การลดข้อมูลภาพ

IMAGE DATA COMPRESSION

โดย

นาย จักรกฤษณ์ ใจมั่น

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

การลดข้อมูลภาพ

(IMAGE DATA COMPRESSION)

โดย

นาย จักรกฤษณ์ ใจมั่น

เลขประจำตัว 33100050

อาจารย์ที่ปรึกษา

รศ.ดร. มนัส สังวรศิลป์

รายงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2536

ปริญญานิพนธ์ ปีการศึกษา 2536

เรื่อง การลดข้อมูลภาพ (Image Data Compression)

ผู้จัดทำ

นาย จักรกฤษณ์ ใจมั่น
เลขประจำตัว 33100050

ภาควิชาอิเลคทรอนิคส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยี พระจอมเกล้าเจ้าคุณทหารลาดกระบัง

.....

(รศ.ดร.มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การลดข้อมูลภาพ

(Image Data Compression)

นาย จักรกฤษณ์ ใจมั่น 33100050

อาจารย์ที่ปรึกษา รศ.ดร.มนัส สังวรศิลป์

ปีการศึกษา 2536

Mr. Chakkrit Jaiman 33100050

Assoc. Prof. Dr. Manus Sangworasil

Academic Year 1993

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ ได้นำเสนอการลดข้อมูลภาพโดยใช้วิธี Block Truncation Coding (BTC) โดยวิธีนี้จะแบ่งภาพออกเป็นส่วนๆ แล้วจัดให้แต่ละส่วนมีระดับสี 2 ระดับต่อ Block ต่อ plane โดยเน้นการพัฒนาที่ การลดข้อมูลภาพให้เล็กลง แต่ยังคงรักษารายละเอียดและความคมชัดของภาพไว้ให้ได้มากที่สุด ภาพที่ได้มีคุณภาพ ดีที่สุด โดยการทำให้ค่า Error ของข้อมูลภาพต่ำที่สุด ซึ่งมี 4 วิธีด้วยกัน คือ 1.Simple BTC 2.Minimum Root Mean Square Error 3.Minimum Mean Absolute Error 4.Avarage Minimum of RMSE and MAE

ซึ่งวิธี Block Truncation Coding นี้ นับว่าเป็นวิธีที่ใช้หลักการง่ายๆ ไม่ซับซ้อน แต่ใช้ได้จริงในทางปฏิบัติเพราะ เป็นวิธีที่ลดข้อมูลภาพได้มากพอสมควร ด้วยคุณภาพของภาพที่อยู่ในขั้นดี ประมวลผลได้รวดเร็ว ซึ่งทำให้ช่วยอำนวยความสะดวกในการจัดเก็บข้อมูลภาพ และ การส่งข้อมูลภาพ ได้อย่างแท้จริง

Abstract

This thesis aim to present Image Data Compression technigue by Block Truncation Coding (BTC) method that uses a two-level (one bit) nonparametric quantizer that adapts to local properties of the image. Developed to reduce the number of image data,but still preserves the most detail and sharpness of the image for best quality. By minimizing error of image to minimum error that have 4 method is 1.Standard BTC 2.Minimum Root Mean Square Error 3.Minimum Mean Absolute Error and 4.Avarage Minimum of RMSE and MAE

This Block Truncation Coding method is non-complex method and can use to real performance. Because this method can compress high data image , good quality, and high speed in processing that can help to high quantity store and a lot of transmitting time can be save.

สารบัญ

บทที่ 1

บทนำ

(INTRODUCTION)

ในยุคปัจจุบันคอมพิวเตอร์(computer) ได้เข้ามามีบทบาทอย่างมากต่อชีวิตความเป็นอยู่แทบจะทุกๆด้านก็ว่าได้ คอมพิวเตอร์จะถูกนำไปใช้ในสาขาวิชาชีพต่างๆ เช่น ด้านวิศวกรรมด้านการแพทย์ และงานอุตสาหกรรมแขนงต่างๆ โดยเฉพาะงานทางด้านวิศวกรรมนั้นคอมพิวเตอร์จะถูกนำมาใช้แทบทุกสาขา ไม่ว่าจะเป็น ด้านการสื่อสาร งานด้านการออกแบบ โครงสร้าง งานประมวลผลภาพของ remote sensing ตลอดไปถึงงานด้านเอกสารในสำนักงานที่มีการติดต่อเชื่อมโยงเครือข่ายของข้อมูลเป็นระบบที่เรียกว่า office automation และในปัจจุบันได้เริ่มเข้าสู่ยุคของการแข่งขันทางเทคโนโลยีข่าวสาร ความสามารถของคอมพิวเตอร์ถูกวัดด้วยความเร็วในการประมวลผล และเวลาที่ใช้ในการติดต่อแลกเปลี่ยนข้อมูลข่าวสารระหว่างเครื่องและแหล่งข้อมูล

ข้อมูลข่าวสารต่างๆ ที่สำคัญนั้นนอกจากจะอยู่ในรูปของเสียง เอกสาร และสัญลักษณ์ต่างๆแล้ว ข้อมูลอีกอย่างหนึ่งที่มีความสำคัญไม่ยิ่งหย่อนไปกว่ากันนั้นก็คือภาพ โดยที่จากนั้นอาจจะอยู่ในรูปของภาพถ่ายภาพทางจอโทรทัศน์ และข้อมูลภาพอื่นๆ ที่แสดงทางจอภาพ (monitor) ข้อมูลภาพเหล่านี้สามารถนำมาใช้ประโยชน์ได้อย่างกว้างขวาง เป็นต้นว่าข้อมูลภาพถ่ายทางดาวเทียมได้นำมาใช้ประโยชน์ได้อย่างกว้างขวาง เป็นต้นว่า ข้อมูลภาพถ่ายทางดาวเทียมได้นำมาใช้ในการพัฒนาและอนุรักษ์ทรัพยากรธรรมชาติ สภาพแวดล้อมของโลก และการสำรวจหรือพยากรณ์ในกระยะทางไกล(remote sensing) อื่นๆ นอกจากนี้ยังมีการนำข้อมูลภาพไปใช้ในระบบฐานมูลที่เป็นแฟ้มประวัติบุคคล เพื่อสามารถตรวจสอบได้ทั้งประวัติและหน้าตาของผู้เป็นเจ้าของประวัตินั้น และตัวอย่างการนำภาพของคู่สนทนา (video phone) ซึ่งกำลังจะเข้ามาแทนที่โทรศัพท์ระบบเดิมในอีกไม่ช้านี้ก็เป็นได้ โดยมากภาพที่นำมาใช้ในงานต่างๆ ที่กล่าวมาแล้วนั้นจะอยู่ในรูปของข้อมูลทางดิจิตอล การที่จะให้ภาพแต่ละภาพมีรายละเอียดหรือความคมชัดเพียงพอต่อการใช้งานนั้น จะต้องใช้หน่วยความจำเป็นจำนวนมากสำหรับการเก็บข้อมูล และในกรณีที่ระบบมีการรับส่งข้อมูลภาพด้วยแล้วจะทำให้สิ้นเปลืองเวลาที่ใช้ในการรับส่งข้อมูลภาพแต่ละภาพ เนื่องจากข้อมูลมีขนาดใหญ่ซึ่งหมายถึงค่าใช้จ่ายในการรับส่งข้อมูลสูงตามไปด้วย จากปัญหาดังกล่าวจึงเป็นที่มาของปริญญานิพนธ์ฉบับนี้ ดังรายละเอียดในหัวข้อต่อไป

1.1 วัตถุประสงค์ของปริญญานิพนธ์

เทคนิคการลดขนาดข้อมูล เป็นวิธีการที่นำมาใช้ในการลดขนาดของหน่วยความจำที่ใช้ในการเก็บภาพ และยังสามารถที่จะลดเวลาที่ต้องสูญเสียไปในการส่งรับข้อมูลภาพเหล่านี้ไปยังปลายทางที่ไกลออกไป โดยผ่านทางช่องการสื่อสารความเร็วต่ำ(low speed communication channal) อย่างเช่น การส่งภาพอ่านทางสายโทรศัพท์ซึ่งเป็นระบบที่มีความเร็วต่ำ เมื่อทำการลดข้อมูลแล้วเวลาที่ใช้ในการรับส่งข้อมูลภาพแต่ละภาพก็จะน้อยลงไปด้วย เปรียบเสมือนว่าเป็นการเพิ่มอัตราบิทเรท(bit rate) ในการส่งข้อมูลภาพให้สูงขึ้นนั้นเอง

เทคนิคในการลดขนาดข้อมูลภาพที่ศึกษาในปริญญานิพนธ์ฉบับนี้ สามารถนำไปประยุกต์ใช้กับการสร้างฐานข้อมูลภาพ และโดยเฉพาะการรับส่งภาพในระยะไกลโดยผ่านโครงข่ายโทรศัพท์สาธารณะ ซึ่งโดยเฉพาะอย่างยิ่ง โทรศัพท์ภาพ (video phone) หรือการประชุมระยะไกลผ่านทางสาย(video conferencing) ได้

1.2 ขอบเขตของวิทยานิพนธ์

จุดมุ่งหมายของโครงงานนี้ เพื่อศึกษาและวิจัย แนวทางในการนำเอา เครื่องไมโครคอมพิวเตอร์ มาใช้ในงานประมวลผลภาพทางดิจิตอล โดยเฉพาะอย่างยิ่งในเรื่องของการลดขนาดข้อมูลภาพ ภาพ ขาว-ดำ 2 ระดับ ภาพ ขาว-ดำ 256 ระดับเทา และภาพสี แต่ในปริญญานิพนธ์นี้ เน้นศึกษาเฉพาะภาพขาว-ดำ 256 ระดับเทา และ ภาพสี สำหรับเนื้อหาของปริญญานิพนธ์ฉบับนี้แต่ละบทมีหัวข้อและเนื้อหา ดังต่อไปนี้

บทที่ 1 เป็นบทนำกล่าวถึงวัตถุประสงค์ และขอบเขตของวิทยานิพนธ์

บทที่ 2 กล่าวถึงหลักการประมวลผลภาพ

บทที่ 3 กล่าวถึงหลักการเบื้องต้นของการลดข้อมูลภาพ เทคนิคต่างๆที่นิยมใช้ในกระบวนการลดข้อมูลภาพ รวมถึงเกณฑ์การวัดความเหมือนจริงของภาพ

บทที่ 4 เป็นวิธีการลดขนาดข้อมูลภาพด้วยวิธี Block Truncation Compression แบบต่างๆ ที่ได้รับการพัฒนาขึ้นมา

บทที่ 5 เป็นการทดลองและผลการทดลองลดข้อมูลภาพ ด้วยเทคนิคต่างๆ กัน โดยใช้ไมโครคอมพิวเตอร์ การเปรียบเทียบข้อได้เปรียบและเสียเปรียบของเทคนิคต่างๆ

บทที่ 6 เป็นบทสรุปและวิจารณ์

บทที่ 2

หลักการประมวลผลภาพ

(Image processing)

ในการประมวลผลสัญญาณภาพด้วยระบบคอมพิวเตอร์ จำเป็นต้องเปลี่ยนข้อมูลหรือสัญญาณภาพที่อยู่ในรูปอนาลอก ให้เป็นสัญญาณทางดิจิตอล เพื่อประโยชน์ในการคำนวณและประมวลผลได้ง่าย ในบทนี้ จะกล่าวถึง ความหมายของภาพในระบบดิจิตอลและคณิตศาสตร์ที่เกี่ยวข้อง

2.0 ลักษณะข้อมูลภาพ

ซึ่งแบ่งตามการจัดเก็บข้อมูลได้เป็น

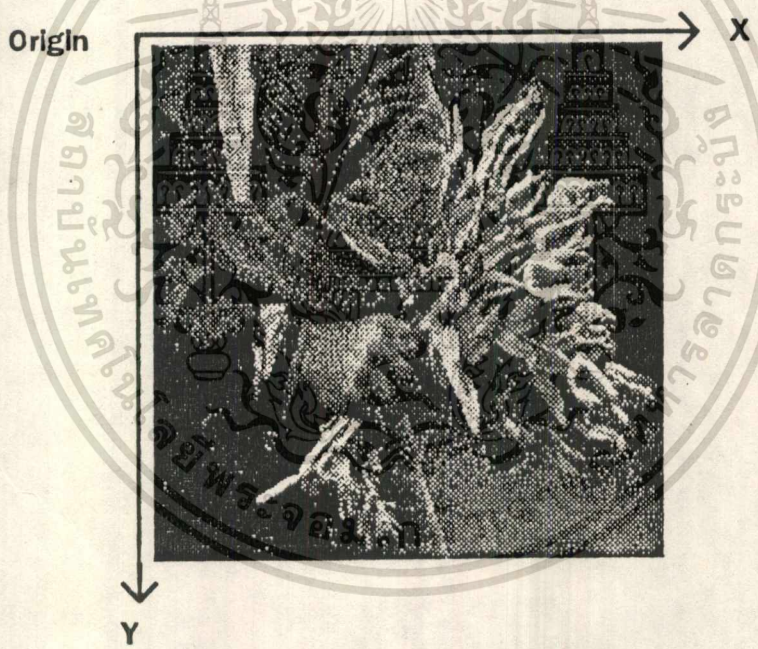
1. ภาพ 2 ระดับ คือ มีแค่จุดขาวกับดำเท่านั้น โดยแต่ละจุดเป็นข้อมูล 1 bit
2. ภาพ 16 ระดับ ซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูล 4 bit ซึ่งทำให้สามารถแสดงภาพได้ 16 ระดับสี หรือ 16 ระดับ Graylevel ขึ้นอยู่กับว่าภาพนั้นเป็น ภาพสี หรือภาพขาว-ดำ
3. ภาพ 256 ระดับ ซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูล 8 bit ซึ่งทำให้สามารถแสดงภาพได้ 256 ระดับสี หรือ ระดับ Graylevel ขึ้นอยู่กับว่าภาพนั้นเป็นภาพสี หรือ ภาพขาว-ดำ
4. ภาพ TRUE COLOR ซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูลขนาด 24 bit ทำให้สามารถแสดงผลภาพได้เหมือนภาพจริงที่สุด เพราะสามารถ แสดงสีได้ถึง 16,777,216 สี ภาพ True color สามารถแสดงได้เฉพาะภาพสีเท่านั้น ไม่สามารถแสดงภาพขาว-ดำ ได้

การแสดงภาพนี้ใช้วิธี ตั้งค่าของแม่สีในตารางสี โดยอาจเลือกสีเป็นแบบ 16 สี จาก 64 สี หรือ 16 สีจาก 262144 สี หรือ 256 สี จาก 262,144 สี ขึ้นอยู่กับmode การแสดงผล สำหรับ Ture Color ไม่มีการเลือก สี แสดงผลโดยการส่งค่าสี RGB ผ่าน D/A สีละ 8 bit ออกไปเลย ความแตกต่างของการแสดงภาพสีและภาพขาวดำคือ ภาพขาวดำจะต้องตั้งให้แม่สีทั้ง 3 สีมีค่าเท่ากัน เนื่องจาก VGA กำหนดให้แม่สีแต่ละสีใช้ register 6 bit ทำให้แต่ละแม่สีแสดงผลได้เพียง 34 ระดับเท่านั้น ยังผลให้เราแสดงภาพ 256 ระดับให้เห็นได้เพียง 64 ระ

ระดับเท่านั้น หากต้องการให้เห็นจริงทั้ง 256 ระดับต้องแสดงใน mode True color แล้วให้ RGB มีค่าเท่ากัน ซึ่ง mode นี้ใช้ register 8 bit สำหรับแม่สีแต่ละสี

2.1 ความหมายและนิยามของภาพในระบบดิจิตอล

ภาพ (Image) ในเชิงคณิตศาสตร์จะหมายถึง ฟังก์ชัน 2 มิติ f(x,y) โดย x และ y เป็นแกนพิกัดในระนาบ 2 มิติ ค่าฟังก์ชัน f(x,y) จะเป็นสัดส่วนกับความสว่างหรือความเข้มของภาพ ที่ตำแหน่ง (x,y) ซึ่งเราเรียกว่า ระดับสีเทา (Gray level) ในรูปที่ 2.1 แสดงให้เห็นถึงระนาบและจุดพิกัดของภาพ ซึ่งปกติเราจะให้จุดกำเนิดของแกนพิกัด (Coordinate) อยู่ทางมุมบนซ้ายของภาพ



รูป 2.1 ระนาบและพิกัดที่ใช้ในระบบภาพ

ภาพ 2 มิติที่แทนด้วยฟังก์ชัน f(x,y) โดย x และ y เป็นแกนในระนาบของภาพ ค่าของฟังก์ชันที่จุด (x,y) คือความเข้มของแสงที่จุดนั้น เนื่องจากแสงเป็นพลังงานรูปหนึ่ง ดังนั้น f(x,y) ต้องไม่เป็นศูนย์ และมีค่า (finite) นั่นคือ

$$0 < f(x,y) < \alpha \qquad ...(2.1.1)$$

โดยธรรมชาติของแสง ซึ่งจะต้องมีแหล่งกำเนิดแสงและส่วนที่สะท้อนของแสง ดังนั้นเราสามารถแยกฟังก์ชัน f(x,y) ออกเป็น 2 ส่วนคือ อิลูมินชั่นคอมโพเนนท์(illuminstion component) และ รีฟล็กแทนท์คอมโพเนนท์ (reflectant component) จะได้ว่า

$$f(x,y) = i(x,y) \times r(x,y) \qquad ...(2.1.2)$$

เมื่อ

$$0 < i(x,y) < \alpha \qquad ...(2.1.3)$$

และ

$$0 < r(x,y) < 1 \qquad ...(2.1.4)$$

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

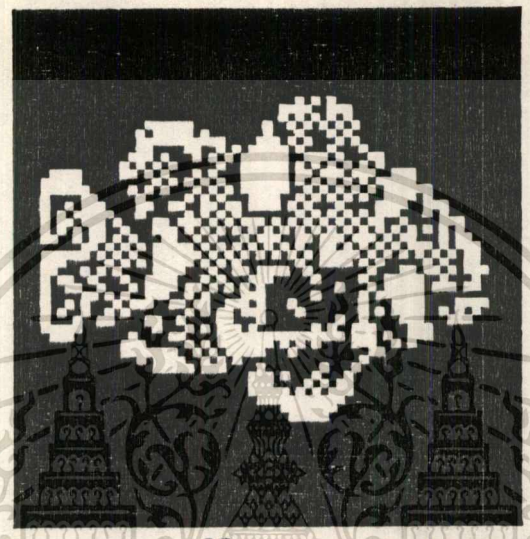
ดังที่กล่าวมาแล้ว ความเข้มของภาพที่จุด(x,y) เราเรียกว่า ระดับสีเทา(Gray level) l จากสมการที่ (2.2) ถึง (2.4) จะเห็นว่า l ควรอยู่ในช่วง

$$L_{min} \leq l \leq L_{max} \qquad ...(2.1.5)$$

ในทางทฤษฎี L_{min} ต้องมีค่าบวก ในขณะที่ L_{max} ต้องมีค่าน้อยกว่าอนันต์ ในทางปฏิบัติ $L_{min} = L_{min}\, r_{min}$ และ $L_{max} = L_{max}\, r_{max}$ ช่วงของ (L_{min}, L_{max}) เราเรียกว่าช่วงของระดับสีเทา ในทางปฏิบัติโดยใช้หลักคณิตศาสตร์ เรานิยมปรับช่วง (L_{min}, L_{max}) ให้เป็นช่วง (0,L) โดย L = 0 หมายถึง ดำสนิท และ L = 1 หมายถึง ขาว

2.2 การแทนภาพด้วยข้อมูลแบบดิจิตอล

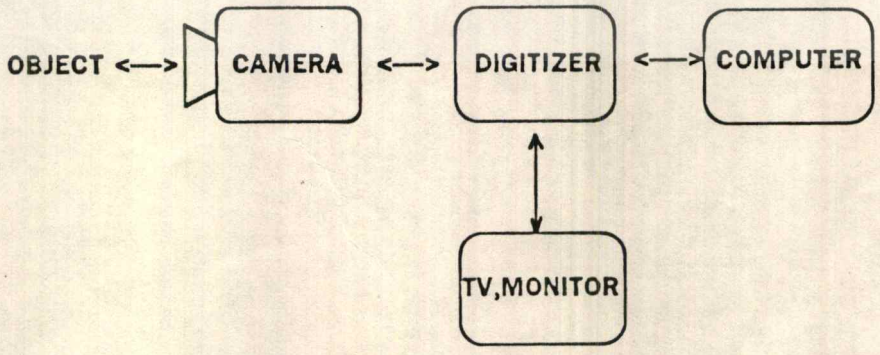
ภาพดิจิตอล (digital image) เป็นภาพที่ถูกแปลงมาจากภาพอนาลอก อยู่ในรูปตัวเลข โดยภาพอนาลอกถูกแบ่งเป็นพื้นที่สี่เหลี่ยมเล็กๆ ที่เรียกว่า Pixel ในแต่ละ Pixel จะถูกระบุตำแหน่งโดย (x,y) และค่าระดับสีเทาของ Pixel นั้น คือค่าของ f(x,y) รูป 2.2 เป็นภาพดิจิตอลขนาด 64 x 64 Pixel^2



รูป 2.2 แสดงภาพดิจิตอลขนาด 64 x 64 Pixel^2

2.3 ระบบการประมวลผลทางดิจิตอล

ระบบการประมวลผลภาพประกอบด้วย 3 ส่วนใหญ่ๆ คือ ส่วนเปลี่ยนสัญญาณอนาลอก ให้เป็นสัญญาณทางด้านดิจิตอล ซึ่งเรียกว่า ดิจิไตเซอร์(Digitizer) ส่วนประมวลผล (Processing) และส่วนแสดงผล (display) แสดงในรูป 2.3



รูปที่ 2.3 ระบบประมวลผลภาพดิจิตอล

จากรูปที่ 2.3 ส่วนแรกคือ ส่วนที่เปลี่ยนสัญญาณอนาลอก ให้เป็นสัญญาณดิจิตอล กล้อง (CAMERA) เปรียบเสมือนดวงตาของมนุษย์ ทำหน้าที่เปลี่ยนภภาพวัตถุมาเป็นสัญญาณทางไฟฟ้าและส่งให้ดิจิไตเซอร์(Digitizer) ซึ่งทำหน้าที่เปลี่ยนสัญญาณไฟฟ้าให้เป็นสัญญาณดิจิตอล อุปกรณ์ส่วนนี้ได้แก่ กล้องโทรทัศน์ดิจิไตเซอร์ ซึ่งภายในประกอบด้วย หลอด วิดิคอน ทำหน้าที่เป็นสื่อนำไฟฟ้าทางแสง ภาพถูกโฟกัสลงบนผิวของหลอด และถูกเปลี่ยนให้เป็นสัญญาณไฟฟ้าที่สอดคล้องกับความสว่างของภาพในตำแหน่งนั้นๆ จานนั้น ทำการควอนไตซิ่ง (quantizing) ข้อมูลภาพที่ได้เป็นสัญญาณดิจิตอล

ส่วนประมวลผลคือ คอมพิวเตอร์ ซึ่งเปรียบเสมือนสมอง ทำหน้าที่ประมวลผลและวิเคราะห์ข้อมูลภาพ

ส่วนแสดงผล ทำหน้าที่เปลี่ยนข้อมูลตัวเลข (ซึ่งเป็นระดับสีเทา) ที่เก็บเป็น array ในคอมพิวเตอร์ ให้อยู่ในรูปที่เหมาะสม และสื่อความหมายกับมนุษย์ได้ คือเป็นภาพที่ปกติทั่วๆไป อุปกรณ์ในส่วนนี้ได้แก่ monitor ทีวี เครื่องพิมพ์ที่สามารถแสดงผล ในรูปกราฟฟิกได้

ภาพ 1 ภาพ ที่ถูกเปลี่ยนจาก สัญญาณ ดิจิตอล สำหรับคอมพิวเตอร์นี้มีขนาดใหญ่ขึ้นอยู่กับความละเอียดของภาพที่ต้องการ และจะมีผลทำให้ใช้เนื้อที่ในหน่วยความจำมากในการเก็บข้อมูลภาพ 1 ภาพ เช่น การเก็บ ภาพ 1 ภาพ ขนาด 256 x 256 จุด^2 ที่มีความแตกต่างของระดับความเข้มของแต่ละจุด เท่ากับ 256 ระดับ จะต้องใช้เนื้อที่ในหน่วยความจำถึง 64 Kbyte ในการเก็บภาพนี้ดังนั้นในปัจจุบันนี้ได้มีการค้นคว้าวิจัย หาวิธีการที่จะเก็บภาพด้วยคอมพิวเตอร์ โดยให้ใช้เนื้อที่ในหน่วยความจำให้น้อยที่สุด และยังรักษาความละเอียดของภาพตามการใช้งานได้อีกด้วย

2.4 การสุ่มแบบสม่ำเสมอและควอนไตเซชั่น (Unifrom sampling and Quantization)

เพื่อที่จะประมวลสัญญาณภาพด้วยระบบคอมพิวเตอร์ ฟังก์ชันของภาพ f(x,y) จะถูกทำให้เป็นสัญญาณไม่ต่อเนื่อง ทั้งระนาบของภาพ ซึ่งเราเรียบกว่า การสุ่มภาพ (Image sampling) ของฟังก์ชันที่ได้เรียกว่า การควอนไตเซชั่นระดับสีเทา(gray level quantization)

สมมุติว่าสัญญาณภาพต่อเนื่อง f(x,y) ถูกดิจิไตซ์ ในระนาบ X Y เป็นช่วงเท่าๆกัน เราสามารถจัด f(x,y) ให้อยู่ในรูปเมตริกซ์ ขนาด N x N ได้ดังสมการ (2.4.1)

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,N-1) \end{bmatrix} \quad \dots(2.4.1)$$

ทางขวาของสมการ จะเรียกว่า ภาพดิจิตอล และทุกๆ สมาชิกของเมตริกซ์จะเรียกว่า พิกเซล จากขบวนการสร้างภาพดิจิตอลข้างต้น จะเห็นว่า เราต้องทราบขนาดความละเอียดของภาพ N x N พิกเซล และจำนวนระดับของ Gray level ในทางปฏิบัติการทำควอนไตเซชั่นในระบบภาพดิจิตอล จะเป็นค่าของ 2 ยกกำลังจำนวนเต็ม คือ

$$N = 2^n \quad \dots(2.4.2)$$

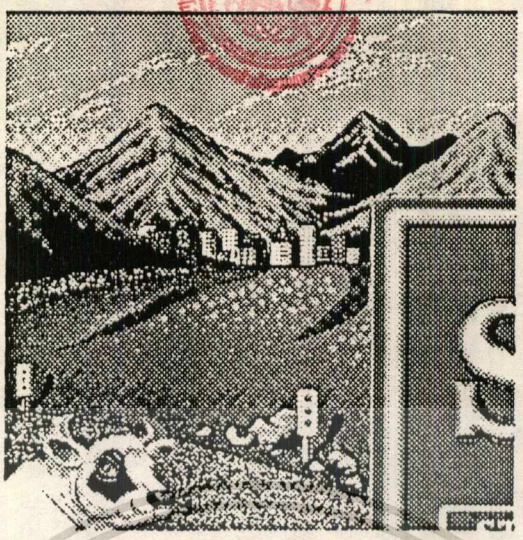
และ

$$G = 2^m \quad \dots(2.4.3)$$

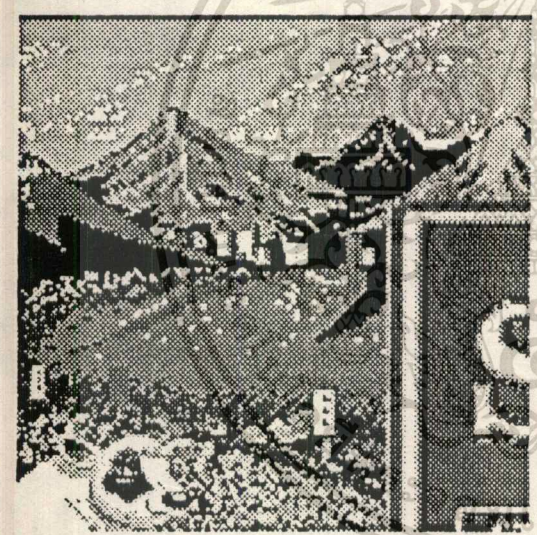
เมื่อ G คือ จำนวนระดับของ Gray level ดังนั้นจำนวนบิท(bit) ที่ใช้ในการเก็บภาพหนึ่งภาพที่ถูกดิจิไตซ์ คือ

$$B = N \times N \times m \text{ บิท} \quad \dots(2.4.4)$$

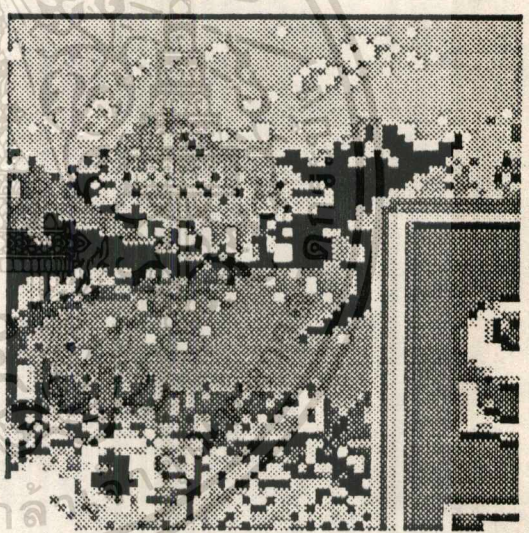
ดังตัวอย่างภาพขนาด 128 x 128 Pixel และระดับ Gray level จำนวน 256 ระดับ ต้องใช้หน่วยความจำขนาด 131,072 บิท ในรูปที่ 2.4 ได้แสดงการเปรียบเทียบภาพ เมื่อลดความละเอียดของภาพลง และตาราง 2.1 แสดงจำนวนไบท์ที่ใช้ในการเก็บภาพ เมื่อ N และ M เปลี่ยนไป



256 x 256



128 x 128



64 x64

รูป 2.4 เปรียบเทียบภาพเมื่อลดความละเอียดของภาพลง

ตาราง 2.1 จำนวน BYTE ที่ใช้ในการเก็บภาพ 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	32768	65536	131072	131072	262144	262144	262144	262144

2.5 เทคนิคต่าง ๆ สำหรับการประมวลผลภาพ

เทคนิคต่างๆ สำหรับการประมวลผลภาพ แบ่งเป็น 4 พวกใหญ่ๆ คือ

1. อิเมจดิจิไดเซชั่น (Image digitization)
2. อิเมจเอนฮานเมนต์และรีสตอเรชั่น (Image enhancement and restoration)
3. อิเมจรีคอมสตรัคชั่น(image reconstruction)

2.5.1 อิเมจดิจิไดเซชั่น (Image digitization)

ดังได้กล่าวมาแล้วถึงความหมายของการ digitize ภาพ ซึ่งความละเอียดของภาพที่ได้ก็ขึ้นอยู่กับการจัดระดับภาพ ในปัจจุบันเครื่องมือที่ใช้ทำขบวนการนี้ที่เรียกว่า ดิจิไดเซอร์ (digitizer) ดิจิไดเซอร์สามารถเปลี่ยนสัญญาณอนาลอกเป็นสัญญาณดิจิตอลได้ ดังนั้นข้อมูลที่ได้จึงเก็บเป็นเลขไบนารี่ โดยใช้ดิจิไดเซอร์เป็นตัวจัดการ

2.5.2 อิเมจเอนฮานเมนต์และรีสตอเรชั่น (Image enhancement and retoration)

อิเมจเอนฮานเมนต์เป็นการทำภาพให้อยู่ในรูปที่เหมาะสมขึ้น มีความคมชัดมากยิ่งขึ้น สำหรับการนำไปใช้งานเฉพาะอย่าง กล่าวคือ วิธีที่ทำภาพ หรือปรับปรุงภาพ X-ray อาจจะไม่เป็นวิธีที่ดี เมื่อนำมาปรับปรุงภาพถ่ายดาวเคราะห์ที่ส่งมาจากการสำรวจทางอวกาศ

วิธีปรับปรุงคุณภาพของภาพ(enhancement) มีหลายวิธี ดังนี้

1. คอนทราสท์เอนฮานเมนท์(Contrast enhancdment) เป็นวิธีที่ทำให้ภาพคมชัดขึ้น โดยอาศัยฮีสโตแกรม อาจใช้แบบลิเนียร์สเทรท(linear stretch), พีซลิเนียร์สเทรท(piecewise linear stretch) หรือ อีควอไลเซชั่น(equalization)

2. เอดจ์เอนฮาดเมนท์(Edge enhancdment) เป็นการแยกความแตกต่างของจุดภาพที่ใกล้เคียงกัน เพื่อหาขอบเขตของภาพ

3. การประมวลผลภาพสีเทียม(Pseudo-color imane processing) เป็นการใช้เทคนิคของการทำ density slicing และการใส่สีเทียมให้กับภาพขาว-ดำ ที่มีระดับ Gray level ต่างๆกัน

4. การกรองภาพ (Filtering) เพื่อให้ภาพเรียบ(smoothing) หรือ คมชัด (sharpening) โดยใช้ตัวกรองความถี่ต่ำ (low pass filter) หรือ ตัวกรองความถี่สูง(high pass filter)ตามลำดับ

อิเมจรีสตอเรชั่น (Image restoration)

เป็นขบวนการในการสร้างภาพกลับคืน โดยการหาค่า ชดเชย และแก้ความคลาดเคลื่อน เนื่องมาจากข้อมูลในภาพผิดพลาดไป หรือเป็นขบวนการสร้างภาพกลับคืน จากภาพที่ถูกทำให้เสียไป เนื่องจากปรากฏการณ์ต่างๆ โดยใช้หลักการของพีชคณิตเชิงเส้น (linear algebra)

