การปรับปรุงความเร็วเทคนิคการแบ่งส่วนภาพด้วยทฤษฎีกราฟ

IMPROVING SPEED OF SEGMENTATION METHOD BASE ON GRAPH THEORY

นายชัยวัช วัฒนมงคลลาภ
MR. CHAIWAT WATTANAMONGKONLAP

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2538

ISBN 974-621-404-7

IMPROVEING SPEED OF SEGMENTATION METHOD BASE ON GRAPH THEORY

CHAIWAT WATTANAMONGKONLAP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
GRADUATE SCHOOL
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
1995

ISBN 974-621-404-7

หัวข้อวิทยานิพนธ์	การปรับปรุงความเร็วเทคนิคการแบ่งส่วนภาพด้วยทฤษฎีกราฟ
นักศึกษา	นายชัยวัช วัฒนมงคลลาภ
อาจารย์ผู้คุมวิทยานิพนธ์	รศ.ดร.ฟูศักดิ์ ชีวสุวิทย์
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมอิเล็กทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2538

บทคัดย่อ

การลดข้อมูลภาพจากวิธีการแบ่งส่วน ซึ่งอาศัยพื้นฐานทฤษฎีกราฟที่รู้จักกันในชื่อของ Recursive shortest spanning tree จะให้ความถูกต้องแม่นยำของขอบเขตพื้นที่ย่อยสูง แต่จะใช้เวลาในการคำนวนสูง โดยขั้นตอนของการค้นหาทรีจะแปรผันโดยตรงกับจำนวนจุดภาพยกกำลังสอง และจะเกิดปัญหาหน่วยความจำไม่เพียงพอในระหว่างการประมวลผลเมื่อภาพมีขนาดใหญ่ ดังนั้นวิทยานิพนธ์ฉบับนี้จึงได้เสนอวิธีการปรับปรุงความเร็วในการประมวลผลภาพ ด้วยการแบ่งภาพใหญ่ออกเป็นภาพย่อย โดยผลรวมของพื้นที่ย่อยในภาพย่อยเหล่านั้นจะยังคงถูกรักษาไว้ให้เท่ากับจำนวนพื้นที่ย่อยที่กำหนดไว้ในภาพใหญ่เดิม ผลที่ตามมาจะสามารถช่วยลดปัญหาเรื่องหน่วยความจำไม่เพียงพอได้ระดับหนึ่ง

Thesis Title	Improving speed of segmentation method base on graph theory
Student	Mr.Chaiwat Wattanamongkonlap
Thesis Advisor	Assoc.Prof.Dr.Fusak Cheevasuvit
Level of Study	Master of Egineering in Electrical Egineering
Department	Electronic Egineering King Mongkut's Institute of Techonology Ladkrabang
Year	1995

ABSTRACT

The thesis presents a method of image compression by segmentation method based on graph theory. The recursive shortest spanning tree is used in the graph theory for forming the region boundaries. Therefore, the obtained result gives the high accuracy of region boundaries. However, to create a tree it consumes high computational time according to the number of operation in searching step of the recursive shortest spanning tree depends on square of the number of pixels in the image. So the objective of this thesis is to improve the speed of segmentation method. The purpose has to divide an image into several subimages. These subimages will be processed in the same manner as the big image, while the total number of regions in subimages still preserve as the number of the big image. Therefore, the sum of total computation time for the mosaicked image will be reduced. By consequence, the limitation of storage memory can be reduced, and a certain big size of image can be achieved.

กิติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จขึ้นมาได้ด้วยความกรุณาจาก รศ.ดร. ฟูศักดิ์ ชีวสุวิทย์ ที่ได้ให้คำแนะนำและชี้แนวทางที่เป็นประโยชน์ในการแก้ปัญหาการทำวิจัยตลอดระยะเวลาที่ทำการศึกษาอยู่ ซึ่งผู้เขียนขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้ด้วย นอกจากนี้ขอขอบคุณ คุณวิรงรอง กลมสินธ์ ที่ให้ความช่วยเหลือในสิ่งต่างๆ และเป็นกำลังใจจงสำเร็จลุล่วงไปด้วยดี

นายชัยวัช วัฒนมงคลลาภ

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 คำนำ

ความก้าวหน้าทางอิเล็กทรอนิคส์ มีผลกระทบโดยตรงต่อการพัฒนาทางคอมพิวเตอร์และการสื่อสาร จนส่งผลให้ยุคนี้เป็นยุคการสื่อสารข้อมูล ข้อมูลที่ใช้ในการสื่อสารนั้นจะอยู่ในรูปลักษณะต่างๆ เช่น เสียง, ตัวอักษรและภาพ เป็นต้น โดยเฉพาะภาพนั้นเป็นข้อมูลข่าวสารที่เริ่มนิยมใช้ส่งผ่านระบบสื่อสารกันอย่างแพร่หลาย ทั้งนี้เพราะข้อมูลภาพได้ถูกนำมาใช้ในด้านต่างๆ อย่างเช่นทางการแพทย์ ทางการวางแผนการใช้ทรัพยากรธรรมชาติ ทางการตรวจสอบมลภาวะหรือทางการทหารเป็นต้น แต่ปัญหาที่เกิดขึ้นจากการสื่อสารข้อมูลภาพเชิงเลขคือ ปริมาณข้อมูลแต่ละภาพมีจำนวนมหาศาล ซึ่งถ้าหากใช้ส่งผ่านตัวกลางที่เป็นคู่สายโทรศัพท์ธรรมดาจะต้องกินเวลานานทำให้ต้องเสียค่าใช้จ่ายสูงสำหรับการเช่าคู่สายเพื่อส่งข้อมูลเชิงเลข ดังนั้นการสื่อสารข้อมูลภาพเชิงเลขจึงต้องมีการอัดข้อมูล(compression) เพื่อลดจำนวนข้อมูลลง ซึ่งจะมีผลช่วยลดปริมาณของหน่วยความจำที่ใช้เก็บข้อมูลหรือเพื่อลดเวลาในการส่งข้อมูล

จากความสำคัญของการอัดข้อมูลภาพที่มีต่อระบบการสื่อสารนี้ จึงได้มีการทำการวิจัยหาวิธีการอัดข้อมูลกันอย่างแพร่หลาย พบว่าวิธีการแปลงดีสกรีตโคไซน(discrete cosine transform) เป็นพื้นฐานที่ใช้ในการวิจัยพัฒนาการอัดข้อมูลให้ได้มากที่สุด แต่ปัญหาที่เกิดจากการแปลงดีสกรีตโคไซนคือข้อมูลความถี่สูงจะถูกกำจัดทิ้งไปจึงทำให้ภาพผลลัพธ์หลังการแปลงกลับจากการอัดข้อมูลจะขาดความคมชัดบริเวณที่เป็นขอบ(edge) ในภาพจะถูกทำลาย สำหรับการแก้ปัญหาที่เกิดขึ้นนี้จึงได้มีการทำวิธีการแบ่งส่วนภาพ(segmentation) ด้วยทฤษฎีกราฟมาทำการอัดข้อมูลภาพ ซึ่งวิธีการนี้จะยังคงรักษาส่วนที่เป็นขอบในภาพเอาไว้ได้แต่จะใช้เวลาในการประมวลผลสำหรับการอัดข้อมูลมากทั้งนี้เพราะเวลาการคำนวนจะแปรผันโดยตรงกับขนาดภาพยกกำลังสอง ดังนั้นในวิทยานิพนธ์จึงเสนอวิธีการลดเวลาในการประมวลผลจากวิธีการแบ่งส่วนด้วยทฤษฎีกราฟ โดยการแบ่งภาพออกเป็นซับอิมเมจ(subimage) โดยมีเงื่อนไขว่าผลรวมของจำนวนพื้นที่ย่อย(segmented region) ของภาพย่อยหลังการแบ่งจะยังคงรักษาไว้ให้เท่ากับจำนวนพื้นที่ย่อยในภาพใหญ่ ซึ่งในการกำหนดพื้นที่ย่อยของแต่ละซับอิมเมจได้จากการพิจารณาความเบี่ยงเบนมาตรฐาน

1.2 ขอบเขตการวิจัย

ในส่วนของการวิจัยได้เสนอการนำภาพใหญ่ที่ต้องการทำภาพเซ็กเม้นต์มาแบ่งเป็นภาพเล็กๆ โดยนำภาพเล็กมาประมวลผลทีละส่วนและภาพผลลัพท์ที่ได้จะเกิดจากการต่อภาพเล็กๆ เข้าด้วยกัน ซึ่งจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่จะถูกแบ่งให้กับภาพเล็กๆ แต่ละส่วนตามค่าเบี่ยงเบนมาตรฐานของภาพเล็ก รายละเอียดการวิจัยของวิทยานิพนธ์ฉบับนี้ แบ่งออกเป็น 6 บท โดยที่ละบทมีหัวข้อและเนื้อหา ดังต่อไปนี้

บทที่ 1 เป็นบทนำ กล่าวถึงวัตถุประสงค์ และขอบเขตของวิทยานิพนธ์

บทที่ 2 เป็นหลักการของทฤษฎีกราฟ (Graph theory) ที่ใช้ในการทำเซ็กเม้นเตชั่นภาพ (Image segmentation) ที่เรียกว่า Shortest spanning tree (SST) ซึ่งเป็นวิธีที่สามารถที่จะกำหนดจำนวนของเซ็กเม้นต์ในภาพได้ตามต้องการ

บทที่ 3 เป็นการทำเซ็กเม้นเตชั่นภาพที่ใช้เทคนิคที่ปรับปรุงมาจาก Shortest spanning tree ที่เรียกว่า Recursive shortest spanning tree โดยมีการปรับเปลี่ยนค่าลิ้งค์เวทให้สอดคล้องกับค่าเฉลี่ยของรีเจียนที่เชื่อมต่อลิ้งค์เวทนั้นเข้าด้วยกันภายหลังการรวมรีเจียน ซึ่งยังช่วยลดปัญหาของสัญญาณรบกวนที่ถูกแยกออกมาเป็นเซ็กเม้นต์เล็กๆ ทำให้จำนวนรีเจียนที่ได้ไม่มีคุณภาพ และปัญหาของจุดภาพขาวกับจุดภาพดำที่แตกต่างกันมาก อาจถูกนำมารวมอยู่ในเซ็กเม้นต์เดียวกัน

บทที่ 4 กล่าวถึงการปรับปรุงความเร็วของการทำ RSST เซ็กเม้นเตชั่น ที่เกิดปัญหาจากการกระทำซ้ำ เพื่อเปรียบเทียบหาลิ้งค์เวทต่ำสุดระหว่างจุดยอดสองจุดของจำนวนจุดภาพทั้งหมด และการแก้ปัญหาด้วยการลดการกระทำซ้ำเหล่านั้นด้วยการลดจำนวนจุดภาพในการเปรียบเทียบโดยการนำภาพใหญ่มาแบ่งออกเป็นภาพเล็กๆ แล้วนำมาประมวลผลทีละส่วน

บทที่ 5 กล่าวถึงการกำหนดรีเจียนให้กับภาพเล็กๆ เมื่อนำภาพเล็กๆ แต่ละส่วนมาต่อรวมกันเป็นภาพ จะยังคงให้จำนวนรีเจียนของภาพเท่ากับที่กำหนดให้ภาพใหญ่ โดยใช้ค่าเบี่ยงเบนมาตรฐาน (Standard Deviation) ของภาพเล็กๆ แต่ละภาพมาเป็นตัวเปรียบเทียบ เพื่อนำไปกำหนดจำนวนรีเจียนให้กับภาพเล็กๆ เหล่านั้น ซึ่งมีผลรวมของจำนวนรีเจียนในภาพเล็กๆ จะยังรักษาไว้ให้เท่ากับจำนวนรีเจียนที่กำหนดไว้ในภาพใหญ่

บทที่ 6 เป็นบทสรุปรวมของผลงานวิจัยสำหรับทุก ๆ บท

และในส่วนสุดท้ายซึ่งเป็นภาคผนวกกับรายละเอียดของโปรแกรมที่ใช้ในงานวิจัย เพื่อความสะดวกของผู้ที่จะค้นคว้าต่อไป

บทที่ 2
ทฤษฎีกราฟ (Graph Theory)

ในบทนี้เป็นการกล่าวถึงความหมายของคำศัพท์กับรายละเอียดของทฤษฎีกราฟที่จำเป็นต้องนำมาใช้ในการทำการแบ่งส่วนภาพ และการหาช็อตเตสท์สแพนนิ่งทรีของกราฟ

2.1 คำจำกัดความของทฤษฎีกราฟ

ทฤษฎีกราฟเป็นการศึกษาเกี่ยวกับกราฟและการนำมาประยุกต์ใช้งานของกราฟ โดยให้ G = (V,E) เป็นกราฟที่ประกอบด้วยจุดยอด (vertices) V_i เชื่อมต่อกับจุดยอดอื่น ๆ โดยตัวเชื่อม (links) $E_{i,j}$ จะเชื่อมต่อจุดยอด V_i และ V_j เข้าด้วยกัน ในขณะที่น้ำหนัก (weight) ของกราฟเกิดขึ้นจากจุดยอด v_i กับตัวเชื่อมต่อ $e_{i,j}$ จุดยอดแต่ละจุดของกราฟไม่จำเป็นที่จะต้องเชื่อมต่อกับจุดยอดอื่นๆ ทุกจุด แต่ถ้ามีการเชื่อมทุก ๆ จุด ก็จะเป็นกราฟที่สมบูรณ์

กราฟย่อย (Partial graph) เป็นกราฟที่ประกอบด้วยจำนวนจุดยอดเท่ากับกราฟต้นแบบ (original graph) แต่จะมีตัวเชื่อมเป็นซับเซ็ต(subset) ของกราฟต้นแบบ และซับกราฟ(sub graph) เป็นซับเซ็ตของจุดยอดของกราฟต้นแบบ แต่ประกอบด้วยตัวเชื่อมต่อของจุดยอดภายในซับเซ็ตนั้น

ลูกโซ่ (Chain) คือจุดยอดที่ต่อเนื่องกัน โดยจุดยอดแต่ละจุดจะเชื่อมต่อกับจุดยอดถัดไป ด้วยตัวเชื่อมภายในกราฟ

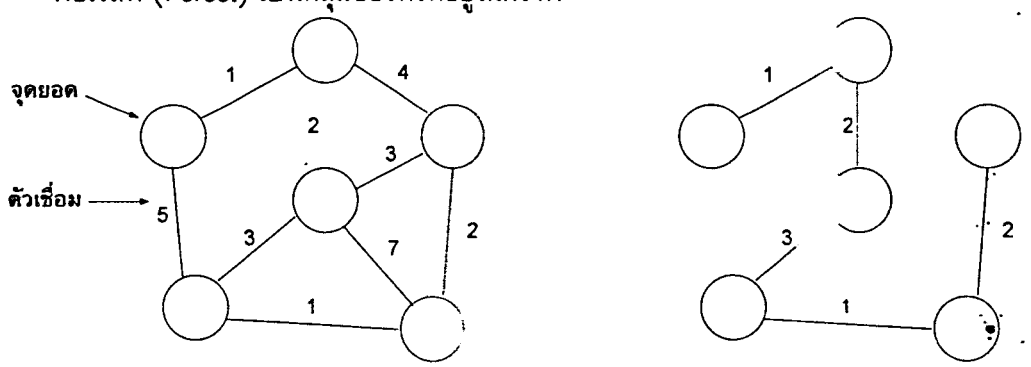
ไซเคิล (Cycle) คือ จุดยอดที่ต่อเนื่องกัน ที่มีจุดยอดท้ายสุดต่ออยู่กับจุดเดียวกับจุดเริ่มต้นของจุดยอดที่ต่อเนื่องนั้น

ทรี (Tree) คือ การต่อจุดยอดที่ต่อเนื่องกันเป็นแบบลูกโซ่หลาย ๆ ตัวเข้าด้วยกัน โดยไม่มีจุดยอดที่ต่อเนื่องใดต่อแบบไซเคิล

สแพนนิ่งทรี (Spanning tree) คือ ทรีที่เป็นกราฟย่อย

ช็อตเตสท์สแพนนิ่งทรี (Shortest spanning tree) หรือ SST เป็นสแพนนิ่งทรี ที่มีผลรวมของน้ำหนักตัวเชื่อมมีค่าต่ำสุด

ฟอเรสท์ (Forest) เป็นกลุ่มของทรีที่อยู่ในกราฟ



a) ตัวอย่างกราฟ b) SST ของกราฟ

รูปที่ 2.1 แสดงตัวอย่างของกราฟ และสแพนนิ่งทรี ที่ผลรวมของน้ำหนักตัวเชื่อมต่ำสุด

2.2 การแปลงข้อมูลภาพไปเป็นกราฟ

การวิเคราะห์ข้อมูลภาพด้วยการใช้ทฤษฎีกราฟ ข้อมูลภาพต้นแบบ (original image) จะต้องถูกเปลี่ยนให้อยู่ในรูปของกราฟ โดยการแปลงจุดภาพแต่ละจุดให้เป็นจุดยอดของกราฟ และน้ำหนักของจุดยอดบนกราฟขึ้นอยู่กับระดับความเข้มของจุดภาพ

ถ้าความเข้มของข้อมูลภาพที่ตำแหน่ง (x,y) คือ $I_{x,y}$ ดังนั้น น้ำหนักของจุดยอดของกราฟ คือ

$$v_i = I_{x,y} \qquad (2.1)$$

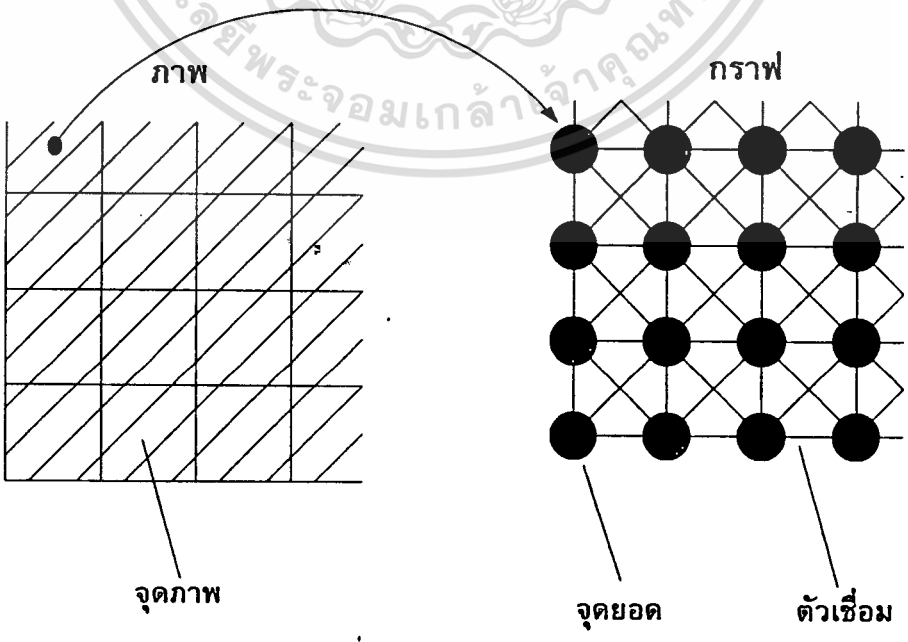
โดยที่ x,y คือตำแหน่งของข้อมูลภาพแต่ละชุดที่ถูกแปลงเป็น i ในลักษณะจุดภาพต่อจุดภาพ

ถ้ากำหนดค่าน้ำหนักตัวเชื่อมของจุดยอดที่ได้มาจากค่าความแตกต่างระหว่าง น้ำหนักของจุดยอดที่เชื่อมต่อกันเป็นค่าสมบูรณ์ (absolute value)

$$e_{i,j} = \left| v_i - v_j \right| \qquad (2.2)$$

ดังนั้น ค่าความแตกต่างระหว่างจุดยอดของกราฟที่เป็นค่าสมบูรณ์นี้จะเป็นการบอกถึงความคล้ายกัน (similarily) ของระดับความเข้มระหว่างจุดภาพที่อยู่ติดกัน

จุดยอดของกราฟแต่ละจุดสามารถที่เชื่อมต่อกับจุดยอดอื่น ๆ ในกราฟ แต่เพื่อลดความยุ่งยากจะใช้การเชื่อมต่อของจุดยอดเฉพาะจุดยอดที่อยู่ใกล้กันที่สุด โดยสามารถใช้การเชื่อมต่อกับจุดใกล้เคียงแบบ 4 ทิศทาง หรือ 8 ทิศทาง รูป 2.2 แสดงการแปลงข้อมูลภาพไปเป็นกราฟ



รูปที่ 2.2 การแปลงข้อมูลภาพไปเป็นกราฟ

2.3 การหาซ็อตเตสท์สแพนนิ่งทรีของกราฟ (SST)

วิธีสร้างซ็อตเตสท์สแพนนิ่งทรี โดยการเชื่อมทรีของกราฟเข้าด้วยกันด้วยตัวเชื่อมที่มีค่าน้ำหนักต่ำสุด เริ่มจากจุดยอดทุก ๆ จุด ทำการหาค่าตัวเชื่อมที่มีค่าต่ำสุด แล้วจึงสร้างทรีขึ้นมาจากจุดยอดที่ต่ำสุดนี้ และการเชื่อมต่อทรีตัวต่อมา ได้จากการหาค่าตัวเชื่อมตัวที่มีค่าต่ำสุดต่อจากตัวแรก การเชื่อมต่อทรีในซ็อตเตสท์สแพนนิ่งทรีเป็นฟอเรสท์ต่อเนื่องกันไปด้วยการต่อตัวเชื่อมตัวใหม่ที่มีค่าต่ำสุด ถึงแม้ว่าตัวเชื่อมที่มีค่าต่ำสุดอยู่ชิดกันต้องไม่ต่อทรีเป็นแบบไซเคิล

