การบีบอัดภาพนิ่งด้วยวิธี JPEG ชนิด Sequential Baseline System

JPEG Image Compression using Sequential Baseline System

โดย

นางสาววรลักษณ์ คงเด่นฟ้า

นางสาววาสนา กล้าการนา

ตรวจสอบแล้ว 04

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การบีบอัดภาพนิ่งด้วยวิธี JPEG ชนิด Sequential Baseline System

JPEG Image Compression using Sequential Baseline System

โดย

นางสาววรลักษณ์ คงเด่นฟ้า 37014376

นางสาววาสนา กล้าการนา 37014393

อาจารย์ที่ปรึกษา

ผศ.ดร. ไกรสิน ส่งวัฒนา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การบีบอัดภาพนิ่งด้วยวิธี JPEG ชนิด Sequential Baseline System**

JPEG Image Compression using Sequential Baseline System

ผู้จัดทำ

1. นางสาววรลักษณ์ คงเด่นฟ้า 37014376
2. นางสาววาสนา กล้าการนา 37014393

อาจารย์ที่ปรึกษา

(ผศ.ดร.ไกรสิน ส่งวัฒนา)

การบีบอัดภาพนิ่งด้วยวิธี JPEG ชนิด Sequential Baseline System

JPEG Image Compression using Sequential Baseline System

โดย นางสาววรลักษณ์ คงเด่นฟ้า 37014376

นางสาววาสนา กล้าการนา 37014393

อาจารย์ที่ปรึกษา ผศ.ดร.ไกรสิน ส่งวัฒนา

บทคัดย่อ

โครงงานนี้เป็นการศึกษาการบีบอัดภาพนิ่งด้วยวิธี JPEG (Joint Photographic Expert Group) ชนิด Sequential Baseline System ซึ่งนำภาพนิ่งจากไฟล์ข้อมูลแบบ Windows Bitmap file (.bmp) ชนิด 1,4,8 หรือ 24 บิท/จุดภาพ มาทำการบีบอัดด้วยวิธีการของ JPEG โดยการแบ่งภาพออกเป็นบล็อกย่อยๆเรียกว่า MCU จากนั้นจะนำบล็อกย่อยๆนั้นมาเข้ากระบวนการบีบอัดโดยอาศัยหลักการของ Forward Discrete Cosine Transform (FDCT) ซึ่งจะทำการแปลงสัญญาณภาพใน Spatial Domain ไปเป็นค่าสัมประสิทธิ์การแปลงใน Frequency Domain และจะนำค่าสัมประสิทธิ์การแปลงนี้มาทำการควอนไตเซชัน ซึ่งจะลดค่าสัมประสิทธิ์ลง เพื่อจะลดจำนวนบิทที่ใช้ในการเข้ารหัส และนำค่าที่ได้จากการควอนไตเซชันมาทำการอ่านแบบ Zigzag เพื่อให้ข้อมูลมาอยู่ในรูปแบบที่สะดวกต่อการเข้ารหัส จากนั้นจะทำการเข้ารหัสข้อมูลโดยใช้วิธีการของ Huffman encoding และจะเก็บรหัสข้อมูลเหล่านี้ไว้ในรูปของ JFIF (JPEG File Interchange Format) file (.jpg)

ABSTRACT

This project is the studying of image compression by using JPEG (Joint Photo graphic Expert Group) technique type sequential baseline system. Image from Windows Bitmap file (.bmp) type 1,4,8 or 24 bits/pixel are compressed by JPEG technique. By dividing image to many blocks then compress each block by using FDCT (Forward Discrete Cosine Transform) which will transform image signal in spatial domain into the coefficient in frequency domain, quantization will decrease FDCT coefficient to reduce the number of bits for encoding, zigzag, encoding by using Huffman encoding. And write code to JFIF file (JPEG File Interchange Format) file (.jpg).

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1

บทนำ

รูปแบบข้อมูลภาพโดยทั่วไป มีรูปแบบดังนี้คือ GIF IFF/LBM WPG BMP PIC TGA PCX และ TIFF สำหรับรูปแบบใหญ่ ๆ ที่นิยมใช้กันมากคือ PCX GIF BMP และ TIFF แต่ รูปแบบหนึ่งที่ได้รับความนิยมมากที่สุด คือ BMP หรือบิทแมพ สาเหตุก็เพราะว่าข้อมูลแบบบิทแมพนั้นไม่มีการบีบข้อมูล (compression) จึงมีความรวดเร็วและสะดวกในการอ่านข้อมูลอย่างมากและเป็นรูปแบบข้อมูลกลางที่ใช้ในการแลกเปลี่ยนระหว่างกันของโปรแกรมประยุกต์ ต่าง ๆ ใช้ในการส่งข้อมูลไปให้อุปกรณ์ต่าง ๆ เช่น อุปกรณ์การแสดงผล เป็นต้น แต่ในรูปแบบบิทแมพนี้ต้องเปลืองเนื้อที่ฮาร์ดดิสในการจัดเก็บข้อมูลมากกว่ารูปแบบอื่นที่ใช้การเข้ารหัสบีบข้อมูล เราได้นำเทคนิคการบีบอัดภาพนิ่งแบบเจเป็กมาใช้บีบอัดภาพบิทแมพไฟล์เพื่อลดขนาดข้อมูลภาพทำให้ลดเนื้อที่ในการจัดเก็บ

เจเป็กเป็นมาตรฐานการบีบอัดและขยายข้อมูลภาพนิ่งที่กำหนดโดย Joint Photographic Experts Groups ซึ่งเป็นกลุ่มที่ถูกจัดตั้งขึ้นโดยความพยายามของ ISO และ CCITT การบีบอัดภาพนิ่งแบบเจเป็กนั้นมีพื้นฐานอยู่ที่การแปลงแบบดิสครีตโคโซน์ทรานสฟอร์มหรือ DCT (Discrete Cosine Transform) ซึ่งในเทคนิคของการบีบอัดข้อมูลนั้นแบ่งออกได้เป็น 2 ประเภทหลัก คือ

1. การบีบอัดข้อมูลที่ไม่มีการสูญเสีย (lossless compression) เป็นวิธีการที่ข้อมูลข่าวสารเมื่อผ่านกระบวนการนี้แล้วผลลัพธ์ของข้อมูลที่ได้จะเหมือนข้อมูลเริ่มต้นเสมอ

2. การบีบอัดข้อมูลแบบยอมให้มีการสูญเสีย (lossy compression) วิธีนี้ข้อมูล ผลลัพธ์ที่ได้หลังจากการขยายข้อมูลกลับจะมีความแตกต่างกับข้อมูลก่อนทำการบีบอัดเริ่มต้น

สำหรับระบบที่เราทำการศึกษาคือระบบการบีบอัดข้อมูลที่ยอมให้มีการสูญเสียข้อมูลบางส่วน (lossy) ในขั้นตอนการบีบอัดข้อมูล ซึ่งเราเลือกใช้ระบบซีเควนเชียลเบสไลน์ (Sequential baseline system) ซึ่งระบบนี้มีพื้นฐานอยู่ที่การแปลงข้อมูลแบบดิสครีตโคซายน์ทรานสฟอร์ม

มาตรฐานการบีบอัดภาพนิ่งแบบเจเป็กนั้นมี 4 โหมดการทำงาน ดังนี้

1) Sequential DCT - based encoding คือองค์ประกอบของภาพจะถูกเข้ารหัสแบบ single left-to-right และ top-to-bottom scan

2) progressive DCT - based encoding เข้ารหัสแบบ multiple scan

3) การเข้ารหัสแบบไม่มีการสูญเสีย (lossless encoding) ภาพจะถูกเข้ารหัสเพื่อให้ได้ภาพผลลัพธ์ที่เหมือนเดิม

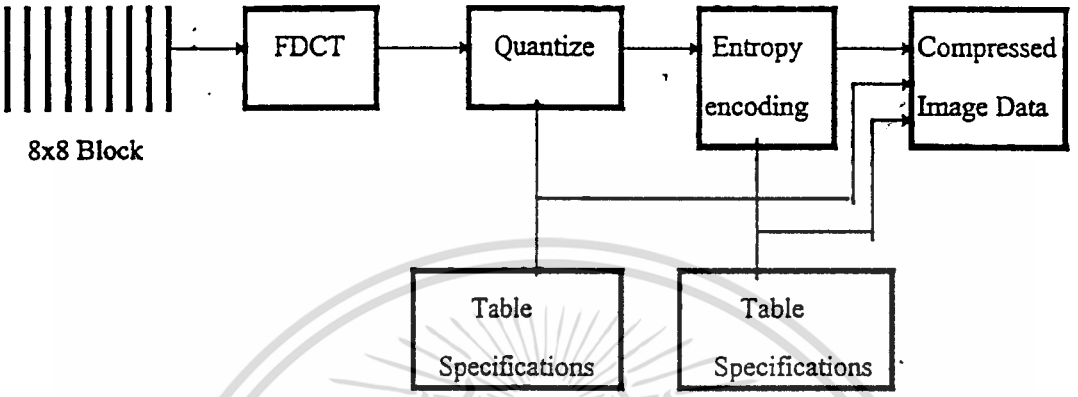
4)hierarchical encoding ภาพจะถูกเข้ารหัสแบบ multiple resolution

สำหรับระบบที่เราทำการศึกษาคือระบบการบีบอัดข้อมูลแบบที่ยอมให้มีการสูญเสียข้อมูลบางส่วน ในขั้นตอนการบีบอัดข้อมูล ซึ่งเราเลือกใช้ระบบ Sequential Baseline System ซึ่งระบบนี้มีพื้นฐานอยู่ที่การแปลงข้อมูลแบบ DCT ซึ่งนำภาพนิ่งจาก Windows Bitmap file(.bmp) ชนิด 1,4,8 หรือ 24 บิท/พิกเซล มาเข้าขั้นตอนของฟอร์เวิร์ดดิสครีตโคซายน์ทรานสฟอร์มหรือ FDCT (Forward Discrete Cosine Tranform) ซึ่งจะทำการแปลงสัญญาณภาพใน Spatial Domain ไปเป็นค่าสัมประสิทธิ์การแปลงในโดเมนทางความถี่ (Frequency Domain) แล้วผ่านขั้นตอนต่อไปคือการควอนไตเซชั่น ซึ่งจะลดค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีต

โคซายน์ เพื่อลดจำนวนบิทที่ใช้ลง และนำข้อมูลที่ได้ไปทำการเรียงข้อมูลแบบซิกแซก (Zigzag) แล้วจึงนำไปเข้ารหัสแบบฮัฟแมนในขั้นตอนของการควอนไตเซชั่นนั้นจะมีการแปลงสีจากระบบสี RGBในไฟล์ชนิดบิทแมพไปเป็นระบบสี YCbCr และทำการซับแซมปลิงในอัตราส่วน Y:Cb:Cr เท่ากับ 4:1:1 และ ในอัตราส่วน 4:2:2 ด้วย และจัดเก็บข้อมูลที่ได้ไว้ในรูป JFIF (JPEG File Interchange Format) file (.jpg)

บทที่ 2
ทฤษฎีและหลักการ

2.1 กระบวนการเข้ารหัสแบบเจเป็ก (JPEG CODEC)



รูปที่ 2.1 แสดงการเข้ารหัสแบบเจเป็ก

ลำดับของการเข้ารหัสแบบเจเป็กเป็นไปตามรูปที่ 2.1 โดยที่ภาพต้นฉบับจะถูกแบ่งออกเป็นบล็อก (Block) เล็ก ๆ ขนาด 8x8 พิกเซล ถ้าเป็นภาพที่แสดงผลด้วยจำนวน n บิทต่อพิกเซลแล้ว ค่าข้อมูลพิกเซลจะมีค่าตั้งแต่ 0 ถึง 2^n - 1 ก่อนที่จะผ่าน การแปลงฟอร์เวิร์ดดิสครีตโคซายน์ (Forward Discrete Cosine Transform หรือ FDCT) จะมีการสร้างค่าของพิกเซลให้เป็นจำนวนเต็มแบบมีเครื่องหมาย ซึ่งมีค่าตั้งแต่ -2^n ถึง 2^n - 1 ตัวอย่าง เช่นภาพแบบเกรย์สเกลที่มี 8 บิทต่อพิกเซล (ความเข้ม 2^8 = 256 ระดับ) มี ค่า 0 - 255 เมื่อจะผ่านการแปลงต้องทำข้อมูลให้เป็น - 128 ถึง 127 ก่อน

เมื่อผ่านการแปลงโคซายน์แบบไม่ต่อเนื่องแล้วเอาท์พุทที่ได้จะเป็นค่าสัมประสิทธิ์ทางความถี่ ต่อมาจะเป็นการควอนไตซ์ค่าสัมประสิทธิเหล่านี้ ในขั้นตอนนี้เองที่จะมีการลดขนาดของข้อมูลและเกิดมีความผิดพลาดจากการควอนไตซ์ขึ้น ในการควอนไตซ์นี้จะเป็นแบบไม่เป็นยูนิฟอร์ม (Non - Uniform) กล่าวคือ ค่าสัมประสิทธิ์แต่ละค่าจะถูกควอนไตซ์ด้วยระดับการควอนไตซ์ (Step Size) ที่ไม่เท่ากันเสมอ ขนาดของระดับขั้นหรือสเตป (Step) ในการควอนไตซ์จะถูกกำหนดโดยค่าในตารางเรียกว่าเป็นตารางค่าควอนตัมนั่นเอง ซึ่งค่าในตารางนี้จะเป็นค่าที่ใช้เฉพาะกับสัมประสิทธิ์ ที่ได้จากการแปลงโคซายน์แบบไม่ต่อเนื่องแต่ละค่าเท่านั้น

ค่าสัมประสิทธิ์ที่ถูกควอนไตซ์และจะถูกนำมาจัดตามลำดับซิกแซก เพื่อความสะดวกในการเข้ารหัสซึ่งอาจใช้วิธีของฮัฟแมน(Huffman Encoding) หรือวิธีการเข้ารหัสแบบความยาวของรหัสไม่คงที่ (Variable Length Encoding) เอาท์พุทจากกระบวนการนี้จะเป็นข้อมูลภาพที่ถูกลดขนาดเรียบร้อยแล้ว

จากบล็อกไดอะแกรม การเข้ารหัสแบบเจเป็กประกอบด้วยบล็อกที่สำคัญ 3 บล็อก คือ

1) ฟอร์เวิร์ดดิสครีตโคซายน์ทรานสฟอร์ม (Forward Discrete Cosine Transform (FDCT))
2) ควอนไตเซชัน(Quantization)
3) การเข้ารหัสเอนโทรปี (Entropy encoding)

ที่อินพุทของวงจรเข้ารหัสแซมเปิลของข้อมูลภาพที่ไม่มีเครื่องหมาย (original unsigned samples) ที่อยู่ในช่วง $[0, 2^{P-1}]$ จะถูกเปลี่ยนให้เป็นจำนวนเต็มแบบมีเครื่องหมายซึ่งอยู่ในช่วง $[-2^{P-1}, 2^{P-1}-1]$ เช่น ภาพแบบเกรย์สเกลมี P=8 แซมเปิลซึ่งอยู่ในช่วง [0, 255] จะถูกเปลี่ยนให้อยู่ในช่วง [-128, +127]

จากนั้นข้อมูลภาพจะถูกแบ่งเป็นบล็อกแบบ 8 x 8 และแซมเปิลจากแต่ละบล็อกจะถูกแปลงให้อยู่ในโดเมนของความถี่ โดยใช้การแปลงแบบดิสครีตโคซายน์แปลงสัญญาณภาพใน spatial domain ไปเป็นค่าสัมประสิทธิ์ของการแปลงโดเมนของความถี่ โดยมีสมการของการแปลงค่าเป็น

$$C(u,v) = \frac{1}{\sqrt{2N}}\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad (2.1)$$

และมีสมการการแปลงกลับเป็น

$$f(x,y) = \frac{1}{\sqrt{2N}}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad (2.2)$$

เมื่อ $C(u,v)$ คือ ค่าสัมประสิทธิ์การแปลงแบบดิสครีตโคซายน์

$f(x,y)$ คือ ค่าของระดับสีในพิกเซล

โดย $\alpha(i) = \frac{1}{\sqrt{2}}$ เมื่อ $i = 0$ และ $\alpha(i) = 1$ เมื่อ $i > 0$

ทั้งนี้เนื่องจากสมการ (1) ในการแปลงค่าสัมประสิทธิ์การแปลงแบบดิสครีตโคซายน์ ถ้าจำนวนข้อมูลยิ่งมาก (N ยิ่งมาก) จะทำให้ใช้เวลาและกำลังในการคำนวณมาก ดังนั้นเจเป็กจึงกำหนดการแปลงภาพโดยแบ่งการแปลงเป็นบล็อกย่อย ๆ ของภาพขนาดบล็อกละ 8 x 8 พิกเซล (N=8)

ขั้นตอนต่อไปคือควอนไตเซชันซึ่งเป็นการปรับลดค่าสัมประสิทธิ์ที่ได้จากการทำการแปลงแบบดิสครีตโคซายน์เพื่อลดจำนวนบิทที่ใช้ในการเก็บซึ่งมีผลทำให้ความแม่นยำ (precision) ของข้อมูลลดลง คือ ทำให้เกิดการสูญเสียข้อมูลบางส่วนไป โดยสัมประสิทธิ์จากการแปลงแบบดิสครีตโคซายน์ทั้ง 64 ค่าจะถูกทำการควอนไตซ์โดยใช้ตารางการควอนไตซ์ขนาด 64 อิลิเมนต์ (element) การควอนไตเซชันจะลดค่าแอมปลิจูดของสัมประสิทธิ์ที่มีผลกับคุณภาพของภาพน้อยที่สุด เพื่อให้เป็นไปตามหลักการเพิ่มค่าสัมประสิทธิ์ที่มีค่าศูนย์ให้มากขึ้น และลดค่าสัมประสิทธิ์ที่ไม่มีความสำคัญลง การควอนไตเซชันกระทำได้ดังสมการต่อไปนี้

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Q(u,v)}\right] \qquad (2.3)$$

และมีสมการดิควอนไตเซชันดังนี้

$$T^*(u,v) = \hat{T}(u,v)Q(u,v) \qquad (2.4)$$

เมื่อ $T(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์

$\hat{T}(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ เมื่อถูกควอนไตซ์

$T^*(u,v)$ คือ ค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ เมื่อถูกดิควอนไตซ์

$Q(u,v)$ คือ ค่าควอนไตเซชัน

และ *round* คือ การหาจำนวนเต็มที่มีค่าใกล้เคียงที่สุดโดย $0 \le u \le 7, 0 \le v \le 7$