2.5.3 อิเมจเอนโค๊ดดิ้ง (Image Encoding)

เป็นการใช้เทคนิคต่างๆ เพื่อเข้ารหัสข้อมูล เนื่องจากข้อมูลภาพที่ได้จะถูกเก็บในลักษณะเป็นจำนวนไบท์ ดังตาราง 2.1 ซึ่งถ้าภาพมีขนาดใหญ่ ก็ต้องใช้พื้นที่ในการเก็บมาก ด้วยข้อจำกัดของเครื่องไมโครคอมพิวเตอร์ ที่มีขนาดหน่วยความจำจำกัด

การเข้ารหัสข้อมูลจึงมีประโยชน์ ในด้านการลดพื้นที่ในการเก็บข้อมูลภาพดังกล่าวมาก ผลของการเข้ารหัสข้อมูลนี้ เรียกว่าเป็นการลดข้อมูล(Data reduction หรือ Data compression) ซึ่งเป็นเนื้อหาที่ทำในโครงงานนี้ รายละเอียดของการลดข้อมูลภาพ ได้อธิบายในบทที่ 3 นอกจากนี้ การเข้ารหัสข้อมูลยังมีประโยชน์ในการช่วยลดปริมาณข้อมูล

ภาพ ที่ใช้ในระบบสื่อสาร เช่น การส่งภาพถ่ายจากอวกาศมายังโลก การส่งข้อมูลภาพผ่านโมเด็ม(modem) เป็นต้น

2.5.4 อิเมจรีคอนสตรัคชั่น (Image reconstruction)

เป็นวิธีการสร้างภาพตัดขวางของวัตถุ โดยไม่ต้องผ่า หรือทำลายวัสดุ เพื่อประมวลผลโดยใช้คอมพิวเตอร์ เราเรียกการสร้างภาพตัดขวางด้วยคอมพิวเตอร์ว่า คอมพิวเตอร์โทโมกราฟฟี่(Computer tomography)

บทที่ 3

การลดข้อมูลภาพ

(DATA REDUCTION OR DATA COMPRESSION)

สัญญาณภาพโดยธรรมชาติแล้วจะเป็นสัญญาณต่อเนื่อง (Analog) ที่มีความกว้างของย่านความถี่สูงมาก ซึ่งเมื่อเทียบกับสัญญาณเสียงแล้วความกว้างของย่านความถี่สูงกว่าของสัญญาณเสียงเป็น 1000 เท่าหรือ มากกว่า การนำข้อมูลภาพมาใช้ในการประมวลผลในด้านต่างๆ ที่มีเครื่องคอมพิวเตอร์เป็นตัวประมวลผลแล้วนั้น สัญญาณภาพจะต้องแปลงให้เป็นสัญญาณที่เครื่องสามารถเข้าใจได้เสียก่อนนั้นคือต้องแปลงให้อยู่ในรูปของสัญญาณดิจิตอล ซึ่งอยู่ในลักษณะที่ไม่ต่อเนื่อง(discrete signal) โดยผ่านการสุ่มและจัดระดับ(sampled and quantised) เมื่อรายละเอียดของภาพๆหนึ่ง ถูกแทนที่ด้วยข้อมูลแบบดิจิตอล ขนาดของข้อมูลภาพจึงมีจำนวนสูงมากเพื่อที่จะสามารถเก็บรายละเอียดของภาพได้เพียงพอ ทำให้ต้องใช้ขนาดของหน่วยความจำในการเก็บข้อมูลเหล่านี้มีขนาดใหญ่ตามไปด้วย ในกรณีที่มีการรับส่งข้อมูลภาพเหล่านี้ด้วยแล้วจะต้องเสียเวลาในการรับส่งเป็นอันมาก โดยเฉพาะอย่างยิ่งหากเป็นการสื่อสารข้อมูลผ่านดาวเทียมแล้ว เวลาที่มากนั้นก็หมายถึงค่าใช้จ่ายที่สูงมากตามไปด้วย ดังนั้นจึงจำเป็นอย่างยิ่งที่จะต้องลดเวลาในการส่งข้อมูลให้น้อยที่สุดเท่าที่จะเป็นไปได้

การลดขนาดของข้อมูลภาพ จึงถูกนำมาใช้ในการแก้ปัญหาเหล่านี้ เพื่อเป็นการประหยัดทางด้านเศรษฐกิจในส่วนของจำนวนหน่วยความจำที่ลดลง และในส่วนของเวลาที่ใช้ไปในการรับส่งข้อมูลภาพ สิ่งที่เป็นตัวกำหนดว่าข้อมูลของภาพสามารถที่จะลดลงไปได้เท่าไรนั้นขึ้นอยู่กับงานที่ใช้ว่าต้องการรายละเอียดมากน้อยเพียงใด เมื่อทำการลดข้อมูลแล้วจึงจะสามารถนำข้อมูลเดิมกลับมา(reconstruct) ได้อย่างเหมาะสม ซึ่งงานแต่ละอย่างมีความต้องการรายละเอียดของภาพที่ไม่เท่ากัน อย่างเช่นถ้านำไปใช้ในการตรวจสอบและจดจำรูปแบบของวัตถุต่างๆ โดยใช้คอมพิวเตอร์แล้ว ภาพที่ใช้จะเน้นเฉพาะส่วนของขอบวัตถุในภาพนั้นๆ ก็เพียงพอ โดยไม่จำเป็นต้องมีรายละเอียดโครงสร้างภายในของภาพ ในขณะที่การลดข้อมูลภาพเพื่อการลดเวลาที่ใช้ไปในการรับส่งภาพผ่านช่องสัญญาณความเร็วต่ำอย่างเช่น ระบบโครงข่ายโทรศัพท์สาธารณะนั้น มีความจำเป็นอย่างยิ่งที่จะต้องเก็บรายละเอียดต่างๆ ของภาพให้ได้มากที่สุด หลักการพื้นฐานที่นิยมใช้ในการลอข้อมูลภาพมีอยู่สามหลักการด้วยกัน คือ

Predictive Coding ซึ่งเป็นการเข้ารหัสใน data domain หลักการที่สอง คือ Transform Coding เป็นวิธีที่ประมวลผล(process)ในโดเมนของความถี่ ส่วนหลักการสุดท้ายเป็นการรวมเอาข้อดีต่างๆ จากสองหลักการแรกเข้าด้วยกัน เรียกว่า Hybrid coding

3.1 Predictive Coding

การลดขนาดข้อมูลภาพด้วยวิธี Predictive Coding นี้ เป็นการอาศัยคุณสมบัติของข้อมูลภาพที่มักจะมีค่าซ้ำๆกัน และเมื่อข้อมูลอินพุทถูกกำหนดให้มีความเกี่ยวพันธ์กันนั้นก็คือจุดภาพที่มีตำแหน่งอยู่ใกล้ๆกัน มักจะมีค่าระดับแอมปลิจูดใกล้เคียงกันหรือเท่ากัน ดังนั้นจึงสามารถที่จะใช้ค่าของจุดภาพหนึ่งจุดหรือหลายๆจุด ที่ผ่านมาใน Line นั้น Lineก่อนหน้านั้นหรือในเฟรมที่ผ่านมา เป็นตัวคาดคะเนหรือแทนค่าของจุดภาพปัจจุบัน ซึ่งโดยธรรมชาติทางสถิติของข้อมูลภาพเราสามารถที่จะคาดคะเนค่าของข้อมูลได้ไม่ผิดพลาดมากนัก จากค่าที่ได้จากการคาดคะเนนี้เอาไปลบกับค่าจริงของจุดภาพ จะได้เป็นค่าความแตกต่างระหว่างค่าจริงกับค่าที่เราคาดคะเนเอาไว้ ซึ่งค่านี้จะมีขนาดเล็ก และค่าผลต่างนี้จะถูกนำไปเข้ารหัสเพื่อจะเก็บไว้ใช้ในตอนถอดรหัส พร้อมกับค่าที่เราคาดคะเนเอาไว้ ดังนั้นในการที่จะเก็บในหน่วยความจำหรือต้องการส่งก็ใช้ค่าสองค่านี้ เมื่อถึงตอนที่จะนำภาพเดิมกลับมาหรือตอนถอดรหัส จะนำเอาค่าที่เราคาดคะเนไว้ในตอนแรกบวกกับค่าของผลต่างของจุดภาพนั้นกับค่าคาดคะเน ก็จะได้เป็นค่าของจุดภาพนั้นๆ ค่าผิดพลาดที่ได้ในตอนถอดรหัสเกิดขึ้นเพียงกรณีเดียวเท่านั้นคือตอนจัดระดับ (quantised) ค่าความแตกต่างของการเข้ารหัสเท่านั้น วิธีนี้เป็นวิธีที่ง่ายแก่การสร้างระบบ และสามารถลดขนาดข้อภาพให้หลือประมาณ 1-2 bit/element

3.2 Transform Coding

การลดข้อมูลภาพด้วยวิธีของ Transform Coding นี้เป็นวิธีที่ซับซ้อน และมีขั้นตอนมากกว่าวิธีการลดข้อมูลภาพโดย Predictive Coding หลักการของวิธี Transform Coding จะทำการแปลงข้อมูลอินพุทที่อยู่ในรูปของ data domain ให้อยู่ในรูปของ spectral หรือ frequency domain โดยใช้วิธีการ Transform แบบต่างๆ เช่น Fourier Transform จะเป็นการแปลงข้อมูลที่อยู่ในรูปของ spatial domain ให้อยู่ในรูปสัมประสิทธิ์ของพลังงานความถี่ โดยค่าความถี่ต่ำๆ จะมีพลังงานสูง ที่ความถี่สูงพลังงานจะลดลงไป การเข้ารหัสจึงใช้จำนวนบิทสำหรับแต่ละช่วงความถี่ไม่เท่ากัน เมื่อต้องการอัตราบิทเรทสูงๆ ค่าของพลังงานความถี่สูงจะถูกตัดทิ้งไปเป็นส่วนใหญ่ เป็นเหตุให้รายละเอียดส่วนที่เป็นขอบภาพขาดหายไป ทำให้ภาพที่ได้เบลอ ขาดความคมชัด

การ Transform ที่นิยมใช้ในการลดข้อมูลภาพมีอยู่ด้วยกันหลายวิธี เช่น Fast Fourier Transform, Fast Walsh-Hadamard Transform, Fast Slant Transform, Fast Discrete Cosing Transform, Fast Discrete Sine Transform เป็นต้น ซึ่งแต่ละวิธีก็มีข้อดีข้อเสียที่แตกต่างกันไป การ Transform ที่นิยมใช้กันมากได้แก่ Discrete Cosine Transform เพราะเป็นวิธีที่สามารถคำนวณได้ง่ายเนื่องจากมีการคำนวณเฉพาะค่าจริงไม่ค่าจินตภาพ(Imaginary)

วิธี Transform Coding ถึงแม้ว่าจะเป็นวิธีที่ยุ่งยากแต่ก็สามารถสร้างระบบได้ด้วยอุปกรณ์ทางฮาร์ดแวร์ (Hardware)ดิจิตอลความเร็วสูงได้ง่าย และเป็นระบบที่ยืดหยุ่น (adaptive system) สามารถที่จะลดขนาดของข้อมูลให้อยู่ในอัตราบิทเรทประมาณ 0.5-1 บิทต่อจุดภาพ ซึ่งเป็นช่วงอัตราการลดที่สามารถสร้างภาพเดิมกลับมาได้สมบูรณ์เพียงพอต่อการนำไปใช้งานต่าง ๆ

3.3 Hybrid Coding

Hybrid Codint เป็นเทคนิคที่นำข้อดีของวิธี Predictive Coding และ Transform Coding มาใช้ร่วมกัน ทั้งนี้เพราะบางครั้งการลดขนาดข้อมูลภาพด้วย Transform Coding ไม่อาจจะให้รายละเอียดของภาพเพียงพอ ดังนั้นการเพิ่มรายละเอียดของภาพจึงต้องใช้ความสัมพันธ์ของจุดภาพที่มีอยู่ทั้งทางแนวนอนและทางแนวตั้ง โดยการใช้การ Transform แบบมิติเดียวในทิศทางแกนใดแกนหนึ่งของภาพ เช่น ใช้การ Transform กับข้อมูลภาพในทางแนวนอนต่อจากนั้นจึงใช้ Predictive Coding กับสัมประสิทธ์ของการ Transform ที่ได้ เพื่อที่จะใช้ในการคาดคะเนค่าของกลุ่มสัมประสิทธ์ที่เหมือนๆกัน ที่ตามมาในแนวนอน ในกรณีนี้ความสัมพันธ์ของข้อมูลในทางแนวตั้งจะเป็นตัวกำหนดผลที่ได้ว่าถูกต้องมากน้อยเพียงใด แต่โดยเฉลี่ยแล้วค่าสัมประสิทธ์ที่ตามมามักจะมีค่าที่เหมือนๆ กัน ซึ่งเหมือนกับการทำ Predictive ใน data domain นั้นเอง และในกรณีของการ Transform แบบสองมิติสามารถทำได้เร็วขึ้นอาจจะใช้การ Predictive กับค่าสัมประสิทธ์ของการ Transform ของภาพบริเวณเดียวกันแต่เป็นเฟรมที่แตกต่างกันที่ตามมา การใช้ Hybrid Coding ในลักษณะนี้สามารถเป็นไปได้อีกลักษณะหนึ่งคือ ใช้การ Predictive ในขั้นตอนแรกก่อนแล้วจึงทำการ Transform แต่ในกรณีนี้ต้องการระบบที่ซับซ้อนมากกว่า

การลดข้อมูลภาพด้วยวิธีของ Hybrid Coding นี้ อาจจะพูดได้ว่ามีคุณสมบัติอยู่ระหว่างวิธีของ Predictive Coding และ Transform Coding ดังนั้นอัตราบิทเรทต่ำสุดของวิธีนี้จึงไม่สามารถทำให้ต่ำกว่าวิธีของ Transform Coding เพียงแต่สามารถสร้างได้ง่ายกว่า โดย

อัตราบิทเรทของระบบจะมีค่าประมาณ 1 บิทต่อจุดภาพ ซึ่งเป็นอัตราบิทเรทที่สามารถสร้างภาพเดิมกลับมาใหม่ (reconstruction) ได้ภาพที่มีคุณภาพดีพอสมควร

3.4 การลดข้อมูลภาพด้วยวิธีรันเลงจ์ (Runlength Compression)

การลดข้อมูลด้วยวิธีนี้ อาศัยลักษณะทั่วไปของข้อมูลภาพที่จะต้องมีส่วนของฉากหลัง (Blackground) และ พื้นหน้า (foreground) ในส่วนของ ฉากหลังจะมีรายละเอียดของภาพไม่มากนัก ส่วนนี้เองจะมีการเปลี่ยนแปลงของข้อมูลน้อย เมื่อเทียบกับส่วนของ foreground ซึ่งมีรายละเอียดและการเปลี่ยนแปลงของข้อมูลมาก ในส่วนที่มีการเปลี่ยนแปลงข้อมูลน้อยนี้เองที่เราสามารถนำการเข้ารหัสแบบ Runlength มาประยุกต์ใช้ได้อย่างมีประสิทธิภาพ

การเข้ารหัสแบบนี้ จะจัดข้อมูลภาพเดิม ให้อยู่ในรูปของคู่ลำดับ (G_i,L_i) โดย G_i แทนระดับความเข้ม หรือ Graylevel L_i แทนความยาวของข้อมูลหรือ จำนวนจุดที่มีระดับ Graylevel G_i การเข้ารหัสแบบนี้มีด้วยกัน 2 วิธีใหญ่คือ

วิธีที่ 1 จะทำการอ่านข้อมูลเข้ามาโดยนับจำนวนข้อมูลที่ซ้ำกันกับข้อมูลนั้นเข้ามาด้วย แล้วแปลงข้อมูลไป เป็น 2 Byte คือ ใน ไบท์แรก จะเก็บจำนวนตัวข้อมูลที่ซ้ำกัน โดยจะมีไบท์ที่สอง เก็บค่าข้องระดับสีที่ซ้ำกันนั้นเอาไว้ โดยใน 1 ชุดข้อมูลเอาท์พุท(2 Byte)จะนับจำนวน จุดที่ซ้ำกันได้ 256 จุดสี ตัวอย่างการเข้ารหัสข้อมูลเช่น

ข้อมูลเป็น(ค่าที่แสดงเป็นเลขฐาน 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

จะเห็นได้ว่าเราสามารถ ลดข้อมูลจาก 19 Byte เหลือ 12 Byte ซึ่งหากข้อมูลซ้ำกันถึง 256 จุด แล้วจะทำให้ เราลดข้อมูลลงไปได้อย่างมหาศาล แต่ในขณะเดียวกันหากเกิดกรณีที่แย่ที่สุดของการใช้วิธีนี้ก็คือ กรณีที่ข้อมูลแต่ละตัวไม่ซ้ำกับจุดข้างเคียงเลย ซึ่งจะทำให้ผลของการเข้ารหัส จะ ได้รหัสที่ยาวเป็น 2 เท่าของข้อมูล อินพุท

วิธีที่ 2 จากความบกพร่องของวิธีการทำ RunLength Encoding วิธีแรก ตรงที่โปรแกรมจะทำการเข้ารหัสข้อมูล เมื่อนับจำนวนข้อมูลได้ตั้งแต่ 1-255 จุด ดังนั้นถ้าเกิดข้อมูลของเรามี จุดสีที่อยู่โดดๆ(ไม่ซ้ำกับจุดข้างเคียง) อยู่มากมายจะทำให้การเข้ารหัสไม่ทำให้ข้อมูลเล็กลงมากนัก หรืออาจจะใหญ่กว่า Origin ด้วยซ้ำไป และ Pixel ที่ซ้ำกันทีเดียวถึง 255 ตัว

คงไม่เกิดขึ้นบ่อยนัก จึงได้มีแนวความคิดที่จะทำการแก้ไขปัญหาของการทำ Runlength วิธีแรก เกี่ยวกับการมีข้อมูลที่ไม่ซ้ำกันกับตัวข้างเคียง โดยมีหลักการอยู่ 2 ข้อดังนี้

1. จะไม่ทำการลดข้อมูลกับส่วนที่มีจำนวนซ้ำกันน้อยกว่า 3 ตัว
2. ให้มีการทำเครื่องหมายเพื่อที่จะแยกส่วนที่มีการลดข้อมูลกับส่วนที่ไม่มีการลดข้อมูลออกจากกัน โดยใช้บิทๆหนึ่งเป็นตัวบอกว่าถูกลดขนาดหรือไม่

จากหลักการดังกล่าวนี้ จะใช้บิทที่ 7 ของไบท์บอกขนาดเป็นตัวบอกว่าข้อมูลที่ต่อจากนี้ไปมีการลดขนาดข้อมูลหรือไม่ ส่วนบิทที่เหลือเราก็ยังคงใช้บอกขนาดต่อไป เมื่อเรานับ pixel ได้ซ้ำกันตั้งแต่ 3 ตัวขึ้นไป บิทที่ 7 ของ ไบท์บอกขนาดจะถูกเซท ให้เป็น 1 หรือและเราจะเริ่มนับ 1 ตั้งแต่Pixelที่ซ้ำกันตั้งแต่ตัวที่ 4 เป็นต้นไป ทำให้บิทบอกขนาดมีค่าได้ตั้งแต่ 3 - 130 หรือใน 2 Byte นี้เราอาจเก็บข้อมูลได้ถึง 130 byte ส่วนในกรณีที่ไม่การลดขนาด บิทที่ 7 ของ Byte บอกขนาดก็จะถูก set ให้เป็น 0 และ 7 bit ที่เหลือจะบอกจำนวนของข้อมูลที่มีค่าไม่ซ้ำกันนั้นโดยค่าที่ตามมาจะเป็นข้อมูลที่ไม่ถูกลดขนาด เป็นจำนวน Byte เท่ากับจำนวนที่แสดงไว้ใน ไบท์บอกขนาด ซึ่งจะมีค่าระหว่าง 0 ถึง 127 โดยจำนวนจริงจะมีค่าเท่ากับ ไบท์บอกขนาดบวกหนึ่งตัวอย่างเช่น

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 6 ไบท์มีค่า) 15

: 03(ไม่ถูกลดขนาด 4 ไบท์มีค่า) 45 45 78 20

: 81(ถูกลดขนาดลง 4 ไบท์มีค่า) 10

โดยวิธีนี้ผลลัพธ์ที่ได้จะแย่ที่สุดในกรณีที่เกิดมีข้อมูลที่ไม่ซ้ำกันเลยมากกว่า 128 ไบท์ เมื่อทำการเข้า รหัสจะได้รหัสถึง 129 ไบท์สำหรับทุกๆ 128 ไบท์ของข้อมูลอินพุท และในกรณีที่ดีที่สุดก็คือเมื่อข้อมูลซ้ำกันถึง 128 ไบท์ เมื่อเข้ารหัสแล้วจะได้ข้อมูลที่ลดลงเหลือเพียง 2 ไบท์

ความสามารถในการลดข้อมูลภาพด้วย Runlength

ในกรณีที่ข้อมูลภาพ มีความแตกต่างของระดับเทามากอัตราของข้อมูลต่อ 1 จุดภาพ (Data Bite Rate)จะมากตามไปด้วย เนื่องจากมีความเป็นไปได้ที่ข้อมูลจะมีการเปลี่ยนแปลงสูง

การลดข้อมูลด้วย Runlength นี้จะไม่มี ความคลาดเคลื่อนเกิดขึ้น หลังจากการถอดรหัส การใช้วิธีนี้ต้องพิจารณาถึง ลักษณะของภาพต้นแบบว่ามีความเหมาะสมหรือไม่ ควรมีรายละเอียดของภาพน้อย เช่น ภาพเอกสาร ภาพทางด้านกราฟจากตาราง เราสามารถลดข้อมูลภาพได้ ดียิ่งขึ้นได้ โดยการปรับข้อมูลให้มี ความแตกต่างของระดับเทาให้น้อยลง ด้วยวิธีของ Gray scale Transformation

3.5 Discrete Cosine Transform coding

Discrete Cosine Transform Coding เป็นวิธีการที่ใช้ในการลดข้อมูลภาพที่ได้รับความนิยมอย่างแพร่หลาย ทั้งนี้เพราะค่าสัมประสิทธิ์ในโดเมนของความถี่ที่ได้จะเป็นเทอมของค่าจริง(real time) เท่านั้น อีกทั้งยังสามารถคำนวณได้แบบรวดเร็ว(fast algorithms) และยังสามารถใช้งานจริงในลักษณะ real time โดยใช้ฮาร์ดแวร์ได้ไม่ยาก ในปัจจุบันหลักการของ Discrete Cosine Transform ยังคงมีการวิจัยกันอยู่ต่อไปเพื่อให้สามารถลดขนาดของข้อมูลให้ได้มากที่สุดพร้อมทั้งหาวิธีที่จะเพิ่มความเร็วในการคำนวณให้ได้เร็วขึ้นไปอีก

ดังนั้นในที่นี้จึงยกตัวอย่าง ระบบการลดข้อมูลภาพโดยใช้เทคนิคของ Discrete Cosine Transform Coding โดยมีโครงสร้างของระบบเป็นดังรูปที่ 3.1 จาก block diagram ของการ Transform Coder ภาพอินพุทที่เข้ามาจะถูกแยกออกเป็นบล็อคเล็กๆ โดยเราสามารถกำหนดขนาดของบล็อคได้ว่าให้เป็นเท่าไร ขนาดของบล็อคที่เหมาะสมจะเป็นตัวเพิ่มประสิทธิภาพในการรวมพลังงานของการ Transform ซึ่งจะทำให้ภาพที่ได้มีรายละเอียดที่ดี ความเหมาะสมของขนาดของบล็อคค่าหนึ่งๆ จะเหมาะสมที่ค่าบิทเรทค่าหนึ่งๆ แต่อย่างไรก็ตามขนาดของบล็อคที่เหมาะสมสามารถคำนวณได้ด้วยคณิตศาสตร์ที่ยุ่งยากพอๆ กับการ Transform เลยทีเดียว โดยส่วนมากแล้วขนาดของบล็อคจะมีค่าเป็นเลขยกกำลังของสองอย่าง เช่น 4, 8, 16 ซึ่งได้มาจาก 2^2, 2^3, 2^4 เป็นต้น หลังจากการแยกข้อมูลออกเป็น

บล็อคย่อยๆ แล้ว ก็จะทำการ Transform ไปโดยอิสระของแต่ละบล็อค โดยมีสมการของการ Transform มิติเดียว และ สองมิติ ดังต่อไปนี้เป็น

$$F(k) = \sqrt{(2/N)}\ \alpha(k) \sum_{n=0}^{N-1} f(n) \cos[(2n+1)\pi k/2N] \quad \text{เมื่อ } k = 0,1,\ldots,N-1 \quad \ldots(3.7)$$

เมื่อ

$$\alpha(0) = \sqrt{(1/2)} \qquad \text{และ} \qquad \alpha(k) = 1 \quad \text{เมื่อ } k \text{ ไม่เท่ากับ } 0 \quad \ldots(3.8)$$

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

ส่วนสมการของการ Transform แบบสองมิติสามารถเขียนได้ดังนี้คือ

$$F(u,v) = \frac{2}{N}\ \alpha(u)\alpha(v) \sum_{m=0}^{N-1} \sum_{m=0}^{N-1} f(m,n) \cos\left[\frac{(2m+1)u\pi}{2N}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right] \quad \ldots(3.9)$$

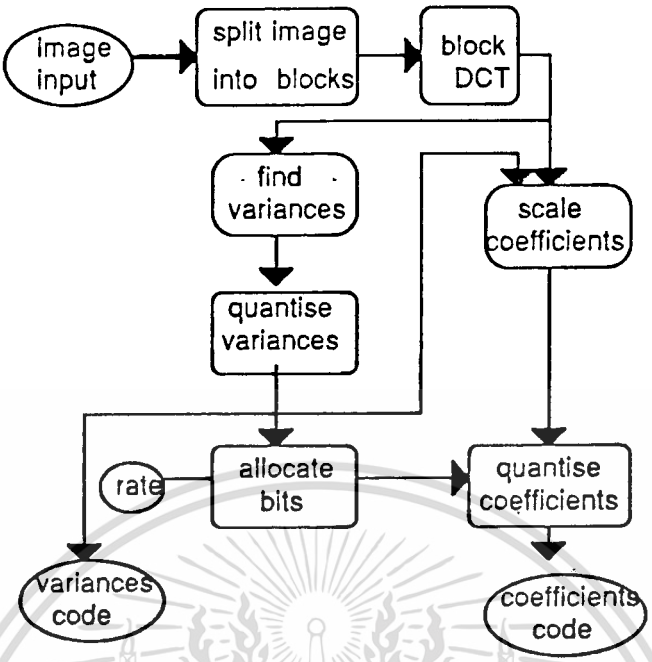
เมื่อ

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

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

F(u,v) เป็นผลจากการทำ discrete cosine Transform

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



รูปที่ 3.1 บล็อคไดอะแกรม ตอน Forward Transform ของ DCT

หลังจากผ่านการ Transform โดยใช้ Discrete Cosine Transform เพื่อเปลี่ยนข้อมูลให้อยู่ในรูปของ frequency domain พลังงานรวมส่วนใหญ่ของข้อมูลจะถูกเก็บ(pack)อยู่ในสัมประสิทธิ์เพียงไม่กี่ตัวของการ Transform จากสัมประสิทธิ์ที่ได้จะถูกนำมาหาค่า variances ของแต่ละบล็อค เพื่อที่จะนำค่า standard deviation ไปคำนวณหา bit allocate สำหรับจัดให้กับสัมประสิทธิ์แต่ละตัวในบล็อค และนอกจากนั้นยังส่งค่า standard deviation ไปยังทางด้านรับด้วยเพื่อใช้ในตอน decode ดังนั้นจึงต้องทำการจัดระดับ(quantise) ค่านี้ก่อนที่จะเก็บหรือส่งไปยังด้านรับ และค่าสัมประสิทธิตัวอื่นๆ ก็จะถูกปรับระดับ (normalise) ด้วยค่าของ variances ที่ได้หลังจากการกำหนด bit allocate เพื่อปรับค่าของสัมประสิทธิ์ให้อยู่ในช่วงของค่าบิทเรทที่กำหนด ก่อนที่จะทำการ quantise ส่วนการกำหนดค่า bit allocate ของสัมประสิทธิ์ที่ตำแหน่ง (u,v) หลังการ Transform ในแต่ละบล็อคคือ

$$b_{uv} = R+0.5 \log_2 \{\sigma_{uv}^2/[\prod_{k=0}^{n-1}\prod_{k=0}^{n-1} \sigma_{kl}^2]^{1/N}\}$$

$$= \log_2(\sigma_{uv}) + R - (1/N)\log_2[\prod_{k=0}^{n-1}\prod_{k=0}^{n-1} \sigma_{kl}]$$

$$= \log_2(\sigma_{uv}) + \beta \qquad \ldots(3.10)$$

เมื่อ σ_{uv} คือค่า standard deviation ของสัมประสิทธิ์ของการ Transform ที่ความถี่ (u,v) และ σ_{kl} เป็นค่า standard deviation ของสัมประสิทธิ์ ของการ Transform ที่ตำแหน่ง kl โดย k และ l มีค่าเป็น 0 ถึง n-1 เมื่อ n คือขนาดของบล็อค และ N คือจำนวนของสัมประสิทธิ์ในแต่ละบล็อค โดยค่าคงที่ β กำหนดได้จาก

$$\beta = R - (1/N)\log_2[\prod_{k=0}^{n-1}\prod_{k=0}^{n-1}\sigma_{kl}]$$

และค่าสูงสุดของบิทที่จะกำหนดให้กับสัมประสิทธิ์แต่ละตัวจะกำหนดให้อยู่ภายใต้ค่า b_{max} $0 \leq b_{uv} \leq b_{max}$ และค่า b_{uv} เป็นค่าจำนวนเต็ม

โดยที่ $\sum_u\sum_v b_{uv} = B$ และ $B = R$ เมื่อ R คือ ค่าบิทเรทที่ต้องการ

หลังจากการคำนวณ bit allocation ค่าสัมประสิทธิ์ของการ Transform แต่ละตัวในบล็อค จะถูกปรับค่า (normalise) ให้อยู่ในช่วงขนาดของของค่าไม่เกินจำนวนบิทที่กำหนด แล้วจึงทำการ quantise ค่าของสัมประสิทธิ์ แต่ละตัว โดยอาศัยคุณสมบัติการกระจายพลังงานของสัมประสิทธิ์ ของการ Transform ที่กล่าวไว้ว่าพลังงานส่วนใหญ่จะอยู่ที่ช่วงความถี่ต่ำๆ หรือ อาจนำรูปแบบการกระจายความหนาแน่นแบบ Gaussian หรือ Laplacian มาใช้ก็ได้ ดังนั้น สัมประสิทธิ์ที่อยู่บนมุมซ้ายบน เป็นกลุ่มของความถี่ต่ำซึ่งถือว่าสำคัญจะถูกกำหนดด้วยจำนวนบิทสูงในขณะที่กลุ่มสัมประสิทธิ์ที่อยู่มุมขวาล่างเป็นกลุ่มของความถี่สูงซึ่งถือว่าไม่สำคัญจะถูกกำหนดด้วยจำนวนต่ำๆ

8	8	8	7	7	7	5	5	4	4	4	4	4	4	4	4
8	8	7	6	5	5	3	3	3	3	3	2	2	2	2	2
8	7	6	4	4	4	3	3	2	2	2	2	2	2	2	2
7	6	4	3	2	2	2	2	1	1	1	1	0	0	0	0
7	5	4	2	2	2	2	1	1	1	0	0	0	0	0	0
7	5	4	2	2	2	1	1	0	0	0	0	0	0	0	0
5	5	3	2	2	1	1	0	0	0	0	0	0	0	0	0
5	3	3	2	1	1	0	0	0	0	0	0	0	0	0	0
4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0
4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0
4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0
4	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0

รูปที่ 3.2 ตัวอย่างการกำหนดบิทให้กับสัมประสิทธิ์ในโดเมนความถี่หลังการแปลงด้วย DCT กับบล็อกขนาด 16 x 16 ซึ่งสามารถลดข้อมูลลงได้จาก 8 บิทต่อจุดภาพให้เหลือ 1.5บิทต่อจุดภาพ

สำหรับขั้นตอนที่จะนำภาพกลับคืนมาหลังจากที่ได้เข้ารหัสไว้แล้ว สิ่งที่ต้องนำมาใช้คือค่าของสัมประสิทธิ์ และค่าของ Variances ของการ Transform โดยที่ค่าของ variances ถูกนำมาคำนวณค่า bit allocation จากค่าบิทเรทที่กำหนด แล้วจึงนำค่าที่ได้นี้ไปใช้ในการคำนวณปรับค่าของสัมประสิทธิ์ที่ได้จากการเข้ารหัสในตอนแรก เพื่อให้ได้ค่าสัมประสิทธิ์เดิมกลับมา (rescale coefficients) เพื่อจะได้ทำ Inverse Transform ต่อไป โดยมีสมการของการ Inverse Transform เป็นดังนี้

กรณี One dimensional Inverse Transform

$$f(n)= \sqrt{(2/N)}\ \alpha(k) \sum_{k=0}^{N-1} F(k) \cos [(2n+1)\pi k/2N] \quad \text{เมื่อ} \quad k = 0,1,...,N-1 \qquad ...(3.11)$$

