การประมวลผลสัญญาณภาพเคลื่อนไหว

(MOTION IMAGE PROCESSING)

โดย

นายญาณพล อุ่นโอสถ

นายวิโรจน์ ปัญญาชาติรักษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเลคทรอนิคส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

การประมวลผลสัญญาณภาพเคลื่อนไหว

MOTION IMAGE PROCESSING

โดย

นาย ญาณพล อุ่นโอสถ

นาย วิโรจน์ ปัญญาชาติรักษ์

อาจารย์ที่ปรึกษา

รศ.ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเลคทรอนิคส์

สถาบันบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 3539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชา วิศวกรรมอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประมวลผลสัญญาณภาพเคลื่อนไหว

ผู้จัดทำ

1. นาย ญาณพล อุ่นโอสถ 36014128

2. นาย วิโรจน์ ปัญญาชาติรักษ์ 36014412

(รศ.ดร.มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การประมวลผลสัญญาณภาพเคลื่อนไหว

MOTION IMAGE PROCESSING

1. นาย ญาณพล อุ่นโอสถ 36014128

2. นาย วิโรจน์ ปัญญาชาติรักษ์ 36014412

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการทดสอบได้

(รศ.ดร.มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การประมวลผลสัญญาณภาพเคลื่อนไหว

นาย ญาณพล อุ่นโอสถ 36014128
นาย วิโรจน์ ปัญญาชาติรักษ์ 36014412
อาจารย์ที่ปรึกษา รศ.ดร.มนัส สังวรศิลป์
ปีการศึกษา 2539

บทคัดย่อ

ปริญญานิพนธ์นี้ ได้เสนอหลักการพื้นฐานของการประมวลผลสัญญาณภาพ เช่น การแปลงโคซายน์เต็มหน่วย , การเข้ารหัสแบบฮัฟแมน เป็นต้น หลังจากนั้นก็ได้นำเสนอ หลักการเบื้องต้นของการบีบอัดข้อมูลภาพเคลื่อนไหว และขั้นตอนของการบีบอัดภาพเคลื่อนไหว เช่น การหาโมชั่นเวคเตอร์ ซึ่งทำได้โดยการแบ่งภาพเป็นบล็อกเล็ก ๆ แล้วหาว่าบล็อกใดมีความแตกต่างกันน้อยที่สุด , การเข้ารหัสแบบรันเลนจ์ , การเข้ารหัสแบบฮัฟแมน การบีบอัดภาพเคลื่อนไหวในขณะนี้มีประโยชน์ในการสื่อสารข้อมูลภาพ ซึ่งคาดว่าจะเป็นที่นิยมมากในอนาคต

MOTION IMAGE PROCESSING

Mr.Yanapol Unosot 36014128

Mr.Wirote Panyachatiraksa 36014412

Advisor Asst.Prof. Dr. Manus Sungvornsilp

Academic Year 1996

Abstract

This Thesis aims to present basic concept of Image Processing such as Discrete Cosine Transfore (DCT) , Huffmann coding. In the section of moving image , we present the theory of motion image and algorithm of motion image compression such as motion vector that can determine by dividing image to the blocks and find which block has the least difference , Runlength encoding , Huffmann encoding . The use of motion image compression is the multimedia communication that expected to be popular in the future.

สารบัญ

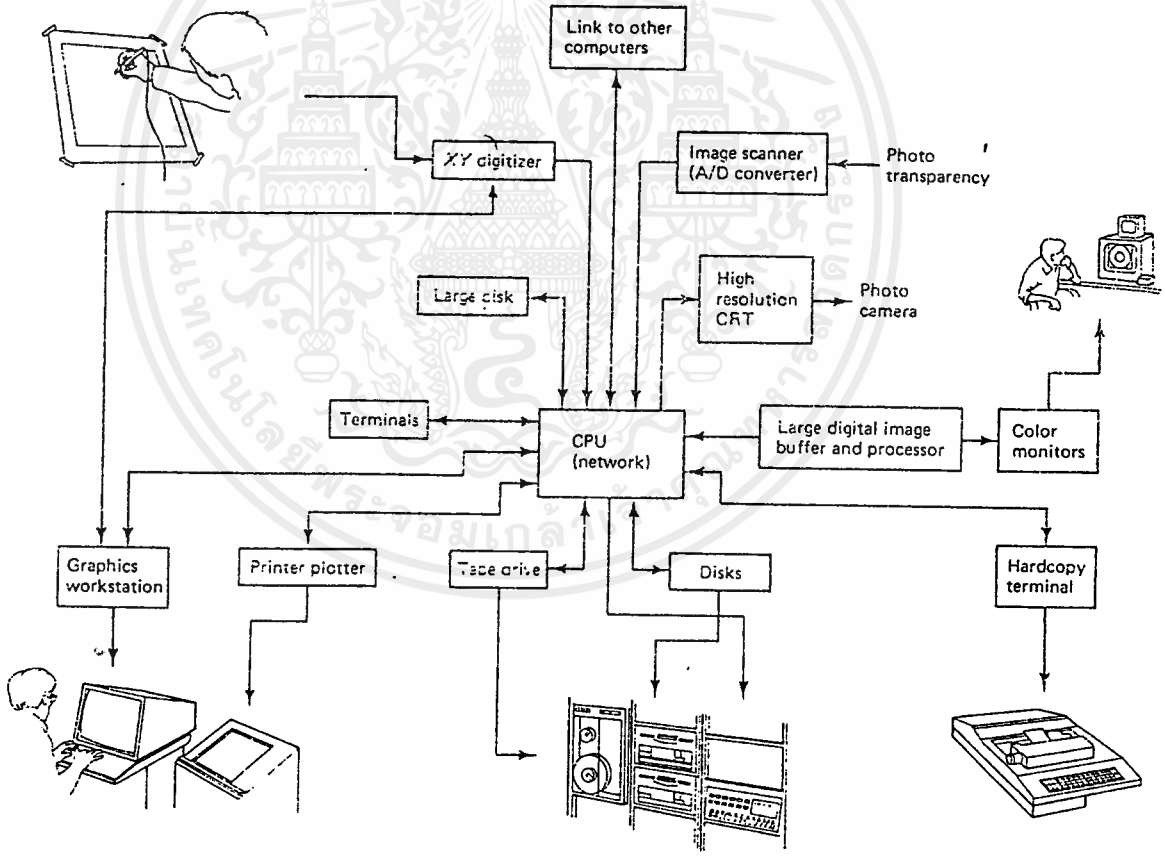
สารบัญภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ดิจิตอลอิมเมจโปรเซสซิ่ง (Digital Image Processing)

ดิจิตอลอิมเมจโปรเซสซิ่ง โดยทั่วไปหมายถึง การประมวลผลของภาพใน 2 มิติ โดยคอมพิวเตอร์ หรือถ้าจะกล่าวในความหมายที่กว้างขึ้น จะหมายถึง การประมวลผลของข้อมูล 2 มิติใด ๆ ดังในรูปที่ 1.1 เป็นห้องทดลองคอมพิวเตอร์ที่ University of California



รูปที่ 1.1 แสดงถึงระบบดิจิตอลอิเมจโปรเซสซิ่งของ University of California

จากรูป ภาพจากสไลด์ (Slide) , รูปถ่าย จะถูกแปลงให้เป็นสัญญาณดิจิตอล และเก็บไว้ในหน่วยความจำ โดยภาพนี้สามารถจะนำไปประมวลผลได้ต่อไป ส่วนมินิ หรือ ไมโครคอมพิวเตอร์ใช้ในการควบคุมการสื่อสารข้อมูลของระบบเครือข่ายคอมพิวเตอร์ ผู้ใช้สามารถพิมพ์โปรแกรมผ่านทางเทอร์มินอลและในทำนองเดียวกันสามารถดูข้อมูลได้จากเทอร์มินอล เช่นกัน

ดิจิตอลอิเมจโปรเซสซิ่ง ได้มีการใช้งานอย่างกว้างขวาง เช่นการส่งข้อมูลผ่านดาวเทียมและยานอวกาศ ,การส่งข้อมูลและเก็บข้อมูลภาพในธุรกิจ , เรดาร์ , โซนาร์ , การตรวจสอบชิ้นส่วนอัตโนมัติในโรงงาน

รูปที่ได้จากดาวเทียมมีประโยชน์ในการหาทรัพยากรธรรชาติ , การทำแผนที่ , การทำนายผลิตผลทางการเกษตร , การเจริญเติบโตของเมือง , การพยากรณ์อากาศ และอื่น ๆ อีกมาก

การส่งและบันทึกข้อมูลภาพได้ใช้ในการออกอากาศทางโทรทัศน์ , การประชุมทางไกล (teleconference) , การรับ-ส่ง แฟกซ์ , การสื่อสารของระบบเครือข่ายคอมพิวเตอร์ , โทรทัศน์วงจรปิด , การสื่อสารทางทหาร ในทางการแพทย์ก็เช่น การประมวลผลภาพจาก x-ray และ ultrasonic scaning โดยภาพดังกล่าวสามารถนำมาวินิจฉัยโรคได้

เราสามารถสรุปได้ว่า เมื่อใดก็ตามที่มนุษย์หรืออุปกรณ์ใด ๆ ก็ตามได้รับข้อมูลและประมวลผลข้อมูลที่เป็น 2 มิติ เราก็จะเรียกได้ว่าเป็น อิเมจโปรเซสซิ่ง

การประมวลผลสัญญาณภาพได้มีการประยุกต์ใช้งานอย่างกว้างขวาง ตัวอย่างเช่น อิมเมจเอนฮานเมนต์(Image Enhancement) ,อิมเมจรีสตอเรชั่น (Image restoration) ดังนั้นการใช้งานก็ต่างกันไป อย่างเช่น อิเมจเอ็นฮานซ์เมนต์ ใช้ปรับปรุงคุณภาพของภาพให้ดีขึ้น

1.2 วัตถุประสงค์ของปริญญานิพนธ์

เพื่อจะนำเสนอวิธีการลดข้อมูลภาพสำหรับลดพื้นที่ในการเก็บข้อมูล อีกทั้งยังประหยัดค่าใช้จ่ายและเวลาในการส่งข้อมูลเป็นระยะทางไกลๆ เช่น การส่งข้อมูลผ่านดาวเทียม การส่งข้อมูลในอินเทอร์เน็ต

1.3 ขอบเขตของปริญญานิพนธ์

เนื้อหาของปริญญานิพนธ์นี้จะเกี่ยวกับหลักการเบื้องต้นของอิเมจโปรเซสซิ่ง และการประยุกต์การใช้งาน รวมถึงการลดข้อมูลภาพเคลื่อนไหวเบื้องต้น

บทที่ 2
หลักการประมวลผลภาพ

2.1 หลักการพื้นฐานของอิมเมจโปรเซสซิ่ง

อิมเมจโปรเซสซิ่ง (image processing) เป็นกระบวนการที่ใช้ในการจัดการข้อมูลที่เป็นรูปภาพต่าง ๆ ให้อยู่ในลักษณะของสัญญาณไฟฟ้าเพื่อที่จะได้นำข้อมูลที่เป็นสัญญาณไฟฟ้านั้นไปใช้ประโยชน์ในทางอื่น เป็นต้นว่า การตกแต่ง , การส่งรูปภาพไปตามสายนำสัญญาณจากที่แห่งหนึ่งไปยังอีกแห่งหนึ่ง (ซึ่งก็คือหลักการของโทรสาร) , การเก็บข้อมูลรูปภาพไว้ในหน่วยความจำเพื่อทำอัลบัมภาพอิเล็กทรอนิกส์เพื่อใช้ประโยชน์เป็นแฟ้มข้อมูลพนักงาน , แฟ้มอาชญากรรม เป็นต้น นอกเหนือไปจากนี้ยังสามารถนำไปใช้ในงานด้านการรักษาความปลอดภัย , ตรวจสอบลายนิ้วมือ ได้อีกด้วย

พิกเซล (pixel)

ภาพที่จะถูกแปลงเป็นสัญญาณไฟฟ้า จะได้รับการแบ่งราบละเอียดของภาพเป็นตารางเล็กๆ และตารางเล็ก ๆนี้เองคือพิกเซล (pixel) และเมื่อมีการจัดเรียงพิกเซลก็จะเกิดพิกเซลที่เป็นแถว (row) และคอลัมน์ (column) โดยมีจำนวนแถวทางแนวนอนเป็น N แถว และมีจำนวนแถวทางแนวตั้ง M แถว ซึ่งในแต่ละตำแหน่งของพิกเซลจะแทนด้วย P(j,j) โดยที่ I และ j จะเป็นเลขจำนวนเต็มและที่เรียกการจัดเรียวของพิกเซลว่าพิกเซลเมตริกซ์ (pixel matrix)

เมื่อทราบตำแหน่งของพิกเซลแล้วก็จำเป็นต้องทราบว่าที่ตำแหน่งนั้น ๆ พิกเซลมีค่าเท่าไหร่ ซึ่งค่าที่ว่านี้ก็คือค่าเฉลี่ยความเข้มของแสงที่ตกกระทบบนตำแหน่งของแต่ละพิกเซลซึ่งมีค่าอยู่ระหว่าง 0 กับ 1 เมื่อทำเป็นภาพที่มีความเข้มสองระดับ

ลักษณะข้อมูลภาพ

1. ภาพ 2 ระดับ คือ มีแค่จุดขาวกับดำเท่านั้น โดยแต่ละจุดเป็นข้อมูล 1 บิต

2. ภาพ 16 ระดับ ซึ่งในแต่ละจุดภาพจะเป็นข้อมูล 4 บิต ซึ่งทำให้สามารถแสดงภาพได้ 16 ระดับสี หรือ 16 ระดับสีเทา ขึ้นอยู่กับว่าภาพนั้นเป็นภาพสีหรือภาพขาว-ดำ

3. ภาพ 256 ระดับซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูล 8 บิต ซึ่งทำให้สามารถแสดงภาพได้ 256 ระดับสี หรือระดับสีเทา ขึ้นอยู่กับส่าภาพนั้นเป็นภาพสี หรือ ขาว-ดำ

4. ภาพ true Color ซึง ในแต่ละจุดจะเป็นข้อมูลขนาด 24 บิต ทำให้สามารถแสดงผลภาพได้เหมือนภาพจริงที่สุด เพราะสามารถปสดงสีได้ถึง 16,777,216 สี ภาพ true color สามารถแสดงได้เฉพาะภาพสีเท่านั้น ไม่สามารถแสดงภาพขาว-ดำได้

การแสดงภาพนี้ใช้วิธี ตั้งค่าของแม่สีในตารางสี โดยอาจเลือกสีเป็นแบบ 16 สี จาก 64 สี หรือ 16 สี จาก 262,144 สี หรือ 256 สี จาก 262,144 สี ขึ้นอยู่กับโหมด การแสดงผล สำหรับ true color ไม่มีการเลือกสี แสดงผลโดยการส่งค่าสี RGB ผ่าน D/A สีละ 8 บิต ออกไปเลย ความแตกต่างของการแสดงภาพสีและภาพขาวดำคือ ภาพขาวดำจะต้องตั้งให้แม่สีทั้ง 3 สีมีค่าเท่ากัน เนื่องจาก VGA กำหนดให้แม่สีแต่ละสีใช้รีจิสเตอร์ (register) 6 บิต ทำให้แม่สีแต่ละแม่สีแสดงผลได้เพียง 34 ระดับเท่านั้น ยังผลให้เราแสดงภาพ 256 ระดับให้เห็นได้เพียง 64 ระดับเท่านั้น หากต้องการให้เห็นจริงทั้ 256 ระดับต้องแสดงใน true color mode แล้วให้ RGB มีค่าเท่ากัน ซึ่ง โหมดนี้ใช้ รีจิสเตอร์ 8 บิต สำหรับแม่สีแต่ละสี

เกรย์สเกล (gray scale)

เกรย์สเกล หมายถึงความแตกต่างของระดับความเข้มของแสง โดยเกรย์สเกลหนึ่ง ๆ อาจแบ่งเป็น 13 , 20 หรือ 9 ระดับ โดยระดับที่ว่านี้ก็คือ ระดับสีเทา ในภาพหนึ่ง ๆ ถ้าต้องการแบ่งระดับความเข้มแสงหรือระดับสีเทาให้มีหลาย ๆ ค่านั้น จำเป็นอย่างยิ่งที่จะต้องเพิ่มจำนวนบิต ที่แสดงค่าของพิกเซล ตัวอย่างเช่น ถ้าต้องการภาพที่มีระดับสีเทา 4 ระดับต้องแทนด้วยเลขฐานสองจำนวน 2 บิต ถ้าต้องการภาพที่มีระดับสีเทา 16 ระดับต้องแทนด้วยเลขฐานสองจำนวน 4 บิต และถ้าต้องการภาพที่มีระดับสีเทา 256 ระดับต้องแทนด้วยเลขฐานสองจำนวน 8 บิตเป็นต้น

จำนวนระดับสีเทาที่ต้องการนี้ก็คือค่าเลขยกกำลังของ 2 นั่นเอง ซึ่งค่าต่ำสุดหรือ 0 จะแทนสีดำคือไม่มีความสว่างเลย และค่าที่มากที่สุดก็คือค่าที่น้อยกว่าจำนวนระดับสีเทาอยู่ 1 เช่น ค่า 15 ในระบบที่มีระดับสีเทา 16 ระดับก็จะเป็นสีขาวหรือสว่างมากที่สุด เป็นต้น

ในยุคแรก ๆ ระบบการมองเห็น (vision system) จะใช้ระบบเลขฐานสองเพราะสะดวดต่อการนำเซนเซอร์มาใช้ นอกจากนี้การรวบรวมข้อมูล การเก็บรักษาข้อมูลยังสามารถทำได้ง่ายอีกด้วย

ในปัจจุบัน ไมโครโปรเซสเซอร์ที่ใช้กันโดยทั่วไป ขนาดที่เล็กที่สุดก็คือ 8 บิต ดังนั้นเกรย์สเกลขนาด 8 ค่า , 16 ค่า และ 256 ค่า จึงไม่เป็นปัญหาในการประมวลผล

ความสามารถในการแบ่งแยกระดับความแตกต่างของสายตามนุษย์ โดยทั่ว ๆ ไปจะอยู่ระหว่าง 10 ถึง 15 ระดับ ดังนั้น เกรย์สเกลขนาด 16 ระดับ จึงถือได้ว่าใกล้เคียงกับสายตามนุษย์หรืออาจจะละเอียดน้อยกว่าสายตาของมนุษย์บ้างเล็กน้อย (ในบางคน) ในขณะที่เกรย์สเกลขนาด 64 หรือ 256 นั้นละเอียดเกินไปสำหรับมนุษย์

ฮีสโตแกรม (Histogram)

ฮีสโตแกรมคือกราฟที่บอกให้ทราบถึงจำนวนของระดับสีเทาในภาพหนึ่งๆ โดยที่แกน x จะเป็นค่าของระดับสีเทา และแกน y เป็นจำนวนของพิกเซล ในฮีสโตแกรมหนึ่ง ๆ จะประกอบไปด้วย

1. จำนวนพิกเซลทั้งหมดของภาพ
2. จำนวนพิกเซลในแต่ละค่าของระดับสีเทา
3. กราฟที่แสดงจำนวนในแต่ละค่าของระดับสีเทา

กราฟที่ใช้ในฮีสโตแกรมจะเป็นกราฟแท่ง ซึ่งสามารถแสดงจำนวนพิกเซลในแต่ละค่าระดับสีเทาได้เป็นอย่างดี

รูปร่างหรือขนาดของฮีสโตแกรมจะเป็นข้อมูลที่แสดงคุณสมบัติของภาพว่า มีความคมชัด (contrast) มากน้อยเพียงใด ซึ่งข้อมูลนี้ก็คือประโยชน์ของฮีสโตแกรมที่จะใช้ในการกำหนดค่าเทรชโฮล (threshold) ค่าเทรชโฮลนี้จะใช้ในการแปลงรูปภาพให้กลายเป็นภาพที่มีระดับความเข้ม 2 ระดับ คือขาวกับดำ หรือ "0" กับ "1" ตามที่ได้กล่าวมาแล้วในเรื่องพิกเซลนั่นเอง

การสร้างฮีสโตแกรม

1. ต้องกำหนดก่อนว่าภาพที่จะนำมาสร้างฮีสโตแกรมนั้น จะแบ่งเป็นกี่พิกเซล
2. สร้างพิกเซลเมตริกซ์จากพิกเซลเล็ก ๆ
3. นำค่าของพิกเซลในพิกเซลเมตริกซ์ที่ได้จากข้อ 2 มาสร้างตารางความสัมพันธ์ระหว่างค่าระดับสีเทากับจำนวนของพิกเซลในแต่ละค่าระดับสีเทาว่ามีกี่พิกเซล
4. นำค่าที่ได้จากตารางในข้อ 3 มาพล็อตเป็นกราฟแท่ง โดยแกนทางแนวนอนเป็นค่าระดับสีเทา และแกนทางแนวตั้งเป็นแกนของจำนวนของพิกเซล และกราฟนี้คือฮีสโตแกรมนั่นเอง

