL_MAX;
  unsigned int _histogram[LEVEL_MAX];
  struct Dir dir_image[16][16];

  if (argc!=2)
```

```
 {
   printf("\n ... Fingerprint Recognition  ...");
   printf("\n Usage: FINGER file.bmp");
   printf("\n\nFile picture [filename.bmp] == ");
   scanf("%s",&st);
 }
else
 {
   strcpy(st,args[1]);
 }

switch(open_bmp(st))                    // Keep bmp file and check
  {
    case 1: printf("File not Found !\n");
            exit(0);

    case 2: printf("Not BMP File !\n");
            exit(0);

    case 3: printf("Not 256 gray level !\n");
            exit(0);
  }

opengraph(0x101);                       // Open graph(svga)
load_bmp();
gray_clrscr();                          // clear screen in svga 256 gray scale
show_256_level(x_org,y_org,original_image);

//******** Smooth Grey - Level
 smooth_gray_image(original_image,grey_image);
 show_256_level(x_new,y_new,grey_image);
 save_bmp(st,0,grey_image);

//******** Threshold
```

```
  read_frequency(grey_image,_histogram,L);
  threshold = thresholding(_histogram,L);

//******** Binarization
  Binarization(grey_image,binary_image,threshold);
  show_binary_image(x_new,y_new,binary_image);
  save_bmp(st,1,binary_image);

//******** Smooth Binary
  smooth_binary_image(binary_image);
  save_bmp(st,2,binary_image);

  copy(binary_image,tbinary_image);
  copy(binary_image,dbinary_image);

//******** Direction
  show_binary_image(x_org,y_org,dbinary_image);          // direction
  box_(x_org,y_org,16);

  direction(dbinary_image,dir_image);

  show_binary_image(x_new,y_new,dbinary_image);
  box_(x_new,y_new,16);

  save_bmp(st,3,dbinary_image);

  rectangle(10,400,60,450,0);
  getch();
  rectangle(10,400,60,450,255);

//******** Thinning
  show_binary_image(x_org,y_org,tbinary_image);          // thinning
  show_binary_image(x_new,y_new,tbinary_image);
```

```
  thin_zhang(tbinary_image);
  show_binary_image(x_new,y_new,tbinary_image);

  save_bmp(st,4,tbinary_image);

//******** Topology
  minutia(tbinary_image,minutia_image);

  rectangle(10,400,60,450,0);
  getch();
  rectangle(10,400,60,450,255);

  closegraph();                          // close graphics

  save_txt(threshold,_histogram,dir_image);          // text file

}//end Main Program


//*********************************************************************
//
//  VESAMODE.CPP
//
//  Vesa Mode : Set display is to 256 color
//
//*********************************************************************

struct MODEinfo{
        unsigned        ModeFlag;
        char            WinAflag;
        char            WinBflag;
        unsigned        WinGranularity;
        unsigned        Winsize;
        unsigned        WinASegment;
```

```
        unsigned        WinBsegment;
        void far        (*WinFuncPtr)();
        unsigned        BPL;
        unsigned        xres;
        unsigned        yres;
        char            Xcharsize;
        char            Ycharsize;
        char            Bitplane;
        char            Bitperpixel;
        char            Memblock;
        char            Memmodel;
        char            Blocksize;
        }VESA_info;

struct VGAinfo{
        char            VESAsignature[4];
        char            MajorVersion;
        char            MinorVersion;
        void far        *OEMstr;
        long            reserved;
        unsigned far    *VideoModeList;
        unsigned        Banktotal;
        char            reserved2[242];
        }VESA;

int OLD_mode;
int OLD_bank = 0;
long LinearVideoRAM;
int VESA_mode;
int VESA_xline;
int VESA_page;
int VESA_frameread;
int VESA_bank;
int VESA_size;
```

```
#define VESA_framewrite 0

#define OK 1
#define ERR 0

int opengraph(int mode);
void closegraph(void);

int getbitperpix(void);
int getxres(void);
int getyres(void);
int getxline(void);
int getbanktotal(void);
int getbanksize(void);
int getbankoff(void);
int getpixel(int x,int y);
int getmaxx(void);
int getmaxy(void);

void setbankread(int bank);
void setbankwrite(int bank);
void putpixel(int x,int y,int color);
void setdac(int index,int R,int G,int B);

#define directpixel(x,y,R,G,B) (*Directpixel)(x,y,R,G,B)
void direct32k(int x,int y,int R,int G,int B);
void direct64k(int x,int y,int R,int G,int B);
void direct16m(int x,int y,int R,int G,int B);
void (*Directpixel)(int,int,int,int,int);          // pointer-to-function, write RGB data to VideoRAM
void far (*setbank)(void);                         // pointer-to-function -> far-call bank routine
void line(int x1,int y1,int x2,int y2,int index);
void rectangle(int left,int top,int right,int bottom,int color);
```

```
//*********************************************************************
int opengraph(int mode)
{
  /* store old mode int OLD_mode */
  OLD_mode = peekb(0,0x0449);

  /* check VESA compatible */
  _ES = FP_SEG((void far *)&VESA);
  _DI = FP_OFF((void far *)&VESA);
  _AX = 0x4f00;

  geninterrupt(0x10);
  if(_AX != 0x4f) return ERR;                    // VESA not supported
  _ES = FP_SEG((void far *)&VESA_info);
  _DI = FP_OFF((void far *)&VESA_info);
  _CX = mode;
  _AX = 0x4f01;

  geninterrupt(0x10);
  if(_AX != 0x4f) return ERR;                    // mode not supported
  /* initial data */
  VESA_mode = mode;
  VESA_xline = VESA_info.BPL;
  VESA_page = VESA_info.WinGranularity;

  /* set shift value for putpixel */
  switch(VESA_info.WinGranularity)
   {
     case 1 : VESA_bank = 10;
              VESA_size = 0x03ff; break;

     case 2 : VESA_bank = 11;
              VESA_size = 0x07ff; break;
```

```
    case 4 : VESA_bank = 12;
             VESA_size = 0x0fff; break;

    case 8 : VESA_bank = 13;
             VESA_size = 0x1fff; break;

    case 16: VESA_bank = 14;
             VESA_size = 0x3fff; break;

    case 32: VESA_bank = 15;
             VESA_size = 0x7fff; break;

    case 64: VESA_bank = 16;
             VESA_size = 0xffff; break;

    default: return ERR;
  }

/* adjust to standard for some card */
 switch(VESA_mode)
  {
    case 0x110:
    case 0x113:VESA_info.Bitperpixel = 15;
  }

 /* set pointer-to-function */
 switch(VESA_info.Bitperpixel)
  {
    case 24:Directpixel = direct16m; break;

    case 16:Directpixel = direct64k; break;

    case 15:
```

```
    default:Directpixel = direct32k;
   }

  setbank = VESA_info.WinFuncPtr;

  /* set frame number for read */
  if(VESA_info.WinAflag&2)
    VESA_frameread = 0;
  else
    VESA_frameread = 1;

  /* open VESA mode */
  _BX = mode;
  _AX = 0x4f02;

  geninterrupt(0x10);
  if(_AX != 0x4f) return ERR;

  /* can't open video mode */
  setbankwrite(0);
  return OK;
}

void closegraph(void)
{
  _AL = OLD_mode;
  /* restore old mode */
  _AH = 0;
  geninterrupt(0x10);
}

int getbitperpix(void)
{
  return (int)VESA_info.Bitperpixel;
```

```
}

int getxres(void)
{
  return VESA_info.xres;
}



int getyres(void)
{
  return VESA_info.yres;
}

int getxline(void)
{
  return VESA_info.BPL;
}

int getbanktotal(void)
{
  return VESA.Banktotal;
}

int getbanksize(void)
{
  return VESA_info.Winsize;
}

int getbankoff(void)
{
  return VESA_info.WinGranularity;
}
```

```
void setbankread(int bank)
{
  _DX = bank;
  _BL = VESA_frameread;
  _BH = 0;
  (*setbank)();
}


void setbankwrite(int bank)
{
  _DX = bank;
  _BL = VESA_framewrite;
  _BH = 0;
  (*setbank)();
}

void setdac(int index,int R,int G,int B)
{
  /* set index */
  outportb(0x03c8,(unsigned char)index);

  /* set 18 bit DAC */
  outportb(0x03c9,(unsigned char)R);
  outportb(0x03c9,(unsigned char)G);
  outportb(0x03c9,(unsigned char)B);
}

void putpixel(int x,int y,int color)
{
  if(OLD_bank != ((LinearVideoRAM = (long)y*(long)VESA_xline+
     (long)x)>>VESA_bank)) setbankwrite(OLD_bank = LinearVideoRAM>>VESA_bank);
  pokeb(VESA_info.WinASegment,LinearVideoRAM&VESA_size,(char)color);
}
```