การคำนวณในการหาซ็อตเตสท์สแพนนิ่งทรีมีหลายรูปแบบ วิธี Kruskals's algorithm [4] เป็นวิธีที่ให้ความเร็วในการคำนวณเป็นสัดส่วนกับจำนวนของตัวเชื่อมคูณด้วยค่าล๊อก (log) ของจุดยอด (E. log(v)) เมื่อ E คือจำนวนของตัวเชื่อม และ V คือจำนวนของจุดยอดในกราฟ ถ้าเวลาที่ใช้ในการคำนวณมีส่วนสำคัญ วิธีนี้ให้ผลการคำนวณที่เร็ววิธีหนึ่ง

วิธีการหาซ็อตเตสท์สแพนนิ่งทรี Kruskal's algorithm มีดังนี้

(1) เริ่มจากฟอเรสท์ของกราฟที่ยังไม่มีตัวเชื่อม
(2) กระทำซ้ำจากนี้ไป
(3) หาตัวเชื่อมที่มีค่าน้ำหนักต่ำสุดตัวต่อไป
(4) ถ้าได้ตัวเชื่อมที่ต่อกันแล้วไม่เป็นไซเคิลในฟอเรสท์
(5) รวมตัวเชื่อมนี้เข้าไปในฟอเรสท์
(6) ถ้าทำให้เกิดไซเคิล
(7) ตัวเชื่อมที่ทิ้งไป
(8) กระทำซ้ำจนกระทั่งฟอเรสท์กลายเป็น สแพนนิ่งทรีของกราฟ

2.4 การแบ่งส่วนภาพจากสแพนนิ่งทรีของกราฟ (Segmentation from spanning tree)

สแพนนิ่งทรีของกราฟสามารถแบ่งออกเป็นแต่ละเซ็กเม้นต์หรือส่วน ได้จากการตัดตัวเชื่อมของทรีออก ซึ่งฟอเรสท์ประกอบด้วยกลุ่มของทรีที่ต่อเชื่อมกันเป็นสแพนนิ่งทรี เมื่อตัดฟอเรสท์ของสแพนนิ่งทรีออกแทนเป็นแต่ละส่วน โดยสแพนนิ่งทรีแต่ละส่วน เป็นส่วนหนึ่งของภาพที่ได้มาจากการแปลงข้อมูลภาพ จึงสามารถใช้ในการแบ่งส่วนของภาพได้ เพราะจุดยอดในแต่ละทรีของฟอเรสท์มีลักษณะเป็นจุดภาพ (pixels) ถ้า T คือ ทรีในฟอเรสท์, P(T) คือ แต่ละส่วนที่ได้

$$P(T)_i = \begin{cases} 1, \text{ ถ้า } V_i \text{ เป็นสมาชิกของ } T \\ 0, \text{ เมื่อเป็นอย่างอื่น} \end{cases} \qquad (2.3)$$

ถ้ากราฟถูกแยกออกเป็นส่วน ๆ ตามการตัดฟอเรสท์สแพนนิ่งทรีแล้วนำกราฟที่แบ่งออกแต่ละส่วนแปลงกลับไปเป็นภาพ จะทำให้ภาพถูกแบ่งออกเป็นเซ็กเม้นต์ (segment) ตามการตัดทรีใน

ฟอเรสท์ ให้ $S_{x,y}$ คือเซ็กเม้นต์ของภาพ ที่ประกอบด้วยจุดภาพแต่ละจุดในพื้นที่ (region) ของเซ็กเม้นต์นั้น ๆ ซึ่งแต่ละจุดภาพมีค่าความเข้มหรือค่าระดับสีเทา (gray scale) ตามภาพต้นแบบ เมื่อกำหนดให้ P(T) เป็นค่าความเข้มของเซ็กเม้นต์นั้น ๆ โดยนำความเข้มของจุดภาพแต่ละจุดในเซ็กเม้นต์มาหาค่าเฉลี่ย หรือค่าเฉลี่ยน้ำหนักของจุดยอดของทรี จะได้ว่า

$$p(T) = \frac{\sum_i P(T)_i \cdot v_i}{\sum_i P(T)_i} \qquad \text{สำหรับทุกค่าของ } i \qquad (2.4)$$

ถ้า $M_{x,y}$ คือการแแปลง (mapping) จาก (x,y) ไปเป็นทรีในฟอเรสท์ (T)

$$S_{x,y} = p(M_{x,y}) \qquad (2.5)$$

วิธีในการแบ่งส่วนภาพจากซ็อตเตสท์สแพนนิ่นทรี ซึ่งกำหนด N รีเจียน ได้ดังนี้

(1) ทำการแปลงข้อมูลภาพให้อยู่ในรูปของกราฟ

(2) ทำการหา SST ของกราฟ

(3) ทำการตัด SST ของกราฟเป็นจำนวน N-1 ครั้ง จะทำให้ได้ภาพที่ประกอบด้วย N เซ็กเม้นต์

(4) ทำการแปลงกราฟที่ถูกตัดแล้วซึ่งเรียกว่าฟอเรสท์กลับเป็นภาพเซ็กเม้นต์

ดังนั้น ภาพที่ได้หลังจากการแปลงกลับจากภาพต้นแบบ โดยตามการตัดฟอเรสท์ของสแพนนิ่งทรีจะถูกแบ่งออกเป็นพื้นที่ย่อย ๆ ที่ไม่ซ้อนทับกันและทุก ๆ พื้นที่ย่อยยังคงเป็นส่วนประกอบของภาพ อยู่เช่นเดิม ส่วนค่าความเข้มของจุดภาพในพื้นที่ย่อยนั้น ๆ ขึ้นอยู่กับค่าเฉลี่ยของจุดภาพในพื้นที่ย่อยจากภาพต้นแบบ

บทที่ 3
RSST เซ็กเมนเตชั่น

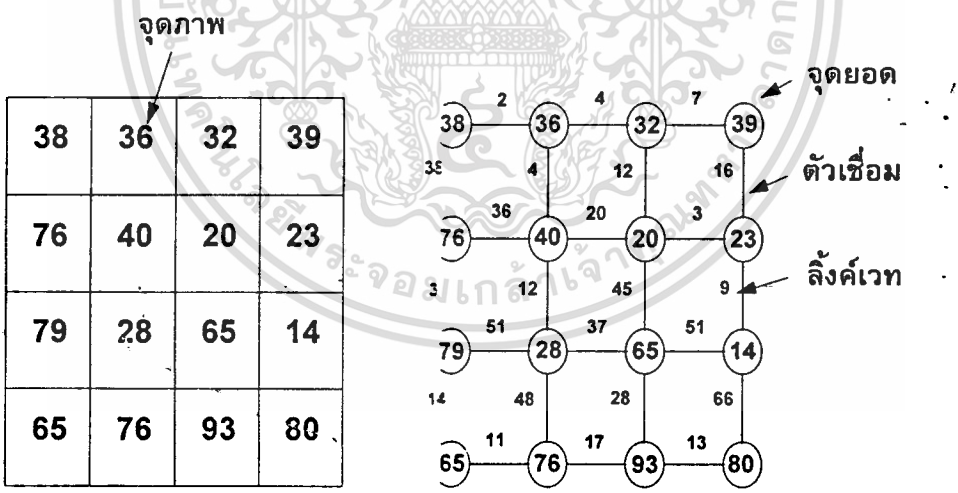
ทฤษฎีกราฟที่กล่าวมาใช้ในการทำเซ็กเม้นเตชั่นภาพ โดยอาศัยค่าความเข้มหรือค่าสเกลสีเทา (gray scale) ของจุดภาพนั้นแล้วแยกจุดภาพตามความใกล้เคียงออกเป็นรีเจียน ซึ่งต้องคำนึงถึงความถูกต้องของรีเจียนที่ได้ตามโครงสร้างของภาพต้นแบบ และยังต้องคำนึงถึงความยุ่งยากในวิธีการที่ไม่ทำให้เสียเวลาในการคำนวณ หลักการและขั้นตอนในการทำเซ็นเม้นเตชั่นภาพในบทนี้ ใช้พื้นฐานทางทฤษฎีกราฟที่ให้ขอบเขตพื้นที่มีความถูกต้องแม่นยำสูง (Region accuracy)

3.1 การแบ่งส่วน โดยอาศัยพื้นฐานทางทฤษฎีกราฟ

ขั้นตอนในการจำแนกภาพ (Image Classification) โดยวิธีการแบ่งภาพที่อาศัยพื้นฐานทฤษฎีกราฟ มีขั้นตอนดังนี้

(1) กำหนดค่า เธร็ชโฮลด์ ความเป็นเนื้อเดียว (ε) ที่ต้องการให้กับพื้นที่บริเวณภาพ

(2) สร้างกราฟโดยให้จุดภาพแต่ละจุดเป็นจุดยอดของกราฟ (Vertex) ที่มีน้ำหนักของจุดยอดตรงกับระดับสีเทาของภาพและมีตัวเชื่อม (Link) ที่ใช้เชื่อมต่อจุดยอดมีลักษณะเป็น 4 ทิศทาง ดังรูป 3.1



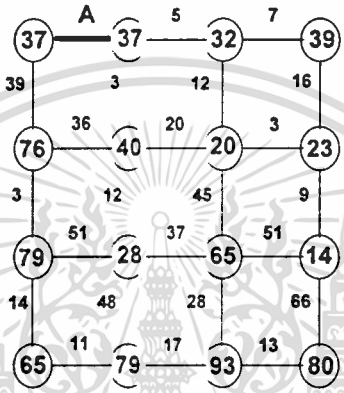
(a) ภาพ (b) กราฟ

รูปที่ 3.1 การแปลงข้อมูลตัวเลขในรูป (a) ให้เป็นกราฟในรูป (b)

(3) หาค่าลิ้งค์เวท (Link weight) ด้วยค่าสมบูรณ์ของความแตกต่างระหว่างน้ำหนักจุดยอดที่อยู่ข้างเคียง ดังรูปที่ 3.1 (b) ซึ่งหมายถึงการวัดความเหมือนหรือความคล้ายคลึงระหว่างจุดภาพหรือจุดยอดข้างเคียง

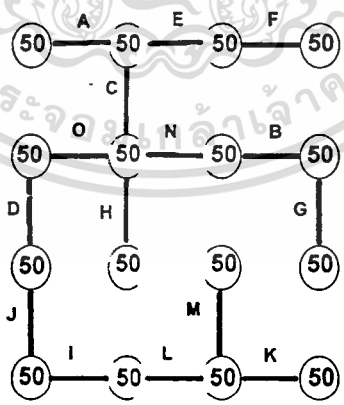
(4) นำตัวเชื่อมต่อของกราฟออก โดยเริ่มจากลิ้งค์เวทต่ำสุดที่หาได้จากค่าความแตกต่างของระดับสีเทาของจุดยอดสองจุดที่ต่อถึงกันด้วยลิ้งค์เวทนี้

- ถ้าค่าความแตกต่างน้อยกว่า หรือเท่ากับค่าเธร็ชโฮลด์ความเป็นเนื้อเดียวกันที่กำหนดไว้ ให้ทำการรวบรวมจุดยอด หรือพื้นที่ทั้งสองเข้าด้วยกัน และตัดตัวเชื่อมที่มีลิ้งค์เวทต่ำสุดนี้ออก ค่าจุดยอดใหม่ที่ได้จะถูกเปลี่ยนเป็นค่าเฉลี่ยระดับสีเทาของจุดยอดสองจุดนี้ และค่าลิ้งค์เวทข้างเคียงจะถูกเปลี่ยนเป็นค่าใหม่ ตัวอย่างผลลัพธ์ที่ได้ในรูป 3.1 (b) แสดงในรูปที่ 3.2



รูปที่ 3.2 ผลลัพธ์จากการหาลิ้งค์เวทแรกที่มีค่าต่ำสุด ε = 20

(5) กระทำซ้ำในขั้นตอนที่ (4) เพื่อหาลิ้งค์เวทต่ำสุดตัวต่อไปจนกระทั่งถึงตัวสุดท้ายผลลัพธ์สุดท้ายที่ได้แสดงดังในรูปที่ 3.3



รูปที่ 3.3 ทรีของกราฟได้จากการคิดลิ้งค์สุดท้าย

จากรูปที่ 3.4 เป็นผลลัพธ์ที่ได้หลังการหาลิ้งค์เวทตัวสุดท้าย จากการกำหนดให้ค่า ε = 20 โดยรูปที่ 3.4(a) แสดงให้เห็นถึงบริเวณที่ถูกแบ่งด้วยเลเบล และรูปที่ 3.4(b) เป็นผลลัทธ์แสดงขอบเขตบนข้อมูลตัวเลขที่สอดคล้องกับรูป 3.1(a)

1	1	1	1
2	1	3	3
2	1	4	3
2	2	5	5

38	36	32	39
76	40	20	23
79	28	65	14
65	76	93	80

(a) เลเบล (Label) ของพื้นที่ต่าง ๆ (b) พื้นที่ต่าง ๆ ที่ถูกล้อมรอบขอบเขตด้วยเส้นปะ

รูปที่ 3.4 แสดงผลลัพธ์ที่ได้จากขั้นตอนที่ 3 และ 4 เพื่อกำหนดให้ $\varepsilon = 20$

3.2 การแบ่งส่วนภาพจาก รีเดอร์ซีฟช็อตเตสท์สแพนนิ่งทรี (RSST)

จากขั้นตอนในการสร้างช็อตเตสท์สแพนนิ่งทรีของกราฟนั้น ได้จากการจัดเรียงค่าลิ้งค์เวทจากค่าต่ำสุดไปยังค่าสูงสุดในการจัดเรียงค่าลิ้งค์เวทแต่ละครั้ง หมายถึงการรวมเอารีเจียนทั้งสองที่ถูกเชื่อมต่อตัวลิ้งค์เวทนั้นเข้าด้วยกัน ภายหลังการรวมรีเจียนแล้วลิ้งค์เวทอื่นๆ ที่เชื่อมต่อกับรีเจียนนั้น มิได้มีการปรับเปลี่ยนค่าให้สอดคล้องกับค่าเฉลี่ยของรีเจียน ขั้นตอนดังกล่าวทำให้เกิดผลเสียบางประการ คือ ถ้าหากเกิดสัญญาณรบกวนในภาพ ก่อนทำการเซ็กเม้นท์เตชั่น จะทำให้จุดภาพที่เป็นสัญญาณรบกวนถูกแยกออกมาเป็นเซ็กเม้นท์เล็ก ๆ และอีกประการหนึ่ง ถ้าลิ้งค์เวทสุดท้ายที่เหลือเชื่อมต่อสองรีเจียนอยู่ โดยสองรีเจียนนั้นควรจะต้องแยกจากกัน เพราะมีความแตกต่างกันมาก แต่กลับต้องถูกรวมกัน เช่น จุดภาพขาวกับจุดภาพดำที่แตกต่างกันมากกลับถูกนำมารวมอยู่ในเซ็กเม้นท์เดียวกัน ถ้าหากจุดภาพทั้งสองถูกเชื่อมต่อด้วยกลุ่มจุดภาพที่มีค่าระดับความเข้มสว่างเรียงไล่กันไป ดังนั้นถ้าความแตกต่างของระดับความเข้มสว่างของจุดภาพที่ใกล้กันจะมีค่าแตกต่างกันน้อยทำให้ค่าลิ้งค์เวทที่ได้จากกลุ่มจุดภาพดังกล่าวมีค่าน้อย จึงมีผลทำให้ต้องรวมจุดภาพขาวกับจุดภาพดำอยู่ในเซ็กเม้นท์เดียวกัน

การปรับปรุงการแบ่งส่วนภาพด้วยช็อตเตสท์สแพนนิ่งทรีให้มีประสิทธิภาพ ต้องมีการปรับลิ้งค์เวททุกครั้งเมื่อมีการรวมรีเจียนที่ถูกเชื่อมต่อด้วยลิ้งค์เวทต่ำสุดแล้วจึงเรียงค่าลิ้งค์เวทใหม่เพื่อหาลิ้งค์เวทต่ำสุดตัวต่อไป โดยใช้วิธีของรีเดอร์ซีฟช็อตเตสท์สแพนนิ่งทรี (Recursive shortest spanning tree) ตัวอย่างง่ายๆ ในการหา N รีเจียน กับภาพที่มี M จุดภาพ คือ

1) เก็บข้อมูลภาพต้นแบบไว้ในภาพชั่วคราว
2) กระทำซ้ำ(I) เมื่อ I = M-1 จนกระทั่งเท่ากับ N
3) สร้างเซ็กเม้นท์เตชั่นให้มี I รีเจียน โดยใช้ SST ของกราฟจากข้อมูลภาพชั่วคราว
4) เก็บภาพเซ็กเม้นท์เตชั่น ไว้ในภาพชั่วคราว

ตัวอย่างข้างต้นนี้แสดงให้เห็นถึงข้อดีของหลักการเนื่องจาก ขั้นตอนของการกระทำซ้ำเพื่อการแบ่งส่วนภาพจะทำให้เกิดเป็นรีเจียน โดยค่าความเข้มของจุดภาพได้จากค่าเฉลี่ย(mean) ของจุดภาพในรีเจียนนั้นๆ จากภาพต้นแบบหรือกล่าวได้ว่าค่าความเข้มของจุดภาพจากภาพต้นแบบจะถูกเปลี่ยนทุกๆ จุดภาพ ซึ่งเป็นไปได้ที่จุดภาพหนึ่งจุดภาพจะมีผลทำให้ค่าความเข้มของจุดภาพในรีเจียน มีค่าไกล้เคียงกับรีเจียนที่อยู่ค้างเคียง และค่าลิ้งค์เวทระหว่างรีเจียนก็มีผลต่อจุดภาพทั้งสองรีเจียนที่ต่อถึงกัน

เนื่องจากทุกๆ จุดภาพในแต่ละรีเจียนจะมีค่าระดับสีเทาเท่ากัน ซึ่งจะกล่าวได้ว่าจุดยอดของกราฟจุดหนึ่งสามารถใช้แสดงแทนตลอดทั้งรีเจียน ถ้าหากตัดซ็อตเตสท์สแพนนิ่งทรีออกเป็นฟอร์ เรส (เพื่อแบ่งเป็นรีเจียนต่างๆ) แต่ละทรีจะกลายเป็นจุดยอดจุดหนึ่ง ดังนั้นในการแปลงกราฟกลับไปเป็นภาพจะเป็นการแปลงแบบหนึ่งจุดไปยังหลายจุด และแต่ละ Partition $P(i)_{x,y}$ ถูกกำหนดโดยจุดยอด V_i ของกราฟจะเป็นไปตามสมการ

$$P(i)_{x,y} = \begin{cases} 1, \text{ ถ้า } V_i \text{ แปลงไปเป็น } (x,y) \\ 0, \text{ เมื่อเป็นอย่างอื่น} \end{cases} \qquad (3.1)$$

ค่าเวทของจุดยอดของกราฟสามารถกำหนดได้จากสมการ

$$v_i = \frac{\sum_{x,y} P(i)_{x,y} \cdot I_{x,y}}{\sum_{x,y} P(i)_{x,y}} \qquad \text{สำหรับทุก ๆ ค่าของ x และ y} \qquad (3.2)$$

$I_{(x,y)}$ ค่าระดับสีเทาของจุดภาพที่ตำแหน่ง x,y

การแปลงข้อมูลภาพไปเป็นกราฟในตอนเริ่มแรกนั้นเหมือนกับวิธีแรกที่กล่าวมาแล้ว แต่ในขั้นตอนต่อๆ ไปโครงสร้างของกราฟจะถูกทำให้เปลี่ยนแปลงตลอดเมื่อมีการคำนวณซ้ำ(recursion) ในตอนเริ่มต้นจุดภาพแต่ละจุดจะถูกพิจารณาเป็นส่วนของภาพแยกกันเป็นแต่ละรีเจียน จากนั้นจะทำการรวมเอาสองรีเจียนเข้าด้วยกัน การรวมจุดยอดจะเริ่มจากจุดยอด V_i และ V_j ที่มีค่าลิ้งค์เวทต่ำสุด จะถูกรวมเข้าด้วยกันได้เป็นจุดยอดใหม่ V_k ซึ่งเป็นตัวกำหนดค่าเฉลี่ยของรีเจียนใหม่ดังสมการ

$$v_k = \frac{\sum_{x,y} (P(i)_{x,y} \cdot I_{x,y} + P(j)_{x,y} \cdot I_{x,y})}{\sum_{x,y} (P(i)_{x,y} + P(j)_{x,y})} \qquad \text{สำหรับทุกค่าของ x,y} \qquad (3.3)$$

ลิ้งค์ต่ำสุดที่เชื่อมต่อจุดยอด V_i และ V_j จะถูกตัดออกและเก็บเอาไว้เพื่อนำมาใช้ในขั้นต่อนการแปลงกราฟกลับเป็นภาพ ลิ้งค์อื่นทุกลิ้งค์ที่มีจุดยอดใกล้เคียงกับจุดยอด V_i และ V_j จะถูกคำนวณค่าของลิ้งค์เวทใหม่ ดังนั้นลิ้งค์ใหม่ใดที่มีค่าต่ำสุดก็จะถูกตัดออกแล้วคำนวนหาลิ้งค์เวทใหม่อีก ซึ่งจะกระทำซ้ำจนกระทั่งไม่มีลิ้งค์เหลืออยู่

จากการตัดจุดยอดออกแต่ละจุดและเก็บลิ้งค์นั้น จะเกิดขึ้นทุกครั้งของการกระทำซ้ำ ลิ้งค์ที่อยู่ตัวถัดไปจะไม่สามารถต่อรวมกับลิ้งค์ที่เก็บไปแล้ว สิ่งเหล่านี้มาจากสแพนนิ่งทรีของกราฟต้นแบบที่ไม่จำเป็นในซ็อตเดสท์สแพนนิ่งทรี แต่ใช้ในการทำรีเคอร์ซีพซ็อตเดสท์สแพนนิ่งทรี เซ็กเม้นท์เตชั่นของภาพได้ตามจำนวน N รีเจียน ที่กำหนดโดยการตัดทรีของกราฟเป็นจำนวนเท่ากับ N-1 ลิ้งค์

ขั้นตอนและวิธีการของการทำเซ็กเม้นท์เตชั่น โดยใช้รีเคอร์ซีพซ็อตเดสท์สแพนนิ่งทรีของกราฟมีจำนวน N รีเจียน