2.2 ความหมายและนิยามของภาพในระบบดิจิตอล

ภาพ (Image) ในเชิงคณิตศาสตร์จะหมายถึง ฟังก์ชัน 2 มิติ f(x,y) โดย x และ y เป็นแกนพิกัดในระนาบ 2 มิติ ค่าฟังก์ชัน f(x,y) จะเป็นสัดส่วนกับความสว่างหรือความเข้มของภาพที่ตำแหน่ง x,y ซึ่งเราเรียกว่า ระดับสีเทา (Gray Scale) ซึ่งปกติเราจะให้จุดกำเนิดของแกนพิกัด (Coordinate) อยู่ทางมุมซ้ายของภาพ ภาพ 2 มิติที่แทนด้วยฟังก์ชัน f(x,y) โดย x และ y เป็นแกนในระนาบของภาพ ค่าของฟังก์ชั่นที่จุด (x,y) คือความเข้มแสงที่จุดนั้น เนื่องจากแสงเป็นพลังงานรูปหนึ่ง ดังนั้น f(x,y) ต้องไม่เป็นศูนย์ และมีค่า (finite) นั่นคือ

$$0 < f(x,y) < a \qquad (2.1)$$

โดยธรรมชาติของแสง ซึ่งจะต้องมีแหล่ากำเนิดแสงและส่วนที่สะท้อนของแสง ดังนั้นเราสามารถแยกฟังก์ชัน f(x,y) ออกเป็นสองส่วนคือ อิลูมิเนชันคอมโพเนนท์ (illumination component) และ รีเฟล็กแทนท์คอมโพเนนท์ (reflectant component) จะได้ว่า

$$f(x,y) = I(x,y) * r(x,y) \qquad (2.2)$$

เมื่อ

$$0 < I(x,y) < a \qquad (2.3)$$

และ

$$0 < r(x,y) < 1 \qquad (2.4)$$

สมการที่ 2.4 แสดงให้เห็นว่า ฟังก์ชันการสะท้อนถูกตำกัดขอบเขตระหว่าง 0 (ซึ่งหมายถึง การดูดซึมสมบูรณ์) และ 1 (ซึ่งหมายถึง การสะท้อนโดยสมบูรณ์) ธรรมชาติของ i(x,y) ขึ้นอยู่กับแหล่งกำเนิดแสง ในขณะที่ r(x,y) ขึ้นอยู่กับวัตถุที่สะท้อนแสงมาเข้าตา

ดังที่กล่าวมาแล้ว ความเข้มของภาพที่จุด (x,y) เราเรียกว่าระดับสีเทา (gray level) I จากสมการที่ (2.2) ถึง (2.4) จะเห็นว่า I อยู่ในช่วง

$$L_{min} < I < L_{max} \qquad (2.5)$$

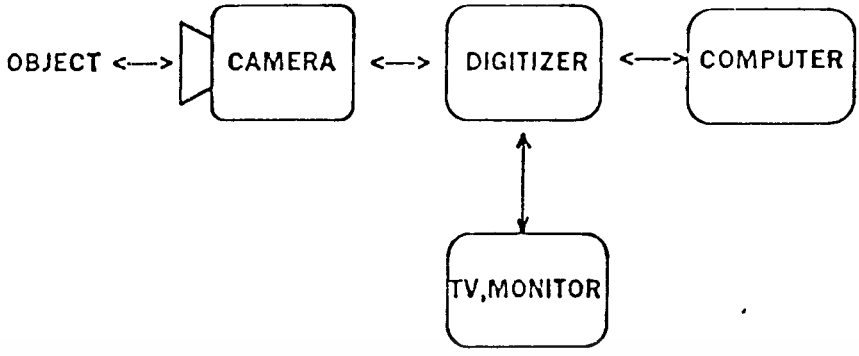
ในทางทฤษฎี L ต้องมีค่าเป็นบวก ในขณะที่ L ต้องมีค่าน้อยกว่าอนันต์ ในทางปฏิบัติ L = Lr และ L = L r ช่วงของ (L ,L) เราเรียกว่าช่วงของระดับสีเทา ในทางปฏิบัติโดยใช้หลักคณิตศาสตร์ เรานิยมปรับช่วง (L ,L) ให้เป็นช่วง (0,L) โดย L = 0 หมายถึง ดำสนิท และ L = 1 หมายถึง ขาว

2.3 การแทนรูปภาพด้วยระบบดิจิตอล

ภาพดิจิตอล (digital image) เป็นภาพที่ถูกแปลงมาจากภาพอนาลอก อยู่ในรูปตัวเลข โดยภาพอนาลอกถูกแบ่งเป็นพื้นที่สี่เหลี่ยมเล็ก ๆ ที่เรียกว่าพิกเซล ในแต่ละพิกเซลจะถูกระบุตำแหน่งโดย (x,y) และค่าระดับสีเทาของ พิกเซล นั้น คือค่าของ f(x,y)

2.4 ระบบการประมวลผลทางดิจิตอล

ระบบการประมวลผลทางดิจิตอลประกอบด้วย 3 ส่วนใหญ่ ๆ คือ ส่วนเปลี่ยนสัญญาณอนาลอก ให้เป็นสัญญาณดิจิตอล ซึ่งเรียกว่า ดิจิไตเซอร์ (Digitizer) ส่วนประมวลผล (Processing) และส่วนแสดงผล (Display) ดังแสดงในรูป



รูปที่ 2.1 ระบบการประมวลผลทางดิจิตอล

จากรูปที่ 2.1 ส่วนแรกคือ ส่วนที่เปลี่ยนสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอล กล้อง (camera) เปรียบเสมือนดวงตาของมนุษย์ ทำหน้าที่เปลี่ยนภาพวัตถุมาเป็นสัญญาณทางไฟฟ้าและส่งให้ดิจิไตเซอร์ (Digitizer) ซึ่งทำหน้าที่เปลี่ยนสัญญาณไฟฟ้าให้เป็นสัญญาณดิจิตอล อุปกรณ์ส่วนนี้ได้แก่ กล้องโทรทัศน์ดิจิไตเซอร์ ซึ่งในภาพประกอบด้วยหลอดที่ทำหน้าที่เป็นสื่อไฟฟ้าทางแสง ภาพถูกโฟกัสลงบนผิวของหลอด และถูกเปลี่ยนให้เป็นสัญญาณไฟฟ้าที่สอดคล้องกับความสว่างของภาพในตำแหน่งนั้น ๆ จากนั้น ทำการควอนไตซิ่ง (quantizing) ข้อมูลภาพที่ได้เป็นสัญญาณดิจิตอล

ส่วนประมวลผล คือ คอมพิวเตอร์ ซึ่งเปรียบเสมือนสมอง ทำหน้าที่ประมวลผลและวิเคราะห์ข้อมูลภาพ

ส่วนแสดงผล ทำหน้าที่เปลี่ยนข้อมูลตัวเลข (ซึ่งเป็นระดับสีเทา) ที่เก็บเป็น อะเรย์ (array) ในคอมพิวเตอร์ ให้อยู่ในรูปที่เหมาะสม และสื่อความหมายกับมนุษย์ได้ คือเป็นภาพที่ปกติทั่ว ๆ ไป อุปกรณ์ในส่วนนี้ได้แก่ จอ (monitor) ทีวี เครื่องพิมพ์ดีดที่สามารถผลในรูปกราฟฟิกส์ได้

ภาพ 1 ภาพ ที่ถูกเปลี่ยนจากสัญญาณดิจิตอล สำหรับคอมพิวเตอร์ที่มีขนาดใหญ่ขึ้นอยู่กับความละเอียดของภาพที่ต้องการ และจะมีผลทำให้ใช้เนื้อที่ในหน่วยความจำมากในการเก็บข้อมูลภาพ 1 ภาพ เช่น การเก็บภาพขนาด 256*256 จุด ที่มีความแตกต่างของระดับความเข้มของแต่ละจุดเท่ากับ 256 ระดับ จะต้องใช้เนื้อที่ในหน่วยความจำถึง 64,000 ไบท์ ในการเก็บภาพนี้ ดังนั้นในปัจจุบันนี้ได้มีการวิจัยหาวิธีที่จะเก็บภาพด้วยคอมพิวเตอร์ โดยใช้เนื้อที่หน่วยความจำให้น้อยที่สุดและยังคงรักษาความละเอียดของภาพตามการใช้งานได้อีกด้วย

2.5การสุ่มแบบสม่ำเสมอและควอนไตเซชัน (Uniform sampling and Quantization)

เพื่อที่จะประมวลผลสัญญาณภาพด้วยระบบคอมพิวเตอร์ ฟังก์ชันของภาพ f(x,y) จะถูกทำให้เป็นสัญญาณไม่ต่อเนื่อง ทั้งระนาบของภาพ ซึ่งเราเรียกว่า การสุ่มภาพ (Image Sampling) ของฟังก์ชันที่ได้เรียกว่า การควอนไตเซชันระดับสีเทา (gray level quantization)

สมมติว่าสัญญาณภาพต่อเนื่อง f(x,y) ถูกดิจิไตซ์ ในระนาบ X Y เป็นช่วงเท่า ๆ กัน เราสามารถจัด f(x,y) ให้อยู่ในรูปเมตริกซ์ขนาด NxN ได้ดังสมการ

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \ldots\ldots\ldots\ldots & f(0,N-1) \\ f(1,0) & f(1,1) & \ldots\ldots\ldots\ldots & f(1,N-1) \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ f(N-1,0) & f(N-1,1) & \ldots\ldots\ldots\ldots & f(N-1,N-1) \end{bmatrix} \qquad (2.6)$$

ทางขวามือของสมการ จะเรียกว่า ภาพดิจิตอล และทุก ๆ สมาชิกของเมตริกซ์จะเรียกว่า พิกเซล จากขบวนการสร้างภาพดิจิตอลข้างต้น จะเห็นว่า เราต้องทราบขนาดความละเอียดของภาพ NxN พิกเซล และจำนวนระดับของ Gray level ในทางปฏิบัติการทำควอนไตซ์เซชันในระบบดิจิตอล จะเป็นค่าของ 2 ยกกำลังจำนวนเต็ม คือ

$$N=2^n \qquad (2.7)$$

$$G=2^m \qquad (2.8)$$

เมื่อ G คือ จำนวนระดับ Gray level ดังนั้นจำนวนบิท (bit) ที่ใช้ในการเก็บภาพหนึ่งภาพที่ถูกดิจิไตซ์ คือ

$$B = N \times N \times m \quad \text{บิท} \qquad (2.9)$$

ดังตัวอย่างขนาด 128 x 128 pixels และระดับ Gray level จำนวน 256 ระดับ ต้องใช้หน่วยความจำขนาด 131,072 บิท

ตารางที่ 2.1 แสดงจำนวนไบท์ที่ใช้ในการเก็บภาพ เมื่อ N และ M เปลี่ยนไป

m n	1	2	3	4	5	6	7	8
32	128	256	512	512	1024	1024	1024	1024
64	512	1024	2048	2048	4096	4096	4096	4096
128	2048	4096	8192	8192	16384	16384	16384	16384
256	8192	16384	32768	32768	65536	65536	65536	65536
512	32764	65536	131072	131072	262144	262144	262144	262144

2.6 เทคนิคต่าง ๆ ในการประมวลผลภาพ

เทคนิคต่าง ๆ สำหรับการประมวลผลภาพ แบ่งเป็น 4 พวกใหญ่ ๆ คือ

1. อิเมจดิจิไตเซชัน (image digitization)
2 อิเมจเอนฮานเมนต์และรีสตอเรชัน (image enhancement and restoration)
3. อิเมจเอ็นโค้ดดิ้ง (image encoding)
4. อิเมจรีคอนสตรัคชั่น (image reconstruction)

2.6.1 อิเมจดิจิไตเซชัน (image digitization)

ดังที่ได้กล่าวมาแล้วถึงความหมายของการ digitize ภาพ ซึ่งความละเอียดของภาพที่ได้ก็ขึ้นอยู่กับการจัดระดับภาพ ในปัจจุบันเครื่องมือที่ใช้ทำกระบวนการนี้เรียกว่า ดิจิไตเซอร์ (digitizer) ดิจิไตเซอร์สามารถเปลี่ยนสัญญาณอนาลอกเป็นสัญญาณดิจิตอลได้ ดังนั้นข้อมูลที่ได้จึงเก็บเป็นเลขไบนารี โดยใช้ดิจิไตเซอร์เป็นตัวจัดการ

2.6.2 อิเมจเอนฮานเมนต์และรีสตอเรชัน (image enhancement and restoration)

อิเมจเอนฮานเมนต์เป็นการทำภาพให้อยู่ในรูปที่เหมาะสมขึ้น มีความคมชัดมากยิ่งขึ้น สำหรับการนำไปใช้งานเฉพาะอย่าง กล่าวคือ วิธีทำภาพ หรือปรับปรุงภาพ X-ray อาจจะไม่เป็นวิธีที่ดี เมื่อนำมาปรับปรุงภาพถ่ายดาวเคราะห์ ที่ส่งมาจากการสำรวจทางอวกาศ

วิธีปรับปรุงคุณภาพของภาพ (enhancement) มีหลายวิธี ดังนี้

1. คอนทราสท์เอนฮานเมนท์ (Contrast enhancement) เป็นวิธีที่ทำให้ภาพคมชัดขึ้น โดยอาศัยวิธีฮีสโตแกรม อาจใช้แบบลิเนียร์สเทรท (linear stretch) , พีซลิเนียร์สเทรท (piecewise linear stretch) หรือ อีควอไลเซชัน (equalization)
2. เอจเอนฮานเมนต์ (Edge enhancement) เป็นการแยกความแตกต่างของจุดภาพที่ใกล้เคียงกัน เพื่อหาขอบเขตของภาพ
3. การประมวลผลภาพสีเทียม (Pseudo-color image processing) เป็นการใช้เทคนิคของการทำ density slicing และการใส่สีเทียมให้กับภาพขาว-ดำ ที่มีระดับ Gray level ต่าง ๆ กัน
4. การกรองภาพ (Filtering) เพื่อให้ภาพเรียบ (smoothing) หรือ คมชัด (sharpening) โดยใช้ตัวกรองความถี่ต่ำ (low pass filter) หรือ ตัวกรองความถี่สูง (high pass filter) ตามลำดับ

อิเมจรีสตอเรชัน (Image restoration)

เป็นขบวนการในการสร้างภาพกลับคืน โดยการหาค่าชดเชย และแก้ความคลาดเคลื่อนเนื่องมาจากข้อมูลในภาพผิดพลาดไป หรือเป็นขบวนการสร้างภาพกลับคืน จากภาพที่ถูกทำให้เสียไป เนื่องจากปรากฏการณ์ต่าง ๆ โดยใช้หลักการของพีชคณิตเชิงเส้น (linear algrbra)

2.6.3 อิเมจเอ็นโค๊ตดิ้ง (image encoding)

เป็นการใช้เทคนิคต่าง ๆ เพื่อเข้ารหัสข้อมูล เนื่องจากข้อมูลภาพที่ได้จะถูกเก็บในลักษณะเป็นจำนวนไบท์ ดังตาราง 2.1 ซึ่งถ้าภาพมีขนาดใหญ่ ก็ต้องใช้พื้นที่ในการเก็บมาก ด้วยข้อจำกัดของไมโครคอมพิวเตอร์ ที่มีขนาดหน่วยความจำจำกัด

การเข้ารหัสข้อมูลจึงมีประโยชน์ ในด้านการลดพื้นที่ในการเก็บข้อมูลภาพดังกล่าวมาก ผลของการเข้ารหัสข้อมูลนี้ เรียกว่าเป็นการลดข้อมูล (Data reduction หรือ Data compression) ซึ่งเป็นเนื้อหาที่ทำโครงงานนี้ รายละเอียดของการลดข้อมูลภาพ ได้อธิบายในบทที่ 3 นอกจากนี้ การเข้ารหัสข้อมูลยังมีรหัสข้อมูลยังมีประโยชน์ในการช่วยลดปริมาณข้อมูลภาพ ที่ใช้ในระบบการสื่อสาร เช่น การส่งภาพถ่ายจากอวกาศมายังโลก การส่งข้อมูลภาพผ่านโมเด็ม (modem) เป็นต้น

2.6.4 อิเมจรีคอนสตรัคชัน (image reconstruction)

เป็นวิธีการสร้างภาพตัดขวางของวัตถุ โดยไม่ต้องผ่า หรือทำลายวัสดุ เพื่อประมวลผลโดยใช้คอมพิวเตอร์ เราเรียกการสร้างภาพตัดภาพตัดขวางด้วยคอมพิวเตอร์ว่า คอมพิวเตอร์โทโมกราฟฟี่ (Computer tomography)

บทที่ 3
การลดข้อมูลภาพ

สัญญาณภาพโดยธรรมชาติแล้วจะเป็นสัญญาณต่อเนื่อง (Analog) การนำข้อมูลภาพมาใช้ในการประมวลผลในด้านต่างๆ ที่มีเครื่องคอมพิวเตอร์เป็นตัวประมวลผลแล้วนั้น สัญญาณจะต้องแปลงให้เป็นสัญญาณที่เครื่องสามารถเข้าใจได้เสียก่อน นั้นคือต้องแปลงให้อยู่ในรูปของสัญญาณดิจิตอลซึ่งมีลักษณะที่ไม่ต่อเนื่อง (discrete siganl) โดยผ่านการสุ่มและจัดระดับ (sample and quantised) เมื่อรายละเอียดของภาพหนึ่ง ๆ ถูกแทนที่ด้วยข้อมูลแบบดิจิตอล ขนาดของข้อมูลภาพจึงมีจำนวนสูงมากเพื่อที่จะสามารถเก็บรายละเอียดของภาพได้เพียงพอ ทำให้ต้องใช้ขนาดของหน่วยความจำในการเก็บข้อมูลเหล่านี้มีขนาดใหญ่ตามไปด้วย ในกรณีที่มีการรับส่งข้อมูลภาพเหล่านี้ด้วยแล้วจะต้องเสียเวลารับส่งเป็นจำนวนมาก โดยเฉพาะเป็นการสื่อสารข้อมูลผ่านดาวเทียมด้วยแล้ว เวลาที่มากนั้นก็หมายถึงค่าใช้จ่ายที่สูงมากตามไปด้วย ดังนั้นจึงจำเป็นอย่างยิ่งที่จะต้องลดเวลาในการส่งข้อมูลให้น้อยที่สุดเท่าที่จะเป็นไปได้

การลดขนาดข้อมูลภาพ จึงถูกนำมาใช้ในการแก้ปัญหาเหล่านี้ เพื่อเป็นการประหยัดทางด้านเศรษฐกิจในส่วนของจำนวนหน่วยความจำที่ลดลง และในส่วนของเวลาที่ใช้ไปในการรับส่งข้อมูลภาพ สิ่งที่เป็นตัวกำหนดว่าข้อมูลของภาพสามารถที่จะลดลงไปได้เท่าไรนั้นขึ้นอยู่กับงานที่ใช้ว่าต้องการรายละเอียดมากน้อยเพียงใด เมื่อทำการลดข้อมูลแล้วจึงจะสามารถนำข้อมูลกลับมา(reconstruct) ได้อย่างเหมาะสม ซึ่งงานแต่ละอย่างมีความต้องการรายละเอียดของภาพที่ไม่เท่ากัน อย่างเช่นถ้านำไปใช้ในการตรวจสอบและจดจำรูปแบบของวัตถุต่างๆ โดยใช้คอมพิวเตอร์แล้ว ภาพที่ใช้จะเน้นเฉพาะส่วนของขอบวัตถุในภาพนั้นๆก็เพียงพอ โดยไม่จำเป็นต้องมีรายละเอียดโครงสร้างภายในของภาพ ในขณะที่การลดข้อมูลภาพเพื่อการลดเวลาที่ใช้ไปในการรับส่งภาพผ่านช่องสัญญาณความเร็วต่ำอย่างเช่น ระบบโครงข่ายโทรศัพท์สาธารณะนั้น มีความจำเป็นอย่างยิ่งที่จะต้องเก็บรายละเอียดต่างๆของภาพให้ได้มากที่สุด หลักการพื้นฐานที่นิยมใช้ในการลอกข้อมูลภาพมีอยู่สามหลักการด้วยกัน คือ พรีดิกทีฟโค๊ดดิ้ง ซึ่งเป็นการเข้ารหัสใน โดเมนความถี่ หลักการที่สอง คือ Transform Coding เป็นวิธีที่ประมวลผล(process)ในโดเมนของความถี่ ส่วนหลักการสุดท้ายเป็นการรวมเอาข้อดีต่างๆจากสองหลักการแรกเข้าด้วยกัน เรียกว่า ไฮบริดโค๊ดดิ้ง

3.1 พรีดิกทีฟโค๊ดดิ้ง (predictive coding)

การลดขนาดข้อมูลภาพด้วยวิธี พรีดิกทีฟโค๊ดดิ้งนี้ เป็นการอาศัยคุณสมบัติของข้อมูลภาพที่มักจะมีค่าซ้ำๆกัน และเมื่อข้อมูลอินพุทถูกกำหนดให้มีความเกี่ยวพันธ์กันนั้นก็คือจุดภาพที่มีตำแหน่งอยู่ใกล้ๆกัน มักจะมีค่าระดับแอมปลิจูดใกล้เคียงกันหรือเท่ากัน ดังนั้นจึงสามารถที่จะใช้ค่าของจุดภาพหนึ่งจุดหรือหลายๆจุด ที่ผ่านมาใน Line นั้น Line ก่อนหน้านั้นหรือในเฟรมที่ผ่านมา เป็นตัวคาดคะเนหรือแทนค่าของจุดภาพปัจจุบัน ซึ่งโดยธรรมชาติทางสถิติของข้อมูลภาพเราสามารถที่จะคาดคะเนค่าของข้อมูลได้ไม่ผิดพลาดมากนัก จากค่าที่ได้จากการคาดคะเนนี้เอาไปลบกับค่าจริงของจุดภาพ จะได้เป็นค่าความแตกต่างระหว่างค่าจริงกับค่าที่เราคาดคะเนเอาไว้ ซึ่งค่านี้จะมีขนาดเล็ก และค่าผลต่างนี้จะถูกนำไปเข้ารหัสเพื่อจะเก็บไว้ใช้ในตอนถอดรหัส พร้อมกับค่าที่เราคาดคะเนเอาไว้ ดังนั้นในการที่จะเก็บในหน่วยความจำหรือต้องการส่งก็ใช้ค่าสองค่านี้ เมื่อถึงตอนที่จะนำภาพเดิมกลับมาหรือตอนถอดรหัส จะนำเอาค่าที่เราคาดคะเนไว้ในตอนแรกบวกกับค่าของผลต่างของจุดภาพนั้นกับค่าคาดคะเน ก็จะได้เป็นค่าของจุดภาพนั้นๆ ค่าผิดพลาดที่ได้ในตอนถอดรหัสเกิดขึ้นเพียงกรณีเดียวเท่านั้นคือ ตอนจัดระดับ (quantised) ค่าความแตกต่างของการเข้ารหัสเท่านั้นวิธีนี้เป็นวิธีที่ง่ายแก่การสร้างระบบ และสามารถลดขนาดของภาพให้เหลือประมาณ 1-2 bit/element