เมื่อ f(n) เป็นผลที่ได้จากการทำ Inverse Transform กับ F(k) ซึ่งเป็นสัมประสิทธิ์จากการ Transform

กรณี two dimensional Inverse Transform

$$f(m,n) = \frac{2}{N}\ \alpha(u)\alpha(v) \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) \cos \left[\frac{(2m+1)u\pi}{2N}\right] \cos \left[\frac{(2n+1)v\pi}{2N}\right] \qquad ...(3.12)$$

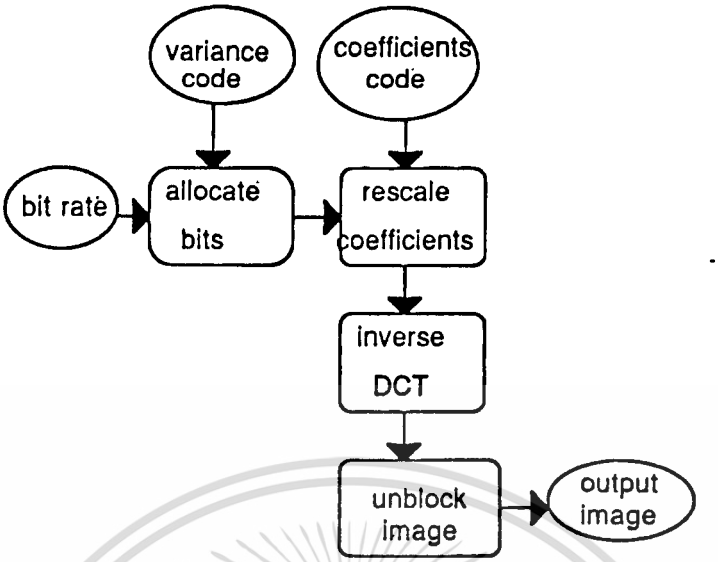
เมื่อ

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

m,n เป็นตำแหน่งของจุดภาพ มีค่าตั้งแต่เป็น 0,1,2,...N-1

F(u,v) เป็นค่าสัมประสิทธิ์ที่ได้จากการ Transform ภาพขนาด N x N

f(m,n) เป็นผลลัพธ์ของการทำ Inverse Transform



รูปที่ 3.3 แสดงบล็อกไดอะแกรมของ DCT decoding

ถึงแม้ว่าการลดข้อมูลภาพโดยใช้หลักการของการ Transform จะเป็นที่ยอมรับและได้รับความนิยมแพร่หลายมาจนทุกวันนี้เพราะสามารถสร้างในลักษณะของ real time ได้ด้วย แต่ค่าของบิทเรทที่ใช้จะอยู่ในช่วง 1 ถึง 0.5 เพราะถ้าต่ำกว่านี้คุณภาพของภาพที่ได้จะแย่มาก คือ เป็นภาพที่ขาดรายละเอียด ส่วนของขอบภาพจะเบลอไม่มีความคมชัด ทั้งนี้เนื่องจากการแปลง data domain ไปอยู่ในรูปของ frequency domain นั้น ที่ช่วงความถี่สูงๆ ซึ่งเป็นส่วนบริเวณขอบต่างๆ ในภาพมักจะถูกตัดทิ้งไปเพื่อให้ขนาดของข้อมูลลดลง จึงทำให้ขอบต่างๆ ในภาพไม่คมชัด และภาพที่ได้ยังมีลักษณะเป็นบล็อค ๆ ตามขนาดของบล็อคที่ถูกแบ่งในตอน Transform

3.6 เกณฑ์การวัดความเหมือนจริงของภาพ (Image Fidelity)

ในการลดข้อมูลภาพนั้น จะมีส่วนหนึ่งที่เกิดผิดพลาดหรือสูญเสียไป ข้อผิดพลาดที่เกิดขึ้นนี้จะมีผลในตอนที่สร้างภาพกลับคืนมา(reconstruction) และค่าความผิดพลาดนี้จะอยู่ในช่วงหนึ่งที่สามารถยอมรับได้ ดังนั้นเกณฑ์การวัดความเหมือนจริงของภาพสามารถนำมาใช้ในการวัด ประสิทธิภาพของระบบได้ ตัวอย่างเกณฑ์ที่นิยมใช้ในการวัดคุณภาพของภาพ คือ ค่า root-mean-square(rms) ของ error ระหว่างข้อมูลภาพอินพุทและข้อมูลภาพเอาท์พุท(Signal-to-Noise Ratio) เมื่อกำหนดให้ข้อมูลภาพอินพุทประกอบด้วยอาเรย์ขนาด N x N ของจุดภาพ f(x,y) โดยที่ x,y มีค่าเป็น 0,1,...N-1 แต่ละจุดภาพมีค่าของระดับสีเทาที่เป็นไปได้คือ 2^m เมื่อ m เป็นจำนวนบิทของเลขไบนารี

สำหรับทุกค่าของ x และ y ในช่วง 0,1,...N-1 ค่า error ระหว่างจุดภาพอินพุทและเอาท์พุท คือ

$$e(x,y) = g(x,y) - f(x,y) \qquad \ldots(3.1)$$

เมื่อ f(x,y) คือ ค่า input image ณ. จุด x,y ใดๆ

g(x,y) คือ ค่า output image ณ. จุด x,y ใดๆ

ค่าเฉลี่ยของ error กำลังสองของภาพ (mean square error) คือ

$$e^2 = (1/N^2) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} (e(x,y))^2$$

$$= (1/N^2) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} [g(x,y) - f(x,y)]^2 \qquad \ldots(3.2)$$

ดังนั้นค่า rms ของ error จึงสามารถเขียนได้ดังนี้ คือ

$$e_{rms} = [e^2]^{1/2} \qquad \ldots(3.3)$$

ค่า root mean square error เป็นค่าที่ใช้ในการวัดความแตกต่างของข้อมูลอินพุทกับข้อมูลเอาท์พุท แต่เมื่อพิจารณาขนาดของข้อมูลเอาท์พุทต่อขนาดของสัญญาณรบกวน(noise) ก็จะได้เป็นค่า Signa-to-Noise (SNR) เมื่อกำหนดได้สัญญาณภาพเอาท์พุทแต่ละจุดประกอบด้วยสัญญาณอินพุทบวกด้วยค่าสัญญาณรบกวน นั้นคือ

$$g(x,y) = f(x,y) + e(x,y) \qquad ...(3.4)$$

ดังนั้นค่า mean square Signal-to-Noise ของข้อมูลภาพเอาท์พุท สามารถหาได้โดยค่าเฉลี่ยของสัญญาณอินพุทกำลังสอง $f^2(x,y)$ หารด้วยค่าเฉลี่ยของสัญญาณรบกวนกำลังสอง $e^2(x,y)$ ของข้อมูลภาพทั้งหมด ซึ่งสามารถเขียนได้ดังนี้

$$(SNR)_{ms} = \frac{(1/N^2)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f^2(x,y)}{(1/N^2)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} e^2(x,y)} \qquad ...(3.5)$$

ค่า rms ของ SNR จึงสามารถเขียนได้ดังต่อไปนี้

$$(SNR)_{rms} = \left[(1/N^2)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} g^2(x,y) \Big/ (1/N^2)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} e^2(x,y) \right]^{1/2} \qquad ...(3.6)$$

โดยที่เทอมส่วนของสมการข้างบนเป็นสมการของสัญญาณรบกวนที่อยู่ในรูปของผลต่างระหว่างข้อมูลอินพุทกับเอาท์พุท

หรือ เราอาจคิดเป็นค่า$(SNR)_{rms}$ dB ได้ โดย

$$(SNR)_{rms}\ dB = 20 \log (SNR)_{rms} \qquad ....(3.7)$$

จากวิธีการที่ใช้ในการวัดความเหมือนจริงของภาพที่กล่าวมาข้างบนนี้ ไม่สามารถที่จะบ่งบอกหรือใช้เป็นเกณฑ์ในการพิจารณาได้แต่เพียงอย่างเดียว ในกรณีของภาพเอาท์พุทที่ได้จากการประมวลผลหรือรับส่งสัญญาณ โดยมีสายตาของมนุษย์เป็นตัวรับภาพจากจอภาพอีกทีหนึ่งอย่างเช่น broadcast TV, picture phone, หรืองานต่างที่มีการใช้ Image processihg เป็นต้น ระบบการมองเห็นของมนุษย์จะมีลักษณะคุณสมบัติพิเศษกล่าวคือ ระบบการมองเห็นของตาจะไวต่อความเข้มของแสงในลักษณะของ logarithmic ดังนั้น error ในบริเวณที่เป็นที่มืดของภาพจะเห็นได้ชัดกว่า error ที่อยู่ในบริเวณที่สว่าง และระบบการมองเห็นยังไวต่อการเปลี่เปลี่ยนแปลงอย่างทันทีทันใดของระดับเทาด้วย error ที่อยู่บนขอบหรือใกล้ๆ ขอบของภาพจะเห็นได้ชัดมากกว่า error ที่อยู่ในโครงสร้างที่เป็นฉากหลังของภาพ ด้วยเหตุนี้ถึงแม้ว่าภาพสองภาพจะมีค่าของ rms error เท่ากัน แต่อาจจะปรากฏความแตกต่างของคุณภาพของการมองเห็นที่แตกต่างกันได้

บทที่ 4

การลดข้อมูลภาพด้วยวิธี

Block Truncation Compression

4.1 Standard Block Truncation Compression

เป็นวิธีการเข้ารหัสที่อาศัยการแบ่งภาพออกเป็นบล็อค จำนวน N x N บล็อคแต่ละบล็อคมีขนาด M x M จุด โดย M << N จากนั้น จะอาศัยทฤษฎีทางสถิติ ทำการเปลี่ยนข้อมูลภาพให้แต่ละบล็อคมีระดับเทาเพียง 2 ระดับ โดยข้อมูลที่ถูกนี้จะเปลี่ยนไปอย่างเหมาะสม เมื่อข้อมูลในแต่ละบล็อค ถูกเปลี่ยนให้มีระดับเทาเพียง 2 ระดับ เราก็จะสามารถแทนข้อมูลเหล่านั้น ด้วยตัวเลขไบนารี่ได้ ซึ่งจะช่วยลดข้อมูลลงได้ ดังแสดงในรูป 4.1

Pixel gray levels represented symbolically

x_1 x_2 x_3 x_4	B B A A	1 1 0 0
x_5 x_6 x_7 x_8	A B B B	0 1 1 1
x_9 x_{10} x_{11} x_{12}	A A A B	0 0 0 1
x_{13} x_{14} x_{15} x_{16}	A A A B	0 0 0 1

An actual example with numerical values

121 114 056 047	204 204 017 017	1 1 0 0
037 200 247 255	017 204 204 204	0 1 1 1
016 000 012 169	017 017 017 204	0 0 0 1
043 005 007 251	017 017 017 204	0 0 0 1

(a) (b) (c)

รูปที่ 4.1 **(a) ส่วนแสดงภาพที่ถูกตัดออกเป็นบล็อค ขนาด 4 x 4 จุด**
(b) ส่วนของภาพหลังจากถูกเปลี่ยนเป็นข้อมูล 2 ระดับสี
(c) การแทนข้อมูลรูป 3.4 (b) ด้วยตัวเลขไบนารี่

จากรูป 4.1 (a) แสดงว่าส่วนของภาพขนาด 4 x 4 จุด^2 ซึ่งเราจะแทนระดับเทา ด้วย $X_1, X_2, \ldots$ เราจะคำนวณหา ค่าเฉลี่ย

$$\langle X \rangle = (1/m) \sum_{i=1}^{m} Xi = [(m-q)A + qB] / m \qquad \ldots(4.1)$$

$$\langle X^2 \rangle = (1/m) \sum_{i=1}^{m} Xi^2 = [(m-q)A^2 + qB^2] / m \qquad \ldots(4.2)$$

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 \qquad \ldots(4.3)$$

เมื่อ $m = N^2$ และ q = จำนวนของ Xi ที่มากกว่า X_{th}

ในการเปลี่ยนแปลงระดับ Gray level ของแต่ละบล็อคให้เหลือเพียง 2 ระดับเราจะนำระดับเทา ของข้อมูลจริง Xi ไปเปรียบเทียบกับ ค่า X_{th} (threshold) ซึ่งเป็นค่าเฉลี่ยของแต่ละบล็อก ($X_{th} = \langle X \rangle$)

ถ้า $Xi \geq X_{th}$ แล้ว เปลี่ยนค่า Xi เป็น A

ถ้า $Xi < X_{th}$ แล้ว เปลี่ยนค่า Xi เป็น B

เมื่อ $i = 1, 2, \ldots, m$

เมื่อ แก้สมการ (4.1) และสมการ (4.2) แล้ว จะได้ค่า A และ ค่า B ดังนี้

$$A = \langle X \rangle - \sigma (q / (m - q))^{(1/2)} \qquad \ldots(4.4)$$

และ

$$B = \langle X \rangle + \sigma ((m - q) / q)^{(1/2)} \qquad \ldots(4.5)$$

โดย

$$\sigma = (\langle X^2 \rangle - \langle X \rangle^2) \qquad \ldots(4.6)$$

ในการเข้ารหัส เพื่อเก็บลงไฟล์ (file) เราจะเก็บข้อมูล 2 ส่วน คือ ส่วนของตัวเลข Binary (Bit Plane) ดังรูป 3.4 (c) ส่วนของค่า $\langle X \rangle$ และ ค่า σ ในกรณีที่เราแบ่งภาพออกเป็นบล็อค ขนาด 4 x 4 $Pixel^2$ เราสามารถลดข้อมูลภาพต้นแบบ ที่มี ระดับเทา 256 ระดับ 8 Bit ต่อ Pixel ให้เหลือเพียง 2 บิทต่อ Pixel

4.2 Minimum Root Mean Square Error

เป็นการหาค่า A และ B เพื่อ เป็นสีแทนใน แต่ละ Block โดยเน้นที่ความพยายามทำให้มีค่า Error ที่วัดโดย วิธี Root Mean Square Error มีค่าต่ำที่สุด เท่าที่จะเป็นไปได้

โดยถ้ากำหนดให้

$$J_{MSE} = \sum_{i=1}^{m-q-1} (Yi - A)^2 + \sum_{i=m-q}^{m} (Yi - B)^2 \quad ...(4.7)$$

จะได้ว่า ค่า Root Mean Square Error เท่ากับ

$$E_{rms} = \{J_{MSE} / m\}^{(1/2)} \quad ...(4.8)$$

โดยที่

Yi คือ ค่า Pixel Xi ที่ได้รับการจัดเรียงข้อมูลจาก น้อยไปหามาก

m คือจำนวนจุดใน 1 Block

q คือจำนวนจุดที่มีค่ามากกว่า Xth ($=\langle X \rangle$)

นั่นคือการที่จะทำให้ค่า Root Mean Square Error มีค่าต่ำสุด จะต้องให้เทอมของ $(Yi-A)^2$ และ เทอมของ $(Yi - B)^2$ มีค่าน้อยที่สุด ซึ่งค่าของ ขึ้นอยู่กับค่าของ A และ B ซึ่งเราจะได้ว่า

$$A = \left(\sum_{i=1}^{m-q-1} Yi \right)/(m-q) \quad ...(4.9)$$

$$B = \left(\sum_{i=m-q}^{m} Yi \right)/q \quad ...(4.10)$$

4.3 Minimum Mean Absolute Error

การลดข้อมูลภาพวิธีนี้จะหาค่าของ A และ B แทนสีของแต่ละ จุดภาพ โดยพยายามให้ได้ค่าของ MAE ต่ำที่สุด เท่าที่จะเป็นไปได้ โดยปัญหาของ MAE พบว่าใกล้เคียงกับ MSE มาก ค่าของ A และ B หาได้ จาก การทำ Minimizing

$$J_{MAE} = \sum_{i=1}^{m-q-1} | Yi - A | + \sum_{i=m-q}^{m} | Yi - B |^2 \quad ...(4.7)$$

ที่ซึ่ง

$$A = \text{median of}(Y_1, Y_2, \ldots, Y_{m-q-1})$$
$$B = \text{median of}(Y_{m-q}, \ldots, Y_m)$$

4.4 Average Minimum of Mean Square Error and Mean Absolute Error

วิธีนี้เป็นวิธีผสมระหว่างวิธีการทั้ง 2 วิธี คือ วิธี Minimum Root Mean Square Error และ วิธี Minimum Mean Average Error โดยการหาค่า A และ B จากสมการทั้งสองแล้วนำมารวมกัน หาค่าเฉลี่ย ทั้งนี้เพื่อให้ได้ค่าของข้อมูลถูกต้องซึ่งจะได้ค่าของ MRMSE และ

4.5 ความสามารถในการลดข้อมูลด้วยวิธี Block Truncation Coding

ด้วยวิธีของ BTC นี้ เรายังสามารถลดข้อมูลลงให้มีอัตราข้อมูลน้อยกว่า 2 Bit/Pixel กล่าวคือ ค่า $\langle X \rangle$ และ σ สามารถแทนได้ด้วยข้อมูลขนาด 6 และ 4 Bit ตามลำดับ ซึ่งจะไม่ทำให้ภาพหลังการถอดรหัสแตกต่างจากภาพเดิมมากนักนั้นหมายถึง อัตราข้อมูลต่อ 1 จุดภาพ จะเหลือเพียง 1.63 Bit/Pixel นอกจากนี้แล้วบล็อคที่มีค่า $\sigma = 0$ แล้ว จะมีค่าของ A และ B เท่ากันดังสมการที่ (3.16) และ (3.17) ซึ่งทำให้ส่วนของ Bit Plane ถูกตัดทิ้งไปได้ นั้นคืออัตราข้อมูล ต่อ 1 จุดภาพจะลดลงเป็นลำดับ และสุดท้าย คือการกำหนดขนาดของบล็อคที่เหมาะสม ยังช่วยให้อัตราข้อมูลลดลงด้วย เช่น ภาพขนาด 256 x 256 pixel^2 256 Gray level ขนาดของบล็อค คือ 4 x 4 Pixel^2 จะมีอัตราข้อมูล 2 Bit/Pixel ถ้าขนาดของบล็อค คือ 8 x 8 Pixel^2 จะมีอัตราข้อมูล 1.25 Bit/Pixel แต่ก็ต้องคำนึงถึงค่าความคลาด

เคลื่อนของภาพ หลังการถอดรหัสด้วย เพราะยิ่งบล็อคยิ้งมีขนาดใหญ่ ค่าความคลาดเคลื่อนจะยิ่งมากขึ้นด้วย

ขนาดของบล็อค	BIT RATE (Bits/Pixel)	ปริมาณการลดข้อมูล (%)
2 x 2	5.0000	37.50000
4 x 4	2.0000	75.00000
8 x 8	1.2500	84.37500
16 x 16	1.0625	86.71875

ตาราง 3.1 แสดงความแตาต่างของปริมาณการลดข้อมูลภาพของ **BTC** ที่กำหนด **Block** ต่าง ๆ กัน

นอกจากนี้แล้วในวิธีที่ 4.2 ถึง 4.4 จะเก็บส่วน header เป็น ข้อมูลของ ค่า สี A และ B ที่หาได้จากสมการดังกล่าว ดังนั้น จึง ทำให้ต้องใช้ 16 Bit สำหรับส่วน header นี้ อย่างไรก็ตาม เรายังสามารถ จัดเก็บ ค่าของสี A และ B เป็นแบบ ข้อมูล 6 Bit ได้อีกด้วย ซึ่งจะทำให้ ได้ข้อมูล ส่วนที่เป็น header ลดลงเหลือ 12 Bit และ หากค่าของ สี A และ B มีค่าเท่ากัน เราก็สามารถ ตัด ส่วนของ Bit plan ออกได้อีกด้วย

บทที่ 5

การประยุกต์ใช้เทคนิคการลดข้อมูลภาพ

(Image data Compression Application)

และผลการทดลอง

จากการศึกษาถึงเทคนิคในการลดข้อมูลภาพ ดังได้กล่าวมาแล้วในบทที่ 3 นั้น พบว่า แต่ละเทคนิค มีรูปแบบและความเหมาะสม สำหรับภาพแต่ละประเภทแตกต่างกันไป บางเทคนิคสามารถลดข้อมูลได้น้อยตั้งแต่ 10 - 20 % แต่ภาพที่ได้จากการถอดรหัส จะมีข้อผิดพลาดน้อยมากหรือไม่ผิดพลาดเลย เพราะฉะนั้นในการเลือกใช้เทคนิคที่เหมาะสมก็ขึ้นอยู่กับลักษณะข้อมูลภาพ และการนำไปใช้งานว่าต้องการประสิทธิภาพ หรือต้องการปริมาณการลดข้อมูลมาก ๆ หรือต้องการความรวดเร็ว ในบทนี้จะเป็นการศึกษาเปรียบเทียบคุณสมบัติของการลดข้อมูลภาพ แบบต่างๆ กัน เพื่อค้นหา และ ปรับปรุง หรือ ออกแบบ อัลกอริทึม ในการลด ขนาดข้อมูลภาพ ต่อไป

จากการศึกษาและทดสอบความเป็นไปได้ที่จะลดข้อมูลภาพ โดยทดสอบกับภาพดิจิติลขนาด 256 จุด^2 ใช้ระดับเทา 256 ระดับ และ ภาพ สี 24 bit 16.7 ล้านสี โดยมีขั้นตอนดังนี้

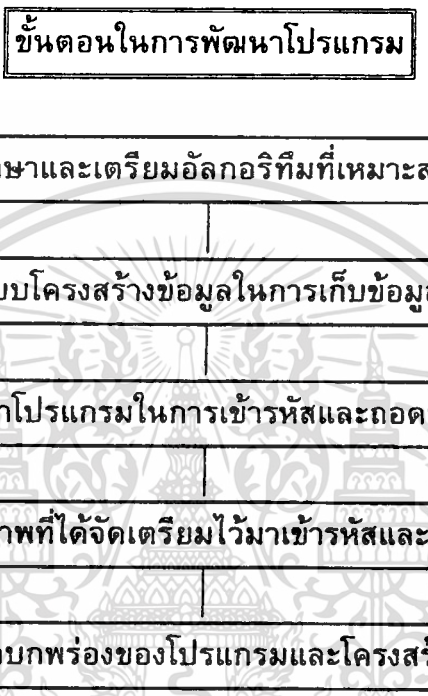
ขั้นตอนการวิจัย

แบ่งแยกประเภทตามลักษณะของการกระจายของข้อมูล

จัดเตรียมอัลรีอริทึม ของ BTC แบบต่าง ๆ สำหรับลดขนาดข้อมูล โดยสอดคล้องกับประเภทของภาพ

ประเมินผล เปรียบเทียบการใช้ฟังก์ชันต่าง ๆ เพื่อลดข้อมูลภาพ

หลังจากได้ทำการประเมินผลที่ได้ จากการใช้ฟังก์ชันต่างๆ เพื่อลดข้อมูลภาพแบบต่าง ๆ แล้ว เราก็ต้องทำการพัฒนาโปรแกรม เพื่อให้สามารถนำไปใช้งานได้จริง โดยมีขั้นตอนในการพัฒนาโปรแกรม ดังนี้



วัตถุประสงค์ของการลดข้อมูลภาพ ที่ได้พัฒนาขึ้นในโครงงานวิจัยนี้ เพื่อประหยัดเนื้อที่หน่วยความจำ และช่วยในการส่งข้อมูลให้ได้อย่างรวดเร็ว เพราะฉะนั้นการเลือกวิธีการลดข้อมูลภาพจากทฤษฎีที่ได้ศึกษามาทั้งหมด ประกอบกับวัตถุประสงค์ ในการส่งข้อมูลที่ได้ทำการเข้ารหัส เพื่อประหยัดค่าใช้จ่ายและความรวดเร็ว สิ่งที่ต้องคำนึงถึงสำหรับการลดข้อมูลภาพคือความเร็วในการเข้ารหัส และ ถอดรหัส ว่าจะต้องอยู่ในช่วงเวลาที่เหมาะสม โดยมีคุณภาพของภาพเป็นที่ยอมรับได้ เทคนิคที่เหมาะสมมีหลายวิธี เหมาะกับภาพต่างๆ กัน สำหรับในปริญญานิพนธ์ฉบับนี้ ได้ศึกษาเทคนิคการลดข้อมูลภาพด้วยวิธี Block Trancation Coding ด้วย วิธี แตกต่างกัน 4 วิธี ดังนี้

1. Simple BTC
2. Minimum Root Mean Square Error
3. Minimum Mean Absolute Error
4. Average Minimum of RMSE and MAE

โดยแต่ละ วิธี ยังมีการเข้า รหัส ต่างกัน อีก แบบละ 2 วิธี คือ

1. เข้ารหัสมาตรฐาน
2. เข้ารหัสให้ได้ขนาดเล็กที่สุด

โดยการทดสอบนี้อยู่ภายใต้ ข้อกำหนดดังนี้

- เครื่อง 80486-33 ใช้ VGA Gard
- ใช้ BIOS ของ Cirrus Logic
- ใช้ โหมด แสดงผล 640 x 480 Pixel 16.7 ล้านสี
- ใช้ภาพขนาด 256 Pixel x 256 Pixel 256 Gray level
- และใช้ภาพ ขนาด 256 Pixel x 256 Pixel RGB Color 24 Bit

โดยแบ่งการทดสอบเป็นการทดสอบ ใน 3 เรื่องด้วยกัน คือ

1. ในเรื่องของความเร็ว เปรียบเทียบความเร็ว ของทุกวิธี
2. ในเรื่องของคุณภาพ จะ เปรียบเทียบโดย กำหนดให้เข้ารหัสแบบมาตรฐาน เพื่อให้ได้ ขนาดของภาพเท่ากัน ง่ายต่อการเปรียบเทียบ
3. ในเรื่องของขนาด โดยการหาอัตราการลดข้อมูลสูงสุดของแต่ละวิธี

โดยทดสอบกับภาพจำนวน 8 ภาพด้วยกัน เป็นภาพสี 24 bit 4 ภาพ และ ภาพ Gray level 8 bit (256 ระดับ) 4 ภาพ คือ

1.ภาพ Mandrill	24 bit
2.ภาพ Can	24 bit
3.ภาพ Flora	24 bit
4.ภาพ Ramanoy	24 bit
5.ภาพ Mandrill	Gray level 8 bit
6.ภาพ Can	Gray level 8 bit
7.ภาพ Flora	Gray level 8 bit
8.ภาพ Ramanoy	Gray level 8 bit

ผลการทดลองเป็นดังนี้

ตาราง 5.1 แสดง ผลการ Compress ภาพ Mandrill 24 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	9.95	7.36	22.87	13.90	37.59
high simple	10.48	7.82	22.43	13.85	57.07
Mrms	8.41	5.53	24.34	12.75	37.59
high Mrms	8.55	5.69	24.19	12.69	50.56
Mmae	11.44	5.33	21.66	15.49	37.59
high Mmae	11.69	6.26	21.47	15.33	50.56
Am	9.26	5.42	23.50	20.88	37.59
high Am	9.43	5.86	23.34	21.54	50.26

ตาราง 5.2 แสดง ผลการ Compress ภาพ Mandrill 24 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	15.91	12.36	18.80	7.64	75.00
high simple	16.32	12.56	18.58	7.64	79.77
Mrms	14.34	10.33	19.70	7.58	75.00
high Mrms	14.42	10.38	19.65	7.58	78.17
Mmae	14.82	10.04	19.41	9.62	75.00
high Mmae	15.03	10.37	19.29	9.62	78.17
AM	14.46	10.18	19.63	12.31	75.00
high AM	14.57	10.35	19.56	12.31	78.15

ตาราง 5.3 แสดง ผลการ Compress ภาพ Mandrill 24 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	18.42	14.41	17.52	6.15	84.38
high simple	18.80	14.61	17.35	5.99	85.56
Mrms	16.91	12.45	18.27	6.21	84.38
high Mrms	16.98	12.50	18.23	6.21	85.16
Mmae	16.96	12.41	18.24	8.24	84.38
high Mmae	17.14	12.58	18.15	8.02	85.16
AM	16.92	12.43	18.26	9.95	84.38
high AM	17.02	12.55	18.21	10.00	85.16

ตาราง 5.4 แสดง ผลการ Compress ภาพ Mandrill 24 bit ด้วย Block 16X16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	20.38	15.96	16.65	5.60	86.72
high simple	20.73	16.19	16.50	5.60	87.01
Mrms	18.93	14.17	17.29	5.93	86.72
high Mrms	18.99	14.21	17.26	5.93	86.91
Mmae	18.92	14.16	17.29	7.69	86.72
high Mmae	19.08	14.30	17.22	7.64	86.91
AM	18.92	14.16	17.29	9.34	86.72
high AM	19.02	14.26	17.25	9.45	86.91

ตาราง 5.5 แสดง ผลการ Compress ภาพ CAN 24 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	3.82	1.58	28.61	12.69	46.79
high simple	4.10	2.61	27.99	12.58	66.73
Mrms	2.30	0.79	33.00	11.65	46.79
high Mrms	3.05	2.05	30.55	11.43	60.13
Mmae	3.08	0.76	30.47	12.64	46.79
high Mmae	3.74	2.21	28.79	12.25	60.13
AM	2.52	0.77	32.22	16.76	46.79
high AM	3.24	2.09	30.04	17.20	59.69

ตาราง 5.6 แสดง ผลการ Compress ภาพ CAN 24 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.84	3.57	23.54	7.42	80.29
high simple	6.73	3.81	23.69	7.42	88.20
Mrms	4.53	1.72	27.13	7.25	78.17
high Mrms	4.94	2.73	26.38	6.92	85.34
Mmae	4.77	1.63	26.67	8.85	78.17
high Mmae	5.29	2.96	25.77	7.91	85.34
AM	4.58	1.65	27.03	11.21	78.17
high AM	4.98	2.72	26.32	11.37	84.51

ตาราง 5.7 แสดง ผลการ Compress ภาพ CAN 24 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	9.19	5.63	20.98	5.99	84.57
high simple	9.24	5.37	20.94	5.93	91.14
Mrms	6.66	2.77	23.77	6.21	84.57
high Mrms	6.90	3.38	23.47	5.99	88.55
Mmae	6.73	2.70	23.69	7.97	84.57
high Mmae	7.15	3.84	23.16	7.36	88.55
AM	6.67	2.70	23.76	9.89	84.57
high AM	6.92	3.46	23.44	9.89	88.35

ตาราง 5.8 แสดง ผลการ Compress ภาพ CAN 24 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	11.08	6.91	19.36	5.66	86.82
high simple	11.64	7.14	18.92	5.66	90.04
Mrms	9.16	4.32	21.01	5.88	86.82
high Mrms	9.28	4.61	20.90	5.88	87.48
Mmae	9.16	4.28	21.00	7.69	86.82
high Mmae	9.51	5.14	20.68	7.53	87.48
AM	9.15	4.28	21.02	9.40	86.82
high AM	9.34	4.71	20.84	9.45	87.03

ตาราง 5.9 แสดง ผลการ Compress ภาพ FLORA 24 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	5.38	3.02	20.49	13.79	39.28
high simple	6.25	4.16	19.18	13.30	62.63
Mrms	4.60	2.23	21.85	12.64	38.83
high Mrms	5.04	2.98	21.05	12.14	54.24
Mmae	6.24	2.14	19.21	15.05	38.83
high Mmae	6.80	3.52	18.45	14.12	54.24
AM	5.06	2.18	21.03	20.27	38.83
high AM	5.40	3.03	20.45	21.04	52.82

ตาราง 5.10 แสดง ผลการ Compress ภาพ FLORA 24 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	9.83	5.67	15.25	7.58	75.11
high simple	10.42	6.47	14.75	7.42	84.18
Mrms	9.13	4.81	15.90	7.47	75.08
high Mrms	9.36	5.32	15.67	7.31	80.36
Mmae	9.52	4.66	15.53	9.56	75.08
high Mmae	9.90	5.54	15.19	9.01	80.36
AM	9.22	4.72	15.81	12.14	75.08
high AM	9.41	5.19	15.63	12.36	79.40

ตาราง 5.11 แสดง ผลการ Compress ภาพ FLORA 24 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	13.09	7.67	12.76	6.15	84.38
high simple	13.61	8.34	12.43	5.99	88.82
Mrms	12.37	6.75	13.28	6.21	84.38
high Mrms	12.58	7.10	13.11	6.15	86.34
Mmae	12.42	6.72	13.22	8.02	84.38
high Mmae	12.76	7.33	12.99	7.80	86.34
AM	12.37	6.73	13.25	9.95	84.38
high AM	12.51	7.04	13.16	10.00	85.96

ตาราง 5.12 แสดง ผลการ Compress ภาพ FLORA 24 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	16.41	9.74	10.80	5.66	86.72
high simple	16.91	10.41	10.54	5.66	89.34
Mrms	15.61	8.80	11.23	5.93	86.72
high Mrms	15.83	9.05	11.11	5.88	87.43
Mmae	15.61	8.82	11.24	7.69	86.72
high Mmae	15.95	9.26	11.05	7.53	87.43
AM	15.60	8.82	11.24	9.40	86.72
high AM	15.72	9.04	11.18	9.45	87.52

ตาราง 5.13 แสดง ผลการ Compress ภาพ Ramanoy 24 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.51	4.64	22.74	15.18	34.58
high simple	7.42	5.51	21.60	15.07	58.14
Mrms	5.12	3.20	24.3	13.08	37.56
high Mrms	5.36	3.45	24.42	12.91	50.82
Mmae	6.95	3.09	22.17	15.71	37.56
high Mmae	7.37	4.13	21.66	15.44	50.82
AM	5.63	3.14	24.00	21.15	37.56
high AM	5.91	3.69	23.57	21.65	50.28