(1) แปลงข้อมูลภาพไปอยู่ในรูปจุดยอดของกราฟ
(2) กระทำซ้ำในขณะที่มีจุดยอดมากกว่าหนึ่งจุด
(3) หาค่าเวทของลิ้งค์ที่มีค่าต่ำสุดตัวต่อไป
(4) เก็บลิ้งค์ของตำแหน่งนั้นไว้
(5) รวมจุดยอดทั้งสองที่ต่อด้วยลิ้งค์ตัวนี้
(6) คำนวณค่าเวทของจุดยอดใหม่และของสิ่งต่าง ๆ ที่เชื่อมต่อกับจุดยอดนี้
(7) ตัดลิ้งค์ที่ซ้ำกันออก
(8) สร้างสแพนนิ่งทรีจากสิ่งที่ได้เก็บเอาไว้แล้ว
(9) ทำการตัดสแพนนิ่งทรีของกราฟเป็นจำนวนเท่ากับรีเจียนของภาพลบหนึ่ง (N-1)
(10) แปลงกราฟกลับเป็นภาพเซ็กเม้นต์

3.3 คุณสมบัติของ RSST เซ็กเม้นเตชั่น

การเปรียบเทียบวิธีการทำ RSST เซ็กเม้นเตชั่นกับ SST เซ็กเม้นเตชั่น แสดงดังตารางที่ 3.1 ทั้ง 2 วิธีนี้สามารถที่จะกำหนดจำนวนของรีเจียนได้ตามความต้องการ แต่คุณสมบัติของ RSST เซ็กเม้นเตชั่น จะมีข้อแตกต่างจาก SST เซ็กเม้นเตชั่น ในด้านของความถูกต้องของรีเจียน(Region accuracy) ที่ดีกว่า เพราะวิธี RSST เซ็กเม้นเตชั่นนั้นใช้ข้อมูลส่วนใหญ่ของภาพที่มีการปรับเปลี่ยนตัวลิ้งค์เวทตลอดทุกครั้งที่มีการสร้างทรี ดังนั้นเวลาในการคำนวณจะเสียไปในการหาลิ้งค์เวทใหม่นี้ ซึ่งทำให้วิธี SST เซ็กเม้นเตชั่นใช้เวลาในการคำนวณที่น้อยกว่า เนื่องจากใช้ข้อมูลภาพบางส่วน แต่จำนวนของรีเจียนที่ได้จาก SST เซ็กเม้นเตชั่นบางรีเจียนอาจจะเกิดจากสัญญาณรบกวนที่ถูกแยกออกเป็นรีเจียน และถ้าสัญญาณรบกวนนั้นมีมากความถูกต้องของรีเจียนและค่าความเข้มของรีเจียนที่ได้จากค่าเฉลี่ยของพื้นที่อาจผิดพลาด เนื่องจากในการทำ SST เซ็กเม้นท์เตชั่นนั้นจะไม่มีการปรับค่าลิ้งค์เวทที่ได้หลังจากสร้างทรีที่ต่างจาก RSST เซ็กเม้นเตชั่น

คุณสมบัติ	SST	รีเคอร์ซีฟ SST
กำหนดจำนวนรีเจียน	ได้	ได้
ความปลอดจากสัญญาณรบกวน	แย่	ดี
ความถูกต้องของรีเจียน	ดี	ดีเยียม
ใช้ข้อมูลของจุดภาพข้างเคียง	บางส่วน	ทั้งหมด

ตารางที่ 3.1 เปรียบเทียบคุณสมบัติของ RSST และ SST เซ็กเม้นเตชั่น

3.4 ตัวอย่างของการแบ่งส่วนด้วย RSST

รูปที่ 3.5(b),(c),(d) แสดงภาพที่ผ่านการทำ RSST เซ็กเม้นเตชั่น จากภาพต้นแบบในรูปที่ 3.5(a) โดยใช้ข้อมูลทั้งหมดของภาพ(Gobal Information) ในการรวมเข้าเป็นรีเจียน ซึ่งผลจากการแบ่งส่วนภาพด้วยจำนวนรีเจียนที่ต้องการเท่ากัน ในการทำ RSST เซ็กเม้นเตชั่นจะแบ่งระดับของสีเทาและแยกแยะระดับความเข้มระหว่างรีเจียนได้ดีกว่าในการทำ SST เซ็กเม้นเตชั่น จะเห็นว่าระดับความเข้มของสีเทาของรีเจียนที่ได้จากข้อมูลส่วนใหญ่ของภาพจะมีค่ามากกว่าระดับสีเทาของรีเจียนจากข้อมูลภาพบางส่วน และรูปที่ 3.5(e) แสดงจำนวนรีเจียนที่เกิดขึ้นใหม่จากการเพิ่มการกำหนดจำนวนรีเจียนจาก 19 เป็น 20 รีเจียน โดยพื้นที่ของรีเจียนเดิมไม่มีการเคลื่อนย้ายหรือเปลี่ยนรูปร่าง เพรียงแต่มีการแบ่งแยกออกเป็นรีเจียน ใหม่

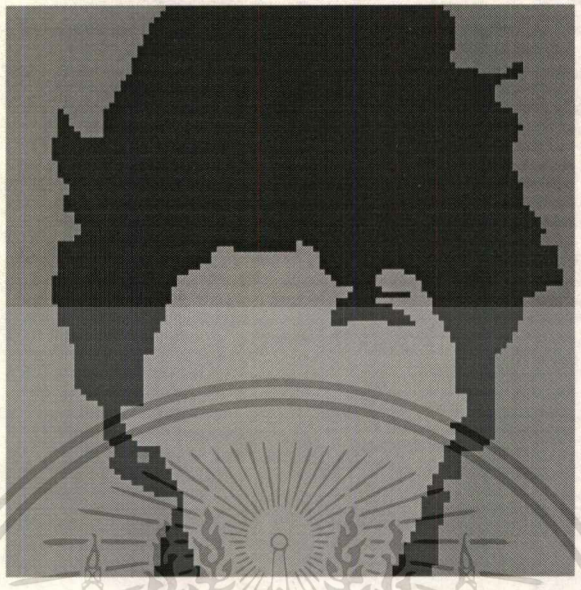
a) ภาพต้นแบบขนาด 100 x 100 จุดภาพ

รูปที่ 3.5 แสดงภาพตัวอย่างของ RSST เซ็กเม้นเตชั่น

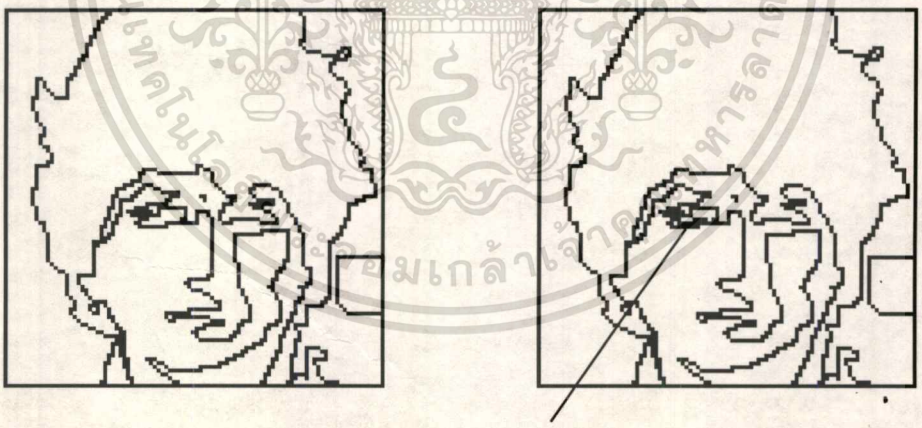
b) กำหนด 826 รีเจียน

c) กำหนด 20 รีเจียน

รูปที่ 3.5 (ต่อ) แสดงภาพตัวอย่างของ RSST เซ็กเม้นเตชั่น



d) กำหนด 5 รีเจียน



e) เปรียบเทียบภาพกำหนด 19 และ 20 รีเจียน แสดงรีเจียนที่เกิดขึ้นใหม่

รูปที่ 3.5 (ต่อ) แสดงภาพตัวอย่างของ RSST เซ็กเม้นเตชั่น

บทที่ 4
การปรับปรุงความเร็วของการทำ RSST เซ็กเม้นเตชั่น

จากขั้นตอนที่กล่าวมา สแพนนิ่งทรีเป็นส่วนของกราฟที่ได้มาจากการแปลงข้อมูลภาพ โดยการเชื่อมทรีของฟอเรสท์เข้าด้วยกัน ด้วยตัวเชื่อมที่มีน้ำหนักต่ำสุดเริ่มขึ้นจากจุดยอดทุกๆ จุดทำการหาค่าตัวเชื่อมที่มีค่าต่ำสุด แล้วจึงทำการสร้างทรีขึ้นมาจากจุดยอดเหล่านี้ ด้วยการเชื่อมจุดยอดของกราฟที่มีค่าน้ำหนักของตัวเชื่อมน้อยที่สุดขึ้นเป็นทรีก่อน แล้วจึงเชื่อมทรีเหล่านี้เข้าด้วยกันเป็นซ็อคเตสท์สแพนนิ่งทรีด้วยตัวเชื่อมที่มีค่าน้ำหนักต่ำสุดต่อไปจนกระทั่งจุดยอดทุกจุดบนกราฟกลายเป็นสแพนนิ่งทรีของกราฟ

ในการจัดเรียงค่าต่ำสุดในแต่ละครั้งเพื่อหาตัวเชื่อมที่ต่ำที่สุด หมายถึงการรวมเอาจุดภาพทั้งสองที่ถูกเชื่อมต่อด้วยลิ้งค์เวทนั้นเข้าด้วยกัน หลังจากการรวมจุดภาพแล้วลิ้งค์เวทอื่นๆ ที่เชื่อมต่อกับจุดภาพนั้นจะทำการปรับค่าลิ้งค์เวทที่เชื่อมต่อนั้นใหม่ให้สอดคล้องกับค่าเฉลี่ยของจุดภาพที่ได้จากการกำจัดค่าลิ้งค์เวทต่ำสุด คือเป็นการรวมจุดภาพเข้ามาเป็นส่วนหนึ่งของรีเจียนนั้น ดังนั้นในการรวมจุดภาพแต่ละครั้งมีการปรับปรุงค่าลิ้งค์เวทใหม่ตลอดเวลา วิธีนี้จึงถูกเรียกว่า ริเคอร์ซีฟซ็อตเตสท์สแพนนิ่งทรี (RSST)

4.1 ปัญหาของการแบ่งส่วนภาพด้วย RSST

ในการหาลิ้งค์เวทต่ำสุดระหว่างจุดยอดสองจุดของจำนวนจุดภาพทั้งหมด ต้องใช้การเปรียบเทียบข้อมูลของค่าลิ้งค์เวทกับตัวข้างเคียง จึงจะได้ค่าลิ้งค์เวทที่ต่ำสุดแล้วตัดลิ้งค์นั้นออก และหาค่าลิ้งค์เวทต่ำสุดตัวต่อไป โดยจำนวนครั้งของการกระทำซ้ำในการหาลิ้งค์เวทต่ำสุดนี้ ขึ้นอยู่กับจำนวนจุดภาพ ถ้าภาพนั้นมีขนาด X * Y จุดภาพจะมีลิ้งค์อยู่จำนวน [X(Y-1) + Y(X-1)] ลิ้งค์ ดังนั้น จำนวนครั้งในการหาลิ้งค์เวทต่ำสุด ต้องใช้การเปรียบเทียบ (If statement) ลิ้งค์ตัวแรกกับตัวที่สองไปจนถึงตัวสุดท้าย และวนกลับเพื่อเปรียบเทียบตัวที่สองกับตัวที่สามจนถึงตัวสุดท้าย กระทำซ้ำเช่นนี้จนเหลือคู่สุดท้าย เพื่อเปรียบเทียบ ซึ่งจะต้องใช้การเปรียบเทียบ (If statement) เท่ากับ

$$\beta = \frac{Link \,.\, (Link + 1)}{2} - 1 \quad \text{ครั้ง} \qquad (4.1)$$

เมื่อ β คือ จำนวนครั้งของการเปรียบเทียบ เพื่อหาค่าลิ้งค์เวทต่ำสุด

Link คือ จำนวนลิ้งค์ของจุดภาพนั้น มีค่าเท่ากับ [X(Y-1) + Y(X-1)]

โดย X และ Y คือจำนวนจุดภาพในส่วนแนวนอนและแนวตั้งตามลำดับ

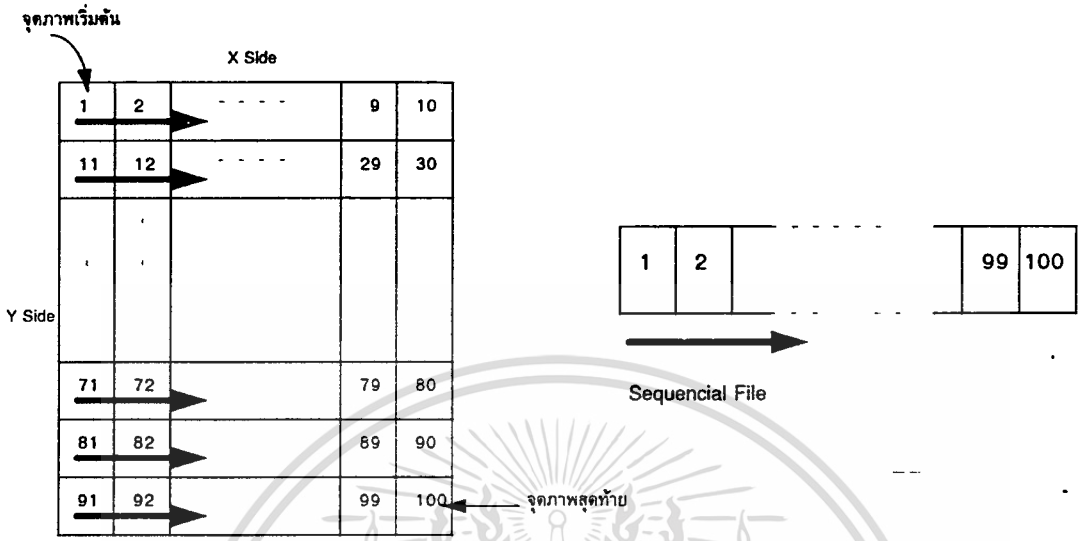
ด้วยขนาดของจุดภาพที่นำมาหาค่าลิ้งค์เวทจะแปรผันโดยตรง กับจำนวนครั้งในการเปรียบเทียบ ถ้านำภาพที่มีขนาดใหญ่นั้นมาแบ่งออกเป็นส่วน ๆ ให้มีขนาดเล็กกว่าภาพเดิม และนำภาพแต่ละส่วนไปหาลิ้งค์เวทต่ำสุดจากวิธีดังกล่าว (4.1) ผลรวมของจำนวนครั้งแต่ละส่วนรวมกัน จะมีค่าน้อยกว่าเมื่อเทียบกับการใช้ภาพใหญ่ทั้งภาพแสดงดังตารางที่ 4.1 ถ้ามีการแบ่งภาพออกเป็นซับอิมเมจให้มากเท่าไร โดยมีผลทำให้ได้จำนวนจุดภาพในแต่ละส่วนน้อยลง ก็จะได้ผลรวมของจำนวนครั้งที่ใช้เปรียบเทียบน้อยลงเท่านั้น ดังตัวอย่าง เช่น มีภาพต้นแบบที่มีขนาดของจุดภาพด้านแนวนอนและแนวตั้ง คือ 100 * 100 จุดภาพ เมื่อนำมาหาลิ้งค์เวทต่ำสุดจะต้องใช้จำนวนครั้งในการเปรียบเทียบ 196,029,899 ครั้ง และถ้านำภาพนั้นแบ่งออกเป็น 4 ส่วน จะได้ภาพใหม่ที่มีขนาดของภาพเหลือเพียง 50 * 50 จุดภาพจำนวน 4 ภาพย่อย ซึ่งในภาพแต่ละส่วนใช้จำนวนครั้งในการหาลิ้งค์เวท 12,007,449 ครั้ง และเมื่อหาจำนวนลิ้งค์เวทครบทั้ง 4 ภาพย่อย จะใช้จำนวนครั้งรวมในการเปรียบเทียบ คือ 48,029,796 ครั้ง แต่ถ้านำภาพเดิมแบ่งส่วนให้น้อยกว่าเดิมเป็น 16 ส่วน โดยให้แต่ละส่วนมีขนาดของภาพ 25 * 25 จุดภาพเท่ากันทั้ง 16 ภาพย่อย ในแต่ละส่วนจะใช้จำนวนครั้งในการเปรียบเทียบ 720,599 ครั้ง และเมื่อหาลิ้งค์เวทครบทั้ง 16 ภาพย่อย จะได้จำนวนครั้งรวมเป็น 11,529,584 ครั้ง

ส่วนในการแบ่ง	ขนาดของรูปภาพ n * n	β ของแต่ละส่วน	β รวมทั้งภาพ
1	100 * 100	196,029,899	196,029,899
4	50 * 50	12,007,449	48,029,796
16	25 * 25	720,599	11,529,584

ตารางที่ 4.1 แสดงจำนวนครั้งในการเปรียบเทียบเพื่อหาลิ้งค์เวทต่ำสุด

4.2 หลักการแบ่งภาพใหญ่ให้เป็นภาพเล็กๆ

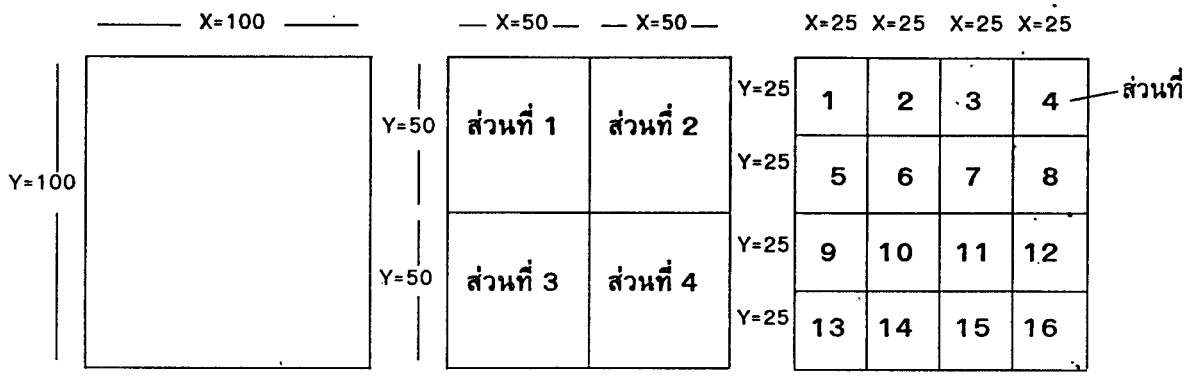
โดยปกติข้อมูลภาพที่ใช้ในการประมวลผลภาพ มักเป็นสัญญาณดิจิตอลที่มีระดับความแตกต่างของค่าระดับสีเทา(Gray scale) แต่ละจุดภาพขึ้นอยู่กับจำนวนบิท (bits) ของข้อมูลภาพที่เก็บไว้ในแผ่นจานแม่เหล็กถาวรหรือเทปแม่เหล็ก ซึ่งข้อมูลภาพที่ใช้ในการวิจัยนี้เป็นข้อมูลภาพขนาด 8 บิท ที่มีลักษณะการเก็บเป็นแบบแฟ้มลำดับ(Sequential file) คือ จำนวนจุดภาพทั้งหมดของข้อมูลภาพ เริ่มจากจุดภาพแถวบนสุดซ้าย เป็นจุดภาพแรก และไปทางขวามือตัวต่อไปเป็นจุดภาพที่สอง จนกระทั่งบนสุดขวามือเป็นตัวสุดท้ายของจุดภาพในแกน X (x_size) จากนั้นกลับมาเริ่มแถวที่สองของแกน Y (y_size) ซ้ายมือสุดเป็นตัวต่อมา ซึ่งข้อมูลภาพตัวสุดท้ายที่อ่านได้จากแฟ้มลำดับเป็นจุดภาพจุดสุดท้ายที่อยู่ทางด้านล่างขวาสุด ดังตัวอย่างรูป 4.1



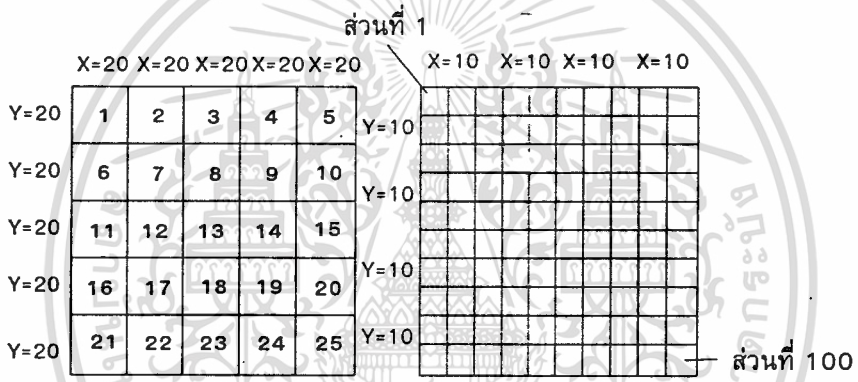
รูปที่ 4.1 แสดงการจำลองการเก็บข้อมูลภาพในการประมวลผลภาพ