3.2 ทรานส์ฟอร์มโค๊ดดิ้ง (Transform Coding)

การลดข้อมูลภาพด้วยวิธีของ ทราส์ฟอร์มโค๊ดดิ้งนี้เป็นวิธีที่ซับซ้อน และมีขั้นตอนมากกว่าวิธีลดข้อมูลภาพโดยพรีดิกทีฟโค๊ดดิ้ง หลักการของวิธีทรานส์ฟอร์มโค๊ดดิ้งจะทำการแปลงข้อมูลอินพุทที่อยู่ในรูปของโดเมนข้อมูล (data domain) ให้อยู่ในรูปของแถบความถี่ (spectral) หรือโดเมนความถี่ (frequency domain) โดยวิธีการแปลง แบบต่างๆ เช่น ฟูเรียทรานส์ฟอร์ม (Fourier Transform) จะเป็นการแปลงข้อมูลที่อยู่ในรูปของสเปตเชียลโดเมน (spatial domain) ให้อยู่ในรูปสัมประสิทธิ์ของพลังงานความถี่ โดยความถี่ต่ำ ๆ จะมีพลังงานสูง ที่ความถี่สูงพลังงานจะลดลงไป การเข้ารหัสจึงใช้จำนวนบิทสำหรับแต่ละช่วงความถี่ไม่เท่ากัน เมื่อต้องการอัตราบิทเรทสูง ๆ ค่าของพลังงานความถี่สูงจะถูกตัดทิ้งเป็นส่วนใหญ่ เป็นเหตุให้รายละเอียดส่วนที่เป็นรายละเอียดส่วนที่เป็นขอบภาพหายไป ทำให้ภาพที่ได้เบลอ ขาดความคมชัด

การแปลงที่นิยมใช้ในการลดข้อมูลภาพมีอยู่ด้วยกันหลายวิธี เช่น Fast Fourier Transform (FFT) , Fast-Hadamard Transform , Fast Slant Transform ,Fast Discrete Cosing Transform ,Fast Discrete Sine Transform เป็นต้น ซึ่งในแต่ละวิธีก็มีข้อดีข้อเสียที่แตกต่างกันไป

การ แปลงที่นิยมใช้กันมากได้แก่ Discrete Cosine Transform เพราะเป็นวิธีที่สามารถคำนวณได้ง่ายเนื่องจากมีการคำนวณเฉพาะค่าจริงไม่ใช้ค่าจินตภาพ (imaginary)

วิธีทรานส์ฟอร์มโค๊ดดิ้ง ถึงแม้ว่าจะเป็นวิธีที่ยุ่งยากแต่ก็สามารถสร้างระบบได้ด้วยอุปกรณ์ทางฮาร์ดแวร์ (Hardware) ดิจิตอลความเร็วสูงได้ง่าย และเป็นระบบที่ยืดหยุ่น (adaptive system) สามารถที่จะลดขนาดของข้อมูลให้อยู่ในอัตราบิทเรทประมาณ 0.5 - 1 บิทต่อจุดภาพ ซึ่งเป็นช่วงอัตราการลดที่สามารถสร้างภาพเดิมกลับมาได้สมบูรณ์เพียงพอต่อการนำไปใช้งานต่าง ๆ

3.3 ไฮบริดโค๊ดดิ้ง (Hybrid Coding)

ไฮบริดโค๊ดดิ้งเป็นเทคนิคที่นำข้อดีของวิธีพรีดิกทีฟโค๊ดดิ้ง และ ทรานส์ฟอร์มโค๊ดดิ้ง มาใช้ร่วมกัน ทั้งนี้เพราะบางครั้งการลดขนาดข้อมูลภาพด้วยทรานส์ฟอร์มโค๊ดดิ้ง ไม่อาจจะให้รายละเอียดของภาพเพียงพอ ดังนั้นการเพิ่มรายละเอียดของภาพจึงต้องใช้ความสัมพันธ์ของจุดภาพที่มีอยู่ที่มีอยู่ทั้งทางแนวนอนและทางแนวตั้ง โดยการใช้การแปลงแบบมิติเดียวในทิศทางแกนใดแกนหนึ่งของภาพ เช่นการแปลงกับข้อมูลภาพในทางแนวนอนต่อจากนั้นจึงใช้พรีดิกทีฟโค๊ดดิ้ง กับสัมประสิทธิ์ของการแปลงที่ได้ เพื่อที่จะใช้ในการคาดคะเนค่าของกลุ่มสัมประสิทธิ์ที่เหมือน ๆ กัน ซึ่งเหมือนกับการทำพรีดิกทีฟโค๊ดดิ้ง ใน นั่นเอง และในกรณีของการแปลง แบบ 2 มิติ สามารถทำได้เร็วขึ้นอาจจะใช้การทำพรีดิกทีฟโค๊ดดิ้ง กับค่าสัมประสิทธิ์ของการแปลงของภาพในบริเวณเดียวกันแต่เป็นเฟรมที่แตกต่างกันที่ตามมา การใช้ไฮบริดโค๊ดดิ้ง ในลักษณะนี้สามารถเป็นไปได้อีกลักษณะหนึ่งคือ ใช้การใช้พรีดิกทีฟโค๊ดดิ้ง ในขั้นตอนแรกก่อนแล้วจึงทำการแปลง แต่ในกรณีนี้ต้องการระบบที่ซับซ้อนมากกว่า

การลดข้อมูลภาพด้วยวิธีของไฮบริดโค๊ดดิ้งนี้ อาจจะพูดได้ว่ามีคุณสมบัติอยู่ระหว่างวิธีของ พรีดิกทีฟโค๊ดดิ้ง และ ทรานส์ฟอร์มโค๊ดดิ้ง ดังนั้นอัตราบิทเรทต่ำสุดของวิธีนี้จึงไม่สามารถทำให้ต่ำกว่าวิธีของทรานส์ฟอร์มโค๊ดดิ้ง เพียงแต่สามารถสร้างได้ง่ายกว่า โดยอัตราบิทเรทของระบบจะมีค่าประมาณ 1 บิตต่อจุดภาพ ซึ่งอัตราบิทเรทที่สามารถสร้างภาพเดิมกลับมาใหม่ (reconstruction) ได้ภาพที่มีคุณภาพดีพอสมควร

3.4 การลดข้อมูลภาพด้วยวิธีรันเลงจ์ (Runlength Compression)

การลดข้อมูลด้วยวิธีนี้ อาศัยลักษณะทั่วไปของที่จะต้องมีส่วนของฉากหลัง (background) และ พื้นหน้า (foreground) ในส่วนของ ฉากหลังจะมีรายละเอียดของภาพไม่มากนัก ส่วนนี้เองจะมีการเปลี่ยนแปลงของข้อมูลน้อย เมื่อเทียบกับส่วนของ foreground ซึ่งมีรายละเอียดและการเปลี่ยน

แปลงของข้อมูลมาก ในส่วนที่มีการเปลี่ยนแปลงข้อมูลน้อยนี้เองที่เราสามารถนำการเข้ารหัสแบบรั่นเลนจ์มาประยุกต์ใช้ได้อย่างมีประสิทธิภาพ

การเข้ารหัสแบบนี้ จะจัดข้อมูลภาพเดิม ให้อยู่ในรูปของคู่ลำดับ (G ,L) โดย G แทนระดับความเข้ม หรือระดับสีเทา L แทนความยาวของข้อมูลหรือ จำนวนจุดที่มีระดับสีเทา G การเข้ารหัสแบบนี้มีด้วยกัน 2 วิธีใหญ่คือ

วิธีที่ 1 จะทำการอ่านข้อมูลเข้ามาโดยนับจำนวนข้อมูลที่ซ้ำกันกับข้อมูลนั้นเข้ามาด้วย แล้วแปลงข้อมูลเข้าไปเป็น 2 ไบท์ คือ ไบท์แรกจะเก็บจำนวนตัวข้อมูลที่ซ้ำกัน โดยจะมีไบท์ที่สองเก็บค่าของระดับสีที่ซ้ำกันนั้นเอาไว้ โดยใน 1 ชุดข้อมูลเอาท์พุท (2 ไบท์) จะนับจำนวน จุดที่ซ้ำกันได้ 256 จุดสี ตัวอย่างการเข้ารหัสข้อมูล เช่น

ข้อมูลเป็น(ค่าที่แสดงเป็นเลนฐาน 16)

AC AC AC AC 15 15 15 15 15 15 45 45 78 20 10 10 10 10

เข้ารหัสได้เป็น

04 AC 06 15 02 45 01 78 01 20 04 10

จะเห็นได้ว่าเราสามารถลดข้อมูลจาก 19 ไบท์ เหลือ 12 ไบท์ ซึ่งหากข้อมูลซ้ำกันถึง 256 จุด แล้วจะทำให้เราลดข้อมูลไปได้อย่างมหาศาล แต่ในขณะเดียวกันหากเกิดกรณีที่แย่ที่สุดของการใช้วิธีนี้ก็คือ กรณีที่ข้อมูลแต่ละตัวไม่ซ้ำกับจุดข้างเคียงเลย ซึ่งจะทำผลของการเข้ารหัสจะได้รหัสที่ยาวเป็น 2 เท่าของข้อมูลอินพุท

วิธีที่ 2 จากความบกพร่องของวิธีการทำรันเลนจ์เอ็นโค๊ดดิ้ง วิธีแรก ตรงที่โปรแกรมจะทำการเข้ารหัสข้อมูล เมื่อนับจำนวนข้อมูลที่ได้ตั้งแต่ 1-255 จุด ดังนั้นถ้าเกิดข้อมูลของเรามี จุดสีที่อยู่โดด ๆ (ไม่ซ้ำกับจุดข้างเคียง) อยู่มากมายจะทำให้การเข้ารหัสไม่ทำให้ข้อมูลเล็กลงมากนัก หรืออาจจะใหญ่กว่าข้อมูลเดิมด้วยซ้ำไป และพิกเซลที่ซ้ำกันทีเดียวถึง 255 ตัว คงเกิดขึ้นไม่บ่อยนัก จึงได้มีแนวความคิดที่จะทำการแก้ไขปัญหาของการทำรันเลนจ์เอ็นโค๊ดดิ้ง วิธีแรกเกี่ยวกับการมีข้อมูลที่ไม่ซ้ำกันกับตัวข้างเคียง โดยมีหลักการอยู่ 2 ข้อดังนี้

1. จะไม่ทำการลดข้อมูลกับส่วนที่มีจำนวนซ้ำกันน้อยกว่า 3 ตัว

2. ให้มีการทำเครื่องหมายเพื่อที่จะแยกส่วนที่มีการลดข้อมูลกับส่วนที่ไม่มีการลดข้อมูลออกจากกัน โดยใช้บิท บิทหนึ่งเป็นตัวบอกว่าถูกลดขนาดหรือไม่

จากหลักการดังกล่าวนี้ จะใช้บิทที่ 7 ของไบท์บอกขนาดเป็นตัวบอกว่าข้อมูลที่ต่อจากนี้ไปมีการลดขนาดข้อมูลหรือไม่ ส่วนบิทที่เหลือเราก็เราก็ยังคงใช้บอกขนาดต่อไป เมื่อเรานับพิกเซลได้ซ้ำกันตั้งแต่ 3 ตัวขึ้นไป บิทที่ 7 ของไบท์บอกขนาดจะถูกเซทให้เป็น 1 หรือแบะเราจะเริ่มนับ 1 ตั้งแต่พิกเซล ที่ซ้ำกันตั้งแต่ตัวที่ 4 เป็นต้นไป ทำให้บิทบอกขนาดมีค่าได้ตั้งแต่ 3-130 หรือใน 2

ไบท์ นี้ เราอาจเก็บข้อมูลได้ถึง 130 ไบท์ ส่วนในกรณีที่ไม่มีการลดขนาดบิทที่ 7 ของไบท์ บอกขนาดก็จะถูก set ให้เป็น 0 และ 7 บิท ที่เหลือจะบอกจำนวนของข้อมูลที่มีค่าไม่ซ้ำกันนั้นโดยค่าที่ตามมาจะเป็นข้อมูลที่ไม่ถูกลดขนาด ซึ่งจะมีค่าระหว่าง 0 ถึง 127 โดยจำนวนจริงจะมีค่าเท่ากับไบท์บอกขนาดบวกหนึ่ง ตัวอย่างเช่น

ข้อมูลเป็น (ค่าที่แสดงเป็นเลขฐาน 16)

AC AC AC AC 15 15 15 15 15 15 45 45 78 20 10 10 10 10

เข้ารหัสได้เป็น

81 AC 83 15 03 45 45 78 20 81 10

ความหมาย : 81 (ถูกลดขนาดลง 4 ไบท์มีค่า AC)

: 83 (ถูกลดขนาดลง 8 ไบท์มีค่า 15)

: 03 (ถูกลดขนาดลง 4 ไบท์มีค่า 45 45 78 20)

: 81 (ถูกลดขนาดลง 4 ไบท์มีค่า 10)

โดยวิธีนี้ผลลัพธ์ที่ได้จะแย่ที่สุดในกรณีที่เกิดมีข้อมูลที่ไม่ซ้ำกันเลยมากกว่า 128 ไบท์ เมื่อทำการเข้ารหัสจะได้รหัสถึง 129 ไบท์ สำหรับทุก ๆ 128 ไบท์ของข้อมูลอินพุท และในกรณีที่ดีที่สุดก็คือเมื่อข้อมูลซ้ำกันถึง 128 ไบท์ เมื่อเข้ารหัสแล้วจะได้ข้อมูลที่ลดลงเหลือเพียง 2 ไบท์

3.5 ความสามารถในการลดข้อมูลภาพด้วยรันเลนจ์

ในกรณีที่ข้อมูลภาพมีความแตกต่างของระดับเทามากอัตราของข้อมูลต่อ 1 จุดภาพ (data bit rate) จะมากตามไปด้วย เนื่องจากมีความเป็นไปได้ที่ข้อมูลจะมีการเปลี่ยนแปลงสูง

การลดข้อมูลด้วยวิธีรันเลนจ์นี้จะไม่มีความคลาดเคลื่อนเกิดขึ้น หลังจากการถอดรหัส การใช้วิธีนี้ต้องพิจารณาถึงลักษณะของภาพต้นแบบว่ามีความเหมาะสมหรือไม่ ควรมีรายละเอียดของภาพน้อย เช่น ภาพเอกสาร ภาพทางด้านกราฟจากตาราง เราสามารถลดข้อมูลภาพได้ ดียิ่งขึ้นได้โดยการปรับข้อมูลให้มีความแตกต่างของระดับสีเทาให้น้อยลง ด้วยวิธีของ gray scale transmission

3.6 ข่าวสาร ปริมาณข่าวสารเฉลี่ย และส่วนที่ซ้ำ

ข่าวสาร (information) ในความหมายของ ทฤษฎีข่าวสาร คือปริมาณความไม่แน่นอนของเหตุการณ์ (measure of uncertainly) เช่น สำหรับเหตุการณ์ (event) ที่เกิดค่อนข้างแน่นอน เหตุการณ์นั้นมีข่าวสารน้อย สำหรับเหตุการณ์ที่เกิดไม่มากครั้ง แสดงว่าเหตุการณ์นั้นมีข่าวสารมาก

การวัดปริมาณข่าวสารของเหตุการณ์ คำนวนได้จาก

$$I_i = - \log_2 P_i \qquad \text{บิท}$$

โดยที่ P_i คือความน่าจะเกิด (probability) ของเหตุการณ์ I

ปริมาณข่าวสาร ของเหตุการณ์ I บอกเราว่าควรจะใช้เนื้อที่ในการเก็บเหตุการณ์ I เท่าไร ตัวอย่าง เช่น เหตุการณ์ x มีความน่าจะเกิดเท่ากับ 1/8 แล้วแสดงว่า เนื้อที่ที่ใช้เก็บเหตุการณ์ x คือ 3 บิท

ปริมาณข่าวสารเฉลี่ยของเหตุการณ์ทั้งหมดหรือ entropy คำนวณได้จาก

$$H = -(P_1 \log_2 P_1 + P_2 \log_2 P_2 + \ldots + P_m \log_2 P_m)$$

โดยที่ M คือ เหตุการณ์ที่เป็นไปได้ทั้งหมด

ปริมาณข่าวสารเฉลี่ย บอกเราว่าควรจะเก็บข้อมูลทั้งหมดด้วยอย่างน้อยที่สุดเท่าไร เช่น ข้อมูล 'e' มีความน่าจะเกิดเท่ากับ 1/16 ดังนั้นข้อมูล 'eeeee' ใช้เนื้อที่ในการเก็บเท่ากับ 20 บิท

ข่าวสารเฉลี่ย มีเงื่อนไขที่น่าสนใจ 2 กรณี คือ

1. เมื่อ M = 1 คือมีเหตุการณ์เดียว ดังนั้น $P_i = 1$ ทำให้ H = 0 หมายความว่า ไม่มีข่าวสารเฉลี่ยของเหตุการณ์ที่ทราบแน่นอนแล้ว

2. เมื่อ $P_i = 0$ ทำให้ H = 0 หมายความว่าไม่มีข่าวสารเฉลี่ยของเหตุการณ์ที่เป็นไปไม่ได้

นั่นคือ $H = - P\log_2 P = 0$ (เมื่อ P เข้าใกล้ 0)

ส่วนที่ซ้ำเกิดขึ้นเนื่องจาก การกระจายของข้อมูลไม่เท่ากัน คือความน่าจะเกิดของข้อมูลทุกตัวไม่เท่ากัน ถ้าการกระจายของข้อมูลเท่ากัน แล้ว $H = \log_2 M$ ซึ่ง H จะมีค่าสูงสุดดังนั้น R =0 แต่ข้อมูลโดยทั่วไปมักกระจายไม่เท่ากัน

ดังนั้นเทคนิคการลดขนาดข้อมูลจะใช้ประโยชน์ จากการกระจายไม่เท่ากันของข้อมูลนี้ เพื่อลดขนาดของข้อมูล โดยการเข้ารหัสข้อมูลด้วยรหัสที่เท่ากับ หรือ ต่างจาก entropy น้อยที่สุด ตัวอย่างของรหัสที่เข้าข้อมูล คือ Huffman หรือ Shannon Fano

3.7 ดิสครีตโคไซน์ทรานส์ฟอร์มโค๊ดดิ้ง (Discrete Cosine Transform coding)

ดิสครีตโคไซน์ทรานส์ฟอร์มโค๊ดดิ้งเป็นวิธีการที่ใช้ในการลดข้อมูลภาพที่ได้รับความนิยมอย่างแพร่หลาย ทั้งนี้เพราะค่าสัมประสิทธิ์ในโดเมนของความถี่ที่ได้จะเป็นเทอมของค่าจริง (real time) เท่านั้น อีกทั้งยังสามารถคำนวณได้แบบรวดเร็ว (fast algorithms) และยังสามารถใช้งานจริงในลักษณะ real time โดยใช้ฮาร์ดแวร์ได้ไม่ยาก ในปัจจุบันหลักการของดิสครีตโคไซน์ทราสฟอร์มโค๊ดดิ้ง ยังคงมีการวิจัยกันอยู่ต่อไปเพื่อให้สามารถลดขนาดของย้อมูลให้ได้มากที่สุดพร้อมทั้งหาวิธีที่จะเพิ่มความเร็วในการคำนวณให้ได้เร็วขึ้นไปอีก

ดังนั้นในที่นี้จึงยกตัวอย่าง ระบบการลดข้อมูลภาพโดยใช้เทคนิคของดิสครีตโคไซน์ทรานส์ฟอร์มโค๊ดดิ้ง โดยมีโครงสร้างของระบบเป็นดังรูปที่ 3.1 จาก block diagram ของการแปลงภาพอินพุทที่เข้ามาจะถูกแยกออกเป็นบล็อกเล็กๆ โดยเราสามารถกำหนดขนาดของบล็อกได้ว่าให้เป็นเท่าไร ขนาดของบล็อกที่เหมาะสมจะเป็นตัวเพิ่มประสิทธิภาพในการรวมพลังงานของการแปลง ซึ่งจะทำให้ภาพที่ได้มีรายละเอียดที่ดี ความเหมาะสมของขนาดของบล็อกค่าหนึ่งๆ จะเหมาะสมที่ค่าบิทเรทค่าหนึ่งๆ แต่อย่างไรก็ตามขนาดของบล็อกที่เหมาะสมสามารถคำนวณได้ด้วยคณิตศาสตร์ที่ยุ่งยากพอๆ กับการแปลง เลยที่เดียว โดยส่วนมากแล้วขนาดของบล็อกจะมีค่าเป็นเลขยกกำลังของสองอย่างเช่น 4,8,16 ซึ่งได้มาจาก 2^2, 2^3, 2^4 เป็นต้น หลังจากการแยกข้อมูลออกเป็นบล็อกย่อยๆแล้ว ก็จะทำการแปลง ไปโดยอิสระของการแปลงแบบมิติเดียว และ สองมิติ ดังต่อไปนี้เป็น

$$F(k) = \sqrt{\frac{2}{N}}\alpha(k)\sum_{n-1}^{N-1} f(n)\cos[\frac{(2n+1)k\pi}{2N}]$$