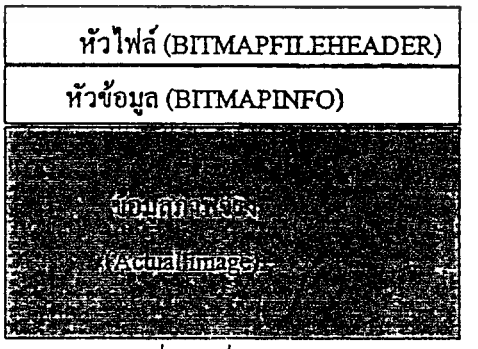
หลังจากผ่านการควอนไตเซชั่นแล้ว ขั้นตอนต่อไปคือการเรียงข้อมูลแบบซิกแซก โดยเป็นการเรียงค่าสัมประสิทธิ์ที่ได้จากการแปลงแบบดิสครีตโคซายน์ให้เป็นในบล็อกข้อมูล 8 x 8 ค่าให้เป็นแถวข้อมูล 64 ค่า โดยเรียงลำดับจากค่าสัมประสิทธิ์ของความถี่ต่ำไปหาค่าสัมประสิทธิ์ของความถี่สูงซึ่งทำให้ค่าสัมประสิทธิ์ที่มีค่าศูนย์ส่วนใหญ่ในช่วงความถี่สูงอันเนื่องมาจากการควอนไตเซชันมาอยู่ติดกันซึ่งจะเป็นประโยชน์ในการเข้ารหัสในขั้นตอนต่อไป ขั้นตอนสุดท้ายคือ การเข้ารหัสแบบเอนโทรปี (entropy encoding) ซึ่งเป็นการเข้ารหัสค่าที่ได้จากการควอนไตเซชันให้อยู่ในรูปลำดับเลขฐานสองที่กระชับขึ้น ซึ่งมาตรฐานเจเป็กมีการเข้ารหัสเอนโทรปี 2 แบบ คือ การเข้ารหัสฮัฟแมน (Huffman Encoding) และ การเข้ารหัสเชิงอริธเมติก (Arithmetic Encoding) ซึ่งในระบบซีเควนเชียลเบสไลน์ จะใช้การเข้ารหัสแบบฮัฟแมนเท่านั้น โดยวิธีการเข้ารหัสฮัฟแมน นั้นเป็นวิธีการเข้ารหัส (encode) ข้อมูลด้วยการสร้างรหัส ขนาดต่าง ๆ ขึ้นจากสถิติของจำนวนข้อมูลที่เกิดขึ้น เพื่อนำรหัสที่สร้างขึ้นนั้นไปใช้แทนตัวข้อมูลเดิม โดยพยายามสร้างรหัส ที่มีขนาดสั้นสำหรับใช้แทนตัวข้อมูลที่เกิดขึ้นจำนวนมากและรหัส ที่มีขนาดยาวขึ้นเพื่อแทนตัวข้อมูลที่เกิดขึ้นน้อยลงลดหลั่นกันไป สำหรับวิธีการเข้ารหัสฮัฟแมน ที่ใช้ใน JPEG นั้นได้มีการอาศัยเทคนิคการการเข้ารหัส วิธีอื่นมาช่วยด้วยคือ การเข้ารหัสแบบแวริเอเบิลเลนจ์ (variable-length encoding) และ การเข้ารหัสแบบรันเลนจ์ (runlength encoding) ซึ่งจะแบ่งการเข้ารหัส เป็น 2 ส่วนคือ การเข้ารหัส ค่า DC และค่า AC ตามลำดับ

2.2 รูปแบบข้อมูลภาพแบบ BMP

รูปแบบสำหรับใช้กับโปรแกรมระบายสีในวินโดว์เวอร์ชัน (version) 3.0 เป็นต้นมา โดยรูปแบบข้อมูลแบบนี้สามารถเป็นข้อมูลสีได้ ตั้งแต่ 1 ถึง 24 บิท มีส่วนหัว (header) ในการบอกรายละเอียดต่าง ๆ ของภาพ โดยกำหนดในลักษณะโครงสร้าง(structure) ในภาษาระดับสูงที่นี้ใช้ภาษาซี ข้อมูลภาพในรูปแบบบิทแมพประกอบด้วยสามส่วนคือ

- หัวไฟล์ (BITMAPFILEHEADER)
- หัวข้อมูล (BITMAPINFO) ซึ่งรวมทั้งข้อมูลต่าง ๆ และพาเลตของสี (color palette)
- ข้อมูลภาพจริง

รูปที่ 2.2 แสดงส่วนประกอบกันของโครงสร้างที่ใหญ่ที่สุดของภาพถ่ายในรูปแบบบิทแมพ ซึ่งข้อมูลพาเลตสีและข้อมูลภาพจะเปลี่ยนโครงสร้างไปตามรูปแบบของจำนวนสีของภาพและวิธีการถอดรหัส ที่ใช้ในการบีบข้อมูลภาพ



รูปที่ 2.2 แสดงโครงสร้างที่ใหญ่ที่สุดของภาพถ่ายในรูปแบบบิทแมพ

2.2.1 หัวไฟล์ (BITMAPFILEHEADER)

หัวของไฟล์ (BITMAPFILEHEADER) จะประกอบด้วยข้อมูลเกี่ยวกับชนิดของ รูปแบบข้อมูลภาพ ขนาดของภาพและโครงร่าง ดังนี้

BITMAPFILEHEADER ประกอบด้วย	
-WORD	bfType;
-DWORD	bfSize;
-WORD	bfReserved1;
-WORD	bfReserved2;
-DWORD	bfOffBits;

โดย bfType เป็นชนิดของรูปแบบข้อมูลภาพ ในที่นี้สำหรับบิทแมพคือ "BM"

bfSize เป็นขนาดของไฟล์ทั้งหมด

bfReserved1,2 ไม่ได้ใช้งาน ปกติกำหนดให้มีค่าเท่ากับศูนย์

bfOffBits เป็นขนาดของหัวไฟล์ทั้งหมด ใช้เพื่อข้ามไปจุดเริ่มต้นของข้อมูลภาพจริง

2.2.2 หัวข้อมูล (BITMAPINFO)

หัวข้อมูลจะเป็นตัวกำหนดขนาดต่าง ๆ (dimensions) และข้อมูลสี (color information) ดังนี้

BITMAPINFO ประกอบด้วย	
-BITMAPINFOHEADER	bmiHeader;
-RGBQUAD	bmiColors[1];

โดยที่ bmiHeader เป็นข้อมูลเกี่ยวกับขนาดต่าง ๆ จำนวนสี และรูปแบบของสี

bmiColors เป็นอาร์เรย์ข้อมูลสี RGB ที่เป็นตัวกำหนดสีในการแสดงผลของภาพ

รายละเอียดต่าง ๆ ที่อยู่ใน BITMAPINFOHEADER ดังนี้

STRUCTURE BITMAPINFOHEADER ประกอบด้วย	
-DWORD	biSize;
-DWORD	biWidth;
-DWORD	biHeight;
-WORD	biPlanes;
-WORD	biBitCount;
-DWORD	biCompression;
-DWORD	biSizeImage;
-DWORD	biXPelsPerMeter;
-DWORD	biYPelsPerMeter;
-DWORD	biClrUsed;
-DWORD	biClrImportant;

โดยที่

biSize	เป็นขนาดของส่วนหัวข้อมูล (BITMAPINFOHEADER) มีหน่วยเป็นไบต์
biWidth	เป็นความกว้างของภาพ (มีหน่วยเป็น พิกเซล/เส้น)
biHeight	เป็นความสูงของภาพ (มีหน่วยเป็น เส้น)
biPlanes	เป็นจำนวนหน้าของสี (color plane) สำหรับอุปกรณ์เป้าหมาย (targetdevice) ปกติกำหนดให้เป็น 1 เสมอ
biBitCount	จำนวนบิทต่อพิกเซล (1, 4, 8, 24) ดูรายละเอียดในตารางที่ 2.1
biCompression	ชนิดของการบีบอัด (compression) ดูรายละเอียดในตารางที่ 2.2
biSizeImage	เป็นขนาดของภาพ (มีหน่วยเป็นไบต์)
biXPelsPerMeter	ความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวนอนต่อหนึ่งเมตร
biYPelsPerMeter	ความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวตั้งต่อหนึ่งเมตร
biClrUsed	จำนวนดัชนีสี (color index) ในตารางสี (color table) มีค่าได้ 3 กรณี โดยดูรายละเอียดได้ในตารางที่ 3
biClrImportant	จำนวนความสำคัญของดัชนีสี (color index) ในการแสดงผล ถ้าเป็นศูนย์ทุกสีจะมีความสำคัญเท่ากันหมด

ค่า	อธิบาย	จำนวนอาร์เรย์สี (bmiColors)
1	บิทแมพ 2 สี (monochrome bitmap)	2 อาร์เรย์
4	บิทแมพ 16 สีสูงสุด	16 อาร์เรย์
8	บิทแมพ 256 สีสูงสุด	256 อาร์เรย์
24	บิทแมพ 16.7 ล้านสีสูงสุด	NULL (ไม่มีอาร์เรย์)

ตารางที่ 2.1 แสดงจำนวนบิทต่อพิกเซล ของ biBitCount

ชนิด	อธิบาย
BI_RGB	แสดงว่าข้อมูลไม่มีการบีบอัด
BI_RLE4	ทำการเข้ารหัสรันเลนสจ์ (run-length) ด้วยขนาด 4 บิทต่อพิกเซล โดยในสองไบต์จะประกอบด้วยไบต์นับ (count byte) และ ดัชนีสีขนาด 1 ไบต์ที่แสดงผลได้ 2 พิกเซล
BI_RLE8	ทำการเข้ารหัสรันเลนสจ์ (run-length) ด้วยขนาด 8 บิทต่อพิกเซล โดยในสองไบต์จะประกอบด้วยไบต์นับ (count byte) และ ดัชนีสีขนาด 1 ไบต์ในการแสดงผล 1 พิกเซล

ตารางที่ 2.2 แสดงรูปแบบการบีบอัด (compression formats) ของ biCompression

ค่า	อธิบาย	
0	ใช้จำนวนของสีสูงสุดเท่ากับ $2^{biBitCount}$	
ไม่เป็นศูนย์	ถ้า biBitCount<24 biClrUsed จะระบุจำนวนสีที่ใช้ในการแสดงผล เช่น เมื่อ biBitsCount เท่ากับ 8 จะมีค่า biClrUsed เท่ากับ 256 สี เป็นต้น	ถ้า biBitsCount=24 biClrUsed จะระบุขนาดของตารางอ้างอิง (reference table) เพื่อให้วินโดว์ได้รับรู้และใช้ในการแสดงผลพาเลตสีให้เหมาะสมที่สุด

ตารางที่ 2.3 แสดงจำนวนดัชนีสี (color index) ในตารางสี (color table) ของ biClrUsed

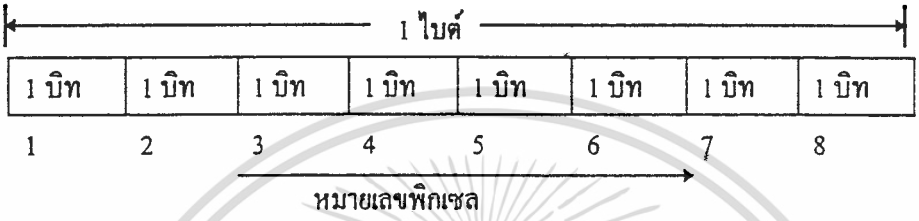
สำหรับส่วน RGBQUAD ที่เก็บรายละเอียดของพาเลตสีประกอบด้วยส่วนประกอบดังนี้

```
STRUCTURE RGBQUAD ประกอบด้วย
BYTE        rgbBlue;
BYTE        rgbGreen;
BYTE        rgbRed;
BYTE        rgbReserved; ไม่ใช้งาน (ปกติตั้งให้เป็นศูนย์)
```

2.2.3. ข้อมูลภาพจริง (actual image information)

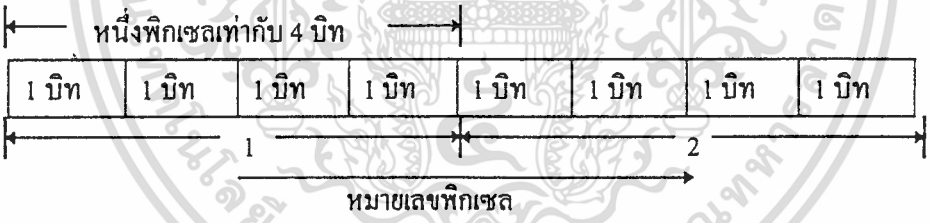
ส่วนของข้อมูลบิทแมพจริงจะมีการเก็บที่แตกต่างกันตามจำนวนบิทข้อมูลภาพ (biBitCount) และลักษณะการบีบอัด biCompression แต่โดยทั่วไปแล้วข้อมูลในรูปแบบบิทแมพปกติที่นิยมในปัจจุบันในการส่งข้ามระหว่างโปรแกรมประยุกต์ต่าง ๆ นั้น จะไม่ใช้การบีบอัดข้อมูล ดังนั้นจึงแบ่งรูปแบบข้อมูลในลักษณะที่นิยมใช้กันในปัจจุบันได้เพียงสามกรณีดังนี้

กรณีที่ 1 ถ้าเป็น 1 บิทต่อพิกเซล หมายความว่ามีดัชนีสี bmiColors เป็นอาร์เรย์ จำนวน 2 อาร์เรย์ คือมี 2 สี โดยข้อมูลแต่ละบิทจะแทนหนึ่งพิกเซล ดังรูปที่ 7



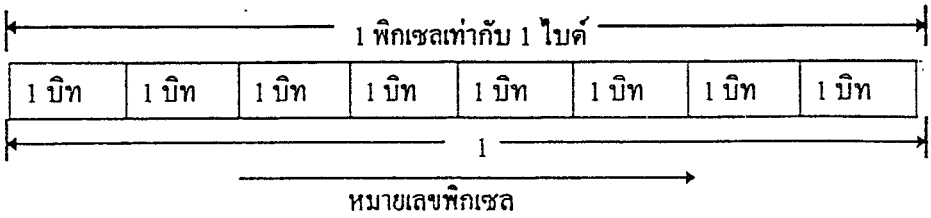
รูปที่ 2.3 แสดงข้อมูลภาพบิทแมพ 2 สี (monochrome Bitmap)

กรณีที่ 2 ถ้าเป็น 4 บิทต่อพิกเซล หมายความว่ามีดัชนีสี bmiColors ที่เป็นอาร์เรย์ จำนวน 16 อาร์เรย์ คือมี 16 สี โดยข้อมูลแต่ละ 4 บิทจะแทนหนึ่งพิกเซล ดังรูปที่ 8



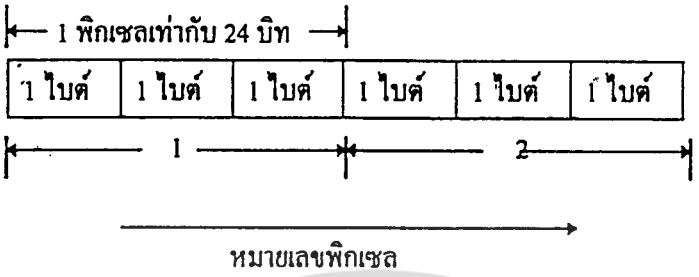
รูปที่ 2.4 แสดงข้อมูลภาพบิทแมพ 16 สี

กรณีที่ 3 ถ้าเป็น 8 บิทต่อพิกเซล หมายความว่ามีดัชนีสี bmiColors ที่เป็นอาร์เรย์ จำนวน 256 อาร์เรย์ คือมี 256 สี โดยข้อมูลแต่ละ 8 บิทจะแทนหนึ่งพิกเซล ดังรูปที่ 2.5



รูปที่ 2.5 แสดงข้อมูลภาพบิทแมพ 256 สี

<u>กรณีที่ 4</u> ถ้าเป็น 24 บิทต่อพิกเซลหมายความว่าจะสามารถแสดงสีได้พร้อมๆกันเท่ากับ 2^{24} คือ 16.7 ล้านสี คือสามารถผสมสีได้โดยตรงไม่ต้องมีการใช้ดัชนีสี ดังนั้นในกรณีนี้ bmiColors = NULL ข้อมูลแต่ละ 3 ไบต์ที่เรียงกันจะแสดงค่าความเข้มสัมพัทธ์ของสีฟ้า สีเขียว และสีแดง (BGR) ตามลำดับ โดยข้อมูลแต่ละ 24 บิทจะแทนหนึ่งพิกเซล ดังรูปที่ 10



รูปที่ 2.6 แสดงข้อมูลภาพบิทแมพ 16.7 ล้านสี

2.3 ทฤษฎีสี

มนุษย์สามารถมองเห็นสีได้เมื่อความยาวคลื่นแสงที่สามารถมองเห็นได้บางช่วงตกกระทบที่เรตินาของดวงตาบางคนสามารถเห็นสีเดียวกันได้จากการรวมกันของความยาวคลื่นที่แตกต่างกันจำนวนมาก จำนวนของการรวมกันของความยาวคลื่นต่างๆไม่มีขีดจำกัดแต่จำนวนของสีที่สามารถแยกแยะได้โดยมนุษย์มีขีดจำกัด ดังนั้น "รูปแบบของสี" จึงเป็นที่ต้องการเพื่อที่จะแทนรหัสที่เป็นเอกลักษณ์สำหรับแต่ละสีที่มนุษย์สามารถเห็นได้ รูปแบบของสีที่มีอยู่มีเป็นจำนวนมาก แต่มีเพียงแค่ 3 รูปแบบที่ถูกใช้ในทางภาพสีซึ่งได้แก่

- รูปแบบสีของ RGB (Red-Green-Blue) ซึ่งถูกใช้สำหรับระบบการแผ่ของแสงสว่างมากกว่ารูปแบบอื่น ๆ เช่น โทรทัศน์และจอคอมพิวเตอร์
- รูปแบบสีของ CMY (Cyan-Magenta-Yellow) ใช้สำหรับระบบการดูดกลืนแสง เช่น การพิมพ์
- รูปแบบสีของ RYB (Red-Yellow-Blue) สำหรับการผสมสีในทางศิลปะ

2.3.1รูปจำลองของสี (color model)

รูปจำลองสีมีลักษณะเป็นรูประบบพิกัดแบบสามมิติ แต่ละสีเกิดจากการผสมระหว่างแกนสีทั้งสาม โดยรูปแบบจำลองของสีในปัจจุบันได้หันเหไปตามฮาร์ดแวร์ เช่น จอภาพสี เครื่องพิมพ์ และการประยุกต์ใช้งานต่าง ๆ รูปแบบจำลองของสีที่นิยมใช้กันมากที่สุดสำหรับมอนิเตอร์สีและกล้องวิดีโอ คือ RGB model (RED, GREEN, BLUE) สำหรับรูปจำลองของสีแบบ CMY model (CYAN, MAGENTA, YELLOW) นิยมใช้กับเครื่องพิมพ์สี ส่วนรูปแบบการส่งแบบ YIQ model เป็นมาตรฐานสำหรับการส่งกระจายภาพโทรทัศน์สี โดย Y คือ ค่าการเปลี่ยนแปลงความเข้มของแสง และ I กับ Q คือ ส่วนของ ค่าการเปลี่ยนแปลงความเข้มของสี ที่เป็น inphase

และ quadrature ตามลำดับ ส่วนรูปจำลองกลางที่ใช้ในการเปลี่ยนไปเปลี่ยนมาระหว่างรูปจำลองสีแบบต่าง ๆ คือ ระบบรูปจำลองแบบ HSI (Hue, Saturation, Intensity) และรูปจำลองแบบ HSV (Hue, Saturation, Value)

2.3.2 รูปแบบจำลองสีแบบอาร์จีบี (RGB color model)

ลำของแสงเมื่อผ่านแก้วปริซึมแล้ว ลำของแสงที่ออกมาไม่ได้มีสีขาว แต่ประกอบไปด้วยสเปกตรัมที่ต่อเนื่องของสี มีย่านจากสีม่วง (violet) ไปจนถึงแดง (red) โดยพื้นฐานแล้วสีที่มนุษย์สามารถมองเห็นวัตถุเกิดมาจากธรรมชาติของแสงในการสะท้อนมาจากวัตถุ วัตถุที่สะท้อนแสงทั้งหมดทุกแสงจะมองเห็นเป็นสีขาว วัตถุที่สะท้อนบางส่วนของสเปกตรัมจะให้บางเฉดของสี ตัวอย่างเช่น วัตถุสีเขียวสะท้อนช่วงที่มีความยาวคลื่น 500 ถึง 570 nm โดยดูดซับเอาความยาวคลื่นที่เหลือนอกนั้นไว้ทั้งหมด ทำให้เราสามารถมองเห็นวัตถุชิ้นนั้นเป็นสีเขียวนั่นเอง ในย่านที่มนุษย์สามารถมองเห็นได้นั้นความยาวคลื่นของแสงจะเป็นตัวกำหนดสี ที่อยู่ในหน่วยนาโนเมตร ในระบบรูปแบบจำลองแบบอาร์จีบี สีแดง สีเขียว และสีฟ้าจะถูกเรียกว่าสีปฐมภูมิ (primary color) แสงที่มีความยาวคลื่นประมาณ 430 นาโนเมตรคือสีฟ้า ความยาวคลื่นประมาณ 550 นาโนเมตรคือสีเขียว และที่มีความยาวคลื่นประมาณ 700 นาโนเมตรคือสีแดง ในการผสมของแสงสีแดง เขียว และฟ้า เข้าด้วยกันในหลาย ๆ รูปแบบ ทำให้สามารถได้สีต่าง ๆ อื่น ๆ ออกมามากมาย ถึงแม้จะมีสีอีกมากที่ไม่สามารถสร้างโดยวิธีการนี้ได้ก็ตาม แต่วิธีการนี้ก็มีประสิทธิภาพดีพอเพียงที่จะใช้ในการแสดงผลภาพ และมีความนิยมใช้กันอย่างมาก เช่น การแสดงผลในมอนิเตอร์สีสำหรับคอมพิวเตอร์ทั่วไป เป็นต้น