ขั้นตอนการแบ่งภาพออกเป็นส่วน ๆ เริ่มจากการอ่าน ข้อมูลภาพในแผ่นจานแม่เหล็ก (Disk Storage) เก็บลงในหน่วยความจำชั่วคราว (RAM) ตามจำนวนจุดภาพที่เก็บในแฟ้มลำดับ โดยจำนวนของข้อมูลมีขนาดเท่ากับข้อมูลภาพด้านแกน X คูณกับข้อมูลภาพด้านแกน Y (X_Pixel * Y_Pixel) ข้อมูลภาพที่อยู่ในหน่วยความจำชั่วคราว จะอยู่เรียงกันจากจุดภาพเริ่มต้นไปจนกระทั่งจุดภาพสุดท้าย และจำนวนส่วนของการแบ่ง (X_Part , Y_Part) แต่ละด้านต้องมีขนาดของจุดภาพเท่ากันทุกส่วนในแกนนั้นๆ เช่น จุดภาพในแกน X และ Y มีจำนวนทั้งหมด 100 จุดภาพ จำนวนส่วนในแกน X และ Y แบ่งได้เป็น 2,4,5 และ 10 ส่วน ซึ่งทำให้จำนวนจุดภาพในแกน X และ Y ของแต่ละส่วนที่แบ่งเหลือเพียง 50,25,20 และ 10 จุดภาพตามลำดับ ดังนั้น ภาพใหญ่หนึ่งภาพจะถูกแบ่งออกเป็นภาพเล็กๆ 4,16,25 และ 100 ภาพตามลำดับ จำนวนจุดภาพทั้งหมดของภาพใหญ่จากเดิมมีขนาด 10,000 จุดภาพ จะลดลงเหลือเพียง 2,500 , 625 , 100 และ 25 จุดภาพ ดังรูปที่ 4.2

เมื่อกำหนดจำนวนส่วนของการแบ่ง (X_Part และ Y_Part) ให้กับแกน X และ Y สอดคล้องกับจำนวนจุดภาพรวมของแกนนั้น (X_Pixel และ Y_Pixel) ทำให้ภาพใหญ่ถูกแบ่งออกเป็นภาพเล็ก ๆ ขึ้นอยู่กับจำนวนส่วนของแกน X คูณกับจำนวนส่วนของแกน Y ภาพเล็กแต่ละส่วนนี้ จะนำมาผ่านการทำ RSST เซ็กเม้นเตชั่นทีละภาพ โดยอ่านข้อมูลภาพจากภาพใหญ่ที่เก็บไว้ในหน่วยความจำชั่วคราว ตามจำนวนจุดภาพที่อยู่ในพื้นที่ของภาพเล็กแต่ละส่วนตามลำดับ ดังรูป 4.3 แสดงโฟลว์ชาร์ตของการอ่านข้อมูลภาพเล็กแต่ละส่วนจากภาพใหญ่ผ่านกระบวนการแบ่งภาพด้วย RSST จากนั้นวนกลับมาอ่านข้อมูลภาพของส่วนต่อไป จนครบตามจำนวนส่วนของการแบ่งเป็นภาพเล็กเหล่านั้น



a) ภาพต้นแบบ b) แบ่งแกน x,y ออกเป็น 2 ส่วน c) แบ่งแกน x,y ออกเป็น 4 ส่วน



d) แบ่งแกน x,y ออกเป็น 5 ส่วน c) แบ่งแกน x,y ออกเป็น 10 ส่วน

รูปที่ 4.2 แสดงการแบ่งภาพออกเป็นส่วน ๆ

4.3 ตัวอย่างและผลการทดลอง

ในวิทยานิพนธ์นี้ทำการทดลองโดยใช้ภาพที่มีขนาด 100 * 100 จุดภาพ เมื่อผ่านกระบวนการแบ่งส่วนภาพที่อาศัยทฤษฎีกราฟ โดยใช้โปรแกรมภาษา C ทำงานบนเครื่องคอมพิวเตอร์ IBM PC/AT 80486SX-50 ทำการประมวลผลภาพ และจับเวลาที่ใช้ในการเปรียบเทียบเพื่อการหาทรี ซึ่งแบ่งเป็นการนำภาพทั้ง 100 * 100 จุดภาพ มาประมวลผลครั้งเดียวกับการแบ่งเป็น 50 * 50 จุดภาพ มาประมวลผลที่ส่วนรวม 4 ส่วน และการแบ่งเป็น 25 * 25 จุดภาพ มาประมวลผลทีละส่วนรวม 16 ส่วน แสดงดังตารางที่ 4.2 เปรียบเทียบผลรวมของจำนวนเวลาที่ใช้ในโปรแกรมเพื่อหาทรีจากการแบ่งส่วน ซึ่งการแบ่งภาพออกเป็น 16 ส่วนจะใช้เวลาในการประมวลลดลง 11.59 วินาที และภาพที่แบ่ง 4 ส่วน ใช้เวลาลดลง 7.41 วินาที เทียบกับการนำภาพใหญ่ทั้งภาพมาประมวลผลรูปที่ 4.4 แสดงภาพหลังการประมวลผลด้วยการกำหนดผลรวมของจำนวนรีเจียนในภาพใหญ่เท่ากับ 96 รีเจียน โดยภาพที่แบ่ง 4

ส่วนกำหนดรีเจียนให้ส่วนละ 24 รีเจียนและภาพที่แบ่ง 16 ส่วนกำหนดจำนวนรีเจียนให้ส่วนละ 6 รีเจียน โดยผลรวมของจำนวนรีเจียนในภาพที่แบ่ง 4 ส่วนและ 16 ส่วนมีค่าเท่ากับการกำหนดรีเจียนให้กับภาพใหญ่ คือ 96 รีเจียน

จำนวนส่วนในการแบ่ง	เวลาที่ใช้ (วินาที)
1	70.10
4	62.69
16	58.51

ตารางที่ 4.2 เปรียบเทียบเวลาที่ใช้ในการประมวลผลหาทรีของการแบ่งภาพออกเป็นส่วน ๆ

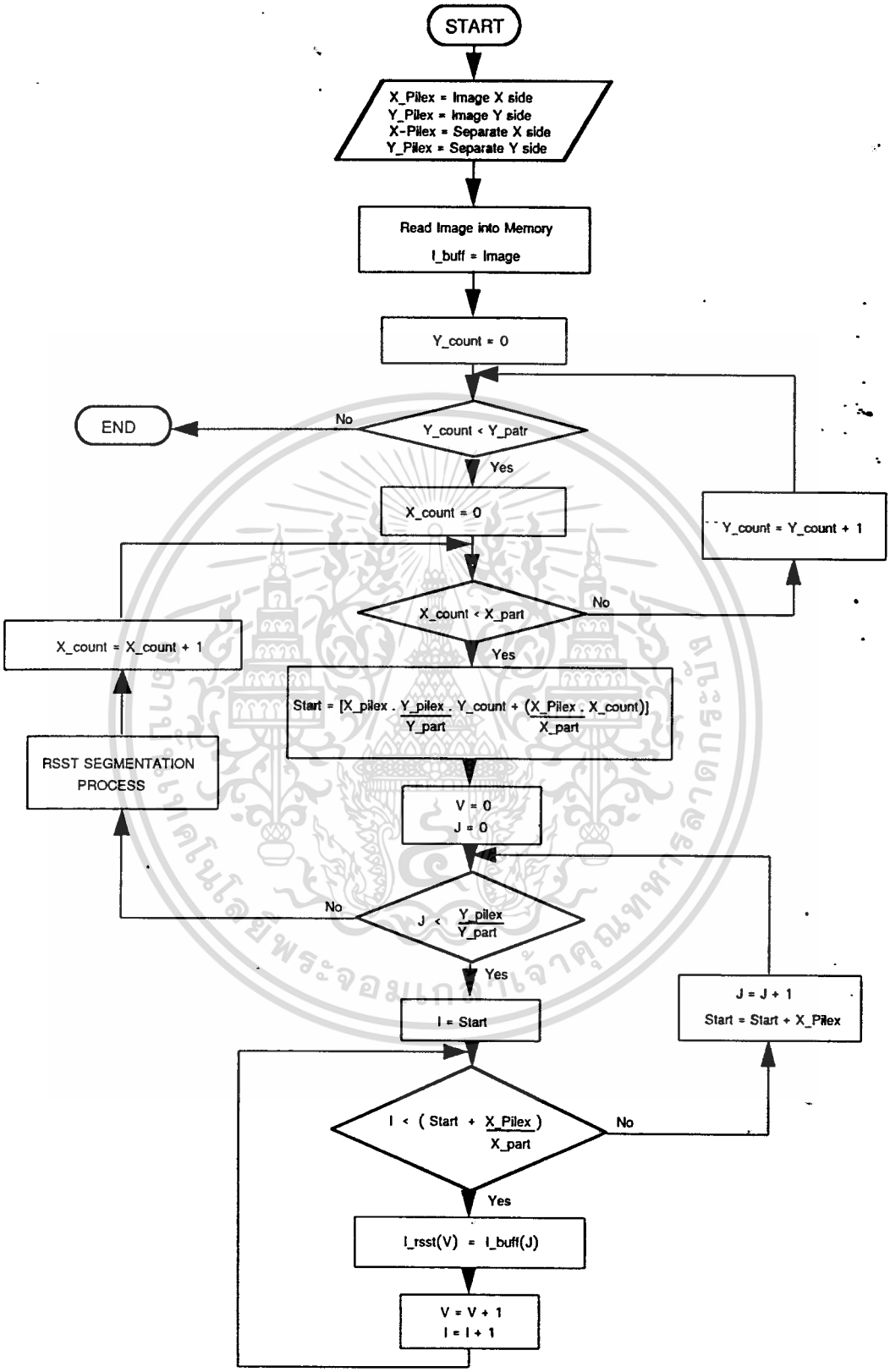
เมื่อนำภาพเซ็กเม้นต์หลังจากการประมวลผลมาเปรียบเพื่อหาความเหมือน(similarity) ของภาพระหว่างภาพต้นแบบกับภาพเซ็กเม้นต์ที่แบ่งออกเป็นส่วนเล็กๆ โดยหา mean squared error (MSE) ของภาพ ที่ได้จากผลรวมของผลต่างระหว่างภาพต้นแบบกับภาพที่นำมาเปรียบเทียบยกกำลังสองแล้วหารด้วยจำนวนจุดภาพทั้งหมด ดังสมการ(4.2) และแสดงผลการเปรียบเทียบดังตารางที่ 4.3

$$MSE = 1/n \sum_{i=0}^{n} (x_i - y_i)^2 \qquad (4.2)$$

เมื่อ MSE คือ mean squared error
n คือ จำนวนจุดภาพทั้งหมด
x_i คือ จุดภาพต้นแบบ ณ.ตำแหน่งที่ i
y_i คือ จุดภาพที่นำมาเปรียบเทียบ ณ.ตำแหน่งที่ i

ส่วนในการแบ่ง	MSE
1	224.2
4	200.2
16	217.2

ตารางที่ 4.3 แสดงผลการเปรียบเทียบหาค่าความเหมือนของภาพ(MSE) ระหว่างภาพต้นแบบกับภาพเซ็กเม้นต์ที่แบ่งออกเป็นส่วนเล็กๆ



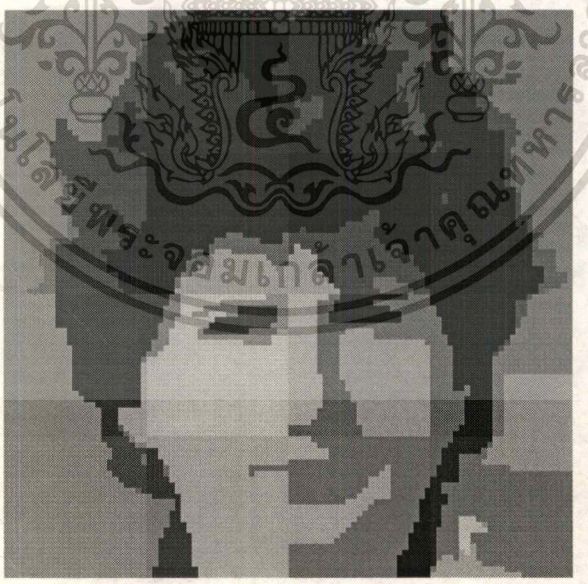
รูปที่ 4.3 โฟลว์ชาร์ตการแบ่งภาพใหญ่ออกเป็นภาพเล็ก ๆ

a) ภาพต้นแบบขนาด 100 x 100 จุดภาพ

b) ภาพที่ไม่มีการแบ่งส่วน

รูปที่ 4.4 ภาพหลังการทำเซ็กเม้นเตชั่นโดยกำหนดจำนวนรีเจียนให้กับทั้งภาพเท่ากับ 96 รีเจียน

c) ภาพจากการแบ่ง 4 ส่วน



d) ภาพจากการแบ่ง 16 ส่วน

รูปที่ 4.4 (ต่อ) ภาพหลังการทำเซ็กเม้นเตชั่นโดยกำหนดจำนวนรีเจียน
ให้กับทั้งภาพเท่ากับ 96 รีเจียน

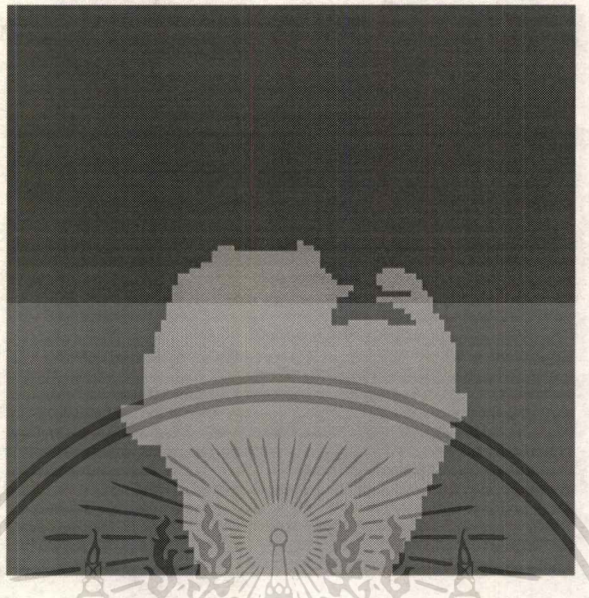
บทที่ 5
การกำหนดรีเจียนให้กับภาพเล็กๆ

จากการนำภาพถ่ายทั้งภาพมาแบ่งออกเป็นส่วนเล็กๆ เพื่อนำมาแยกการทำ RSST เซ็กเม้นเตชั่น เนื่องจากเวลาที่ใช้ในการประมวลผลจะน้อยกว่าการนำภาพใหญ่ทั้งภาพมาประมวลผล แต่ภาพเซ็กเม้นต์ที่ได้แต่ละส่วนเมื่อนำมาต่อรวมกันเป็นภาพใหญ่จะยังคงได้จำนวนรีเจียนของภาพเท่ากับที่กำหนด ซึ่งภาพเล็กๆ แต่ละส่วนต้องมีการแยกการกำหนดจำนวนรีเจียนให้แต่ละภาพตามความสำคัญของวัตถุในภาพหรือตามรายละเอียดของวัตถุ ถ้ากำหนดจำนวนรีเจียนให้แต่ละภาพโดยแบ่งเท่าๆ กันทุกภาพจะทำให้ภาพบางส่วนที่มีการกำหนดจำนวนรีเจียนไม่เหมาะสมกับรายละเอียดของวัตถุในภาพ คือภาพบางส่วนที่มีรายละเอียดของวัตถุในภาพมากควรกำหนดจำนวนรีเจียนให้มากกว่า ภาพส่วนที่มีรายละเอียดของวัตถุในภาพน้อย ดังนั้นต้องมีการพิจารณาการกำหนดจำนวนรีเจียนให้กับภาพเล็กๆ แต่ละภาพจากจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ที่ไม่มีการแบ่งส่วน

5.1 ความสำคัญในการกำหนดรีเจียน

การกำหนดรีเจียนเป็นการให้รายละเอียดของภาพหลังจากการทำเซ็กเม้นเตชั่นภาพที่มีจำนวนรีเจียนมาก จะมีรายละเอียดของวัตถุในภาพได้มากกว่าภาพที่มีจำนวนรีเจียนน้อย ถ้านำภาพที่มีรายละเอียดของวัตถุส่วนอยู่บริเวณกลางภาพและส่วนบริเวณด้านริมของภาพเป็นแบ็คกราวด์ (background) เมื่อกำหนดจำนวนรีเจียนให้กับภาพนั้นด้วยจำนวนที่น้อยจะทำให้ภาพเซ็กเม้นต์แบ่งเป็นพื้นที่บริเวณกลางภาพตามรูปร่างของวัตถุตัดกับพื้นที่บริเวณส่วนริมของภาพเท่านั้น โดยรายละเอียดของวัตถุในส่วนกลางและส่วนริมของภาพจะเพิ่มขึ้นได้เมื่อกำหนดจำนวนรีเจียนให้ภาพนั้นมากขึ้นตามลำดับ ดังตัวอย่างรูปที่ 5.1 แสดงความแตกต่างของการกำหนดรีเจียนที่เพิ่มขึ้น

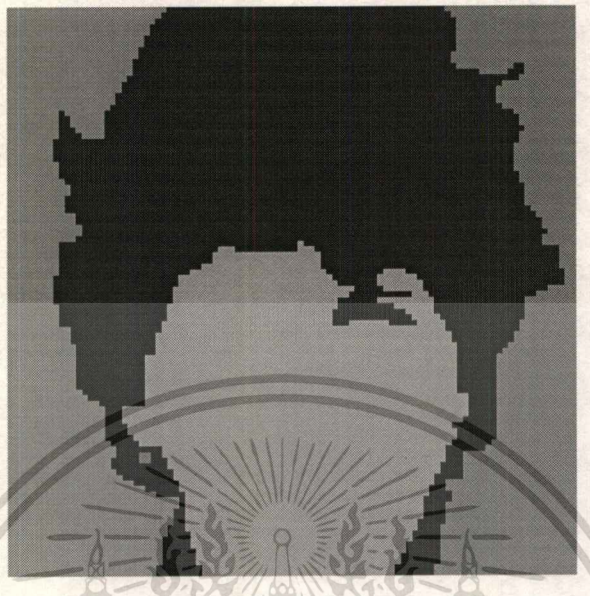
จำนวนรีเจียนที่กำหนดให้กับภาพเซ็กเม้นต์เพื่อเน้นส่วนสำคัญในภาพให้เด่นชัดออกมาเป็นส่วนๆ ตามจำนวนที่เพิ่มขึ้น ขึ้นอยู่กับโครงสร้างของภาพและองค์ประกอบของวัตถุในภาพ การกำหนดจำนวนของรีเจียนจึงขึ้นอยู่กับความต้องการให้ภาพเซ็กเม้นต์นั้นมีรายละเอียดของวัตถุในภาพมากน้อยเพียงไร ซึ่งถ้านำภาพเซ็กเม้นต์ไปใช้ในการลดข้อมูลภาพโดยการเข้ารหัสพื้นที่ของภาพ จำนวนรีเจียนที่น้อยจะทำให้อัตราการลดข้อมูลภาพมากแต่ต้องคำนึงถึงรายละเอียดในภาพว่าพอเพียงต่อการสื่อความหมายในภาพ กล่าวคือ การเข้ารหัสพื้นที่อาศัยขอบ(edge) ของเซ็กเม้นต์ที่มีทิศทางตามรูปร่างของแต่ละเซ็กเม้นต์ โดยเปลี่ยนรูปร่างของเซ็กเม้นต์ให้กลายเป็นข้อมูลด้านทิศทางของขอบเซ็กเม้นต์บวกกับความเข้มของเซ็กเม้นต์นั้น ดังนั้นข้อมูลภาพที่ทำการเข้ารหัสพื้นที่จะมีขนาดเล็กกว่าข้อมูลภาพเซ็กเม้นต์เดิมที่มีขนาดใหญ่เท่ากับจำนวนจุดภาพ ในภาพ



(a) กำหนด 2 รีเจียน

(b) กำหนด 3 รีเจียน

รูปที่ 5.1 แสดงภาพการกำหนดรีเจียนด้วยขนาดที่แตกต่างกัน

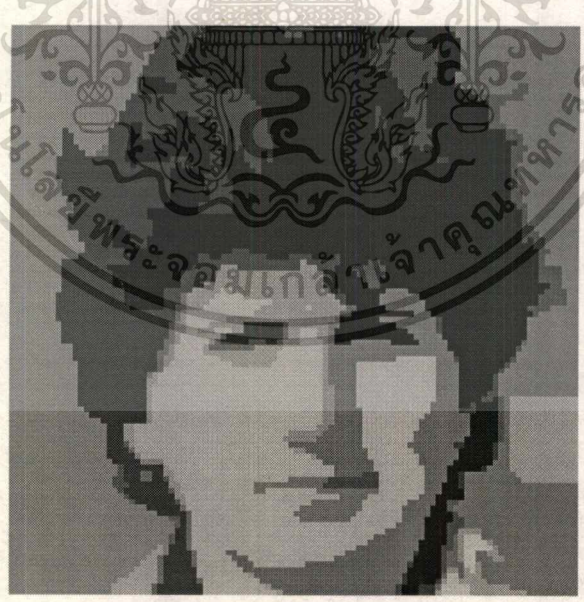


(c) กำหนด 5 รีเจียน

(d) กำหนด 15 รีเจียน

รูปที่ 5.1 (ต่อ) แสดงภาพการกำหนดรีเจียนด้วยขนาดที่แตกต่างกัน

(e) กำหนด 50 รีเจียน



(f) กำหนด 100 รีเจียน

รูปที่ 5.1(ต่อ) แสดงภาพการกำหนดรีเจียนด้วยขนาดที่แตกต่างกัน

5.2 ค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานของภาพ

การหาค่าที่จะนำไปใช้เป็นตัวแทนของข้อมูลชุดนั้นในการสื่อความหมายให้เข้าใจเกี่ยวกับลักษณะของข้อมูลควรจะเป็นค่าที่อยู่ตรงกลางๆ ของชุดข้อมูล การหาค่ากลางของข้อมูลมีวิธีหาได้หลายวิธีแต่ละวิธีต่างก็มีข้อดีข้อเสียแตกต่างกันออกไปและมีความเหมาะสมในการใช้งานแต่ละอย่างไม่เหมือนกันขึ้นอยู่กับลักษณะข้อมูลและวัตถุประสงค์ของผู้ใช้ข้อมูลชนิดนั้นๆ ค่ากลางของข้อมูลที่นิยมใช้ คือ ค่าเฉลี่ยเลขคณิต (mean), ค่ามัธยฐาน (median) และฐานนิยม (mode) ค่ากลางเหล่านี้ใช้ในทางสถิติเพื่อวิเคราะห์ข้อมูล โดยเฉพาะทฤษฎีทางสถิติใช้กันมากในการลดข้อมูล (data reduction) ทำให้เหลือจำนวนข้อมูลน้อยลงโดยที่ยังสื่อความหมายได้เพียงพอเหมือนกับข้อมูลเดิม และยังใช้ในการประมวลผลภาพเพื่อวิเคราะห์ข้อมูลภาพ ค่ากลางแต่ละตัวนี้ให้คุณสมบัติที่แตกต่างกันในวิทยานิพนธ์นี้เลือกค่ากลางที่เป็นค่าเฉลี่ยเลขคณิต ($\bar{X}$) เพราะเป็นการหาค่ากลางที่นำตัวเลขของข้อมูลทุกตัวมาใช้ ซึ่งต่างจากการหาค่ากลางชนิดอื่นที่ใช้เพียงตำแหน่งของข้อมูลหรือความถี่ของข้อมูลเท่านั้น