เมื่อ $\alpha(0) = \sqrt{\frac{1}{2}}$ และ $\alpha(k) = 1$ เมื่อ k ไม่เท่ากับ 0

F(k) เป็นผลที่ได้จากการ Transform และ f(n) เป็นข้อมูลอินพุทตัวที่ n

ส่วนสมการของการ Transform แบบสองมิติสามารถเขียนได้ดังนี้คือ

$$F(u,v) = \frac{2}{N}\alpha(u)\alpha(v)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(m,n)\cos[\frac{(2m+1)u\pi}{2N}]\cos[\frac{(2n+1)v\pi}{2N}]$$

เมื่อ

u,v เป็นตัวแปรของ discrete frequency มีค่าเป็น 0, 1, 2, ...N-1

f(m,n) เป็นตำแหน่งของจุดภาพภายในบล็อกขนาด N x N (0, 1, 2,...,N-1)

F(u,v) เป็นผลจากการทำ discrete cosine Transform

$\alpha(0) = \sqrt{\frac{1}{2}}$ และ $\alpha(j) = 1$ เมื่อ j ไม่เท่ากับ 0

3.8 ฮัฟแมนโคดดิ้ง (Huffman Coding)

ผู้ที่คิดวิธีการลดขนาดข้อมูลนี้คือ นาย D.A. Huffman ศิษย์เก่าจาก MIT เขาได้เสนอกระบวนการเข้าระหัส แบบนี้มาตั้งแต่ต้นทศวรรษ 1950 สำหรับหลักการสำคัญของการลดข้อมูลนี้คือการลดจำนวนบิทที่ใช้แทนตัวอักษรที่มีการใช้บ่อยๆ และเพิ่มจำนวนบิทที่ใช้แทนตัวอักษรที่มีการใช้น้อยๆ

การทำงานชนิดนี้อยู่บนพื้นฐานความจริงที่ว่า ในประโยคหรือไฟล์ข้อมูลที่เป็นภาษาอังกฤษหนึ่ง ๆ จะมีการใช้อักษรภาษาอังกฤษซ้ำไปซ้ำมาเสมอ และบางตัวก็ไม่ได้ใช้บ่อยสักเท่าไร ถ้าภาษาอังกฤษตัวหนึ่งถูกแทนด้วยเลขฐานสองจำนวน 8 บิท แล้วเราพบว่าในไฟล์ข้อมูลมีตัวอักษร "a" ด้วยรหัส 4 บิท และให้อักษร "z" แทนด้วยรหัส 14 บิท เป็นต้น

ปัจจุบันนี้ได้มีโปรแกรมแอพพลิเคชันหลายๆ ชนิดที่ได้ประยุตก์ใช้วิธีการลดขนาดของฮัฟแม่นนี้ เช่น การลดขนาดแบบ MNP-5 ของโมเด็ม ซึ่งได้ใช้การลดข้อมูลฮัฟแมนแบบไดนามิก การเข้ารหัสแบบ Shannon-Fano การเข้ารหัสบางส่วนของโปรแกรม PKZIP การทำงานตอนท้ายการลดขนาดแบบ JPEG

การสร้างรหัสของฮัฟแมนก่อนอื่นอ่านข้อมูลหนึ่งรอบ เพื่อหาความถี่ของแต่ละข้อมูล จากนั้นเรียงลำดับของข้อมูล ที่มีความถี่มากอยู่ด้านหนึ่ง และข้อมูลที่มีความถี่น้อยอยู่อีกด้านหนึ่ง และกำหนดให้ข้อมูลแต่ละตัวคือโนด (Node) ใดๆ ของทรี (tree) แล้วนำมาสร้าง ทรี ตามวิธีต่อไปนี้

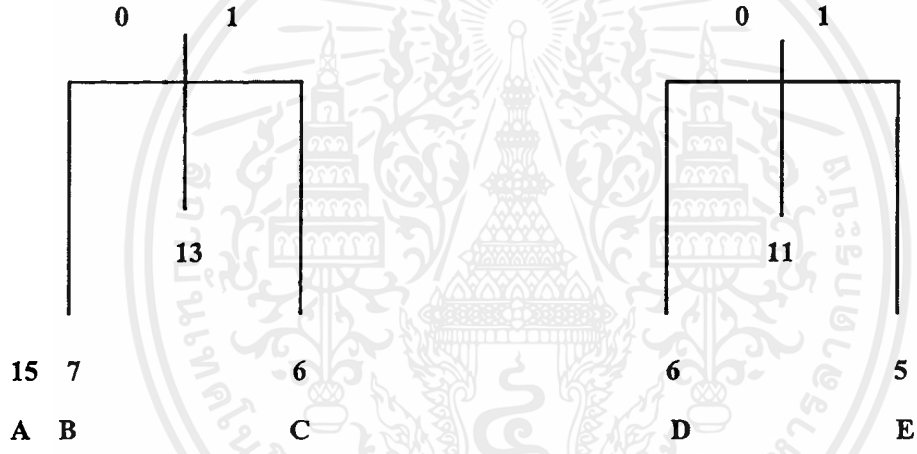
1. นำเอาข้อมูลหรือโนดที่มีความถี่น้อยที่สุดมาเป็น โนดลูก (Child Node) ของโนดพ่อ (Parent Node) ซึ่งความถี่ของโนดพ่อนี้จะเท่ากับผลรวมของความถี่ของโนดลูกทั้งสอง
2. โนดพ่อจะถูกนำไปรวมกับโนดอื่น ๆ ที่เหลือและโนดลูกที่ใช้แล้วทั้งสองจะไม่ถูกนำมาใช้อีก
3. โนดลูก หนึ่งจะถูกกำหนดให้เป็นบิท 0 และอีกโนดหนึ่งให้เป็นบิท 1
4. กระทำซ้ำตั้งแต่ 1 จนกระทั่งเหลือโนดเดียว เรียกว่า รากของทรี

ตัวอย่างเช่นข้อมูลมีลักษณะดังนี้

"AEDBAACBADCABECADACEDADACAEBABABEADABAD"

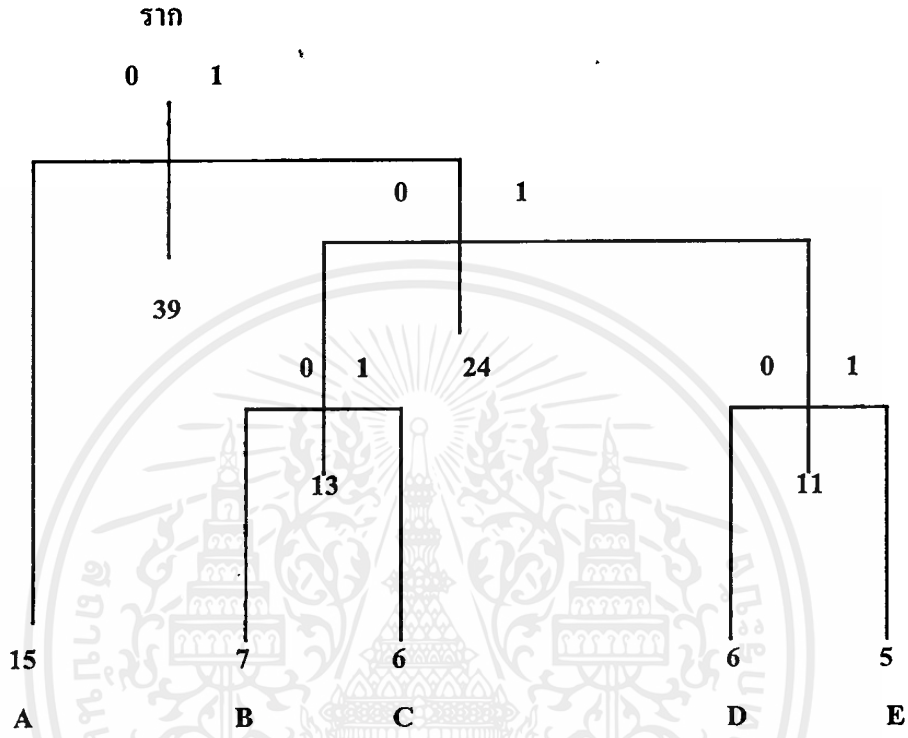
ความถี่	15	7	6	6	5
	A	B	C	D	E

โนดที่มีน้ำหนักน้อยที่สุดคือ D และ E ทั้งสองโนดจะถูกเชื่อมให้เป็นโนดพ่อซึ่งมีน้ำหนัก 11 แล้วเรากำหนดให้ โนด D มีค่าเป็น 0 และโนด E มีค่าเป็น 1 รอบที่สอง B และ C จะถูกนำมาเชื่อมเป็นโนดพ่อใหม่อีก ซึ่งมีน้ำหนักเป็น 13 ซึ่งจะเป็นดังรูป



รูปที่ 3.1 ฮัฟแมนทรี เมื่อผ่านไป 2 รอบ

รอบต่อมา 2 โนด ที่มีน้ำหนักน้อยที่สุด คือโนดพ่อ ของ B/C กับ D/E จะถูกเชื่อมเป็นโนดพ่อ หลังจากนั้นจะเหลือเพียง 2 โนด เมื่อนำมารวมกันจะได้รากของทรีซึ่งมีลักษณะดังรูป 3.2



รูปที่ 3.2 แผนภาพแสดงฮัฟแมนทรี

การหารหัสของข้อมูลทำได้โดยการ เดินทางไปยังเส้นทางที่ข้อมูลตัวนั้นผ่าน ผ่านโนดใด ให้จำเอาไว้ รหัสของข้อมูลคือ ค่าของโนดที่จำไว้ตามลำดับ ดังนั้นรหัสของข้อมูลคือ

A = 0
B = 100
C = 101
D = 110
E = 111

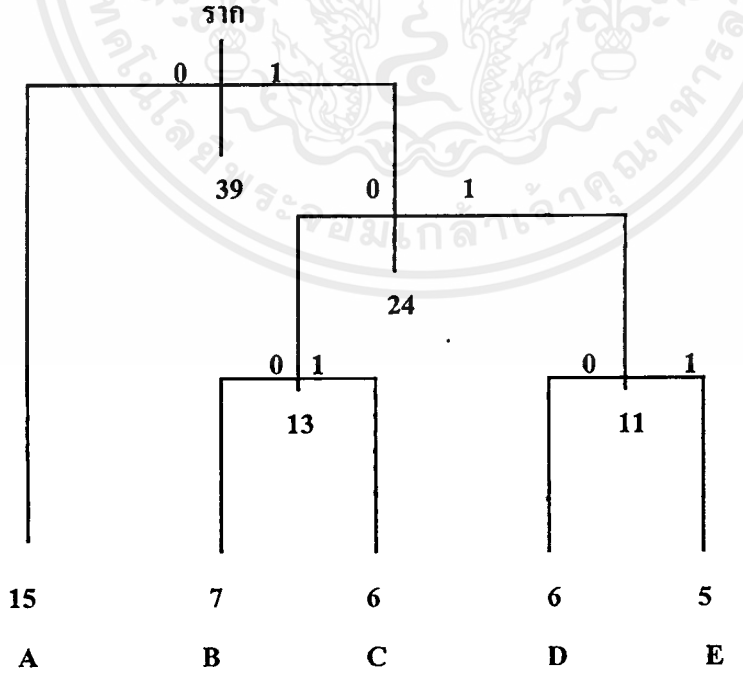
หลังจากได้รหัสของข้อมูลแล้วจึงอ่านข้อมูลอีกรอบเพื่อลดขนาดข้อมูลทั้งหมดโดยการแทนข้อมูลด้วยรหัสของข้อมูล ดังนั้นจะได้รหัสของข้อมูลเรียงกันเป็นลักษณะเป็นลำดับเช่นเดียวกับลำดับของข้อมูล ดังรูปที่ 3.3

"0 111 110 100 0 0 101..." รหัสของข้อมูลมีลักษณะเป็นบิท

"A E D B A A C....." ข้อมูลมีลักษณะเป็นไบท์

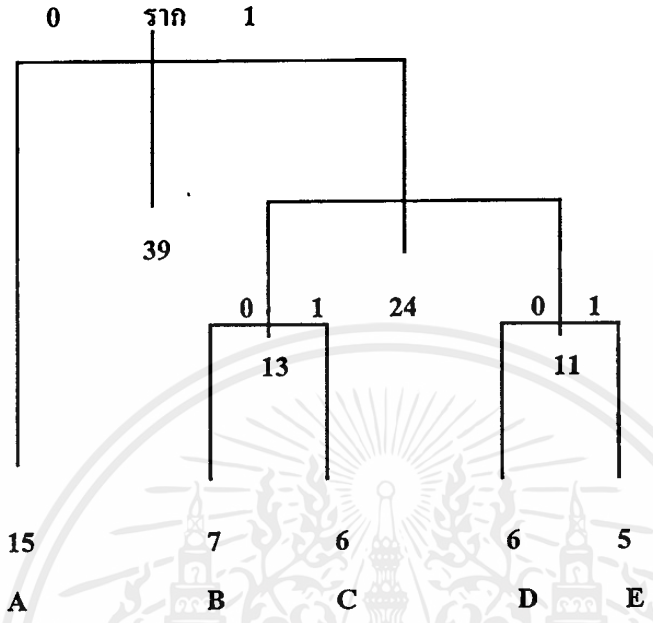
รูปที่ 3.3 ตัวอย่างการเข้ารหัสข้อมูล

สำหรับการขยายข้อมูลกลับทำได้โดย การสร้างทรีขึ้นมาก่อนโดยทรีนี้คือทรีอันเดียวกับที่ใช้ในการลดขนาดข้อมูล โดยจะอ่านรหัสเข้ามาทีละบิทและเริ่มเดินทางจากรากไปยังกิ่งที่มีค่าเท่ากับรหัส 1 บิทที่ป้อนเข้ามา และกระทำเช่นนี้ต่อไปจนกระทั่งไม่มี โนดย่อยลงไปอีกก็จะได้ข้อมูลคืนกลับมาหนึ่งตัว หลังจากนั้น เริ่มเดินทางจากรากใหม่ จนกระทั่งได้ข้อมูลกลับคืนมาจนครบหมด จากรหัสของข้อมูลข้างต้น บิทตัวแรกคือ 0 ดังนั้นสามารถถอดรหัสออกมาได้เป็น A ดังรูป 3.4



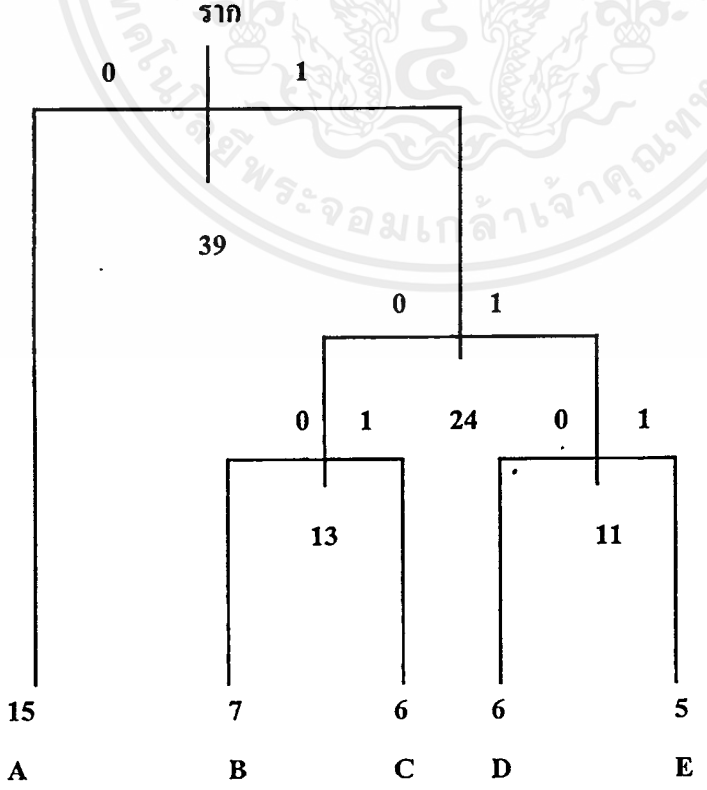
รูปที่ 3.4 แสดงการถอดรหัสเมื่อได้รับรหัส 0

จากนั้นอ่านรหัสข้อมูลบิทที่ 2 คือ1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 3.5



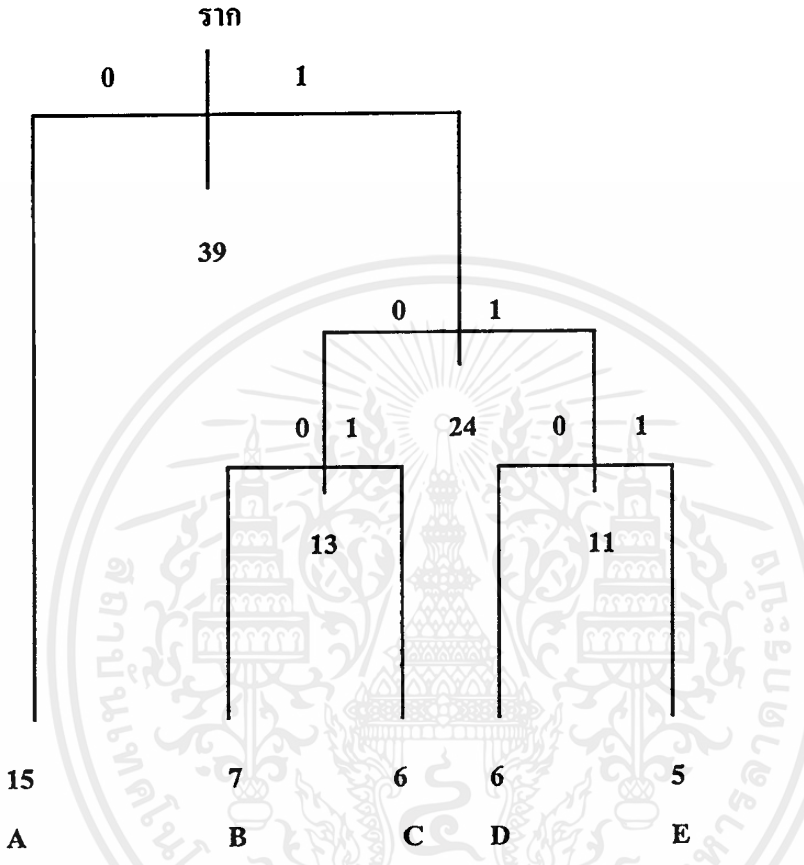
รูปที่ 3.5 แสดงการถอดรหัสเมื่อได้รับรหัส 1

จากนั้นอ่านรหัสข้อมูลบิทที่ 3 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 3.6



รูปที่ 3.6 แสดงการถอดรหัสเมื่อได้รับรหัส 11

จากนั้นอ่านข้อมูลบิทที่ 1 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 3.7



รูปที่ 3.7 แสดงการถอดรหัสเมื่อได้รับรหัส 111

ก็จะสามารถถอดรหัสได้เป็น A และจะกระทำเช่นนี้ต่อไปจนกระทั่งไม่มีข้อมูลเหลือ แล้วจะได้ข้อมูลกลับมาเหมือนเดิมทุกประการ

บทที่ 4
การลดข้อมูลภาพเคลื่อนไหว

4.1 พื้นฐานของการลดข้อมูลภาพเคลื่อนไหว

ในการงานที่เกี่ยวกับการสร้างภาพเคลื่อนไหวหรือระบบการส่งสัญญาณภาพโทรทัศน์โดยทั่วไปนั้นจะเกิดจากการสร้างภาพสิ่งที่มีความต่อเนื่องกันหลายๆภาพแล้วทำการส่งภาพออกไปให้ปรากฏที่หน้าจอภาพด้วยอัตราเร็วที่เหมาะสม(อย่างน้อยประมาณ 15 ภาพต่อวินาที[frames/sec]) ก็จะทำให้สามารถมองเห็นเป็นภาพเคลื่อนไหวต่อเนื่องกันไปได้โดยในที่นี้จะยกตัวอย่างการแพร่กระจายสัญญาณโทรทัศน์ในระบบ PAL ซึ่งจะใช้อัตราเร็วในการส่งสัญญาณภาพประมาณ 25 ภาพต่อวินาที โดยในจำนวนของภาพที่ทำการส่งไปนี้จะมีบางส่วนของภาพที่ไม่มีการเปลี่ยนแปลง ซึ่งจากทฤษฎีและหลักการของการ Interpolation นั้นทางด้านเครื่องรับเราสามารถที่จะอาศัยข้อมูลจากเฟรมที่ผ่านมาและเฟรมในอนาคตเพื่อทำการสร้างเฟรมภาพขึ้นมาใหม่ระหว่าง 2เฟรมนี้ จากหลักการนี้ทำให้ทางด้านส่งไม่จำเป็นต้องส่งภาพทั้งหมดคือ 25 ภาพต่อวินาที แต่จะทำการส่งมาเพียงแค่บางส่วนเท่านั้น และทางด้านรับก็จะมาทำการสร้างภาพกลับคืนเองจากหลักการดังกล่าวทั้งหมดข้างต้น มีขั้นตอนที่สำคัญ เช่นการทำ Motion Estimation โดยจะเป็นการคำนวณหา Motion Vector และ Motion-Compensated ซึ่งเป็นส่วนที่สำคัญที่จะกล่าวถึงเป็นขั้น ๆ ดังนี้

ขั้นตอนการลดข้อมูลภาพเคลื่อนไหว (motion image compression)

1. การทำโมชั่นเอสติเมชั่น (motion estimation)