```
void direct32k(int x,int y,int R,int G,int B)
{
 if(OLD_bank != ((LinearVideoRAM = (long)y*(long)VESA_xline+((long)x<<1))
    >>VESA_bank)) setbankwrite(OLD_bank = LinearVideoRAM>>VESA_bank);
 poke(VESA_info.WinASegment,LinearVideoRAM&VESA_size,
     ((R&0xf8)<<7)|((G&0xf8)<<2)|(B>>3));
}

void direct64k(int x,int y,int R,int G,int B)
{
 if(OLD_bank != ((LinearVideoRAM = (long)y*(long)VESA_xline+((long)x<<1))
    >>VESA_bank)) setbankwrite(OLD_bank = LinearVideoRAM>>VESA_bank);
 poke(VESA_info.WinASegment,LinearVideoRAM&VESA_size,
     ((R&0xf8)<<8)|((G&0xfc)<<3)|(B>>3));
}

void direct16m(int x,int y,int R,int G,int B)
{
 if(OLD_bank !=
    ((LinearVideoRAM = (long)y*(long)VESA_xline+((long)x*3))>>VESA_bank)
    ) setbankwrite(OLD_bank = LinearVideoRAM>>VESA_bank);
 pokeb(VESA_info.WinASegment,LinearVideoRAM&VESA_size,(char)B);
 pokeb(VESA_info.WinASegment,(LinearVideoRAM&VESA_size)+1,(char)G);
 pokeb(VESA_info.WinASegment,(LinearVideoRAM&VESA_size)+2,(char)R);
}

int getmaxx(void)
{
 return (getxres() - 1);
}

int getmaxy(void)
{
```

```
  return (getyres() - 1);
}

/* Draw line in Svga */
void line(int x1,int y1,int x2,int y2,int index)
{
  int a;
  int ydif = y2-y1;
  int xdif = x2-x1;
  if(abs(ydif) > abs(xdif))
    {
      if (ydif == 0) return;
      if (y2 > y1)
      for(a = y1;a<=y2;a++)
            putpixel(x1+(int)((((long)xdif*(long)(a-y1)))/ydif),a,index);
      else
      for(a = y1;a>=y2;a--)
            putpixel(x1+(int)((((long)xdif*(long)(a-y1)))/ydif),a,index);
    }
  else
    {
      if (xdif == 0) return;
      if (x2 > x1)
            for(a = x1;a<=x2;a++)
              putpixel(a,y1+(int)( ( (long)ydif*(long)(a-x1) )/xdif),index);
      else
            for(a = x1;a>=x2;a--)
              putpixel(a,y1+(int)( ( (long)ydif*(long)(a-x1) )/xdif),index);
    }
}

int getpixel(int x,int y)
{
  if(OLD_bank != ((LinearVideoRAM = (long)y*(long)VESA_xline+
```

```
    (long)x)>>VESA_bank)) setbankread(OLD_bank = LinearVideoRAM>>VESA_bank);
  return (int)(unsigned char) peekb(VESA_info.WinASegment,LinearVideoRAM&VESA_size);
}

void rectangle(int left,int top,int right,int bottom,int color)
{
  int a,b,i;
  a = right - left;
  b = bottom - top;

  for(i=0;i<=a;i++)
    putpixel(left+i,top,color);
for(i=0,i>=a,i++)
  for(i=0;i<=b;i++)
    putpixel(right,top+i,color);
  for(i=0;i<=a;i++)
    putpixel(left+i,bottom,color);
  for(i=0;i<=b;i++)
    putpixel(left,top+i,color);
}


//*********************************************************************
//
// BMP.CPP
//
// Picture BMP : Open picture and store to arrary[][]
//
//*********************************************************************

struct BMPHEAD
{
 char id[2];                    // file type : must be ASCII text "BM"
 long filesize;                 // Size of teh file : in double words (32 bit integers)
```

```
int  reserved[2];          // Reserved for future use : must be zero
long headersize;           // byte offser to.bitmap data : offset from the Bitmap file header (i.e,the startof
                           // the file)
long infosize;             // number of bytes in header : currenty 40 bytes
long width;                // width of bitmap : in pixels
long depth;                // height of bitmap : in pixels
int  blplanes;             // number of color planese : must be set to 1
int  bits;                 // number of bits per pixel : valid choices are 1,4,8,24;if not 24,determines the
                           // size of the palette
long blcompression;        // type of compression : {0:no compression}{1:run legth 8 bits per pixel}{2:run
                           // legth 4 bits per pixel}
long bisizeimage;          // size of image : in byte
long bixpelspermeter;      // horizontal resolution : in pixels/meter
long biypelspermeter;      // vertical   resolution : in pixels/meter
long biclrused;            // number of color indexes used by the bitmap : zero indicates all color
                           // important
long biclrimportant;       // number of color indexes important for displaying bitmap : a value of size
                           // indicates all colors are important
}bmphead;

FILE *fbmp;

int  open_bmp(char *st);
void gray_clrscr(void);
void load_bmp(void);
void show_256_level(int x,int y,unsigned char huge image[][Y_RANGE]);
void copy(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE]);

//*****************************************************************
int open_bmp(char *st)
//
// input : file name of bmp
// output : result of open picture is true or error
//
```

```
{
  if((fbmp=fopen(st,"rb"))==NULL)
    return 1;

  fseek(fbmp,0,SEEK_SET);
  fread(&bmphead,sizeof(bmphead),1,fbmp);
  if(memcmp(bmphead.id,"BM",2)!=0)          /* Check extention whether is "bmp" or not */
    {
      fclose(fbmp);
      return 2;
    }

  if(bmphead.bits!=8)
    {
      fclose(fbmp);
      return 3;
    }
  return 0;
}//end sub open_bmp

void load_bmp(void)
//
// input : NULL
// output : picture of original image & set palette of 256 grey - level
//
{
  int x,y,i=0;
  int r,g,b;

  for (x=0;x<256;x++)
    for (y=0;y<256;y++)
      original_image[x][y]=255;

  fseek(fbmp,sizeof(bmphead),SEEK_SET);          // Set Palette of Picture
```

```
while (ftell(fbmp)<bmphead.headersize)          // Keep palette of 255 in r_255,g_255,b_255
 {
   b=fgetc(fbmp)>>2;
   g=fgetc(fbmp)>>2;
   r=fgetc(fbmp)>>2;
   fgetc(fbmp);
   setdac(i,r,g,b);
   i++;
 }

fseek(fbmp,bmphead.headersize,SEEK_SET);

long width = ((bmphead.width+7)/8)*8;

for(y=0;y<bmphead.depth;y++)
  for(x=0;x<width;x++)
    {
        original_image[x][255-y]=fgetc(fbmp);
    }

fclose(fbmp);

}//end sub load_bmp

void gray_clrscr(void)
//
//  input : NULL
// output : screen
//
{
  int a = getmaxy();
  int b = getmaxx();

  for(int y=0;y<=a;y++)
```

```
    for(int x=0;x<=b;x++)
      putpixel(x,y,255);
}

void show_256_level(int x,int y,unsigned char huge image[][Y_RANGE])
//
//  input : position x,y of screen and image
// output : screen
//
{
  for(int _y=0;_y<bmphead.depth;_y++)
   for(int _x=0;_x<bmphead.width;_x++)
     putpixel(x+_x,y+_y,(int)image[_x][_y]);

  line(x,y,x+256,y,180);
  line(x,y,x,y+256,180);

  line(x,y+256,x+256,y+256,180);
  line(x+256,y,x+256,y+256,180);

}//end sub show_256_level

void copy(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE])
//
//  input : image, image0
// output : image => image0
//
{
  for (int y=0; y<256 ; y++)
    for (int x=0; x<256 ; x++)
      image0[x][y] = image[x][y];

}//end sub copy
```

```
//*********************************************************************
//
// Histogra.cpp
//
// Histogram : Show histogram and read frequncy from image[][]
//
//*********************************************************************

void clear_histogram(int xpos,int ypos,const int L);
void read_frequency(unsigned char huge image[][Y_RANGE],unsigned int *h,const int L);
void read_frequency_binary(unsigned char huge image[][Y_RANGE],unsigned int  *h_bin);
void histogram(int xpos,int ypos,unsigned int *h,const int L);
void histogram_binary(int xpos,int ypos,unsigned int *h_bin,const int L);
//*********************************************************************
void read_frequency(unsigned char huge image[][Y_RANGE],unsigned int *h,const int L)
//
//  input : grey image, maximum level
// output : frequency of level
//
{

  for(int i=0;i<(int)L;++i)              // Initial array of gray level
    *(h+i) = 0;

  for(int y=0;y<bmphead.depth;++y)
    for(int x=0;x<bmphead.width;++x)
      *(h+image[x][y]) = *(h+image[x][y]) + (unsigned int)1;

}//end sub read_frequency

void histogram(int xpos,int ypos,unsigned int *h,const int L)
//
```

```
// input : position x,y of screen, frequency of level
// output : screen
//
{
  unsigned int depth=150,space=40,basex,basey;
  double dif;
  unsigned int max;

  max = *h;                                    // Find maximum value
  for(int i=1;i<L;++i)
    if(max < *(h+i)) max = *(h+i);

 dif = (double) max / depth;
  basey = ypos + depth + space - 1;
  basex = xpos + space;

  // Plot axis of histogram
  line(basex,basey,basex,basey-depth+1,200);
  line(xpos+space,ypos+depth+space-1,xpos+L+space-1,ypos+depth+space-1,0);

  for(i=0;i<L;++i)
    {
      line(basex+i,basey,basex+i,basey-(unsigned int)((*(h+i))/dif),0);
    }
}//end sub histogram

void clear_histogram(int xpos,int ypos,const int L)
//
// input : position x,y of screen, maximum level
// output : screen
//
{
  unsigned int depth=150,space=40;
```

```
  for(int y=ypos;y<(ypos+depth+2*space);++y)
    for(int x=xpos;x<(xpos+L+2*space);++x)
      putpixel(x,y,255);