2.3.3 แซมเปิลของสีต่อ pixel

การแสดงสี โดยทั่วไปใช้หนึ่งแซมเปิลต่อพิกเซลสำหรับภาพขาวดำและสามแซมเปิลต่อพิกเซลสำหรับค่าสี RGB ค่าสีแสดงเปอร์เซ็นต์ของความเข้มแสงและสี โดยแซมเปิลหนึ่งแสดงเปอร์เซ็นต์ความเข้มของแสงและอีกสองแซมเปิลแสดงเปอร์เซ็นต์ความเข้มของสี

1. ตาของมนุษย์ไม่ไวต่อจำนวนการเปลี่ยนแปลงจริง ๆ ระหว่างความเข้มของแสงสองความเข้ม แต่จะไวต่ออัตราส่วนของการเปลี่ยนแปลงเท่านั้น เช่น การเปลี่ยนแปลงของความเข้มของแสงจาก 0.1 เป็น 0.2 และจาก 0.2 เป็น 0.4 นั้นมนุษย์จะเห็นจำนวนของความเข้มที่เปลี่ยนแปลงเท่ากันเพราะอัตราส่วนเท่ากับ 2 เท่ากัน เมื่อกลุ่มของความเข้มแสงไม่เปลี่ยนแปลงด้วยจำนวนที่คงที่ดังนั้นมันจึงถูกเรียกว่าไม่เป็นเชิงเส้น ซึ่งระบบที่เป็นเส้นทั้งหมดไม่เป็นเชิงเส้นรวมทั้งตาของมนุษย์ด้วย ดังนั้นในระบบสีจึงแทนค่าด้วยค่าการเปลี่ยนแปลงความเข้มของแสง

2. การที่เจเป็กเลือกใช้ระบบสี YCbCr เนื่องจากระบบสี YCbCr นั้นมีการแยกสีออกเป็นค่าการเปลี่ยนแปลงความเข้มของแสง (luminance) และค่าการเปลี่ยนแปลงความเข้มของสี (chrominance) อย่างชัดเจน และในระบบการมองเห็นของตามนุษย์นั้น ตามนุษย์จะมีความไวต่อการเปลี่ยนแปลงความเข้มของแสงมากกว่าการเปลี่ยนแปลงความเข้มของสี ดังนั้นระบบสี YCbCr จึงมีประโยชน์ต่อการลดจำนวนข้อมูลสีต่อจำนวนจุดพิกเซลโดยการลดจำนวนสี ค่าการเปลี่ยนแปลงความเข้มของสี ด้วยวิธีซับแซมปลิง ซึ่งจะไม่ทำให้คุณภาพของภาพเปลี่ยนไปมากนักเพราะตามนุษย์สังเกตไม่เห็น

เนื่องจากบิทแมพไฟล์ ซึ่งเป็นรูปแบบข้อมูลภาพที่เราจะนำมาทำการบีบอัด ใช้ระบบสี RGB แต่ในระบบของเจเป็กจะใช้รูปแบบสีของระบบสี YCbCr ดังนั้นจึงต้องมีการแปลง ระบบสีจากระบบสี RGB มาเป็นระบบสี YCbCr ซึ่งการแปลงระบบสีสามารถทำได้ 2 วิธีดังนี้

1) การแปลงระบบสีโดยตารางสี

สิ่งที่ใช้กันมากอย่างหนึ่งในเรื่องของทฤษฎีสีก็คือตารางสีซึ่งเป็นการแทนค่าสีของแต่ละพิกเซลด้วยค่าตัวเลขค่าหนึ่งในตารางสี (เรียกว่าค่าดัชนีสี) ซึ่งตารางสีสำหรับภาพสีและภาพขาวดำก็จะแตกต่างกัน

ภาพขาวดำใช้ตารางสีเพียงตารางเดียว (เนื่องจากมีค่าความเข้มค่าเดียวต่อ 1 พิกเซล) สำหรับภาพสีบ่อยครั้งที่จะใช้ตารางสีแยกกันสำหรับแต่ละส่วนของความเข้มต่อพิกเซล เช่น ตารางหนึ่งสำหรับสีแดง, ตารางหนึ่งสำหรับสีเขียว และอีกตารางหนึ่งสำหรับสีน้ำเงิน)

สาเหตุที่เราใช้ตารางสีก็เพื่อลดขนาดของหน่วยความจำที่ต้องการสำหรับภาพเนื่องจากจำนวนบิทที่เราใช้แทนแต่ละสีทำให้ต้องการหน่วยความจำสำหรับแทนแต่ละสี ซึ่งจำนวนไบต์ของหน่วยความจำที่ต้องการสำหรับเก็บภาพสามารถคำนวณได้โดย

$$\frac{\text{ความกว้างของภาพ} \times \text{ความสูงของภาพ} \times \text{จำนวนบิท/ความเข้ม} \times \text{จำนวนความเข้ม/พิกเซล}}{\text{8 บิทต่อไบต์}}$$

ตารางข้างล่างจะแสดงให้เห็นว่าหน่วยความจำที่ต้องการสำหรับภาพสีภาพหนึ่งค่อนข้างใหญ่ทีเดียว

STORAGE NEEDED FOR AN IMAGE AS A FUNCTION OF COLOR PRECISION

# of bits per intensity	Intensities per pixel	# of bytes needed for a 512 × 512 image	Comments
1	1	32,768 = 32 KB	Black and white
4	1	131,072 = 128 KB	
8	1	262,144 = 256 KB	Grayscale
12	1	393,216 = 384 KB	
8	3	786,432 = 768 KB	Full color
12	3	1,048,576 = 1152 KB	

ตารางที่ 2.4 แสดงหน่วยความจำที่ต้องการสำหรับภาพสีภาพหนึ่งๆ

ทฤษฎีสีในอุดมคติจะลดจำนวนบิทที่ต้องการสำหรับแสดงแต่ละสีของพิกเซลซึ่งจะต้องการใช้ได้กับช่วงของสีที่กว้าง ตารางสีก็จัดเป็นทฤษฎีนี้ด้วยเช่นกัน ซึ่งสีของแต่ละพิกเซลถูกแสดงโดยค่าตัวเลขค่าหนึ่งในตารางนี้ โดยใช้ n บิทเพื่อที่จะแทนค่าดัชนีของสีซึ่งทำให้ขนาดของตารางสีเท่ากับ 2^n และให้จำนวนบิทที่แทนความเข้มของแต่ละสีเท่ากับ m บิท จำนวนของจำนวนสีจริง ๆ จะเท่ากับ 2^m ซึ่งแสดงดังตาราง

# of bits per index	Table size	# of bits per color	Possible colors	Comments
2	4	5	32	4 colors out of a possible 32
4	16	12	4,096	16 colors out of a possible 4,000
8	256	24	16,777,216	256 colors out of a possible 16,000,000

ตารางที่ 2.5 ตารางสี

2) การแปลงระบบสีโดยตรง (true color)

เมื่อค่าของสีค่าหนึ่งถูกใช้เป็นความเข้มหนึ่งของสีโดยตรง ซึ่งบ่อยครั้งที่จะหมายถึงสีจริง (true color) ซึ่งชื่อนี้อาจทำให้เข้าใจผิดเล็กน้อยเพราะว่ามันไม่ได้หมายความว่าค่าของสีนั้นเป็นค่าจริง แต่หมายถึงว่าค่าของสีนั้นแทนค่าของสีเลย โดยไม่ต้องใช้ตารางสี

2.3.4 การแปลงระบบสี RGB เป็นระบบสี YCbCr

เนื่องจากระบบสีในบิทแมพไฟล์ นั้นจะเป็นระบบ RGB แต่ระบบสีที่ใช้ใน JPEG นั้นจะเป็นระบบสี YCbCr ซึ่งจะประกอบด้วยการเปลี่ยนแปลงความเข้มของแสง (luminance) ซึ่งแทนด้วย Y และการเปลี่ยนแปลงความเข้มของสี (chrominance) ซึ่งแทนด้วย Cb และ Cr โดยมีสมการการแปลงสีในระบบสี RGB เป็นระบบสี YCbCr ดังนี้

$$Y = 0.299\,R + 0.587\,G + 0.114\,B$$
$$Cb = -0.1687\,R - 0.3313\,G + 0.5\,B$$
$$Cr = 0.5\,R - 0.4187\,G - 0.0813\,B$$

โดย ค่า Y อยู่ ในช่วง 0 ถึง 255
ค่า Cb อยู่ในช่วง 0 ถึง 255
ค่า Cr อยู่ในช่วง 0 ถึง 255

โดยการแปลงสีจากระบบสี RGB เป็นระบบสี YCbCr ของระบบสีในบิทแมพไฟล์ มี 2 แบบคือ

1) การแปลงสีจากระบบ RGB ใน บิทแมพไฟล์ เป็นระบบสี YCbCr โดยใช้ตารางสี

เนื่องจากบิทแมพไฟล์ ชนิด 1, 4 และ 8 บิท/พิกเซล สีที่ใช้ทั้งหมดจะเก็บไว้ในตารางสี ซึ่งจะมีจำนวนสีเท่ากับ 2, 16, 256 สีตามลำดับ ซึ่งจุด pixel แต่ละจุดในบิทแมพไฟล์ ชนิดนี้จะเก็บค่าดัชนีของสีที่ใช้ในตารางสี ดังนั้นการแปลงสีจากระบบสี RGB ไปเป็นระบบสี YCbCr จะทำโดยการแปลงสีแต่ละสีตามลำดับใน

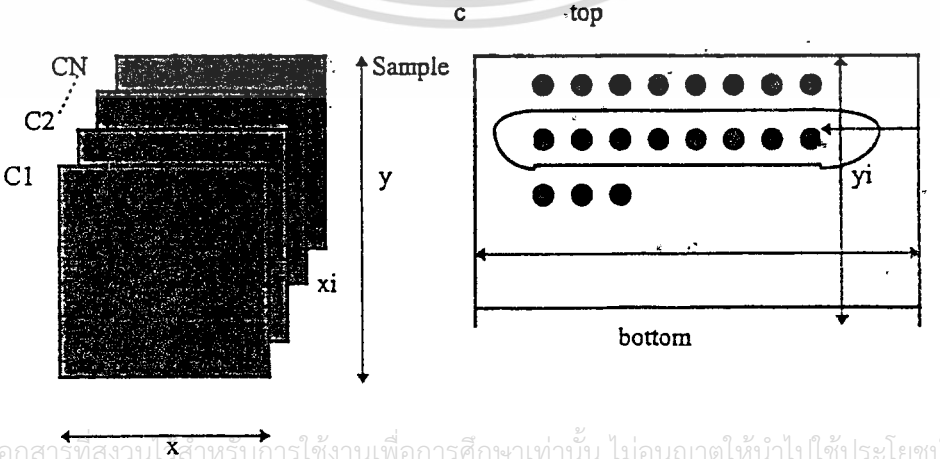
ตารางสีซึ่งประกอบด้วยค่าสี R, G, B ไปเป็นค่าสี Y, Cb, Cr และเก็บไว้ในตารางสีตารางใหม่ที่สร้างขึ้นอีกตารางหนึ่ง ด้วยวิธีนี้ค่าดัชนีที่เก็บไว้ในจุดพิกเซล สามารถใช้เลือกสีในระบบสี YCbCr จากตารางสีตารางใหม่ได้ทันที

2) การแปลงสีจากระบบ RGB ในบิทแมพไฟล์ เป็นระบบสี YCbCr โดยการแปลงโดยตรงซึ่งเรียกว่าสีจริง

เนื่องจากบิทแมพไฟล์ ชนิด 24 บิท/พิกเซล จะเก็บค่าสี R, G, B ไว้ในจุดพิกเซล โดยตรง ดังนั้นการแปลงระบบสีในบิทแมพไฟล์ ชนิดนี้จึงต้องทำการแปลงสีของจุด พิกเซล ทีละจุดโดยนำค่าสี R, G, B จากจุด พิกเซลมาทำการแปลงสีเป็นค่าสี Y, Cb, Cr และเก็บค่าสีที่ได้ไว้ตามตำแหน่งของจุดพิกเซล นั้น ๆ

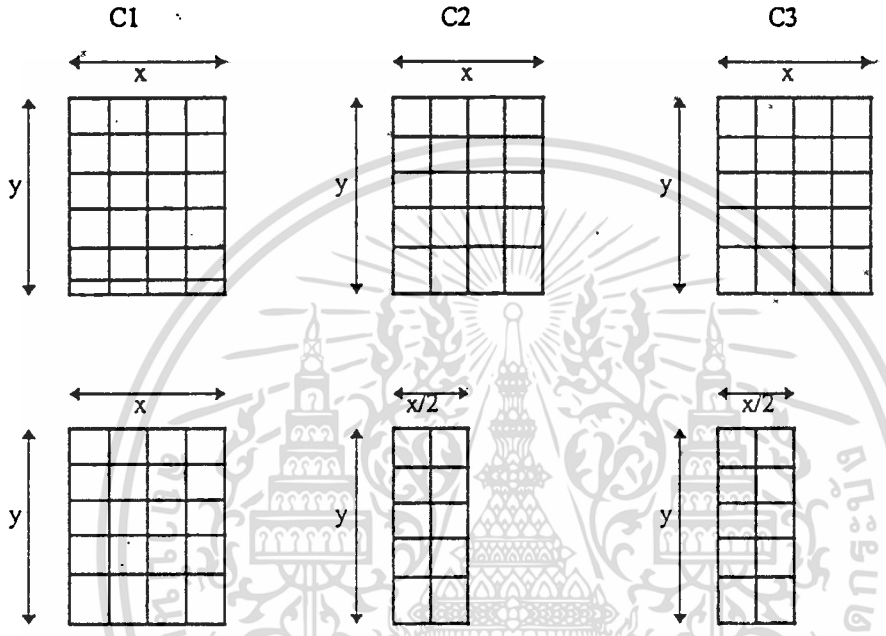
2.4 การซับแซมปลิง (Subsampling)

การซับแซมปลิง เป็นการลดจำนวนข้อมูลด้วยการแทนกลุ่มของข้อมูลด้วยค่าเฉลี่ยของกลุ่มข้อมูลนั้น วิธีของการซับแซมปลิง ข้อมูลสีสำหรับข้อมูลภาพดิจิตอลมีหลายวิธี วิธีหนึ่งถูกแสดงโดยชุดของค่า 3 ค่าซึ่งแยกโดยเครื่องหมายโคลอน (:) ซึ่งค่าเหล่านี้จะแสดงจำนวนความสัมพันธ์ของตัวอย่างจากแต่ละส่วนประกอบของภาพ ค่าแรกแทน Y ซึ่งคือ การเปลี่ยนแปลงของความเข้มแสงค่าที่สองแสดง Cr (R-Y) และค่าที่สามแสดง Cb (B-Y) ซึ่งค่าที่สองและสามคือ การเปลี่ยนแปลงของความเข้มสีสองรูปแบบที่ใช้กันโดยทั่วไปใช้อัตราส่วนการซับแซมปลิง เป็น 4:2:2 และ 4:1:1 รูปแบบ 4:2:2 แสดงว่าสำหรับทุก ๆ สี่ ตัวอย่างของ ค่าการเปลี่ยนแปลงความเข้มของแสง จะมีสองตัวอย่างของแต่ละแบบของ ค่าการเปลี่ยนแปลงความเข้มของสี ซึ่งอาจกล่าวได้ว่าทุก ๆ สองตัวตัวอย่างของ ค่าการเปลี่ยนแปลงความเข้มของแสง มีหนึ่งตัวอย่างของแต่ละ ค่าการเปลี่ยนแปลงความเข้มของสี ส่วนรูปแบบ 4:1:1 แสดงว่าสำหรับทุก ๆ สี่ตัวอย่างของ ค่าการเปลี่ยนแปลงความเข้มของแสงจะมีหนึ่งตัวอย่างของแต่ละ ค่าการเปลี่ยนแปลงความเข้มของสี ซึ่งในที่นี้เราใช้การซับแซมปลิง แบบ 4:1:1 ซึ่งการซับแซมปลิง สามารถทำได้ใน 2 ทิศทาง คือ ในแนวตั้ง (Vertical:V) และในแนวนอน (Horizontal:H) จากอัลกอริทึมของเจเป็กสำหรับภาพเกรย์สเกล ถ้าเรานำมาบีบอัดข้อมูลภาพสี อาจกล่าวได้ว่าจะต้องเป็นการบีบอัดข้อมูลแบบหลาย ๆ คอมโพเนนท์ของภาพ จากโมเดล (Model) ของภาพต้นฉบับของเจเป็กจะประกอบไปด้วย 1 ถึง 255 คอมโพเนนท์ของภาพที่เรียกกันว่า สี หรือ แบนด์ ของสเปกตรัม (Spectrum Band) ดังรูปที่ 2.7



รูปที่ 2.7 แสดงคอมโพเนนท์ของภาพสี

จากพื้นฐานของสีที่กล่าวว่า สีต่าง ๆ ประกอบขึ้นจากสีหลัก คือ สีแดง สีเขียว และสีน้ำเงิน ที่อัตราส่วนผสมต่าง ๆ กัน เราอาจพิจารณาได้ ตัวอย่างเช่น การแสดงผลแบบ RGB และ YCrCb ก็จะประกอบไปด้วย 3 คอมโพเนนท์ ซึ่งแต่ละคอมโพเนนท์อาจจะมีจำนวนของพิกเซลในแนวตั้ง (X_i) และแนวนอน (Y_i) ที่ไม่เท่ากันก็เป็นได้ ดังรูปที่ 2.8 แสดงให้เห็นถึง 2 กรณี สำหรับภาพสีที่มี 3 คอมโพเนนท์กรณีแรกทุก ๆ คอมโพเนนท์มีความละเอียดในการแสดงผลเท่ากัน แต่ในกรณีที่สอง แต่ละคอมโพเนนท์มีการแสดงผลที่ความละเอียดไม่เท่ากัน

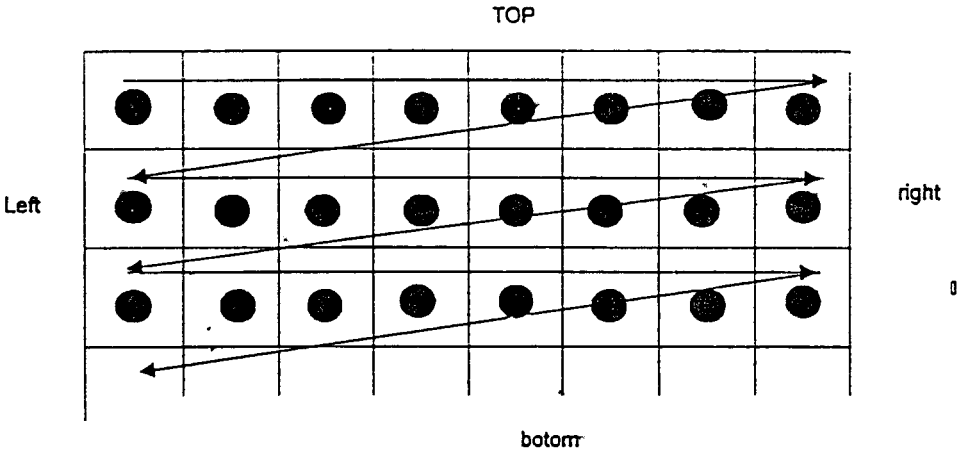


รูปที่ 2.8 แสดงความละเอียดของแต่ละคอมโพเนนท์ในการประมวลผลข้อมูล

การประมวลผลคอมโพเนนท์ต่าง ๆ

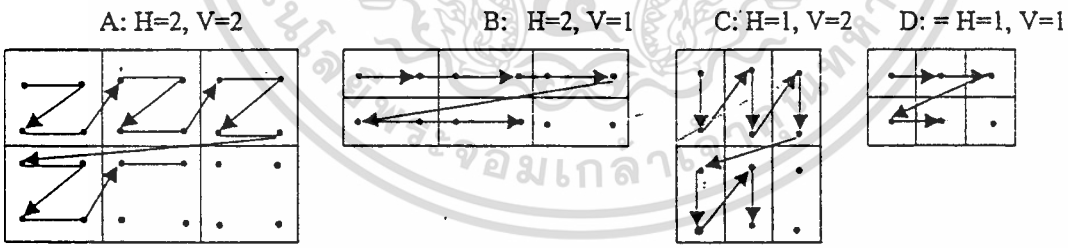
คอมโพเนนท์ต่าง ๆ ของสีสามารถถูกประมวลผลได้ 2 ลักษณะ ด้วยกัน คือ