    1.1 การหาโมชั่นเวคเตอร์ (motion vector) เป็นการหาตำแหน่งที่เปลี่ยนไปของภาพ

    2.2 การทำโมชั่นคอมเพนเสท (motion compensate) เป็นการลบภาพระหว่างเฟรมปัจจุบันกับเฟรมถัดไป

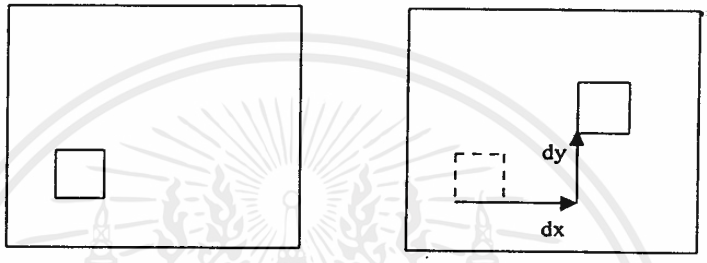
2. การทำการลดข้อมูลแบบรันเลนจ์

ขั้นตอนการขยายภาพเคลื่อนไหวกลับคืน (Motion Image Uncompression)

1. การดีโค๊ท (decode) ข้อมูลที่ได้ลดแบบรันเลนจ์
2. การทำอิมเมจรีสตอเรชั่นซึ่งก็คือการนำข้อมูลของโมชั่นเวคเตอร์และโมชั่นคอมเพนเสทมาบวกกันซึ่งจะได้กล่าวต่อไปในภายหลัง

4.2 โมชั่นเอสติเมชั่น (motion estimation)

ในกรณีที่เรามีเฟรมของภาพเคลื่อนไหวที่ต่อเนื่องกัน 2 เฟรม ซึ่งในที่นี้สมมุติให้เป็นเฟรมที่ fn (เฟรมปัจจุบัน) และ fn-1(เฟรมก่อน)ตามลำดับ เราสามารถที่จะคำนวณหาทิศทางการเคลื่อนที่ของวัตถุในภาพระหว่างทั้ง 2 เฟรมนี้ได้ โดยผลลัพธ์ที่ได้จะอยู่ในรูปของเวคเตอร์ที่แสดงถึงทิศทางการเคลื่อนที่ของวัตถุในภาพซึ่งจะเรียกกันว่า โมชั่นเวคเตอร์

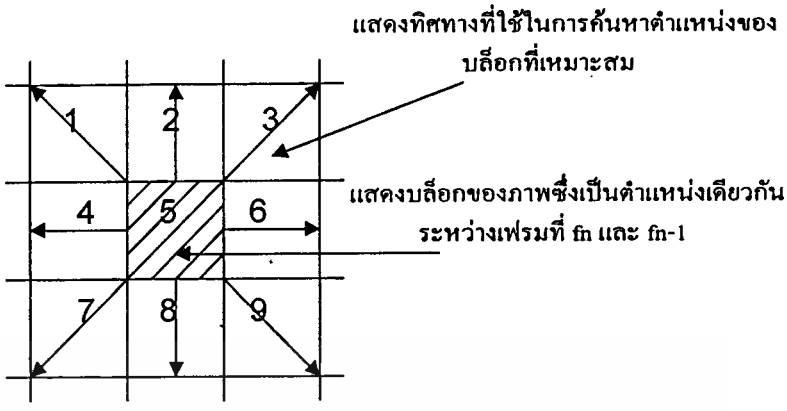


a)เฟรม fn-1(ก่อน) b)เฟรม fn(ปัจจุบัน)

รูปที่ 4.1 แสดงการเคลื่อนที่ของวัตถุในภาพด้วยระยะ(เวคเตอร์)ตามแนวแกน X และแนวแกน Y

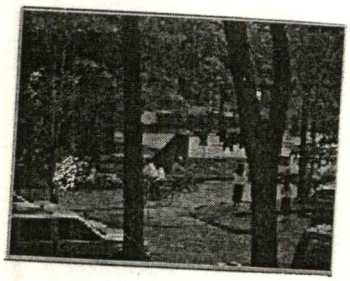
วิธีที่ใช้ในการคำนวณหาโมชั่นเวคเตอร์นี้ ก็มีด้วยกันหลายวิธี โดยในรายงานฉบับนี้จะเลือกใช้วิธีที่เรียกว่า Block matching methods โดยจะทำการแบ่งเฟรมภาพออกเป็นบล็อกเล็กๆ จากนั้นจึงนำไปทำการเปรียบเทียบตามทิศทางต่างๆ เพื่อค้นหาตำแหน่งที่เหมาะสม (match) ที่สุด โดยสมการที่ใช้ในการคำนวณสามารถแสดงได้ดังสมการที่ 1 ซึ่งจะแสดงในรูปของค่า ERROR นั่นก็คือ บล็อกที่ให้ค่าของ ERROR น้อยที่สุด สำหรับการ search แต่ละครั้ง ก็คือ บล็อกที่เหมาะสมกันที่สุด โดยถ้าเกิดค่าของ ERROR มีค่าเป็น 0 จะแสดงว่าบล็อกทั้งสองนั้นเป็นค่าๆเดียวกันหรือเป็นบล็อกที่ตำแหน่งเดียวกันนั่นเอง ซึ่งรายละเอียดของขั้นตอนการทำงานต่างๆ มีดังนี้

$$Error = \sum_i \sum_j \left| fn(x_i, y_j) - f_{n-1}(x_i + dx, y_j + dy) \right|$$

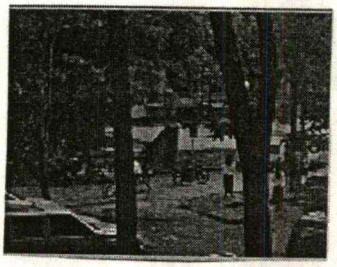


รูปที่ 4.2 แสดงตำแหน่งของบล็อกและทิศทางที่ใช้ในการค้นหาตำแหน่งของบล็อกที่ Match

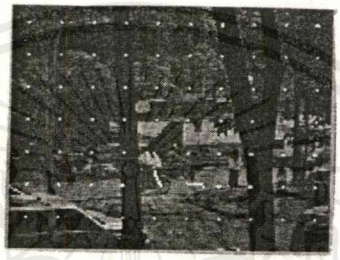
ในขั้นแรกจะทำการแบ่งเฟรมของภาพทั้งสองออกเป็นบล็อกเล็กๆ ในที่นี้กำหนดให้มีขนาก 16 x 16 พิกเซลจากนั้นจึงนำแต่ละบล็อกเหล่านี้ไปทำการเปรียบเทียบตามทิศทางต่างๆที่กำหนดไว้ซึ่งทิศทางที่กล่าวถึงนี้ก็ คือ ที่บล็อกตำแหน่งเดียวกันของเฟรมภาพปัจจุบัน (fn) และเฟรมของภาพก่อน(fn-1) บล็อกของเฟรมภาพก่อนจะถูกกำหนดให้คงที่โดยที่บล็อกของเฟรมภาพปัจจุบันจะถูกกำหนดให้เป็นบล็อกที่ใช้ค้นหา (search) หรือเป็นบล็อกที่ถูกเลื่อน ไปในทิศทางต่างๆ นั่นเอง ซึ่งทิศทางที่ใช้ในการ ค้นหาทั้งหมดจะกำหนดให้มีทิศทางละ 8 พิกเซล ดังแสดงในรูปที่ 2 นั่นคือ ตำแหน่งแรกของการ search บล็อกของเฟรมภาพปัจจุบันจะถูกเลื่อนไปทางซ้าย 8 พิกเซล และเลื่อนขึ้นด้านบน 8 พิกเซล (เมื่อเทียบกับตำแหน่งของเฟรมภพก่อน) นั่นก็คือจะไปเริ่มต้นค้นหาที่บล็อกตำแหน่งที่ 1 (เป็นตำแหน่งของเฟรมภาพก่อน) ดังแสดงในรูปที่ 2 จากนั้นจึงทำการเลื่อนบล็อกของเฟรมภาพปัจจุบันไปตามแนวแกน X ทีละพิกเซลจนครบ 16 ตำแหน่ง (ซ้าย 8 ตำแหน่ง + ขวา 8 ตำแหน่ง) หลังจากนั้นก็จะทำการเลื่อนบล็อกลงมาตามแนวแกน Y 1 ตำแหน่ง แล้วจึงทำการเลื่อนไปตามแนวแกน X อีกครั้งจนครบ 16 ตำแหน่ง ทำตามขั้นตอนนี้ไปเรื่อยๆ จนค่าที่เลื่อนตามแนวแกน Y ครบทั้ง 16 ตำแหน่ง ซึ่งในแต่ละครั้งของบล็อกที่เลื่อนไปนี้ก็จะทำการคำนวณหาค่า ERROR ตามสมการที่ 1 โดยจากสมการจะเห็นว่าเป็นการนำมาทำการลบกันพิกเซลต่อพิกเซลภายในบล็อกขนาด 16 x 16 แล้วนำค่าขนาดของผลต่างที่แต่ละพิกเซลทั้งหมดภายในบล็อกมารวมกันก็จะได้เป็นค่าของ ERROR 1ค่า จากนั้นนำค่าของ ERROR ที่ตำแหน่ง X,Y ต่างๆ ที่เลื่อนไปมาทำการเปรียบเทียบกันว่าที่ตำแหน่ง X,Y ใดให้ค่าของ ERROR น้อยที่สุดซึ่งก็คือตำแหน่งที่บล็อกMatch ที่สุดนั่นเองโดยค่าของตำแหน่ง X และ Y นี้ก็คือ เวคเตอร์ที่แสดงถึงทิศทางการเคลื่อนที่ของวัตถุในภาพนั่นเอง ซึ่งเราจะพบว่าขั้นตอนที่ได้กล่าวมาทั้งหมดนี้เป็นการเปรียบเทียบเพียงแค่บล็อกเดียวเท่านั้น โดยในความเป็นจริงแล้วเราจะต้องนำบล็อกที่มีอยู่ทั้งหมดในภาพของเฟรม fn ที่แบ่งไว้มาทำการเปรียบเทียบให้ครบทุกบล็อก



a) previous frame (fn-1)



b) current frame (fn)



c) show motion vector

รูปที่ 4.3 แสดง motion vector ระหว่างเฟรมภาพของ a) frame fn และ b) frame fn-1

จากรูปเราจะพบว่าการแสดงทิศทางของ Vector นั้น เราจะใช้จุดกึ่งกลางของแต่ละบล็อกที่ได้แบ่งไว้ (16x16)ในที่นี้คือตำแหน่งที่ 8.8 (ที่พล็อตเป็นจุดสีขาว) ในแต่ละบล็อกเป็นจุดเริ่มต้น นั่นก็คือเราสามารถที่จะแสดงการเคลื่อนที่ของเวคเตอร์ได้ทุกทิศทางจากรูปจุดสีขาวที่รูปแสดงได้จะหมายถึง ภายในบล็อกนั้นไม่มีการเปลี่ยนแปลงหรือไม่เกิดการเคลื่อนที่ของวัตถุภายในบล็อกสำหรับในส่วนที่เป็นเส้นตรงตามแนวนอนที่ลากไปทางขวาจะหมายถึงมีการเคลื่อนที่ของวัตถุภายในบล็อกไปตามทิศทางของแกน +X (แต่ถ้าลากไปทางซ้ายจะหมายถึงมีการเคลื่อนที่ของวัตถุไปตามทิศทางของแก -X) โดยค่าตามแนวแกน Y จะไม่มีการเคลื่อนที่ (แกน Y ก็คือเส้นตรงที่ลากลงมาตั้งฉากกับจุดสีขาวซึ่งนั่นก็คือแกน Y นั่นเอง)

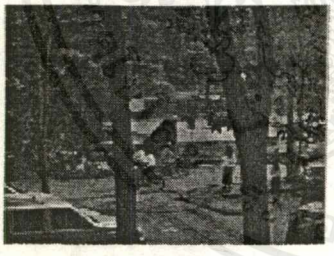
4.3 โมชั่นคอมเพนเสท (motion-compensated)

เป็นส่วนที่ใช้ในการแสดงให้เห็นถึงความแตกต่างระหว่างเฟรมภาพ 2 เฟรมที่มีความแตกต่างกัน และนอกจากนั้นในส่วนของโมชั่นคอมเพนเสทนี้ ยังสามารถที่จะใช้ในการสร้างภาพเฟรมปัจจุบันกับคืนมาโดยอาศัยเพียงข้อมูลของโมชั่นเวคเตอร์ ร่วมกับข้อมูลภาพของเฟรมก่อน ก็สามารถที่จะสร้างภาพเฟรมปัจจุบันกลับคืนมาได้ สำหรับในส่วนของการทำโมชั่นคอมเพนเสท นั้น

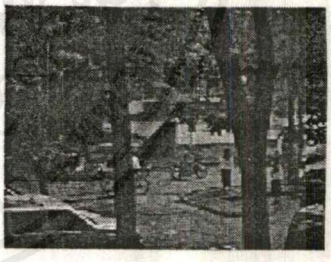
จะเกิดจากการนำภาพของเฟรมทั้งสองมาหาผลตางโดจอาศัยข้อมูลของโมชั่นเวคเตอร์ จากในส่วนที่แล้วเราสามารถคำนวณหา โมชั่นเวคเตอร์ได้ ซึ่งข้อมูลในส่วนนี้ก็คือ ค่าของตำแหน่งที่เปลี่ยนแปลงไปจากเฟรมก่อน โดยสามารถทำการคำนวณหาโมชั่นคอมเพนเสท ได้ดังนี้ ขั้นแรกจะทำการแบ่งภาพของเฟรมทั้งออกเป็นบล็อกเล็กๆ โดยมีขนาดเท่ากับ 16x16 จากนั้นนำแต่ละบล็อกมาลบกับแบบพิกเซลต่อพิกเซลแต่มีเงื่อนไขว่าบล็อกของเฟรมก่อนจะต้องมีการเลื่อนไปตามตำแหน่งของโมชั่นเวคเตอร์ (ค่า vector ที่ทำให้บล็อกนั้นๆ match กันมากที่สุด) ซึ่งสามารถทำได้โดยการนำค่าของเวคเตอร์ มาบวกเข้าไปจากตำแหน่งเริ่มต้นของบล็อกของเฟรมก่อนเมื่อได้ตำแหน่งที่ต้องการแล้วจึงนำบล็อกจของเฟรมปัจจุบันมาทำการลบกันทำเช่นนี้ไปเรื่อยๆ จนครบทุกบล็อกของเฟรมภาพปัจจุบัน ซึ่งสามารถแสดงสมการที่ใช้ได้ดังนี้

บล็อกที่ I ของเฟรม fn - [บล็อกที่ I ของเฟรม fn-1+ ค่า Vector] + ค่าระดับสีเทา = Motion-Compensated

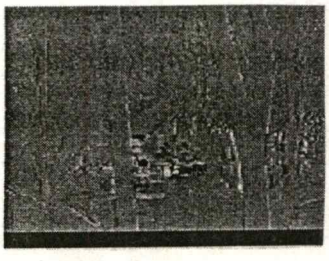
โดยค่าของระดับสีเทาที่บวกเพิ่มเข้าไปนั้นเพื่อให้สามารถมองเห็นผลต่างของภาพทั้งสองได้ชัดเจนยิ่งขึ้น ซึ่งสามารถแสดงผลของการทำ Motion-Compensated ได้ดังนี้



a)previous frame(fn-1)



b)current frame(fn)



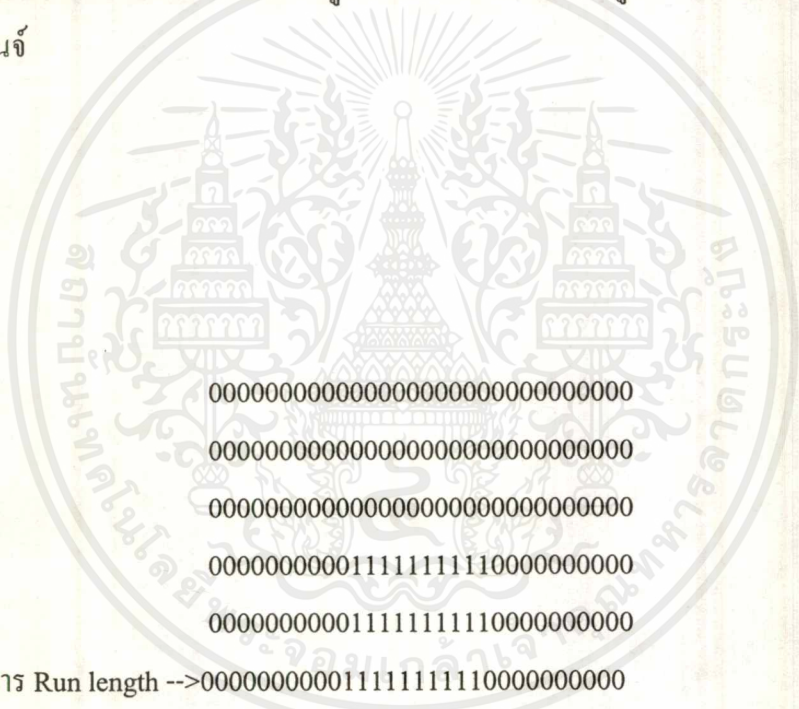
c)Motion-Compensate(MC)

รูปที่ 4.4 แสดง Motion-Compensated ระหว่างเฟรมภาพของ a) frame fn และ b) frame fn-1

ส่วนสาเหตุที่ต้องนำเฟรมภาพมาลบกันนี้เพื่อที่จะให้เกิดข้อมูลซ้ำกันให้มากที่สุด เพื่อที่จะนำไปลดข้อมูลแบบ Run length ต่อไป

4.4 ลดข้อมูลโดยใช้วิธีรันเลนจ์

การลดข้อมูลโดยใช้วิธีรันเลนจ์ เป็นวิธีการลดข้อมูลที่ไม่ซับซ้อน โดยจะทำการตรวจหาสข้อมูลที่ซ้ำกันและแทนที่ด้วยค่าๆ หนึ่ง กับ คำ count ที่บอกว่าซ้ำกันที่ตัว วิธีนี้ถ้าหากว่าใช้กับภาพซึ่งเป็นข้อมูล 2 มิติ จะต้องทำการถอดข้อมูลให้เรียงกันซะก่อน ดังรูปที่ 4.5 แสดงการลดข้อมูลโดยใช้วิธี รันเลนจ์



000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000011111111110000000000
000000000011111111110000000000

ข้อมูลที่จะการ Run length -->000000000011111111110000000000

000000000000000000000000000000

ข้อมูลที่จะทำการ Run length 000000000011111111110000000000

สามารถลดข้อมูลได้ดังนี้ 0000000000 ถูกแทนที่ด้วย 10,0 และ 1111111111 ถูกแทนที่ด้วย 10,1 ส่วน 0000000000 ถูกแทนที่ด้วย 10,0

รูปที่ 4.5 แสดงถึงการลดข้อมูลโดยใช้วิธี Run length

การที่จะปรับปรุงวิธีพื้นฐานของ Run length ให้ดียิ่งขึ้นก็โดยการเปลี่ยนให้ code แต่ละตัวมีความยาวต่างกันไป ซึ่งที่ได้โดยการหาความน่าจะเป็นของข้อมูลแต่ละตัว ซึ่งเป็นการผสมกันระหว่าง Run length Encoding กับ Statiscal Encoding

การทำ Run length Encoding นี้ไม่เหมาะสำหรับภาพที่มีโทนสีต่อเนื่อง เพราะจะลดข้อมูลได้ไม่มากนัก

4.5 การสร้างภาพกลับคืน (Image restoration)

ในการสร้างภาพกลับคืนมานั้นเราก็เพียงแต่ใช้วิธีการตรงกันข้ามกับการลดข้อมูลภาพ ซึ่งมีขั้นตอนดังนี้

1.การ Decode ข้อมูลที่ทำการ Run length Encoding

2.การบวกเฟรมภาพโดยอาศัยข้อมูลของ Motion Vector และ Motion Compensate ซึ่งมีขั้นตอนคร่าว ๆ ดังนี้

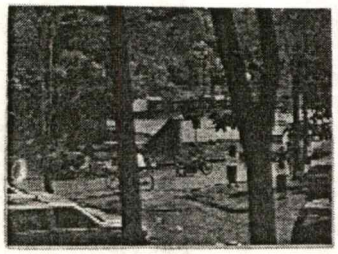
ให้เฟรมภาพปัจจุบันเป็น fn-1

ให้เฟรมภาพที่ลบกันคือ fn

เราต้องการเพียงข้อมูลของโมชั่นคอมเพนเสท โมชั่นเวคเตอร์ และ ข้อมูลภาพในเฟรมก่อนเพื่อใช้ในการอ้างอิง สำหรับขั้นตอนที่ใช้ในการสร้างภาพกลับคืนมานั้น จะมีขั้นตอนเหมือนกันกับการทำ โมชั่นคอมเพนเสท โดยเราจะใช้ Inverse process ของโมชั่นคอมเพนเสท ก็จะทำให้ได้ข้อมูลภาพกลับคืนมา ซึ่ง สามารถแสดงได้ดังสมการต่อไปนี้

บล็อกที่1 ของเฟรม fn = [(Motion-Compensated) - ค่าระดับสีเทา] + [บล็อกที่1 ของเฟรม fn-1 + ค่า Vector]

ซึ่งสามารถแสดงผลลัพธ์ที่ได้จากการทดลองดังนี้



รูปที่ 4.6 แสดงภาพของเฟรมปัจจุบันที่ส่งคืนมา

4.6 การใช้การ์ด TARGA+ มาใช้ในการเก็บข้อมูลภาพเคลื่อนไหว

การ์ด TARGA+ เป็นผลิตภัณฑ์ของบริษัท Truevision โดยที่การ์ดนี้มีความสามารถในการให้รายละเอียดของภาพที่สูง และสามารถนำภาพเหล่านี้ไปประมวลผลต่อไปได้