ตาราง 5.14 แสดง ผลการ Compress ภาพ RAMANOY 24 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	10.83	7.91	18.32	7.64	6.00
high simple	11.47	8.46	17.81	7.64	4.83
Mrms	9.66	6.54	19.31	7.58	6.00
high Mrms	9.79	6.65	19.19	7.58	5.24
Mmae	9.96	6.38	19.04	9.67	6.00
high Mmae	10.26	6.79	18.79	9.62	5.24
AM	9.73	6.45	19.25	12.25	6.00
high AM	9.91	6.69	19.09	12.36	5.25

ตาราง 5.15 แสดง ผลการ Compress ภาพ RAMANOY 24 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	13.67	10.11	16.29	6.04	84.38
high simple	14.30	10.67	15.90	5.99	85.60
Mrms	12.57	8.78	17.02	6.21	84.38
high Mrms	12.71	8.87	16.92	6.21	85.17
Mmae	12.61	8.76	16.99	8.02	84.38
high Mmae	12.82	8.99	16.80	8.08	85.17
AM	12.58	8.77	17.02	9.95	84.38
high AM	12.71	8.90	16.92	9.95	85.16

ตาราง 5.16 แสดง ผลการ Compress ภาพ RAMANOY 24 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	16.48	12.40	14.67	5.60	86.72
high simple	17.00	12.86	14.40	5.55	87.04
Mrms	15.39	11.05	15.27	5.93	86.72
high Mrms	15.55	11.13	15.17	5.99	86.93
Mmae	15.38	11.05	15.27	7.69	86.72
high Mmae	15.66	11.22	15.11	7.69	86.93
AM	15.38	11.05	15.27	9.45	86.72
high AM	15.49	11.12	15.21	9.45	86.91

ตาราง 5.17 แสดง ผลการ Compress ภาพ Mandrill Gray level 8 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	9.92	6.67	23.02	9.51	37.53
high simple	9.93	7.52	22.39	9.56	57.06
Mrms	7.57	5.05	24.75	9.12	37.52
high Mrms	7.74	5.22	24.56	9.12	50.40
Mmae	10.30	4.88	22.08	10.00	37.52
high Mmae	10.60	5.83	21.82	10.00	50.40
AM	8.33	4.97	23..91	12.03	37.52
high AM	8.52	5.39	23.72	12.09	50.13

ตาราง 5.18 แสดง ผลการ Compress ภาพ Mandrill Gray level 8 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	14.41	11.31	19.15	5.16	75.00
high simple	14.98	11.67	18.82	5.05	79.74
Mrms	12.83	9.31	20.16	5.05	75.00
high Mrms	12.96	9.39	20.08	5.05	78.12
Mmae	13.24	9.07	19.89	5.77	75.00
high Mmae	13.49	9.42	19.73	5.77	78.12
AM	12.93	9.19	20.10	6.81	75.00
high AM	13.07	9.37	20.01	6.81	78.12

ตาราง 5.19 แสดง ผลการ Compress ภาพ Mandrill Gray level 8 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	16..60	13.12	17.93	4.07	84.38
high simple	17.05	13.41	17.69	4.01	85.55
Mrms	15.09	11.20	18.76	4.07	84.38
high Mrms	15.19	11.25	18.70	4.12	85.16
Mmae	15.12	11.17	18.74	4.73	84.38
high Mmae	15.34	11.35	18.61	4.73	85.16
AM	15.09	11.18	18.75	5.49	84.38
high AM	15.21	11.30	18.69	5.49	85.16

ตาราง 5.20 แสดงผลการ Compress ภาพ Mandrill Gray level 8 bit Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	18.13	14.30	17.16	3.79	86.72
high simple	18.47	16.57	17.00	3.79	87.01
Mrms	16.78	12.69	17.84	3.85	86.72
high Mrms	16.83	12.73	17.81	3.85	86.91
Mmae	16.77	12.69	17.84	4.45	86.72
high Mmae	16.92	12.82	17.76	4.45	86.91
AM	16.77	12.68	17.84	5.11	86.72
high AM	16.87	12.77	17.79	5.16	86.91

ตาราง 5.21 แสดง ผลการ Compress ภาพ CAN Gray level 8 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	3.62	1.49	28.36	9.29	96.60
high simple	3.96	2.63	27.57	9.23	66.74
Mrms	2.03	0.68	33.38	8.90	46.38
high Mrms	2.89	2.04	30.30	8.79	60.06
Mmae	2.70	0.66	30.90	9.18	46.38
high Mmae	3.48	2.22	28.70	9.01	60.06
AM	2.22	0.67	32.62	10.77	46.38
high AM	3.04	2.09	29.87	10.77	59.60

ตาราง 5.22 แสดง ผลการ Compress ภาพ CAN Gray level 8 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.19	3.07	23.70	5.05	81.69
high simple	6.23	3.70	23.65	5.05	88.35
Mrms	3.92	1.56	27.67	5.00	81.12
high Mrms	4.45	2.72	26.57	4.95	86.22
Mmae	4.11	1.49	27.25	5.38	81.12
high Mmae	4.75	2.91	26.01	5.22	86.22
AM	3.96	1.51	27.58	6.26	81.12
high AM	4.50	2.76	26.48	6.21	85.78

ตาราง 5.23 แสดง ผลการ Compress ภาพ CAN Gray level 8 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	8.30	4.86	21.16	4.07	87.21
high simple	8.31	5.00	21.14	4.01	91.78
Mrms	5.74	2.41	24.36	4.07	86.11
high Mrms	6.10	3.41	23.82	4.01	90.89
Mmae	5.78	2.35	24.29	4.62	86.11
high Mmae	6.29	3.66	23.57	4.34	90.89
AM	5.74	2.36	24.36	5.38	86.11
high AM	6.08	3.35	23.85	5.33	89.71

ตาราง 5.24 แสดง ผลการ Compress ภาพ CAN Gray level 8 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	10.21	6.99	19.35	3.85	86.82
high simple	10.41	6.73	19.19	3.85	89.75
Mrms	7.83	3.60	21.66	3.90	86.82
high Mrms	8.01	4.13	21.46	3.90	88.92
Mmae	7.82	3.54	21.67	4.56	86.82
high Mmae	8.22	4.63	21.24	4.40	88.92
AM	7.82	3.54	21.67	5.22	86.82
high AM	8.07	4.31	21.39	5.22	88.67

ตาราง 5.25 แสดง ผลการ Compress ภาพ Flora Gray level 8 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.29	3.67	19.54	9.67	38.75
high simple	7.04	4.65	18.57	9.56	61.65
Mrms	5.60	2.86	20.55	9.29	38.49
high Mrms	5.90	3.44	20.10	9.18	53.25
Mmae	7.64	2.76	17.86	10.05	38.49
high Mmae	8.06	4.05	17.39	9.89	53.25
AM	6.17	2.80	19.71	12.20	38.49
high AM	6.45	3.61	19.33	12.20	52.31

ตาราง 5.26 แสดง ผลการ Compress ภาพ Flora Gray level 8 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
xsimple	11.32	6.83	14.44	5.16	75.05
high simple	11.90	7.48	14.01	5.11	83.32
Mrms	10.56	5.85	15.04	5.16	75.03
high Mrms	10.78	6.18	14.87	5.11	79.34
Mmae	11.02	5.67	14.68	5.82	75.03
high Mmae	11.38	6.40	14.39	5.71	79.34
AM	10.67	5.75	14.95	6.87	75.03
high AM	10.84	6.19	14.81	6.81	79.39

ตาราง 5.27 แสดง ผลการ Compress ภาพ Flora Gray level 8 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	14.49	8.90	12.29	4.01	84.39
high simple	14.95	9.45	12.02	4.01	88.06
Mrms	13.71	7.81	12.78	4.12	84.39
high Mrms	13.89	8.02	12.66	4.12	85.60
Mmae	13.76	7.77	12.74	4.73	84.39
high Mmae	14.07	8.24	12.55	4.73	85.60
AM	13.71	7.79	12.77	5.44	84.39
high AM	13.85	8.09	12.69	5.44	86.16

ตาราง 5.28 แสดง ผลการ Compress ภาพ Flora Gray level 8 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	17.91	11.12	10.46	3.85	86.72
high simple	18.45	11.72	10.19	3.79	88.82
Mrms	17.02	10.01	10.90	3.90	86.72
high Mrms	17.24	10.15	10.79	3.90	86.96
Mmae	17.02	10.03	10.90	4.56	86.72
high Mmae	17.34	10.35	10.73	4.51	86.96
AM	17.01	10.03	10.90	5.22	86.72
high AM	17.12	10.23	10.85	5.22	87.70

ตาราง 5.29 แสดงผลการ Compress ภาพ Ramanoy Gray level 8 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.67	4.82	24.62	9.62	37.75
high simple	7.29	5.32	23.85	9.67	59.23
Mrms	4.30	2.61	28.43	9.34	37.72
high Mrms	4.60	2.94	27.85	9.29	51.57
Mmae	5.87	2.52	25.73	10.22	37.72
high Mmae	6.32	3.62	25.09	10.05	51.57
AM	4.74	2.56	27.59	12.36	37.72
high AM	5.09	3.18	26.98	12.36	50.63

ตาราง 5.30 แสดงผลการ Compress ภาพ Ramanoy Gray level 8 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	10.71	7.86	20.51	5.16	75.04
high simple	11.24	8.19	20.09	5.11	80.14
Mrms	9.06	5.99	21.96	5.16	75.04
high Mrms	9.15	6.10	21.88	5.16	78.29
Mmae	9.34	5.84	21.70	5.82	75.04
high Mmae	9.58	6.27	21.47	5.82	78.29
AM	9.12	5.90	21.90	6.87	75.04
high AM	9.30	6.18	21.73	6.87	78.19

ตาราง 5.31 แสดงผลการ Compress ภาพ Ramanoy Gray level 8 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	13.68	10.08	18.39	4.01	84.39
high simple	14.21	10.51	18.06	4.01	85.63
Mrms	12.40	8.56	19.24	4.12	84.39
high Mrms	12.48	8.64	19.18	4.12	85.21
Mmae	12.42	8.53	19.22	4.73	84.39
high Mmae	12.65	8.78	19.06	4.73	85.21
AM	12.40	8.54	19.24	5.49	84.39
high AM	12.54	8.71	19.14	5.49	85.17

ตาราง 5.32 แสดงผลการ Compress ภาพ Ramanoy Gray level 8 bit Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	17.45	12.99	16.27	3.79	86.72
high simple	17.84	13.35	16.08	3.85	87.01
Mrms	16.20	11.57	16.91	3.90	86.72
high Mrms	16.28	11.64	16.87	3.96	86.91
Mmae	16.20	11.57	16.92	4.51	86.72
high Mmae	16.38	11.73	16.82	4.51	86.91
AM	16.20	11.57	16.92	5.22	86.72
high AM	16.30	11.67	16.86	5.22	86.91

จากผลการทดลองดังกล่าว สามารถนำมาประมวลประสิทธิภาพ โดยเฉลี่ย ของแต่ละวิธี โดยใช้ Block ขนาด ต่างๆ ได้ ดังตารางต่อไปนี้

ตาราง 5.33 แสดง ประสิทธิภาพ Compress ภาพ 24 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.42	4.15	23.68	13.64	40.31
high simple	7.06	5.03	22.80	13.45	61.15
Mrms	5.11	2.94	26.00	12.53	40.13
high Mrms	5.50	3.54	25.05	12.29	53.94
Mmae	6.93	2.83	23.38	14.72	40.13
high Mmae	7.40	4.03	22.59	14.29	53.94
AM	5.62	2.88	25.14	19.77	40.13
high AM	6.00	3.67	24.35	20.35	53.26

ตาราง 5.34 แสดง ประสิทธิภาพ Compress ภาพ 24 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	10.85	7.38	18.98	7.57	76.35
high simple	11.24	7.83	18.71	7.53	83.02
Mrms	9.42	5.85	20.51	7.47	75.81
high Mrms	9.63	6.27	20.22	7.35	80.51
Mmae	9.77	5.68	20.16	9.43	75.81
high Mmae	10.12	6.42	19.76	9.04	80.51
AM	9.50	5.75	20.43	11.98	75.81
high AM	9.72	6.24	20.15	12.11	80.05

ตาราง 5.35 แสดง ประสิทธิภาพCompress ภาพ 24 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	13.59	9.45	16.89	6.08	84.43
high simple	13.99	9.75	16.65	5.98	87.78
Mrms	12.13	7.69	18.09	6.21	84.43
high Mrms	12.29	7.96	17.93	6.14	86.31
Mmae	12.18	7.65	18.04	8.06	84.43
high Mmae	12.49	8.19	17.78	7.82	86.31
AM	12.14	7.66	18.07	9.94	84.43
high AM	12.29	7.99	17.93	9.96	86.16

ตาราง 5.36 แสดง ประสิทธิภาพ Compress ภาพ 24 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	16.09	11.25	15.37	5.63	86.75
high simple	16.57	11.65	15.09	5.62	88.36
Mrms	14.77	9.59	16.20	5.92	86.75
high Mrms	14.91	9.75	16.11	5.92	87.19
Mmae	14.77	9.58	16.20	7.69	86.75
high Mmae	15.05	9.98	16.02	7.60	87.19
AM	14.76	9.58	16.21	9.40	86.75
high AM	14.89	9.78	16.12	9.44	87.09

ตาราง 5.37 แสดง ประสิทธิภาพ Compress ภาพ Gray level 8 bit ด้วย Block 2x2

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	6.63	4.16	23.89	9.52	40.16
high simple	7.06	5.03	23.10	9.51	61.17
Mrms	4.88	2.80	26.78	9.16	40.03
high Mrms	5.28	3.41	25.70	9.10	53.82
Mmae	6.63	2.71	24.14	9.86	40.03
high Mmae	7.12	3.93	23.25	9.74	53.82
AM	5.37	2.75	25.96	11.84	40.03
high AM	5.78	3.57	24.98	11.86	53.17

ตาราง 5.38 แสดง ประสิทธิภาพ Compress ภาพ Gray level 8 bit ด้วย Block 4x4

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	10.66	7.27	19.45	5.13	76.70
high simple	11.09	7.76	19.14	5.08	82.89
Mrms	9.09	5.68	21.21	5.09	76.55
high Mrms	9.34	6.10	20.85	5.07	80.49
Mmae	9.43	5.52	20.88	5.70	76.55
high Mmae	9.80	6.25	20.40	5.63	80.49
AM	9.17	5.59	21.13	6.70	76.55
high AM	9.43	6.13	20.76	6.68	80.37

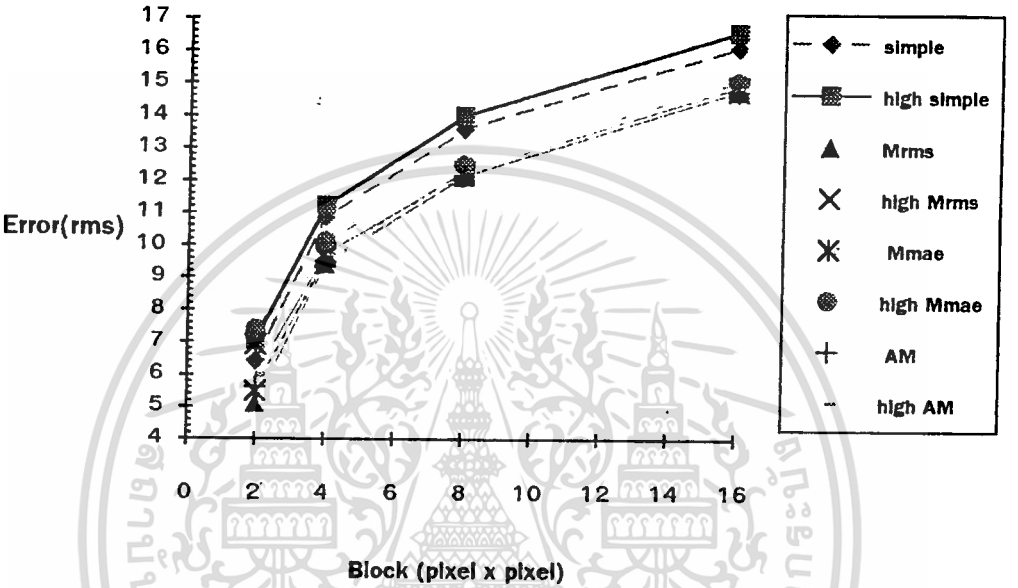
ตาราง 5.39 แสดง ประสิทธิภาพ Compress ภาพ Gray level 8 bit ด้วย Block 8x8

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	13.27	9.24	17.44	4.04	85.09
high simple	13.63	9.59	17.23	4.01	87.76
Mrms	11.74	7.50	18.79	4.10	84.82
high Mrms	11.92	7.83	18.59	4.09	86.72
Mmae	11.77	7.40	18.75	4.70	84.82
high Mmae	12.09	8.01	18.45	4.63	86.72
AM	11.74	7.47	18.78	5.44	84.32
high AM	11.92	7.86	18.59	5.44	86.55

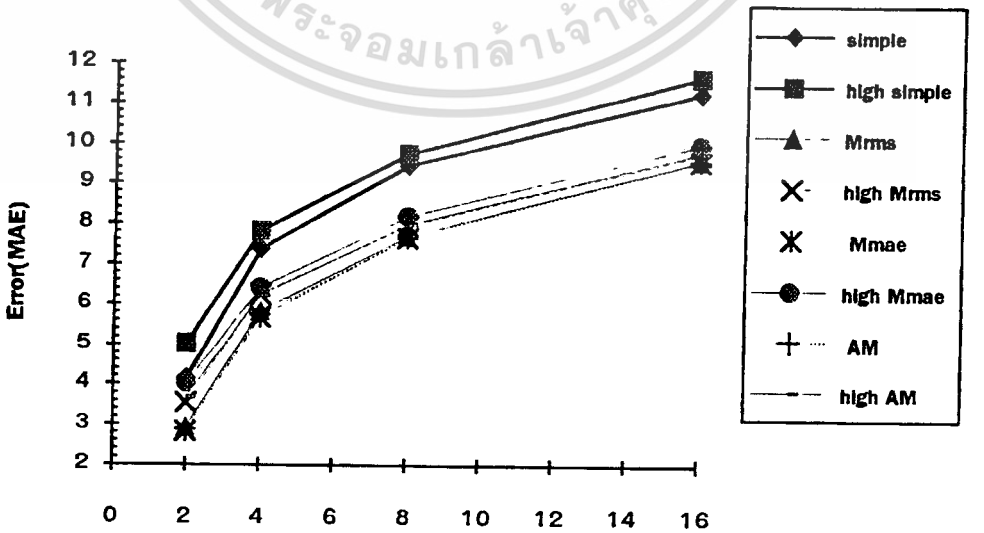
ตาราง 5.40 แสดงประสิทธิภาพ Compress ภาพ Gray level 8 bit ด้วย Block 16x16

Method	rms Error	MAE	S/N (dB)	time	% Compress
simple	15.93	11.35	15.81	3.81	86.75
high simple	16.29	11.59	15.62	3.82	88.15
Mrms	14.46	9.47	16.83	3.89	86.75
high Mrms	14.59	9.66	16.73	3.90	88.18
Mmae	14.45	9.46	16.83	4.52	86.75
high Mmae	14.72	9.88	16.64	4.47	87.43
AM	14.45	9.46	16.83	5.19	86.75
high AM	14.59	9.75	16.72	5.21	87.55

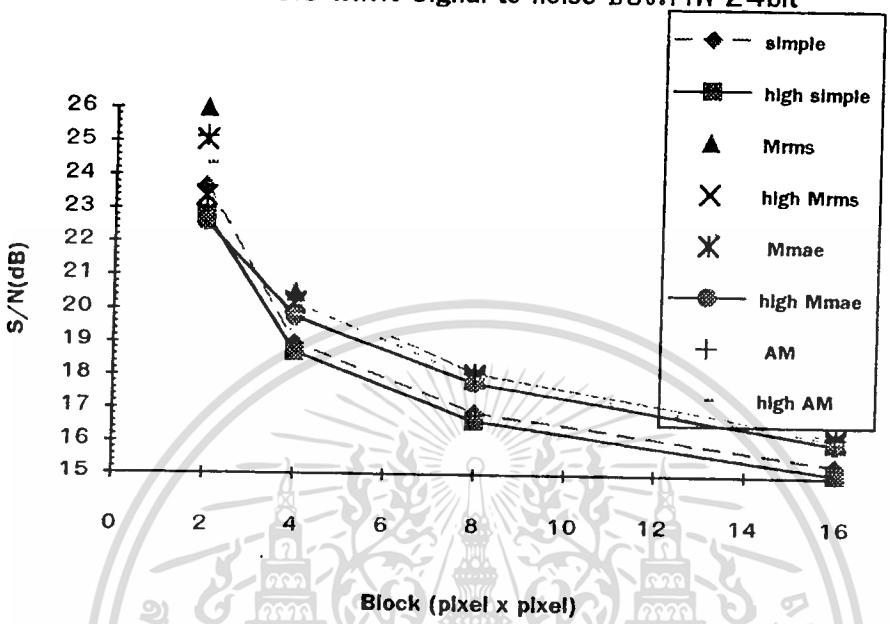
กราฟที่ 5.1 แสดง ค่า RMS Error ของภาพ 24 bit



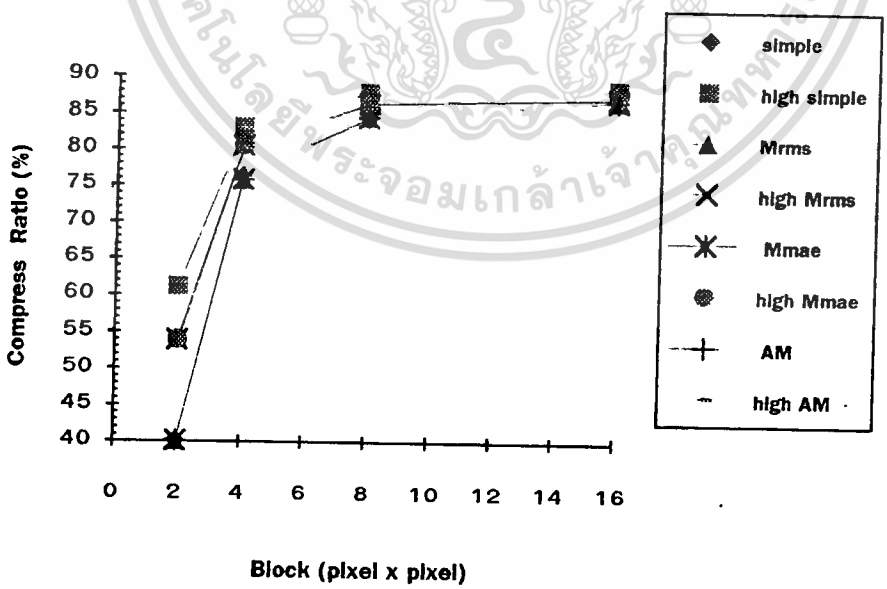
กราฟที่ 5.2 แสดง ค่า MAE Error ของภาพ 24 bit



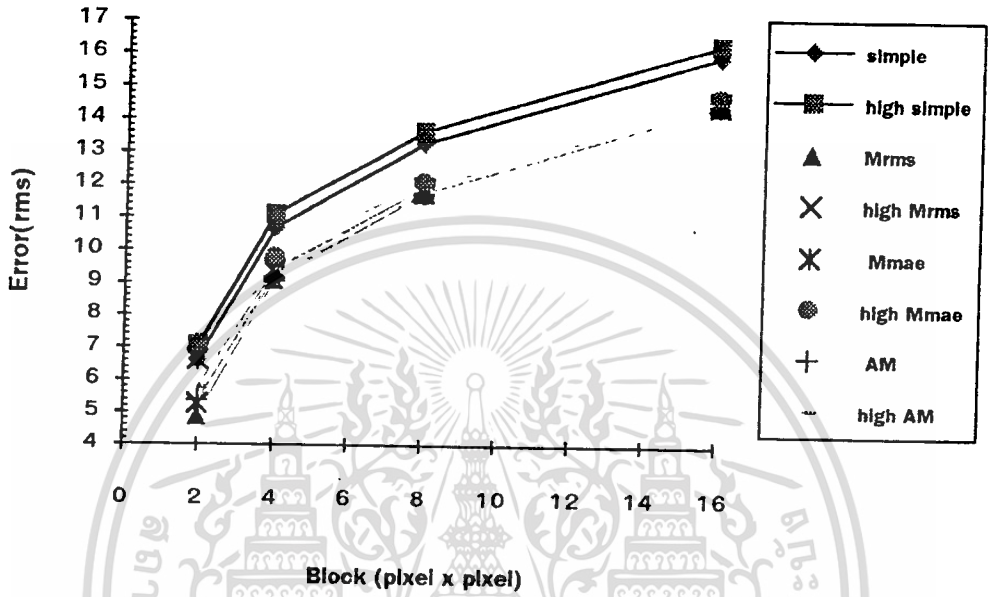
กราฟที่ 5.3 แสดง Signal to noise ของภาพ 24bit



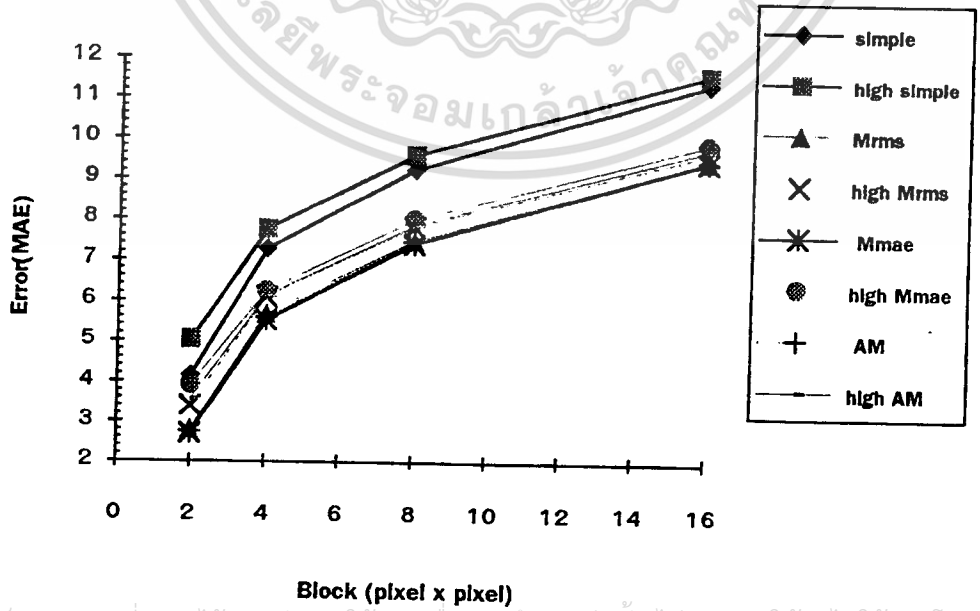
กราฟที่ 5.4 แสดง อัตราการลดข้อมูล ภาพ 24 bit



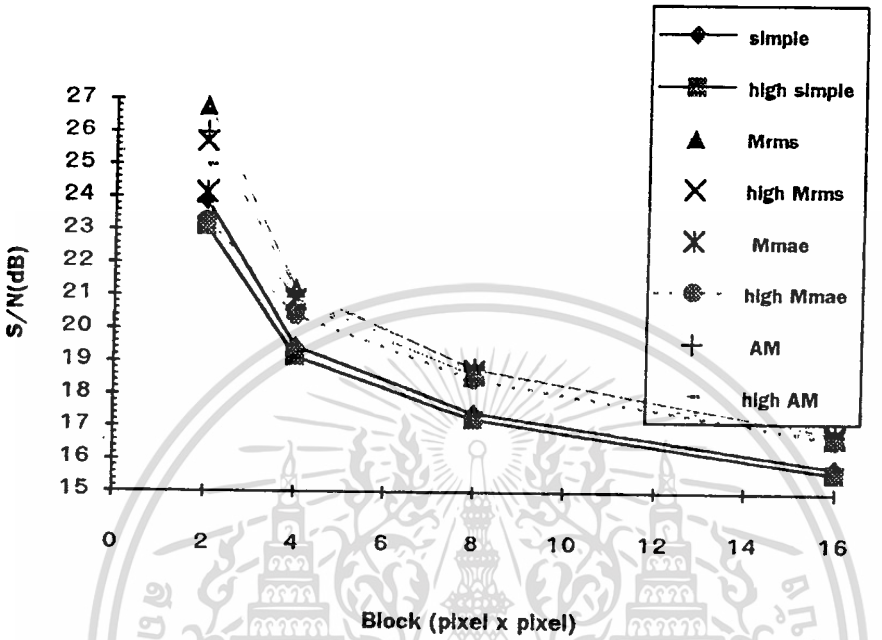
กราฟที่ 5.5 แสดง ค่า RMS Error ของภาพ Gray level



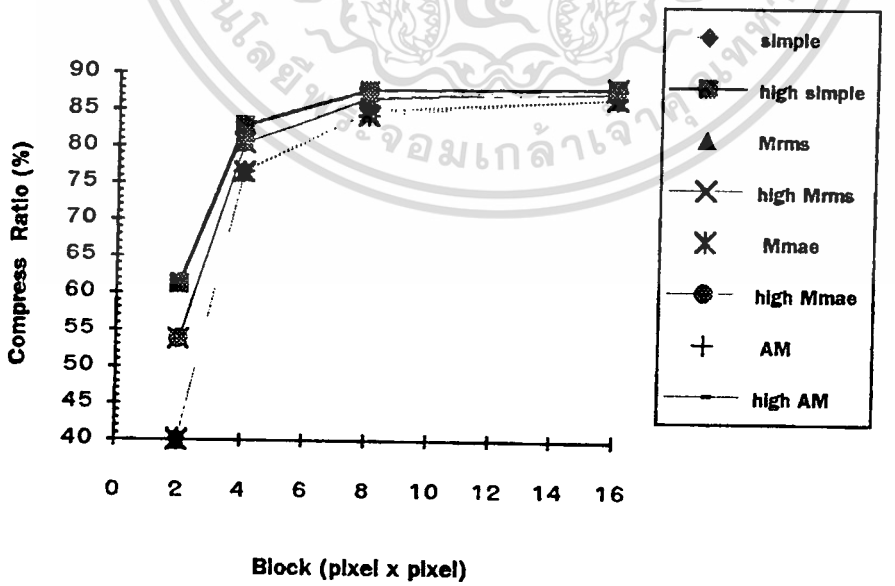
กราฟที่ 5.6 แสดง ค่า MAE Error ของภาพGray level