1. แบบลำดับการประมวลผลข้อมูลของแต่ละคอมโพเนนท์ตามลำดับ คือ จะทำการบีบอัดข้อมูลทีละคอมโพเนนท์ จากซ้ายไปขวา และจากบนลงล่าง สมมติเป็นโหมด RGB เราก็จะทำการบีบอัด R (สีแดง) ก่อน G (สีเขียว) และ B (สีน้ำเงิน) ตามกันไปเป็นลำดับ แสดงตามรูปที่ 5.15 ซึ่งในการแสดงผลการขยายข้อมูลกลับคืนนั้นจะเห็นหน้าจอเป็น สีแดงจนเต็มจอก่อน แล้วค่อยแสดงสีเขียว ซ้อนจนเต็มจอ และสีน้ำเงิน ซ้อนทับเป็นลำดับสุดท้าย เป็นต้น



รูปที่ 2.9 แสดงลักษณะการประมวลผลข้อมูลแต่ละคอมโพเนนท์แบบไม่ซ้อนกันทับกัน

2. แบบประมวลผลแต่ละคอมโพเนนท์ไปพร้อม ๆ กัน โดยที่คอมโพเนนท์ต่าง ๆ เหล่านั้นจะถูกนำมารวมกัน เรียกว่า หน่วยที่ถูกเข้ารหัสแบบน้อยที่สุด (Minimum Coded Units หรือ MCUs) วิธีนี้เหมาะสำหรับงานที่จะเป็นต้องแสดงผลหรือพิมพ์ พร้อม ๆ ไปกับกระบวนการขยายข้อมูลคืนในทันที เนื่องจากคอมโพเนนท์ต่าง ๆ จะต้องซ้อนทับเข้าด้วยกัน ดังนั้นจึงจำเป็นจะต้องทำการแบ่งแต่ละคอมโพเนนท์แบบสี่เหลี่ยม ตาม H_i และ V_i ตัวอย่างแสดงตามรูปที่ 2.10ซึ่งภาพประกอบด้วย 4 คอมโพเนนท์ คือ $C_1 C_2 C_3 C_4$ ซึ่งแต่ละคอมโพเนนท์ก็มีความละเอียด H_i , V_i ที่ไม่เท่ากัน

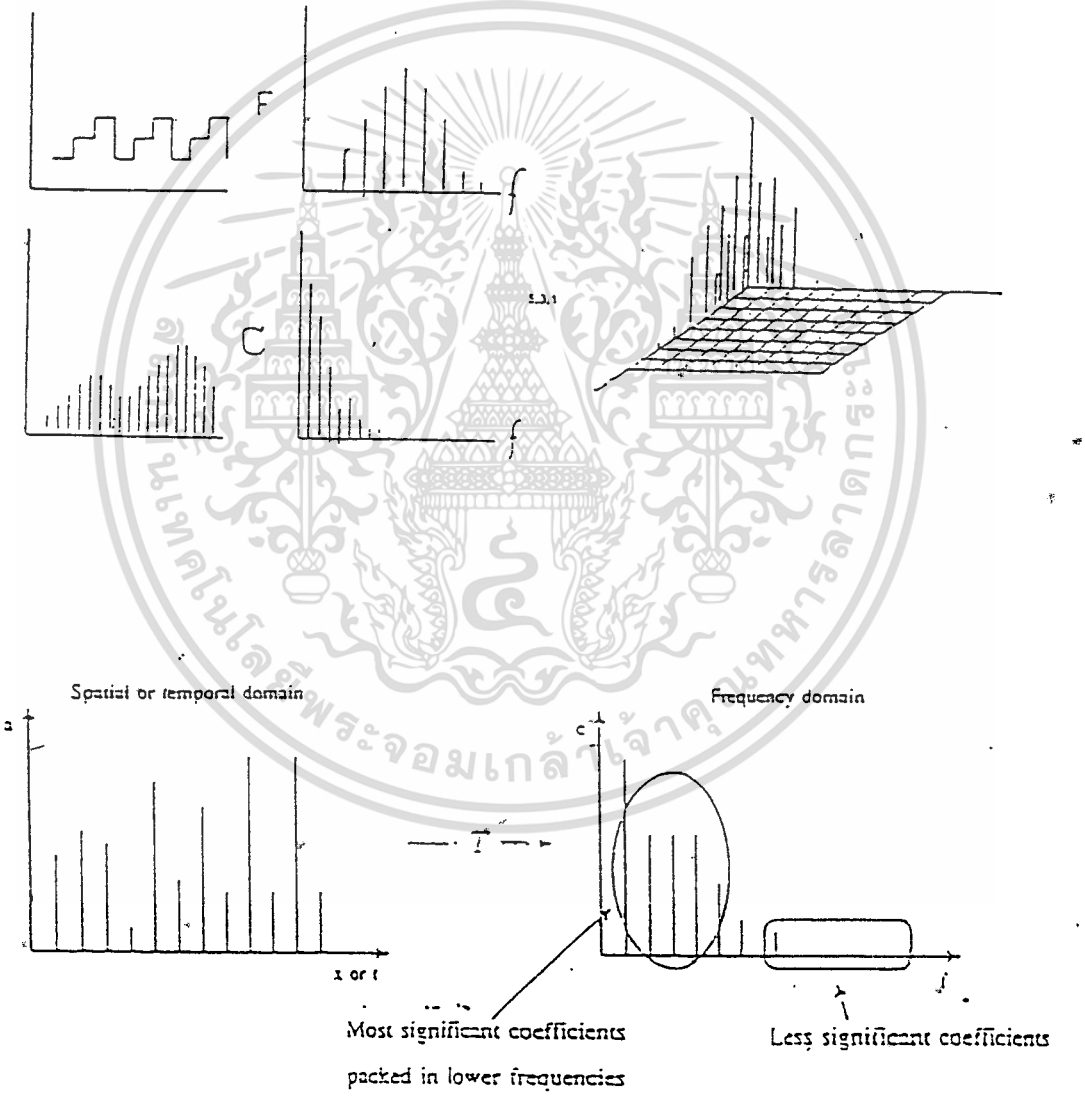


Data unit	A	B	C	D
MCU1 =	(A00,A01,A10,A11,	B00,B01	C00,C10	D00)
MCU2 =	(A02,A03,A12,A13,	B02,B01	C01,C11	D01)
MCU3 =	(A04,A05,A14,A15,	B04,B01	C02,C12	D02)
MCU4 =	(A20,A21,A30,A31,	B10,B01	C20,C30	D10)

รูปที่ 2.10 แสดงทิศทางการประมวลผลข้อมูลแต่ละคอมโพเนนท์ แบบที่ความละเอียดของแต่ละคอมโพเนนท์ไม่เท่ากัน

2.5 การแปลงแบบโคซายน์

การแปลงรูปสัญญาณก็คือ การนำข้อมูลเริ่มต้นซึ่งเป็นข้อมูลที่อยู่ในสปาเชี่ยลโดเมน (Spatial-doiam) มาผ่านกระบวนการทางคณิตศาสตร์ เพื่อแปลงให้อยู่ในรูปของโดเมนอื่น ๆ ที่เหมาะสมสำหรับวิธีการบีบอัดนั้น ๆ ซึ่งโดยส่วนใหญ่แล้วการแปลงจะอยู่ในโดเมนของความถี่ กระบวนการแปลงแต่ละแบบสามารถทำกลับให้อยู่ในรูปแบบเดิมได้โดยการแปลงข้อมูลกลับ (Inverse transform) ดังตัวอย่างในรูปที่ 2.11 การแปลงจากโดเมนสปาเชียลไปสู่โดเมนของความถี่ โดยหลังจากการแปลงแล้วจะได้สัมประสิทธิ์ที่มีนัยสำคัญสูง (Most significant coefficients) และนัยสำคัญต่ำ (Less significant coefficients)



รูปที่2.11 แสดงตัวอย่างการเข้ารหัสการแปลงจากโดเมนหนึ่งไปยังอีกโดเมนหนึ่ง
(a เป็นค่าขนาดของสัญญาณ และ c เป็นค่าขนาดของกลุ่มพลังงาน)

รูปแบบของการแปลงมีด้วยกันหลายแบบ เช่นการแปลงแบบฟูริเยร์ (Fourier transform) การแปลงแบบโคซายน์ (Cosine transform) การแปลงแบบฮาร์ (Haar transform) การแปลงแบบฮาร์คามาร์ด (Hadamard transform) หรือการแปลงแบบคาร์ฮุเนน (Karhunen transform) เป็นต้น และรูปแบบที่นิยมมากในการนำมาใช้สำหรับการบีบอัดข้อมูลภาพก็คือการแปลงแบบโคซายน์

การแปลงแบบโคซายน์เป็นการแปลงข้อมูลจากโดเมนของภาพ (Spatial domain)ไปเป็นโดเมนทางความถี่ (Frequency domain) โดยมีสมการการแปลงดังนี้

$$F(u) = \int f(x)\cos(2\pi fx)dx \qquad \text{———————} (2.5)$$

ซึ่งจะเห็นได้ว่าการแปลงแบบโคซายน์คือการแปลงแบบฟูเรียร์โดยที่ฟังก์ชัน ที่จะมาทำการแปลงนั้นต้องเป็นฟังก์ชันคู่ซึ่งจะทำให้ผลการแปลงมีเฉพาะสัมประสิทธิ์ของฟังก์ชันโคซายน์

2.6 การแปลงแบบดิสครีตโคซายน์

การแปลงโคซายน์นั้น เป็นวิธีที่นิยมมากในเรื่องของการบีบอัดของข้อมูลที่ต้องการความรวดเร็วและมีประสิทธิภาพ และวิธีการนี้ก็เป็นกระบวนการของการบีบอัดข้อมูลที่มีการสูญเสีย (Lossy compression) โดยวิธีที่ใช้เป็นมาตรฐานอยู่ขณะนี้ก็คือ JPEG (Joint Photographics Expert Group) โดยมีพื้นฐานของการแปลงดีสครีตโคไซน์เป็นหลัก

การแปลงโคซายน์แบบไม่ต่อเนื่อง เป็นการประมวลผลทางคณิตศาสตร์ที่เป็นการคำนวณส่วนประกอบทางความถี่ของแซมเปิลของสัญญาณที่อัตราการแซมปลิงใดๆ ซึ่งการแปลงโคซายน์แบบไม่ต่อเนื่องนี้จะต้องประยุกต์ใช้กับจำนวนของแซมเปิลที่รู้ค่าแน่นอน ในทางคณิตศาสตร์สามารถพิจารณาได้ง่ายๆคือ ถ้าจำนวนที่ใช้เป็นกำลังคู่ของ 2 การแปลงโคซายน์แบบไม่ต่อเนื่องแบบทิศทางเดียวใช้ในการเปลี่ยนอาร์เรย์หนึ่งของจำนวนที่แสดงค่าแอมปลิจูดของสัญญาณที่จุดที่ทราบค่าใดๆบนโดเมนของเวลาไปเป็นอีกอาร์เรย์หนึ่ง ซึ่งใช้แสดงค่าแอมปลิจูดที่เป็นคอมโพเนนท์ทางความถี่ของสัญญาณที่เป็นอินพุทอาร์เรย์ผลลัพธ์จะมีอีเลเมนท์แรกเป็นค่าเฉลี่ยของทุกๆแซมเปิลของอาร์เรย์อินพุทเรียกว่าเป็นสัมประสิทธิ์กระแสตรง(DC Coefficient)และอีเลเมนท์ที่เหลือก็เริ่มมีความถี่เข้ามาเกี่ยวข้องจึงเรียกว่าสัมประสิทธิ์กระแสสลับ (AC Coefficient) ที่เป็นค่าเฉพาะของแต่ละอีเลเมนท์ (ค่าสัมประสิทธิ์ คือ แอมปลิจูดของคอมโพเนนท์ทางความถี่)

สมการที่ใช้ในการแปลงโคซายน์แบบไม่ต่อเนื่อง 1 ทิศทางนั้นเป็นดังสมการที่ 2.6

$$\text{Coeff}(k,m) = C(k)\cos[(2m+1)k\pi/(2N)] \qquad \text{.....(2.6)}$$

เมื่อ $C(k) = 1/\sqrt{2}$ เมื่อ $k = 0$

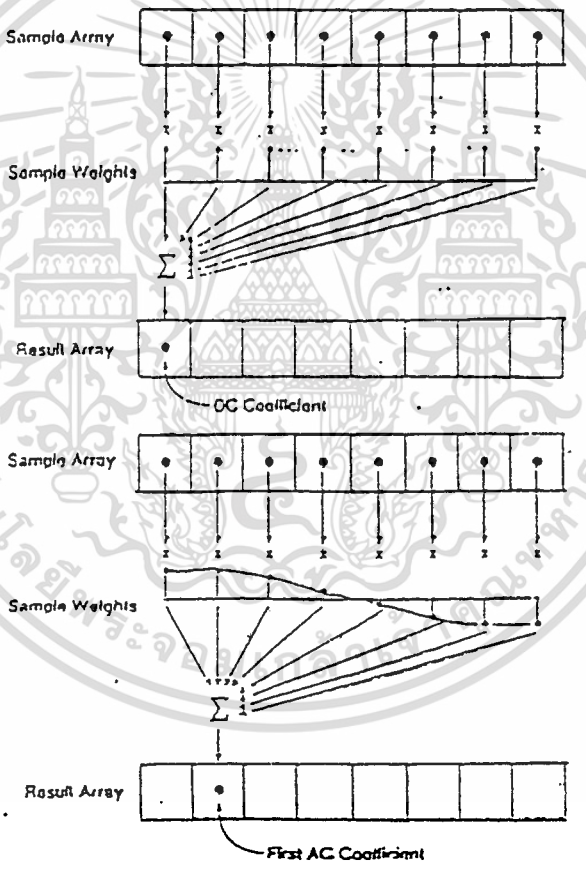
$= 1$ เมื่อ $k \neq 0$

(k = ดัชนีของอาร์เรย์ผลลัพธ์)

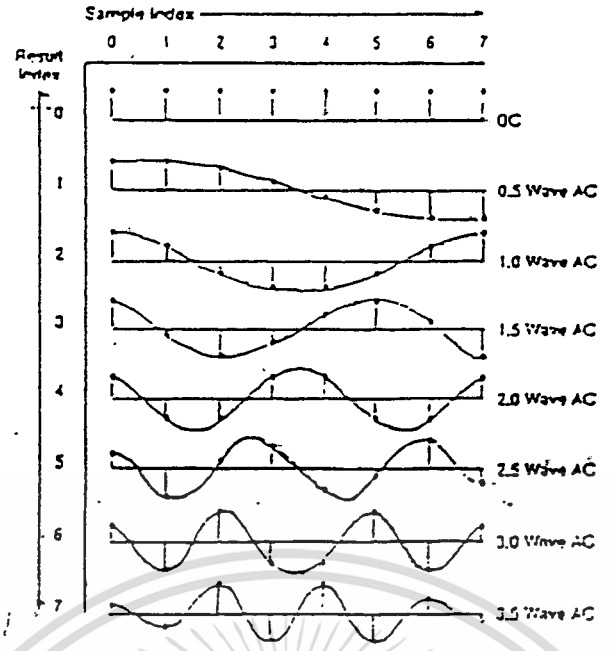
(m = ดัชนีของแซมเปิลอาร์เรย์)

(n = ขนาดของแซมเปิลอาร์เรย์)

ความถี่ที่ได้จากการแปลงแสดงโดยสัมประสิทธิ์ในอาร์เรย์ผลลัพธ์จะเป็นฟังก์ชั่นของค่าดัชนีอาร์เรย์ (Array Index) สำหรับอีเลเมนท์นั้น ส่วนประกอบของความถี่ของเซตของแซมเปิลที่แต่ละความถี่จะถูกคำนวณโดยการถ่วงน้ำหนักค่าเฉลี่ย (Weighted Average) ให้กับเซตที่นำมาคิด ค่าในการถ่วงน้ำหนักสำหรับแต่ละแซมเปิลอินพุท หาโดยการคูณดัชนีปัจจุบันของอาร์เรย์ผลลัพธ์กับค่าคงที่ π และดัชนีของแซมเปิลอินพุท วิธีนี้จะมีผลต่อการสร้างลำดับของสัมประสิทธิ์การถ่วงน้ำหนักโดยที่การประมาณค่าของคลื่นโคซายน์ที่ความถี่เป็นสัดส่วนกับดัชนีของอาร์เรย์ผลลัพธ์ ในรูป 2.12 แสดงความสัมพันธ์ในการประมาณค่าระหว่างสัมประสิทธิ์การถ่วงน้ำหนักของแต่ละแซมเปิลอินพุทสำหรับแต่ละค่าดัชนีของอาร์เรย์ผลลัพธ์ ในการแปลงโคซายน์แบบไม่ต่อเนื่อง 1 ทิศทางบนอาร์เรย์ของ 8 แซมเปิลอินพุท



รูปที่ 2.12 การหาค่าสัมประสิทธิ์การถ่วงน้ำหนักของแซมเปิล



รูปที่ 2.13 แสดงค่าสัมประสิทธิ์การถ่วงน้ำหนักสัมพัทธ์ของการแปลงโคซายน์แบบไม่ต่อเนื่อง

ค่าถ่วงน้ำหนักสูงสุดสำหรับแซมเปิลใดๆจะมีค่าเป็น 1 ซึ่งจะมีค่าอยู่ในช่วง -1 ถึง 1 พิจารณาจากรูปที่ 2.13 สามารถพิจารณาได้ว่าแต่ละสเตปของดัชนีอาร์เรย์ผลลัพธ์จะขึ้นอยู่กับการเพิ่มขึ้นทีละครึ่งไซเคิลของคอมโพเนนท์ทางความถี่ที่สัมพันธ์กัน สำหรับอาร์เรย์ของ 8 แซมเปิลความถี่สูงสุด 3.5 ไซเคิล ซึ่งมีค่าใกล้กับความถี่สูงสุดที่สามารถแสดงออกมาได้นั่นคือ 4 ไซเคิลหรือครึ่งเท่าของความถี่ในการแซมเปิล ตารางที่ 2.6 แสดงค่าสัมประสิทธิ์ที่ใช้ในการคำนวณค่าเฉลี่ยตัวถ่วงน้ำหนักสำหรับแต่ละคอมโพเนนท์ทางความถี่สมการที่ใช้ในการคำนวณแสดงตามสมการที่ 2.7

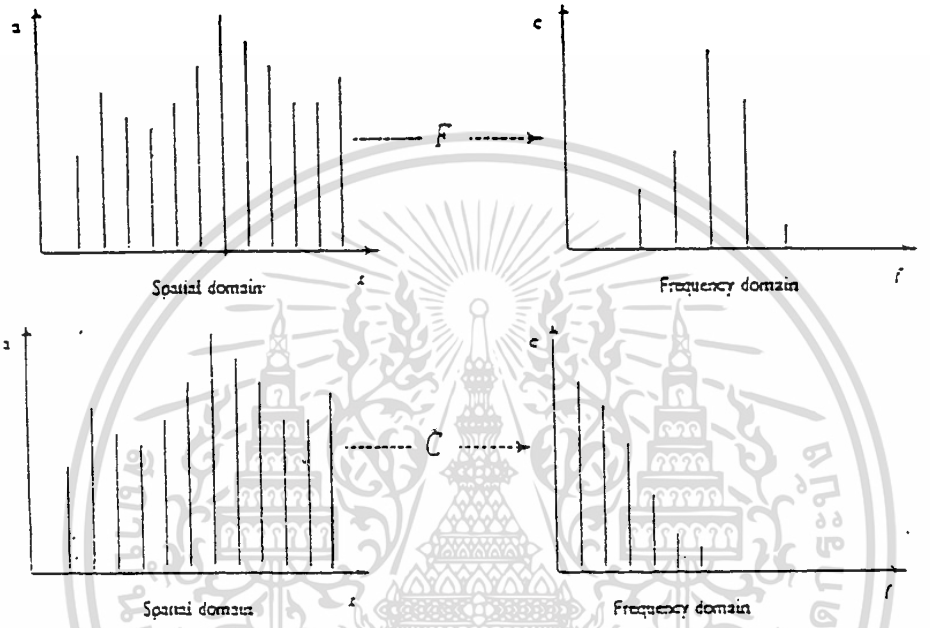
ดัชนีผลลัพธ์	ดัชนีของแซมเปิล							
↓	0	1	2	3	4	5	6	7
0	+0.707	+0.707	+0.707	+0.707	+0.707	+0.707	+0.707	+0.707
1	+0.981	+0.831	+0.556	+0.195	-0.195	-0.556	-0.831	-0.981
2	+0.924	+0.383	-0.383	-0.924	-0.924	-0.383	+0.383	+0.924
3	+0.831	-0.195	-0.981	-0.556	+0.556	+0.981	+0.195	-0.831
4	+0.707	-0.707	-0.707	+0.707	+0.707	-0.707	-0.707	+0.707
5	+0.556	-0.981	+0.195	+0.831	-0.831	-0.195	+0.981	-0.556
6	+0.383	-0.924	+0.924	-0.383	-0.383	+0.924	-0.924	+0.383
7	+0.195	-0.556	+0.831	-0.981	+0.981	-0.831	+0.556	-0.195