4.6.1 โครงสร้างของการ์ด TARGA+

การ์ด TARGA+ มีส่วนประกอบที่สำคัญดังนี้

VRAM(Video Ram) - memory array ที่ไว้ในการเก็บข้อมูลภาพ

Mixer - เป็นตัวรวมสัญญาณ

CTL - เป็นส่วนควบคุม video timing,display และส่วนประกอบอื่น ๆ ของบอร์ด

Decoder - แปลงสัญญาณ S-Video เป็น RGB

Encoder - แปลงสัญญาณ RGB เป็น สัญญาณ S-Video

ADC - เป็นตัวแปลงสัญญาณ analog เป็น digital

4.6.2 โหมดการแสดงผลของการ์ด TARGA+

การ์ดTARGA+ จึงต้องสามารถเนื่องจากจอภาพที่ใช้แสดงผลมีหลายชนิด ดังนั้นในการแสดงผลเหล่านี้ด้วยควบคุมจอแสดงผลที่ต่างกันได้ทุกชนิด ดังนั้นจึงต้องมีการกำหนดโหมดการทำงานของการ์ด TARGA+ ให้ตรงกับชนิดของจอที่แสดงผล

โหมดการทำงานที่ต่างกันของการ์ด TARGA+ จะหมายถึงการกำหนดค่าความแตกต่างของจอชนิดต่าง ๆ เช่น จำนวนจุดในแนวนอน (Horizontal Resolution) , จำนวนจุดภาพในแนวตั้ง (vertical resolution) , อัตราส่วนมุมมองของจอภาพ (aspect ratio) ,จำนวนบิทต่อจุดต่อภาพ (pixel depth) , รูปแบบการ scan ภาพ ซึ่งค่าของโหมดการทำงานต่างๆ ได้แสดงดังตารางที่ 4.1

Mode	Horizontal Resolution	Vertical Resolution	Aspect Ratio	Pixel Depth	Display
1	512	400	1.071	16 b/p	Interlaced
2	512	400	7.042	16 b/p	non-interlaced
3	512	400	1.071	32 b/p	Interlaced
4	512	400	1.042	32 b/p	non-interlaced
5	512	476	1.269	16 b/p	Interlaced
6	512	476	1.240	16 b/p	non-interlaced
7	512	476	1.269	32 b/p	Interlaced
8	512	476	1.240	32 b/p	non-interlaced
9	512	480	1.266	16 b/p	Interlaced
10	512	486	1.266	16 b/p	non-interlaced
11	512	486	1.266	32 b/p	Interlaced
12	512	486	1.266	32 b/p	non-interlaced

ตารางที่ 4.1 แสดงถึงโหมดการทำงานต่างๆ ของ TARGA+

4.6.3 พื้นฐานการเขียนโปรแกรมควบคุมการ์ด TARGA+

ในการเขียนโปรแกรมควบคุมการ์ด TARGA+ สำหรับโครงงานนี้ มีรีจิสเตอร์ที่สำคัญคือ รีจิสเตอร์ mode 2 ซึ่งใช้สำหรับการจับภาพ ซึ่งจะขอกล่าวถึงรายละเอียดของรีจิสเตอร์ตัวนี้สักเล็กน้อย

รีจิสเตอร์โหมด 2 มีรายละเอียดของแต่ละบิทดังนี้

7	6	5	4	3	2	1	0
genlock	capture	dispMode		zoom		reserve	

บิต 0 และ 1 สงวนไว้

ใน advance operating mode บิทเหล่านี้สงวนไว้ และควรถูกทำให้เป็น 0 ทุกครั้ง

zoom (bits 2-3)

zoom field นี้เป็นตัวกำหนดค่าตัวประกอบการขยาย (zoom factor) โดยกำลังการขยายจะมีค่าเป็นกำลังสองของค่าที่ระบุใน zoom ดังตาราง

zoom	zoom factor
0	1x
1	2x
2	3x
3	4x

ตารางที่ 4.2 แสดงถึงความสัมพันธ์ระหว่างค่า zoom และ ค่า zoom factor

ข้อควรจำ

เมื่อออกจากโปรแกรมแล้ว จะต้อง set ค่าของ zoom factor ให้มีค่าเป็น 0

dispMode (bits 4-5)

ค่าของ dispMode จะทำให้สามารถแสดงผลได้หลายแบบ ดังตาราง

dispMode	Display Mode
0	แสดงภาพจาก memory
1	live mode
2	overlay mode
3	live mode

ตารางที่ 4.3 แสดงถึง Display Mode ของ TARGA+

capture (bit 6)

capture bit เป็นตัวกำหนดว่าจะมีการจับภาพขึ้นหรือไม่ เมื่อ ให้ค่า 1 มีการจับภาพ , และค่า 0 จะไม่มีการจับภาพและก่อนที่จะมีการจับภาพ genlock bit ต้องถูก set ซะก่อน และ dispMode จะต้องอยู่โหมด 1,2, หรือ 3 เท่านั้น

genlock (bit 7)

genlock bit ถ้าให้ค่าเป็น 0 TARGA+ จะอยู่ใน master mode และจะจัดการเรื่อง timing ด้วยตัวเอง แต่ถ้าหาก genlock มีค่าเป็น 1 TARGA+ จะพยายาม synchronize กับสัญญาณนาฬิกาที่มาจากสัญญาณวีดีโอ แต่ถ้าหากไม่มีสัญญาณมีทาง อินพุท TARGA+ จะเปลี่ยนกลับไปเป็น master mode โดยอัตโนมัติ

ส่วนการเขียนโปรแกรมในการรับภาพเข้ามานั้นเรา ส่งค่าออกไปให้กับรีจิสเตอร์ mode2 ค่าที่ส่งออกไปนั้นคือ 0x80 ซึ่งจะไป set ให้บิท capture มีค่าเป็น 1

```
outportb ( TARGACARD+MODE2,0x80);
```

โดยค่า TARGACARD นั้นมีค่าเท่ากับ 0x220

ส่วนค่า MODE2 นั้นมีค่าเท่ากับ 0xC01

และภาพที่ถูกถ่ายจะเก็บไว้ใน memory แอดเดรสที่ D000H เมื่อนำค่าใน memory มา process เสร็จเรียบร้อยแล้ว เราก็จะทำการเรียกใช้ก็จะทำการ capture ต่อไป ดังต่อไปนี้

```
outportb ( TARGACARD+MODE2,0xD0);
```

โดยส่งค่า 0xD0 ให้กับ รีจิสเตอร์โหมด 2 ซึ่งค่านี้จะไปควบคุมบิทต่าง ๆ ดังนี้

genlock = 1 ซึ่งจะทำให้ TARGA+ พยายาม synchronize กับ timing ของสัญญาณที่เข้ามา

capture = 1 สั่งให้ TARGA+ จับภาพเอาไว้

dispMode = 0 1 set ให้ card อยู่ใน mode 1

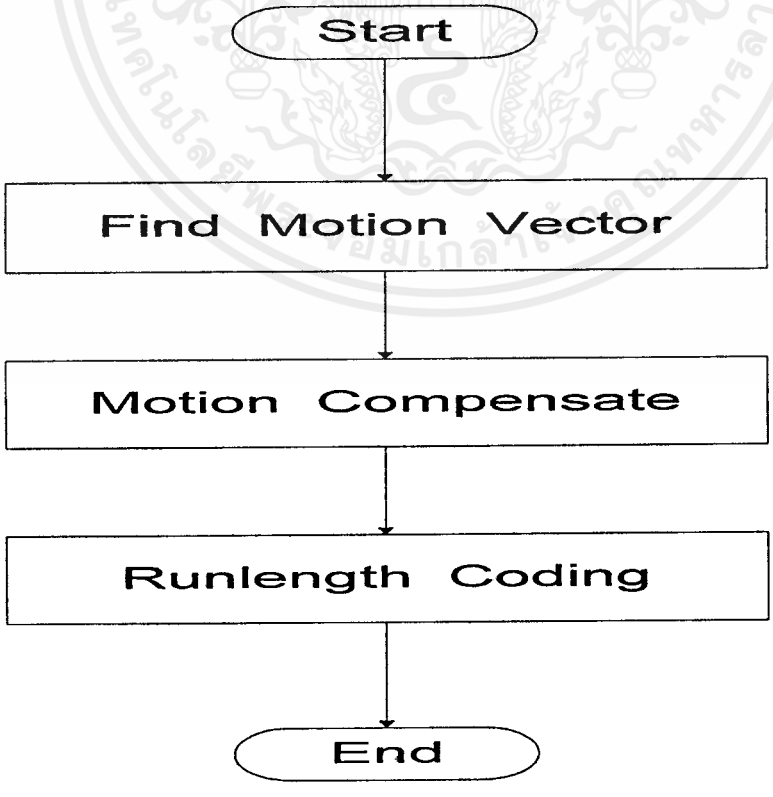
บทที่ 5
อัลกอริทึมของการลดสัญญาณภาพเคลื่อนไหว

ในการลดข้อมูลภาพเคลื่อนไหวนี้ ก่อนอื่นเราจะต้องรู้ถึงหลักการคร่าวๆ ของการลดสัญญาณภาพนิ่งก่อนดังที่กล่าวไว้ในบทก่อนๆ แล้ว จะทำให้เราสามารถเข้าใจของการลดสัญญาณภาพเคลื่อนไหวได้ดีมากขึ้น ขั้นตอนที่สำคัญ เช่น การนำรูปมาแปลงจากโดเมนข้อมูลไปเป็นโดเมนความถี่ ใช้วิธี DCT , การทำฮัฟแมนโคดดิ้ง , การทำรันเลนจ์โคดดิ้ง เป็นต้น ส่วนการนำข้อมูลที่ compress กลับมาใช้ใหม่นั้นมีวิธีตรงข้ามกันกับวิธีการข้างต้น เช่น การทำ IDCT , การทำ ฮัฟแมนดีโคดดิ้ง

ดังนั้นในบทนี้จึงได้พูดถึงเรื่องภาพเคลื่อนไหวแต่เพียงอย่างเดียวเท่านั้น

5.1 อัลกอริทึมของการลดสัญญาณของภาพเคลื่อนไหว

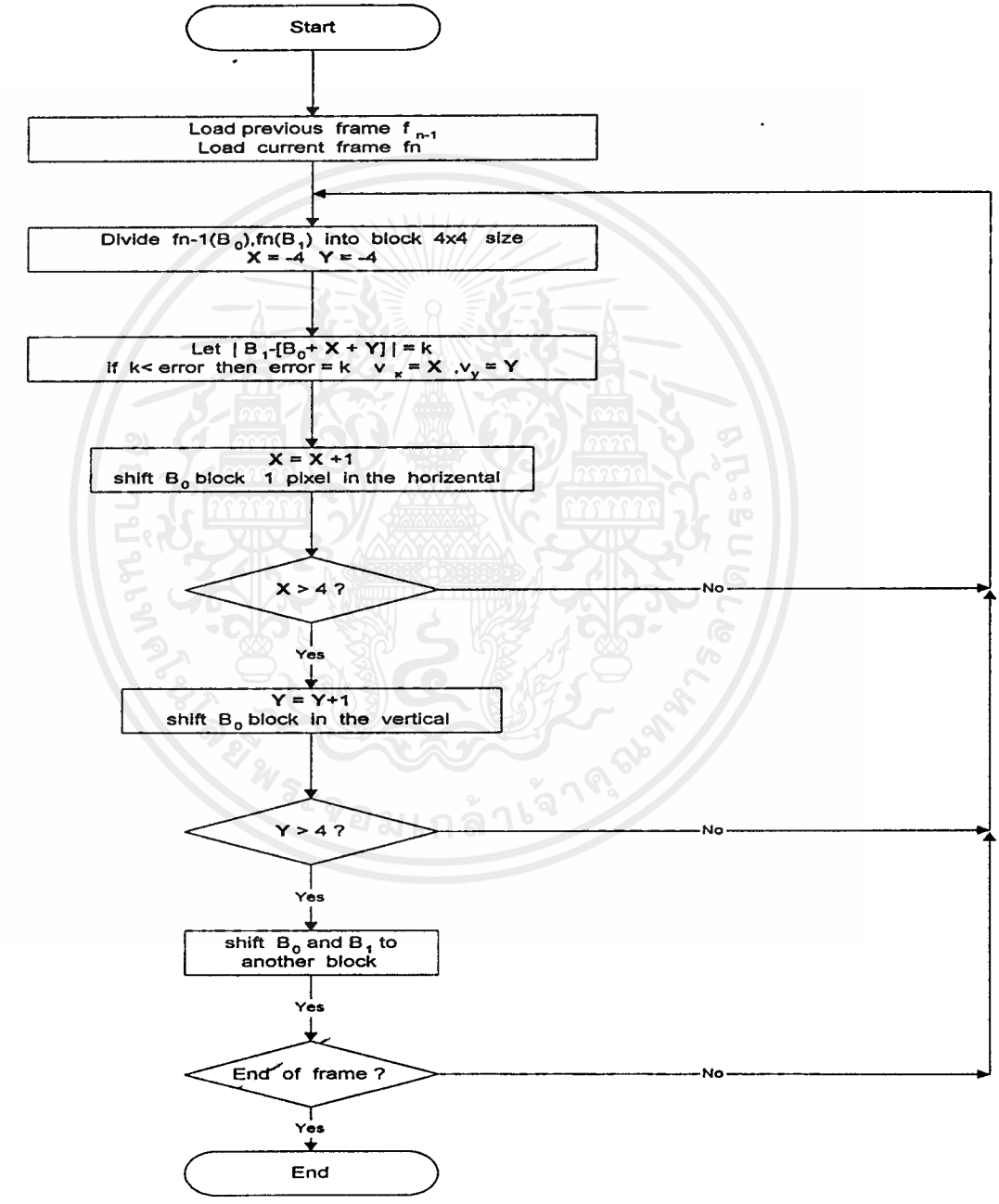
ในอัลกอริทึมของการลดข้อมูลของภาพเคลื่อนไหวนั้น สามารถแบ่งขั้นตอนได้คร่าว ๆ ดังนี้



รูปที่ 5.1 แสดงถึงขั้นตอนการลดข้อมูลของภาพเคลื่อนไหว

5.2 การหาโมชั่นเวคเตอร์

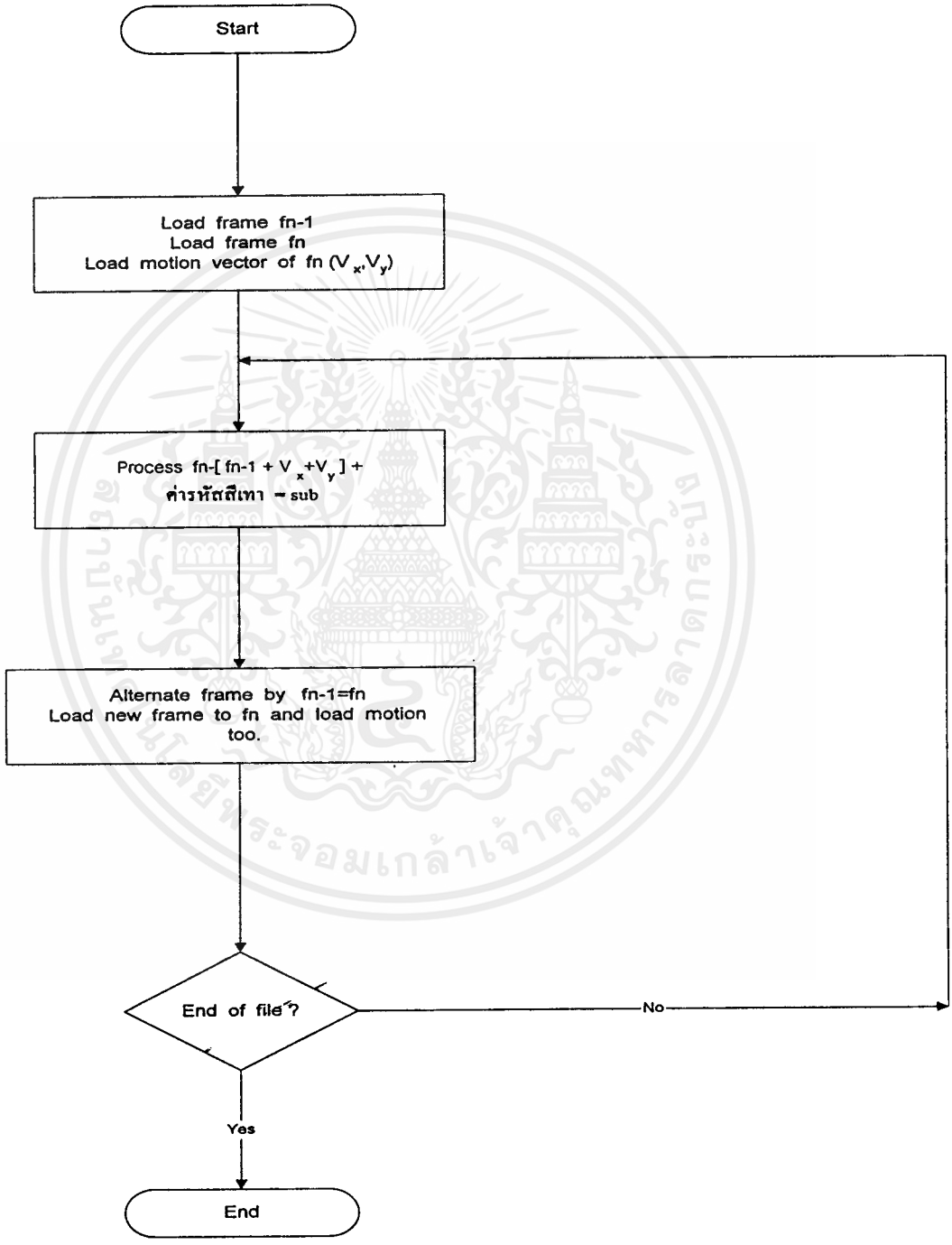
การหาโมชั่นเวคเตอร์ มี ขั้นตอนดังนี้



รูปที่ 5.2 แสดงถึงขั้นตอนการหาโมชั่นเวคเตอร์ของภาพเคลื่อนไหว

5.3 การหาโมชั่นคอมเพนเสท

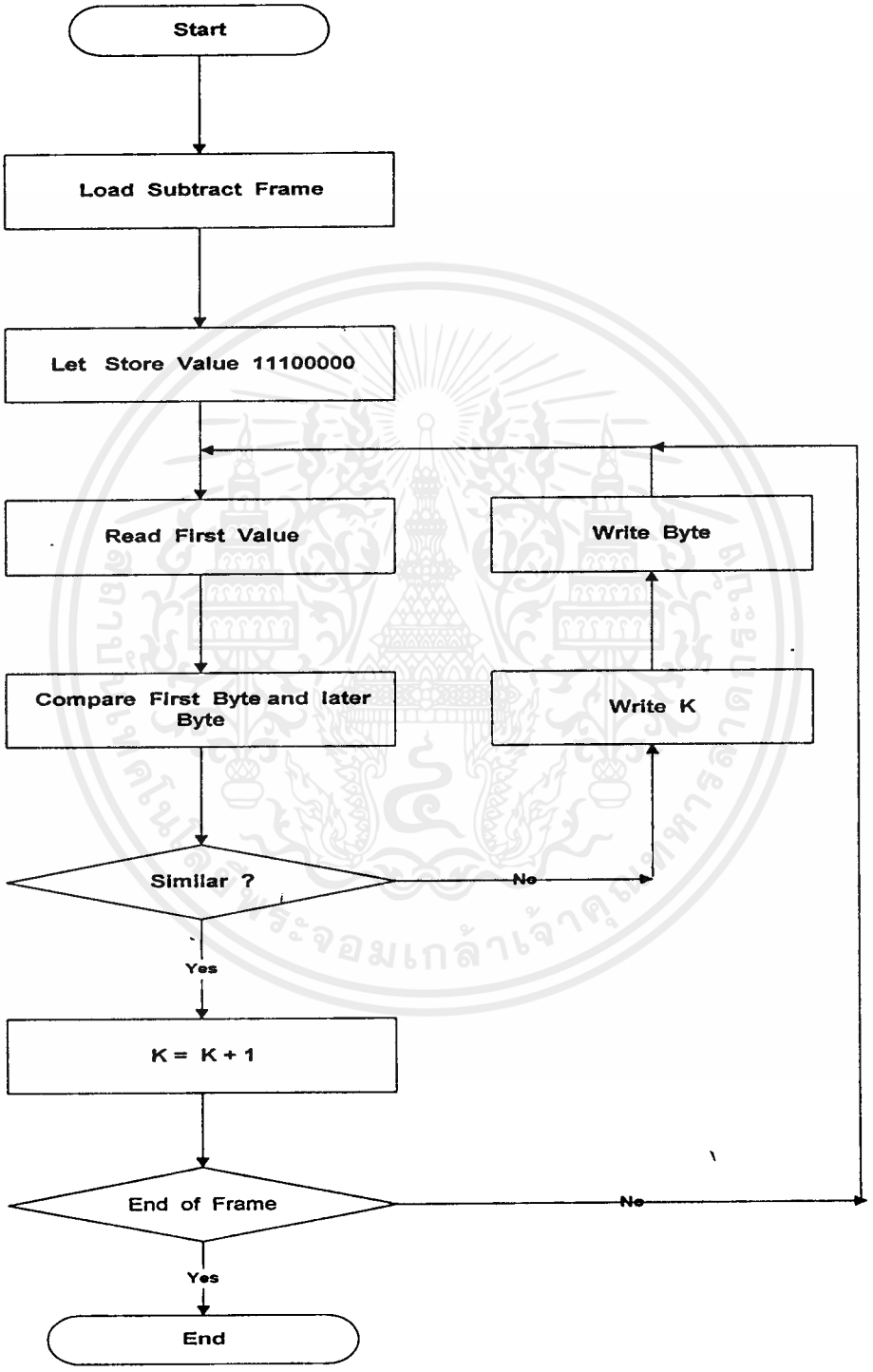
อัลกอริทึมของการหาโมชั่นคอมเพนเสทเป็นดังนี้