}//end sub clear_histogram

void read_frequency_binary(unsigned char huge image [][Y_RANGE],unsigned int      *h_bin)
//
// input : binary image
// output : frequency of level
//
{
  for (int x=0;x<256;x++)
    *(h_bin+x)=0;

  for( x=0;x<bmphead.width;x++)
    for(int y=0;y<bmphead.depth;y++)
      *(h_bin+(unsigned int)image[x][y]) += (unsigned int)1;

}//end sub read_frequency_binary

void histogram_binary(int xpos,int ypos,unsigned int *h_bin,const int L)
//
// input : position x,y of screen, frequency of level
// output : screen
//
{
  unsigned int depth=150,space=40,basex,basey;
  double dif;
  unsigned int max;

  // Find maximum value
  max = *h_bin;
  if(max < *(h_bin+1)) max = *(h_bin+1);
```

```
dif = (double) max / depth;
basey = ypos + depth + space - 1;
basex = xpos + space;

// Plot axis of histogram
line(basex,basey,basex,basey-depth+1,200);
line(xpos+space,ypos+depth+space-1,xpos+L+space-1,ypos+depth+space-1,0);
line(basex,basey,basex,basey-(unsigned int)((*(h_bin))/dif),0);
line(basex+255,basey,basex+255,basey-(unsigned int)((*(h_bin+1))/dif),0);

}//end sub histogram_binary
```

```
//*********************************************************************
//
// Smooth.cpp
//
// Smooth : Smooth Grey - Level image
//          Smooth Binary image
//
//*********************************************************************

int T(int y);
int B(int y);
int R(int x);
int L(int x);
int X(int i,int x,int y,unsigned char huge image[][Y_RANGE]);
unsigned char X_bin(int i,int x,int y,unsigned char huge image[][Y_RANGE]);
```

```
void smooth_gray_image(unsigned char huge image[][Y_RANGE],unsigned char huge
                       image0 [] [Y_RANGE]);
void smooth_binary_image(unsigned char huge image[][Y_RANGE]);

//*********************************************************************
int T(int y)
{
  if (y<0)
    return (int)1;
  else
    return (int)0;
}

int B(int y)
{
  if (y>(bmphead.depth-1))
    return (int)1;
  else
    return (int)0;
}
int R(int x)
{
  if (x>(bmphead.width-1))
    return (int)1;
  else return
    (int)0;
}

int L(int x)
{
  if (x<0)
    return (int)1;
  else
    return (int)0;
```

```
}

int X(int i,int x,int y,unsigned char huge image[][Y_RANGE])
{
//          4  3  2
// X(i) =   5  0  1
//          6  7  8

  switch(i)
   {
     case 0 : return (int)image[x][y];

     case 1 : if (R(x+1))
                    return (int)255;
              else
                    return (int)image[x+1][y];

     case 2 : if (R(x+1)||T(y-1))
                    return (int)255;
              else
                    return (int)image[x+1][y-1];

     case 3 : if (T(y-1))
                    return (int)255;
              else
                    return (int)image[x][y-1];

     case 4 : if (T(y-1)||L(x-1))
                    return (int)255;
              else
                    return (int)image[x-1][y-1];

     case 5 : if (L(x-1))
                    return (int)255;
```

```
                else
                    return (int)image[x-1][y];

    case 6 : if (L(x-1)||B(y+1))
                    return (int)255;
                else
                    return (int)image[x-1][y+1];

    case 7 : if(B(y+1))
                    return (int)255;
                else
                    return (int)image[x][y+1];

    case 8 : if(B(y+1)||R(x+1))
                    return (int)255;
                else
                    return (int)image[x+1][y+1];
  }//end switch(i)
 return -1;
}//end sub  X

void smooth_gray_image(unsigned char huge image[][Y_RANGE],unsigned char huge
                       image0[][Y_RANGE])
//
// input : image, image0
// output : average neibourhood of image => image0
//
{
  int x,y,sum;
  double fraction,integer;

  for(y=0;y<bmphead.depth;y++)
    for(x=0;x<bmphead.width;x++)
    {
```

```
        //  1 1 1                   4 3 2
        //  1 1 1  / 9      X(i) =  5 0 1
        //  1 1 1                   6 7 8
        sum  = X(0,x,y,image);
        sum += X(1,x,y,image);
        sum += X(2,x,y,image);
        sum += X(3,x,y,image);
        sum += X(4,x,y,image);
        sum += X(5,x,y,image);
        sum += X(6,x,y,image);
        sum += X(7,x,y,image);
        sum += X(8,x,y,image);

        fraction = modf((double)sum/9,&integer);

        if(fraction>=(double)0.5)
          integer = integer + 1;

        image0[x][y] = (unsigned char)integer;
    }//end for.. for..
}//ens sub smooth_gray_image

unsigned char X_bin(int i,int x,int y,unsigned char huge image[][Y_RANGE])
{
//        4 3 2
// X(i) = 5 0 1
//        6 7 8
  switch(i)
   {
    case 0 : return (unsigned char)image[x][y];

    case 1 : if (R(x+1))
                return (unsigned char)0;
             else
```

```
                return (unsigned char)image[x+1][y];

case 2 : if (R(x+1)||T(y-1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x+1][y-1];

case 3 : if (T(y-1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x][y-1];

case 4 : if (T(y-1)||L(x-1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x-1][y-1];

case 5 : if (L(x-1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x-1][y];

case 6 : if (L(x-1)||B(y+1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x-1][y+1];

case 7 : if (B(y+1))
                return (unsigned char)0;
            else
                return (unsigned char)image[x][y+1];

case 8 : if (B(y+1)||R(x+1))
                return (unsigned char)0;
```

```
            else
                return (unsigned char)image[x+1][y+1];
    }//end switch(i)
 return 0;
}//end sub X_bin


void smooth_binary_image(unsigned char huge image[][Y_RANGE])
//
//  input : image is binary
// output : smooth image => image
//
{
 unsigned char a,b,c,d,p,e,f,g,h;
 int x,y;

 for(y=0;y<bmphead.depth;y++)
  for(x=0;x<bmphead.width;x++)
  {
    a = X_bin(4,x,y,image);
    b = X_bin(3,x,y,image);       //   a  b  c
    c = X_bin(2,x,y,image);       //
    d = X_bin(5,x,y,image);       //   d  p  e
    p = X_bin(0,x,y,image);       //
    e = X_bin(1,x,y,image);       //   f  g  h
    f = X_bin(6,x,y,image);
    g = X_bin(7,x,y,image);
    h = X_bin(8,x,y,image);

    // From Robotic Book
    if (p)
      image[x][y] =  ( (a|b|d)&(e|h|g) ) | ( (b|c|e)&(g|f|d) );
    else
      image[x][y] = (b&g&(d|e)) | (d&e&(b|g));
```

```
  }//end for.. for..
}//end sub smooth_binary_image
```

```
//*********************************************************************
//
// Threshol.cpp
//
// Threshold : minimum cross entropy for
//                autonmatic treshold form histogram
//
//*********************************************************************

#define INITIAL  2
#define DYNAMICS 1
#define STATICS  0

int thresholding(unsigned int *h,const int L);

double pOi(unsigned int *h,int i,int S,int lower_bound,int status);
double pBi(unsigned int *h,int i,int S,int lower_bound,int status);
double lambdaB(unsigned int *h,int S,int lower_bound,int upper_bound,int status);
double qOi(unsigned int *h,int i,int S,int lower_bound,int status);
double qBi(unsigned int *h,int i,int S,int lower_bound,int upper_bound,int status);
double DO(unsigned int *h,int S,int lower_bound);
double DB(unsigned int *h,int S,int lower_bound,int upper_bound);
double D(unsigned int *h,int S,int lower_bound,int upper_bound);
double lambdaO(unsigned int *h,int S,int lower_bound,int status);

unsigned long iPs(unsigned int *h,int S,int lower_bound,int status);
```

```
unsigned long Ps(unsigned int *h,int S,int lower_bound,int status);
unsigned long Ps_inv(unsigned int *h,int S,int lower_bound,int status);

//*********************************************************************
unsigned long Ps(unsigned int *h,int S,int lower_bound,int status)
{
 static unsigned long sum;

 if(status==INITIAL)
   {
    sum = (unsigned long)(*(h+lower_bound));
    return (unsigned long)0;
   }
 else
  if(status==DYNAMICS)
   {
    sum = sum + (unsigned long)(*(h+S));
    return (unsigned long)0;
   }
return sum;
}

double pOi(unsigned int *h,int i,int S,int lower_bound,int status)
{
 static double buffer;

 if(status==INITIAL)
   buffer = Ps(h,S,lower_bound,STATICS);
 else
   return ((double)(*(h+i))/buffer);

 return (double)0;
}
```

```
double pBi(unsigned int *h,int i,int S,int lower_bound,int status)
{
  static double buffer;

  if(status==INITIAL)
    buffer = Ps_inv(h,S,lower_bound,STATICS);
  else
    return ((double)(*(h+i))/buffer);

  return (double)0;
}

unsigned long iPs(unsigned int *h,int S,int lower_bound,int status)
{
  static unsigned long sum;

  if(status==INITIAL)
    {
      sum = (unsigned long)lower_bound * (unsigned long)(*(h+lower_bound));
      return (unsigned long)0;
    }
  else
    if(status==DYNAMICS)
    {
      sum = sum + ((unsigned long)S * (unsigned long)(*(h+S)));
      return (unsigned long)0;
    }

  return sum;
}

double lambdaO(unsigned int *h,int S,int lower_bound,int status)
{
  static double buffer;
```