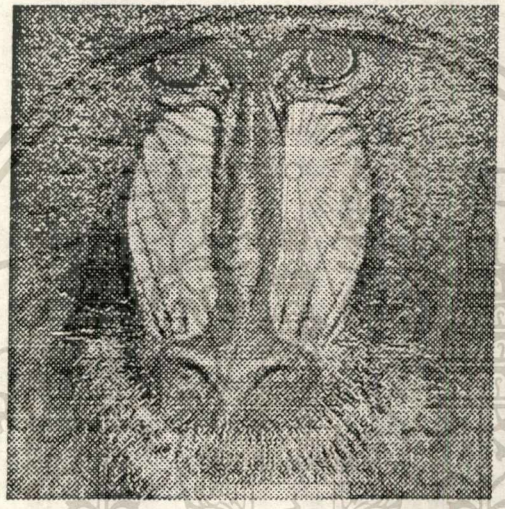


กราฟที่ 5.7 แสดง Signal to noise ของภาพ Gray level



กราฟที่ 5.8 แสดง อัตราการลดข้อมูล ภาพ Gray level

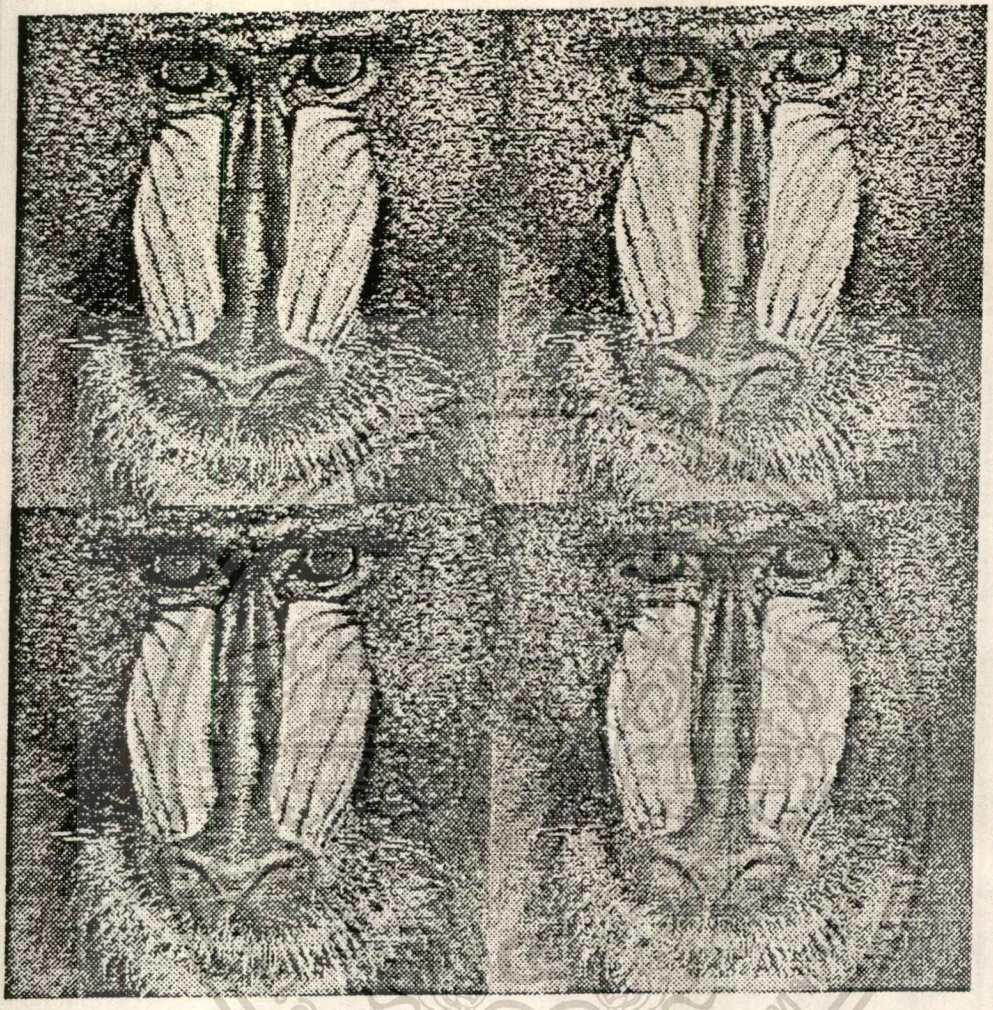




รูปที่ 5.1

ภาพต้นแบบ ชื่อภาพ Mandrill

เป็นภาพ Gray level 256 ระดับ



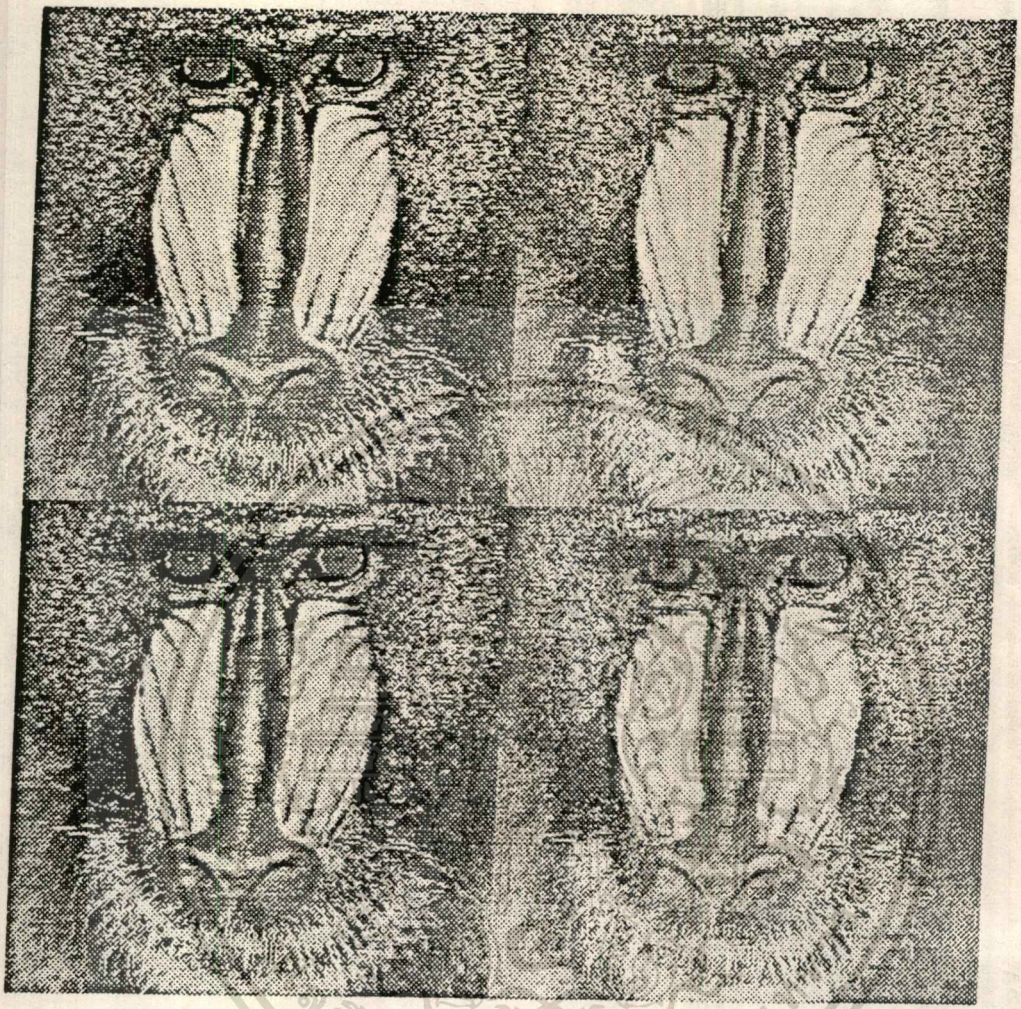
รูปที่ 5.2

(บนซ้าย) ใช้ BTC โดย Simple BTC Block 2 x 2 pixel^2

(บนขวา) ใช้ BTC โดย Simple BTC Block 8 x 8 pixel^2

(ล่างซ้าย) ใช้ BTC โดย Minimum Root Mean Square Error Block 2 x 2 pixel^2

(ล่างขวา) ใช้ BTC โดย Minimum Root Mean Square Error Block 8 x 8 pixel^2



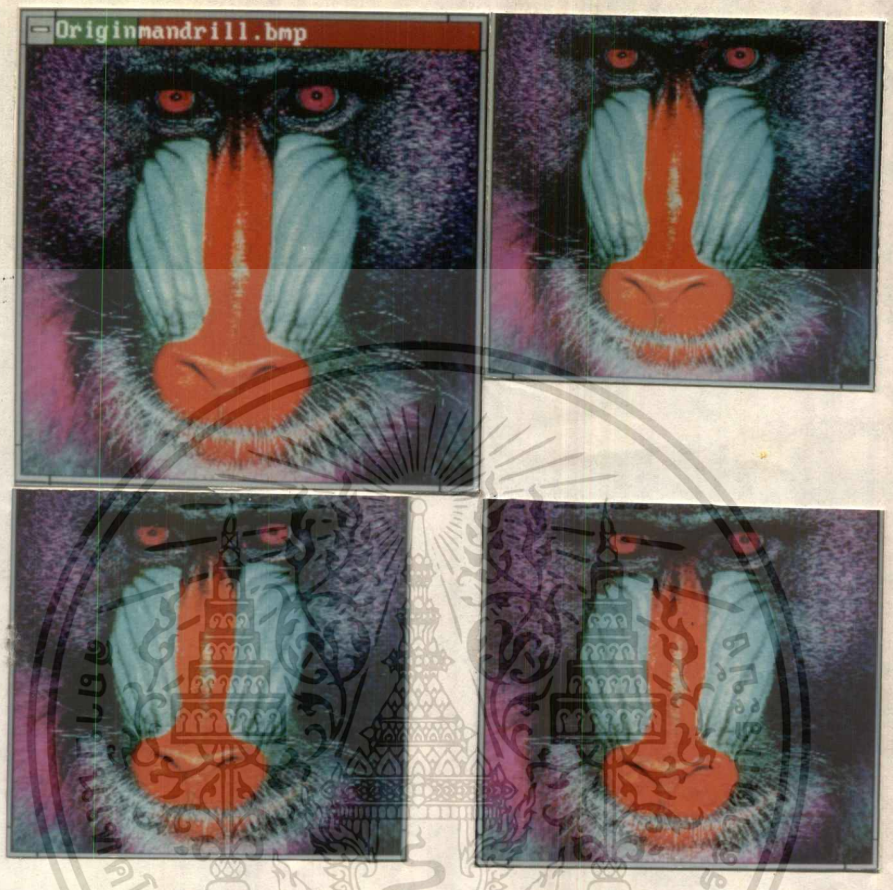
รูปที่ 5.3

(บนซ้าย) ใช้ BTC โดย MMAE Block 2 x 2 pixel^2

(บนขวา) ใช้ BTC โดย MMAE Block 8 x 8 pixel^2

(ล่างซ้าย) ใช้ BTC โดย Average Minimum Block 2 x 2 pixel^2

(ล่างขวา) ใช้ BTC โดย Average Minimum Block 8 x 8 pixel^2



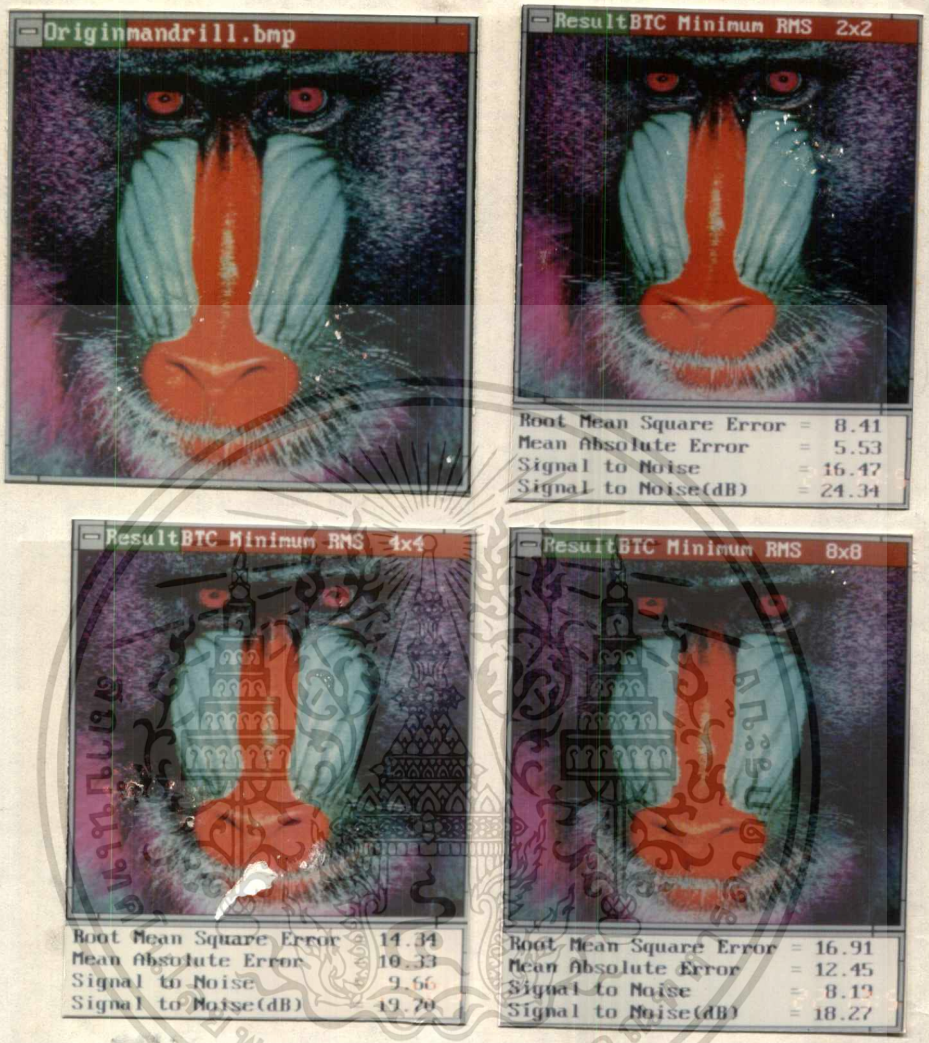
รูปที่ 5.4 แสดงผลของภาพที่ใช้ BTC โดยวิธี Simple BTC

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ Mandrill (24 Bit)

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 37.59 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 75.00 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.38 %



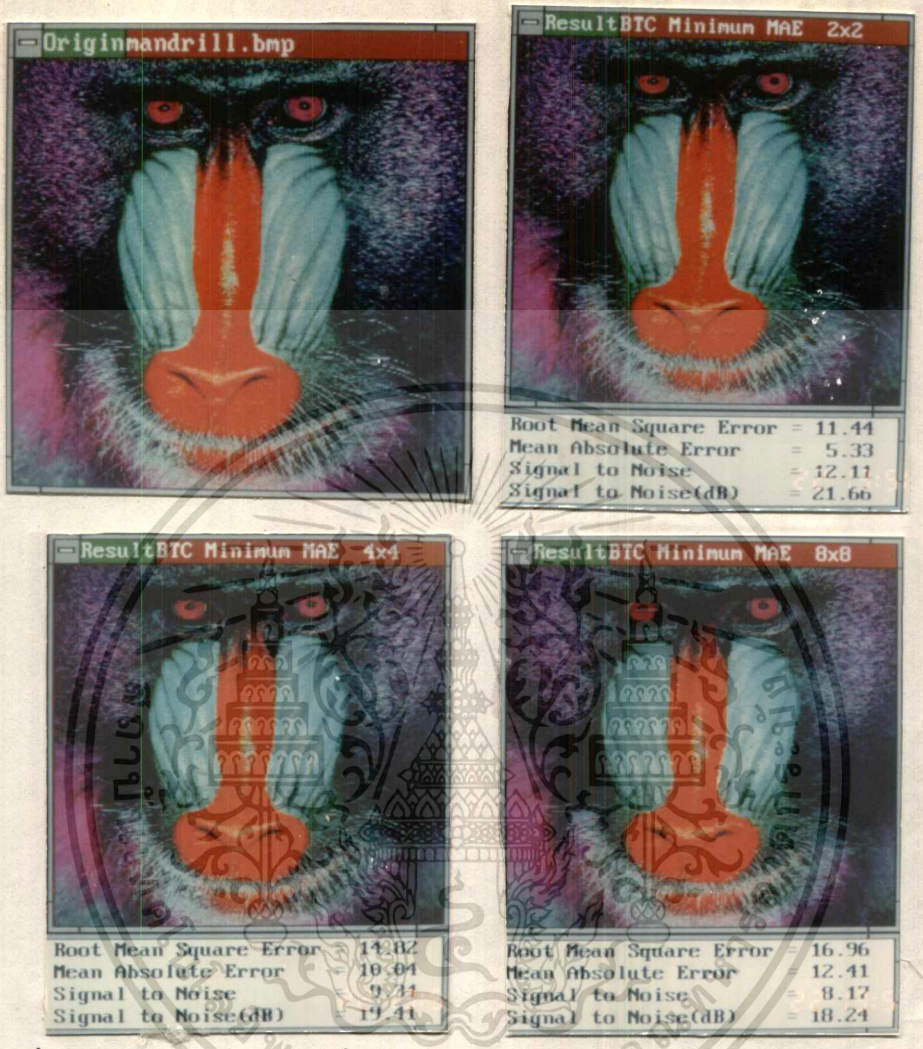
รูปที่ 5.5 แสดงผลของภาพที่ใช้วิธี **Minimum Root Mean Square Error BTC**

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **Mandrill (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 $pixel^2$ Compression Ratio = 37.59 %

(ล่างซ้าย) ใช้ Block 4 x 4 $pixel^2$ Compression Ratio = 75.00 %

(ล่างขวา) ใช้ Block 8 x 8 $pixel^2$ Compression Ratio = 84.38 %



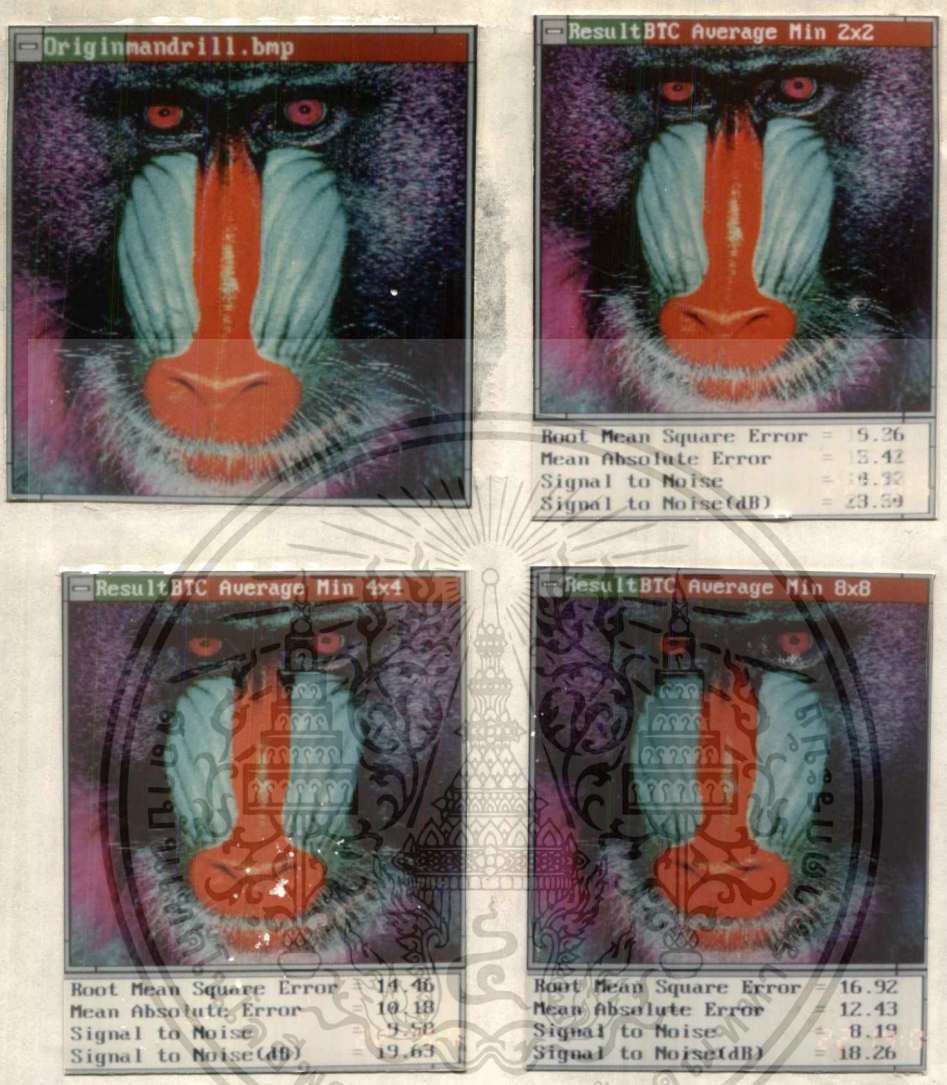
รูปที่ 5.6 แสดงผลของภาพที่ใช้วิธี **Minimum Absolute Error BTC**

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **Mandrill (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 37.59 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 75.00 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.38 %



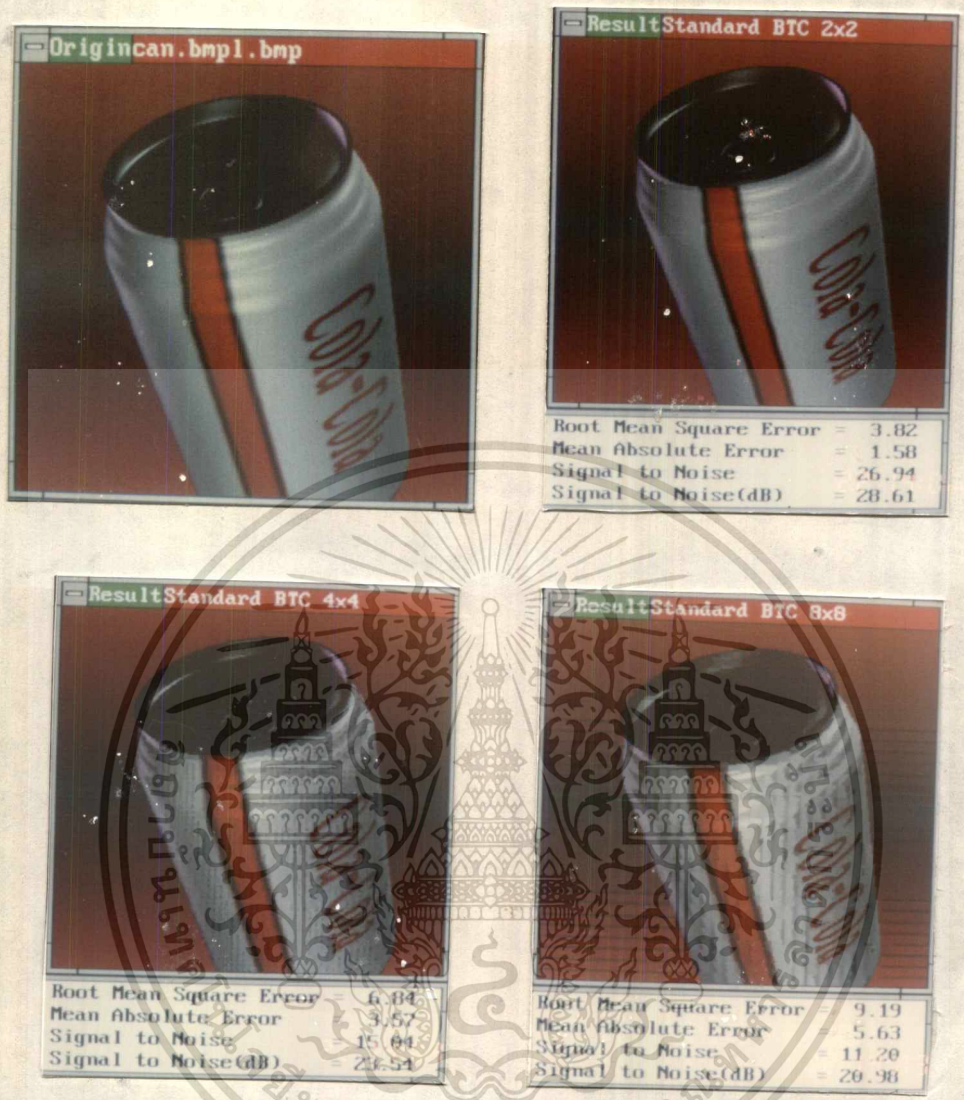
รูปที่ 5.7 แสดงผลของภาพที่ใช้วิธี Average Minimum of RMSE and MAE BTC

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **Mandrill (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 37.59 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 75.00 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.38 %



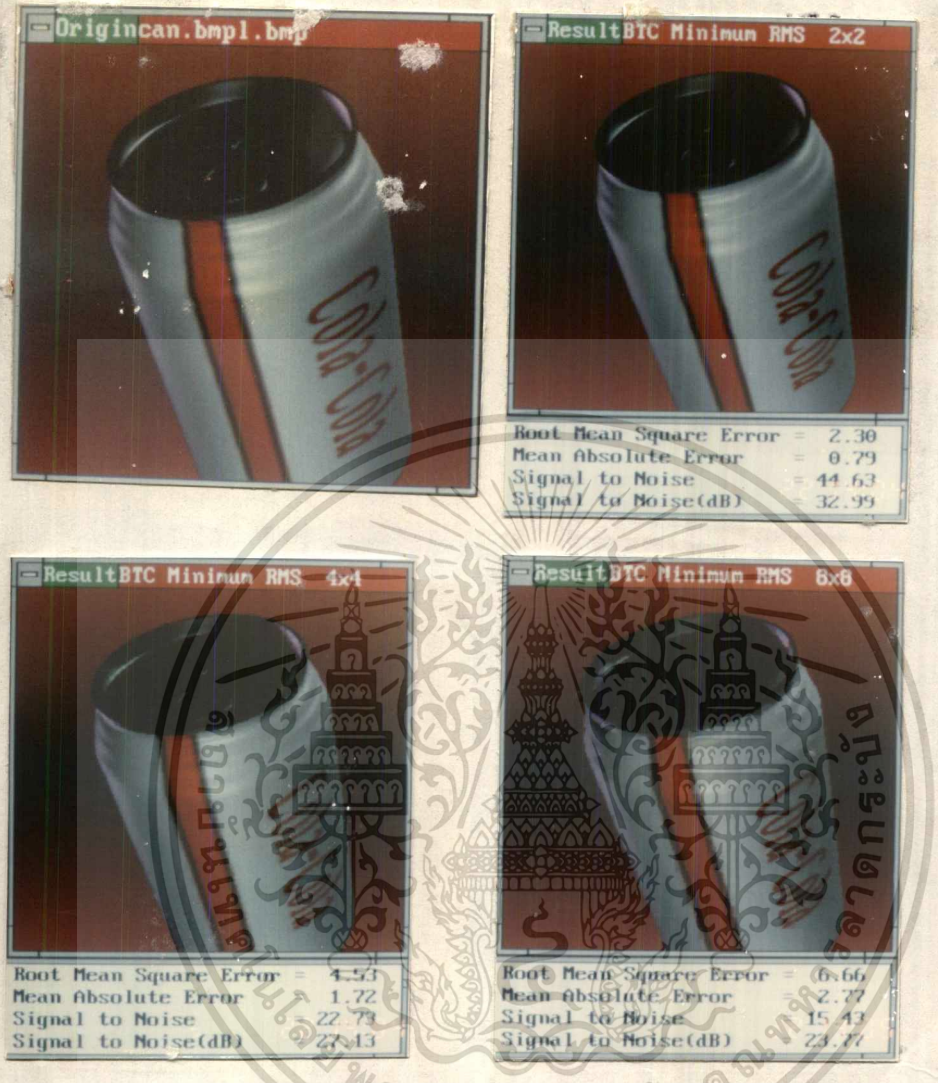
รูปที่ 5.8 แสดงผลของภาพที่ใช้วิธี **Simple BTC**

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **CAN (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 46.79 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 80.29 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.57 %



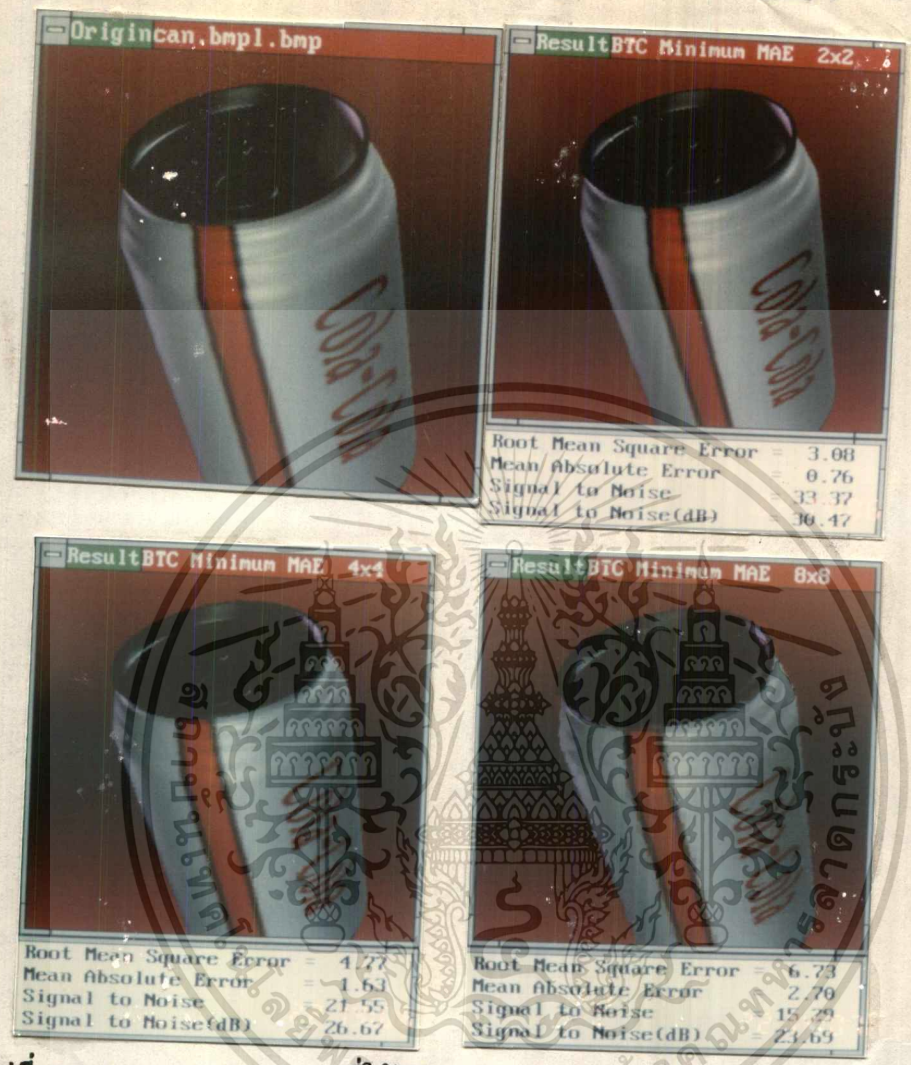
รูปที่ 5.9 แสดงผลของภาพที่ใช้วิธี **Minimum Root Mean Square Error BTC**

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **CAN (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 46.79 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 78.17 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.57 %



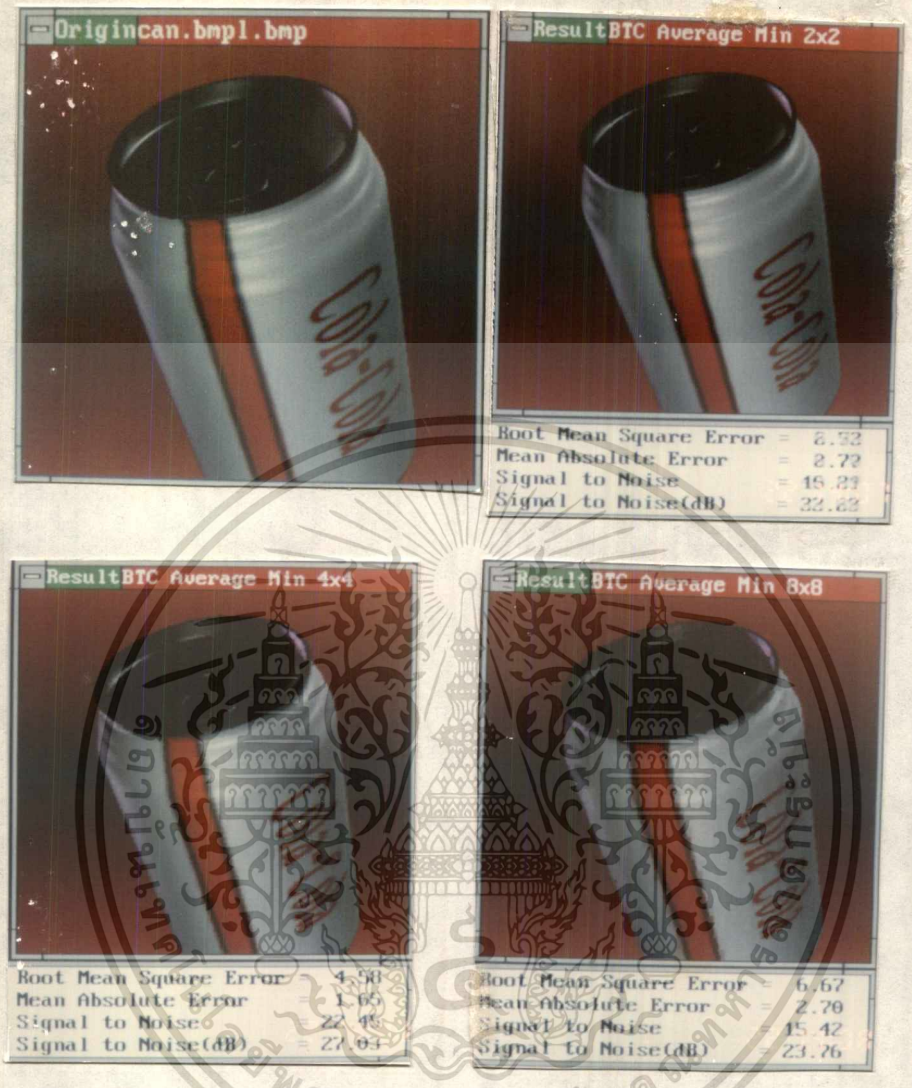
รูปที่ 5.10 แสดงผลของภาพที่ใช้วิธี Minimum Mean Absolute Error BTC

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ **CAN (24 Bit)**

(บนขวา) ใช้ Block 2 x 2 $pixel^2$ Compression Ratio = 46.79 %

(ล่างซ้าย) ใช้ Block 4 x 4 $pixel^2$ Compression Ratio = 78.17 %

(ล่างขวา) ใช้ Block 8 x 8 $pixel^2$ Compression Ratio = 84.57 %



รูปที่ 5.11 แสดงผลของภาพที่ใช้วิธี Average Minimum of RMSE and MAE BTC

(บนซ้าย) ภาพต้นแบบ ชื่อภาพ CAN (24 Bit)

(บนขวา) ใช้ Block 2 x 2 pixel^2 Compression Ratio = 46.79 %

(ล่างซ้าย) ใช้ Block 4 x 4 pixel^2 Compression Ratio = 78.17 %

(ล่างขวา) ใช้ Block 8 x 8 pixel^2 Compression Ratio = 84.57 %

วิเคราะห์ผลการทดลอง

1. ในเรื่องของคุณภาพของภาพที่ได้หลังจาก Decompress แล้ว พบว่า

- วิธี Simple BTC ให้คุณภาพต่ำสุด (จากการวัดค่า Error ทั้งสามวิธี)

- วิธี Minimum Root Mean Square Error จะให้ค่า Error ที่วัด โดยใช้วิธี Root Mean Square Error ต่ำสุด

- วิธี Minimum Mean Absolute Error จะให้ค่า Error ที่วัด โดยใช้วิธี Mean Absolute Error มีค่า ต่ำสุด

- วิธี Average Minimum of Root Mean Square Error and Mean Absolute Error จะให้ ค่าของ Error เฉลี่ยระหว่าง 2 วิธี ดังกล่าวคือ

จะได้ค่า Root Mean Square Error น้อยกว่า วิธี Minimum Minimum Mean Absolute Error แต่ มากกว่า วิธี Minimum Root Mean Square Error โดยมีแนวโน้มใกล้เคียงกับวิธี MRMSE มากกว่า

และจะได้ ค่า Mean Absolute Error ต่ำกว่า วิธี MRMSE แต่ มากกว่า วิธี MMAE โดยมีแนวโน้มใกล้เคียง กับวิธี MMAE มากกว่า