รูปที่ 5.3 แสดงถึงขั้นตอนของการหาโมชั่นคอมเพนเสท

5.4 การลดข้อมูลแบบรันเลนจ์โคดดิ้ง

อัลกอริทึมของการทำรันเลนจ์โคดดิ้ง มีขั้นตอนดังนี้



รูปที่ 5.4 แสดงถึงขั้นตอนของการทำรันเลนจ์โคดดิ้ง

บทที่ 6

การทดลองและผลการทดลอง

การทดลองนี้ได้จัดทำขึ้นเพื่อแสดงให้เห็นว่าโครงงานนี้สามารถที่จะลดข้อมูลได้จริง และทำการเปรียบเทียบผลของการลดข้อมูลของภาพที่มีความละเอียด แตกต่างกันไปดังที่จะได้แสดงต่อไป

6.1 ขั้นตอนการทดลอง

ลำดับของการทดลองเป็นดังนี้

1. ในขั้นแรกทำการติดตั้งกล้องเข้ากับคอมพิวเตอร์
2. หลังจากนั้นทำการถ่ายภาพจากกล้องโดยใช้ฉากที่แตกต่างกัน 3 แบบดังนี้
    1. ภาพที่เป็นวงกลมขนาดใหญ่ เป็นภาพที่มีรายละเอียดน้อย ให้ไฟล์ผลลัพธ์ชื่อ bigc.huf
    2. ภาพวงกลมขนาดเล็กหลาย ๆวง เป็นภาพที่มีรายละเอียดมากขึ้น ให้ไฟล์ผลลัพธ์ชื่อ smlc.huf
    3. ภาพบรรยากาศภายในห้อง เป็นภาพที่มีรายละเอียดมากที่สุด ให้ไฟล์ผลลัพธ์ชื่อ view.huf

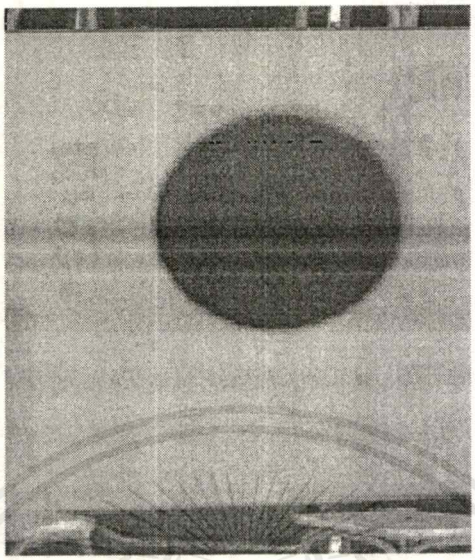
และทำการเลื่อนภาพช้า ๆ โดยทำการถ่ายฉากละ31 เฟรม เท่ากันหมด โดยที่ขณะถ่ายภาพอยู่นั้นโปรแกรมก็จะทำการ compress ข้อมูลไปในตัว ทำให้ภาพที่ได้มีการกระตุกบ้าง เนื่องจากเวลาในการ compress นั้นต้องใช้เวลาพอสมควร

3.ทำการขยายไฟล์ที่ compress ไว้แล้ว

6.2 ผลการทดลอง

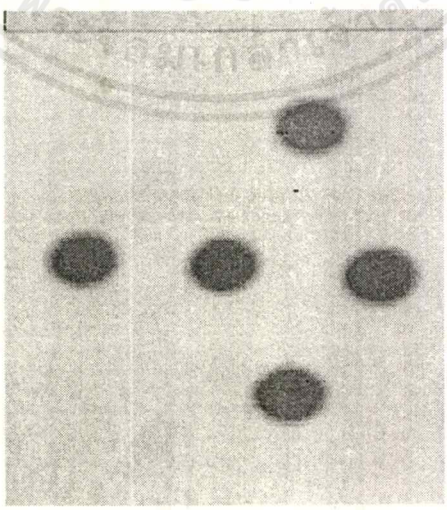
เมื่อทำการทดลองแล้วจะได้ผลดังนี้

การทดลองที่ 1 เมื่อทดลองโดยใช้ภาพวงกลมขนาดใหญ่ ดังรูปที่ 6.1 มีจำนวนเฟรมทั้งหมด 31 เฟรม ดังนั้นจะได้จำนวนข้อมูลทั้งหมด 1.5872 ล้านไบท์ แต่ไฟล์ bigc.huf ซึ่งเป็นไฟล์ข้อมูลที่ทำการลดข้อมูลแล้วนั้น สามารถลดข้อมูลเหลือ 519,104 ไบท์



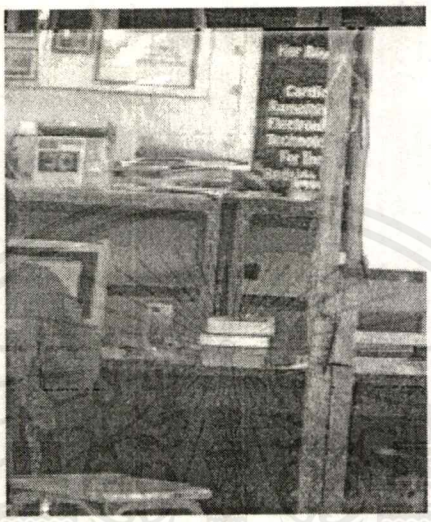
รูปที่ 6.1 แสดงถึงภาพที่ใช้ในการทดลองแรก

การทดลองที่ 2 เมื่อทดลองโดยใช้วงกลมที่มีขนาดเล็กกว่า ซึ่งมีรายละเอียดมากกว่าภาพแรก ดังแสดงในรูปที่ 6.2 ซึ่งได้ผลการทดลองว่าสามารถลดข้อมูลได้น้อยกว่าภาพแรก โดยสามารถลดข้อมูลเป็นไฟล์ smlc.huf เหลือ 569,996 ไบท์



รูปที่ 6.2 แสดงถึงรูปที่ใช้ในการทดลองที่ 2

การทดลองที่ 3 เมื่อทดลองโดยใช้ฉากเป็นภาพภายในห้อง ซึ่งเป็นภาพที่มีรายละเอียดสูง ดังแสดงในรูปที่ 6.3 พบว่าสามารถลดข้อมูลได้น้อยที่สุด โดยสามารถลดข้อมูล เป็นไฟล์ view.huf 773,871 ไบท์



รูปที่ 6.3 แสดงถึงฉากที่ใช้ในการทดลองที่ 3

บทที่ 7
สรุปและวิจารณ์ผลการทดลอง

จากการทดลองนี้สามารถสรุปได้ว่าโปรแกรมที่ได้นำเสนอนี้สามารถลดข้อมูลได้จริง โดยจะลดได้เท่าไรนั้นก็ขึ้นอยู่กับรายละเอียดของภาพที่จะนำมา compress ซึ่งสามารถสรุปได้ดังนี้

1. ภาพที่มีรายละเอียดน้อยสามารถที่จะลดข้อมูลได้มาก โดยจากการทดลองนี้ แบ่งได้ 2 รูป คือ

รูปวงกลมใหญ่ ถ้าไม่ compress จะมีขนาด 1.5872 ล้านไบท์ เมื่อ compress แล้วเหลือเพียง 519,104 ไบท์ หรือสามารถลดข้อมูลได้เหลือ 32.7 %

รูปวงกลมเล็ก ถ้าไม่ compress จะมีขนาด 1.5872 ล้านไบท์ เมื่อ compress แล้วเหลือ 569,996 หรือ 35.9 %

2. ภาพที่มีรายละเอียดมาก(ในที่นี้ก็คือภาพของบรรยากาศภายในห้อง) การลดข้อมูลจะทำได้น้อยลง โดยถ้าไม่ compress จะมีขนาด 1.5872 ล้านไบท์ เมื่อ compress แล้ว เหลือ 773,871 ไบท์ หรือ 48.7 %

สามารถสรุปได้ว่าโครงงานนี้สามารถที่จะลดข้อมูลได้จริง โดยสามารถที่จะลดข้อมูลได้ประมาณ 50-70 % ซึ่งขึ้นกับรายละเอียดของภาพด้วย

หนังสืออ้างอิง

1. Image Processing theory algorithms & architecture, Maher.A. Sid Ahhmed , McGraw-Hill International ,1992
2. Practical digital video with programming examples in C ,Phillip E. Mattisonn,John Wiley & Sons, Inc. , 1994
3. TARGA+ hardware technical reference manual ,Truevision , 1991

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จลงได้เนื่องจากบุคคลหลายฝ่ายที่ได้กรุณาช่วยเหลือให้ผู้จัดทำสามารถทำงานให้สำเร็จลุล่วงได้ด้วยดี และวิทยานิพนธ์เล่มนี้คงจะไม่สามารถสำเร็จลงได้หากไม่ได้รับความช่วยเหลือทางด้านอุปกรณ์ คำแนะนำ ข้อมูล และตำราจาก รศ. ดร. มนัส สังวรศิลป์

และขอขอบคุณพี่ปริญญาโทที่ได้ให้คำแนะนำต่าง ๆ อันเป็นประโยชน์ในการทำวิทยานิพนธ์เล่มนี้ สุดท้ายนี้ขอขอบคุณเพื่อนทุกคนที่คอยให้กำลังใจตลอดมา