ค่าเฉลี่ยเลขคณิตใช้วัดความสว่างของภาพสเกลสีเทา เพื่อเปรียบเทียบภาพสองภาพ ภาพที่มีค่าเฉลี่ยเลขคณิตมากจะมีความสว่างในภาพมากเสมอ ซึ่งในบางกรณีภาพสองภาพนั้นอาจมีความแตกต่างกันมาก แต่มีค่าเฉลี่ยเลขคณิตที่ใกล้เคียงกัน ทั้งนี้เพราะ ค่าเฉลี่ยเลขคณิตที่ได้จากผลรวมของข้อมูลภาพแต่ละจุดภาพและหารด้วยจำนวนของจุดภาพในข้อมูลภาพนั้น การหาค่าเฉลี่ยเลขคณิต มีสมการดังนี้ :-

$$\bar{X} = \frac{1}{n} \cdot \sum_{i=0}^{n} I_i \qquad (5.1)$$

โดย $\bar{X}$ = ค่าเฉลี่ยเลขคณิต (ที่ได้จากข้อมูลภาพ)

I_i = ค่าความเข้มของภาพ ณ ตำแหน่งที่ i

n = จำนวนของ จุดภาพทั้งหมด

ตัวอย่างการหาค่าเฉลี่ยเลขคณิต ถ้ามีข้อมูลข้อมูลภาพทั้งหมด 10 จุดภาพ จะสามารถหาค่าเฉลี่ยของจุดภาพทั้ง 10 จุดภาพ โดยใช้สมการที่ (5.1) ได้ดังนี้

I	23	61	32	18	79	54	80	36	26	9
i	1	2	3	4	5	6	7	8	9	10

โดย I = ค่าความเข้มของภาพ

i = ตำแหน่งจุดภาพ

ตารางที่ 5.1 แสดงตัวอย่างข้อมูลภาพทั้ง 10 จุดภาพ

จากตารางที่ 5.1 เป็นการแสดงตัวอย่างข้อมูลภาพทั้ง 10 จุดภาพ จะสังเกตได้ว่าจุดภาพที่ตำแหน่ง i=1 จะมีค่าความเข้มของจุดภาพ (I_1) เท่ากับ 23 และตำแหน่งที่ i = 2 มีค่าความเข้มของจุดภาพ (I_2) เท่ากับ 61 และจุดภาพสุดท้าย คือ ตำแหน่งที่ i = 10 มีค่าความเข้มของจุดภาพ (I_{10}) เท่ากับ 9 เมื่อได้ค่าความเข้มของจุดภาพทั้ง 10 จุดภาพแล้ว สามารถนำมาแทนค่าในสมการที่ 5.1 ดังนั้นค่าเฉลี่ย ($\bar{X}$) ของข้อมูลภาพทั้ง 10 จุดภาพ คือ 41.8

ค่าเบี่ยงเบนมาตรฐาน (Standard Deviation) ของภาพสามารถแยกถึงคุณลักษณะของข้ อมูลภาพที่มีค่าสเกลสีเทาคล้ายกันได้ดีกว่าค่าเฉลี่ยเลขคณิตเพียงอย่างเดียว และค่าเบี่ยงเบนมาตรฐานที่มีค่ามากยังบ่งบอกถึงความแตกต่างของข้อมูลภาพได้ดีกว่าค่าเฉลี่ยเลขคณิต แต่อย่างไรก็ตามสัญญาณรบกวนในภาพมีจำนวนมากจะทำให้เกิดผลกระทบต่อค่าเบี่ยงเบนมาตรฐาน การคำนวนหาค่าเบี่ยงเบนมาตรฐานแสดงได้ดังสมการ :-

$$\sigma = \left[\left(\frac{1}{n} \cdot \sum_{i=0}^{n} I_i^2 \right) - \bar{X}^2 \right]^{1/2} \qquad (5.2)$$

โดย σ คือ ค่าเบี่ยงเบนมาตรฐาน

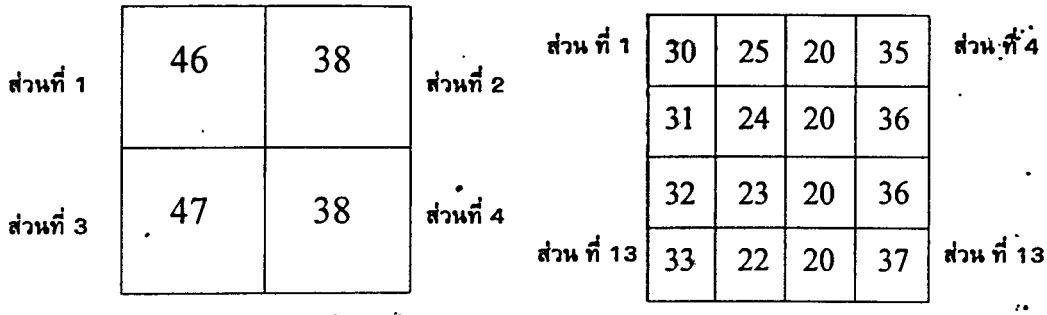
$\bar{X}$ คือ ค่าเฉลี่ยเลขคณิต

n คือ ข้อมูลภาพทั้งหมด

I_i คือ ค่าความเข้มของภาพ ณ ตำแหน่งที่ i

ดังนั้น ค่าเบียเบนมาตรฐานของข้อมูลภาพในตารางที่ 5.1 คือ 23.94

จากบทที่ 4 ได้กล่าวถึงการแบ่งส่วนภาพด้วยการแบ่งภาพออกเป็นส่วนเล็กๆ โดยนำภาพแต่ละส่วนมาประมวลผลทีละภาพ ในขั้นตอนการแปลงสแพนนิ่งทรีของภาพกลับเป็นกราฟต้องมีการกำหนดจำนวนรีเจียนให้กับภาพใหญ่ทั้งภาพก่อนแล้วจึงนำจำนวนรีเจียนนั้นมาแบ่งให้กับภาพเล็กแต่ละภาพ จากวิธีในบทที่ 4 จะกำหนดจำนวนรีเจียนให้กับภาพเล็กแต่ละภาพมีจำนวนเท่าๆ กันทุกภาพ ซึ่งทำให้ภาพผลลัพธ์ที่ได้ในแต่ละภาพบางภาพไม่มีความจำเป็นต้องกำหนดจำนวนรีเจียนให้มากเท่ากับที่กำหนด และบางภาพก็กำหนดจำนวนรีเจียนน้อยเกินไป ซึ่งทำให้รายละเอียดของภาพบางส่วนหลังการทำภาพเซ็กเม้นต์ไม่เหมาะสม ดังนั้นในบทนี้จึงนำเสนอวิธีการแก้ไขการกำหนดจำนวนรีเจียนที่มากเกินไปและน้อยเกินไปของแต่ละส่วน โดยนำภาพแต่ละภาพมาหาค่าเบี่ยงเบนมาตรฐาน ซึ่งค่าเบี่ยงเบนมาตรฐานที่ได้ยังไม่สามารถใช้กำหนดจำนวนรีเจียนให้กับภาพแต่ละส่วนได้จะต้องนำค่าเบี่ยงเบนมาตรฐานของแต่ละภาพมาใช้เปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ เพื่อหาค่าการกำหนดจำนวนรีเจียนให้กับภาพเล็กแต่ละภาพเหล่านั้น ตัวอย่างเช่นต้องทำเซ็กเม้นเตชั่นภาพขนาด 100 * 100 จุดภาพ กำหนดจำนวนรีเจียนให้กับภาพเท่ากับ 96 รีเจียน โดยแบ่งภาพออกเป็น 4 ส่วน และ 16 ส่วน ในการทำเซ็กเม้นเตชั่นจากตัวอย่างในรูป 4.4 (a),(b) สามารถหาค่าเบี่ยงเบนมาตรฐานของภาพแต่ละภาพดังรูป 5.2



รูปที่ 5.2 แสดงค่าเบี่ยงเบนมาตรฐานแต่ละส่วนจากรูป 4.4(c),(d)
โดยมีขนาดของภาพ 100 * 100 จุดภาพ

เมื่อได้ค่าเบี่ยงเบนมาตรฐานของภาพแต่ละภาพแล้วนำมาเปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ โดยนำจำนวนรีเจียนของภาพใหญ่หารด้วยผลรวมของค่าเบี่ยงเบนมาตรฐานของภาพเล็กทุกภาพและคูณด้วยค่าเบี่ยงเบนมาตรฐานของภาพเล็กส่วนที่ต้องการกำหนดจำนวนรีเจียน ดังสมการ

$$\text{จำนวนรีเจียนภาพเล็ก} = \frac{\text{จำนวนรีเจียนภาพใหญ่} * \text{ค่าเบี่ยงเบนมาตรฐานของภาพเล็ก}}{\text{ผลรวมค่าเบี่ยงเบนมาตรฐานของภาพเล็กทุกภาพ}} \qquad (5.3)$$

ตัวอย่างรูป 5.2 (a) ภาพจากการแบ่ง 4 ส่วน ค่าเบี่ยงเบนมาตรฐานของภาพในส่วนที่ 1, ส่วนที่ 2, ส่วนที่ 3 และส่วนที่ 4 มีค่าเท่ากับ 46, 38, 47 และ 38 ตามลำดับ ค่าผลรวมของค่าเบี่ยงเบนมาตรฐานของภาพเล็กทุกส่วนมีค่าเท่ากับ 169 และจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ คือ 96 รีเจียน ดังนั้นจำนวนรีเจียนที่ต้องกำหนดให้กับภาพส่วนที่ 1 คำนวนได้จากสมการที่ 5.3 ได้ดังนี้

$$\text{จำนวนรีเจียนภาพส่วนที่ 1} = \frac{96 * 46}{169}$$

$$= 26.13 \text{ รีเจียน}$$

จากตัวอย่างการคำนวณหาจำนวน รีเจียน ของภาพส่วนที่ 1 จากภาพรูป 5.2 (a) มีค่าเท่ากับ 26.13 รีเจียน จากนั้นคำนวณหาจำนวนรีเจียนของภาพในส่วนอื่นๆ โดยใช้วิธีการคำนวณเหมือนกับการหาจำนวนรีเจียนให้กับภาพส่วนที่ 1 แต่เปลี่ยนค่าบียงเบนมาตรฐานของภาพเล็กตามภาพที่ต้องการ

ซึ่งหลังจากการคำนวณหาจำนวนรีเจียนทุกๆ ส่วนแล้วจะได้ค่าการกำหนดจำนวนรีเจียน แสดงดังรูป 5.3 ที่ใช้ภาพจากการแบ่ง 4 ส่วนและ 16 ส่วน

26.13	21.59
26.69	21.59

a) ภาพจากการแบ่ง 4 ส่วนรวม 96 รีเจียน

6.49	5.41	4.32	7.57
6.70	5.19	4.32	7.78
6.92	4.98	4.32	7.78
7.14	4.76	4.32	8.00

b) ภาพจากการแบ่ง 16 ส่วนรวม 96 รีเจียน

รูปที่ 5.3 แสดงจำนวนรีเจียนที่กำหนดให้กับภาพเล็กๆ แต่ละส่วนมีผลรวมเท่ากับรีเจียนที่กำหนดให้กับภาพใหญ่มีค่า 96 รีเจียน

จากรูป 5.3 จะสังเกตุได้ว่ารีเจียนที่คำนวณได้ของภาพแต่ละส่วนนั้นเป็นเลขจำนวนจริงที่มีทั้งเลขจำนวนเต็มและเลขจุดทศนิยม ซึ่งค่าที่ใช้ในการกำหนดจำนวนรีเจียนให้กับภาพต้องเป็นเลขจำนวนเต็ม ดังนั้นหลังจากการคำนวณหาจำนวนรีเจียนให้กับภาพแต่ละส่วนจากสมการ 5.3 และแสดงจำนวนรีเจียนของแต่ละภาพในรูป 5.3 ที่เป็นเลขจำนวนจริง ต้องนำค่าจำนวนรีเจียนที่มีค่าจุดทศนิยม มาทำการปรับค่าให้เป็นเลขจำนวนเต็มทั้งหมด ถ้าใช้หลักการปัดจุดทศนิยมแบบทั่วไป คือ เลขจำนวนที่ได้มีจุดทศนิยมมากกว่าหรือเท่ากับจุด 5 ให้ทำการปัดเศษขึ้น เช่น ค่าที่คำนวณได้เท่ากับ 41.5 ให้ทำการปัดเศษขึ้น มีค่าเท่ากับ 42 และถ้าค่าที่คำนวณได้มีจุดทศนิยมน้อยกว่าจุด 5 ให้ปัดเศษลง เช่น ค่าที่คำนวณได้เท่ากับ 41.4 ให้ทำการปัดเศษลง ค่าที่ได้เท่ากับ 41 ดังรูป 5.4 แสดงจำนวนรีเจียนที่เป็นค่าจำนวนเต็มที่ไม่มีจุดทศนิยม โดยใช้การปัดจุดทศนิยมแบบทั่วไป จากข้อมูลรูป 5.3

26	22
27	22

a) ภาพจากการแบ่ง 4 ส่วนรวม 97 รีเจียน

6	5	4	8
7	5	4	8
7	5	4	8
7	5	4	8

b) ภาพจากการแบ่ง 16 ส่วนรวม 95 รีเจียน

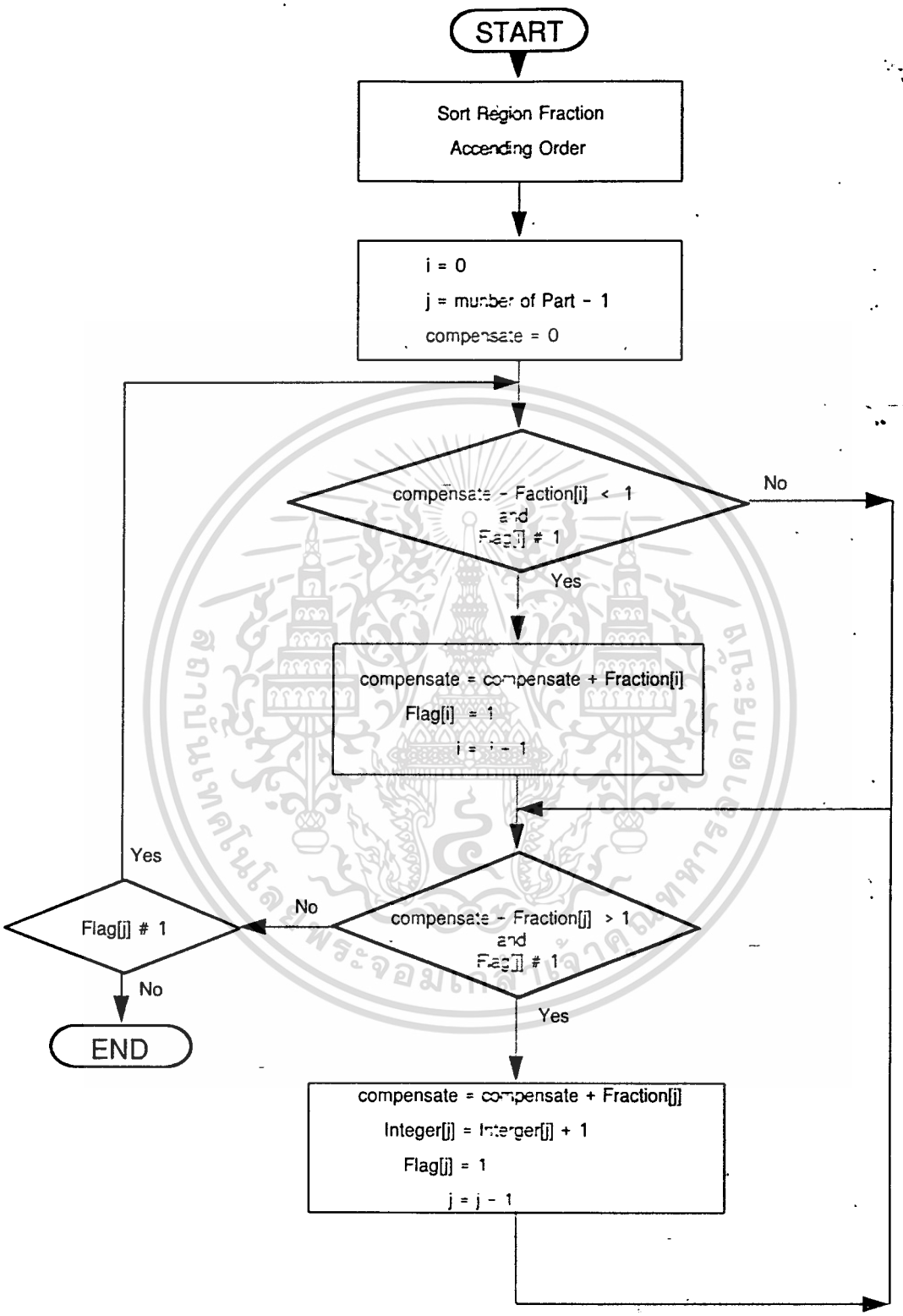
รูปที่ 5.4 แสดงจำนวนรีเจียนที่ทำการปัดจุดทศนิยมแบบทั่วไป โดยใช้เลขจำนวนจริงในรูป 5.3

จะเห็นได้ว่า ค่าผลรวมของรีเจียนหลังจากการปัดจุดทศนิยมในรูป 5.4 (a) มีค่าเท่ากับ 97 รีเจียนและ (b) มีค่าเท่ากับ 95 รีเจียนซึ่งมีค่าไม่เท่ากับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ คือ 96 รีเจียนซึ่งเราจะใช้วิธีการปัดเศษทศนิยมแบบทั่วไปไม่ได้ เนื่องจากค่าผลรวมหลังการปัดจุดทศนิยมที่ได้บางครั้งอาจน้อยกว่า หรือมากกว่าจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ ดังนั้นจึงใช้หลักการปัดเศษทศนิยมแบบทดแทน เพื่อช่วยในการพิจารณาการปัดจุดทศนิยม

5.3 หลักการปัดจุดทศนิยมแบบทดแทน

ปัญหาจากการเปลี่ยนค่าจำนวนจริงให้เป็นเลขจำนวนเต็มมักจะทำให้เกิดผลรวมของเลขจำนวนเต็มนั้น มีค่าไม่เท่ากับผลรวมของเลขจำนวนจริง เพราะเลขจำนวนจริงที่มีจุดทศนิยมจะถูกทำการปัดจุดทศนิยมนั้นขึ้นหรือปัดลง โดยการพิจารณาค่าของจุดทศนิยมนั้นถ้าเลขจำนวนจริงใดทำการปัดจุดทศนิยมขึ้นจะทำให้เลขจำนวนเต็มนั้นมีค่าเพิ่มขึ้นอีก 1 และถ้าเลขจำนวนจริงใดทำการปัดจุดทศนิยมลงจะทำให้เลขจำนวนเต็มนั้นมีค่าเท่าเดิม ดังนั้น ผลรวมของเลขจำนวนเต็มที่มีค่าน้อยกว่าผลรวมของเลขจำนวนจริง จึงเกิดจากเลขจำนวนจริงเหล่านั้นมีการปัดจุดทศนิยมลงมากกว่าการปัดจุดทศนิยมขึ้น และในทางกลับกันผลรวมของเลขจำนวนเต็มหลังการปัดจุดทศนิยมที่มีค่ามากกว่าผลรวมของเลขจำนวนจริงจึงเกิดจากเลขจำนวนจริงเหล่านั้นมีการปัดจุดทศนิยมขึ้นมากกว่าการปัดจุดทศนิยมลง ซึ่งการพิจารณากำหนดการปัดจุดทศนิยมขึ้นหรือลงไม่สามารถใช้ค่าของจุดทศนิยมเป็นเกณฑ์ในการกำหนด ต้องอาศัยค่าทดแทนของจุดทศนิยมเหล่านั้นมาประกอบ