```
  if(status==INITIAL)
    buffer = (double)iPs(h,S,lower_bound,STATICS)/
                (double) Ps(h,S,lower_bound,STATICS);
  else
    return buffer;

  return (double)0;
}

unsigned long Ps_inv(unsigned int *h,int S,int lower_bound,int status)
{
  static unsigned long sum;

  if(status==INITIAL)
    {
      sum = ((unsigned long)bmphead.width * (unsigned long)bmphead.depth)
            - (unsigned long)(*(h+lower_bound));
      return (unsigned long)0;
    }
  else
    if(status==DYNAMICS)
      {
          sum = sum - (unsigned long)(*(h+S));
          return (unsigned long)0;
      }

 return sum;
}

unsigned long iPs_inv(unsigned int *h,int S,int lower_bound,int upper_bound,int status)
{
  static unsigned long sum;
```

```
  if(status==INITIAL)
    {
      sum = (unsigned long)0;

      for(int i=lower_bound;i<=upper_bound;++i)
          sum = sum + ((unsigned long) i * (unsigned long)(*(h+i)));

      sum = sum  - ((unsigned long)lower_bound * (unsigned long)(*(h+lower_bound)));
      return (unsigned long)0;
    }
  else
    if(status==DYNAMICS)
      {
          sum = sum - ((unsigned long)S * (unsigned long)(*(h+S)));
          return (unsigned long)0;
      }

 return sum;
}

double lambdaB(unsigned int *h,int S,int lower_bound,int upper_bound,int status)
{
  static double buffer;

  if(status==INITIAL)
    buffer = (double)iPs_inv(h,S,lower_bound,upper_bound,STATICS)/
              (double) Ps_inv(h,S,lower_bound,STATICS);
  else
    return buffer;

  return (double)0;
}

double qOi(unsigned int *h,int i,int S,int lower_bound,int status)
```

```
{
 static double lambdaObuffer;
 double product;
 int j;

 if(status==INITIAL)
   {
    lambdaObuffer = lambdaO(h,S,lower_bound,STATICS);
    return (double)0;
   }
 else
  if(i==0)
    return (exp((-1) * lambdaObuffer));
  else
    {
     product = exp((-1) * lambdaObuffer);
     for(j=1;j<=i;++j)
          product = product * (lambdaObuffer / j);
    }

 return product;
}

double qBi(unsigned int *h,int i,int S,int lower_bound,int upper_bound,int status)
{
 static double lambdaBbuffer;
 double product;
 int j;

 if(status==INITIAL)
   {
    lambdaBbuffer = lambdaB(h,S,lower_bound,upper_bound,STATICS);
    return (double)0;
   }
```

```
  else
    if(i==0)
      return (exp((-1) * lambdaBbuffer));
    else
      {
          product = exp((-1) * lambdaBbuffer);
          for(j=1;j<=i;++j)
            product = product * (lambdaBbuffer / j);
      }

 return product;
}

double DO(unsigned int *h,int S,int lower_bound)
{
  int i;
  double sum1=0,sum2=0;
  double buffer1,buffer2;

  pOi(h,i,S,lower_bound,INITIAL);
  qOi(h,i,S,lower_bound,INITIAL);
  for(i=lower_bound;i<=S;++i)
    {
      buffer1 = pOi(h,i,S,lower_bound,STATICS);
      buffer2 = qOi(h,i,S,lower_bound,STATICS);

      if((buffer1!=0)&&(buffer2!=0))
          {
           if((buffer1/buffer2==0)||(buffer2/buffer1==0)) ;
           else
             {
               sum1 = sum1 + (buffer1 * log10(buffer1/buffer2));
               sum2 = sum2 + (buffer2 * log10(buffer2/buffer1));
             }
```

```
        }
    }

 return (sum1+sum2);
}

double DB(unsigned int *h,int S,int lower_bound,int upper_bound)
{
  int i;
  double sum1=0,sum2=0;
  double buffer1,buffer2;

  pBi(h,i,S,lower_bound,INITIAL);
  qBi(h,i,S,lower_bound,upper_bound,INITIAL);

  for(i=S+1;i<=upper_bound;++i)
   {
     buffer1 = pBi(h,i,S,lower_bound,STATICS);
     buffer2 = qBi(h,i,S,lower_bound,upper_bound,STATICS);

     if((buffer1!=0)&&(buffer2!=0))
         {
          if((buffer1/buffer2==0)||(buffer2/buffer1==0)) ;
          else
            {
              sum1 = sum1 + (buffer1 * log10(buffer1/buffer2));
              sum2 = sum2 + (buffer2 * log10(buffer2/buffer1));
            }
         }
   }

 return (sum1+sum2);
}
```

```
double D(unsigned int *h,int S,int lower_bound,int upper_bound)
{
  return (DO(h,S,lower_bound) + DB(h,S,lower_bound,upper_bound));
}

int thresholding(unsigned int *h,const int L)
{
  int i,upper_bound,lower_bound;
  int S,threshold;
  double Mindiv = 10000;
  double buffer;

  i=0;

  while(h[i]==0)
    ++i;

  lower_bound = i;
  i = L-1;

  while(h[i]==0)
    --i;

  upper_bound = i;

  Ps(h,S,lower_bound,INITIAL);
  iPs(h,S,lower_bound,INITIAL);
  Ps_inv(h,S,lower_bound,INITIAL);
  iPs_inv(h,S,lower_bound,upper_bound,INITIAL);

  for(S=lower_bound+1;S<=upper_bound-1;++S)
    {
      Ps(h,S,lower_bound,DYNAMICS);
      iPs(h,S,lower_bound,DYNAMICS);
```

```
    Ps_inv(h,S,lower_bound,DYNAMICS);
    iPs_inv(h,S,lower_bound,upper_bound,DYNAMICS);
    lambdaO(h,S,lower_bound,INITIAL);
    lambdaB(h,S,lower_bound,upper_bound,INITIAL);
    buffer = D(h,S,lower_bound,upper_bound);
    if(Mindiv > buffer)
        {
         Mindiv = buffer;
        threshold = S;
        }
  }
return threshold;
}
```

```
//*********************************************************************
//
// Binary.cpp
//
// Binaization : convert grey-level to binary image
//
//*********************************************************************

void Binarization(unsigned char huge image[][Y_RANGE],unsigned char huge image_0[][Y_RANGE],
                  int threshold);

void show_binary_image(int x,int y,unsigned char huge image[][Y_RANGE]);
void show_binary_image_invert(int x,int y,unsigned char huge image[][Y_RANGE]);
```

```
//*********************************************************************
/* Binarization */
void Binarization(unsigned char huge image[][Y_RANGE],unsigned char huge
                  image_0[][Y_RANGE],int threshold)
//
// input : grey image, threshold
// output : binary image
//
{
  for(int x=0;x<bmphead.width;x++)
    for(int y=0;y<bmphead.depth;y++)
      { '
          if(image[x][y]<=threshold)
            image_0[x][y]=1;
          else
            image_0[x][y]=0;
      }

}//end sub Binarization

void show_binary_image(int x,int y,unsigned char huge image[][Y_RANGE])
//
// input : position x,y of screen, image
// output : screen
//
{
  for(int _y=0;_y<bmphead.depth;_y++)
    for(int _x=0;_x<bmphead.width;_x++)
      {
          if(image[_x][_y] == 1)
            putpixel(x+_x,y+_y,0);
          else
            putpixel(x+_x,y+_y,255);
      }
```

```
  line(x,y,x+256,y,180);
  line(x,y,x,y+256,180);

  line(x,y+256,x+256,y+256,180);
  line(x+256,y,x+256,y+256,180);

}//end sub show_binary_image

void show_binary_image_invert(int x,int y,unsigned char huge image[][Y_RANGE])
//
// input : position x,y of screen, image
// output : screen
//
{
  for(int _y=0;_y<bmphead.depth;_y++)
    for(int _x=0;_x<bmphead.width;_x++)
      {
          if(image[_x][_y] == 1)
            putpixel(x+_x,y+_y,255);
      }

  line(x,y,x+256,y,180);
  line(x,y,x,y+256,180);

  line(x,y+256,x+256,y+256,180);
  line(x+256,y,x+256,y+256,180);

}//end sub show_binary_image_invert
```

```
//*********************************************************************
//
// Thin_zh.cpp
//
// Thinning of Zhang & Suen Algorithms
//
//*********************************************************************

void thin_zhang(unsigned char huge image[][Y_RANGE]);
void thin_smooth(unsigned char huge image[][Y_RANGE]);
int  restorage(unsigned char huge image[][Y_RANGE],unsigned char huge temp_image[][Y_RANGE]);
int  condition_0(int x,int y,unsigned char huge image[][Y_RANGE]);
int  condition_1(int x,int y,unsigned char huge image[][Y_RANGE]);

//*********************************************************************
int  restorage(unsigned char huge image[][Y_RANGE],unsigned char huge
               temp_image[][Y_RANGE])
//
// input : binary image, temp image
// output : result of binary == temp => 1
//                    binary != temp => 0
//
{
  int x,y,z;

  z=1;                                    // set return TURE

  for (x=0;x<256;x++)
    for (y=0;y<256;y++)
      {
       if (temp_image[x][y]!=1)           // if temp set delete
         {
```