ภาคผนวก

```
/*****************************************************************/
/*                    DEFINE ADDRESS REGISTER                    */
/*****************************************************************/
/*                         TARGA+                                */
/*****************************************************************/
/*                    TOTAL OF INDIRECT                          */
#define CLOCK     0x20  /* register number for rt                 */
#define GENCTRL   0x21  /* register number for gctrl              */
#define VTOTAL    0x40  /* register number for vt                 */
#define HTOTAL    0x41  /* register number for ht                 */
#define SYNC      0x42  /* register number for vs and hs          */
#define HPHASE    0x43  /* register number for hp                 */
#define VBEND     0x44  /* register number for vbe                */
#define HBSTRT    0x45  /* register number for hbs                */
#define HBEND     0x46  /* register number for hbe                */
#define VSTRT     0x47  /* register number for vas                */
#define VEND      0x48  /* register number for vae                */
#define HSTRT     0x49  /* register number for Horizontal Active Start */
#define HEND      0x4a  /* register number for Horizontal Active End   */
#define BURST     0x4b  /* register number for Burst Gate Horizontal Start */
#define SGCTRL1   0x4c  /* register number for Sync Generator Control 1 */
#define SGCTRL2   0x4d  /* register number for Sync Generator Control 2 */
#define SGSTATUS  0x4e  /* register for sync register sync generator status */
#define LINECNT   0x53  /* sync register */
#define TOP       0x80  /* register number for Top                */
#define BOT       0x81  /* register number for Bottom             */
#define VPAN      0x82  /* register number for Vpan               */
#define DSCAN     0x84  /* register number for Nint               */
#define CLOCKMODE 0x85  /* register number for Clock Mode         */
#define ADVANCED  0x90  /* register number for the PERM register  */
#define WAIT      0x91  /* register number for DAC clock and data */
#define CEM       0x92  /* register number for Byte Capture Enable Mask */
#define TAP       0xa0  /* register for display register - Horizontal*/
#define MEMORY    0xa1  /* Configuration Register */
#define BITCAP    0xb0  /* Configuration Register */
```

```
#define VGA       0xd0   /* Mixer Register */
#define COMP0     0xd1   /* Mixer Register */
#define COMP1     0xd2   /* Mixer Register */
#define COMP2     0xd3   /* Mixer Register */
#define VGAMASK   0xd3   /* Mixer Register */
#define LUTWRITE  0xd8   /* register number for lut write address        */
#define LUTCOLOR  0xd9   /* register number for lut color port           */
#define LUTMASK   0xda   /* register number for lut mask value           */
#define LUTREAD   0xdb   /* register number for lut read register        */
#define LUTCOMMAND 0xde  /* register number for Brooktree 473 register   */
#define LIVEMIXZERO    0xe5   /* register number for vs and hs           */
#define SVIDEO         0xe7   /* register number for the SVIDEO flag     */
#define VIDEOMODE      0xe8   /* register number for Buffer Port Source  */
#define BUFFERPORT     0xe9   /* register number for Buffer prt color    */
#define MIXCTRL3       0xea   /* register number for Overlay Source      */
#define LIVEPORT       0xeb   /* register number for Old Page Select     */
#define INVERT         0xec   /* register number for Live Port invert ctl */
#define NOTOVLLEVEL    0xed   /* LIVEMIX value during not overlay        */
#define OVLLEVEL       0xee   /* LIVEMIX value during overlay            */

/************************* END OF INDIRECT REGISTER ************************/
#define COLOR1          0x001
#define COLOR2          0x002
#define COLOR3          0x003
#define VIDCON          0x400
#define INDIRECT        0x401
#define HUESAT          0x402
#define MASKL           0x800
#define MASKH           0x801
#define LBNK            0x802
#define HBNK            0x803
#define MODE1           0xC00
#define MODE2           0xC01
#define WBL             0xC02
#define WBH             0xC03
```

```
#define VIDSTAT     0x000
#define CTL         0x002
#define READAD      0x401
#define RBL         0xC02
#define RBH         0xC03
```

```
// This Program capture the image from camera , and compress while capturing
// By          Yanapol      Unosot          36014128
               Wirote       Panyachatiraksa 36014412
#include <dos.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <mem.h>
#include <bios.h>
#include <alloc.h>
#include <string.h>
#include <io.h>
#include "bmp.h"
#include "reg.h"
#include "params.h"
#define  speed   23
#define  PARALLEL 0x378
#define  C_BLUE    1

typedef unsigned char DacPalette256[256][3];
void init(void);
void initcard();
void setvgapalette256(DacPalette256 * PalBuf);
void set_palette(void);
void mvec(unsigned char far *prv,unsigned char far *cur);
void left(void);
void right(void);
void rlc(unsigned char far *sub);

void freeze();
void unfreeze();
void getframe1();
void getframe2();
void process();
```

```
void huffman();

extern int far _cdecl Svga256_fdriver[];

int huge DetectVGA256(void)
{
        return 2;
}
#define  TARGACARD 0x220
#define  resolution 0x20
int count,countframe,nframe;
int ram,dat1,dat2;
unsigned char far *screen;
unsigned char far *frame1_1;
unsigned char far *frame2_1;
int a.b,c,i,j,DY_MAX,DX_MAX:
int key,ascii,diff,scan,TURN,zoom,change,a_start,b_start.countline;
long int center;
unsigned char store[X_SIZE];
FILE *fi,*fo,*fx;
PPR header;

void main(void)
{   /****************** SET IMAGE CAPTURE ******************/

        frame1_1=(unsigned char far *)farcalloc(Y_SIZE*X_SIZE,sizeof(unsigned char));
        if(frame1_1==NULL)
         {
          perror("Can't allocate memory\n");
          exit(1);
         }
        frame2_1=(unsigned char far *)farcalloc(Y_SIZE*X_SIZE,sizeof(unsigned char));
```

```
 if(frame2_1==NULL)
  {
   perror("Can't allocate memory\n");
   exit(1);
  }
 change=0;
 zoom=0;
 j=0;
 ram=0x00;
 printf("DY_MAX : ");
 scanf("%d",&DY_MAX);
 printf("DX_MAX : ");
 scanf("%d",&DX_MAX);
 initcard();
 init();
 set_palette();
 delay(40);
//  DisplayBMP("VISUAL.BMP");
 setfillstyle(1,255);
 bar(0,0,639,479);
 setcolor(C_BLUE);
 settextstyle(1,0,4);
 setfillstyle(1,C_BLUE);
 outtextxy(270,50,"IMAGE PROCESSING");
 bar(20,100,getmaxx()-20,getmaxy()-20);

 nframe=0;
 fo=fopen("image.tga","wb");
 if(fo==NULL)
  {
   perror("Can't copy image.tga for write");
   exit(1);
```

```
        }
        strcpy(header.file_ID,"PROJ");
        fwrite(&header,sizeof(header),1,fo);
        getch();
        freeze();
        getframe1();
        unfreeze();
        nframe++;
        for(i=0;i<Y_SIZE;i++)
        {
         _fmemcpy(&store[0],&frame1_1[i*X_SIZE],X_SIZE);
         fwrite(store,sizeof(store),1,fo);
        }
LOOP:  do  {
                freeze();
                getframe2();
                unfreeze();
                nframe++;
                mvec(frame1_1,frame2_1);
                _fmemcpy(&frame1_1[0],&frame2_1[0],Y_SIZE*X_SIZE);
         }  while(!kbhit());
         key=bioskey(0);
         ascii=key&0xff;  // (low byte)
         scan=key>>8;     // (high byte)
         if ((ascii==0x0d)&&(scan==0x1c)) goto quitprog;  //Enter
         goto LOOP;

quitprog:       farfree(frame1_1);
                farfree(frame2_1);
                header.x_size=(int)X_SIZE;
                header.y_size=(int)Y_SIZE;
                header.frames=nframe;
```

```
                    fseek(fo,0L,SEEK_SET);
                    fwrite(&header,sizeof(header),1,fo);
                    fseek(fo,0L,SEEK_END);
                    fcloseall();
                    closegraph();
                    printf("Frame of image = %d\n",nframe);
                    fi=fopen("image.tga","rb");
                    if(fi==NULL)
                     {
                      perror("Can't copy image.tga for read");
                      exit(1);
                     }
                    fo=fopen("image.huf","wb");
                    if(fo==NULL)
                     {
                      perror("Can't copy image.huf for write");
                      exit(1);
                     }
                    huffman();
                    fcloseall();
} /**************** END OF MAIN ***********************************/
void init(void)
{
        int gd = DETECT, gm, errorcode;
        installuserdriver("Svga256", DetectVGA256);
        registerfarbgidriver(Svga256_fdriver);
        initgraph(&gd,&gm,"");
        return;
}

void setvgapalette256(DacPalette256 * PalBuf)
{
```

```
        struct REGPACK  reg;
        reg.r_ax = 0x1012;
        reg.r_bx = 0;
        reg.r_cx = 256;
        reg.r_es = FP_SEG(PalBuf);
        reg.r_dx = FP_OFF(PalBuf);
        intr(0x10, &reg);
}

void set_palette(void)
{
        int i;
        unsigned char palette[256][3];

        for (i = 0; i < 256; i++){
         palette[i][0] = palette[i][1] = palette[i][2] = i>>2;
        }
         palette[C_BLUE][0] = 0x00;
         palette[C_BLUE][1] = 0x00;
         palette[C_BLUE][2] = 0xff;
         setvgapalette256(&palette);
}

void freeze()
{       outportb(TARGACARD+MODE2,0x80+zoom);

}
void unfreeze()
{       outportb(TARGACARD+MODE2,0xD0+zoom);
}
void getframe1()
{  // start at bank 16  1 bank == 64 row
```

```
    c=16;//DEFAULT 16
    a_start=0;
    countframe=0;
    count=0;
    for(a=a_start;a<400;a++)    //256*200   x=256 y=200
    {  // set bank
                    if(!(a%64))
                     {  outportb(TARGACARD+LBNK, c);
                       c++;
                       screen = (unsigned char *)MK_FP(0xD000,0);
                      }
                    for(b=0;b<512;b=b+2) // size of col 256
                    {
                      ram = *screen;
                      screen=screen+2;
                      if (ram<4) ram=4;
                      ram=(ram>>3)<<3;
                      if((a%2)==1)
                       {
                        putpixel(192+(b)/2, 400-(a/2),ram);
                        frame1_1[count]=ram;
                        count++;
                       }
                     }
    } //end for LOOP
}
void getframe2()
{   c=16;//DEFAULT 16
    a_start=0;
    countframe=0;
    count=0;
    for(a=a_start;a<400;a++)
```

```
{  // set bank
                if(!(a%64))
                 {
                   outportb(TARGACARD+LBNK, c);
                   c++;
                   screen = (unsigned char *)MK_FP(0xD000,0);
                 }
                for(b=0;b<512;b=b+2) // size of col 256
                 {
                   ram = *screen;
                   screen=screen+2;
                   if (ram<4) ram=4;
                   ram=(ram>>3)<<3;
                   if((a%2)==1)
                    {
                     putpixel(192+(b)/2,400-(a/2),ram);
                     frame2_1[count]=ram;
                     count++;
                    }
                 }

        } //end for LOOP
};
void process()
{
  /*  countframe=0;
    count=0;
                for(a=a_start;a<400;a++)
                {
                  for(b=0;b<512;b=b+2) // size of col 256
                   {
                    if((a%2)==1)
```

```
                {
                    dat1=frame1_1[count];
                    dat2=frame2_1[count];
                    count++;
                }
                if (abs(dat1-dat2)>resolution)
                    { putpixel(50+(b)/2,400-(a/2),4);
                     store[b/2]=1;
                    }
                    else
                    { putpixel(50+(b)/2,400-(a/2),255);
                     store[b/2]=0;
                    }

                } // end for  b

            } */ //end for a
        // **********detect edge of moving picture
 mvec(frame1_1,frame2_1);
}

void initcard()
{       outportb(PARALLEL,0x00);            //CLR  PORT
        outportb(TARGACARD+VIDCON,0x20);      /* SET RGB    */
        outportb(TARGACARD+INDIRECT.SVIDEO):   /* SET SVIDEO */
        outportb(TARGACARD+WBL,0xFF);
        outportb(TARGACARD+MODE1,0x01);       /* SET FOR USE INDIRECT */
        outportb(TARGACARD+INDIRECT.MIXCTRL3): /* SET CM3 */
        outportb(TARGACARD+WBL,0x05);
        outportb(TARGACARD+INDIRECT,VIDEOMODE); /* SET
CM2=0,CM1=0,MONOSRC,BUFFERPORT -->  BLACK & WHITE */
        outportb(TARGACARD+WBL,0x80);
```

```
        outportb(TARGACARD+INDIRECT,LIVEPORT);  /* SET
  LIVEPORTWORD,LIVE8,FGB,LIVEPORTSRC */
        outportb(TARGACARD+WBL,0x30);
        outportb(TARGACARD+INDIRECT,INVERT);   /* SET INVERT */


        outportb(TARGACARD+WBL,0x27);
        outportb(TARGACARD+INDIRECT,BUFFERPORT);/* SET
 LUTBYPASS,LIVEMIXCOLOR,BUFFERPORTCOLOR*/
        outportb(TARGACARD+WBL,0x20);
        outportb(TARGACARD+INDIRECT,LIVEMIXZERO);
        outportb(TARGACARD+WBL,0x00);
        outportb(TARGACARD+INDIRECT,VGA);
        outportb(TARGACARD+WBL,0x0e);
        outportb(TARGACARD+INDIRECT,NOTOVLLEVEL);
        outportb(TARGACARD+WBL,0x00);
        outportb(TARGACARD+INDIRECT,OVLLEVEL);
        outportb(TARGACARD+WBL,0xff);
        outportb(TARGACARD+INDIRECT,ADVANCED);
        outportb(TARGACARD+WBL,0xc4);
        outportb(TARGACARD+INDIRECT,BITCAP);
        outportb(TARGACARD+WBL,0xff);
        outportb(TARGACARD+WBH,0xff);
        outportb(TARGACARD+MODE2,0xd0);       /* SET
CAPTURE,GENLOCK,ZOOM,DISPLAY*/
        clrscr();
        screen = (unsigned char*)MK_FP(0xD000,0);
        outportb(TARGACARD+VIDCON, 0x20);  // set to S-video input
}

void mvec(unsigned char far *prv,unsigned char far *cur)
{
```

```
unsigned char image_pr,image_ct,image_sb;
char vx[Y_SIZE/Y_BLOCK][X_SIZE/X_BLOCK];
char vy[Y_SIZE/Y_BLOCK][X_SIZE/X_BLOCK];
unsigned char far *sub;
int a,b,c,i,j,dx,dy,ddx,ddy;
int k,p;
int x,y,xs,ys,xd,yd;
unsigned int m,n;
float d,min;
unsigned char store[X_SIZE];

sub=(unsigned char far *)farcalloc(Y_SIZE*X_SIZE,sizeof(unsigned char));
if(sub==NULL)
 {
 perror("Can't allocate memory\n");
 exit(1);
 }
for(i=0;i<Y_SIZE/Y_BLOCK;i++){/*mvec*/
        printf("Frame %d-%d Line %d \r",a,a+1,i);
        for(j=0;j<X_SIZE/X_BLOCK;j++){
                for(dy=-DY_MAX;dy<=DY_MAX;dy++){
                 for(dx=-DX_MAX;dx<=DX_MAX;dx++){
                        d=0;
                        for(y=0;y<Y_BLOCK;y++){
                                yd=i*Y_BLOCK+y;
                                ys=yd+dy;
                                if(ys<0)    ys=0;
                                if(ys>Y_SIZE-1)  ys=Y_SIZE-1;
                                for(x=0;x<X_BLOCK;x++){
                                 xd=j*X_BLOCK+x;
                                 xs=xd+dx;
                                 if(xs<0)   xs=0;
```

```
                    if(xs>X_SIZE-1)xs=X_SIZE-1;

                    image_ct=cur[yd*X_SIZE+xd];

                    image_pr=prv[ys*X_SIZE+xs];

                    d+=abs(image_ct-image_pr);
                  }
                }
                if(dx==-DX_MAX&&dy==-DY_MAX){
                    min=d;ddx=dx;ddy=dy;
                                  }
                if(d<min){
                    min=d;ddx=dx;ddy=dy;
                  }
               if(min==0) break;
              }
              if(min==0) break;
            }
        vx[i][j]=ddx;vy[i][j]=ddy;
/*----------subtract----------*/
        for(y=0;y<Y_BLOCK;y++){
         yd=i*Y_BLOCK+y;
         ys=yd+vy[i][j];
         if(ys<0)  ys=0;
         if(ys>Y_SIZE-1)  ys=Y_SIZE-1;
    for(x=0;x<X_BLOCK;x++){
         xd=j*X_BLOCK+x;
         xs=xd+vx[i][j];
         if(xs<0)  xs=0;
         if(xs>X_SIZE-1)  xs=X_SIZE-1;
```

```
            image_ct=cur[yd*X_SIZE+xd];
            image_pr=prv[ys*X_SIZE+xs];

            k=image_ct-image_pr+LEVEL/2;
            if(k<0)  k=0;
            if(k>255) k=255;

            sub[yd*X_SIZE+xd]=k;

          }
          }
 /*---------subtract---------*/
    }
  }
//  for(y=0;y<Y_SIZE;y++)
     {
//     _fmemcpy(&store[0],&sub[y*X_SIZE],X_SIZE);
//     fwrite(store,sizeof(store),1,fo);
//    }
  fwrite(vx,sizeof(vx),1,fo);
  fwrite(vy,sizeof(vy),1,fo);

  rlc(sub);
 farfree(sub);
}

 void rlc(unsigned char far *sub)
 {
  int p,k;
  long step;

  step=0;
```

```
while(step<(Y_SIZE*X_SIZE))
 {
 p=sub[step];
 k=0;
 if(step==((Y_SIZE*X_SIZE)-1))
  {
  if((p&MASK)==MASK)
        {
         k=k|MASK;
         putc(k,fo);
         putc(p,fo);
        }
  else putc(p,fo);
  break;
 }
step++;
 while(p==sub[step])
  {
  step++;
  k++;
  if((k==NMASK)||(step==Y_SIZE*X_SIZE)) break;
 }
if(k==0)
 {
 if((p&MASK)==MASK)
        {
         k=MASK;
         putc(k,fo);
         putc(p,fo);
        }
       else putc(p,fo);
 }
```

```
    else
     {
      k|=MASK;
      putc(k,fo);
      putc(p,fo);
     }
   }
}

void huffman()
{
 float prob[NUM],z;
 float tree[NUM];
 int code[3][NUM];
 int temp[NUM+1][NUM];
 int count.over.k;
 PPR header:
 unsigned char length[NUM],tmp;
 unsigned char sub1,store[X_SIZE];
 unsigned int buf;
 unsigned long ck=0,az,ck1=0,step1;

  for(i=0:i<NUM;i++){
  prob[i]=0;
  tree[i]=0;
  code[0][i]=0;
  code[2][i]=0;
  for(j=0;j<NUM+1:j++)temp[j][i]=NUM+1;
  }

 j=fileno(fi);
 ck=filelength(j)-sizeof(header);
```

```
fread(&header.sizeof(header),1,fi);
fwrite(&header,sizeof(header),1,fo);
fwrite(&ck,sizeof(ck),1,fo);
fwrite(&ck1,sizeof(ck1),1,fo);

sub1=0;
for(az=0,k=255,j=0;az<ck;az++)
 {
 if(i!=(int)(az*50/(ck-1)))
 {
  i=(int)(az*50/(ck-1));
  printf("%d%\r",i);
 }
 fread(&tmp,sizeof(tmp),1,fi);
 prob[(tmp>>SYM)]+=1.0;
 if((j+SYM)<=8)
  {
   sub1<<=SYM:
   sub1+=(tmp&(k>>(8-SYM)));
   j+=SYM;
  }
 else
  {
   fwrite(&sub1,sizeof(sub1),1,fo);
   ck1++;
   sub1<<=SYM;
   sub1+=(tmp&(k>>(8-SYM)));
   j=0;
   j-=SYM:
  }
 if((az==ck-1)&&(j<=8))
  {
```

```
    sub1<<=(8-j);
    fwrite(&sub1,sizeof(sub1),1,fo);
    ck1++;
   }
  }
az=(-(ck1+sizeof(ck1)));
fseek(fo,az,SEEK_CUR);
fwrite(&ck1,sizeof(ck1),1,fo);
fseek(fo,ck1,SEEK_CUR);

for(i=0;i<NUM;i++) {
 temp[0][i]=1;
 prob[i]/=ck;
 tree[i]=prob[i];
 temp[1][i]=i;
 code[1][i]=i:
}
for(i=0;i<NUM;i++){
 for(j=i:j<NUM;j++){
  if(i!=j && tree[i]<tree[j]){
   z=tree[j];
   tree[j]=tree[i];
   tree[i]=z;
   z=temp[1][j];
   temp[1][j]=temp[1][i];
   temp[1][i]=z;
  }
 }
}

for(b=NUM-1:b>=0:b--){
 if((tree[0]==1.0)&&(temp[0][0]==1))
```

```
 {
  code[0][temp[1][0]]=1;
  code[2][temp[1][0]]=1;
 }
else
{
if((tree[b]!=0)&&(b!=0))
{
 tree[b-1]+=tree[b];
 tree[b]=0;

 for(i=0;i<temp[0][b-1];i++){
 code[0][temp[i+1][b-1]]<<=1;
 code[2][temp[i+1][b-1]]+=1;
 }

 for(j=0;j<temp[0][b];j++){
 code[0][temp[j+1][b]]<<=1;
 code[0][temp[j+1][b]]+=1;
 code[2][temp[j+1][b]]+=1;
 }

 for(k=0;k<temp[0][b];k++){
 temp[0][b-1]++;
 temp[temp[0][b-1]][b-1]=temp[k+1][b];
 }

 for(i=0;i<NUM;i++){
  for(j=i;j<NUM;j++){
   if(i!=j && tree[i]<tree[j]){
    z=tree[j];
    tree[j]=tree[i];
```

```
    tree[i]=z;
    for(k=0;k<NUM+1;k++){
     a=temp[k][j];
     temp[k][j]=temp[k][i];
     temp[k][i]=a;
     }
    }
   }
  }
  }
 }
}

b=1;
for(i=0;i<NUM;i++){
 for(j=0,k=0;j<code[2][i];j++){
  k=((code[0][i]>>j)&b);
  if(j!=code[2][i]-1)k<<=1;
 }
 code[0][i]=k;
}

for(i=0;i<NUM;i++){
  for(j=i+1;j<NUM;j++){
   if(i!=j && code[2][i]<code[2][j]){
    a=code[2][j];
    code[2][j]=code[2][i];
    code[2][i]=a;
    a=code[1][j];
    code[1][j]=code[1][i];
    code[1][i]=a;
    a=code[0][j];
```

```
    code[0][j]=code[0][i];
    code[0][i]=a;
  }
 }
}
rewind(fi);
fread(&header,sizeof(header),1,fi);

fwrite(code,sizeof(code),1,fo);
printf("                    \r");
printf("                    \r");
b=16;
k=0;
count=0;
over=0;
buf=0;
c=0;
j=1;
for(az=0;az<ck;az++)
{
 if(j!=(int)(az*50/(ck-1)))
  {
  j=(int)(az*50/(ck-1));
  printf("%d%%\r",j+50);
  }
 if(over==0)
 {
  fread(&tmp,sizeof(tmp),1,fi);
  for(i=0;i<NUM;i++) if((tmp>>SYM)==code[1][i]) {a=i;break;}
  if((b-code[2][a])<0)
  {
   c=code[0][a]>>(code[2][a]-b);
```

```
    b=16-(code[2][a]-b);
    over=b;
   }
   else
   {
    c=code[0][a]<<(b-code[2][a]);
    b-=code[2][a];
    if(b==0){b=16;count=1;}
   }
  }
  else
  {
   az--;
   c=code[0][a]<<over;
   over=0;
  }
  buf+=c;
  if((over!=0)||(count==1)||(az==ck-1)){
   fwrite(&buf,sizeof(buf),1,fo);
   buf=0;
   count=0;
  }
  }
}
```

```
/* add motion vector and frames image file */
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<alloc.h>
#include<string.h>
#include"params.h"
#include"bmp.h"
#define POS_X 192
#define POS_Y 340
#define C_BLUE 1

extern int far _Cdecl Svga256_fdriver[];
int huge DetectVGA256(void)
{
        return 2;
}

void error_graph(int errorcode)
{
 if(errorcode != grOk){
  printf("Graphics error %d\n",errorcode);
  exit(1);
  }
}

void init(void)
{
        int gd = DETECT, gm, errorcode;
        installuserdriver("Svga256", DetectVGA256);
```

```
        registerfarbgidriver(Svga256_fdriver);
        initgraph(&gd,&gm,"");
        errorcode=graphresult();
        error_graph(errorcode);
}

void setvgapalette256(DacPalette256 * PalBuf)
{
        struct REGPACK  reg;
        reg.r_ax = 0x1012;
        reg.r_bx = 0;
        reg.r_cx = 256;
        reg.r_es = FP_SEG(PalBuf);
        reg.r_dx = FP_OFF(PalBuf);
        intr(0x10, &reg);
}

void set_palette(void)
{
        int i;
        unsigned char palette[256][3];

        for (i = 0; i < 256; i++){
         palette[i][0] = palette[i][1] = palette[i][2] = i>>2;
        }
         palette[C_BLUE][0] = 0x00;
         palette[C_BLUE][1] = 0x00;
         palette[C_BLUE][2] = 0xff;
         setvgapalette256(&palette);
}

void main(void)
```

```
{
 unsigned char nframe,store[X_SIZE],buff[Y_BLOCK][X_BLOCK];
 unsigned char image_pr,image_ct,image_sb;
 unsigned char far *prv,far *cur;
 char vx[Y_SIZE/Y_BLOCK][X_SIZE/X_BLOCK];
 char vy[Y_SIZE/Y_BLOCK][X_SIZE/X_BLOCK];
 FILE *fi,*fo,*fx;
 char namei[14],nameo[14],namev[14],name[14];
 int i,j,k,p,dx,dy;
 int x,y,xs,ys,xd,yd;
 int a,b,c;
 unsigned int m,n;
 int code[3][NUM];
 unsigned char length[NUM],buf,sub1;
 unsigned int buf1,buf2,temp,cmp=0xffff;
 int count,over;
 unsigned long ck,az,ck1,bz;
 long STEP,step1,step2;
 PPR header;

 /* huffman decode all*/
 clrscr();
 printf("Please enter NAME of file :");
 gets(name);
 strcpy(namei,name);
 strcat(namei,".huf");
 strcpy(nameo,name);
 strcat(nameo,".out");

 fi=fopen(namei,"rb");
 if(fi==NULL){
  perror("Can't open input file\n");
```

```
 exit(0);
}
fo=fopen(nameo,"wb");
if(fo==NULL){
 perror("Can't open output file\n");
 exit(0);
}
 fread(&header,sizeof(header),1,fi);
 fwrite(&header,sizeof(header),1,fo);
 fread(&ck,sizeof(ck),1,fi);
 fread(&ck1,sizeof(ck1),1,fi);
 STEP=ck1;
 if(ck1>65536) STEP=65536;

cur=(unsigned char far *)farcalloc(STEP,sizeof(unsigned char));
if(cur==NULL)
 {
  perror("Can't allocate memory\n");
  exit(0);
 }

 step1=ftell(fi):
 fseek(fi.ck1.SEEK_CUR);
 fread(code,sizeof(code),1,fi);

for(i=0;i<NUM;i++)if(code[2][i]==0){code[0][i]=255:code[2][i]=255;}

step2=ftell(fi);
fseek(fi,step1-step2,SEEK_CUR);
for(bz=0;bz<STEP;bz++)
  {
   fread(&sub1,sizeof(sub1),1,fi);
```

```
  cur[bz]=sub1;
 }
step1+=STEP;
fseek(fi,step2-step1,SEEK_CUR);

over=0;
c=16;
count=16;
b=0;
fread(&buf1,sizeof(buf1),1,fi);
fread(&buf2,sizeof(buf2),1,fi);
temp=buf2;
fread(&buf2,sizeof(buf2),1,fi);
for(az=0,ck1=0,a=ck/50,k=255,c=0;az<ck;az++)
{
 if(a!=(int)(az*100/ck))
  {
   a=(int)(az*100/ck);
   printf("%d%\r",a);
  }
 for(j=15;j>=0;j--)
 {
  for(i=0;i<NUM;i++)
  {
   if((code[0][i]==(buf1>>j))&&(code[2][i]==(16-j)))
   {
    over=1;
    buf=code[1][i];
    buf<<=SYM;
    if((c+SYM)>8)
    {
    c=0;
```

```
ck1++;
if(ck1==STEP)
 {
  ck1=0;
  step2=ftell(fi);
  fseek(fi,step1-step2,SEEK_CUR);
  for(bz=0;bz<STEP;bz++)
       {
        fread(&sub1,sizeof(sub1),1,fi);
        cur[bz]=sub1;
       }
       step1+=STEP;
       fseek(fi,step2-step1,SEEK_CUR);
 }
}
buf+=((cur[ck1]>>(8-SYM-c))&(cmp>>(16-SYM)));
c+=SYM;
fwrite(&buf,sizeof(buf),1,fo);
if(count<=0)
{
 fread(&buf2,sizeof(buf2),1,fi);
 b+=abs(count);
 count=16;
 if(b!=0)
 {
  temp1=buf2>>(16-b);
  buf2<<=b;
  count-=b;
  b=0;
 }
}
buf1<<=(16-j);
```

```
    buf1|=temp>>j;
    temp<<=(16-j);
    temp|=buf2>>j;
    buf2<<=(16-j);
    count+=(16-j);
   }
    if(over!=0)break;
  }
   if(over!=0){over=0;break;}
 }
}
farfree(cur);
fcloseall();
printf("%s                \n",nameo);

strcpy(namei.nameo);

fi=fopen(namei."rb");
if(fi==NULL)
 {
  perror("Can't open input file");
  exit(1);
 .
prv=(unsigned char far *)farcalloc(Y_SIZE*X_SIZE,sizeof(unsigned char));
if(prv==NULL)
 {
  perror("Can't allocate memory\n");
  exit(1);
 }
cur=(unsigned char far *)farcalloc(Y_SIZE*X_SIZE,sizeof(unsigned char));
if(cur==NULL)
 {
```

```
 perror("Can't allocate memory\n");
 exit(1);
 }
fread(&header,sizeof(header),1,fi);
for(i=0;i<Y_SIZE;i++)
 {
 fread(store,sizeof(store),1,fi);
 _fmemcpy(&prv[i*X_SIZE],&store[0],X_SIZE);
 }
nframe=header.frames;
init();
set_palette();
setfillstyle(SOLID_FILL,1);
bar(0,0,639,479);
setcolor(255);
settextstyle(1.0,4);
outtextxy(270,50,"IMAGE PROCESSING");
getch();
for(a=0;a<nframe-1;a++)
 {
 fread(vx,sizeof(vx),1,fi);
 fread(vy,sizeof(vy),1,fi);
 step1=0;
 while(1)
  {
  if(step1==Y_SIZE*X_SIZE) break;
  k=getc(fi);
  if(k==EOF) break;
  if((k&MASK)==MASK)
   {
   p=getc(fi);
   if(p==EOF)
```

```
        {
//        putc(k,fo);
          cur[step1]=k;
          step1++;
          break;
        }
    k=k&(NMASK);
    for(j=0;j<(k+1);j++)
        {
//        putc(p,fo);
          cur[step1]=p;
          step1++;
        }
   }
  else
   {
//   putc(k,fo);
    cur[step1]=k;
    step1++;
   }
  } //end of while


for(i=0;i<Y_SIZE;i++)
 for(j=0;j<X_SIZE;j++)
  putpixel(j+334,POS_Y-i,cur[i*X_SIZE+j]);


for(i=0;i<Y_SIZE/Y_BLOCK;i++){ /*mc_add*/
  for(j=0;j<X_SIZE/X_BLOCK;j++){
    for(y=0;y<Y_BLOCK;y++){
          yd=i*Y_BLOCK+y;
          ys=yd+vy[i][j];
```

```
            if(ys<0)  ys=0;
            if(ys>Y_SIZE-1)  ys=Y_SIZE-1;
        for(x=0;x<X_BLOCK;x++){
            xd=j*X_BLOCK+x;
            xs=xd+vx[i][j];
            if(xs<0)  xs=0;
            if(xs>X_SIZE-1)  xs=X_SIZE-1;

            image_sb=cur[yd*X_SIZE+xd];
            image_pr=prv[ys*X_SIZE+xs];

            k=image_pr+(image_sb-LEVEL/2);
            if(k<0)  k=0;
            if(k>255) k=255;
            cur[yd*X_SIZE+xd]=k;
           }
           }
     }
  }
 for(i=0;i<Y_SIZE;i++)
  for(j=0;j<X_SIZE;j++)
   putpixel(j+50,POS_Y-i,prv[i*X_SIZE+j]);
 delay(200);
 if(a<nframe-2)
  for(i=0;i<Y_SIZE;i++)
  {
   _fmemcpy(&store[0],&cur[i*X_SIZE],X_SIZE);
   _fmemcpy(&prv[i*X_SIZE],&store[0],X_SIZE);
   fwrite(store,sizeof(store),1,fo);
  }

 }
```

```
for(i=0;i<Y_SIZE;i++)
 for(j=0;j<X_SIZE;j++)
     putpixel(j+POS_X,POS_Y-i,cur[i*X_SIZE+j]);
 farfree(cur);
 farfree(prv);
 fcloseall();
 closegraph();
}
```