ตารางที่ 2.6 แสดงค่าสัมประสิทธิ์การถ่วงน้ำหนักแซมเปิลสำหรับการแปลงโคซายน์แบบไม่ต่อเนื่องแบบ 1 ทิศทาง

พิจารณาจากรูป2.14 เป็นการเปรียบเทียบระหว่างการแปลงโดเมนทางเวลาของสัญญาณใดๆเป็นโดเมนของความถี่โดยใช้การแปลงฟูเรียร์ (Fourier Tranform) และการแปลงโคซายน์แบบไม่ต่อเนื่องจะเห็นได้อย่าง

ชัดเจนว่าทั้งสองวิธีมีการรวบรวมส่วนประกอบที่สำคัญของข้อมูลไว้ได้เหมือนกันแต่ที่ความถี่ต่างกันกล่าวคือ การแปลงโคซายน์แบบไม่ต่อเนื่องจะสามารถเก็บส่วนประกอบส่วนใหญ่ของภาพไว้ที่ความถี่เท่ากับศูนย์ ส่วนการแปลงฟูเรียร์ความถี่ที่รวบรวมข้อมูลส่วนใหญ่ที่สำคัญของภาพจะไม่ได้อยู่ที่ความถี่เท่ากับศูนย์ ซึ่งจากรูปจะเห็นว่าข้อมูลในโดเมนความถี่จะมีลักษณะเด่นอยู่3 ประการที่เห็นได้ชัดคือ

1. ค่าความถี่ที่ศูนย์ จะเป็นค่าของความเข้มเฉลี่ยของข้อมูล
2. ค่าความถี่สูงเป็นค่าที่บอกถึงข้อมูลที่มีการเปลี่ยนแปลงสูง
3. ค่าความถี่ต่ำ เป็นค่าที่บอกรายละเอียดโดยรวมของข้อมูล



รูปที่ 2.14 แสดงการแปลงแบบฟูริเยร์และการแปลงแบบดิสครีตโคซายน์ตามลำดับ

การที่การแปลงโคซายน์แบบไม่ต่อเนื่องสามารถรวบรวมส่วนประกอบส่วนใหญ่ของภาพไว้ที่ความถี่ต่ำๆได้ ซึ่งสิ่งนี้มีความสำคัญมากเพราะภาพส่วนใหญ่นั้นข้อมูลของภาพจะรวมกันอยู่ในช่วงของความถี่ต่ำและส่วนประกอบที่พบในจุดที่แถวและคอลัมน์เป็นศูนย์ (ส่วนประกอบ DC) จะเก็บข้อมูลที่มีประโยชน์เกี่ยวกับภาพมากกว่าส่วนประกอบที่ความถี่ที่สูงกว่านี้ ส่วนประกอบตัวที่อยู่ห่างจากส่วนประกอบ DC นั้นจะพบว่าไม่เพียงแต่ค่าสัมประสิทธิ์มีแนวโน้มที่จะต่ำลงแต่ยังมีความสำคัญต่อการอธิบายภาพน้อยลงอีกด้วย ดังนั้นการแปลงแบบดิสครีตโคซายน์จะทำให้สามารถตัดข้อมูลบางส่วนออกไปได้โดยไม่มีผลต่อคุณภาพของภาพมากนัก ซึ่งถ้าภาพไม่ได้ถูกทำการแปลงแต่ยังคงอยู่ในรูปของจุดพิกเซลจะไม่สามารถตัดข้อมูลบางส่วนของภาพออกไปได้เช่นนี้เพราะแต่ละพิกเซลมีผลต่อการมองเห็นของภาพโดยรวม

2.7 การแปลงโคซายน์แบบไม่ต่อเนื่อง 2 ทิศทาง

เนื่องจากภาพโดยทั่วๆไปประกอบขึ้นจากฟังก์ชัน 2 ฟังก์ชันดังนั้นในการกระทำกระบวนใดๆกับภาพจำเป็นต้องทำในลักษณะ 2 แนวพร้อมๆกัน ในการทำการแปลงรูปสัญญาณก็เช่นเดียวกันจำเป็นต้องทำใน 2 ทิศทางดังนั้นจึงต้องทำการแปลงโคซายน์แบบไม่ต่อเนื่องในลักษณะ 2 ทิศทางด้วย จากการแปลงโคซายน์แบบไม่ต่อเนื่องที่ได้กล่าวถึงไปแล้วนั้นสามารถอธิบายการแปลงโคซายน์แบบไม่ต่อเนื่อง 2 ทิศทางได้โดยนำภาพเริ่มต้นมาแบ่งเป็นบล็อกของพิกเซลขนาด 8x8 พิกเซล ซึ่งเป็นลักษณะของอาร์เรย์ของพิกเซล 2 มิติ โดยในแต่ละบล็อกของ 8x8 นั้นจะมีด้วยกันทั้งหมด 64 ค่า และแต่ละค่าจะแสดงขนาด (Amplitude) ของแต่ละค่าที่เข้ามา ขนาดของสัญญาณนี้จะเป็นฟังก์ชันของจุดพิกัด 2 จุด โดยสมมติให้ a = f(x,y) โดย x,yจะเป็นมิติของสปาเชียล โดเมน เมื่อพิจารณาอาร์เรย์ของพิกเซลขนาด 8x8 พิกเซล การแปลงโคซายน์แบบไม่ต่อเนื่อง 1 ทิศทางจะถูกนำมาใช้แยกกันสำหรับแต่ละแถวและแต่ละหลัก การแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มจะทำการแปลงข้อมูลในลักษณะของพิกเซลไปเป็นสัมประสิทธิ์ทางความถี่ โดยจะรวมข้อมูลของภาพทั้งหมดแล้วแทนด้วยค่าสัมประสิทธิ์ทางความถี่ ซึ่งผลจากการแปลงจะมีลักษณะเป็นจุดๆเช่นเดียวกับจุดพิกเซลก่อนที่จะทำการแปลง แต่จุดต่างๆนี้จะแทนค่าสัมประสิทธิ์ทางความถี่แต่ละค่า และสมการที่ใช้ในการแปลงโคซายน์แบบไม่ต่อเนื่อง 2 ทิศทางนั้นมีสมการดังนี้

สมการการแปลงดีสครีตโคไซน์ (Forward DCT)

$$F(u,v) = \frac{1}{\sqrt{2N}} C(u)C(v) \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j) \cos\left[\frac{\pi(2i+1)u}{2N}\right] \cos\left[\frac{\pi(2j+1)v}{2N}\right] \qquad (2.7)$$

สมการการแปลงกลับดีสครีตโคไซน์ (Inverse DCT)

$$f(i,j) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} C(u)C(v)F(u,v) \cos\left[\frac{\pi(2i+1)u}{2N}\right] \cos\left[\frac{\pi(2j+1)v}{2N}\right] \qquad (2.8)$$

โดย

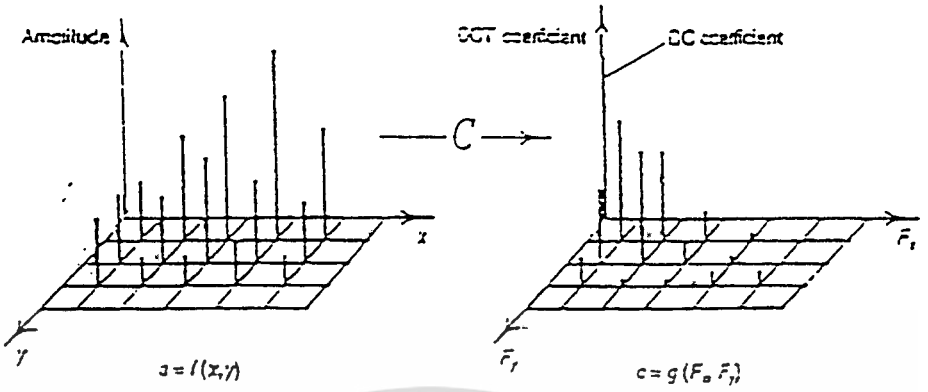
$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & ;u,v = 0 \\ 1 & ;other \end{cases}$$

และ

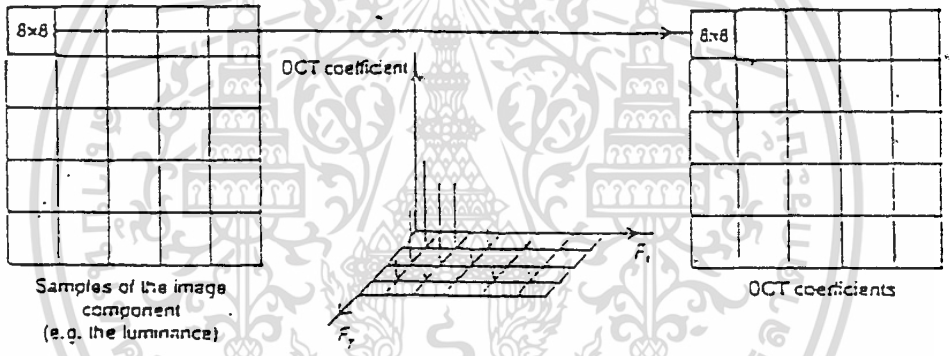
f(i,j)เป็นข้อมูลภาพเริ่มต้นและภาพผลลัพธ์

F(u,v) เป็นสัมประสิทธิ์ของการแปลง

โดยตัวอย่างในรูปที่ 2.15 และ 2.16 แสดงภาพในลักษณะ 3 มิติ ของการแปลงแบบดิสครีตโคซายน์ของบล็อกขนาด 8 แถว x 8 หลัก



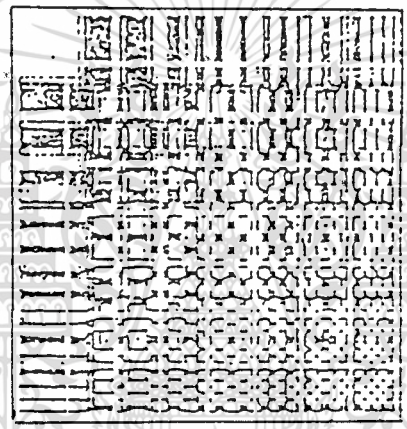
รูปที่ 2.15 แสดงการแปลงดีสครีตโคไซน์แบบ 2 มิติ



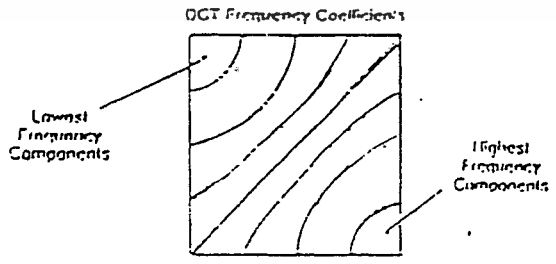
รูปที่ 2.16 แสดงการแปลงดีสครีตโคไซน์ที่ทำในบล็อกขนาด 8x8

หลังจากผ่านกระบวนการแปลงด้วยดีสครีตโคไซน์มาจะได้รูปแบบของฟังก์ชันที่อยู่ในโดเมนของความถี่ ซึ่งจะได้สัมประสิทธิ์ทางความถี่ที่อยู่ในรูปของบล็อกขนาด 8x8 ค่า (8 แถว และ 8 หลัก) โดยสมมติให้มีฟังก์ชันเป็น $c = g(F_x,F_y)$ โดย c จะเป็นค่าของสัมประสิทธิ์ ส่วนค่าของ F_x และ F_y จะแสดงถึงค่าที่เกิดขึ้นในแกนของความถี่ในแต่ละทิศทาง สัมประสิทธิ์ของ g(0,0) นั้นจะเป็นสัมประสิทธิ์ที่ค่าความถี่ศูนย์ซึ่งจะเรียกว่าสัมประสิทธิ์ DC ซึ่งจะเป็นค่าเฉลี่ยของค่าในแต่ละบล็อก ค่าต่างๆในแถวที่ศูนย์จะมีส่วนประกอบทางความถี่เป็นศูนย์ในทิศทางหนึ่งและค่าทั้งหมดในคอลัมน์ที่ศูนย์ก็จะมีส่วนประกอบทางความถี่เป็นศูนย์ในอีกทิศทางหนึ่ง เมื่อแถวและคอลัมน์เพิ่มขึ้นในทิศทางที่ห่างออกจากจุดเริ่มต้นสัมประสิทธิ์ในเมตริกที่ได้จากการแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มจะแทนความถี่ที่สูงขึ้น และความถี่ที่สูงที่สุดคือที่ตำแหน่ง N-1 ของบล็อกข้อมูล ซึ่งโดยปกติพิกเซลจะมีการเปลี่ยนแปลงน้อยลงจากจุดถึงจุด ดังนั้นขนาดของความถี่ที่ต่ำที่สุดจะสูงที่สุด ส่วนค่าของความถี่กลางและสูงจะมีค่าที่น้อยหรืออาจจะเป็นศูนย์ซึ่งในส่วนนี้จะไม่นำมาใช้อาจจะตัดทอนลงไป ดังจะเห็นจากรูปที่ 2.16 ซึ่งจะเป็นประโยชน์ต่อการลดขนาดของภาพลงได้ในขั้นตอนของการควอนไตเซชัน

และเมื่อนำแต่ละสัมประสิทธิ์ทางความถี่มาผ่านการแปลงกลับโคซายน์แบบไม่ต่อเนื่องแบบ 2 ทิศทางแล้ว จะได้ว่าแอมปลิจูดเป็นพื้นที่ที่มีความขาวดำด้วยอัตราความคมชัด(Constrast Ratio) ซึ่งกำหนดโดยแอมปลิจูดซึ่งในรูปที่2.17 แสดงถึงแพทเทิร์นที่สร้างจากค่าสัมประสิทธิ์ทางความถี่ของการแปลงโคซายน์แบบไม่ต่อเนื่องของ 8 x8 พิกเซล สมมุติว่าแต่ละค่าเป็นค่าสูงที่สุดแล้ว จะเห็นได้ว่าบล็อกบนซ้ายมือจะเป็นสีเทาทึบแสดงถึงค่าเฉลี่ยของความเข้มทั้งบล็อกขนาด 8x8 พิกเซล ส่วนบล็อกที่อยู่มุมล่างขวามือจะมีลักษณะเป็นตารางหมากรุก เพราะมีส่วนคอมโพเนนท์ทางความถี่ที่สูงสุดตามแนวตั้งและแนวนอน บล็อกที่มุมขวามือด้านบนและซ้ายมือด้านล่างแสดงแถบตามแนวนอนและแนวตั้งตามลำดับ เพราะมีความถี่สูงสุดทางแนวนอนแต่ต่ำสุดตามแนวตั้งและความถี่สูงสุดทางแนวตั้งแต่ต่ำสุดทางแนวนอนตามลำดับ ดังรูปที่ 2.17



รูปที่ 2.17 แสดงแพทเทิร์นของความเข้มของสัมประสิทธิ์การถ่วงน้ำหนักการแปลงโคซายน์แบบไม่ต่อเนื่อง



รูปที่ 2.18 การกระจายของส่วนประกอบทางความถี่ (ประมาณ) ของการแปลงโคซายน์แบบต่อเนื่อง 2 ทิศทาง

2.8 การ เข้ารหัสข้อมูล

กระบวนการสุดท้ายของ JPEG คือการเข้ารหัสข้อมูลที่ผ่านการควอนไตซ์แล้ว การเข้ารหัสข้อมูลประกอบด้วย 3 ขั้นตอนคือ

1. การเก็บค่าสัมประสิทธิ์กระแสตรงของบล็อกพิกเซล [8x8] บล็อกแรกก่อน แล้วทำการเปลี่ยนค่าสัมประสิทธิ์กระแสตรงที่จุด (0,0) ของบล็อกต่อมาเป็นค่าที่สัมพัทธ์กับค่าแรกหรือเป็นค่าผลต่างนั่นเอง แล้วจึงทำการเข้ารหัสข้อมูลผลต่างนี้แทน เพื่อประโยชน์คือสามารถเข้ารหัสโดยใช้ค่าผลต่างที่ถือว่าน้อยด้วยจำนวนบิท.ที่น้อยกว่าการเข้ารหัสค่าจริง (เพราะถือว่าบล็อกของพิกเซลที่อยู่ใกล้กันจะมีส่วนที่แตกต่างกันน้อยมากและสัมประสิทธิ์กระแสตรงก็ได้รวมข้อมูลที่สำคัญของภาพไว้แล้ว)

2. การจัดลำดับสัมประสิทธิ์กระแสสลับเป็นลำดับแบบซิกแซกเพื่อให้ 0 ที่เกิดขึ้นจากกระบวนการควอนไตเซชันมีความต่อเนื่องกันยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย

3. เลือกทำการเข้ารหัสสัมประสิทธิ์กระแสสลับที่จัดลำดับแล้ว

3.1 ด้วยวิธีความต่อเนื่องของข้อมูล RLE (Run Length Encoding) ซึ่งข้อดีของวิธีนี้คือถ้า 0 ที่เรียงติดกันมีความต่อเนื่องกันยาวๆ มาก จะทำให้สามารถเข้ารหัสได้ง่าย สั้นลง

3.2 การเข้ารหัสแบบเอนโทรปี (Entropy Encoding) ซึ่งอาจจะทำได้โดยวิธีฮัฟแมน หรือการประมวลผลทางคณิตศาสตร์(Arithmatic) ก็ได้ ซึ่งในที่นี้จะใช้การเข้ารหัสแบบฮัฟแมน

ซึ่งในวิธีการของเจเป็กจะใช้วิธีการเข้ารหัสข้อมูลแบบ Huffman Encoding ซึ่งเป็นการเข้ารหัสข้อมูลด้วยการสร้างรหัสขนาดต่าง ๆ ขึ้นจากสถิติของจำนวนข้อมูลที่เกิดขึ้น เพื่อนำรหัสที่สร้างนั้นไปใช้แทนตัวข้อมูลเดิม โดยพยายามสร้างรหัสที่มีขนาดสำหรับใช้แทน ตัวข้อมูลที่เกิดขึ้นจำนวนมาก และ รหัสที่มีขนาดยาวขึ้นเพื่อแทน ตัวข้อมูลที่เกิดขึ้นน้อยลงลดหลั่นกันไปตามลำดับ

การเข้ารหัสข้อมูลแบบฮัฟแมน

การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นเป็นการเข้ารหัสเฉพาะข้อมูลที่มีค่าไม่เป็นศูนย์ ดังนั้นจากการพยายามลดค่าของข้อมูลให้เป็นศูนย์มากๆในขั้นตอนของการควอนไตเซชันและการอ่านข้อมูลแบบซิกแซกที่พยายามทำให้ค่าที่เป็นศูนย์มากอยู่เรียงกันนั้นจึงมีประโยชน์ต่อการเข้ารหัสข้อมูลแบบฮัฟแมนมาก เพราะจะทำให้ข้อมูลที่ต้องทำการเข้ารหัสมีจำนวนน้อยลง การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นจะแบ่งการเข้ารหัสข้อมูลเป็นสองส่วนคือการเข้ารหัสข้อมูลของค่า DC และการเข้ารหัสข้อมูลของค่า AC