```
            if (image[x][y]==1)                    // if (image = 1) do
              {
                image[x][y]=0;
                z=0;                               // set return FALSE
              }//end if
            }//end if
        }//end for.. for..

 return z;

}//end sub restorage

int condition_0(int x,int y,unsigned char huge image[][Y_RANGE])
//
// input : position x,y of image
// output : result of condition in thinning (undelete or delete)
//
{
  int a=0,b=0,c=0,d=0;

// A()
  if ((P1==0)&&(P2==1))  a++;
  if ((P2==0)&&(P3==1))  a++;
  if ((P3==0)&&(P4==1))  a++;
  if ((P4==0)&&(P5==1))  a++;
  if ((P5==0)&&(P6==1))  a++;
  if ((P6==0)&&(P7==1))  a++;
  if ((P7==0)&&(P8==1))  a++;
  if ((P8==0)&&(P1==1))  a++;

// B()
  b = P1+P2+P3+P4+P5+P6+P7+P8;

// P2*P4*P6
```

```
  c = P2*P4*P6;

// P4*P6*P8
  d = P4*P6*P8;

// Decision
  if ((a==1)&&(1<b)&&(b<7)&&(c==0)&&(d==0))
    return 0;                    // delete
  else
    return 1;                    // undelete

}//end sub condition0

int condition_1(int x,int y,unsigned char huge image[][Y_RANGE])
//
//  input : position x,y of image
// output : result of condition in thinning (undelete or delete)
//
{
  int a=0,b=0,c=0,f=0;

// A()
  if ((P1==0)&&(P2==1))  a++;
  if ((P2==0)&&(P3==1))  a++;
  if ((P3==0)&&(P4==1))  a++;
  if ((P4==0)&&(P5==1))  a++;
  if ((P5==0)&&(P6==1))  a++;
  if ((P6==0)&&(P7==1))  a++;
  if ((P7==0)&&(P8==1))  a++;
  if ((P8==0)&&(P1==1))  a++;

// B()
  b = P1+P2+P3+P4+P5+P6+P7+P8;
```

```
// P2*P4*P8
  e = P2*P4*P8;

// P2*P6*P8
  f = P2*P6*P8;

// Decision
  if ((a==1)&&(1<b)&&(b<7)&&(e==0)&&(f==0))
    return 0;                    // delete
  else
    return 1;                    // undelete

}//end sub condition1

void thin_zhang(unsigned char huge image[][Y_RANGE])
//
//  input : binary image
// output : binary image is thinning
//
{
  int a,b;
  int x,y;

  for (x=0;x<bmphead.depth;x++)
    {                                    // 0 0 0 0 0
      image[x][0]=0;                     // x x x x x   set top and bottom
      image[x][Y_RANGE]=0;               // x x x x x
    }                                    // 0 0 0 0 0
  for (y=0;y<bmphead.depth;y++)
    {                                    // 0 0 0 0 0
      image[0][y]=0;                     // 0 x x x 0   set left and right
      image[X_RANGE][y]=0;               // 0 x x x 0
    }                                    // 0 0 0 0 0
```

```
a=0;
b=0;

while (!a&!b)
 {
  for (x=1;x<bmphead.depth-1;x++)
      for (y=1;y<bmphead.depth-1;y++)
       {
        if (image[x][y]==1)
          {

           temp_image[x][y] = 1;                    // reset backup image

           temp_image[x][y] = condition_0(x,y,image);     // out put of image
          }
       }//end for.. for..

  a = restorage(image,temp_image);

  for (x=1;x<bmphead.depth-1;x++)
      for (y=1;y<bmphead.depth-1;y++)
       {
        if (image[x][y]==1)
          {

           temp_image[x][y] = 1;                    // reset backup image

           temp_image[x][y] = condition_1(x,y,image);     // out put of image
          }
       }//end for.. for..

  b = restorage(image,temp_image);

 }//end while
```

```
 thin_smooth(image);

}//end sub thin_zhang


void thin_smooth(unsigned char huge image[][Y_RANGE])
//
//  input : binary image as thinning
// output : binary image
//
{
  int a,b,x,y;

  for (x=1;x<bmphead.depth-1;x++)
    for (y=1;y<bmphead.depth-1;y++)
      {
          if (image[x][y]==1)
            {
              a = (P2&~P6)&(P4&~P8)&(~P7); // P1 P2 P3

              a |= (P4&~P8)&(P6&~P2)&(~P1);// P8 P0 P4

              a |= (P6&~P2)&(P8&~P4)&(~P3);          // P7 P6 P5

              a |= (P2&~P6)&(P8&~P4)&(~P5);

              b = (P1|P2|P3|P4|P5|P6|P7|P8);

              if ((a)|(!b))
                image[x][y] = 0;        // reset backup image
              else
                image[x][y] = 1;  // out put of image
```

```
        }
    }//end for.. for..

}//end sub thin_smooth


//*********************************************************************
//
//  Thin_op.cpp
//
//  Jang's and Chin's One Pass Parallel Thinning Algorithm
//
//*********************************************************************

void thinning(unsigned char huge S[][Y_RANGE]);

//*********************************************************************
void thinning(unsigned char huge S[][Y_RANGE])
 {
  unsigned char last_del,last_store,X[25];
  int flag_change = 1;
  int x,y,i;

  while(flag_change)
  {
   flag_change = 0;
   for(y=0;y<bmphead.depth;++y)
    for(x=0;x<bmphead.width;++x)
    {
```

```
for(i=0;i<25;++i)
 X[i] = X_bin(i,x,y,S);

last_del = 0;
last_store = 0;

if(X[0])
{
 // A1
 if((X[1])&&(!X[3])&&(!X[4])&&(!X[5])&&(X[7]))
  last_del = last_del | (unsigned char)1;

 else
 // A2
 if((!X[1])&&(!X[2])&&(!X[3])&&(X[5])&&(X[7]))
 last_del = last_del | (unsigned char)1;

 else
 // A3
 if((!X[1])&&(X[3])&&(X[5])&&(!X[7])&&(!X[8]))
  last_del = last_del | (unsigned char)1;

 else
 // A4
 if((X[1])&&(X[3])&&(!X[5])&&(!X[6])&&(!X[7]))
  last_del = last_del | (unsigned char)1;

 else
 // A5
 if((X[1])&&(!X[3])&&(X[5])&&(X[6])&&(X[7])&&(X[8]))
```

```
last_del = last_del | (unsigned char)1;

else
// A6
if((!X[1])&&(X[3])&&(X[4])&&(X[5])&&(X[6])&&(X[7]))
 last_del = last_del | (unsigned char)1;

else
// A7
if((X[1])&&(X[2])&&(X[3])&&(X[4])&&(X[5])&&(!X[7]))
 last_del = last_del | (unsigned char)1;

else
// A8
if((X[1])&&(X[2])&&(X[3])&&(!X[5])&&(X[7])&&(X[8]))
 last_del = last_del | (unsigned char)1;

else
// A9
if((!X[1])&&(!X[2])&&(!X[3])&&(!X[4])&&(!X[5])&&(X[7])
    &&(X[6]||X[8]))
 last_del = last_del | (unsigned char)1;

else
// A10
if((!X[1])&&(!X[2])&&(!X[3])&&(X[5])&&(!X[7])&&(!X[8])
    &&(X[4]||X[6]))
 last_del = last_del | (unsigned char)1;

else
```

```
// A11
if((!X[1])&&(X[3])&&(!X[5])&&(!X[6])&&(!X[7])&&(!X[8])
    &&(X[2]||X[4]))
 last_del = last_del | (unsigned char)1;

 else
 // A12
 if((X[1])&&(!X[3])&&(!X[4])&&(!X[5])&&(!X[6])&&(!X[7])
     &&(X[2]||X[8]))
  last_del = last_del | (unsigned char)1;

  else
  // A13
  if((!X[1])&&(!X[2])&&(!X[3])&&(!X[4])&&(!X[5])&&(!X[6])
      &&(!X[7])&&(X[8])&&(X[22])&&(X[23])&&(X[24]))
   last_del = last_del | (unsigned char)1;

   else
   // A14
   if((!X[1])&&(!X[2])&&(!X[3])&&(!X[4])&&(!X[5])&&(X[6])
       &&(!X[7])&&(!X[8])&&(X[18])&&(X[19])&&(X[20]))
    last_del = last_del | (unsigned char)1;

    else
    // A15
    if((!X[1])&&(!X[2])&&(!X[3])&&(X[4])&&(!X[5])&&(!X[6])
        &&(!X[7])&&(!X[8])&&(X[14])&&(X[15])&&(X[16]))
     last_del = last_del | (unsigned char)1;

     else
```

```
// A16
if((!X[1])&&(X[2])&&(!X[3])&&(!X[4])&&(!X[5])&&(!X[6])
    &&(!X[7])&&(!X[8])&&(X[10])&&(X[11])&&(X[12]))
 last_del = last_del | (unsigned char)1;

else
// A17
if((X[1])&&(X[2])&&(X[3])&&(!X[4])&&(X[5])&&(X[6])
    &&(X[7])&&(X[8])&&(X[9])&&(X[10])&&(X[12])&&(X[13])
    &&(X[17])&&(X[18])&&(X[20])&&(X[21])&&(X[22])&&
    (X[23])&&(X[24]))
 last_del = last_del | (unsigned char)1;