- เมื่อพิจารณาค่าของ Signal to Noise พบว่า จะได้ค่า S/N เรียงตามลำดับจากมาก ไป น้อย ดังนี้ 1. MRME 2.Average Minimum of MRME and MMAE 3.MMAE 4. Simple BTC

2. ความสามารถในการอัดข้อมูลภาพ เรียงตามลำดับได้ดังนี้ (จากมากไปน้อย) 1.Simple BTC 2.วิธี MRMSE และ วิธี MMAE (ประมาณว่าอัดข้อมูลได้เท่ากัน) 3.Average Minimum of RMSE and MAE

3. ความเร็ว พบว่า สามารถเรียงลำดับได้ดัง ต่อไปนี้ คือ (จากกค่าเฉลี่ย)

1. Minimum Root Mean Square Error ≈ 6.27 วินาที

2. Simple BTC ≈ 6.92 วินาที

3. Minimum Mean Absolute Error ≈ 8 วินาที

4. Average Minimum of Root Mean Squer and Mean Absolute Error.

ทั้งนี้ในเรื่องของความเร็วนี้ยังขึ้นอยู่กับเครื่องที่ใช้ RUN โปรแกรม การ์ดจอ และ วิธีการเขียนโปรแกรมอีกด้วย ซึ่งตัวเลขข้างบนนี้อยู่ในสภาวะที่ถือว่าเหมือนกัน ในการเปรียบเทียบ แต่ละวิธี

บทที่ 6

สรุปและวิจารณ์

การลดข้อมูลภาพ เป็นงานวิจัยที่ต้องใช้ความรู้ทางคณิตศาสตร์มาประยุกต์ใช้ในการเข้ารหัส ถอดรหัส การ Transform ภาพด้วยเทคนิคต่างๆ กัน มีความเหมาะสมกับภาพแต่ละชนิด ทำให้ได้อัตราการลดข้อมูล และ ความคลาดเคลื่อนที่แตกต่างกัน

จากการศึกษาวิจัย ทำให้ผู้ใช้สามารถเลือกเทคนิคที่เหมาะสมกับภาพที่ต้องการได้แล้วนำความรู้นั้น มาคิดค้นอัลกอริทึมที่เหมาะสม เพื่อนำมาโปรแกรมใช้ในทางปฏิบัติได้ทั้งภาพ ขาว-ดำ ภาพ Graylevel และภาพสี อีกทั้งเป็นแนวทางในการศึกษาเทคนิคที่ดียิ่งขึ้นไป จนสามารถนำไปใช้งานได้อย่างมีประสิทธิภาพ

สำหรับใน ปริญญานิพนธ์ ฉบับนี้ ได้ทำการศึกษา วิธีการลดข้อมูลภาพ โดยใช้ เทคนิคBlock Truncation ด้วยกัน 3 วิธี ด้วยกันคือ

1. Simple BTC
2. Minimum Root Mean Square Error
3. Minimun Mean Absolute Error
4. Average Mean of RMSE and MAE

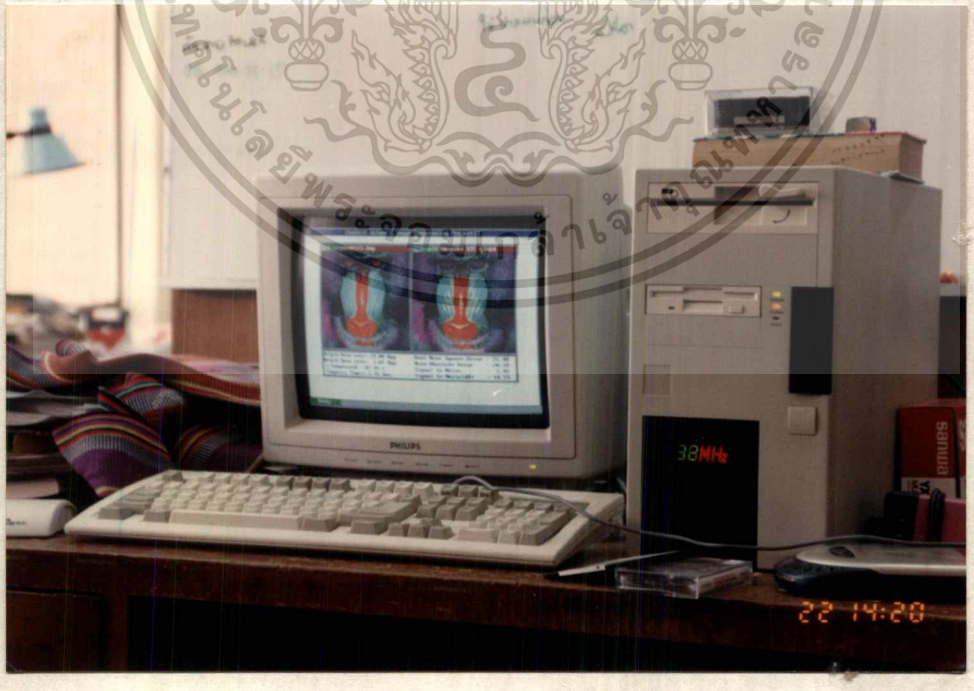
ซึ่งเป็นวิธีการที่น่าสนใจเป็นอย่างยิ่งเพราะให้คุณภาพของ ภาพ ดีมาก เมื่อเทียบกันวิธี Transform ซึ่งใช้หลักการตัดสัมประสิทธ์ ความถี่สูงทิ้ง ซึ่งจะทำให้ภาพ เบรอ ขาดความคมชัด

วิธี BTC นี้ยังสามารถลดขนาดลงไปได้อีก โดยนำเอา out put จาก BTC นี้ ไป Compress ด้วยวิธี Run length encoding หรือ วิธีอื่น ที่เป็น no-loss Compression

วิธีการนี้ สามารถนำไปประยุกต์ใช้ได้ดีในการสื่อสารข้อมูลภาพ ทั้งโทรศัพท์ภาพ และการส่งภาพนิ่งต่างๆ ได้เป็นอย่างดี

ข้อจำกัดอย่างยิ่งสำหรับการเริ่มต้นทำโครงงานนี้ก็คือ ผู้ทำโครงงานไม่มีความรู้ในด้านนี้มาก่อน จึงเสียเวลาอย่างมากในการเรียนรู้ นับตั้งแต่ วิธีการเขียน โปรแกรม ทาง Graphics ไปจนถึง Format ของภาพ ยังรวมไปถึงปัญหาทางด้าน ฮาร์ดแวร์ ทั้งที่ได้พยายามใช้ inturrupt rutine ของ Dos และ VIDIO BIOS แล้วแต่ก็ปรากฏว่า เครื่องที่ บางเครื่องก็ไม่ได้ทำให้ ใน โหมด กราฟิกความละเอียดสูง Support กับ inturrupt rutine ของ Dos จึงไม่สามารถ Run Programs ได้ โดยเฉพาะอย่างยิ่ง การประมวลผลภาพสี ซึ่งต้องใช้ VGA CARD 24 Bit 16.7 ล้านสี ซึ่งไม่สามารถหาคู่มือได้ ต้องใช้วิธี debug เอาเอง จึงจะทราบวิธีการใช้งาน

อย่างไรก็ตามในที่สุดแล้วโครงงานนี้ ก็ได้ลุล่วงไปตามเป้าหมาย ที่ได้วางแผนเอาไว้ แม้ว่าจะล้าช้ากว่าที่วางแผนไว้ก็ตาม



กิตติกรรมประกาศ

ปริญญานิพนธ์เรื่องนี้สำเร็จลุล่วงลงได้ ด้วยคำแนะนำ ของท่านอาจารย์มนัส สังวรศิลป์ อาจารย์ กิติพล ชิตสกุล อาจารย์ ประภากร สุวรรณะ อาจารย์ เทอดศักดิ์ ลิ่วหาทอง และอาจารย์อื่นๆ ที่ได้ให้ความช่วยเหลือ คำแนะนำ และ ประสิทธิประสาทวิชาให้ นอกจากนี้แล้วต้องขอบคุณ เพื่อนๆที่คอยให้กำลังใจและห่วงใยตลอดมา และที่ลืมเสียมิได้ ก็คือ คุณพ่อ คุณแม่ ซึ่งคอยเอาใจใส่ ช่วยเหลือมาโดยตลอด

หนังสืออ้างอิง

1. Edward J. Delp and O.Robert Mitcell, "Image Compression Using Block Truncation Coding", IEEE Transaction on communications, VOL.COM 27, 1979, 1335-1342.
2. N. Ahmed, T. Natarajan. and K.R.Rao,"Discrete cosine transform",IEEE Transation on communications, Vol.com 23, 1974 ,90 - 93.
3. W. Chen, C.H. Smith,and S.Fralick, "A fast computational algorithm for the discrete cosine transform", IEEE Tran. Commu.Technol. COM-25 1977, 1285-1291.
4. Tova F.Fliegel, "The Data Compression Book", Mark Nelson,347-308.