การเข้ารหัสค่า DC ของแถวข้อมูล

ในการเข้ารหัสค่า DC ของแถวข้อมูลนั้น (ข้อมูลตัวแรกในแถวซึ่งคิดค่าสเปกตรัมสัมประสิทธิ์กระแสตรงที่จุด (0,0) ของบล็อกพิกเซลขนาด 8X8) จะถูกเข้ารหัสเฉพาะค่าความแตกต่างระหว่างค่า DC ของแถวข้อมูลปัจจุบัน กับค่า DC ของแถวข้อมูลซึ่งเป็นสีชนิดเดียวกัน ที่ถูกเข้ารหัสแถวล่าสุดที่ผ่านไป เช่น เมื่อกำลังทำการเข้ารหัส ค่า DC ของแถวข้อมูล C_6 อยู่ซึ่งมีค่าเท่ากับ 40 จะทำการหาค่าความแตกต่างกัน ค่า DC ของแถวข้อมูล C_0 ที่ถูกเข้ารหัสผ่านไปแล้วแถวล่าสุด (สมมติว่ามีค่าเท่ากับ 15) ดังนั้นค่าที่จะถูกนำมาเข้ารหัสคิด 25 (40-15) ซึ่งการเข้ารหัสข้อมูลค่า DC จะมีการเข้ารหัสสองส่วนคือ

1. SIZE หมายถึง จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูล (ผลต่าง) ซึ่งจะหาจากตารางฮัฟแมน

2. AMPLITUDE หมายถึง ค่าขนาดของผลต่าง

ดังนั้นรหัสที่เข้า คือ (SIZE) (AMPILTUDE)

ในตารางฮัฟแมนจะเก็บรหัสความยาวต่าง ที่ใช้ในการแทนข้อมูลตามค่า Category ของข้อมูลนั้น ซึ่งการหาค่า Category ของข้อมูลสามารถหาได้จากตาราง JPEG Coefficieat Coding Categories ซึ่งแสดงไว้ในตารางที่ 2.7 เมื่อได้ค่าCategory แล้วก็สามารถหารหัสข้อมูลได้จากตารางค่า DC ของฮัฟแมน(ตารางที่ 2.8)โดยการเทียบหาตามค่า Category ที่ได้จากตาราง 2.7

Range	DC Differrence Category	AC Category
0	0	N/A
-1,1	1	1
-3,-2,2,3	2	2
-7,..,-4,4,..,7	3	3
-15,..,-8,8,..,15	4	4
-31,..,-16.16,..,313	5	5
-63,..,-32,..,63	6	6
-127,..,-64,..,127	7	7
-255,..,-128,128,..,255	8	8
-511,..,-256,511,..,256	9	9
-1023,..,-512,512-,..,1023	A	A
-2047,..,-1024.1027,..,2047	B	B
-4095,..,-2048,2048,..,4095	C	C
-8191,..,-4096,4096,..,-8191	D	D
-16382,..,-8192,8192,..,16383	E	E
-32767,..,-16384,19384,..,32767	F	F

ตารางที่ 2.7 ตารางแสดงค่า Categories ของข้อมูล

Table 6.15 JPEG Default DC Code (Luminance)

Category	*Base Code*	*Length*	*Category*	*Base Code*	*Length*
0	010	3	6	1110	10
1	011	4	7	11110	12
2	100	5	8	111110	14
3	00	5	9	1111110	16
4	101	7	A	11111110	18
5	110	8	B	111111110	20

ตารางที่ 2.8 แสดงตารางค่า DC ของการเข้ารหัสแบบฮัฟแมน

จะเห็นว่ารหัสที่ได้จากตารางที่ 2.8 นั้นใช้เป็นตัวบอกช่วงของข้อมูลที่ถูกเข้ารหัสเท่านั้นซึ่งจะได้รหัสตัวที่หนึ่งของข้อมูลออกมา ส่วนการระบุค่าของข้อมูลนั้นจำเป็นจะใช้รหัสตัวที่สองเป็นตัวระบุ โดยรหัสตัวที่สองนี้จะหามาจากค่าของข้อมูลโดยตรงโดยจะมีจำนวนบิทเท่ากับค่า Category ของข้อมูลที่นำมาเข้ารหัส นั่นเอง และจะเอาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

การเข้ารหัสค่า AC ของแถวข้อมูล

เนื่องจากการจัดเรียงข้อมูลแบบซิกแซก นั้น จะทำให้ค่าศูนย์ที่เกิดขึ้นจากการควอนไตซ์อยู่เรียงกันอย่างต่อเนื่องยาว ๆ ทำให้สามารถเข้ารหัสได้ง่าย และยิ่งมีศูนย์ที่ต่อเนื่องกันมากก็จะสามารถเข้ารหัสด้วยจำนวนที่น้อยลงได้มาก และโดยมากค่าศูนย์จะต่อเนื่องกันอยู่ในส่วนท้าย ๆ ของแถวข้อมูล

การเข้ารหัสค่า AC (ข้อมู ลตั้งแต่ตัวที่ 2 ถึงตัวที่64 ในแถวข้อมูล) จะต่างจากในการเข้ารหัสค่า DC คือ จะทำการเข้ารหัสค่าของข้อมูลโดยตรง (ไม่เข้ารหัสเฉพาะค่าความแตกต่างเหมือนใน DC) และการเข้ารหัสค่า AC นั้นจะทำการเข้ารหัสเฉพาะค่า AC ที่มีค่าไม่เป็นศูนย์เท่านั้น ส่วนค่า AC ที่เป็นศูนย์นั้น JPEG จะอาศัยการเข้ารหัสแบบ run-length มาช่วยในการเข้ารหัสดังนี้คือในการเข้ารหัสค่า AC ที่ไม่เป็นศูนย์แต่ละตัวจะเน้นจำนวนข้อมูล AC ที่เป็นศูนย์ซึ่งอยู่ติดกันด้านหน้าของข้อมูลที่จะเข้ารหัสนั้นโดยถือเป็นค่า Run ของข้อมูลในการเข้ารหัส เช่น ค่า AC ภายในแถวเป็น 5 8 0 0 9 7 0 0 0 0 4 _ _ _ จะทำการเข้ารหัสเฉพาะข้อมูลที่ไม่เป็นศูนย์และมีค่า Run ของข้อมูลแต่ละตัวดังนี้คือ 5 (Run = 0) , 8 (Run = 0) , 9 (Run - 2) , 7 (Run = 0), 4 (Run = 4) ..... โดยมีการเข้ารหัสข้อมูลดังกล่าวคล้ายใน DC แต่จะใช้รหัส 3 ตัว แทนข้อมูลแต่ละตัวดังนี้

1.RUNLENGTH คือ จำนวนค่า 0 ที่ต่อเนื่องกันก่อนหน้าข้อมูลตัวที่จะเข้ารหัส

2. SIZE คือ จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูลที่ไม่เท่ากับ 0 ซึ่งหาจากตารางฮัฟแมน

3. AMPLITUDE คือ ค่าแอมปลิจูดหรือค่าของข้อมูลที่ไม่เท่ากับ 0 นั่นเอง

ดังนั้นรหัสคือ (RUNLENGTH,SIZE) (AMPLITUDE)

หมายเหตุ

มีเงื่อนไขพิเศษ คือ ถ้ามี 0 ต่อเนื่องกัน มากกว่า 16 ตัว สมมุติว่าเป็น 22 ตัว จะได้รหัสเป็น

(15,0) (6,4) (13)

(15,0) ความหมายคือ มี 0 ต่อเนื่องกัน 16 ตัว (เป็นรูปแบบที่กำหนดตายตัว)

(6,4)(13) ความหมายคือ มี 0 อีก 6 ตัว

ใช้ 4 บิทในการแทนค่าข้อมูลที่มีค่าเท่ากับ 13

ซึ่งตารางที่ใช้ในการดูค่าจำนวนบิทที่ใช้ต่างๆกัน เรียกว่าเป็นรหัสแบบความยาวของรหัสไม่คงที่หรือ (Variable Length Code) เนื่องจากข้อมูลที่นำมาเข้ารหัสจะมีค่า Run เข้ามาเกี่ยวข้องดังนั้นในตารางค่า AC ของฮัฟแมน จะต่างจากในตารวงค่า DC ของฮัฟแมน คือจะมีการเปลี่ยนแปลงจาก Category ใน DC เป็น Run/Category นั่นคือการหารหัสข้อมูลจากตารางค่า AC ของฮัฟแมน (ในภาคผนวก) จะต้องทราบค่า Run และ Category ของข้อมูลที่นำมาเข้ารหัส ซึ่งการหาค่า Category ของข้อมูล AC จะเทียบหาจากตาราง JPEG Coefficient Coding Categories ตามตารางที่ 2.7 เช่นเดียวกับใน DC เมื่อทราบค่า Run และ Category ของข้อมูลแล้วก็จะสามารถหา รหัสข้อมูลจากตารางค่า AC ของฮัฟแมนได้โดยเทียบตามค่า Run/Category

และภายในตารางค่า AC ของฮัฟแมนจะมีรหัสพิเศษ 2 ตัวคือ รหัสเมื่อค่า Run/Category เท่ากับ 0/0 ใช้ในกรณีเมื่อทำการเข้ารหัสข้อมูลภายในแถวจนกระทั่งเหลือแต่ข้อมูลที่เป็นศูนย์เพียงอย่างเดียวก็จะใช้รหัสนี้เป็นตัวบอกตัวถอดรหัสว่าข้อมูลที่เหลือทั้งหมดภายในแถวมีค่าเป็นศูนย์ และ รหัสอีกตัวหนึ่งเมื่อ Run/Category เท่ากับ F/0 ใช้เมื่อมีข้อมูลที่มีค่าศูนย์อยู่ติดกัน 16 ตัวภายในแถว โดยยังมีข้อมูลที่ไม่เป็นศูนย์ที่ยังไม่ถูกเข้ารหัสเหลืออยู่ในแถวอีก ก็จะใช้รหัสตัวนี้แทน ข้อมูลที่มีค่าศูนย์ 16 ตัว ดังกล่าว

จะเห็นได้ว่ารหัสข้อมูลที่ได้จากตารางค่า AC ของฮัฟแมนนั้นใช้ในการบอกจำนวนศูนย์ (คือค่า Run) ที่อยู่ด้านหน้าของข้อมูลนั้นและช่วงของค่าข้อมูล(คือค่า Category) ที่นำมาเข้ารหัสดังนั้นจะต้องมีรหัสตัวที่สองสำหรับใช้ในการระบุค่าของข้อมูล ซึ่งการหารหัสตัวที่สองนี้ใช้วิธีเดียวกันกับใน DC คือเอามาจาก LSB จำนวน Category บิทของข้อมูลที่นำมาเข้ารหัสและจะเอามาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

2.9 รูปแบบข้อมูลที่ถูกบีบอัด

ในส่วนนี้เราจะอธิบายถึง มาร์คเกอร์ (marker) และ พารามิเตอร์ ต่าง ๆ ที่สร้าง รูปแบบของเจเป็กแบ่บ DCT baseline โดยการใช้การเข้ารหัสแบบฮัฟแมน ทุก ๆ มาร์ค จะอยู่ในรูปของรหัส 2 ไบต์ ไบต์ 0xFF ตามด้วยไบต์ที่ไม่เท่ากับ 0xFF หรือ ศูนย์ มาร์คไบต์แรกจะเป็นไบต์ใด ๆ ก็ได้ ตามด้วย ไบต์ 0xFF ตารางที่ 2.9 จะสรุป มาร์คเกอร์ เซกเมนต์ซึ่งจะกล่าวถึงในหัวข้อต่อไป

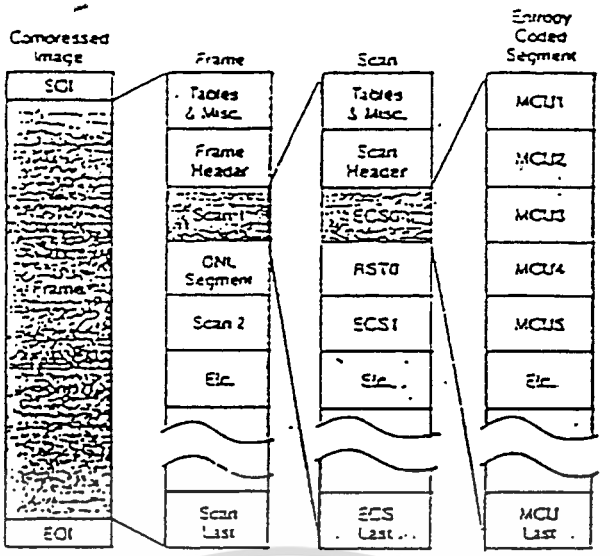
มาร์เกอร์เซกเมนต์ประกอบด้วยรหัสมาร์คเกอร์เซกเมนต์และ จำนวนพารามิเตอร์พารามิเตอร์ตัวแรกในมาร์คเกอร์เซกเมนต์มีค่า 2 ไบต์ จะกำหนดความยาวของเซกเมนต์ค่ากำหนด ความยาวนี้จะรวมความยาวของพารามิเตอร์ แต่ไม่รวมมาร์คเกอร์ 2 ไบต์มาร์คเกอร์ เซกเมนต์ จะกำหนดโดย มาร์คเกอร์เริ่มต้นเฟรม (SOF) ซึ่งเรียกอีกอย่างหนึ่งว่า เฟรมเฮดเดอร์ หรือ หัวเฟรม(frame header)

0xFFC0	(SOF) Start of Frame Marker
0xFFC4	(DHT) Define Huffman Table
0xFFD0 to 0xFFD7	(RST) Restart Interval Termination
0xFFD8	(SOI) Start of Image
0xFFD9	(EOI) End of Image
0xFFDA	(SOS) Start of Scan
0xFFDB	(DQT) Define Quantization Table(s)
0xFFDC	(DNL) Define Number of Lines
0xFFDD	(DRI) Define Restart Interval
0xFFE0 to 0xFFEF	(APP) Application Segment Markers
0xFFFE	(COM) Start of Comment

ตารางที่ 2.9 รหัสมาร์คเกอร์เซกเมนต์

เซกเมนต์ข้อมูลการเข้ารหัสแบบเอนโทรปี จะมีข้อมูลของภาพที่ถูกบีบอัดจริง ๆ แต่ละเซกเมนต์ของข้อมูลจะประกอบด้วย จำนวนจริงของไบต์ซึ่งเกี่ยวกับ บิทสตรีมที่ใช้ในการเข้ารหัสเอนโทรปีจำนวนบิทที่เป็นจำนวนเท่าของ สำหรับการเข้ารหัสฮัฟแมน ถ้าบิทสตรีมไม่สามารถเติมเต็มไบต์ในตอนท้าย ไบต์สุดท้ายจะถูกเติมเต็มด้วยบิทที่มีค่า '1' 8 บิท เกิดขึ้นในบิทสตรีมฮัฟแมน ไบต์ที่ตามมาจะเป็นไบต์ '0' เพื่อตัดไม่ให้เป็นมาร์คเกอร์จากนั้นบิท '1' 8บิท จะถูกถอดรหัสเป็นบิทสตรีมฮัฟแมน และจะไม่สนใจบิท '0' 8 บิท ที่ตามมา รูปที่ 2.19 แสดงบิทสตรีม การบีบอัดสำหรับ การบีบอัดภาพแบบ JPEG ชนิด DCT Sequential base - line

สิ่งที่ควรรู้คือ การบีบอัดภาพแบบ เจเป็ก นั้นไม่สามารถใช้สำหรับการเข้ารหัสแบบหลายเฟรมซึ่งเข้ากันได้ แต่ละเฟรมเดียว ๆ จะเริ่มต้นด้วยมาร์คเกอร์แสดงจุดเริ่มต้นภาพ (SOI) : Start of image และสิ้นสุดด้วยเฟรม ข้อมูลภาพจะแบ่งเป็นเซกเมนต์ต่าง ๆ เฟรมจะเริ่มต้นด้วยหัวเฟรม (frame header) ตามด้วยชุดของการสแกน ก่อนเฟรมเฮดเดอร์อาจจะมีตารางระบุค่าหรือระบุมาร์คเกอร์เซกเมนต์ต่าง ๆ ซึ่งตารางระบุค่าจะกำหนดตารางค่าคงที่ที่ใช้ในกระบวนการ DCT และตารางการเข้ารหัสฮัพแมนรูปแบบของตารางระบุค่าจะกล่าวถึงต่อไป สแกน (Scan) จะมีเซกเมนต์การเข้ารหัสเอนโทรปี ซึ่งแต่ละเซกเมนต์จะประกอบด้วยบล็อก MCU บล็อกหรือมากกว่า



รูปที่ 2.19 ฟอร์แมทของข้อมูลคอมเพรสบิทสตรีม

2.9.1 เฟรมเฮดเดอร์ (Frame header)

เฟรมเฮดเดอร์ประกอบด้วยฟิลด์ข้อมูลที่มีค่าคงที่มากมาย ซึ่งมีขนาดต่าง ๆ กัน รวมทั้งบล็อกของพารามิเตอร์ ซึ่งกำหนดคอมโพเนนท์ต่าง ๆ ส่วนที่คงที่ของเฟรมเฮดเดอร์มี 6 ฟิลด์ ซึ่งแต่ละฟิลด์อธิบายในตารางที่ 2.10

- ฟิลด์เริ่มต้นเฟรม (Start of trame : SOF)

จะระบุตำแหน่งเริ่มต้นของเฟรมเฮดเดอร์ สำหรับในที่นี้ จะใช้การเข้ารหัสแบบ Sequential baseline ค่ามาร์คเกอร์ SOF ที่ใช้คือ 0x FFCD

- ความยาวของเฟรมเฮดเดอร์ (Frame header Length : LP)

เป็นตัวเลข 16 บิท ซึ่งจะระบุความยาวในหน่วยไบต์ ของ เฟรมเฮดเดอร์ซึ่งจะรวมฟิลด์ความยาวด้วย แต่ไม่รวมมาร์คเกอร์ SOF

- ความเที่ยงตรงของแซมเปิล (Sample Precision : P)

ฟิลด์นี้จะชี้ค่าบิทที่ใช้ในแต่ละแซมเปิล สำหรับแต่องค์ประกอบในเฟรมสำหรับการเข้ารหัส แบบ DCT Sequential base line ค่านี้จะเป็น '8' เสมอ

- จำนวนไลน์ (Number of Samples per Line : X)

ฟิลด์นี้จะชี้ค่าแถวพิกเซลของคอมโพเนนท์ด้วย จำนวนแซมเปิลในแนวตั้งสูงสุด ถ้าใช้การบีบแซมเปิล ถ้าไม่ได้ใช้การซับแซมปลิง ฟิลด์นี้จะชี้จำนวนแถวในแต่ละคอมโพเนนท์ ฟิลด์นี้มีค่าตั้งแต่ 0 ถึง 65535

-จำนวนแซมเปิลต่อแถว (Number of Sample per Line : X)

จะระบุค่าของจำนวนพิกเซลตามขวาง ถ้าใช้การซับแซมปลิง จะระบุจำนวนพิกเซลตามขวางในคอมโพเนนท์ ตามขนาดตามขวางที่กว้างที่สุด ฟิลด์นี้สามารถมีค่าตั้งแต่ 1 ถึง 65535

- จำนวนอิมเมจคอมโพแนนท์ในเฟรม (Number of Image Components in frame (Nf))

ตามปกติจะมีค่า 3 สำหรับภาพโมโนโครมจะมีค่า 1 ช่วงของค่านี้จะมีค่าประมาณ 1 ถึง 255 ต่อไปจะเป็นฟิลด์ของบล็อกพารามิเตอร์ระบุคอมโพแนนท์ต่าง ๆ แต่ละบล็อกประกอบด้วย 4 ฟิลด์ 1 บล็อกสำหรับแต่ละคอมโพแนนท์ในเฟรม ซึ่งมีฟิลด์ต่าง ๆ ดังนี้

- ฟิลด์ระบุคอมโพแนนท์ (Component Identifier : C ;)