else
// A18
if((X[1])&&(!X[2])&&(X[3])&&(X[4])&&(X[5])&&(X[6])
    &&(X[7])&&(X[8])&&(X[9])&&(X[13])&&(X[14])&&(X[16])
    &&(X[17])&&(X[18])&&(X[19])&&(X[20])&&(X[21])
    &&(X[22])&&(X[24]))
 last_del = last_del | (unsigned char)1;

else
// A19
if((X[1])&&(X[2])&&(X[3])&&(X[4])&&(X[5])&&(X[6])
    &&(X[7])&&(!X[8])&&(X[9])&&(X[10])&&(X[12])&&(X[13])
    &&(X[14])&&(X[15])&&(X[16])&&(X[17])&&(X[18])
    &&(X[20])&&(X[21]))
 last_del = last_del | (unsigned char)1;

else
```

```
// A20
if((X[1])&&(X[2])&&(X[3])&&(X[4])&&(X[5])&&(!X[6])
    &&(X[7])&&(X[8])&&(X[9])&&(X[10])&&(X[11])&&(X[12])
    &&(X[13])&&(X[14])&&(X[16])&&(X[17])&&(X[21])
    &&(X[22])&&(X[24]))
 last_del = last_del | (unsigned char)1;
// A21
if((!X[1])&&(X[2])&&(X[3])&&(!X[4])&&(X[5])&&(!X[7])
    &&(!X[8])&&(!X[13])&&(!X[14]))
 last_store = last_store | (unsigned char)1;

else
// A22
if((!X[1])&&(!X[2])&&(!X[3])&&(X[5])&&(!X[6])&&(X[7])
    &&(X[8])&&(!X[20])&&(!X[21]))
 last_store = last_store | (unsigned char)1;

else
// A23
if((!X[1])&&(!X[2])&&(!X[3])&&(X[4])&&(X[5])&&(!X[6])
    &&(X[7])&&(!X[17])&&(!X[18]))
 last_store = last_store | (unsigned char)1;

else
// A24
if((!X[1])&&(X[3])&&(!X[4])&&(X[5])&&(X[6])&&(!X[7])
    &&(!X[8])&&(!X[16])&&(!X[17]))
 last_store = last_store | (unsigned char)1;

else
```

```
// A25
if((X[1])&&(!X[3])&&(X[5])&&(X[6])&&(X[7])&&(X[8])
    &&(!X[21]))
 last_store = last_store | (unsigned char)1;

else
// A26
if((X[1])&&(X[2])&&(X[3])&&(!X[5])&&(X[7])&&(X[8])
    &&(!X[9]))
 last_store = last_store | (unsigned char)1;

else
// A27
if((X[1])&&(!X[2])&&(!X[3])&&(!X[4])&&(!X[5])&&(!X[6])
    &&(X[7])&&(!X[9])&&(!X[10])&&(!X[20])&&(!X[21])&&(!X[22])
    &&(!X[23])&&(!X[24]))
 last_store = last_store | (unsigned char)1;

else
// A28
if((!X[1])&&(!X[2])&&(!X[3])&&(!X[4])&&(X[5])&&(X[7])
    &&(!X[8])&&(!X[16])&&(!X[17])&&(!X[18])&&(!X[19])&&(!X[20])
    &&(!X[21])&&(!X[22]))
 last_store = last_store | (unsigned char)1;

else
// A29
if((!X[1])&&(!X[2])&&(X[3])&&(X[5])&&(!X[6])&&(!X[7])
    &&(!X[8])&&(!X[12])&&(!X[13])&&(!X[14])&&(!X[15])&&(!X[16])
    &&(!X[17])&&(!X[18]))
```

```
        last_store = last_store | (unsigned char)1;


      else
      // A30
      if((X[1])&&(X[3])&&(!X[4])&&(!X[5])&&(!X[6])&&(!X[7])
          &&(!X[8])&&(!X[9])&&(!X[10])&&(!X[11])&&(!X[12])&&(!X[13])
          &&(!X[14])&&(!X[24]))
       last_store = last_store | (unsigned char)1;
     }


    if((last_del==1)&&(last_store==0))
    {
     temp_image[x][y] = (unsigned char)1;
     flag_change = 1;
    }
     else
      if((last_del==1)&&(last_store==1))
         temp_image[x][y] = (unsigned char)0;
         else  temp_image[x][y] = (unsigned char)0;
    }


  for(y=0;y<bmphead.depth;++y)
   for(x=0;x<bmphead.width;++x)
    if(temp_image[x][y]==1) S[x][y] = 0;
  }
}
```

```
//*********************************************************************
//
// POST.cpp
//
// Postprocessing : a comboine statistical and structure appoach
//
//*********************************************************************

void minutia(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE]);
void invert(int &x,int &y);
void travel(unsigned char huge image[][Y_RANGE],int &a,int &b,int &p);
void single_point(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE]);
void short_ridge(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE],
                 int lamda);
void broken_ridge(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE],
                  int lamda);
void endridge(unsigned char huge image[][Y_RANGE],int &x,int &y,int &p);

//*********************************************************************
void endridge(unsigned char huge image[][Y_RANGE],int &x,int &y, int &p)
//
// input : endridge point and position x,y is binary image
// output : travel 1 step on ridge, new position x,y and follow to p
//
{
  int a;
```

```
a  = P1*1 + P2*2 + P3*3 + P4*4;
a += P5*5 + P6*6 + P7*7 + P8*8;

switch (a)
  {
    case 1 : x--; y--; p = 5;          // P1 P2 P3        y-
             break;
                                       // P8 P0 P4    x-  x,y  x+
    case 2 : y--;      p = 6;
             break;                    // P7 P6 P5        y+

    case 3 : x++; y--; p = 7;
             break;

    case 4 : x++;      p = 8;
             break;

    case 5 : x++; y++; p = 1;
             break;

    case 6 : y++;      p = 2;
             break;

    case 7 : x--; y++; p = 3;
             break;

    case 8 : x--;      p = 4;
             break;

  }//end switch

}//end sub endridge
```

```
void broken_ridge(unsigned char huge image[][Y_RANGE],unsigned char huge
                  image0[][Y_RANGE],int lamda)
//
// input :     binary image, minutia image
// output : new binary image, minutia image
//
{
  int xx,yy;
  float tmp;

  for (yy = 0; yy < 256 ; yy++)
    for (xx = 0; xx < 256 ; xx++)
      {
          if (image0[xx][yy]==1)
            {
              if (tmp==500)
               {
                 minutia(image,image0);
                 show_binary_image(x_new,y_new,image);

               }
            }
      }//end for.. for..

}//end sub broken_ridge

void single_point(unsigned char huge image[][Y_RANGE],unsigned char huge
                  image0[][Y_RANGE])
//
// input :     binary image, minutia image
// output : new binary image, minutia image
//
{
  int x,y;
```

```
  for (y = 0; y<256 ; y++)
    for (x = 0; x<256 ; x++)
      {
          if (image0[x][y]==5)
            {
              image[x][y] = 0;
              image0[x][y] = 0;
            }//end if(image0 ...
      }//end for.. for..

}//end sub single_point

void short_ridge(unsigned char huge image[][Y_RANGE],unsigned char huge
                 image0[][Y_RANGE],int lamda)
//
// input :     binary image, minutia image
// output : new binary image, minutia image
//
{
  int x,y;
  int a,b,p,count;
  list ls;

  for (y = 0; y<256 ; y++)
    for (x = 0; x<256 ; x++)
      {
          if (image0[x][y]==1)
            {
              ls.add(x,y);
              a = x ;  b = y ;

              endridge(image,a,b,p);
              ls.add(a,b);
```

```
count = 1;
for(;;)
 {
     if ((image0[a][b]==1)|(image0[a][b]==3)|(count==lamda))
       break;
     travel(image,a,b,p);
     ls.add(a,b);
     count++;
 }

if ((count<lamda)&(image0[a][b]!=3))
 {
     a = 1; b = 1;
     while((a!=NULL)&(b!=NULL))
      {
       a = ls.read_x();
       b = ls.read_y();
       image[a][b] = 0;
       image0[a][b] = 0;
       ls.del(a,b);
      }
 }
else
 {
     a = 1; b = 1;
     while((a!=NULL)&(b!=NULL))
      {
       a = ls.read_x();
       b = ls.read_y();
       ls.del(a,b);
      }
 }
}//end if(image0 ...
}//end for.. for..
```

```
}//end sub short_ridge

void minutia(unsigned char huge image[][Y_RANGE],unsigned char huge image0[][Y_RANGE])
//
//  input : binary image {0,1} => image[][]
// output : minutia image      <= image0[][] {1,2,3,4,5,6}
//
// P1 P2 P3    Ns = sum[P(k) ^P(k-1)]  ;k={1,2,3,4,5,6,7,8)
//
// P8 P0 P4 => Ng = sum[P(k)]          ;k={2,4,6,8}
//
// P7 P6 P5
{
  int x,y;
  int ns,ng;

  for (y=1 ; y < 255 ; y++)
    for (x=1 ; x < 255 ; x++)
      {
          image0[x][y] = 0;
          if (image[x][y]==1)
            {
              ns  = (P1^P8)+(P2^P1);
              ns += (P3^P2)+(P4^P3);
              ns += (P5^P4)+(P6^P5);
              ns += (P7^P6)+(P8^P7);
              ng  = (P2+P4+P6+P8);

              switch (ns)
                {
                    case 0 : if (ng==4) image0[x][y]=6;  // internal
                                 else       image0[x][y]=5;  // isolate
                             break;
```