ภาคผนวก

(โปรแกรมลดข้อมูลภาพ)

```
//key.h
#define Esc_Key   0x11B
#define Enter_Key 0x1c0d
#define Home_Key 0x4700
#define End_Key 0x4f00
#define PageUp_Key 0x4900
#define PageDown_Key 0x5100
#define Left_Key 0x4b00
#define Right_Key 0x4d00
#define Up_Key 0x4800
#define Down_Key 0x5000
#define F1_Key 0x3b00
#define F2_Key 0x3c00
#define F3_Key 0x3d00
#define F4_Key 0x3e00
#define F5_Key 0x3f00
#define F6_Key 0x4000
#define F7_Key 0x4100
#define F8_Key 0x4200
#define F9_Key 0x4300
#define F10_Key 0x4400
#define Insert_Key 0x5200
#define Delete_Key 0x5300
#define BackSpace_Key 0x0e08
#define Alt_Q  0x1000
#define Alt_W  0x1100
#define Alt_E  0x1200
#define Alt_R  0x1300
#define Alt_T  0x1400
#define Alt_Y  0x1500
#define Alt_U  0x1600
#define Alt_I  0x1700
#define Alt_O  0x1800
#define Alt_P  0x1900
#define Alt_A  0x1e00
#define Alt_S  0x1f00
```

```
#define Alt_D  0x2000
#define Alt_F  0x2100
#define Alt_G  0x2200
#define Alt_H  0x2300
#define Alt_J  0x2400
#define Alt_K  0x2500
#define Alt_L  0x2600
#define Alt_Z  0x2c00
#define Alt_X  0x2d00
#define Alt_C  0x2e00
#define Alt_V  0x2f00
#define Alt_B  0x3000
#define Alt_N  0x3100
#define Alt_M  0x3200
#define Shift_F1 0x5400
#define Shift_F2 0x5500
#define Shift_F3 0x5600
#define Shift_F4 0x5700
#define Shift_F5 0x5800
#define Shift_F6 0x5900
#define Shift_F7 0x5a00
#define Shift_F8 0x5b00
#define Shift_F9 0x5c00
#define Shift_F10 0x5d00

#define Ctrl_F1 0x5e00
#define Ctrl_F2 0x5f00
#define Ctrl_F3 0x6000
#define Ctrl_F4 0x6100
#define Ctrl_F5 0x6200
#define Ctrl_F6 0x6300
#define Ctrl_F7 0x6400
#define Ctrl_F8 0x6500
#define Ctrl_F9 0x6600
#define Ctrl_F10 0x6700
```

```
#define Alt_F1 0x6800
#define Alt_F2 0x6900
#define Alt_F3 0x6a00
#define Alt_F4 0x6b00
#define Alt_F5 0x6c00
#define Alt_F6 0x6d00
#define Alt_F7 0x6e00
#define Alt_F8 0x6f00
#define Alt_F9 0x7000
#define Alt_F10 0x7100

#define Ctrl_PrtSc 0x7200
#define Ctrl_Left  0x7300
#define Ctrl_Right 0x7400
#define Ctrl_End   0x7500
#define Ctrl_PgDn  0x7600
#define Ctrl_Home  0x7700

#define Ctrl_Alt_1 0x7800
#define Ctrl_Alt_2 0x7900
#define Ctrl_Alt_3 0x7a00
#define Ctrl_Alt_4 0x7b00
#define Ctrl_Alt_5 0x7c00
#define Ctrl_Alt_6 0x7d00
#define Ctrl_Alt_7 0x7e00
#define Ctrl_Alt_8 0x7f00
#define Ctrl_Alt_9 0x8000
#define Ctrl_Alt_0 0x8100
#define Ctrl_Alt__ 0x8200
#define Ctrl_Alt_= 0x8300

#define Ctrl_PgUp 0x8400
```

```
/*      MSVGA.MOD  MODULE for set 640 x 480  Extend VGA mode      */
/*                      for  VESA and  ET-4000 chip and Cirrus     */
/*                      by  Chakkrit  Jaiman                       */
/*                      Dec,11,93                                  */

#include <alloc.h>
#include <dos.h>
#include <stdio.h>
#include <process.h>
#include <bios.h>
#include <mem.h>

#define _MSVGA__   16777216L
int _RED[256],_GREEN[256],_BLUE[256];
int Backgroundtext=0;

int OLDPAGE;
int VGATYPE;

#define VGAMODE 0x112 /* VESA MODE 640 x 480   16M color*/
#define CIRRUSMODE 0x71 /* CIRRUS MODE 640 x 480 16M color*/

char far information[512];
char far *FONTADDR;
void far *VIDEOADDR;
typedef struct {
    unsigned char Blue;
    unsigned char Green;
    unsigned char Red;
}COLOR_16_7M;

#define VIDEOMAXX 640
#define VIDEOMAXY 480
```

```
#define Max_X    639
#define Max_Y    479
#define palettemax 256
/*****************************************************************************************/

#define _BLACK_         0  /*......*/
#define _BLUE_          1  /*.....B*/
#define _GREEN_         2  /*....G.*/
#define _CYAN_          3  /*....GB*/
#define _RED_           4  /*...R..*/
#define _MAGENTA_       5  /*...R.B*/
#define _BROWN_         6  /*.g.R..*/
#define _LIGHTGREY_     7  /*...RGB*/
#define _DARKGREY_      8  /*rgb...*/
#define _LIGHTBLUE_     9  /*rgb..B*/
#define _LIGHTGREEN_    10  /*rgb.G.*/
#define _LIGHTCYAN_     11  /*rgb.GB*/
#define _LIGHTRED_      12  /*rgbR..*/
#define _LIGHTMAGENTA_ 13  /*rgbR.B*/
#define _YELLOW_        14  /*rgbRG.*/
#define _WHITE_         15  /*rgbRGB*/
#define _PINK_          16  /*250 145 190*/
#define _VIOLET_        17  /*185 100 255*/
#define _GOLD_          18  /*255 190 70 */
#define _LIGHTYELLOW_ 19 /* 255 255 232*/


const int _REDTEXT[20]=
                {
                    0,0,0,0,191,191,191,192,
                    128,64,64,64,255,255,255,255,
                    250,185,255,255
                };
```

```
const int _GREENTEXT[20]=
                {
                    0,0,191,191,0,0,64,192,
                    128,64,255,255,64,64,255,255,
                    145,100,190,255
                };
int _BLUETEXT[20]=
                {
                    0,191,0,191,0,191,0,192,
                    128,255,64,255,64,255,64,255,
                    190,255,70,232
                };
char txmp[80];


#define Putpixel(x,y,color) Directpixel(x,y,_RED[color],_GREEN[color],_BLUE[color])
#define Putpixelg(x,y,color) Directpixel(x,y,color,color,color)
#define PutpixelText(x,y,colortext) Directpixel(x,y,_REDTEXT[colortext],_GREENTEXT[colortext],_BLUETEXT[colortext])
#define outtextcl(C,L,Color,text) outtextxy(C*8,L*16,Color,text)
#define getch_(c,l,color) (int)((getkey(c,l,color))&0xff)
#define setvideomode(mode) {\
            _AH=_AH^_AH;\
            _AL=(char)mode;  \
            geninterrupt(0x10);\
            }
#define getpixelg(x,y) (int)(Getpixel(x,y)&0xff)
#define BackGround(color) Backgroundscreen(_REDTEXT[color],_GREENTEXT[color],_BLUETEXT[color])
#define etbankread(bankno) outp(0x3cd,(bankno)<<4)
#define etbankwrite(bankno) outp(0x3cd,bankno)
#define cirrusread(bankno) outport(0x3ce,(unsigned)(((bankno<<12)&0xf000)|0x09))
#define cirruswrite(bankno) cirrusread(bankno)
```

```
#define _setbank(___i) \
        _DX=(___i); \
        _AH=0x4f; \
        _AL=0x05; \
        _BH=0x00; \
        _BL=0x00;\
        geninterrupt(0x10);

#define vebankread(bankno) _setbank((bankno)<<4)
#define vebankwrite(bankno) vebankread(bankno)

/*****************************************************************/

int   opengraph(void);
void closegraph(void);
void Setdac(int,int,int,int);
char getmode(void);
void _fastcall PutSomePixel(int ,int ,int ,int);
void _fastcall Directpixel(int ,int ,int ,int );
void _fastcall lineDirectpixel(int x,int y,int pix,char far *buf);
void _fastcall Backgroundscreen(int Rcolor,int Gcolor,int Bcolor);
long _fastcall Getpixel(int x,int y);
void Setbankread(int x);
void Setbankwrite(int x);
 int Getscrversion(void);
int Getscrtype(void);
void Gbox(int ,int ,int ,int );
long _fastcall Gethlinesize(int ,int );
void _fastcall Gethline(int ,int ,void far *);
void _fastcall Puthline(int ,int ,int ,void far *);
void outchar(int,int,int,char);
void outtextxy(int,int,int,char *);
void _fastcall del(int,int,int);
```

```
void ffloodput(FILE *,int);
void ffloodget(FILE *,int);
long Getimagesize(int,int,int,int);
long Getimage(int,int,int,int,void far *);
void Putimage(int,int,int,int,void far *);
void _fastcall putline_(int,int,int ,int ,int );
void _fastcall putline(int,int,int,int,int);
void _fastcall image_windows(int ,int,int ,int ,int );
int gets_(int,int,int,int ,char *);
unsigned int getkey(int ,int,int);

class Extra_mem
{
 private:

          char  far **sbuf;
          char far *lbuf;
          int oldpage;

public:

          Extra_mem();

          void Lbuf(char mes,unsigned add);
          char Lbuf(unsigned add);
          void Sbuf(char mes,unsigned addl,unsigned addc);
          char Sbuf(unsigned addl,unsigned addc);

};
```

```
class Picture_menager
{
 private:
            int xref;
            int yref;
            int width;
            int high;
            int currentX;
            int currentY;
            int Xinpage;
            int Yinpage;
            int page;

           COLOR_16_7M  far **pixel;
public:
     COLOR_16_7M Colors;
     Picture_menager();
     Picture_menager(int w,int h,int x=0,int y=0);
     void Origin(int x,int y);
     void Size(int w,int h);
     //setpage;
     void Setpage(int x)
         {
         if(OLDPAGE!=x){
         cirrusread(x);
         OLDPAGE=x;
          }
        }

    void  Pixel(int x,int y,int color);
    void  Pixel(int x,int y,unsigned char R,unsigned char  G,unsigned char B);
    void  SomePixel(int x,int y,int R,int G,int B);
    void  Pixel(int x,int y,char type,unsigned char color);
```

```
        COLOR_16_7M  Pixel(int x,int y);
        int   Pixel(int x,int y,char type);
        COLOR_16_7M far * LinePixel(unsigned x,unsigned y);
//       int  PutLine();
};

/*****************************************************************/

void Setbankread(int x)
{
  OLDPAGE = x;
  switch(VGATYPE)
  {
   case 0 . cirrusread(x);break;
   case 1 : etbankread(x);break;
   default: vebankread(x);break;
  }
}

void Setbankwrite(int x)
{
  OLDPAGE = x;
 switch(VGATYPE)
  {
  case 0 : cirruswrite(x);break;
  case 1 : etbankwrite(x);break;
  default: vebankwrite(x);break;
  }
}
```

```
/*******************************MODULE*********************************/

void Setdac(int index,int R,int G,int B)
/* set color for using color in module */
{
 _RED[index]  = R;
 _GREEN[index]= G;
 _BLUE[index] = B;
}

char getmode(void)
{
 return(peekb(0,0x0449));
}

int opengraph(void)
/*initial graph for VGA and SUPERVGA mode */
{
// get poiter for character  Table outtext
 _AH = 0x11,
 _AL = 0x30;
// _BH = 0x07;//Load 9x16 text
// _BH = 0x03;//Load 8x8 text
 _BH = 0x06;//Load 8x16 text
 geninterrupt(0x10);


FONTADDR = (char *)MK_FP(_ES,_BP);
VIDEOADDR = (void *)MK_FP(0xa000,0);
```



```
 if(VGATYPE!=0){
 _ES = FP_SEG(information);
 _DI = FP_OFF(information);
```

```
    _AH = 0x4f;
    _AL = 0x00;
    geninterrupt(0x10);
    if(_AL != 0x4f)
     {
      _AH = 0;
      _AL = 3;
      geninterrupt(0x10);
      return(0);
     }

    _BX = VGAMODE;
    _AH = 0x4f;
    _AL = 0x02;
    geninterrupt(0x10);

    if(_AL != 0x4f)
     {
      _AH = 0;
      _AL = 3;
      geninterrupt(0x10);
      return(0);
     }
  }else
   {
    asm{
          mov AH,0x00
          mov AL,CIRRUSMODE
          int 0x10
        }
   }
  Setbankwrite(0);
  return(1);
```

```
}

/* Close graphics*/
void closegraph(void)
{
 _AH = 0;
 _AL = 3;
 geninterrupt(0x10);
}
/* put pixel in specific color on screen*/

long _l;
void _fastcall Directpixel(int x,int y,int R,int G,int B)
{
 if(OLDPAGE !=((_l=(((long)(y))<<11)+(long)(x)*(long)3)>>16))
   { Setbankwrite(OLDPAGE = (_l>>16));}
 pokeb(0xa000,_l&0xffff,(char)(B));
 pokeb(0xa000,(_l&0xffff)+(long)1,(char)(G));
 pokeb(0xa000,(_l&0xffff)+(long)2,(char)(R));
}
void _fastcall lineDirectpixel(int x,int y,int pix,char far *buf)
{
 if(OLDPAGE !=((_l=(((long)(y))<<11)+(long)(x)*(long)3)>>16))
   { Setbankwrite(OLDPAGE = (_l>>16));}
 movedata(FP_SEG(buf),FP_OFF(buf),0xa000,_l&0xffff,pix*3*sizeof(char));
}

void _fastcall PutSomePixel(int x,int y,int R,int G,int B)
{
 if(OLDPAGE !=((_l=(((long)(y))<<11)+(long)(x)*(long)3)>>16))
   { Setbankwrite(OLDPAGE = (_l>>16));}
 if(B<256)
```

```
    pokeb(0xa000,_I&0xffff,(char)(B));

  if(G<256)
    pokeb(0xa000,(_I&0xffff)+(long)1,(char)(G));
  if(R<256)
    pokeb(0xa000,(_I&0xffff)+(long)2,(char)(R));
}
void _fastcall Backgroundscreen(int Rcolor,int Gcolor,int Bcolor)
{
   int i;
   char far *image;
   image=new char far[1920];
   for(i=0;i<=Max_X;i++)
     {
      image[i*3]=(char)Bcolor;
      image[i*3+1]=(char)Gcolor;
      image[i*3+2]=(char)Rcolor;
     }

   for(i=0;i<=Max_Y;i++)
    {
       Puthline(0,Max_X,i,image);
    }
}

long _fastcall Getpixel(int x,int y)
{
 long i;
 long l,h;

   if(OLDPAGE !=((i=(((long)(y))<<11)+(long)(x)*(long)3)>>16))
    {
     Setbankread(OLDPAGE = (i>>16));
    }
```

```
    l = (unsigned int)peek(0xa000,i&0xffff);
    h = ((unsigned int)peek(0xa000,(i&0xffff)+(long)2))&0x000000ff;
    if(OLDPAGE !=((i=(((long)(y))<<11)+(long)(x)*(long)3)>>16))
    {
     Setbankwrite(OLDPAGE = (i>>16));
    }
    return((h<<16)|l);

}
/*****************************************************************/
int Getscrversion(void)
{
 return 10;
}

int Getscrtype(void)
{
 return 2;
}

void Gbox(int x1,int y1,int x2,int y2,int index)
{
 int x,y;
 char grest[2000];
 for(x=x1;x<=x2;x++)
   PutpixelText(x,y1,index);
    Gethline(x1,x2,y1,(void far *)grest);
for(y=y1+1;y<=y2;y++)
   Puthline(x1,x2,y,grest);
}
```

```
/*****************************************************************/

long _fastcall Gethlinesize(int x1,int x2)
{
 return((long)((x2-x1+1)*3));
}

void _fastcall Gethline(int x1,int x2,int y,void far *image)
{
 long offset = (long)x1*(long)3+((long)y<<11);

 Setbankread(offset>>16);
 movedata(0xa000,offset&0xffff,FP_SEG(image),FP_OFF(image),(x2-x1+1)*3);
}

void _fastcall Puthline(int x1,int x2,int y,void far *image)
{
 long offset = (long)x1*(long)3+((long)y<<11);

 Setbankwrite(offset>>16);
 movedata(FP_SEG(image),FP_OFF(image),0xa000,offset&0xffff,(x2-x1+1)*3);
}
void _fastcall putline_(int xi,int yi,int xf,int yf,int color)
{
 int i;

 if(yi==yf)
 {
   if((yi<0)||(yi>(Max_Y)))return;
   if(xi<0)xi=0;else if(xi>(Max_X))xi=Max_X;
   if(xf<0)xf=0;else if(xf>(Max_X))xf=Max_X;
   if(xi<=xf)
    {
```

```
        for(i=xi;i<=xf;i++)PutpixelText(i,yi,color);
        }else{

                for(i=xi;i>=xf;i--)PutpixelText(i,yi,color);
                }
    }
    if(xi==xf)
    {
     if((xi<0)||(xi>(Max_X)))return;
     if(yi<0)yi=0;else if(yi>(Max_Y))yi=Max_Y;
     if(yf<0)yf=0;else if(yf>(Max_Y))yf=Max_Y;


      if(yi<=yf)
        {
            for(i=yi;i<=yf;i++)PutpixelText(xi,i,color);
        }else{
                for(i=yi;i>=yf;i--)PutpixelText(xi,i,color);
                }
    }
}


void _fastcall image_windows(int x1,int y1,int x2,int y2,int depth,int color)
{
   int  i;
   for(i=0;i<depth;i++)
      {
       putline_(x1,y1-i,x2+depth-1,y1-i,color);
       putline_(x2+i,y1,x2+i,y2+depth-1,color);
       putline_(x2,y2+i,x1-depth+1,y2+i,color);
       putline_(x1-i,y2,x1-i,y1-depth+1,color);
      }
```

```
}

/*****************************************************************************/

void outchar(int x,int y,int indext,char character)
{
 char far *af;int a;

 af = FONTADDR + (unsigned int)character*0x0010;

 for(a = 0;a<16;a++)
  {
   if(((*(af)>>7)&0x01)!=0)
     PutpixelText(x,y+a,indext);
    else
     PutpixelText(x,y+a,Backgroundtext);
  if(((*(af)>>6)&0x01)!=0)
     PutpixelText(x+1,y+a,indext);
    else
     PutpixelText(x+1,y+a,Backgroundtext);
  if(((*(af)>>5)&0x01)!=0)
     PutpixelText(x+2,y+a,indext);
    else
     PutpixelText(x+2,y+a,Backgroundtext);
  if(((*(af)>>4)&0x01)!=0)
     PutpixelText(x+3,y+a,indext);
    else
     PutpixelText(x+3,y+a,Backgroundtext);
  if(((*(af)>>3)&0x01)!=0)
     PutpixelText(x+4,y+a,indext);
    else
```

```
            PutpixelText(x+4,y+a,Backgroundtext);
          if(((*(af)>>2)&0x01)!=0)
            PutpixelText(x+5,y+a,indext);

           else
            PutpixelText(x+5,y+a,Backgroundtext);
          if(((*(af)>>1)&0x01)!=0)
            PutpixelText(x+6,y+a,indext);
           else
            PutpixelText(x+6,y+a,Backgroundtext);
          if(((*(af))&0x01)!=0)
            PutpixelText(x+7,y+a,indext);
           else
            PutpixelText(x+7,y+a,Backgroundtext);

        af++;
      }
    }


    /*
    void outchar(int x,int y,int indext,char character)
    {
     char far *af;
     int a,i,bit=0;
     int temp;
    // y=y+16;
    af = FONTADDR + (unsigned int)character*0x0010;
    temp=*(af);
    temp<<=1;
    for(a = 0;a<16;a++)
     {
    //    temp=*(af);
```

```
    //    temp <<= 1;
    //    x+=16;
        for(i=7;i>=0;i--)
        {

         if(bit==8){
          temp=*(++af);
          temp<<=1;
          bit=0;
         }
         if(((temp>>=1)&0x01)!=0)
         PutpixelText(x+i,y+a,indext);
         bit++;
      }
     }
}
 */


void outtextxy(int x,int y,int index,char *stringin)
{
 char *strin;int a=x;
 strin =stringin;
 while((*(strin)!=0)&&(a<VIDEOMAXX))
  {
   outchar(a,y,index,*(strin));
   a+=8;
   strin++;
  }
}

void ffloodput(FILE *ffile,int bank)
{
 Setbankwrite(bank);
```

```
    fread(VIDEOADDR,16384,4,ffile);
  }

  void ffloodget(FILE *ffile,int bank)

  {
   Setbankread(bank);
   fwrite(VIDEOADDR,16384,4,ffile);
   Setbankwrite(bank);
  }

  long Getimagesize(int x1,int y1,int x2,int y2)
  {
   return(Gethlinesize(x1,x2)*(long)(y2-y1+1));
  }

  long Getimage(int x1,int y1,int x2,int y2,void far *image)
  {
   int a,b;
   long ssize = Gethlinesize(x1,x2);
   b=y2-y1+1;
   for(a=0;a<b;a++)
     Gethline(x1,x2,a+y1,(char far *)image+(ssize*(long)a));
     Setbankwrite(0);
     return(ssize*(long)b);
  }

  void Putimage(int x1,int y1,int x2,int y2,void far *image)
  {
   int a,b;
   long ssize = Gethlinesize(x1,x2);
   b = y2-y1+1;
   for(a=0;a<b;a++)
```

```
        Puthline(x1,x2,a+y1,(char far *)image+(ssize*(long)a));
      Setbankwrite(0);
     }

    void _fastcall del(int x,int y,int ch)

 {
  int temp,xi,yi;
 // x=c*8;
 // y=l*16;

  for(temp=0;temp<ch;temp++)
   {
     for(yi = 0;yi<16;yi++)
      {
          for(xi = 0;xi<8;xi++)
            PutpixelText(x+xi+temp,y+yi,Backgroundtext);
      }
   }
}

int gets_(int x,int y,int colortext,int background,char *s)
{
 int  ctemp;
 char temp[129];
 unsigned int readin;
 int v,h,ct;
// x=c*8;
// y=l*16;
 ct=0;
 ctemp=Backgroundtext;
 Backgroundtext=background;
```

```
for(v=0;v<129;v++)
  temp[v]=0;
for(v=0;v<16;v++)
  {
   PutpixelText(x,y+v,colortext);
   PutpixelText(x+1,y+v,colortext);
  }
while( ( (readin=bioskey(0) ))!=Enter_Key)
  {
   if( (readin&0xff)<0x20)
    {
        switch(readin)
         {
          case Left_Key : if(ct>0)
                           {
                            outchar((ct*8)+x,y,colortext,*(temp+ct));
                            ct--;
                           }
                          break;
          case Right_Key : if(ct<(VIDEOMAXX/8)-1)
                           {
                            outchar((ct*8)+x,y,colortext,*(temp+ct));
                            ct++;
                           }
                          break;
          case Esc_Key  : return(-1);/* cancel get string*/
          case Delete_Key : for(h=ct;h<127;h++)/*del char*/
                             *(temp+h)=*(temp+h+1);
                           *(temp+127)=0x0;
```

```
                    del((x+ct*8+8),(y),127-ct+1);
                    outtextxy(x+ct*8,y,colortext,temp+ct);
                    break;
        case BackSpace_Key : if(ct>0)  /*Backspace*/
                    {
                     ct--;

                    for(h=ct;h<127;h++)/*del char*/
                      *(temp+h)=*(temp+h+1);
                    *(temp+127)=0x0;
                    del((x+ct*8),(y),127-ct);
                    outtextxy(x+ct*8,y,colortext,temp+ct);
                    }
                    break;
        }
    }
   else
    {
        for(h=126;h>=ct;h--)
          *(temp+h+1)=*(temp+h);
        *(temp+ct)=(unsigned char)(readin&0xff);
        outtextxy((x+ct*8),(y),colortext,temp+ct);
        ct++;
        }


   for(v=0;v<16;v++)
   {
    PutpixelText((ct)*8+x,y+v,colortext);
    PutpixelText(((ct)*8)+x+1,y+v,colortext);
   }
   }
   *(temp+ct+1)=0;
```

```
        sprintf(s,"%s",temp);
        Backgroundtext=ctemp;
        return(0);
    }

    unsigned int getkey(int c,int l,int color)

{
    unsigned int key;
    key=bioskey(0);
    if((key&0xff))
      {
       outchar(c*8,l*16,color,key&0xff);
      }
    return(key);
}

Extra_mem::Extra_mem()
{
            char far *address;
            int i;
            address=(char *)MK_FP(0xa000,0x77f);
            lbuf=(char *)MK_FP(0xA000,0x0000);
            sbuf=new char * [32];
            sbuf[0]=address;
            for(i=1;i<32;i++)
            sbuf[i]=(address+=2048);
```

```
}

void Extra_mem::Lbuf(char mes,unsigned add)
{
        oldpage=OLDPAGE;
        Setbankwrite(15);
        lbuf[add]=mes;

        Setbankwrite(oldpage);
}

char Extra_mem::Lbuf(unsigned add)
{
        char temp;
        oldpage=OLDPAGE;
        Setbankread(15);
        temp=lbuf[add];
        Setbankread(oldpage);
        return(temp);

}

void Extra_mem::Sbuf(char mes,unsigned addl,unsigned addc)
{
        int raddl,page;
        if(addl<32)sbuf[addl][addc]=mes;
        else{
            page=addl/32;
            raddl=addl%32;
            Setbankread(page);
            sbuf[raddl][addc]=mes;
        }

}
```

```
char Extra_mem::Sbuf(unsigned addl,unsigned addc)
{
        int raddl,page;
        if(addl<32)return(sbuf[addl][addc]);
        else{
            page=addl/32;
            raddl=addl%32;
            Setbankread(page);
            return( sbuf[raddl][addc]);
        }
}

Picture_menager::Picture_menager()
{
  unsigned address;
  int i;
        pixel=new COLOR_16_7M * [32];
        address=0x0000;
        pixel[0]=(COLOR_16_7M  *)MK_FP(0xa000,0x0000);
        for(i=1;i<32;i++)
        pixel[i]=(COLOR_16_7M  *)MK_FP(0XA000,address+=2048);
        xref=0;
        yref=0;
        width=640;
        high=480;
        currentX=0;
        currentY=0;
        Xinpage=0;
        Yinpage=0;
        page=0;
```

```
}

Picture_menager::Picture_menager(int w,int h,int x,int y)
{
  unsigned address;
  int i;
        pixel=new COLOR_16_7M * [32];
        address=0x0000;

        pixel[0]=(COLOR_16_7M  *)MK_FP(0xa000,0x0000);
        for(i=1;i<32;i++)
        pixel[i]=(COLOR_16_7M  *)MK_FP(0XA000,address+=2048);
        xref=x;
        yref=y;
        width=w;
        high=h;
        currentX=x;
        currentY=y;
        Xinpage=x;
        Yinpage=y%32;
        page=y>>5;
}


void Picture_menager::Size(int w,int h)
{
 width=w;
 high=h;
}

void Picture_menager::Origin(int x,int y)
{
 xref=x;
 yref=y;
```

```
  Xinpage=x;
  Yinpage=y%32;
  page=y>>5;
  currentX=xref;
  currentY=yref;
 }

 //write pixel by  Table
```

```
void  Picture_menager::Pixel(int x,int y,int color)
{
 Picture_menager::currentX=x+xref;
 Picture_menager::currentY=y+yref;
 Picture_menager::Xinpage=Picture_menager::currentX;
 Picture_menager::Yinpage=Picture_menager::currentY%32;
 page=currentY>>5;
 Setpage(page);
 pixel[Yinpage][Xinpage].Red=(unsigned char)_REDTEXT[color];
 pixel[Yinpage][Xinpage].Green=(unsigned char)_GREENTEXT[color];
 pixel[Yinpage][Xinpage].Blue=(unsigned char)_BLUETEXT[color];

}

void  Picture_menager::Pixel(int x,int y,unsigned char R,unsigned char G,unsigned char B)
{
 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;
 page=currentY>>5;
 Setpage(page);
 pixel[Yinpage][Xinpage].Blue = B;
 pixel[Yinpage][Xinpage].Green= G;
```

```
 pixel[Yinpage][Xinpage].Red  = R;


}
void  Picture_menager::SomePixel(int x,int y,int R,int G,int B)
{
 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;




 page=currentY>>5;
 Setpage(page);

 if(B<256)
 pixel[Yinpage][Xinpage].Blue = (unsigned char)B;
 if(G<256)
 pixel[Yinpage][Xinpage].Green= (unsigned char)G;
 if(R<256)
 pixel[Yinpage][Xinpage].Red  = (unsigned char)R;
}

void  Picture_menager::Pixel(int x,int y,char type,unsigned char color)
{
 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;
 page=currentY>>5;
 Setpage(page);
 switch(type){
 case 'B':pixel[Yinpage][Xinpage].Blue = color;break;
 case 'G':pixel[Yinpage][Xinpage].Green= color;break;
 case 'R':pixel[Yinpage][Xinpage].Red  = color;break;
 default :pixel[Yinpage][Xinpage].Blue = color;
```

```
            pixel[Yinpage][Xinpage].Green= color;
            pixel[Yinpage][Xinpage].Red  = color;
            break;
    }
   }
```

```
COLOR_16_7M  Picture_menager::Pixel(int x,int y)
{
 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;
 page=currentY>>5;
 Setpage(page);
 Picture_menager::Colors.Blue = pixel[Yinpage][Xinpage].Blue ;
 Picture_menager::Colors.Green= pixel[Yinpage][Xinpage].Green;
 Picture_menager::Colors.Red  = pixel[Yinpage][Xinpage].Red  ;
 return (Picture_menager::Colors);
}

int Picture_menager::Pixel(int x,int y,char type)
{
 int color;
 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;
 page=currentY>>5;
```

```
    Setpage(page);
    switch(type){
    case 'B':color=(pixel[Yinpage][Xinpage].Blue)  ;break;
    case 'G':color=(pixel[Yinpage][Xinpage].Green) ;break;
    case 'R':color=(pixel[Yinpage][Xinpage].Red  ) ;break;
    case 'g':
    default :color=(pixel[Yinpage][Xinpage].Blue );
            break;
    }
    return(color);
}

COLOR_16_7M far * Picture_menager::LinePixel(unsigned x,unsigned y)
{

 currentX=x+xref;
 currentY=y+yref;
 Xinpage=currentX;
 Yinpage=currentY%32;
 page=currentY>>5;
 Setpage(page);
 return(pixel[Yinpage]+Xinpage);
}
```

```
/* Screenpj  version 2.0*/
#include <dir.h>
#include <string.h>
#include "C:\chakkrit\project\msvga.mod"
#define ImageWidth 256
#define ImageSize  65536L
int Xref0= 59;
int Yref0= 112;
int Xref1= 325;
int Yref1= 112;

#define pixels2bytes(n)  ((n+7)/8)


typedef struct{
                    char id[2]; /*"BM"*/
                    long filesize;
                    int  reserved[2];
                    long headersize;
                    long infoSize;
                    long width;
                    long high;
                    int  biPlanes;
                    int  bits;
                    long biCompression;
                    long biSizeImage;
                    long biXPelsPerMeter;
                    long biYPelsPerMeter;
                    long biClrUsed;
                    long biClrImportant;
               }BMPHEAD;

typedef struct
          {
          unsigned char rgbBlue;
          unsigned char rgbGreen;
```

```
        unsigned char rgbRed;
        /* unsigned char rgbReserved;*/
        }RGBCOLOR;


typedef struct{
        char drive[MAXDRIVE];
        char dir[MAXDIR];
        char file[MAXFILE];
        char ext[MAXEXT];
        }FILENAMEs;
typedef struct {
        FILENAMEs name;
        unsigned int bits;
        long filesize;
        long imagesize;
        long bytes;
        long headersize;
        long width;
        }FILEINFO;

long VGA_COLOR;
int  type_pix;
FILEINFO fi;
BMPHEAD bmp;
COLOR_16_7M colors;
RGBCOLOR colortable[256];
//int _red,_green,_blue;
int getfile(char *);
void placepicture(char *);
void placepicture_(char *file);

void savefile(int);
void windows_gen(int x1,int y1,int width,int high,int border,int for_ground,int back_ground);
void Text_display(int x,int y,int index,int background,char *string);
```

```
//Picture_menager win
/*****************************************************************/



void status(char);
void Compress_Type(int,char *);
void Compress_Bpp(int,unsigned long);
void Compress_Ratio(int,unsigned long ratio);
void Compress_Time(int);
void Compress_Time(void);
void File_(char *);
void File_out(char *);

void putdotw(int,int,int,int);
void DotPixel0(unsigned,int);
void lineDotPixel0(unsigned i,char far *mes);
void DotPixelB(int,unsigned,unsigned,unsigned int);
int ValuepixelB(char type,unsigned int,unsigned int);
int Savepixelw(int w,unsigned int i);
int Readpixelw(char type,int w,int x,int y);
void Error_(float rms);


void screen_initial(void);
class Windows
{
 protected:
        int C_WindowFrame;
        int C_Border;

        int C_Type;
        int C_TypeText;


        int C_Title;
        int C_TitleText;
```

```
    int C_WindowsText;
    int C_WindowsBackground;
    int C_AcButtom;
    int C_InacButtom;

    int High;
    int Width;
    int Border_thin;
    int Titlebar_width;
    int Border_gard_width;
    int Type_width;
    int X;
    int Y;

    char *buf;

public:
    Windows(){
     C_WindowFrame=_BLACK_;
     C_Border=_LIGHTGREY_;

     C_Type=_GREEN_;
     C_TypeText=_WHITE_;

     C_Title=_RED_;
     C_TitleText=_WHITE_;
     C_WindowsBackground=_LIGHTYELLOW_;
     C_WindowsText=_BLACK_;
     C_AcButtom=_BLACK_;
     C_InacButtom=_LIGHTGREY_;

    Border_thin=5;
    Titlebar_width=17;
    Border_gard_width=Titlebar_width;
    Width=256;
```

```
 High=256;
 Type_width=49;
}

void Set_Windows_size(int width,int high,int border_width=5,int titlebar_width=17)
    {
      Width=width;
      High=high;
      Border_thin=border_width;
      Titlebar_width=titlebar_width;
    }
void Set_Ctitle(int title,int titletext)
    {
     C_Title=title;
     C_TitleText=titletext;
    }
void Set_CWindows(int background){C_WindowsBackground=background;}
void Recgular(int x1,int y1);
void Place_windows(int x1,int y1,char *type);
void Windows_gen(int x1,int y1);
void Ptitlebar(char *s){ptitlebar(s);}
void Ptitle(char *s){ptitle(s);}
void Ptype(char *s){ptype(s);}
void Pwindows(int x,int y,char *s)
                                {
                                sprint_(s);

Text_display(x+Windows::getXref(),y+Windows::getYref()+16,C_WindowsText,C_WindowsBackground,buf);
                                  delete buf;
                                }
int getXref(void){return(X+Border_thin);}
int getYref(void){return(Y+Border_thin+Titlebar_width);}
void Iconbuttom(void);
void Titlebar(void);
void Title();
void Type();
```

```
        void Border(void);
        void WinArea(void);
        void sprint_(char *s){

                                buf=new char[strlen(s)*2];
                                sprintf(buf,s);


                               }
        void ptype(char *s){
                               sprint_(s);

Text_display(X+Border_thin+Titlebar_width,Y+Border_thin+16,C_TypeText,C_Type,buf);
                               delete buf;
                               }

        void ptitle(char *s){
                                sprint_(s);

Text_display(X+Border_thin+Titlebar_width+Type_width,Y+Border_thin+16,C_TitleText,C_Title,buf);
                                delete buf;
                                }
        void ptitlebar(char *s){
                                sprint_(s);

Text_display(X+Border_thin+Titlebar_width,Y+Border_thin+16,C_TitleText,C_Title,buf);
                                delete buf;
                               }


};
class Winscreen:public Windows
{

 private:
        int *Xref;
```

```
        int *Yref;

        int X_status;
        int Y_status;
        int X_operating;
        int Y_operating;
        int X_Menu;
        int Y_Menu;
        int Status_Width;
        int Menu_Width;
        int Operating_Width;
        int High_Work;

        int C_statusText;
        int C_Error;
        int C_Ready;
        int C_Compress;
        int C_Compare;

        int C_operating;
        int C_operatingText;
        int C_Menu;
        int C_MenuText;

        int getreturn;
public:
    Picture_menager Image[2];
    Winscreen():Windows()
    {
        C_statusText=_WHITE_;
        C_Error=_RED_;
        C_Ready=_GREEN_;
        C_Compress=_GOLD_;
        C_Compare=_BLUE_;
        C_operating=_LIGHTGREY_;
```

```
    C_operatingText=_YELLOW_;
    C_Menu=_WHITE_;
    C_MenuText=_BLACK_;
    C_Title=_BLUE_;
    C_TitleText=_YELLOW_;
    Menu_Width=18;
    Operating_Width=18;
    High_Work=480-(2*Border_thin+Titlebar_width+Menu_Width+Operating_Width);
    High=High_Work+Menu_Width+Operating_Width;
    Width=630;
    X=0;
    Y=0;
    X_Menu=X+Border_thin;
    Y_Menu=Y+Border_thin+Titlebar_width;

    Status_Width=12*8+1;
    X_status=X+Border_thin;
    Y_status=Y+Border_thin+Titlebar_width+Menu_Width+High_Work;

    X_operating=X_status+Status_Width+1;
    Y_operating=Y_status;
}

Windows Win[3];
void WinArea();
void Place_windows();
void Menubar();




void Menu_File();
void Menu_Compress();


void Operating();
void Status(int color);
int gets_Menu(char *s);
```

```
        void Poperating(char *s){
                                        sprint_(s);

            Text_display(X_operating,Y_operating+17,C_operatingText,C_operating,buf);
                                        delete buf;


                                    }
        void Pstatus(char *s,int color){
                                            sprint_(s);
                                            Text_display(X_status,Y_status+17,C_statusText,color,buf);
                                            delete buf;


                                            }
        void Pready()   {Status(C_Ready);   Pstatus("   Ready   ",C_Ready);}
        void Perror()   {Status(C_Error);   Pstatus("  *Error*  ",C_Error);}
        void Pcompress(){Status(C_Compress);Pstatus("Compress...",C_Compress);}
        void Pcompare() {Status(C_Compare); Pstatus("*Compare...",C_Compare);}


};

Winscreen Screen;
extern unsigned int M,BlockTotal;
extern unsigned int N,BlockWidth;
extern clock_t timein,timeout;
/*********************************************************************/
int Winscreen::gets_Menu(char *s){
            Menubar();
            getreturn=gets_(X_Menu,Y_Menu,C_MenuText,C_Menu,s);
            Menubar();
            if(getreturn==-1)return(-1);
            else return(getreturn);
    }
void Winscreen::Menubar()
{
```

```
 Gbox(X+Border_thin,Y+Border_thin+Titlebar_width,X+Border_thin+Width-
1,Y+Border_thin+Titlebar_width+Menu_Width-2,C_Menu);
 putline_(X+Border_thin,Y+Border_thin+Titlebar_width+Menu_Width-1,X+Border_thin+Width-
1,Y+Border_thin+Titlebar_width+Menu_Width-1,_BLACK_);
}

void Winscreen::WinArea()
{
 Gbox(X+Border_thin,Y+Border_thin+Titlebar_width+Menu_Width,X+Border_thin+Width-
1,Y+Border_thin+Titlebar_width+Menu_Width+High_Work-1,C_WindowsBackground);

}
void status(char a)
{

  switch(a){
     case 'R' :
                    Screen.Pready();
                    break;
     case 'C' :
                    Screen.Pcompress();
                    break;
     case 'E' :
                Screen.Perror();
                    break;
     case 'P' :
                Screen.Pcompare();
     default:break;
  }

}


void Winscreen::Status(int color)
{
 Gbox(X_status,Y_status,X_status+Status_Width-1,Y_status+Operating_Width-1,color);
```

```
 putline_(X_status+Status_Width,Y_status,X_status+Status_Width,Y_status+Operating_Width-
1,C_WindowFrame);
 putline_(X_status,Y_status,X_status+Status_Width-1,Y_status,C_WindowFrame);
}

void Winscreen::Operating()
{
 Gbox(X_operating,Y_operating,Border_thin+Width-1,Y_operating+Operating_Width-1,C_operating);
 putline_(Border_thin,Y_operating,Border_thin+Width-1,Y_operating,C_WindowFrame);
}

void Winscreen::Place_windows()
{
 int x1;
 int y1;

 Xref=new int [3];
 Yref=new int [3];

 Border();
 Titlebar();
 Iconbuttom();
 Menubar();
 WinArea();
 Operating();
 Pready();
 Ptitlebar("          Chakkrit Jaiman`s Image Compression Project");
 x1=((VIDEOMAXX)-(4*Border_thin+512))/2;
 y1=Y+Border_thin+Titlebar_width+Menu_Width;
 Win[0].Place_windows(x1,y1,"Origin");
 Win[1].Place_windows(x1+2*Border_thin+256,y1,"Result");
 Xref[0]=Win[0].getXref();
 Yref[0]=Win[0].getYref();
 Xref0=Xref[0];
 Yref0=Yref[0];
 Xref[1]=Win[1].getXref();
```

```
  Yref[1]=Win[1].getYref();
  Xref1=Xref[1];
  Yref1=Yref[1];

  Win[2].Set_Windows_size(522,4*16);
  Win[2].Set_CWindows(_WHITE_);
  Win[2].Recgular(x1,y1+Border_thin*2+Titlebar_width+256);
  Xref[2]=Win[2].getXref();
  Yref[2]=Win[2].getYref();
  Image[0].Origin(Xref[0],Yref[0]);
  Image[0].Size(256,256);
  Image[1].Origin(Xref[1],Yref[1]);
  Image[1].Size(256,256);
 }

void Erms(float rms)
{
 sprintf(txmp,"Root Mean Square Error = %7.4f",rms);
 Screen.Win[2].Pwindows(266,0,txmp);
}
void Emae(float mae)
{
 sprintf(txmp,"Mean Absolute Error    = %7.4f",mae);
 Screen.Win[2].Pwindows(266,1*16,txmp);
}
void Esnr(float snr)
{
 sprintf(txmp,"Signal to Noise        = %7.4f",snr);
 Screen.Win[2].Pwindows(266,2*16,txmp);
}
void Esnr_dB(float dB)
{
 sprintf(txmp,"Signal to Noise(dB)    = %7.4f",dB);
 Screen.Win[2].Pwindows(266,3*16,txmp);
}
void Error_(float rms,float mae,float snr,float dB)
```

```
 {
   Erms( rms );
   Emae( mae );
   Esnr( snr );
   Esnr_dB(dB);
 }
 void Compress_Bpp(int w,unsigned long length)
 {
   float bit;
   bit=(float)length/65536.0;

   switch(w){
          case 0:sprintf(txmp,"Origin:Data rate= %6.3f Bpp",bit);
                 Screen.Win[2].Pwindows(0,0,txmp);break;
          case 1:sprintf(txmp,"Result:Data rate= %6.3f Bpp",bit);
                 Screen.Win[2].Pwindows(0,1*16,txmp);break;
   }

 }
 void Compress_Ratio(int w,unsigned long length)
 {
   float ratio;
   long i;
   i=(fi.bits==8)?65536L:196608L;
   ratio=(i-(float)(length/8))*100/i;
   sprintf(txmp," %c Compressed = %7.4f %","%",ratio);
   switch(w){
          case 0:
                 Screen.Win[2].Pwindows(0,16,txmp);break;
          case 1:
                 Screen.Win[2].Pwindows(0,2*16,txmp);break;
   }

 }
 void Compress_Time(int w)
 {
```

```
 sprintf(txmp," Compress Time>> %3.2f    Sec.",(double)(timeout-timein)/CLK_TCK_);
 switch(w){
       case 0:outtextcl(7,29,_YELLOW_,txmp);break;
       case 1:outtextcl(42,29,_YELLOW_,txmp);break;
 }
}

void Compress_Time(void)
{
  sprintf(txmp," Compress Time>> %3.2f Sec.",(double)(timeout-timein)/CLK_TCK_);
  Screen.Win[2].Pwindows(0,3*16,txmp);
}

void File_out(char *file)
{
  sprintf(txmp,"%s",file);
  outtextcl(42,4,_BROWN_,txmp);
}
/*****************************************************************/


void putdotw(int w,int x,int y,int color)
{
   x=(w)?(x+Xref1):(x+Xref0);
   y=(w)?(y+Yref1):(y+Yref0);
   switch(fi.bits)
   {
    case 8 : Putpixel(x,y,color);break;
    case 24: Screen.Image[w].Pixel(x,y,color,color,color);break;
   }
}

void putdotw(int w,int x,int y,int colorR,int colorG,int colorB)
{
   Screen.Image[w].Pixel(x,y,colorR,colorG,colorB);
```

```
}
void lineDotPixel0(unsigned i,char far *mes)
{
    unsigned int x,y;
   x=i%ImageWidth;
   y=i/ImageWidth;
   y=255-y;
   lineDirectpixel(x+Xref0,y+Yref0,256,mes);
}
void DotPixel0(unsigned i,int color)
{
   unsigned int x,y;
   x=i%ImageWidth;
   y=i/ImageWidth;
   y=255-y;

   putdotw(0,x,y,color);
}
void DotPixelB(int windows,unsigned Block,unsigned num,unsigned int color)
{