หลังการปัดจุดทศนิยมแบบทดแทน คือนำค่ารีเจียนทุกส่วนที่คำนวณได้มาเรียงลำดับ ซึ่งการเรียงลำดับนี้เราจะพิจารณาเรียงจากค่าทศนิยมที่น้อยที่สุดไปหาค่าทศนิยมที่มากที่สุด โดยการปัดเศษทศนิยมแบบทดแทนนี้จะเริ่มจากการปัดเศษทศนิยมตัวที่น้อยที่สุดก่อนและไปพิจารณาปัดเศษทศนิยมตัวที่มากที่สุดเป็นตัวถัดไป ซึ่งจะพิจารณาจุดทศนิยมที่น้อยสลับกับจุดทศนิยมที่มากไปจนถึงค่าสุดท้ายที่อยู่ส่วนกลาง การปัดจุดทศนิยมขึ้นหรือลงนั้นมีวิธีการพิจารณา คือ จุดทศนิยมใดที่ทำการปัดลงจะนำเศษที่ปัดลงมาเก็บไว้ในตัวแปรทดแทน เช่น ค่ารีเจียนที่คำนวณได้มีค่าเท่ากับ 41.4 เราจะทำการปัดจุดทศนิยมนี้ลงเหลือค่าเท่ากับ 41 โดยนำจุดทศนิยมที่ปัดลงมาเก็บไว้ในตัวแปรทดแทน ซึ่งค่าตัวแปรทดแทนจะมีค่าเท่ากับ 0.4 และจุดทศนิยมใดที่ทำการปัดขึ้นจะต้องนำค่าจากตัวแปรทดแทนมาชดเชยกับส่วนของจุดทศนิยมที่ขาดหายไป(มีค่าครบหนึ่ง) เช่น ค่ารีเจียนที่คำนวณได้มีค่าเท่ากับ 41.5 จะเห็นได้ว่าค่าจุดทศนิยมมีค่าเท่ากับ 0.5 ถ้าจะทำการปัดจุดทศนิยมนี้ขึ้น จะต้องนำตัวแปรทดแทนมาชดเชยให้กับค่านี้เท่ากับ 0.5 จึงจะทำการปัดจุดทศนิยมค่านี้ขึ้นได้ แต่ค่าที่ทำการเก็บไว้ในตัวแปรทดแทน มีค่าเพียง 0.4 ซึ่งค่าตัวแปรทดแทนยังขาดอีก 0.1 เรายังไม่สามารถจะทำการปัดจุดทศนิยมของค่า 41.5 นี้ได้ จะต้องไปพิจารณาปัดเศษทศนิยมตัวที่น้อยสุดตัวถัดลงมาอีกเพื่อนำค่าตัวแปรทดแทนให้มีค่าเพียงพอที่จะมาพิจารณาปัดเศษค่า 41.5 นี้ได้ หลักการปัดจุดทศนิยมแบบทดแทนนี้แสดงดังโฟลว์ชาร์ต รูป 5.5



รูปที่ 5.5 โฟลว์ชาร์ตการปัดจุดทศนิยมแบบทดแทน

เมื่อนำค่าของรีเจียนที่กำหนดให้กับภาพเล็กแต่ละส่วนจากรูป 5.3 ที่ได้จากการเปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ ซึ่งเป็นเลขจำนวนจริงประกอบด้วยเลขจำนวนเต็มที่มีจุดทศนิยม 2 ตำแหน่ง โดยผ่านหลักการปัดจุดทศนิยมแบบทดแทน ทำให้ผลรวมของจำนวนรีเจียนในแต่ละส่วนของภาพเล็กมีจำนวนเท่ากับการกำหนดให้กับภาพใหญ่ ดังเช่น จำนวนรีเจียนรวมจากรูป 5.4 (a) ได้จากการปัดจุดทศนิยมแบบทั่วไปมีค่าเท่ากับ 97 รีเจียน และรูป 5.4 (b) มีค่าเท่ากับ 95 รีเจียน หลังจากผ่านการปัดจุดทศนิยมแบบทดแทนทำให้ค่าของรีเจียนรวมเท่ากับค่าที่กำหนดให้กับภาพใหญ่มีค่าเท่ากับ 96 รีเจียน แสดงดังรูป 5.6

26	21
27	22

a) ภาพจากการแบ่ง 4 ส่วนรวม 96 รีเจียน

7	5	4	8
7	5	4	8
7	5	4	8
7	5	4	8

b) ภาพจากการแบ่ง 16 ส่วนรวม 96 รีเจียน

รูปที่ 5.6 แสดงจำนวนรีเจียนที่ทำการปัดจุดทศนิยมแบบทดแทน โดยใช้เลขจำนวนจริงในรูปที่ 5.3

5.4 ค่าผิดพลาดของการแจกแจงปกติ(The Normal Distribution Error)

ค่าเบี่ยงเบนมาตรฐานของภาพที่ได้จากการคำนวณ โดยใช้ข้อมูลภาพทั้งหมดนั้นทำให้สามารถทราบถึงความแตกต่างของภาพที่มีมากหรือน้อยตามค่าเบี่ยงเบนมาตรฐานที่ได้ ถ้าได้ค่าเบี่ยงเบนมาตรฐานมีค่ามาก นั้นหมายถึง ภาพนั้นมีความแตกต่างของระดับสีเทามาก หรือความแตกต่างของวัตถุในภาพมาก และถ้าได้ค่าเบี่ยงเบนมาตรฐานมีค่าน้อยหรือเข้าใกล้ศูนย์นั้น หมายถึงภาพนั้นมีความแตกต่างของระดับสีเทาที่ใกล้เคียงกันหรือวัตถุในภาพมีโทนสีคล้ายกันนั่นเอง ซึ่งค่าเบี่ยงเบนมาตรฐานที่ได้จากการคำนวณนี้ใช้ข้อมูลภาพทั้งหมด โดยไม่มีการแบ่งแยกข้อมูลภาพบางกลุ่มที่อาจทำให้ผลต่อการเปลี่ยนแปลงของค่ากลางเลขคณิตและค่าเบี่ยงเบนมาตรฐาน สัญญาณรบกวนในภาพบางครั้งอาจเกิดขึ้นจากขบวนการในการถ่ายภาพหรือเกิดจากขั้นตอนในการส่งสัญญาณภาพมายังเครื่องคอมพิวเตอร์และสัญญาณรบกวนนี้มีผลทำให้การคำนวณในการหาค่าเบี่ยงเบนมาตรฐานเกิดการผิดพลาด โดยเฉพาะสัญญาณรบกวนที่อยู่ในช่วงสูงสุดและต่ำสุดของระดับสีเทาของภาพทำให้เกิดจุดขาวและจุดดำในภาพ เมื่อจุดต่ำสุดของภาพ คือ 0 แทนสีดำสุดของภาพ และจุดสูงสุดของภาพ คือ 255 แทนสีขาวสุดของภาพ ค่าเหล่านี้บางครั้งเมื่อถูกรวมเข้าในการหาค่าเฉลี่ยเลขคณิต ทำให้เกิดการผิดพลาด และเมื่อนำค่าเฉลี่ยเลขคณิตนี้เป็นค่ากลางในการหาค่าเบี่ยงเบนมาตรฐานเพื่อเป็นตัวเปรียบเทียบจำนวนรีเจียนให้กับภาพเล็ก เป็นผลทำให้การกำหนดรีเจียนให้กับภาพเล็กไม่ดีเท่าที่ควร

เมื่อข้อมูลภาพที่มีอยู่ในช่วงสูงสุดและต่ำสุดของภาพมีผลทำให้เกิดการผิดพลาด ดังนั้นจะต้องไม่นำข้อมูลภาพที่อยู่ในช่วงสูงสุดและต่ำสุดของภาพมารวมเป็นข้อมูลที่ใช้ในการหาค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานของภาพ ซึ่งการที่จะไม่นำข้อมูลภาพสูงสุดและต่ำสุดมาคำนวณ ทำได้โดยนำข้อมูลภาพมาเปรียบเทียบค่าระดับสีเทาของภาพให้อยู่ในช่วงที่กำหนด ถ้าข้อมูลภาพนั้นไม่อยู่ในช่วงที่กำหนดก็จะไม่นำข้อมูลนั้นมาใช้ในการหาค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐาน เช่นถ้ากำหนดช่วงของข้อมูล คือ 42-225 เป็นช่วงของข้อมูลที่สามารถใช้ในการคำนวณ ถ้าข้อมูลภาพจุดใดมีค่าน้อยกว่า 42 หรือมีค่ามากกว่า 225 จะไม่นำข้อมูลภาพนั้น ๆ มาใช้ในการคำนวณ

การกำหนดช่วงของข้อมูลได้จากการหาค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานในครั้งแรกที่ใช้ข้อมูลภาพทั้งหมดแล้วนำค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานนี้มากำหนดเป็นช่วงของข้อมูลสูงสุดและต่ำสุดตามวิธีค่าผิดพลาดของการแจกแจงปกติ [7] ข้อมูลที่ใช้ในการหาค่าเฉลี่ยเลขคณิตแล้วทำให้ได้ค่าที่มีคุณภาพ (Quality) และข้อมูลที่ใช้ในการหาค่าเบี่ยงเบนมาตรฐาน แล้วทำให้ได้ค่าที่มีความถูกต้อง (Accuracy) ข้อมูลเหล่านั้นควรอยู่ในช่วงดังนี้

$$\text{ช่วงของข้อมูล} = \bar{X} \pm 2\sigma \qquad (5.4)$$

เมื่อ $\bar{X}$ = ค่าเฉลี่ยเลขคณิต

σ = ค่าเบี่ยงเบนมาตรฐาน

ดังตัวอย่าง รูป 5.2 แสดงค่าเบี่ยงเบนมาตรฐานของภาพเล็กแต่ละส่วน ที่ใช้ข้อมูลภาพทั้งหมดของแต่ละส่วนมาคำนวณหาค่าเฉลี่ยเลขคณิต ซึ่งนำภาพใหญ่ขนาด 100 * 100 จุดภาพมาแบ่งออกเป็น 4 ส่วน แสดงในรูป 5.2 (a) และ 16 ส่วน แสดงในรูป 5.2(b) โดยค่าเฉลี่ยเลขคณิตของแต่ละส่วนในรูป 5.2 แสดงในรูป 5.7

84.98	67.86
85.52	67.76

a) ภาพจากการแบ่ง 4 ส่วน

123.56	43.81	51.41	96.56
122.39	42.84	50.98	95.09
121.26	42.38	50.69	93.90
120.16	42.57	49.86	92.77

b) ภาพจากการแบ่ง 16 ส่วน

รูปที่ 5.7 แสดงค่าเฉลี่ยเลขคณิตแต่ละส่วนจากรูปที่ 5.2

เมื่อนำค่าเบี่ยงเบนมาตรฐานและค่าเฉลี่ยเลขคณิตของแต่ละส่วนนี้ ใช้ในการกำหนดช่วงสูงสุดและต่ำสุดของข้อมูลภาพในแต่ละส่วนตามสมการ 5.4 เพื่อใช้หาค่าเบี่ยงเบนมาตรฐาน และค่าเฉลี่ยเลขคณิตครั้งที่สอง ดังตัวอย่างเช่น นำค่าเฉลี่ยเลขคณิตส่วนที่ 1 ของภาพที่แบ่ง 4 ส่วนในรูป 5.7(a) มีค่าเท่ากับ 84.98 และนำค่าเบี่ยงเบนมาตรฐานของภาพส่วนที่ 1 ในรูป 5.2(a) มีค่าเท่ากับ 46 แทนค่าในสมการ 5.4 จะได้ช่วงข้อมูลของภาพในส่วนที่ 1 ของภาพที่แบ่ง 4 ส่วน คือ (-7) - 176 และช่วงของข้อมูลในส่วนอื่น ๆ แสดงดังรูป 5.8

(-7)-176	(-8)-143
(-8)-179	(-8)-143

a) ภาพจากการแบ่ง 4 ส่วน

63-183	(-6)-93	11-91	26-166
60-184	(-5)-90	10-90	23-167
57-185	(-3)-88	10-90	21-165
54-186	(-1)-86	9-89	18-166

b) ภาพจากการแบ่ง 16 ส่วน

รูปที่ 5.8 แสดงช่วงของข้อมูลสูงสุดและต่ำสุดแต่ละส่วนของภาพในรูป 4.3

เมื่อได้ช่วงข้อมูลสูงสุดและต่ำสุดของภาพแต่ละส่วนแล้วนำข้อมูลภาพของแต่ละส่วนมาหาค่าเบี่ยงเบนมาตรฐานครั้งที่สอง โดยใช้ข้อมูลภาพในรูป 4.4 มาแบ่งออกเป็น 4 ส่วน และ 16 ส่วน เช่นเดียวกับการหาค่าเบี่ยนเบนมาตรฐานในครั้งแรก แต่ข้อมูลภาพที่ใช้ในการหาค่าเบี่ยงเบนมาตรฐานในครั้งที่สองนี้ต้องอยู่ในช่วงของข้อมูลที่กำหนดในรูป 5.8 ถ้าข้อมูลใดไม่อยู่ในช่วงที่กำหนดจะไม่นำมาใช้ในการคำนวณ ค่าเบี่ยงเบนมาตรฐานครั้งที่สองแสดงในรูป 5.9

45	36
45	36

a) ภาพจากการแบ่ง 4 ส่วน

22	22	17	35
25	22	17	36
27	21	17	36
29	21	18	37

b) ภาพจากการแบ่ง 16 ส่วน

รูปที่ 5.9 แสดงค่าเบี่ยงเบนมาตรฐานครั้งที่สองใช้ข้อมูลที่อยู่ในช่วง ตามรูปที่ 5.8

ค่าเบี่ยงเบนมาตรฐานครั้งที่สองนี้ เมื่อนำไปเปรียบเทียบตามอัตราส่วนของรีเจียนที่กำหนดให้กับภาพใหญ่จะได้เป็นจำนวนรีเจียนที่ใช้นำไปกำหนดให้กับภาพเล็กๆ แต่ละส่วน ซึ่งจำนวนรีเจียนเหล่านี้ยังเป็นเลขจำนวนจริงอยู่ต้องทำการปัดจุดทศนิยมให้เป็นเลขจำนวนเต็ม โดยใช้หลักการปัดจุดทศนิยมแบบทดแทนที่แสดงไว้ในหัวข้อ 5.3 จะทำให้ผลรวมของจำนวนรีเจียนที่เป็นเลขจำนวนเต็มในแต่ละส่วนมีค่าเท่ากับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ รูปที่ 5.10 แสดงจำนวนรีเจียนที่กำหนดให้กับภาพส่วนเล็กๆ แต่ละส่วนจากการใช้ค่าเบี่ยงเบนมาตรฐานครั้งที่สองที่ผ่านการปัดจุดทศนิยมแบบทดแทน

27	21
27	21

a) ภาพจากการแบ่ง 4 ส่วนรวม 96 รีเจียน

5	5	4	8
6	5	4	9
7	5	4	9
7	5	4	9

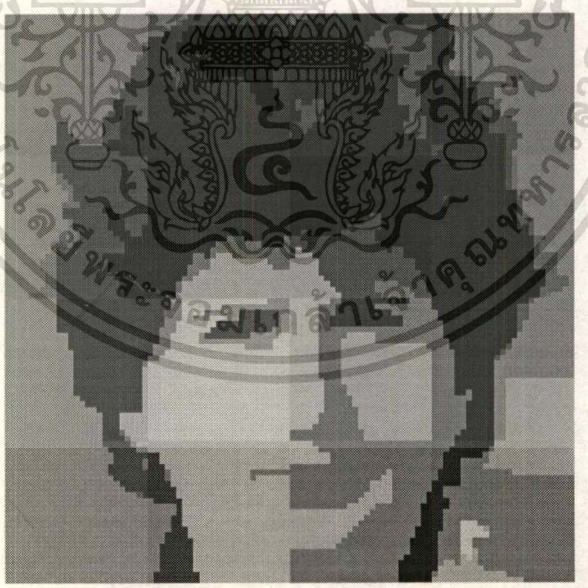
b) ภาพจากการแบ่ง 16 ส่วนรวม 96 รีเจียน

รูปที่ 5.10 แสดงจำนวนรีเจียนที่กำหนดให้กับภาพส่วนเล็กๆ จากการใช้ค่าเบี่ยงเบนมาตรฐานครั้งที่สองที่ผ่านการปัดจุดทศนิยมแบบทดแทน

5.5 ตัวอย่างและผลการทดลอง

จากการทดลองใช้ภาพต้นแบบขนาด 100 x 100 จุดภาพในรูปที่ 4.4 นำมาทำภาพเซ็กเม้นต์โดยแบ่งภาพใหญ่ออกเป็นส่วนๆ 4 ส่วนและ 16 ส่วน แล้วนำภาพแต่ละส่วนมาหาทรีด้วยวิธีของรีเคอร์ซีฟซ็อตเตสท์สแพนนิ่งทรี(RSST) ซึ่งกำหนดจำนวนรีเจียนให้กับภาพใหญ่เท่ากับ 96 รีเจียน เมื่อแปลงทรีกลับเป็นภาพเซ็กเม้นต์ตามการกำหนดรีเจียนให้แต่ละส่วนของภาพเล็กๆ ซึ่งจำนวนรีเจียนที่กำหนดให้แต่ละส่วนแสดงในรูปที่ 5.10 ได้จากการกำหนดช่วงของข้อมูลในการหาค่าเบี่ยงเบนมาตรฐานแล้วนำมาเปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่และผ่านการปัดจุดทศนิยมแบบทดแทน ภาพผลลัพธ์แสดงในรูปที่ 5.11

a) ภาพจากการแบ่ง 4 ส่วน



b) ภาพจากการแบ่ง 16 ส่วน

รูปที่ 5.11 ภาพผลลัพธ์จากการกำหนดรีเจียนที่ได้จากรูป 5.6 โดยใช้ค่าเบียงเบนมาตรฐานที่มีการกำหนดช่วงของข้อมูล

บทที่ 6
สรุป

6.1 สรุปผลการวิจัย

ในการวิเคราะห์และตีความภาพ โดยเฉพาะในบริเวณที่สนใจมักจะมีปัญหาเนื่องจากว่าบริเวณดังกล่าวไม่สามารถแยกให้เด่นจากบริเวณอื่นๆ ที่ใกล้เคียงได้ ดังนั้นได้มีการคิดเทคนิคการแบ่งภาพออกเป็นส่วนๆ (Image segmentation) โดยแต่ละส่วนจะรวมบริเวณของภาพที่มีคุณสมบัติเหมือนหรือใกล้เคียงเข้าด้วยกันจึงทำให้สามารถวิเคราะห์และตีความภาพได้ง่ายขึ้น คุณสมบัติของภาพที่นำมาใช้ในการแบ่งมีหลายอย่าง เช่น ลำดับความเข้มของภาพ(Shade), สี(hue), โครงร่างของภาพ(Texture) หรือความแตกต่างของภาพ(contrast) การแบ่งส่วนภาพด้วยวิธีการแบ่งแยกและรวบรวมโดยตรง (directed split-and-merge)[1] เป็นวิธีที่ใช้เวลาในการคำนวณน้อยแต่มีข้อจำกัดบางอย่าง คือทิศทางการสแกนสำหรับการรวบรวมภาพจะขึ้นอยู่กับทิศทางการสแกนภาพ เป็นผลให้รูปร่างของแต่ละส่วนของภาพที่ได้ขึ้นอยู่กับทิศทางการสแกนของภาพ การแก้ไขข้อเสียนี้ได้นำมาซึ่งการพัฒนาวิธีการเซ็กเม้นต์ภาพใหม่โดยอาศัยทฤษฎีกราฟ เป็นการแบ่งส่วนภาพที่อาศัยซ็อตเตสท์สแพนนิ่งทรีของกราฟ คือทำการแปลงจุดภาพให้อยู่ในรูปของกราฟก่อนแล้วดำเนินการหาซ็อตเตสท์สแพนนิ่งทรี จะเห็นได้ว่าในการหาซ็อตเตสท์สแพนนิ่งทรีนั้นไม่ได้ขึ้นอยู่กับทิศทางการสแกนของภาพ แต่จะขึ้นอยู่กับลิ้งค์เวทของจุดยอดที่น้อยที่สุดไม่ว่าจะอยู่ส่วนใดของกราฟ ดังนั้นเมื่อแปลงจากกราฟกลับมาเป็นภาพ ส่วนของภาพที่ได้แต่ละส่วนก็จะไม่ขึ้นอยู่กับทิศทางการสแกนของภาพ ซึ่งผลที่ได้แต่ละส่วนจะสอดคล้องกับบริเวณของภาพที่มีความเป็นเอกพันธ์(homogeneity) ดีกว่าวิธีการแบ่งส่วนภาพแบบเก่า แต่ก็ยังมีข้อเสียที่ต้องใช้เวลาในการคำนวณมาก เพราะมีการกระทำซ้ำๆ กันเพื่อค้นหาทรีของกราฟ โดยเปรียบเทียบหาจุดยอดในกราฟที่มีตัวเชื่อมที่มีค่าต่ำสุดเพื่อให้ได้ลิ้งค์เวทของจุดยอดที่น้อยที่สุด จำนวนครั้งของการกระทำซ้ำแปรผันโดยตรงกับจำนวนจุดภาพกำลังสอง กล่าวคือ จำนวนครั้งของภาพใหญ่จะมีค่ามากกว่า เมื่อเทียบกับผลรวมของจำนวนครั้งของภาพเล็กๆ แต่ละส่วนรวมกัน ภาพเล็กๆ แต่ละส่วนได้จากการแบ่งภาพใหญ่ออกเป็นส่วนๆ ให้มีสัดส่วนของภาพเล็กเท่ากันทุกส่วน ผลบวกของจำนวนครั้งที่น้อยกว่านี้ทำให้ลดเวลาในการประมวลผล เพื่อสร้างทรีของกราฟ

การนำภาพเล็กๆ แต่ละส่วนมาต่อรวมกันเพื่อให้ได้เป็นภาพใหญ่ จะต้องคำนึงถึงจำนวนรีเจียนรวมของภาพให้มีจำนวนเท่ากับการกำหนดให้ภาพใหญ่ที่ไม่มีการแบ่งส่วน ซึ่งการกำหนดรีเจียนให้กับภาพเล็กๆ เหล่านั้น ต้องมีการพิจารณาการแบ่งจำนวนรีเจียนตามความสำคัญของภาพ หรือรายละเอียดของภาพก่อน โดยอาศัยค่าเบี่ยงเบนมาตรฐานของภาพเล็กๆ แต่ละส่วนเป็นเกณฑ์ในการเปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ ซึ่งการหาค่าเบี่ยงเบนมาตรฐานที่ใช้ข้อมูลของภาพทุกจุดภาพของภาพเล็ก โดยไม่มีการคำนึงถึงข้อมูลภาพบางจุดที่เกิดจากสัญญาณรบกวนที่อยู่ในช่วงสูงสุดหรือต่ำสุด เมื่อนำข้อมูลภาพเหล่านี้รวมเข้ากับข้อมูลภาพอื่นเพื่อหาค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานแล้วทำให้ได้ค่าที่ไม่มีคุณภาพ(Quality) และไม่มีความถูกต้อง(Accuracy) ดังนั้นต้องไม่นำข้อมูลภาพที่อยู่ในช่วงสูงสุดและต่ำสุดเข้ารวมในการหาค่าเฉลี่ยเลขคณิตและค่าเบี่ยงเบนมาตรฐานตามหลักของความผิดพลาดจากการแจกแจงแบบปกติ [7] เมื่อได้ค่าเบี่ยงเบนมาตรฐานของภาพเล็กๆ