```
                case 2 : if (ng<2)  image0[x][y]=1;   // end point
                         else       image0[x][y]=2;   // connect
                         break;

                case 4 : image0[x][y]=2;             // connect
                         break;

                case 6 : image0[x][y]=3;             // bifurcation
                         break;

                case 8 : image0[x][y]=4;             // cross
                         break;
              } // end switch
            } // end if
        } // end for
} // end minutia

void invert(int &x,int &y)
//
//  input : position of box
// output : number of box between x,y
//
{
  if (x > y )
   {
     int tmp;
     tmp = y;
     y = x;
     x = tmp;
   }

  x += ((y-x)/2);              // x < y
```

```
  switch (x)
   {
     case 1 : x = 1;    y = 5;   break;
     case 2 : x = 2;    y = 6; break;
     case 3 : x = 3;    y = 7; break;
     case 4 : x = 4;    y = 8; break;

     case 5 : x = 1;    y = 5;   break;
     case 6 : x = 2;    y = 6; break;
     case 7 : x = 3;    y = 7; break;
     case 8 : x = 4;    y = 8; break;
   }
}//end sub invert


void travel(unsigned char huge image[][Y_RANGE],int &a,int &b,int &p)
//
//  input : position x,y ( P0 = 1 ) and P1..P8 follow to
// output : new position x,y and P1..P8
//
{
  int next=0;
  int x,y;

  x = a;
  y = b;

  switch (p)                                  // find next pixel
    {
      case 1 : next = P3*3 + P4*4 + P5*5 + P6*6 + P7*7;          break;

      case 2 : next = P5*5 + P6*6 + P7*7;                        break;

      case 3 : next = P1*1 + P5*5 + P6*6 + P7*7 + P8*8;          break;
```

```
    case 4 : next = P1*1 + P7*7 + P8*8;                    break;

    case 5 : next = P1*1 + P2*2 + P3*3 + P7*7 + P8*8;      break;

    case 6 : next = P1*1 + P2*2 + P3*3;                    break;

    case 7 : next = P1*1 + P2*2 + P3*3 + P4*4 + P5*5;      break;

    case 8 : next = P3*3 + P4*4 + P5*5;                    break;
   }//end switch

switch (next)                          // new position x,y and next state p
  {
    case 1 : a--; b--; p = 5;      // P1 P2 P3      b-
             break;
                                       // P8 P0 P4    a-  a,b  a+
    case 2 : b--;     p = 6;
             break;                    // P7 P6 P5      b+

    case 3 : a++; b--; p = 7;
             break;

    case 4 : a++;     p = 8;
             break;

    case 5 : a++; b++; p = 1;
             break;

    case 6 : b++;     p = 2;
             break;

    case 7 : a--; b++; p = 3;
             break;
```

```
      case 8 : a--;      p = 4;
                  break;
    }//end switch

}//end sub travel
```

```
//*********************************************************************
//
// SINGLE.CPP
//
//  Fingerprint Pattern Classification
//
//*********************************************************************

struct Dir
  {
    int d;
    float p_0;
    float p_45;
    float p_90;
    float p_135;
  };
```

```
void box_(int x0,int y0,int step);
void line_array(unsigned char huge image[][Y_RANGE],int _x,int _y,int degree);
void direction(unsigned char huge image[][Y_RANGE],float dir[64][64]);
int  find_direction(unsigned char huge image[][Y_RANGE],int x, int y,float &p0,float &p1,float &p2,
                    float &p3);                          // from binarization

//*********************************************************************
void box_(int x0,int y0,int step)
//
// input : original of start (x,y) , block size (step)
// output : display
//
{
  for (int a = 0;a<=256;a+=step)
    line(x0,y0+a,x0+256,y0+a,180);

  for (a = 0;a<=256;a+=step)
    line(x0+a,y0,x0+a,y0+256,180);
}

int find_direction(unsigned char huge image[][Y_RANGE],int x, int y,float &p0,float &p1,
                   float &p2,float &p3)
//
// input  : postion (x,y), direction of box  p = {0,45,90,135}
// output : degree of box ={0,45,90,135} and property ...
//
{
  int m,n,o,p;
  int wr=0,bk=0;
  int xx,yy;
  int sum;

  float a=0,b=0,c=0,d=0,e=0;
```

```
for (int _a=0;_a<16;_a++)          //  array[16][16] = array[_a][_b]
  for (int _b=0;_b<16;_b++)
    {
        xx = x + _a ;
        yy = y + _b ;

        m = image[xx][yy];          //    m    n
        n = image[xx+1][yy];        //
        o = image[xx+1][yy+1];      //    p    o
        p = image[xx][yy+1];        //
                                                  //**********
        if ((m==0)&(n==0)&(o==0)&(p==0))// 0 0
          {                                       // 0 0
            wr++;
            goto end_conition;
          }
        if ((m==1)&(n==1)&(o==1)&(p==1))// 1 1
          {                                       // 1 1
            bk++;
            goto end_conition;
          }
                                                  //********** Type 1
        if ((m==1)&(n==1)&(o==0)&(p==0))// 1 1
          {                                       // 0 0
            a++;    goto end_conition;
          }
        if ((m==0)&(n==0)&(o==1)&(p==1))          // 0 0
          {                                       // 1 1
            a++;    goto end_conition;
          }
                                                  //********** Type 2
        if ((m==1)&(n==1)&(o==0)&(p==1))// 1 1
          {                                       // 1 0
            b++;    goto end_conition;
```

```
  }
if ((m==0)&(n==1)&(o==1)&(p==1))// 0 1
  {                                          // 1 1
    b++;    goto end_conition;
  }
if ((m==0)&(n==0)&(o==1)&(p==0))             // 0 0
  {                                          // 0 1
    b++;    goto end_conition;
  }
if ((m==1)&(n==0)&(o==0)&(p==0))             // 1 0
  {                                          // 0 0
    b++;    goto end_conition;
  }
                                             //********** Type 3
if ((m==0)&(n==1)&(o==1)&(p==0))// 0 1
  {                                          // 0 1
    c++;    goto end_conition;
  }
if ((m==1)&(n==0)&(o==0)&(p==1))             // 1 0
  {                                          // 1 0
    c++;    goto end_conition;
  }
                                             //********** Type 4
if ((m==1)&(n==1)&(o==1)&(p==0))// 1 1
  {                                  // 0 1
    d++;    goto end_conition;
  }
if ((m==1)&(n==0)&(o==1)&(p==1))             // 1 0
  {                                          // 1 1
    d++;    goto end_conition;
  }
if ((m==0)&(n==0)&(o==0)&(p==1))             // 0 0
  {                                          // 1 0
    d++;
```

```
        goto end_conition;
      }
     if ((m==0)&(n==1)&(o==0)&(p==0))          // 0 1
      {                                        // 0 0
        d++;    goto end_conition;
      }
                                               //********** Type 5
     if ((m==0)&(n==1)&(o==0)&(p==1))// 0 1
      {                                        // 1 0
        e++;    goto end_conition;
      }
     if ((m==1)&(n==0)&(o==1)&(p==0))          // 1 0
      {                                        // 0 1
        e++;    goto end_conition;
      }

    end_conition:      // Label ... end condition

  }//end for

b = (b + e) / sqrt(2);              // Type 2 + Type 5 / 2 = { 45 }
d = (d + e) / sqrt(2);              // Type 4 + Type 5 / 2 = { 135 }

sum = a+b+c+d;

if (sum!=0)
  {
    p0 = a / sum;
    p1 = b / sum;
    p2 = c / sum;
    p3 = d / sum;
  }
else
  {
```

```
    p0 = 0.0;
    p1 = 0.0;
    p2 = 0.0;
    p3 = 0.0;
   }

 if ((wr>=200)|(bk>=200))
  {
   m  = 360;
  }
 else
  {
   float temp;

   if (a > b)
    {
        m = 0;   temp = a;
    }
   else
    {
        m = 45; temp = b;
    }
   if (temp < c)
    {
        m = 90; temp = c;
    }
   if (temp < d)
    {
        m = 135; temp = d;
    }
  }
 return m;
}
```

```
void line_array(unsigned char huge image[][Y_RANGE],int x,int y,int degree)
//
// input  : degree of array[x][y] is sixe 16*16
// output : line into array[x][y]
//
{
  int a,b;

  for (a=0;a<16;a++)                                    // clear array[][]
   for (b=0;b<16;b++)
     image[x+a][y+b]=0;

  //*************** Line is 0,45,90,135 degree ********
  switch (degree)
    {
      case   0 : for (a=0;a<16;a++)
                    image[x+a][y+8]=1;
                  break;

      case  45 : for (a=0;a<16;a++)
                    image[x+ a][15+y-a]=1;
                  break;

      case  90 : for (b=0;b<16;b++)
                    image[x+8][y+b]=1;
                  break;

      case 135 : for (a=0;a<16;a++)
                    image[x+a][y+a]=1;
                  break;
    }
}

void direction(unsigned char huge image[][Y_RANGE],struct Dir d[16][16])
```

```
{
  int x,y;
  int a=0,b;
  int degree;
  float p0,p1,p2,p3;

  for (y=0;y<256;y+=16,a++)
   {
     b = 0;
     for (x=0;x<256;x+=16,b++)
         {
           degree = find_direction(image,x,y,p0,p1,p2,p3);
           d[a][b].d     = degree;
           d[a][b].p_0   = p0;
           d[a][b].p_45  = p1;
           d[a][b].p_90  = p2;
           d[a][b].p_135 = p3;

           line_array(image,x,y,degree);
         }
   }
}

//*********************************************************************
//
// SAVE.CPP
//
// Save file : result of process is to *.bmp and *.txt
//
//*********************************************************************

void save_bmp(char name[50],int index,unsigned char huge image[X_RANGE][Y_RANGE]);
void save_txt(int t,unsigned int image[256],struct Dir image0[16][16]);
```