ใช้เพื่อสมมุติค่าให้แต่ละคอมโพแนนท์และแสดงลำดับของคอมโพแนนท์ในเฟรม ตัวเลขเหล่านี้จะถูกนำไปใช้ในตัวเลข สแกนเฮดเดอร์ (Scan header) เพื่อระบุคอมโพแนนท์ที่ใช้ในสแกนปกติจะสมมุติค่าคอมโพแนนท์แรกเป็น 0 , คอมโพแนนท์ ที่ 2 เป็น 1 , คอมโพแนนท์ที่ 3 เป็น 2 เป็นอย่างนี้ต่อไปเรื่อย ๆ

- ค่าแซมปลิงแฟคเตอร์ตามขวาง (Horizontal Sampling Factor : Hi) ค่านี้จะระบุจำนวนบล็อค 8x8 ของคอมโพแนนท์ซึ่งใช้ในแต่ละ MCU ตามขวาง

- ค่าแซมปลิงแฟคเตอร์ แนวตั้ง (Vertical Sampling Factor : Vi)

ค่านี้จะระบุจำนวนบล็อค 8x8 ของคอมโพแนนท์ซึ่งใช้ในแต่ละ MCU ตามขวาง

- ตัวเลือกตารางควอนไตเซชั่น (Quantization Table Selector : Tqi)

เนื่องจากมีตารางควอนไตเซชั่น ที่แตกต่างกันถึง 4 ตาราง เพื่อใช้ในการดีควอนไตเซชั่นฟิลด์นี้จะเลือกตารางที่จะใช้ การเลือกตารางจะไม่เปลี่ยนแปลงจนกว่าทุก ๆ องค์ประกอบในสแกนได้รับการถอดรหัสหมดแล้ว

Parameter	Size (Bits)	Values	Description
SOF	16	0xFFC0	Start of Frame
Lf	16	8+3*Nf	Frame Header Length
P	8	8	Sample Precision
Y	16	0–65535	Number of Lines
X	16	1–65535	Samples per Line
Nf	8	1–255	Number of Image Components

ตารางที่ 2.10 ฟิลด์ฟีกเฮดเดอร์

Parameter	Size (Bits)	Values	Description
C_i	8	0–255	Component Identifier
H_i	4	1–4	Horizontal Sampling Factor
V_i	4	1–4	Vertical Sampling Factor
Tq_i	8	0-3	Quantization Table Select

ตารางที่ 2.11 ตารางองค์ประกอบของฟิลด์เฮดเดอร์

2.9.2 ซินแทกซ์ของสแกนเฮดเดอร์ (Scan header syntix)

เนื่องจากในเฟรมจะประกอบด้วยเฟรม เฮดเดอร์ ตามด้วยบล็อกสแกน (Scan) 1บล็อกหรือมากกว่า บล็อกสแกนประกอบด้วยสแกนเฮดเดอร์ (Scan header) ตามด้วยเซกเมนต์การเข้ารหัสแบบเอนโทรปี เหมือนกับในเฟรมเฮดเดอร์ สแกนเฮดเดอร์ประกอบด้วยฟิลด์ข้อมูลที่มีค่าคงที่ และแปรค่าได้ ฟิลด์แรกเป็นมาร์คเกอร์แสดงถึงตำแหน่ง เริ่มต้นสแกน (Start of Scan) และฟิลด์อื่น ๆ จะได้อธิบายต่อไป

- ความยาวของสแกนเฮดเดอร์ (Scan Header Length : Ls)

ระบุความยาวของสแกนเฮดเดอร์รวมทั้งฟิลด์ความยาว แต่ไม่รวมมาร์คเกอร์แสดงตำแหน่งเริ่มต้นสแกนระบุความยาวของสแกนเฮดเดอร์รวมทั้งฟิลด์ความยาว แต่ไม่รวมมาร์คเกอร์แสดงตำแหน่งเริ่มต้นสแกน

-จำนวน อิเมจคอมโพเนนท์ (Number of Image Components : Ns)

ฟิลด์นี้จะระบุว่ามีพารามิเตอร์ที่ระบุสแกนคอมโพเนนท์เท่าไร ที่ถูกรวมอยู่ในสแกนเฮดเดอร์ฟิลด์ที่ต่อจากฟิลด์นี้จะเป็นชุดพารามิเตอร์ระบุสแกน คอมโพเนนท์ค่าต่างๆ แต่ละชุดจะมี 3 องค์ประกอบ ดังนี้

- ตัวเลือกสแกนคอมโพเนนท์ (Csj) จากที่ในเฟรมเฮดเดอร์มีพารามิเตอร์ที่ระบุค่าคอมโพเนนท์ต่าง ๆ ในฟิลด์นี้ในสแกนแฮดเดอร์จะหมายถึงค่าชี้ค่าคอมโพเนนท์ในเฟรมเฮดเดอร์ต้องมีค่าตรงกัน ถ้ามีมากกว่าหนึ่งคอมโพแนนท์ถูกรวมอยู่ในสแกนจะทำให้ค่าซับแซมปลิงมีค่าจำกัน คือ ผลรวมของผลคูณของค่าซับแซมปลิงในแนวตั้งและแนวนอนของทุก ๆ คอมโพเนนท์ภายในสแกนต้องมีค่าไม่เกิน 10

ยกตัวอย่างเช่น ถ้าใช้ 3 คอมโพเนนท์ 2 ใน 3 ถูกซับแซปลิ้งด้วยค่า z ทั้งในแนวตั้งและแนวนอน ผลรวมของผลคูณคือ x+2*2+2*2 เท่ากับ 9 ถ้าใช้ค่าซับแซมปลิงมากกว่า 2 คอมโพเนนท์เหล่านั้น จะถูกเข้ารหัสแยกแต่ละบล็อกสแกน

- ตัวเลือกตารางการเข้ารหัส DC เอนโทรปี (Dc Entropy coding Table selector : Tdj)

ตัวเลือกนี้จะเลือกตาราง 1 ใน 4 ของตาราง DC เอนโทรปี เพื่อใช้ในการถอดรหัสสัมประสิทธิ์ DC สำหรับคอมโพเนนท์ในฟิลด์ก่อนหน้านี้ ซึ่งหมายถึง 1 ใน ตารางที่ระบุไว้ก่อนหน้านี้ในมาร์คเกอร์เซกเมนต์ระบุตาราง

- ตัวเลือกตารางการเข้ารหัส AC เอนโทรปี (AC Entropy Coding Table Selector : Tdj)

ฟิลด์นี้จะเลือกตาราง 1 ใน 4 ของตาราง AC เอนโทรปี เพื่อใช้ในการถอดรหัส สัมประสิทธิ์ AC ของคอมโพเนนท์ที่กำหนดในสแกน ตารางการเข้ารหัสเอนโทรปี AC ต้องมีการกำหนดไว้ก่อนหน้านี้ในมาร์คเกอร์เซกเมนต์ 1 ชุดของทั้ง 3 ฟิลด์ ที่กล่าวมาแล้วนี้ต้องแสดงในสแกนเฮดเดอร์สำหรับแต่ละคอมโพเนนท์

- เริ่มต้น การเลือกสเปิ้กตรัล (End of Spectral Selection : Se)

ฟิลด์นี้ระบุสัมประสิทธิ์ DCT ตัว สุดท้ายในแต่ละบล็อกซึ่งถูกเข้ารหัสในสแกนสำหรับกระบวนการ DCT sequential base line ฟิลด์นี้จะถูกเซตให้มีค่า 6 เพื่อแสดงว่าเป็นสัมประสิทธิ์ตัวสุดท้ายในแต่ละบล็อก DCT Succesive Approximation Bit Positions High and Low (Ah & Al) ฟิลด์นี้จะถูกเซตให้เป็น 0 ตารางที่ 2.11 สรุปฟิลด์ต่าง ๆ ทั้งหมดในสแกนเฮดเดอร์

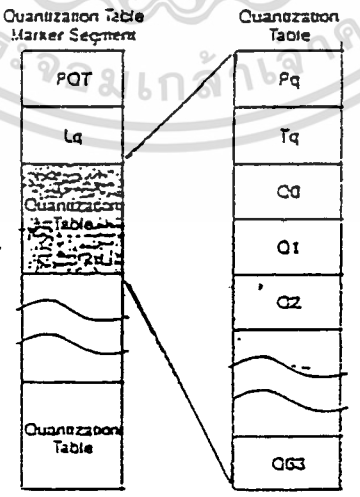
Parameter	Size (Bits)	Values	Description
SOS	16	0xFFDA	Start of Scan
Ls	16	2Ns + 6	Scan Header Length
Ns	8	1–4	Number of Image Components
Csi	8	0–255	Scan Component Selector
Tdj	4	0–1	DC Coding Table Selector
Tai	4	0–1	AC Coding Table Selector
Ss	8	0	Start of Spectral Selection
Se	8	63	End of Spectral Selection
Ah	4	0	Successive Approximation Bit High
Al	4	0	Successive Approximation Bit Low

ตารางที่ 2.12 ฟิลด์สแกนเฮดเดอร์

2.9.3 ซินแทกซ์ของตารางควอนไตเซชั่น

ตารางควอนไตเซชั่นจะถูกกำหนดในมาร์คเกอร์เซกเมนต์ ซึ่งเริ่มต้นด้วย มาร์คเกอร์ ที่ระบุตารางควอนไตเซชั่น (DQT) OxFFD8 ตารางควอนไตเซชั่น ทั้ง 4 ตารางสามารถกำหนดได้ด้วยมาร์คเกอร์เซกเมนต์ค่าเดียว ฟิลด์ 16 บิท นี้จะตามด้วยมาร์คเกอร์ตาราง (Lq) ระบุความยาวทั้งหมดของตาราง

รูปที่ 2.20 แสดงการจัดฟิลด์ ซึ่งต่อจาก Lq แล้วจะเป็นฟิลด์ของแต่ละตารางควอนไตเซชั่นฟิลด์แรกในตารางควอนไตเซชั่น (Pq) จะระบุค่าความแม่นยำ ของสัมประสิทธิ์ควอนไตเซชั่น ถ้าฟิลด์นี้เป็น 0 จะใช้สัมประสิทธิ์การควอนไตซ์ ชนิด 8 บิทถ้าฟิลด์นี้เป็น 1 จะใช้สัมประสิทธิ์การควอนไตซ์ชนิด 16 บิท



รูปที่ 2.20 การจัดตารางควอนไตเซชัน

สำหรับโหมด DCT Sequential base line จะใช้สัมประสิทธิ์การควอนไตซ์ชนิด 8 บิทเท่านั้น ตารางที่ 2.13 สรุปฟิลด์มาร์คเกอร์เซกเมนต์ที่ใช้กำหนดตารางควอนไตเซชั่น

Parameter	Size (Bits)	Values	Description
DQT	16	0xFFDB	Define Quantization Table Marker
Lq	16	65n+2 *	Quantization Table Length .
Pq	4	0	Quantization Table Precision
Tq	4	0-3	Quantization Table Identifier
Qk	8	1-255	Quantization Table Element

ตารางที่ 2.13 ฟิลด์ตารางควอนไตเซชัน

2.9.4 ซินแทกซ์ของตารางฮัฟแมน

ในหัวข้อนี้จะอธิบายว่าตารางฮัฟแมนถูกกำหนดไว้อย่างไร ในมาร์คเกอร์ เซกเมนต์ ระบุตารางฮัฟแมน (DHT) แต่ละมาร์คเซกเมนต์ สามารถกำหนดตารางฮัฟแมนจาก 1 ถึง 4 ตารางได้ เซกเมนต์เริ่มต้นด้วยมาร์คเกอร์ DHT 16 bit (OxFFC4) ซึ่งกำหนดตาราง ตามด้วยฟิลด์ความยาวของตารางฮัฟแมน (Lh) ซึ่งระบุความยาวเซกเมนต์ทั้งหมด ตามด้วย ตารางฮัฟแมน 1 ถึง 4 ตาราง ซึ่งแต่ละตารางประกอบด้วยฟิลด์ต่าง ๆ มากมาย รูปที่ 2.21 แสดงการจัดสรรฟิลด์ในตารางฮัฟแมนมีดังนี้

- Table Class (TC)

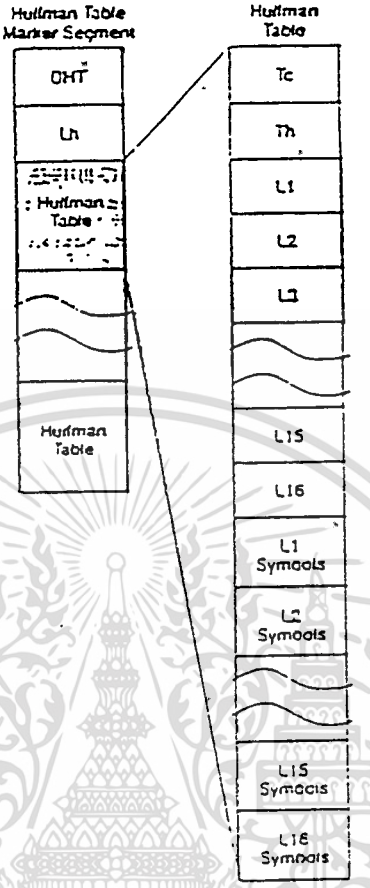
ฟิลด์นี้จะระบุว่าเป็นตารางไหนใน 4 ตาราง จะถูกโหลดค่าเข้ามา

- Huffman Code Counts (Li)

ฟิลด์ 16 ฟิลด์ระบุจำนวนรหัส ฮัฟแมนสำหรับทั้ง 16 เลนจ์ กำหนดโดยมาตรฐานเจเป็กฟิลด์แรกระบุว่ามีรหัส 1 บิท เท่าไร , ฟิลด์ที่ 2 คือมีรหัส 2 บิทอยู่เท่าไร เป็นเช่นนี้ไปเรื่อยๆ

- Huffman Code table (Vij)

เป็นตารางค่าจริงที่เกี่ยวข้องกับแต่ละรหัสฮัฟแมน ยกตัวอย่างเช่นถ้ารหัส 1 บิท ไม่ได้ถูกสมมุติไว้มีรหัส 2 บิทอยู่ 3 รหัสดังนั้นค่า 3 ค่าแรกในเซกเมนต์นี้เกี่ยวข้องกับรหัส 2 บิท ตารางที่ 2.13 สรุปฟิลด์มาร์คเกอร์เซกเมนต์ที่ระบุตารางฮัฟแมนและค่าที่เหมาะสมของแต่ละฟิลด์



รูปที่ 2.21 ข้อมูลมาร์คเกอร์เซกเมนต์ของตารางฮัฟแมน

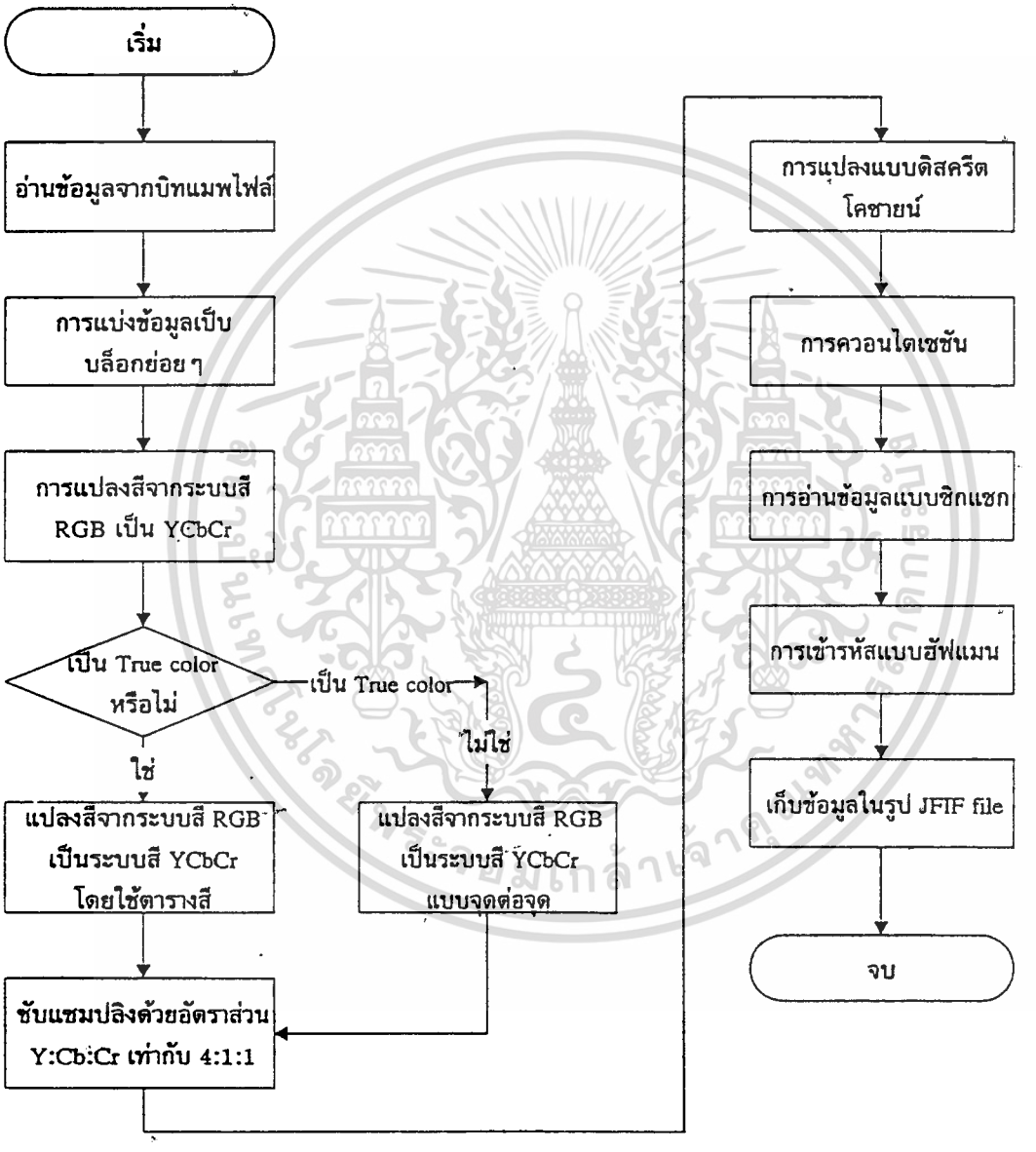
Parameter	*Size (Bits)*	*Values*	*Description*
DHT	16	0xFFC4	Define Huffman Table Marker
Lh	16	*	Huffman Table Definition Length
Tc	4	0,1	Table Class: 0=DC; 1=AC
Th	4	0,1	Huffman Table Identifier
Li	8	0–255	Huffman Code Length
Vij	8	0–255	Huffman Code Value

ตารางที่ 2.14 สรุปฟิลด์มาร์คเกอร์เซกเมนต์ที่ระบุตารางฮัฟแมน

บทที่ 3

การออกแบบและเขียนโปรแกรม

ผู้เขียนได้ออกแบบและเขียนโปรแกรมสำหรับบีบอัดภาพเพื่อใช้ทดลองในโครงงานนี้ โดยโฟลว์ชาร์ต (Flowchart) การทำงานคร่าว ๆ ของโปรแกรมเป็นดังนี้



รูปที่ 3.1 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรมอย่างคร่าว ๆ

โดยให้โปรแกรมอ่านข้อมูลภาพจากวินโดว์บิทแมพไฟล์ (Windows Bitmap file) แล้วนำมาบีบอัดด้วยกระบวนการเจเป็ก จากนั้นจัดเก็บข้อมูลที่ได้ไว้ในรูป JFIF file โดยมีลำดับขั้นตอนการทำงานของโปรแกรมดังนี้

3.1 การเตรียมข้อมูลสำหรับการบีบอัดข้อมูลแบบเจเป็ก

ภาพที่เก็บไว้ในวินโดว์บิทแมพไฟล์นั้นจะเป็นแบบ 2 มิติ (dimensional raster bitmap) มีทั้งชนิด 1, 4, 8 และ 24 บิท/พิกเซล โดยใช้ระบบสีแบบ RGB ซึ่งประกอบด้วยสี R,G,B (Red,Green,Blue) โดยในบิทแมพชนิด 1, 4 และ 8 บิท/พิกเซล สีที่ใช้ทั้งหมดจะเก็บไว้ในตารางสี ซึ่งจะมีจำนวนสีเท่ากับ 2, 16 และ 256 สีตามลำดับ จุดพิกเซลแต่ละจุดในบิทแมพชนิด 24 บิท/พิกเซลจะเก็บค่าสี R,G,B ไว้ในจุดพิกเซลโดยตรง ซึ่งการบีบอัดข้อมูลแบบเจเป็กชนิดซีเควนเชียลเบสไลน์นี้ไม่สามารถนำข้อมูลจากบิทแมพ นี้ไปใช้ได้ทันที จำเป็นต้องมีขั้นตอนในการเตรียมข้อมูลให้อยู่ในรูปแบบที่เหมาะสมก่อนคือ