แต่ละส่วนแล้วนำมาเปรียบเทียบตามอัตราส่วนกับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ที่ไม่มีการแบ่งส่วน เพื่อให้ได้จำนวนรีเจียนกำหนดให้กับภาพเล็กๆ แต่ละส่วน ซึ่งการเปรียบเทียบตามอัตราส่วนของค่าเบี่ยงเบนมาตรฐานของภาพเล็กกับจำนวนรีเจียนของภาพใหญ่จะทำให้ได้ผลเปรียบเทียบเป็นเลขจำนวนจริงที่ประกอบด้วยเลขจำนวนเต็มและจุดทศนิยม โดยจำนวนรีเจียนที่เป็นเลขจำนวนจริงนี้ไม่สามารถใช้ในการกำหนดให้กับภาพเล็กได้ต้องทำการปรับให้เป็นเลขจำนวนเต็มก่อนด้วยการปัดจุดทศนิยมขึ้นหรือลงตามค่าของจุดทศนิยมนั้น การปัดจุดทศนิยมของจำนวนรีเจียนนี้ต้องคำนึงถึงผลรวมหลังการปัดของจำนวนรีเจียนของภาพเล็กๆ แต่ละส่วนให้เท่ากับจำนวนรีเจียนที่กำหนดให้กับภาพใหญ่ โดยมีการทดแทนจำนวนทศนิยมที่มีการปัด

6.2 ปัญหาที่เกิดขึ้นและข้อเสนอแนะ

การแบ่งภาพออกเป็นส่วนเล็กๆ มาทำเซ็กเม้นเตชั่นทำให้ได้สแพนนิ่งทรีของกราฟแต่ละส่วนแยกจากกันอย่างอิสระ จึงทำให้จำเป็นต้องมีการแยกการกำหนดรีเจียนให้กับภาพแต่ละส่วนเมื่อทำการต่อภาพเล็กๆ แต่ละส่วนเข้าด้วยกันทำให้เกิดรอยต่อระหว่างภาพจากค่าระดับสีเทาที่แตกต่างกัน ถ้าไม่ต้องมีการแยกการกำหนดรีเจียนให้กับภาพเล็กๆ แต่ละส่วน โดยทำให้สแพนนิ่งทรีของกราฟแต่ละส่วนของภาพเล็กๆ ต่อถึงกันเฉพาะทรีที่อยู่ส่วนขอบของภาพเล็ก ซึ่งทรีที่ต่อถึงกันนั้นต้องไม่ทำให้เกิดเป็นไซเคิลในฟอเรสท์ ดังนั้นจะทำให้สแพนนิ่งทรีของภาพเล็กๆ แต่ละส่วนรวมกันกลายเป็นสแพนนิ่งทรีของภาพใหญ่เพียงภาพเดียว และการกำหนดรีเจียนให้กับภาพใหญ่ก็ไม่จำเป็นต้องแยกออกเป็นส่วนๆ ตามภาพเล็ก การกระทำดังกล่าวอาจจะทำให้รูปร่างของเซ็กเม้นต์เปลี่ยนแปลงในส่วนที่อยู่บริเวณรอยต่อของภาพ นั่นหมายถึงความถูกต้องของรีเจียนลดลง วิธีแก้ไขปัญหาอีกอย่างที่เกิดจากค่าระดับสีเทาที่แตกต่างกันระหว่างรอยต่อเมื่อต่อภาพเล็กๆ เข้าด้วยกันเป็นภาพใหญ่ คือ นำรีเจียนที่อยู่ระหว่างรอยต่อทั้งสองภาพมาหาค่าเฉลี่ยของระดับสีเทาใหม่ทีละส่วนของรอยต่อ แล้วแทนค่าระดับสีเทาของรีเจียนด้วยค่าใหม่นี้

เอาสารอ้างอิง

[1] S.L. Horowitz and T. Pavlidis, "Picture segmentation by a directed split-and-merge procedure," Proc. 2nd Int. Joint Conf. on Pattern Recognition, pp.424-433, Aug. 13-15, 1974.

[2] Y. Pramotepipop and F. Cheevasuvit, "Modification of split-and-merge algorithm for image segmentation," Proc. 9th Asian Conf. on Remote Sensing, pp. Q26.1-Q26.6, Nov. 23-29, 1988.

[3] O.J. Morris, M. de J. Lee and A.G. Constantinides, "Graph theory for image analysis : an approach based on the shortest spanning tree," Proc. IEE, Vol. 133, Pt. F, no.2, pp. 146-152, April 1986.

[4] J.B. Jun. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. Am. Math. Soc., pp. 48-50, 1956. 7.

[5] D. Cheriton and R.E. Tarjan, "Finding minimum spanning trees," SIAM J. Comput., pp. 724-742., 1976. 5.

[6] A. Low, "Segmentation and edge detection," Introductory computer vision and image processing, pp.84-99., 1991.

[7] J. Topping, "The normal distribution," Error of observation and their treatment, pp. 55-57., 1975.

[8] R.L. Baker and R.M. Gray, "Image compression using non-adaptive spatial vector quantization," Proc. 16th Asilomar Conf. Comput., pp. 55-61., 1982.

ภาคผนวก ก. ผลงานวิจัยที่ได้รับการตีพิมพ์

[1] ชัยวัช วัฒนมงคลลาภ, รศ.ดร.ฟูศักดิ์ ชีวสุวิทย์, "การปรับปรุงความเร็วในการลดข้อมูลภาพจากเทคนิคการแบ่งส่วนภาพ โดยอาศัยพื้นฐานทฤษฎีกราฟ", การประชุมทางวิชาการ ทางวิศวกรรมไฟฟ้า ครั้งที่ 16, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, หน้า 609-611, 24-26 พฤศจิกายน 2536.