```
//*********************************************************************
void save_bmp(char name[50],int index,unsigned char huge image[X_RANGE][Y_RANGE])
//
// input : original filename.bmp, index filename and image
// output : file.bmp
//
{
 FILE *fbmp,*fb;
 char stb[15];
 char ch;

 switch (index)
  {
   case 0 : strcpy(stb,"smg.bmp"); break;     // Smooth Grey level image

   case 1 : strcpy(stb,"bi.bmp");  break;     // Binary image

   case 2 : strcpy(stb,"smb.bmp"); break;     // Smooth Binary image

   case 3 : strcpy(stb,"dir.bmp"); break;     // Direction image

   case 4 : strcpy(stb,"tt.bmp");  break;     // Thinning image

   default: strcpy(stb,"def.bmp"); break;
  }

 if (((fbmp=fopen(name,"rb"))==NULL)&((fb=fopen(stb,"wb"))==NULL))
  {
   printf("Error");
   exit(0);
  }
 else
  {
```

```
    fseek(fbmp,0L,SEEK_SET);                    // Copy Header FIle BMP
    fseek(fb  ,0L,SEEK_SET);

    while(!feof(fbmp))
        {
          ch=fgetc(fbmp);
          fputc(ch,fb);
        }
  }

fseek(fbmp,bmphead.headersize,SEEK_SET);
fseek(fb  ,bmphead.headersize,SEEK_SET);

long width = ((bmphead.width+7)/8)*8;

if (index==0)                                   // Grey Level Image
 {
   for (int y=0;y<bmphead.depth;y++)
     for (int x=0;x<width;x++)
        {
          fputc((char)image[x][255-y],fb);
        }
 }
else                                            // Binary Image
 {
   for (int y=0;y<bmphead.depth;y++)
    for (int x=0;x<width;x++)
        {
          if (image[x][255-y]==0)
           {
             fputc((char)255,fb);
           }
          else
           {
```

```
                fputc((char)0,fb);
              }
            }
  }
  fclose(fb);
  fclose(fbmp);

}//end sub save_bmp

void save_txt(int t,unsigned int image[256],struct Dir image0[16][16])
//
// input : threshold, image
// output : binary file, minutia file and histogram file
//
{
  FILE *_txt;
  char  str[20];
  int   x,y;

//********** Dircetion **********
  _txt=fopen("binary.txt","w+");

  for (y=0;y<256;y++)
   {
    for (x=0;x<256;x++)
      {
          ltoa(binary_image[x][y],str,10);
          fputs(str,_txt);
      }
    putc('\n',_txt);
   }
  fclose(_txt);

//********** Dircetion **********
```

```
_txt=fopen("minutia.txt","w+");

for (y=0;y<256;y++)
 {
  for (x=0;x<256;x++)
    {
        ltoa(minutia_image[x][y],str,10);
        fputs(str,_txt);
    }
  putc('\n',_txt);
 }
fclose(_txt);

//********** Histogram **********
 _txt=fopen("Histogram.txt","w+");

 strcpy(str,"Threshold = ");
 fputs(str,_txt);
 ltoa(t,str,10);                          // threshold
 fputs(str,_txt);
 putc('\n',_txt);

 for (x=0;x<256;x++)                      // frequncy level of histogram
  {
    ltoa(x,str,10);
    fputs(str,_txt);
    putc(' ',_txt);

    ltoa(image[x],str,10);
    fputs(str,_txt);
    putc('\n',_txt);
  }
fclose(_txt);
```

```
//********** Histogram **********
  _txt=fopen("Direction.txt","w+");
  float tmp_0,tmp_1;

  for (x=0;x<16;x++)
  {
   for (y=0;y<16;y++)
     {
          tmp_0 =image0[x][y].p_0  - image0[x][y].p_90;      // T = P0 - P90
          tmp_1 =image0[x][y].p_45 - image0[x][y].p_135;     // U = P45 - P135

          if (0.1<tmp_0)
           {
            strcpy(str," +  ");
           }
          else
           {
            if (-0.1 > tmp_0)
              {
                   strcpy(str," -  ");
              }
            else
              {
                   if (tmp_1 != 0)
                     {
                       if (tmp_1 > 0)
                        {
                          strcpy(str," 0+ ");
                        }
                       else
                        {
                          strcpy(str," 0- ");
                        }
                     }
```

```
                else
                  {
                    strcpy(str,"    ");
                  }
              }
            }
          fputs(str,_txt);
      }
    putc('\n',_txt);
  }

}//end sub save_txt
```

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้ดีด้วยความช่วยเหลือจากหลายท่าน ผู้จัดทำขอขอบพระคุณ อาจารย์ รศ.ดร.ครรชิต ไมตรี ที่คอยให้คำแนะนำและปรึกษาด้วยดีเสมอมา ขอขอบคุณสำนักหอสมุดกลาง ห้องสมุดและห้องหนังสืออ้างอิง คณะวิศวกรรมศาสตร์ ที่ให้บริการด้วยดีตลอดมา ขอขอบคุณรุ่นพี่ปริญญาโทที่ให้คำแนะนำและคอยช่วยเหลือ รวมทั้งพี่ๆ น้องๆ และเพื่อนรวมห้องที่ให้ข้อมูลภาพลายนิ้วมือ และสุดท้ายนี้ขอขอบพระคุณ คุณพ่อ คุณแม่ ที่เป็นกำลังใจและคอยดูแลมาตลอด ทำให้ปริญญานิพนธ์นี้สำเร็จลุล่วงจึงขอขอบคุณ ณ โอกาสนี้ด้วย

บรรณานุกรม

1. B. K. Jang and R. T. Chin , "One-pass parallel thinning analysis properties and quantitative evaluation" , Vol. 14 , No. 11 , November 1992 , pp. 1129-1140
2. B. Moayer and K. S. Fu , "A tree system approach for fingerprint pattern recognition", IEEE Trans. on Computers , Vol. c-25 , No. 3 , March 1976 , pp. 262-274
3. C. V. Kameswara Rao and K. Balck , "Finding the core point in a fingerprint" , IEEE Trans. on Computers , Vol c-27 , No. 1 , January 1978 , pp. 77-81
4. P. A. Maragos and R. W. Schafer , "Morphological skeleton representation and coding of binary images" , IEEE Trans. on Acoustics Speech and Signal Processing , Vol. ASSP-34 , No. 5 , October 1986 , pp. 1228-1244
5. C. Arcelli and G. S. di Baja , "A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform" , IEEE Trans. on Patt. Anal. Mach. Intell. , Vol. 11 , No. 4 , April 1989 , pp. 411-414
6. B. K. Jang and R. T. Chin , "Analysis of thinning algorithms using mathematical morphology" , IEEE Trans. Patt. Anal. Mach. Intell. , Vol. 12 , No. 6 , June 1990 , pp. 541-551
7. F. Y. C. Shih and O. R. Mitchell , "A mathematical morphology approach to euclidean distance transformation" , IEEE Trans. on Image Processing , Vol. 1 , No. 2 , April 1992 , pp. 197-204
8. M. Kawagoe and A. Tojo , "Fingerprint pattern classification" , Pattern Recognition , Vol. 17 , No. 3 , 1984 , pp. 295-303
9. B. M. Mehtre N. N. Murthy and S. Kapoor , "Segmentation of fingerprint images using the directional image" , Pattern Recognition , Vol. 20 , No. 4 , 1987 , pp. 429-435
10. M. R. Verma A. K. Majumdar and B. Chatterjee , "Edge detection in fingerprints" , Pattern Recognition , Vol. 20 , No. 5 , 1987 , pp. 513-522
11. B. M. Mehtre and B. Chatterjee , "Segmentation of fingerprint images--A composite method" , Pattern Recognition , Vol. 22 , No. 4 , 1989 , pp. 381-385

12. V. S. Srinivasan and N. N. Murthy , "Detection of singular points in fingerprint images", Pattern Recognition , Vol. 25 , No. 2 , 1992 , pp. 139-153

13. A. D. Brink , "Thresholding of digital images using two dimensiopnal entropies" , Pattern Recognition , Vol. 25 , No. 8 , 1992 , pp. 803-808

14. Nikhil R. Pal , "On minimum cross entropy thresholding" , Pattern Recognition , Vol. 29 , No. 4 , 1996 , pp. 575-580

15. K. Karu and A. K. Jain , "Fingerprint classification" , Pattern Recognition , Vol. 29 , No. 3 , 1996 , pp. 389-404

16. มิสเตอร์ไฮเทค , "บัตรผ่านอวัยวะคน" , นิตยสารสารคดี , ปีที่ 12 , ฉบับที่ 137 , 2539 , หน้า 34

17. คณาจารย์ภาควิชาคณิตศาสตร์ , "ความน่าจะเป็นและสถิติ" , คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย , 410 หน้า , 2537

18. ธันวา ศรีประโมง , "การเขียนโปรแกรมภาษาซีสำหรับซิศวกรรม" , มหานครเทคโนโลยีมหานคร , 739 หน้า , 2538