  unsigned x,y;
  int xb,yb;

  yb=Block/BlockWidth;
  xb=Block%BlockWidth;
  y=(num/N)+(yb*N);
  x=(num%N)+(xb*N);
  putdotw(windows,x,y,color);
}

void D_DotPixelB(int windows,unsigned Block,unsigned num,unsigned int colorR,unsigned int colorG,unsigned
int colorB)
{
  unsigned x,y;
  int xb,yb;
```

```
  xb=Block%BlockWidth;
  y=(num/N)+(yb*N);
  x=(num%N)+(xb*N);
  Screen.Image[windows].Pixel(x,y,colorR,colorG,colorB);
}

void SD_DotPixelB(int windows,unsigned Block,unsigned num,unsigned int colorR,unsigned int colorG,unsigned
int colorB)
{
  unsigned x,y;
  int xb,yb;

  yb=Block/BlockWidth;
  xb=Block%BlockWidth;
  y=(num/N)+(yb*N);
  x=(num%N)+(xb*N);

    Screen.Image[windows].SomePixel(x,y,colorR,colorG,colorB);
}


int monoReadpixel(int x,int y,char color)
{
  long tempcolor;
  tempcolor=Getpixel(x,y);
  switch(color)
  {
   case 'R':return((tempcolor>>16)&&0x000000ff);
   case 'G':return((tempcolor>>8)&&0x000000ff);
   case 'B':return((tempcolor)&&0x000000ff);
  }

   return(-1);
}
```

```
int ValuepixelB(char type,unsigned int block,unsigned int num)
{
  unsigned int x,y;
  int xb,yb;
  yb=block/BlockWidth;
  xb=fmod(block,BlockWidth);
  y=(num/N)+(yb*N);
  x=(num%N)+(xb*N);
  if(type=='A'){colors=Screen.Image[0].Pixel(x,y);return(-1);}
  else return(Screen.Image[0].Pixel(x,y,type));
}



int Savepixelw(int w,unsigned int i)
{
  unsigned int x,y;
  int color;
  x=i%ImageWidth;
  y=i/ImageWidth;
  y=255-y;
  x=(w)?(x+Xref1):(x+Xref0);
  y=(w)?(y+Yref1):(y+Yref0);
   color=getpixelg(x,y);

   return(color);
}



void screen_initial()
{
    Screen.Place_windows();
}
void Windows::Recgular(int x1,int y1)
{
   Titlebar_width=0;
```

```
    X=x1;
    Y=y1;
    Border();
    WinArea();
}
void Windows::Windows_gen(int x1,int y1)
{
    X=x1;
    Y=y1;
    Border();
    Titlebar();
    Iconbuttom();
    WinArea();
}
void Windows::Place_windows(int x1,int y1,char *type)
{
    X=x1;
    Y=y1;
    Border();
    Title();
    Iconbuttom();
    Type();
    WinArea();
    ptype(type);

}

void Windows::Border(void)
{
  int i,xi,yi,yf,xvi,yvi,xvf;
  image_windows(X,Y,X+2*Border_thin+Width-1,Y+2*Border_thin+High+Titlebar_width-1,1,C_WindowFrame);
  image_windows(X+Border_thin-2,Y+Border_thin-
2,X+Width+Border_thin+1,Y+High+Border_thin+Titlebar_width+1,Border_thin-2,C_Border);
  image_windows(X+Border_thin-1,Y+Border_thin-
1,X+Width+Border_thin,Y+High+Border_thin+Titlebar_width,1,C_WindowFrame);
    for(i=0;i<4;i++)
```

```
  {
    if((i%2)!=0){
    xi=X+Border_thin+(Width-Border_gard_width-1);
    yvi=Y+Border_thin+High+Titlebar_width-Border_gard_width-1;
    }else{
     xi=X+Border_gard_width+Border_thin-1;
     yvi=Y+Border_gard_width+Border_thin-1;
    }
    if(i<2){
     yi=Y;
     yf=yi+Border_thin;
     xvi=X;
     xvf=xvi+Border_thin;
    }else{
     yi=Y+High+Border_thin+Titlebar_width;
     yf=yi+Border_thin-1;
     xvi=X+Width+Border_thin;
     xvf=xvi+Border_thin-1;
    }
  putline_(xi,yi,xi,yf,C_WindowFrame);
  putline_(xvi,yvi,xvf,yvi,C_WindowFrame);
  }
}

void Windows::Iconbuttom()
{
    Gbox(X+Border_thin,Y+Border_thin,X+Titlebar_width+Border_thin-2,Y+Titlebar_width+Border_thin-
2,C_InacButtom);
    putline_(X+Border_thin,Y+Border_thin+Titlebar_width-1,X+Width+Border_thin-
1,Y+Border_thin+Titlebar_width-1,C_WindowFrame);
    image_windows(X+Border_thin+(Titlebar_width-10)/2-1,Y+Border_thin+(Titlebar_width-2)/2-
2,X+Border_thin+(Titlebar_width-10)/2+9,Y+Border_thin+(Titlebar_width-2)/2+1,1,C_WindowFrame);
    Gbox(X+Border_thin+(Titlebar_width-8)/2-1,Y+Border_thin+(Titlebar_width-2)/2,X+Border_thin+(Titlebar_width-
10)/2+8,Y+Border_thin+(Titlebar_width-2)/2,C_WindowsBackground);
    image_windows(X+Border_thin-1,Y+Border_thin-1,X+Titlebar_width+Border_thin-
1,Y+Titlebar_width+Border_thin-1,1,C_WindowFrame);
```

```
}

void Windows::Titlebar()
{
  Gbox(X+Border_thin,Y+Border_thin,X+Width+Border_thin-1,Y+Titlebar_width+Border_thin-2,C_Title);
  putline_(X+Border_thin,Y+Border_thin+Titlebar_width-1,X+Width+Border_thin-1,Y+Border_thin+Titlebar_width-
1,C_WindowFrame);
}

void Windows::Title()
{
  Gbox(X+Border_thin+Titlebar_width+Type_width,Y+Border_thin,X+Width+Border_thin-
1,Y+Titlebar_width+Border_thin-2,C_Title);
  putline_(X+Border_thin,Y+Border_thin+Titlebar_width-1,X+Width+Border_thin-1,Y+Border_thin+Titlebar_width-
1,C_WindowFrame);
}

void Windows::Type()
{
  Gbox(X+Border_thin+Titlebar_width,Y+Border_thin,X+Border_thin+Titlebar_width+Type_width-
1,Y+Border_thin+Titlebar_width-2,C_Type);
  putline_(X+Border_thin+Titlebar_width+Type_width-
1,Y+Border_thin,X+Border_thin+Titlebar_width+Type_width-1,Y+Border_thin+Titlebar_width,C_WindowFrame);
}
void Windows::WinArea()
{
 Gbox(X+Border_thin,Y+Border_thin+Titlebar_width,X+Border_thin+Width-1,Y+Border_thin+Titlebar_width+High-
1,C_WindowsBackground);
}


void Text_display(int x,int y,int index,int background,char *string)
{
 int temp;
 temp=Backgroundtext;
 Backgroundtext=background;
```

```
 outtextxy(x,y-16,index,string);
 Backgroundtext=temp;
}


void Compress_Type(int w,char *type)
{
        Screen.Win[w].Ptitle(type);
}


/*************************************************************/

int getfile(char *file)
{
  FILE *fp;

  long numcolor,i,test=0;
  int r,g,b;

  fp=(FILE *)malloc(sizeof(FILE));
  if((fp=fopen(file,"rb"))==NULL)
  {
        free(fp);
        return(1);
  }
  fnsplit(file,fi.name.drive,fi.name.dir,fi.name.file,fi.name.ext);
    if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)==sizeof(BMPHEAD))
    {
        numcolor=1<<(bmp.bits);

        fi.bits      =bmp.bits;
        fi.filesize  =bmp.filesize;
        fi.imagesize =((bmp.filesize)-(bmp.headersize));
        fi.headersize=bmp.headersize;
```

```
//      fi.bytes    =(fi.bits)*32;
                if(fi.bits==1)
                 fi.bytes=pixels2bytes(fi.width);
                else if(fi.bits==4)
                 fi.bytes=pixels2bytes(fi.width)<<2;
                else if(fi.bits==8)
                 fi.bytes=fi.width;
                else if(fi.bits==24)
                  fi.bytes=fi.width*3;

                if (fi.bytes & 0x0003)
                 {  fi.bytesl=0x0003;
                    fi.bytes++;
                 }

test=0;
// write color table
for(i=0;i<numcolor;i++)
{
      b=fgetc(fp);
      colortable[i].rgbBlue=b;

      g=fgetc(fp);
      colortable[i].rgbGreen=g;

      r=fgetc(fp);
      colortable[i].rgbRed=r;
      if((b==g)&&(g==r)) test++;

         if(bmp.infoSize!=12L)
           fgetc(fp);

      Setdac(i,(int)r,(int)g,(int)b);
      if((b==g)&&(g==r))test++;
}
if(test==numcolor)type_pix=1;else type_pix=0;
```

```
            VGA_COLOR=numcolor;
            fclose(fp);
            free(fp);

    }else{
            fclose(fp);
            free(fp);
//          setvideomode(3);
//          printf("file error or no bmp file");
//          exit(1);
            return(2);
    }
    return(0);

}

void placepicture_(char *file)
{
  FILE *fp;
  char in_buf[3072];
  char picture[768];
  unsigned char color;
  unsigned long length;
  unsigned int  pixel=0;

  fnmerge(file,fi.name.drive,fi.name.dir,fi.name.file,fi.name.ext);
// File_(file);
  Screen.Win[0].Ptitle(file);
  if((fp=fopen(file,"rb"))==NULL)
  {
   fclose(fp);
   setvideomode(3);
   printf("Open Files Error");
   exit(1);
  }
  if(setvbuf(fp,in_buf,_IOFBF,3072)!=0)
```

```
    {
//      status('E') ;
        Screen.Perror();
        Screen.Operating();
        sprintf(txmp,"failed to set up buffer for output file");
        Screen.Poperating(txmp);
//      outtextcl(0,1,_RED_,txmp);

    }

    .
  fseek(fp,fi.headersize,SEEK_SET);
  if(fi.bits==8){
    Compress_Bpp(0,8L*65536L);
    for(length=0;length < fi.imagesize;length++){
            color=getc(fp);
            DotPixel0(pixel++,color);
    }
  }else{
   if(fi.bits==24)
   Compress_Bpp(0,24L*65536L);
    for(length=0;length < fi.imagesize;length+=(768))
     {
       fseek(fp,fi.headersize+length,SEEK_SET);
       fread(picture,1,sizeof(char)*768,fp);
       lineDotPixel0(pixel,picture);
       pixel+=256;
     }
  }
  fclose(fp);
}

void savefile(int w)
{
  FILE *fp;
  char file[MAXPATH];
```

```
unsigned char pixel;
COLOR_16_7M far * pixels;
unsigned long length;
int line=0;

sprintf(txmp,"ENTER FILENAME     ");
Screen.Operating();
Screen.Poperating(txmp);
Screen.gets_Menu(file);
Screen.Operating();
Screen.Menubar();

fp=(FILE *)malloc(sizeof(FILE));
if((fp=fopen(file,"wb"))==NULL)
{
  free(fp);
  Screen.Perror();
  sprintf(txmp,"CAN NOT OPEN FILE");
  Screen.Poperating(txmp);
  return;
}
fwrite((char *)&bmp,1,sizeof(BMPHEAD),fp);
if(fi.bits!=24)
{
for(length=0;length<256;length++)
{
        fputc(colortable[(unsigned int)length].rgbBlue,fp);
        fputc(colortable[(unsigned int)length].rgbGreen,fp);
        fputc(colortable[(unsigned int)length].rgbRed,fp);
        fputc(0,fp);
}
fseek(fp,fi.headersize,SEEK_SET);
for(length=0;length < ImageSize;){
        pixel=Savepixelw(w,(unsigned int)length);
        fputc(pixel,fp);
        length++;
```

```
}
}else{
fseek(fp,fi.headersize,SEEK_SET);
line=255;
for(length=0;length < ImageSize;length+=256){
    pixels=Screen.Image[w].LinePixel(0,line);
    fseek(fp,fi.headersize+(length*3),SEEK_SET);
    fwrite(pixels,sizeof(COLOR_16_7M),256,fp);
    line--;
}
}
fclose(fp);
free(fp);

}
```

```
clock_t clockin,clockout;
int RedC=2;
int GreenC=1;
int BlueC=0;
int GrayC=0;

#define Squaref(x) (float)( ((float)(x) ) * ((float)(x)) )
#define Square(x)  (unsigned long)( ((long)(x)) * ((long)(x)) )
#define ROUNDu(a) (unsigned int) ( ((int)a) + 0.5)
#define ROUND(a)  ( ( (a) < 0 )?(int)((a)-0.5):(int)((a)+0.5))

void Error_cal(int in,int out);
unsigned int Round (float);
unsigned long BTC_process(int method,int ssss=0);

typedef struct{
                unsigned char mean;
                unsigned char std;
                unsigned int data;
                }HEADBTC;

class RMS_Error
{
  private:
          int inR,inG,inB,outR,outG,outB;
//        unsigned long pixels_number;
          float Berr,Gerr,Rerr;

  public:
        float error;
        RMS_Error(void){
                //lw=Square(lmageWidth);
                error=0;
                Berr=0;
                Gerr=0;
                Rerr=0;
```

```
            // pixels_number=0;
    }
    void Reset(void){
            //lw=Square(ImageWidth);
            error=0;
            Berr=0;
            Gerr=0;
            Rerr=0;
            // pixels_number=0;
    }
    void process(COLOR_16_7M inpixel,COLOR_16_7M outpixel)
    {

     inR =inpixel.Red;
     inG =inpixel.Green;
     inB =inpixel.Blue;
     outR=outpixel.Red;
     outG=outpixel.Green;
     outB=outpixel.Blue;
     Berr=Berr+(float)Square(outB-inB);
     Gerr=Gerr+(float)Square(outG-inG);
     Rerr=Rerr+(float)Square(outR-inR);
//   pixels_number++;
     }

     void End(unsigned long pixels_number){
          error=sqrt((Berr+Gerr+Rerr)/(3*pixels_number));
      }
};


class MAE_Error
{
```

```
private:
        int inR,inG,inB,outR,outG,outB;
//      unsigned long pixels_number;
        long Berr,Gerr,Rerr;
public:
        float error;
        MAE_Error(){
                //lw=Square(lmageWidth);
                error=0;
                Berr=0;
                Gerr=0;
                Rerr=0;
                // pixels_number=0;
        }
        void Reset(){
                //lw=Square(lmageWidth);
                error=0;
                Berr=0;
                Gerr=0;
                Rerr=0;
                // pixels_number=0;
        }
        void process(COLOR_16_7M inpixel,COLOR_16_7M outpixel)
        {

          inR =inpixel.Red;
          inG =inpixel.Green;
          inB =inpixel.Blue;
          outR=outpixel.Red;
          outG=outpixel.Green;
          outB=outpixel.Blue;



          Berr=Berr+abs(outB-inB);
          Gerr=Gerr+abs(outG-inG);
          Rerr=Rerr+abs(outR-inR);
```

```
    //      pixels_number++;
        }
        void End(unsigned long pixels_number){
            error=(float)(Berr+Gerr+Rerr)/(float)(3*pixels_number);
        }
};


class SNR_Error
{
 private:
        int inR,inG,inB;
        unsigned long insR,insG,insB;
        unsigned long pixels_number;
 public:
       float error;
       float dB;
       SNR_Error(){
                //lw=Square(ImageWidth);
                insR=0;
                insG=0;
                insB=0;
                error=0;
                pixels_number=0;
       }
       void Reset(){
                //lw=Square(ImageWidth);
                insR=0;
                insG=0;
                insB=0;
                error=0;

                pixels_number=0;
       }
```

```
        void process(COLOR_16_7M in)
        {
         inR=in.Red;
         inG=in.Green;
         inB=in.Blue;
         insR=insR+Square(inR);
         insG=insG+Square(inG);
         insB=insB+Square(inB);

//       pixels_number++;
        }
        void End(unsigned long pixels_number,float rms)
            {
             error=sqrt((float)(insR+insG+insB)/(float)(3*pixels_number));
             error=error/rms;
             dB=20*log10(error);
            }
};



RMS_Error RMS;
MAE_Error MAE;
SNR_Error SNR;

void Error_cal(int in,int out)
{

  int y,x;
  unsigned long count=0;
  COLOR_16_7M colors_in,colors_out;
  float rms_err;
  RMS.Reset();
  MAE.Reset();
  SNR.Reset();
```

```
  for(y=0;y<ImageWidth;y++)
  {
    for(x=0;x<ImageWidth;x++)
    {
      colors_in=Screen.Image[in].Pixel(x,y);
      colors_out=Screen.Image[out].Pixel(x,y);
      RMS.process(colors_in,colors_out);
      MAE.process(colors_in,colors_out);
      SNR.process(colors_in);
//      count++;
    }
  }
    count=(unsigned long)ImageWidth*(unsigned long)ImageWidth;
    RMS.End(count);
    MAE.End(count);
    rms_err=RMS.error;
    SNR.End(count,rms_err);
//  return;
}


class BTC_Standard
{
 private:
          unsigned long length;
          unsigned int block;
          int ColorL[3],ColorH[3];
          int std_int[3],mean_int[3];
          float std[3],mean[3];
          float meansqr[3],color[3];
          unsigned long q[3],Summean[3],Summeansqr[3];
          unsigned int i;
          float roundstd;
          int roundmean;
          unsigned rounding;
```

```
    int headersize;

public:
    int *dot[3];

    BTC_Standard(){
     headersize=16;
    rounding=0xffff;
     roundstd=1;
     roundmean=1;
     length=0;
     block=0;
     dot[BlueC]=new int [M];
     dot[GreenC]=new int [M];
     dot[RedC]=new int [M];
     q[0]=0;
     q[1]=0;
     q[2]=0;
     Summean[0]=0;
     Summean[1]=0;
     Summean[2]=0;
     Summeansqr[0]=0;
     Summeansqr[1]=0;
     Summeansqr[2]=0;

     i=0;
    }
    void Reset(){
     headersize=16;
     rounding=0xffff;
     roundstd=1;
     roundmean=1;
     length=0;
     block=0;
     delete dot[BlueC];
     delete dot[GreenC];
```

```
delete dot[RedC];
dot[BlueC]=new int [M];
dot[GreenC]=new int [M];
dot[RedC]=new int [M];
q[0]=0;
q[1]=0;
q[2]=0;
Summean[0]=0;
Summean[1]=0;
Summean[2]=0;
Summeansqr[0]=0;
Summeansqr[1]=0;
Summeansqr[2]=0;

i=0;
}

void Sum(int count ,int pixel)
{
 dot[GrayC][count]=pixel;
 Summean[GrayC]=pixel+Summean[GrayC];
 Summeansqr[GrayC]=Square(pixel)+Summeansqr[GrayC];
//  i++;
}
void Sum(int count ,int R,int G,int B)
{
 dot[RedC][count]=R;
 dot[GreenC][count]=G;
 dot[BlueC][count]=B;
 Summean[RedC]=R+Summean[RedC];
 Summean[GreenC]=G+Summean[GreenC];
 Summean[BlueC]=B+Summean[BlueC];
 Summeansqr[RedC]=Square(R)+Summeansqr[RedC];
 Summeansqr[GreenC]=Square(G)+Summeansqr[GreenC];
 Summeansqr[BlueC]=Square(B)+Summeansqr[BlueC];
//  i++;
```

```
        }
        void mod_process(int Cp);
        void demod_process(int Cp);
        void display();
        void displayC();
        unsigned long process(unsigned int bock,int size);
        unsigned long process(unsigned int bock,int size,int bbb);
};

void BTC_Standard::mod_process(int Cp)
{
 mean[Cp]    = Summean[Cp]/M;
 mean_int[Cp]= ROUNDu(mean[Cp]/roundmean);
 meansqr[Cp] = Summeansqr[Cp]/M;
 std[Cp]     = meansqr[Cp]-Squaref(mean[Cp]);
 std_int[Cp] = ROUNDu(sqrt(std[Cp])/roundstd)&rounding;
 length+=headersize;
}
void BTC_Standard::demod_process(int Cp)
{

    for(i=0;i<M;i++)
    {
     if(dot[Cp][i]>=(mean_int[Cp]*roundmean))q[Cp]++;
    }

    if(q[Cp]==0||q[Cp]==M)
    {
     ColorL[Cp]=mean_int[Cp]*roundmean;
     ColorH[Cp]=ColorL[Cp];
     std_int[Cp]=0;
    }else{

        std[Cp]=(float)std_int[Cp]*roundstd;
        mean_int[Cp]=mean_int[Cp]*roundmean;//
        // std=0;
```

```
            color[Cp]=(mean_int[Cp])-(std[Cp]*(sqrt((float)q[Cp]/((float)(M-q[Cp])))));
            if(color[Cp]<0)ColorL[Cp]=0;
            else ColorL[Cp]=ROUNDu(color[Cp]);
            color[Cp]=(mean_int[Cp])+(std[Cp]*(sqrt((float)(M-q[Cp])/(float)q[Cp])));
            if(color[Cp]>255)ColorH[Cp]=255;
            else ColorH[Cp]=ROUNDu(color[Cp]);
            }

    if(std_int[Cp])length+=M;
}


void BTC_Standard::display()
{

    for(i=0;i<M;i++){
     if(dot[GrayC][i]>=(mean_int[GrayC]))DotPixelB(1,block,i,(unsigned char)ColorH[GrayC]);
            else DotPixelB(1,block,i,(unsigned char)ColorL[GrayC]);
    }
}

void BTC_Standard::displayC()
{

            for(i=0;i<M;i++)
            {
              if(dot[BlueC][i]>=(mean_int[BlueC]))dot[BlueC][i]=ColorH[BlueC];
              else dot[BlueC][i]=ColorL[BlueC];
              if(dot[GreenC][i]>=(mean_int[GreenC]))dot[GreenC][i]=ColorH[GreenC];
              else dot[GreenC][i]=ColorL[GreenC];
              if(dot[RedC][i]>=(mean_int[RedC]))dot[RedC][i]=ColorH[RedC];
              else dot[RedC][i]=ColorL[RedC];

            }
            for(i=0;i<M;i++)D_DotPixelB(1,block,i,dot[RedC][i],dot[GreenC][i],dot[BlueC][i]);
```

```
}

unsigned long BTC_Standard::process(unsigned int bock,int size,int bbb)
{
    block=bock;
    if(size){
        roundstd=8.5;
        roundmean=4;
        rounding=0x000f;
        headersize=10;
    }

    mod_process(BlueC);
    mod_process(GreenC);
    mod_process(RedC);
    demod_process(BlueC);
    demod_process(GreenC);
    demod_process(RedC);
    displayC();

    return(length);
}
unsigned long BTC_Standard::process(unsigned int bock,int size)
{
  block=bock;
   if(size){
        roundstd=8.5;
        roundmean=4;
        rounding=0x000f;
        headersize=10;
   }
   mod_process(GrayC);
   demod_process(GrayC);
   display();
   return(length);
}
```

```
/*************************************************************************/
class BTC_RMS
{
 private:
          unsigned long length;
          unsigned int block;
          int std_int[3];
          float std[3],mean[3];
          float meansqr[3],color[3];
          unsigned long q[3],Summean[3],Summeansqr[3];
          unsigned int i;
          float roundstd;
          int roundmean;
          unsigned rounding;
          int headersize;

public:
         int ColorL[3],ColorH[3];
         int mean_int[3];
         int *dot[3];
         BTC_RMS(){
          headersize=16;
          rounding=0xffff;
          roundstd=1;
          roundmean=1;
          length=0;
          block=0;
          dot[BlueC]=new int [M];
          dot[GreenC]=new int [M];
          dot[RedC]=new int [M];
          q[0]=0;
          q[1]=0;
          q[2]=0;

          Summean[0]=0;
          Summean[1]= 0;
```

```
  Summean[2]=0;
  Summeansqr[0]=0;
  Summeansqr[1]=0;
  Summeansqr[2]=0;

  i=0;
 }
 void Reset(){
   headersize=16;
  rounding=0xffff;
  roundstd=1;
  roundmean=1;
  length=0;
  block=0;
  delete dot[BlueC];
  delete dot[GreenC];
  delete dot[RedC];
  dot[BlueC]=new int [M];
  dot[GreenC]=new int [M];
  dot[RedC]=new int [M];
  q[0]=0;
  q[1]=0;
  q[2]=0;
  Summean[0]=0;
  Summean[1]= 0;
  Summean[2]=0;
  Summeansqr[0]=0;
  Summeansqr[1]=0;
  Summeansqr[2]=0;
  i=0;
 }

 void Sum(int count,int pixel)
 {
   dot[GrayC][count]=pixel;
   Summean[GrayC]=pixel+Summean[GrayC];
```

```
        }
        void Sum(int count,int R,int G,int B)
        {
         dot[RedC][count]=R;
         dot[GreenC][count]=G;
         dot[BlueC][count]=B;
         Summean[RedC]  =R+Summean[RedC];
         Summean[GreenC]=G+Summean[GreenC];
         Summean[BlueC] =B+Summean[BlueC];


        }
        void mod_process(int Cp);
        void demod_process(int Cp);
        void display();
        void displayC();
        unsigned long process(unsigned int box,int size);
        unsigned long process(unsigned int box,int size,int bits);
        void processing(int size);
        void processing(int size,int bits);
};
void BTC_RMS::mod_process(int Cp)
{
 mean[Cp]    = Summean[Cp]/M;
 mean_int[Cp]= ROUNDu(mean[Cp])/roundmean;
 length+=headersize;
}

void BTC_RMS::demod_process(int Cp)
{
    mean_int[Cp]*=roundmean;

    for(i=0;i<M;i++)
    {
     if(dot[Cp][i]>=(mean_int[Cp]) )q[Cp]++;
```

```
    }

    if(q[Cp]==0||q[Cp]==M)
    {
     ColorL[Cp]=mean_int[Cp];
     ColorH[Cp]=ColorL[Cp];
    }else{
          ColorL[Cp]=0;
          ColorH[Cp]=0;
          for(i=0;i<M;i++)
          {
           if(dot[Cp][i]>=mean_int[Cp])ColorH[Cp]+=dot[Cp][i];
           else ColorL[Cp]+=dot[Cp][i];
          }
          color[Cp]=(float)ColorL[Cp]/((float)M-(float)q[Cp]);
          if(color[Cp]<0)ColorL[Cp]=0;
          else ColorL[Cp]=ROUNDu(color[Cp]);
          color[Cp]=(float)ColorH[Cp]/((float)q[Cp]);
          if(color[Cp]>255)ColorH[Cp]=255;
          else ColorH[Cp]=ROUNDu(color[Cp]);
    }
    if(q[Cp]!=0&&q[Cp]!=M)length+=M;
}

void BTC_RMS::display()
{
    for(i=0;i<M;i++){
     if(dot[GrayC][i]>=(mean_int[GrayC]))DotPixelB(1,block,i,ColorH[GrayC]);
          else DotPixelB(1,block,i,ColorL[GrayC]);
    }
}

void BTC_RMS::displayC()
{
          for(i=0;i<M;i++)
          {
```

```
            if(dot[BlueC][i]>=(mean_int[BlueC]))dot[BlueC][i]=ColorH[BlueC];
            else dot[BlueC][i]=ColorL[BlueC];
            if(dot[GreenC][i]>=(mean_int[GreenC]))dot[GreenC][i]=ColorH[GreenC];
            else dot[GreenC][i]=ColorL[GreenC];
            if(dot[RedC][i]>=(mean_int[RedC]))dot[RedC][i]=ColorH[RedC];
            else dot[RedC][i]=ColorL[RedC];

        }
        for(i=0;i<M;i++)D_DotPixelB(1,block,i,dot[RedC][i],dot[GreenC][i],dot[BlueC][i]);
}

void BTC_RMS::processing(int size)
{
    if(size){
//          roundstd=8.5;
            roundmean=4;
//          rounding=0x000f;
            headersize=12;
    }

    mod_process(GrayC);
    demod_process(GrayC);
//  display();
//  return(length);
}

unsigned long BTC_RMS::process(unsigned int box,int size)
{
  block=box;
  processing(size);
  display();
  return(length);
}

void BTC_RMS::processing(int size,int bits)
```

```
{
    if(size){
  //          roundstd=8.5;
              roundmean=4;
  //          rounding=0x000f;
              headersize=12;
    }
    mod_process(BlueC);
    mod_process(GreenC);
    mod_process(RedC);
    demod_process(BlueC);
    demod_process(GreenC);
    demod_process(RedC);
//    displayC();
   // return(length);
}


unsigned long BTC_RMS::process(unsigned int box,int size,int bits)
{
  block=box;
  processing(size,bits);
  displayC();
  return(length);
}
/*****************************************************************/

class BTC_MAE
{
 private:
          unsigned long length;
          unsigned int block;
          int std_int[3];
          float std[3],mean[3];
          float meansqr[3],color[3];
          unsigned long q[3],Summean[3],Summeansqr[3];
```

```
    unsigned int i;
    float roundstd;
    int roundmean;
    unsigned rounding;
    int headersize;
    float subl,midl,subh,midh,colorl[3],colorh[3];

public:
    int ColorL[3],ColorH[3];
    int mean_int[3];
    int *dot[3];

    BTC_MAE(){
     headersize=16;

    rounding=0xffff;
    roundstd=1;
    roundmean=1;
    length=0;
    block=0;
    dot[BlueC]=new int [M];
    dot[GreenC]=new int [M];
    dot[RedC]=new int [M];
    q[0]=0;
    q[1]=0;
    q[2]=0;
    Summean[0]=0;
    Summean[1]= 0;
    Summean[2]=0;
    Summeansqr[0]=0;
    Summeansqr[1]=0;
    Summeansqr[2]=0;
    i=0;
   }
   void Reset(){
     headersize=16;
```

```
 rounding=0xffff;
 roundstd=1;
 roundmean=1;
 length=0;
 block=0;
 delete dot[BlueC];
 delete dot[GreenC];
 delete dot[RedC];
 dot[BlueC]=new int [M];
 dot[GreenC]=new int [M];
 dot[RedC]=new int [M];
 q[0]=0;
 q[1]=0;
 q[2]=0;

 Summean[0]=0;
 Summean[1]= 0;
 Summean[2]=0;
 Summeansqr[0]=0;
 Summeansqr[1]=0;
 Summeansqr[2]=0;
 i=0;
}

void Sum(int count ,int pixel)
{
 dot[GrayC][count]=pixel;
 Summean[GrayC]=pixel+Summean[GrayC];

}
void Sum(int count,int R,int G,int B)
{
 dot[RedC][count]=R;
 dot[GreenC][count]=G;
 dot[BlueC][count]=B;
 Summean[RedC] =R+Summean[RedC];
```

```
            Summean[GreenC]=G+Summean[GreenC];
            Summean[BlueC] =B+Summean[BlueC];



          }
          void mod_process(int Cp);
          void demod_process(int Cp);
          void display();
          void displayC();

          unsigned long process(unsigned int box,int size);
          unsigned long process(unsigned int box,int size,int bits);
          void processing(int size);
          void processing(int size,int bits);

};
void BTC_MAE::mod_process(int Cp)
{
 mean[Cp]    = Summean[Cp]/M;
 mean_int[Cp]= ROUNDu(mean[Cp])/roundmean;
 length+=headersize;
}

void BTC_MAE::demod_process(int Cp)
{
    mean_int[Cp]*=roundmean;

  for(i=0;i<M;i++)
   {
    if(dot[Cp][i]>=(mean_int[Cp]) )q[Cp]++;
   }

   if(q[Cp]==0||q[Cp]==M)
   {
    ColorL[Cp]=mean_int[Cp];
    ColorH[Cp]=ColorL[Cp];
```

```
}else{
      ColorL[Cp]=0;
      ColorH[Cp]=0;
      for(i=0;i<M;i++)
      {
       if(dot[Cp][i]>=mean_int[Cp])ColorH[Cp]+=dot[Cp][i];
       else ColorL[Cp]+=dot[Cp][i];
      }

      colorl[Cp]=(float)ColorL[Cp]/((float)M-(float)q[Cp]);
      if(color[Cp]<0)ColorL[Cp]=0;
      else ColorL[Cp]=ROUNDu(color[Cp]);
      colorh[Cp]=(float)ColorH[Cp]/((float)q[Cp]);
      if(color[Cp]>255)ColorH[Cp]=255;
      else ColorH[Cp]=ROUNDu(color[Cp]);
      midl=258;
      midh=258;
      for(i=0;i<M;i++)
       {
        subl=fabs((float)dot[Cp][i]-colorl[Cp]);
        subh=fabs((float)dot[Cp][i]-colorh[Cp]);


        if(subl<midl)
        {
         ColorL[Cp]=dot[Cp][i];
         midl=subl;
        }

        if(subh<midh)
        {
         ColorH[Cp]=dot[Cp][i];
         midh=subh;
        }
       }
```

```
        }
        ColorH[Cp]=ColorH[Cp]/roundmean;
        ColorL[Cp]=ColorL[Cp]/roundmean;

        ColorH[Cp]=ColorH[Cp]*roundmean;
        ColorL[Cp]=ColorL[Cp]*roundmean;
        if(q[Cp]!=0&&q[Cp]!=M)length+=M;
    }


    void BTC_MAE::display()
    {

        for(i=0;i<M;i++){
        if(dot[GrayC][i]>=(mean_int[GrayC]))DotPixelB(1,block,i,ColorH[GrayC]);
                else DotPixelB(1,block,i,ColorL[GrayC]);
        }
    }

    void BTC_MAE::displayC()
    {

            for(i=0;i<M;i++)
            {
              if(dot[BlueC][i]>=(mean_int[BlueC]))dot[BlueC][i]=ColorH[BlueC];
              else dot[BlueC][i]=ColorL[BlueC];
              if(dot[GreenC][i]>=(mean_int[GreenC]))dot[GreenC][i]=ColorH[GreenC];
              else dot[GreenC][i]=ColorL[GreenC];
              if(dot[RedC][i]>=(mean_int[RedC]))dot[RedC][i]=ColorH[RedC];
              else dot[RedC][i]=ColorL[RedC];

            }
            for(i=0;i<M;i++)D_DotPixelB(1,block,i,dot[RedC][i],dot[GreenC][i],dot[BlueC][i]);

    }
```

```
void BTC_MAE::processing(int size)
{
    if(size){
  //          roundstd=8.5;
              roundmean=4;
  //          rounding=0x000f;
              headersize=12;
     }

     mod_process(GrayC);
     demod_process(GrayC);
//   display();
//   return(length);
}

unsigned long BTC_MAE::process(unsigned int box,int size)
{
 processing(size);
 display();
 return(length);
}

void BTC_MAE::processing(int size,int bits)
{
    if(size){
  //          roundstd=8.5;
              roundmean=4;
  //          rounding=0x000f;
              headersize=12;
     }

     mod_process(BlueC);
     mod_process(GreenC);
     mod_process(RedC);
     demod_process(BlueC);
     demod_process(GreenC);
```

```
    demod_process(RedC);
//    displayC();

//    return(length);
}
unsigned long BTC_MAE::process(unsigned int box,int size,int bits)
{
 block=box;
 processing(size,bits);
 displayC();
 return(length);
}
/*********************************************************/

class BTC_AM
{
 private:
        unsigned long length;
        unsigned int block;
        int ColorL[3],ColorH[3];
        int std_int[3],mean_int[3];
        float std[3],mean[3];
        float meansqr[3],color[3];
        unsigned long q[3],Summean[3],Summeansqr[3];
        unsigned int i;
        float roundstd;
        int roundmean;
        unsigned rounding;
        int headersize;
        float subl,midl,subh,midh,colorl[3],colorh[3];
        BTC_RMS rootmean;
        BTC_MAE meanabs;

public:
        int *dot[3];
        BTC_AM(){
```

```
 headersize=16;
rounding=0xffff;
roundstd=1;
roundmean=1;
length=0;
block=0;
dot[BlueC]=new int [M];
dot[GreenC]=new int [M];
dot[RedC]=new int [M];
q[0]=0;
q[1]=0;
q[2]=0;
Summean[0]=0;
Summean[1]= 0;
Summean[2]=0;
Summeansqr[0]=0;
Summeansqr[1]=0;
Summeansqr[2]=0;
i=0;
}

void Reset(){
 rootmean.Reset();
 meanabs.Reset();
 headersize=16;
rounding=0xffff;
roundstd=1;
roundmean=1;
length=0;
block=0;
delete dot[BlueC];
delete dot[GreenC];
delete dot[RedC];
dot[BlueC]=new int [M];
dot[GreenC]=new int [M];
dot[RedC]=new int [M];
```

```
  q[0]=0;
  q[1]=0;
  q[2]=0;
  Summean[0]=0;
  Summean[1]= 0;
  Summean[2]=0;
  Summeansqr[0]=0;
  Summeansqr[1]=0;
  Summeansqr[2]=0;
  i=0;
 }

 void Sum(int count,int pixel)
 {
  dot[GrayC][count]=pixel;
  rootmean.Sum(count,pixel);
  meanabs.Sum(count,pixel);;

 }
 void Sum(int count,int R,int G,int B)
 {
   dot[RedC][count]=R;
   dot[GreenC][count]=G;
   dot[BlueC][count]=B;
   rootmean.Sum(count,R,G,B);
   meanabs.Sum(count,R,G,B);

 }
 void Gray_process();
 void Color_process();
 void display();
 void displayC();
 unsigned long process(unsigned int box,int size);
 unsigned long process(unsigned int box,int size,int bits);
};
void BTC_AM::Gray_process()
```

```
{
   rootmean.processing(0);
   meanabs.processing(0);
   mean_int[GrayC]=rootmean.mean_int[GrayC];
   color[GrayC]=ROUNDu(((float)rootmean.ColorL[GrayC]+(float)meanabs.ColorL[GrayC])/2);
   if(color[GrayC]<0)ColorL[GrayC]=0,else ColorL[GrayC]=ROUND(color[GrayC]);
   ColorH[GrayC]=ROUND(((float)rootmean.ColorH[GrayC]+(float)meanabs.ColorH[GrayC])/2);
   length=headersize;
   if(ColorL[GrayC]!=ColorH[GrayC])length+=M;
}

void BTC_AM::Color_process()
{
   rootmean.processing(0,fi.bits);
   meanabs.processing(0,fi.bits);
   mean_int[RedC]=rootmean.mean_int[RedC];
   mean_int[GreenC]=rootmean.mean_int[GreenC];
   mean_int[BlueC]=rootmean.mean_int[BlueC];
   ColorL[RedC]=(unsigned char)ROUND(((float)rootmean.ColorL[RedC]+(float)meanabs.ColorL[RedC])/2);
   ColorL[GreenC]=(unsigned char)ROUND(((float)rootmean.ColorL[GreenC]+(float)meanabs.ColorL[GreenC])/2);
   ColorL[BlueC]=(unsigned char)ROUND(((float)rootmean.ColorL[BlueC]+(float)meanabs.ColorL[BlueC])/2);

   ColorH[RedC]  =(unsigned char)ROUND(((float)rootmean.ColorH[RedC]+(float)meanabs.ColorH[RedC])/2);
   ColorH[GreenC]=(unsigned char)ROUND(((float)rootmean.ColorH[GreenC]+(float)meanabs.ColorH[GreenC])/2);
   ColorH[BlueC] =(unsigned char)ROUND(((float)rootmean.ColorH[BlueC]+(float)meanabs.ColorH[BlueC])/2);
   length=headersize;
   if(ColorL[RedC]!=ColorH[RedC])length+=M;
   if(ColorL[GreenC]!=ColorH[GreenC])length+=M;
   if(ColorL[BlueC]!=ColorH[BlueC])length+=M;
}

void BTC_AM::display()
{
```

```
    for(i=0;i<M;i++){
     if(dot[GrayC][i]>=(mean_int[GrayC]))DotPixelB(1,block,i,ColorH[GrayC]);
          else DotPixelB(1,block,i,ColorL[GrayC]);
    }
}

void BTC_AM::displayC()
{

        for(i=0;i<M;i++)
        {
         if(dot[BlueC][i]>=(mean_int[BlueC]))dot[BlueC][i]=ColorH[BlueC];
         else dot[BlueC][i]=ColorL[BlueC];
         if(dot[GreenC][i]>=(mean_int[GreenC]))dot[GreenC][i]=ColorH[GreenC];
         else dot[GreenC][i]=ColorL[GreenC];
         if(dot[RedC][i]>=(mean_int[RedC]))dot[RedC][i]=ColorH[RedC];
         else dot[RedC][i]=ColorL[RedC];

       }
        for(i=0;i<M;i++)D_DotPixelB(1,block,i,dot[RedC][i],dot[GreenC][i],dot[BlueC][i]);

}

unsigned long BTC_AM::process(unsigned int box,int size)
{

  block=box;
   if(size){

        roundmean=4;
        headersize=12;
   }

   Gray_process();
   display();
   return(length);
```

```
BTC_MAE MAE_method;
BTC_AM  Am_method;

unsigned long BTC_process(int method;int ssss)
{
 unsigned int block=0;
 int pigment,i;
 unsigned long length=0;
 if(fi.bits==8)
 {
   for(block=0;block<BlockTotal;block++){
        Standard.Reset();
        rms_method.Reset();
        MAE_method.Reset() ;
        Am_method.Reset();
      for(i=0;i<M;i++){
        pigment=ValuepixelB('g',block,i);
        switch(method){
         case 1 :Standard.Sum(i,pigment);break;
         case 2 :rms_method.Sum(i,pigment);break;
         case 3 :MAE_method.Sum(i,pigment);break;
         case 4 :Am_method.Sum(i,pigment);break;
        }
      }

      switch(method){
        case 1 :length=length+Standard.process(block,ssss);break;
        case 2 :length=length+rms_method.process(block,ssss);break;
        case 3 :length=length+MAE_method.process(block,ssss);break;
        case 4 :length=length+Am_method.process(block,ssss);break;
      }
   }
 }else{

   for(block=0;block<BlockTotal;block++){
        Standard.Reset();
```

```
            rms_method.Reset();
            MAE_method.Reset() ;
            Am_method.Reset();
          for(i=0;i<M;i++){
            ValuepixelB('A',block,i);
            switch(method){
             case 1 :Standard Sum(i,(int)colors.Red,(int)colors.Green,(int)colors.Blue);break;
             case 2 :rms_method.Sum(i,(int)colors.Red,(int)colors Green,(int)colors Blue);break;
             case 3 :MAE_method Sum(i,(int)colors.Red,(int)colors Green,(int)colors Blue),break;
             case 4 :Am_method.Sum(i,(int)colors.Red,(int)colors Green,(int)colors.Blue);break;
            }
          }

          switch(method){
            case 1 :length=length+Standard.process(block,ssss,fi.bits);break;
            case 2 :length=length+rms_method.process(block,ssss,fi.bits);break;
            case 3 :length=length+MAE_method.process(block,ssss,fi.bits);break;
            case 4 :length=length+Am_method.process(block,ssss,fi.bits);break;
          }
        }
      }

      return(length);
    }

    /*************************************************************************/
    /*************************************************************************/
    /************************** Menagement Function **************************/
```

```
/*************************************************************************/
/*************************************************************************/


void BTC(void)
{
  unsigned key;
  char ch;
  unsigned long length=0;
//  unsigned int block=0;
  char *mess;
  mess=new char [80];
  sprintf(txmp,"Please Select Compression Method(F1:BTCSIF2:RMSIF3:MAEIF4:MiniF5:DCT)");
  Screen.Operating();
  Screen.Poperating(txmp);
  key=bioskey(0);
  switch(key){
    case F2_Key:sprintf(mess,"BTC Minimum RMS ");break;
    case Alt_F2:sprintf(mess,"HBTC Minimum RMS ");break;
    case F3_Key:sprintf(mess,"BTC Minimum MAE ");break;
    case Alt_F3:sprintf(mess,"HBTC Minimum MAE ");break;
    case F4_Key:sprintf(mess,"BTC Average Min");break;
    case Alt_F4:sprintf(mess,"HBTC Average Min");break;
    case F5_Key:sprintf(mess,"DCT ");break;
    case Alt_F1:sprintf(mess,"HStandard BTC");break;
    case F1_Key:
    default:    sprintf(mess,"Standard BTC");break;
  }

  sprintf(txmp,"choice level Compression");
  Screen.Operating();
  Screen.Poperating(txmp);
  ch=getch();
// ch=(char)(bioskey(0)&&0xff);
  Screen.Operating();
```

```
switch(ch){
    case '1':{
                M=4;BlockTotal=16384;
                N=2,BlockWidth=128;
                strcat(mess," 2x2");
                break;
            }
    case '2':{
                M=16;BlockTotal=4096;
                N=4;BlockWidth=64;
                strcat(mess," 4x4");
                break;
            }
    case '3':{
                M=64;BlockTotal=1024;
                N=8;BlockWidth=32;
                strcat(mess," 8x8");
                break;
               }
    case '4' :{
                M=256;BlockTotal=256;
                N=16;BlockWidth=16;
                strcat(mess," 16x16");
                break;
              }
//  case '5' :{
//              M=1024;BlockTotal=64;
//              N=32;BlockWidth=8;
//              windows_=1;
//              strcat(mess," 32x32");
//              break;
//            }
    default :{
                M=16;BlockTotal=4096;
                N=4;BlockWidth=64;
                strcat(mess," 4x4");
```

```
                    break;
                }
        }
        Screen.Win[1].Title();
        Screen.Win[1].Ptitle(mess);
        timein=clock();
        status('C');
         switch(key){
         case F2_Key:length=BTC_process(2);break;
         case Alt_F2:length=BTC_process(2,1);break;
         case F3_Key:length=BTC_process(3);break;
         case Alt_F3:length=BTC_process(3,1);break;
         case F4_Key:length=BTC_process(4);break;
         case Alt_F4:length=BTC_process(4,1);break;
    //   case F5_Key:length=DCT();break;
        case Alt_F1:length=BTC_process(1,1);break;
        case F1_Key:
        default:    length=BTC_process(1);

                    break;
        }
        timeout=clock();
        Compress_Bpp(1,length);
        Compress_Ratio(1,length);
        Compress_Time();
        status('P');
        Error_cal(0,1);
        Error_(RMS error,MAE error,SNR.error,SNR.dB);
        status('R');
}
```