ภาคผนวก ข. โปรแกรมการทดลอง

```
//*********************************************************
//* Program : Recusive shortest spanning tree (RSST.C)
//* Input   : 1) Original image file.
//*           2) Image size.
//*           3) Output spanning tree file name.
//*           4) Number of part to separate.
//* Output  : Spanning tree file.
//*********************************************************
#include    <rsstsmal.h>
#include    <rsstcost.ad>
#include    <stdio.h>
#include    <math.h>
#include    <alloc.h>
#include    <time.h>
#include    <string.h>
#include    <stdlib.h>
#define     sqr(A) ((A)*(A))
/*** Defind grobal variable ***/
struct  LinkElement {
        TYPEREGIONACCESS  r1,r2;
        float  Weight;
       } huge *LinkArray;
struct  RegionRecord {
        TYPEREGIONACCESS  NumPixels;
        TYPELINKACCESS  FirstLinkIndex;
        float           mean;
       } huge *RegionArray;
struct  Region_Rec {
        float  mean;
       } huge *Region_Arr;
struct  TreeEntry {
        TYPEREGIONACCESS  Vertex1;
        char  LinkDir;
       } TreeRecord;
```

```
struct  ListEntry {
        TYPELINKACCESS LinkArrayPtr, NextListEntry;
      } huge *LinkList;
clock_t  start_p,end_p;
unsigned    NumOnHeap, PixelsInImage, RawLinkCount;
unsigned long   PixelsInImage_;
unsigned short  Xsize, Ysize, Xsize_, Ysize_, Xseparate, Yseparate;
unsigned int  Req_part, x_count, y_count;
char     progname[STRINGSIZE],ImageName[STRINGSIZE],*name_close;
char    *TreeName[STRINGSIZE];
TYPELINKACCESS  huge *Heap,huge *Inv,huge *AddMap;
TYPEREGIONACCESS huge *AlreadyLinked, LinkedValue = 0;
TYPEREGIONACCESS LinksOutput;
FILE     *Treefile;

/*********************************************************/
void    AllocateFailure(reqsize)
register unsigned long reqsize;
{
    printf("Exist Memory %lu bytes, Request %lu.\n",farcoreleft(),reqsize);
    fprintf(stderr,"%s: failure in memory allocation",progname);
    exit(1);
}

/*********************************************************/
void    AllocateMemory()
{
    unsigned long lk,la,ra,ra_,in,he,al;
    unsigned long mem_free;
     mem_free = farcoreleft();
     printf("\nAvialable memory before allocate %lu bytes.\n",mem_free);
    lk = (4L*(long)PixelsInImage - 2L*(long)Xsize - 2L*(long)Ysize + 1L) *
        (long)sizeof(struct ListEntry);
    la = (long)RawLinkCount * (long)sizeof(struct LinkElement);
    ra = (long)PixelsInImage * (long)sizeof(struct RegionRecord);
```

```
    in = (long)RawLinkCount * (long)sizeof(TYPELINKACCESS);
    he = (2L * (long)PixelsInImage - (long)Xsize - (long)Ysize + 1L) *
        (long)sizeof(TYPELINKACCESS);
    al = (long)PixelsInImage * (long)sizeof(TYPEREGIONACCESS);
    ra_ = (long)PixelsInImage_ * (long)sizeof(struct Region_Rec);
    if((LinkList = (struct ListEntry*)farmalloc(lk)) == NULL) AllocateFailure(lk);
    if((LinkArray = (struct LinkElement*)farmalloc(la)) == NULL) AllocateFailure(la);
    if((RegionArray = (struct RegionRecord*)farmalloc(ra)) == NULL) AllocateFailure(ra);
    if((Region_Arr = (struct Region_Rec*)farmalloc(ra_)) == NULL) AllocateFailure(ra_);
    if((Inv = (TYPELINKACCESS*)farmalloc(in)) == NULL) AllocateFailure(in);
    if((Heap = (TYPELINKACCESS*)farmalloc(he)) == NULL) AllocateFailure(he);
    if((AddMap = (TYPELINKACCESS*)farmalloc((long)PixelsInImage)) == NULL)
      AllocateFailure(PixelsInImage);
    if((AlreadyLinked = (TYPEREGIONACCESS*)farmalloc(al)) == NULL) AllocateFailure(al);
    printf("Avialable memory after allocate %lu bytes.\n",farcoreleft());
}

/*********************************************************/
void   Initialise(argc,argv)
int    argc;
char   *argv[];
{
char   *stpcpy(),*strcat();
char   huge *TreeName[STRINGSIZE];
int    i, BytesPerPixel;
short  shortbuffer;
FILE   *Infile;
struct Region_Rec  huge *p;

      strcpy(progname,argv[0]);
      switch(argc) {
       case  1: printf("Input image filename: ");
          scanf("%s", ImageName);
       case  2: printf("Image x size : ");
                  scanf("%d",&Xsize_);
```

```
        case  3: printf("Image y size : ");
            scanf("%d",&Ysize_);
        case  4: printf("Output Tree filename: ");
                    scanf("%s", &TreeName);
        case  5: printf("Separate x size : ");
                    scanf("%d", &Xseparate);
            while( (Xsize_ % Xseparate) ) {
              printf("Can't separate X size %d Part...\n",Xseparate);
              printf("Retry Separate x size : ");
              scanf("%d", &Xseparate);
            }
        case  6: printf("Separate y size : ");
                    scanf("%d", &Yseparate);
            while( (Ysize_ % Yseparate) ) {
              printf("Can't separate Y size %d Part...\n",Yseparate);
              printf("Retry Separate y size : ");
              scanf("%d", &Yseparate);
            }
        case  7: break;
          default: fprintf(stderr,"%s: wrong number of arguments\n",progname);
      }
    if((Infile = fopen(ImageName,"rb")) == NULL) {
                    fprintf(stderr,"%s: unable to open file %s\n",progname, ImageName);
                    exit(1);
    }
        Xsize = Xsize_/Xseparate;
        Ysize = Ysize_/Yseparate;
        Req_part  = Xseparate*Yseparate;
        PixelsInImage_ = (long)Xsize_ * (long)Ysize_;
        PixelsInImage = Xsize * Ysize;
        RawLinkCount = 2*PixelsInImage - 1;
        AllocateMemory();
    for(p = Region_Arr; p<Region_Arr+PixelsInImage_; p++) {
        fread((char*)&shortbuffer,1,1,Infile);
        p->mean = (float)shortbuffer;
```

```
    }
        fclose(Infile);
    if((Treefile = fopen(&TreeName,"wb")) == NULL) {
      fprintf(stderr,"%s: unable to open file %s\n",progname, ImageName);
      fclose(Treefile);
      exit(1);
    }
}

/*********************************************************/
void   SetInitialLinks()
{
 register TYPELINKACCESS  LinkIndex,LastLinkIndex;
 register struct  LinkElement *LinkElementPtr;
 register TYPELINKACCESS   LinkPtr;
 register unsigned int     Twoi;
 TYPELINKACCESS  i,CountLinks,LinkEntry;
   LinkEntry = 0;
   for(i=0; i<PixelsInImage; i++) {
          Twoi = i + i;
          CountLinks = 0;
     if(i % Xsize) {
          LinkIndex = ++LinkEntry;
          RegionArray[i].FirstLinkIndex = LinkIndex;
          LinkList[LinkIndex].LinkArrayPtr = Twoi - 1;
          LastLinkIndex = LinkIndex;
          CountLinks++;
     }
     if(i >= Xsize) {
          LinkIndex = ++LinkEntry;
          if(CountLinks) LinkList[LastLinkIndex].NextListEntry = LinkIndex;
          else RegionArray[i].FirstLinkIndex = LinkIndex;
          LinkList[LinkIndex].LinkArrayPtr = Twoi - 2*Xsize + 2;
          LastLinkIndex = LinkIndex;
          CountLinks++;
```

```
        }
        if((i+1) % Xsize) {
                LinkIndex = ++LinkEntry;
                if(CountLinks) LinkList[LastLinkIndex].NextListEntry = LinkIndex;
                else RegionArray[i].FirstLinkIndex = LinkIndex;
                LinkPtr = LinkList[LinkIndex].LinkArrayPtr = Two1 + 1;
                LinkElementPtr = &LinkArray[LinkPtr];
                LinkElementPtr->r1 = i;
                LinkElementPtr->r2 = i+1;
                LinkElementPtr->Weight = FindInitialWeight;
                LastLinkIndex = LinkIndex;
                CountLinks++;
        }
                else if(i != PixelsInImage-1) LinkArray[Two1+1].Weight = -1.0;
        if(i < PixelsInImage - Xsize) {
                LinkIndex = ++LinkEntry;
                LinkList[LastLinkIndex].NextListEntry = LinkIndex;
                LinkPtr = LinkList[LinkIndex].LinkArrayPtr = Two1 + 2;
                LinkElementPtr = &LinkArray[LinkPtr];
                LinkElementPtr->r1 = i;
                LinkElementPtr->r2 = i + Xsize;
                LinkElementPtr->Weight = FindInitialWeight;
                LastLinkIndex = LinkIndex;
                CountLinks++;
               }
               else if(i != PixelsInImage-1) LinkArray[Two1+2].Weight = -1.0;
               LinkList[LastLinkIndex].NextListEntry = 0;
    }
}

/*********************************************************/
void PQDownHeap(k)
register TYPELINKACCESS      k;
{
        register unsigned int      j,HalfNumOnHeap;
```

```
        register unsigned int     HeapOfj,HeapOfjPlus1;
        unsigned int   v;
        register float            vvalue;
        v = Heap[k];
        vvalue = LinkArray[v].Weight;
        HalfNumOnHeap = NumOnHeap/2;
    while (k <= HalfNumOnHeap)  {
            j = k + k;
            HeapOfj = Heap[j];
            HeapOfjPlus1 = Heap[j+1];
            if(j<NumOnHeap)
        if(LinkArray[HeapOfj].Weight > LinkArray[HeapOfjPlus1].Weight)  {
          j++;
          HeapOfj = HeapOfjPlus1;
        }
      if(vvalue <= LinkArray[HeapOfj].Weight ) break;
      Heap[k] = HeapOfj;
      Inv[HeapOfj] = k;
      k = j;
   }
     Heap[k] = v;
     Inv[v]  = k;
}

/*********************************************************/
void    PQConstruct()
{
        register unsigned int     k;
        NumOnHeap = 0;
        Heap[0] = 0;
        LinkArray[0].Weight = -1.0;
        for(k=1; k<RawLinkCount; k++)
     if(LinkArray[k].Weight >= 0.0)  {
            Heap[++NumOnHeap] = k;
            Inv[k] = NumOnHeap;
```

```
        }
    for(k = NumOnHeap/2; k>=1;k--)
        PQDownHeap(k);
}

/*********************************************************/
void    PQUpHeap(k)
register unsigned int k;
{
        register unsigned int     v,Halfk,HalfHeapOfk;
   register float  LinkvWeight;
        v = Heap[k];
        Halfk = k/2;
        HalfHeapOfk = Heap[Halfk];
        LinkvWeight = LinkArray[v].Weight;
   while(LinkArray[HalfHeapOfk].Weight > LinkvWeight) {
            Heap[k] = HalfHeapOfk;
            Inv[HalfHeapOfk] = k;
            k = Halfk;
            Halfk = k/2;
            HalfHeapOfk = Heap[Halfk];
   }
    Heap[k] = v;
    Inv[v]  = k;
}

/*********************************************************/
unsigned int    PQRemove()
{
        register int       LinkNumber;
        LinkNumber = Heap[1];
        Heap[1] = Heap[NumOnHeap];
        if(--NumOnHeap) PQDownHeap(1);
        return(LinkNumber);
}
```

```
/*********************************************************/
void    PQChange(LinkID)
register unsigned int LinkID;
{
        register unsigned int k;
        k = Inv[LinkID];
        if(LinkArray[Heap[k/2]].Weight > LinkArray[Heap[k]].Weight)
                PQUpHeap(k);
        else    PQDownHeap(k);
}

/*********************************************************/
void    PQRid(LinkID)
register unsigned int LinkID;
{
        register unsigned int k;
        k = Inv[LinkID];
        Inv[Heap[k]] = k;
        if(k == NumOnHeap) NumOnHeap--;
        else
          {
            Heap[k] = Heap[NumOnHeap--];
            PQDownHeap(k);
          }
}
/*********************************************************/
void    OutputLink(LinkNumber)
register unsigned int LinkNumber;
{
        TreeRecord.Vertex1 = (LinkNumber-1)/2;
        if(LinkNumber % 2) TreeRecord.LinkDir = 0;
    else
      TreeRecord.LinkDir = 1;
    (void) fwrite(&TreeRecord,sizeof(struct TreeEntry),1,Treefile);
}
```

```
/*********************************************************/
void    ReviseOtherRegionList(RegionNumber,IndexToRemove)
register unsigned int  RegionNumber,IndexToRemove;
{
    register unsigned int    StepIndex,LastStepIndex;
    register struct ListEntry    huge *LinkListPtr;
    char   FirstLink;
        FirstLink = 1;
        StepIndex = RegionArray[RegionNumber].FirstLinkIndex;
        LinkListPtr = &LinkList[StepIndex];
    while(LinkListPtr->LinkArrayPtr != IndexToRemove) {
            FirstLink = 0;
            LastStepIndex = StepIndex;
            StepIndex = LinkListPtr->NextListEntry;
            LinkListPtr = &LinkList[StepIndex];
    }
        if(FirstLink)
         RegionArray[RegionNumber].FirstLinkIndex = LinkListPtr->NextListEntry;
        else
         LinkList[LastStepIndex].NextListEntry = LinkListPtr->NextListEntry;
}

/*********************************************************/
void    MergeRegions(LinkNumber)
unsigned int LinkNumber;
{
  #define mean1       RegionArray[LinkArray[LinkNumber].r1].mean
  #define mean2       RegionArray[LinkArray[LinkNumber].r2].mean
  #define Num1        RegionArray[LinkArray[LinkNumber].r1].NumPixels
  #define Num2        RegionArray[LinkArray[LinkNumber].r2].NumPixels
  #define LinkIndex1  RegionArray[LinkArray[LinkNumber].r1].FirstLinkIndex
  #define LinkIndex2  RegionArray[LinkArray[LinkNumber].r2].FirstLinkIndex
        register unsigned int       LLINextListEntry,LLILinkArrayPtr,LinkIndex,
         LastLinkIndex = 0;
```

```
        unsigned int            TotalPixels,CountLinks,OtherRegion;
        char                    FinishedList,RegionBIsFirst;
        struct ListEntry   *LinkListIndex;
        TotalPixels = Num1 + Num2;
        mean1 = (mean1*Num1 + mean2*Num2)/TotalPixels;
        Num1 = TotalPixels;
        LinkedValue++;
        CountLinks = 0;
        FinishedList = 0;
        LinkIndex = LinkIndex1;
        LinkListIndex = &LinkList[LinkIndex];
        LLINextListEntry = LinkListIndex->NextListEntry;
        LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
    while(!FinishedList) {
      if((LLINextListEntry == 0) && (LLILinkArrayPtr == LinkNumber)) {
        if(CountLinks) LinkList[LastLinkIndex].NextListEntry = LinkIndex2;
        else LinkIndex1 = LinkIndex2;
        LinkIndex = LinkIndex2;
        LinkListIndex = &LinkList[LinkIndex];
        LLINextListEntry = LinkListIndex->NextListEntry;
        LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
        FinishedList = 1;
      }
      else if(LLILinkArrayPtr == LinkNumber) {
            LinkIndex = LLINextListEntry;
            LinkListIndex = &LinkList[LinkIndex];
            LLINextListEntry = LinkListIndex->NextListEntry;
            LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
           if(CountLinks)LinkList[LastLinkIndex].NextListEntry = LinkIndex;
           else LinkIndex1 = LinkIndex;
          }
      else
      {
        LinkArray[LLILinkArrayPtr].Weight = FindWeight(LLILinkArrayPtr);
            PQChange(LLILinkArrayPtr);
```

```
            if(LinkArray[LinkNumber].r1 == LinkArray[LLILinkArrayPtr].r1)
          OtherRegion = LinkArray[LLILinkArrayPtr].r2;
            else
          OtherRegion = LinkArray[LLILinkArrayPtr].r1;
          AlreadyLinked[OtherRegion] = LinkedValue;
          CountLinks++;
        if(LLINextListEntry == 0) {
          LastLinkIndex = LinkIndex;
          LinkListIndex->NextListEntry = LinkIndex2;
          LinkIndex = LinkIndex2;
          LinkListIndex = &LinkList[LinkIndex];
          LLINextListEntry = LinkListIndex->NextListEntry;
          LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
          FinishedList = 1;
        }
        else
        {
          LastLinkIndex = LinkIndex;
          LinkIndex = LLINextListEntry;
          LinkListIndex = &LinkList[LinkIndex];
          LLINextListEntry = LinkListIndex->NextListEntry;
          LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
        }
      }
    }
        FinishedList = 0;
  while(!FinishedList) {
    if((LLINextListEntry == 0)&&(LLILinkArrayPtr == LinkNumber)) {
                LinkList[LastLinkIndex].NextListEntry = 0;
                FinishedList = 1;
               }
     else
      if(LLILinkArrayPtr == LinkNumber) {
        LinkIndex = LLINextListEntry;
        LinkListIndex = &LinkList[LinkIndex];
```

```
  LLINextListEntry = LinkListIndex->NextListEntry;
  LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
  if(CountLinks) LinkList[LastLinkIndex].NextListEntry = LinkIndex;
  else LinkIndex1 = LinkIndex;
          }
else
          {
  if(LinkArray[LinkNumber].r2 == LinkArray[LLILinkArrayPtr].r1) {
              OtherRegion = LinkArray[LLILinkArrayPtr].r2;
              RegionBIsFirst = 1;
  }
  else
            {
              OtherRegion = LinkArray[LLILinkArrayPtr].r1;
              RegionBIsFirst = 0;
            }
 if(AlreadyLinked[OtherRegion] == LinkedValue) {
              PQRid(LLILinkArrayPtr);
              ReviseOtherRegionList(OtherRegion,LLILinkArrayPtr);
   if(LLINextListEntry == 0) {
     LinkList[LastLinkIndex].NextListEntry = 0;
     FinishedList = 1;
   }
              else
   {
     LinkIndex = LLINextListEntry;
     LinkListIndex = &LinkList[LinkIndex];
     LLINextListEntry = LinkListIndex->NextListEntry;
     LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
     LinkList[LastLinkIndex].NextListEntry = LinkIndex;
   }
 }
            else
             {
               if(RegionBIsFirst)
```

```
                    LinkArray[LLILinkArrayPtr].r1 = LinkArray[LinkNumber].r1;
                  else
                    LinkArray[LLILinkArrayPtr].r2 = LinkArray[LinkNumber].r1;
          LinkArray[LLILinkArrayPtr].Weight = FindWeight(LLILinkArrayPtr);
          PQChange(LLILinkArrayPtr);
          CountLinks++;
        if(LLINextListEntry == 0) FinishedList = 1;
        else
        {
          LastLinkIndex = LinkIndex;
          LinkIndex = LLINextListEntry;
          LinkListIndex = &LinkList[LinkIndex];
          LLINextListEntry = LinkListIndex->NextListEntry;
          LLILinkArrayPtr = LinkListIndex->LinkArrayPtr;
        }
              }
          }
    }
}

/*********************************************************/
void    DoKruskalOnLowestWeights()
{
        unsigned int    SmallestLink;
      LinksOutput = 0;
   while(LinksOutput < PixelsInImage - 1) {
          SmallestLink = PQRemove();
          LinksOutput++;
          OutputLink(SmallestLink);
          MergeRegions(SmallestLink);
    }
}

/*********************************************************/
void Separate()
```

```
{
  register TYPELINKACCESS     i,j,Start,add;
  int      x;
  struct RegionRecord   huge *p;
   Start = (Xsize_*Ysize*y_count) + (Xsize*x_count);
   add=0;
   p=RegionArray;
    for ( j=0 ; j<Ysize ; j++ ) {                /* count Ysize */
     for ( i=Start ; i<(Start+Xsize) ; i++ ) {    /* count Xsize */
       AddMap[add++] = i;
       p->mean =  Region_Arr[i].mean;
       p->NumPixels = 1 ;
       ++p ;
     }
     Start = Start+Xsize_ ;
    }
    for(x=0; i<PixelsInImage; x++) AlreadyLinked[x] = 0;
}

/*******************************************************/
/*  Main program                                       */
/*******************************************************/
main(argc,argv)
int  argc;
char *argv[];
{
    unsigned int Part;
    Initialise(argc,argv);
    Part = 0;
    start_p = clock() ;
    printf("Computing..!\n");
  for( y_count = 0 ; y_count < Yseparate ;  y_count++ )  {
   for( x_count = 0 ; x_count < Xseparate ; x_count++ )  {
      printf("--> Part %d\r",Part);
      Separate();
```

```
        SetInitialLinks();
        PQConstruct();
        DoKruskalOnLowestWeights();
        Part++;
      }
    }
      end_p = clock();
      fclose(Treefile);
      printf("\n\nTotal of Processing used %f clock\n",(end_p-start_p)/CLK_TCK);
}
/*** End program RSST.C ***/
```

```
//*********************************************************
//* Program    : SEGMENT.C
//* Decription : Genarate image segment from spanning tree.
//* Input      : 1) Original image file.
//*              2) Image side.
//*              3) Spanning file name.
//*              4) Number of part to separate.
//*              5) Output image segment file name.
//* Output     : Image segment file.
//*********************************************************
#include <stdio.h>
#include <alloc.h>
#include <math.h>
#define STRSIZE        50
/*** Defind grobal variable ***/
char     *LinkArray,*MarkerArray,progname[STRSIZE];
char     ImageFile[STRSIZE], TreeFile[STRSIZE] ,SegFile[STRSIZE];
int      *PixelStack, IncImageIndex[9], RequiredRegions;
int      MaxInt = -32000, MinInt = 32000;
double   x_bar_total;
unsigned char   *ImageArray,*ImageArray_;
unsigned short  Xsize,Ysize,Xsize_,Ysize_;
unsigned int    Req_part, x_count, y_count,Xseparate,Yseparate, Part;
unsigned long   TotalPixels_;
unsigned        TotalPixels;
FILE     *InTreeFile,*OutFile;
struct   Regions_varance
         { int    Array;
           double Varance;
           double Integer;
           double Fraction;
           unsigned double x_bar_array;
           unsigned t_pixels;
           int    min_varance;
```

```
        int    max_varance;
        int    flag;
      } huge *Region;

/*********************************************************/
void AllocFailure()
{
      fprintf(stderr,"%s: Failure to allocate memory\n",progname);
      exit(1);
}

/*********************************************************/
void GetSpace()
{
 long    rp;
  unsigned long mem_free;
  rp = (long)Req_part * (long)sizeof(struct Regions_varance);
   mem_free = farcoreleft();
  printf("\nMemory avialable %lu bytes.\n",mem_free);
   if((ImageArray = (long*)farmalloc((long)TotalPixels*
              (long)sizeof(short)))== NULL) AllocFailure();
  printf("ImageArray used %lu bytes.\n",mem_free-farcoreleft());
  mem_free = farcoreleft();
   if((ImageArray_ = farmalloc(TotalPixels_*
              (unsigned long)sizeof(char)))== NULL) AllocFailure();
  printf("ImageArray_ used %lu bytes.\n",mem_free-farcoreleft());
  mem_free = farcoreleft();
  if((LinkArray = (long)farmalloc((long)TotalPixels*
              (long)sizeof(char)))== NULL) AllocFailure();
  printf("LinkArray used %lu bytes.\n",mem_free-farcoreleft());
  mem_free = farcoreleft();
   if((MarkerArray = (long)farmalloc((long)TotalPixels*
              (long)sizeof(char)))== NULL) AllocFailure();
  printf("MarkerArray used %lu bytes.\n",mem_free-farcoreleft());
  mem_free = farcoreleft();
```

```
    if((PixelStack = (long)farmalloc((long)TotalPixels*
                    (long)sizeof(char))) == NULL) AllocFailure();
    printf("PixelStack used %lu bytes.\n",mem_free-farcoreleft());
    mem_free = farcoreleft();
    if((Region = (struct Reginos_varance*)farmalloc(rp)) == NULL)
    AllocFailure();
    printf("Region used %lu bytes.\n",mem_free-farcoreleft());
}

/*********************************************************/
void GetInputs(argc,argv)
register int argc;
register char *argv[];
{
        register int i;
        char    buff;
        FILE    *InFile;
        strcpy(progname,argv[0]);
   switch(argc) {
     case 1: printf("Original image filename: ");
       scanf("%s",ImageFile);
       if((InFile = fopen(ImageFile,"rb")) == NULL) {
            fprintf(stderr,"\n%s: Error cannot open file %s\n",progname,ImageFile);
            exit(1);
       }
     case 2: printf("Image X size: ");
       scanf("%d",&Xsize_);
     case 3: printf("Image Y size: ");
       scanf("%d",&Ysize_);
     case 4: printf("Tree filename: ");
       scanf("%s",TreeFile);
       if((InTreeFile = fopen(TreeFile,"rb")) == NULL) {
          fprintf(stderr,"\n%s: Error cannot open file %s\n",progname,TreeFile);
          exit(1);
       }
```

```
    case 5: printf("Separate X size : ");
        scanf("%d", &Xseparate);
        while( (Xsize_ % Xseparate) ) {
            printf("Can't separate X size %d Part...\n",Xseparate);
            printf("Retry Separate X size : ");
            scanf("%d", &Xseparate);
        }
    case 6: printf("Separate Y size : ");
        scanf("%d", &Yseparate);
        while( (Ysize_ % Yseparate) ) {
            printf("Can't separate Y size %d Part...\n",Yseparate);
            printf("Retry Separate Y size : ");
            scanf("%d", &Yseparate);
        }
    case 7: printf("Output filename: ");
        scanf("%s",SegFile);
        if((OutFile = fopen(SegFile,"wb")) == NULL) {
          fprintf(stderr,"\n%s: Error cannot open file %s\n",progname,SegFile);
          exit(1);
        }
    case 8: printf("Required no. of regions: ");
        scanf("%d",&RequiredRegions);
        if( RequiredRegions < (Xseparate*Yseparate) ) {
          fprintf(stderr,"\n%s: Error number of regions less than number of part.\n",progname);
          exit(1);
        }
    case 9: break;
      default: fprintf(stderr,"%s: wrong number of arguments\n",progname);
}
        Xsize = Xsize_/Xseparate;
        Ysize = Ysize_/Yseparate;
        Req_part = Xseparate*Yseparate;
        TotalPixels_ = (long)Xsize_ * (long)Ysize_;
        TotalPixels  = Xsize * Ysize;
        GetSpace();
```

```
        /**  Read Image File **/
        x_bar_total = 0;
    for(i=0; i<TotalPixels_; i++)  {
      fread((char*)&buff,1,1,InFile);
      ImageArray_[i] = buff;
      x_bar_total += ImageArray_[i];
          }
   x_bar_total = x_bar_total/TotalPixels_;
        fclose(InFile);
        IncImageIndex[1] = -Xsize;
        IncImageIndex[2] = 1;
        IncImageIndex[4] = Xsize;
        IncImageIndex[8] = -1;
}

/*********************************************************/
void ReadLinks()
{
        register long     i,NumLinks;
   struct  SmallTreeEntry {
      unsigned int Vertext;
      char     LinkDir;
   } SmallBuffer;
    if( Region[Part].Integer < 1 ) {
                fprintf(stderr,"\n%s: Error region set less than Zero\n",progname);
                exit(1);
     }
        NumLinks = (long)TotalPixels - (long)Region[Part].Integer ;
   for(i=0; i<NumLinks; i++)  {
      if(fread((char*)&SmallBuffer,sizeof(struct SmallTreeEntry),1,InTreeFile) != 1)
            {
             fprintf(stderr,"\n%s: Error in reading file %s\n",progname,TreeFile);
             exit(1);
            }
      if(SmallBuffer.LinkDir)
```

```
            {
             LinkArray[SmallBuffer.Vertext] |= 4;
             LinkArray[SmallBuffer.Vertext + Xsize] |= 1;
            }
           else
            {
             LinkArray[SmallBuffer.Vertext] |= 2;
             LinkArray[SmallBuffer.Vertext + 1] |= 8;
            }
    }
    for( i=0 ; i<((long)Region[Part].Integer-1) ; i++) {
      if(fread((char*)&SmallBuffer,sizeof(struct SmallTreeEntry),1,InTreeFile) != 1)
       {
        fprintf(stderr,"\n%s: Error in reading Part %d file %s\n",progname,Part,TreeFile);
        exit(1);
       }
     }
}

/*********************************************************/
int FindMean(Index)
register long     Index;
{
        register long int  TotalIntensity,NumInRegion;
        register long StackPointer = 0,CheckPixel;
        register char     LookAround;

        MarkerArray[Index] = 1;
        TotalIntensity = (int)ImageArray[Index];
        NumInRegion = 1;
        do
         {
          for(LookAround=1; LookAround<16; LookAround<<=1)
            {
             if(LinkArray[Index] & LookAround)
```

```
                {
                 CheckPixel = Index + IncImageIndex[LookAround];
                 if(!MarkerArray[CheckPixel])
                  {
                   MarkerArray[CheckPixel] = 1;
                   TotalIntensity += (int)ImageArray[CheckPixel];
                   NumInRegion++;
                   PixelStack[++StackPointer] = CheckPixel;
                  }
                }
              }
              Index = PixelStack[StackPointer--];
            } while (StackPointer >= 0);
            return((short)((TotalIntensity+NumInRegion/2)/NumInRegion));
}

/*************************************************************/
void SetMean(Index,MeanValue)
register long Index;
register int MeanValue;
{
        register long     CheckPixel,StackPointer = 0;
        register char     LookAround;
        MarkerArray[Index] = 2;
        ImageArray[Index] = MeanValue;
        do
         {
          for(LookAround=1; LookAround<16; LookAround<<=1)
           if(LinkArray[Index] & LookAround)
            {
             CheckPixel = Index + IncImageIndex[LookAround];
             if(MarkerArray[CheckPixel]==1)
                 {
                  MarkerArray[CheckPixel] = 2;
                  ImageArray[CheckPixel] = MeanValue;
```

```
                PixelStack[++StackPointer] = CheckPixel;
              }
            }
          Index = PixelStack[StackPointer--];
          } while (StackPointer >= 0);
}

/*********************************************************/
void  GetSegmentation()
{
        register long Pixel;
        int       MeanIntensity;
        for(Pixel=0; Pixel<(long)TotalPixels; Pixel++)
          if(!MarkerArray[Pixel])
            {
             MeanIntensity = FindMean(Pixel);
             SetMean(Pixel,MeanIntensity);
             if(MaxInt<MeanIntensity) MaxInt = MeanIntensity;
             if(MinInt>MeanIntensity) MinInt = MeanIntensity;
            }
}

/*********************************************************/
void  StoreImage()
{
 register unsigned long add, Start, i;
 register unsigned short j;
   Start = (Xsize_*Ysize*y_count) + (Xsize*x_count);
        add = 0;
        for( j=0 ; j<Ysize ; j++ ) {
          for( i=Start ; i<(Start+(long)Xsize) ; i++ ) {
            ImageArray_[i] = ImageArray[add++];
          }
          Start = Start+(long)Xsize_;
        }
```

```
    if( Part == Req_part-1 ) {
            if(fwrite((char*)ImageArray_,2,TotalPixels_/2,OutFile)==NULL)
              printf("\n Error write out file part 1.\n");
    }
}

/*********************************************************/
void Seperate()
{
   register long add,i,j,Start,count;
   Start = (Xsize_*Ysize*y_count) + (Xsize*x_count);
   add = 0, count = 0;
          for( j=0 ; j<(long)Ysize ; j++ ) {
            for( i=Start ; i<(Start+(long)Xsize) ; i++ ) {
               ImageArray[add++] = ImageArray_[i];
              LinkArray[count] = 0;
              MarkerArray[count++] = 0;
            }
            Start = Start+Xsize_;
          }
}

/*********************************************************/
/* The Bubble sort. */
void bubble(item,count)
register struct Regions_varance huge *item;
register unsigned int count;
{
 register int a,b;
 register struct Regions_varance t;
 for(a=1; a<count; ++a)
    for(b=count-1; b>=a; --b) {
     if(item[b-1].Fraction > item[b].Fraction) {
      /* exchange element */
      t = item[b-1];
```

```
      item[b-1] = item[b];
      item[b] = t;
     }
    }
}

/*************************************************************/
Findvariance()
{
 register int j,loop,start;
 register unsigned long i;
 register double total_varance;
 double   buff,add;
 register  unsigned long x_cnt,y_cnt;
  for( j = 0 ; j < Req_part ; j++ ) {        /* Initial Valiabel */
    Region[j].Array = 0;
    Region[j].min_varance = 0;
    Region[j].max_varance = 255;
    Region[j].flag = 0;
  }
 for( loop = 0 ; loop < 2 ; loop++ ) {       /* NORNAL ERROR SELECT 2 */
    for( j = 0 ; j < Req_part ; j++ ) {
      Region[j].x_bar_array = 0;
      Region[j].t_pixels = 0;
      Region[j].Varance = 0;
      Region[j].Integer = 0;
      Region[j].Fraction = 0;
    }
          /* Xbar  */
    start = 0;
    for( y_cnt = 0 ; y_cnt < Ysize ; y_cnt++ ) {
     for( x_cnt = start ; x_cnt < (start+Xsize) ; x_cnt++ ) {
      for( j=0 ; j<Req_part ; j++) {
       if( ImageArray_[x_cnt+(j*Xsize)] >= Region[j].min_varance &&
              ImageArray_[x_cnt+(j*Xsize)] <= Region[j].max_varance) {
```

```
        Region[j].x_bar_array += ImageArray_[x_cnt+(j*Xsize)];
        Region[j].t_pixels++;
      }
     }
    }
     start = start + Xsize_;
   }
   for( j = 0 ; j < Req_part ; j++ ){
    Region[j].x_bar_array = Region[j].x_bar_array/Region[j].t_pixels;
   }
        /* sum(Xi-Xbar)2 */
   start = 0;
   for( y_cnt = 0 ; y_cnt < Ysize ; y_cnt++ ) {
    for( x_cnt = start ; x_cnt < (start+Xsize) ; x_cnt++ ) {
     for ( j=0 ; j<Req_part ; j++) {
      if( ImageArray_[x_cnt+(j*Xsize)] >= Region[j].min_varance &&
            ImageArray_[x_cnt+(j*Xsize)] <= Region[j].max_varance) {
       Region[j].Varance+=(double) pow(abs(ImageArray_[x_cnt+(j*Xsize)]
                - Region[j].x_bar_array),2);
      }
     }
    }
     start = start + Xsize_;
   }
        /* root( sum/n ) */
    total_varance = 0;
    for( j=0 ; j<Req_part ; j++) {
     Region[j].Varance = (int long)sqrt(Region[j].Varance/Region[j].t_pixels);
     total_varance += Region[j].Varance;
     Region[j].min_varance = Region[j].x_bar_array - (2*Region[j].Varance);
     Region[j].max_varance = Region[j].x_bar_array + (2*Region[j].Varance);
    }
}
     /*  devide equally varance */
  for( j=0 ; j<Req_part ; j++) {
```

```
    Region[j].Array=j;
    Region[j].Varance=Region[j].Varance*RequiredRegions/total_varance;
    Region[j].Fraction=modf(Region[j].Varance,&Region[j].Integer);
  }
  bubble(Region,Req_part);
  i=0;j=Req_part-1;buff=0;
  do {
      while( Region[i].Integer == 0) {
        buff -= Region[i].Fraction;
        Region[i].Integer++;
        Region[i].Fraction=Region[i].Array;
        Region[i].flag=1;
        i++;
      }
      if( (buff+Region[i].Fraction)<1 && Region[i].flag !=1 ) {
         buff += Region[i].Fraction;
         Region[i].Fraction=Region[i].Array;
         Region[i].flag=1;
         i++;
      }
      while((buff+Region[j].Fraction)>=1 && Region[j].flag !=1 ) {
         buff = modf(Region[j].Fraction+buff,&add);
         Region[j].Integer++;
         Region[j].Fraction=Region[j].Array;
         Region[j].flag=1;
         j--;
      }
  } while( Region[j].flag !=1 && i!=j );
  if( buff > 0.5 ) Region[--i].Integer++ ;
  bubble(Region,Req_part);
}
```

```
/*****************************************************************/
/*  Main program                                                 */
/*****************************************************************/
main(argc,argv)
int     argc;
char    argv[];
{
        GetInputs(argc,argv);
        Findvariance();
        Part = 0;
   for( y_count = 0 ; y_count < Yseparate ; y_count++ ) {
    for( x_count = 0 ; x_count < Xseparate ; x_count++ ) {
      Seperate();
      ReadLinks();
      GetSegmentation();
      StoreImage();
      Part++;
    }
   }
        fclose(InTreeFile);
        fclose(OutFile);
    return(0);
}
/***  End program SEGMENT.C  ***/
```

ประวัติผู้เขียน

ชื่อผู้เขียน	นายชัยวัช วัฒนมงคลลาภ
วันเดือนปีเกิด	วันที่ 7 มกราคม พ.ศ 2511
สถานที่เกิด	จังหวัดฉะเชิงเทรา
วุฒิการศึกษาระดับปริญญาตรี	อุตสาหกรรมศาสตร์บัณฑิต สาขาวิชาคอมพิวเตอร์อุตสาหกรรม
สถานที่สำเร็จการศึกษา	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีที่สำเร็จการศึกษา	ปีการศึกษา 2533
สถาณที่ทำงานปัจจุบัน	รองผู้จัดการแผนกคอมพิวเตอร์ บริษัท โตชิบาเซมิคอนดัคเตอร์(ประเทศไทย) จำกัด