3.1.1 การแบ่งส่วนภาพเป็นหน่วย MCU

ในการบีบอัดข้อมูลแบบเจเป็ก นั้นจะแบ่งการบีบอัดภาพเป็นหน่วยย่อย ๆ เรียกว่า MCU (Minimum Coded Unit) โดยมีขนาดหน่วยละ 8x8, 8x16, 16x8 หรือ 16x16 พิกเซล ขึ้นอยู่กับอัตราส่วนการซับแซมปลิงที่ใช้ (อธิบายในขั้นตอนซับแซมปลิง) ดังนั้นภาพที่มีขนาดที่ไม่สามารถแบ่งให้เป็นหน่วย MCU ได้ลงตัวพอดี จำเป็นต้องมีการเพิ่มขนาดของภาพเพื่อให้สามารถแบ่งเป็นหน่วย MCU ได้ครอบคลุมภาพทั้งหมดโดยไม่มีส่วนใดส่วนหนึ่งของภาพขาดหายไป การเพิ่มขนาดของภาพนี้อาจต้องเพิ่มจำนวนจุดพิกเซลทางด้านซ้ายของแต่ละแถวหรือเพิ่มจำนวนแถวของจุดพิกเซลทางด้านล่างของภาพหรือต้องเพิ่มทั้งคู่ สีของจุดพิกเซลในส่วนที่เพิ่มขึ้นนั้นควรมีค่าเท่ากับสีของจุดพิกเซลในภาพบริเวณขอบด้านที่จุดพิกเซลที่เพิ่มนั้นติดต่ออยู่ด้วยเนื่องจากถ้าสีของจุดพิกเซลที่เพิ่มขึ้นมีค่าต่างกันมากกับสีของจุดพิกเซลของภาพบริเวณนั้นจะทำให้ข้อมูลของ MCU ในภาพบริเวณนั้นมีการกระจายตัวอยู่ในระดับความถี่ที่สูง ซึ่งในการบีบอัดข้อมูลภาพจะมีการลดความสำคัญของข้อมูลในระดับความถี่สูงลงในขั้นตอนควอนไตเซชัน ซึ่งจะมีผลทำให้ข้อมูลภาพบริเวณนั้นเมื่อถูกขยายคืน (decompress) มีความผิดพลาด (error) มาก ถึงแม้ว่าในการขยายภาพจุดพิกเซลที่เพิ่มขึ้นจะถูกตัดทิ้งโดยอุปกรณ์หรือโปรแกรมที่ทำหน้าที่ขยายภาพโดยอัตโนมัติก็ตาม

3.1.2 การแปลงระบบสี RGB เป็นระบบสี YCbCr

ระบบสีที่ใช้ในเจเป็กนั้นเป็นระบบสี YCbCr ซึ่งประกอบด้วยค่าลูมิแนนซ์ (Luminance) (สี Y) และโครมิแนนซ์ (Chrominance) (สี Cb,Cr) โดยมีสมการการแปลงสีในระบบสี RGB เป็นสี Y,Cb และ Cr ในระบบ YCbCr ดังนี้

$$Y = 0.299R + 0.587G + 0.114B \tag{3.1}$$

$$Cb = -0.1687R - 0.3313G + 0.5B \tag{3.2}$$

$$Cr = 0.5R - 0.4187G - 0.0813B \tag{3.3}$$

โดย $0 \Leftrightarrow Y \Leftrightarrow 255, 0 \Leftrightarrow Cb \Leftrightarrow 255, 0 \Leftrightarrow Cr \Leftrightarrow 255$

1) การเปลี่ยนระบบสีในบิทแมพชนิดตารางสี (color table)

เนื่องจากสีที่ใช้ทั้งหมดในภาพบิทแมพชนิดใช้ตารางสี จะถูกเก็บไว้ในตารางสี ดังนั้นการแปลงระบบสี จะทำโดยการแปลงสีแต่ละสีตามลำดับใน ตารางสีซึ่งประกอบด้วยค่าสี R,G,B ไปเป็นสีY,Cb,Cr และ

เก็บไว้ในตารางสี ตารางใหม่ที่สร้างขึ้นอีกตารางหนึ่ง ด้วยวิธีนี้ค่าดัชนีที่เก็บไว้ในจุดพิกเซลสามารถใช้เลือกสีในระบบสี YCbCr จาก ตารางสี ตารางใหม่ได้ทันที

2) การแปลงระบบสีในบิทแมพชนิดทรูคัลเลอร์ (true color)

ส่วนบิทแมพ ชนิด 24 บิท / พิกเซล ค่าสี R,G,B ที่ใช้ในแต่ละจุดพิกเซลจะถูกเก็บไว้ที่จุดพิกเซลโดยตรง ดังนั้นการแปลงระบบสีในบิทแมพ ชนิดนี้จึงต้องทำการแปลงสีของจุดพิกเซลทีละจุดโดยนำค่าสี R,G,B จากจุดพิกเซลมาทำการแปลงสีเป็นค่าสี Y,Cb,Cr และเก็บค่าสีที่ได้ไว้ตามตำแหน่งของจุดพิกเซลนั้น ๆ

สาเหตุที่เจเปิกเลือกใช้ระบบสี YCbCr คือในระบบสี YCbCr นั้นได้มีการแยกสีออกเป็นค่าลูมิแนนซ์ และค่าโครมิแนนซ์อย่างชัดเจน และในระบบการมองเห็นของตามนุษย์นั้น ตามนุษย์จะมีความไวต่อการเปลี่ยนแปลงของแสงลูมิแนนซ์มากกว่าความเปลี่ยนแปลงของสีโครมิแนนซ์ ดังนั้นระบบสี YCbCr จึงมีประโยชน์ต่อการลดจำนวนข้อมูลสีต่อจำนวนจุดพิกเซลโดยการลดจำนวนสี ด้วยวิธีการซับแซมปลิง ซึ่งจะไม่ทำให้คุณภาพของภาพเปลี่ยนไปมากนัก (เพราะตามนุษย์สังเกตไม่เห็น)

3.1.3 การซับแซมปลิง ด้วยอัตรา Y:Cb:Cr เท่ากับ 4:1:1

การซับแซมปลิง เป็นการลดจำนวนข้อมูลด้วยการแทนกลุ่มของข้อมูลด้วยค่าเฉลี่ยของกลุ่มข้อมูลนั้น การซับแซมปลิง สามารถทำได้ใน 2 ทิศทางคือในแนวตั้ง (vertical:V)และในแนวนอน (horizontal:H) เช่น การซับแซมปลิง ด้วยอัตราส่วน H:2,V:2 คือการจับกลุ่มข้อมูลในแนวนอน 2 ตัวและในแนวตั้ง 2 ตัวเป็นกลุ่มละ 4 ตัว โดยแต่ละกลุ่มจะถูกแทนที่ด้วยข้อมูล 1 ตัวซึ่งหามาจากค่าเฉลี่ยของข้อมูลในกลุ่มที่ถูกแทนที่นั้นเอง ดังแสดงในรูปที่ 2 ด้วยวิธีนี้จะทำให้จำนวนข้อมูลจะลดลงเหลือเพียงหนึ่งในสี่

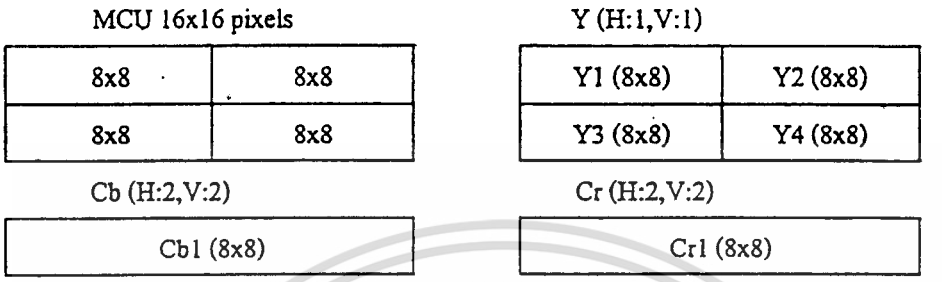
□ แทนข้อมูล 1 ตัว

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

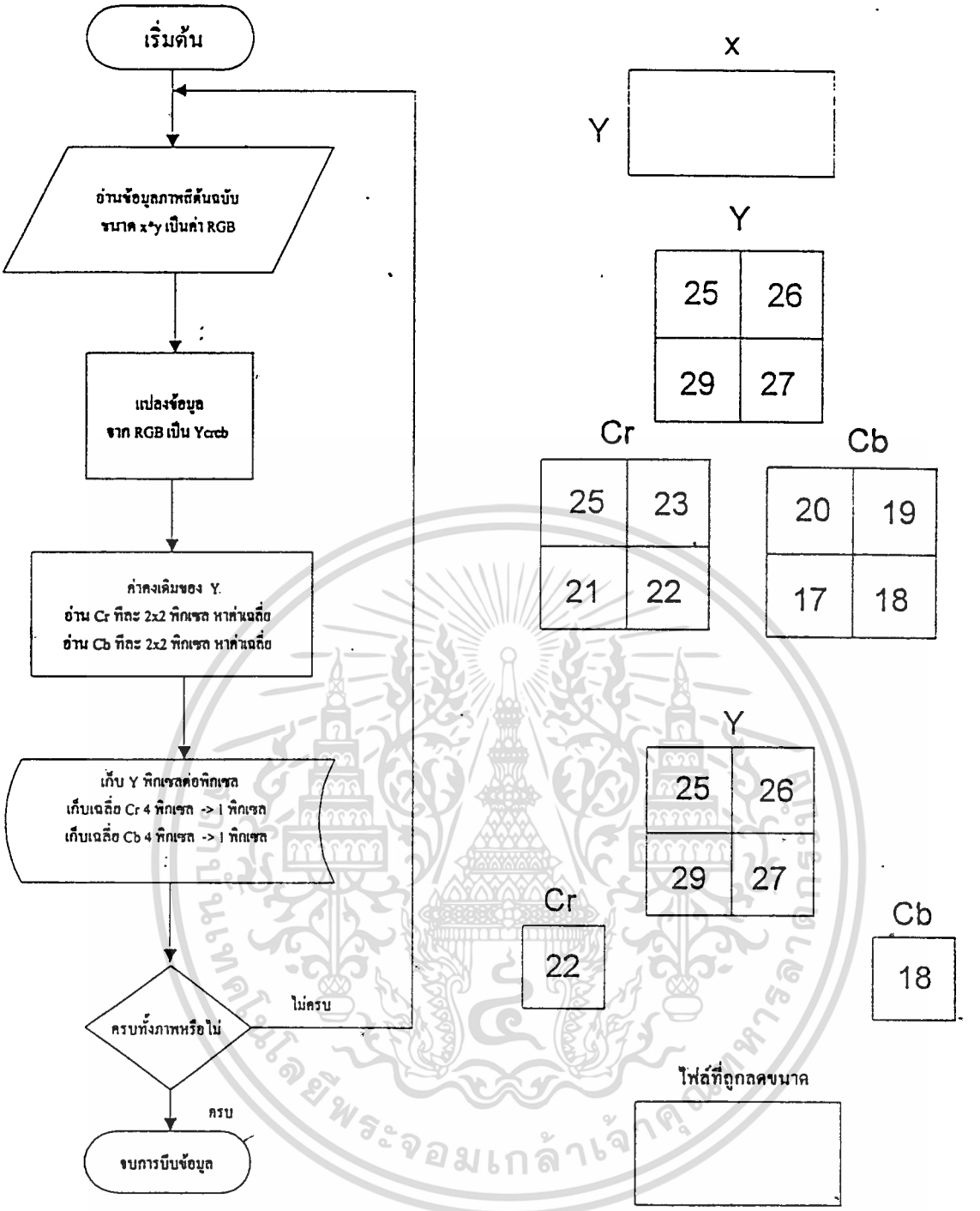
$\frac{(1+2+9+10)}{4}$	$\frac{(3+4+11+12)}{4}$	$\frac{(5+6+13+14)}{4}$	$\frac{(7+8+15+16)}{4}$
$\frac{(17+18+25+26)}{4}$	$\frac{(19+20+27+28)}{4}$	$\frac{(21+22+29+30)}{4}$	$\frac{(23+24+31+32)}{4}$

รูปที่ 3.2 ตัวอย่างการซับแซมปลิง ด้วยอัตราส่วน H:2,V:2

ด้วยวิธีดังกล่าวเจเปิ๊ก จะใช้อัตราส่วน H:1,V:2 หรือ H:2,V:1 หรือ H:2,V:2 ในการซับแซมปลิงค่า Cb และ Cr โดยคงจำนวนค่า Y ไว้ดังเดิม (H:1,V:1) ผู้เขียนได้เลือกใช้อัตราส่วนการซับแซมปลิง ค่า Y:Cb:Cr เท่ากับ 4:1:1 คือใช้อัตราส่วน H:2,V:2 ในการซับแซมปลิง ค่า Cb และ Cr ซึ่งมีผลทำให้หน่วย MCU มีขนาดเท่ากับ 16x16 พิกเซล และภายในแต่ละ MCU จะประกอบด้วยบล็อกข้อมูลของค่า Y จำนวน 4 บล็อก, Cb จำนวน 1 บล็อก และ Cr จำนวน 1 บล็อก ขนาดบล็อกละ 8x8 ค่าดังแสดงในรูปที่ 3.3



รูปที่ 3.3 แสดงองค์ประกอบภายใน MCU เมื่อใช้อัตราส่วน H:2,V:2 ในการซับแซมปลิงค่า Cb และ Cr และมีการเรียงลำดับขององค์ประกอบภายใน MCU คือ Y1,Y2,Y3,Y4,Cb1 และ Cr



รูปที่ 3.4 การบีบข้อมูลแบบโครมา (Chroma) 2 ทิศทาง 4 : 1 :1

3.2 การบีบอัดข้อมูลแบบเจเป็ก

ในการบีบอัดข้อมูลแบบเจเป็ก จะทำการบีดอัดข้อมูล MCU ทีละหน่วยโดยมีลำดับการบีบอัดเรียงจากด้านซ้ายไปด้านขวาและจากด้านบนไปด้านล่างของภาพ และการบีบอัดข้อมูล MCU แต่ละหน่วยจะทำทีละบล็อก (8x8 pixels) ขององค์ประกอบภายใน MCU หน่วยนั้น โดยเรียงลำดับตามลำดับการเรียงตัวของบล็อกองค์ประกอบภายใน MCU ตามที่แสดงไปแล้ว ซึ่งการบีบอัดข้อมูลในแต่ละบล็อกมีขั้นตอนดังนี้

3.2.1 การแปลง (Transformation) บล็อกข้อมูลด้วยวิธี DCT

ก่อนที่จะทำการแปลงบล็อกข้อมูลจะมีการลดระดับ DC ของบล็อกข้อมูลก่อนด้วยการลบข้อมูลทุกตัวภายในบล็อกด้วย 128 (ซึ่งมาจากค่าของ 2^{n-1} เมื่อ n คือจำนวนบิตที่ใช้เก็บข้อมูลสีแต่ละสีในจุดพิกเซลซึ่งในที่นี้จะเท่ากับ 8) จากนั้นจึงทำการแปลง บล็อกข้อมูลด้วย DCT ตามสมการที่ 2.5 ซึ่งในการแปลงแบบ DCT นี้จะทำการแปลงในลักษณะสองทิศทางเพราะรูปแบบของภาพนั้นเป็นฟังก์ชันสองทิศทาง

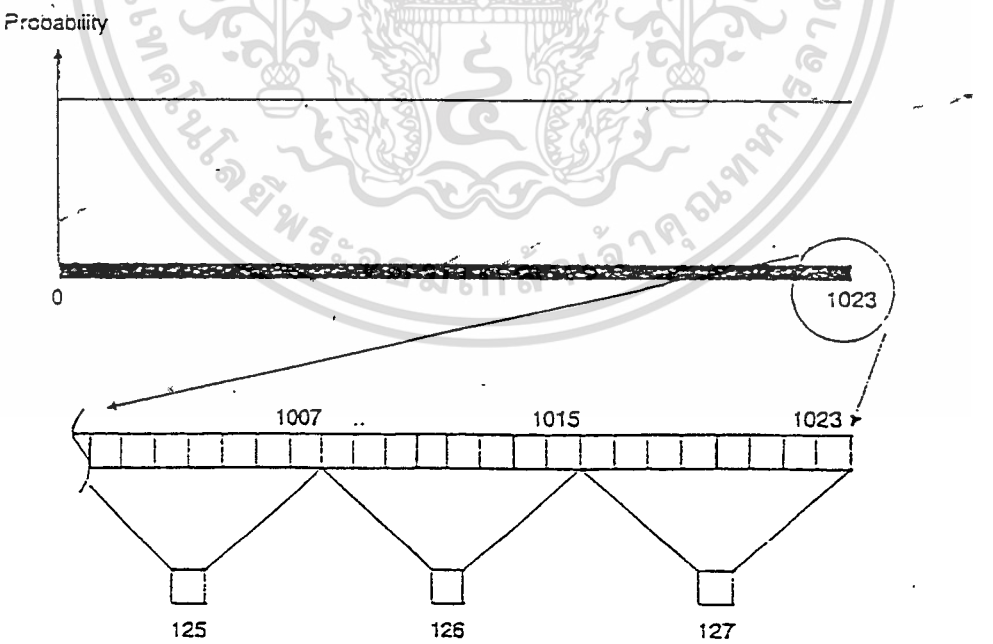
การแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มจะทำการแปลงข้อมูลในลักษณะของพิกเซลไปเป็นสัมประสิทธิ์ทางความถี่ โดยจะรวมข้อมูลของภาพทั้งหมดแล้วแทนด้วยค่าสัมประสิทธิ์ทางความถี่ ซึ่งผลจากการแปลงจะมีลักษณะเป็นจุดๆเช่นเดียวกับจุดพิกเซลก่อนที่จะทำการแปลง แต่จุดต่างๆนี้จะแทนค่าสัมประสิทธิ์ทางความถี่แต่ละค่า ดังนั้นผลที่ได้จากการแปลงคือสัมประสิทธิ์ทางความถี่ที่อยู่ในรูปของบล็อกขนาด 8x8 ค่า (8 แถว และ 8 หลัก) ค่าต่างๆในแถวที่ศูนย์จะมีส่วนประกอบทางความถี่เป็นศูนย์ในทิศทางหนึ่งและค่าทั้งหมดในคอลัมน์ที่ศูนย์ก็จะมีส่วนประกอบทางความถี่เป็นศูนย์ในอีกทิศทางหนึ่ง เมื่อแถวและคอลัมน์เพิ่มขึ้นในทิศทางที่ห่างออกจากจุดเริ่มต้นสัมประสิทธิ์ในเมตริกที่ได้จากการแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มจะแทนความถี่ที่สูงขึ้น และความถี่ที่สูงที่สุดคือที่ตำแหน่ง N-1 ของเมตริก สิ่งนี้มีความสำคัญมากเพราะภาพส่วนใหญ่นั้นข้อมูลของภาพจะรวมกันอยู่ในช่วงของความถี่ต่ำ และส่วนประกอบที่พบในจุดที่แถวและคอลัมน์เป็นศูนย์ (ส่วนประกอบ DC) จะเก็บข้อมูลที่มีประโยชน์เกี่ยวกับภาพมากกว่าส่วนประกอบที่ความถี่ที่สูงกว่านี้ ส่วนประกอบตัวที่อยู่ห่างจากส่วนประกอบ DC นั้นจะพบว่าไม่เพียงแค่ค่าสัมประสิทธิ์มีแนวโน้มที่จะต่ำลง แต่ยังมีความสำคัญต่อการอธิบายภาพน้อยลงอีกด้วย ดังนั้นการแปลงแบบดิสครีตโคซายน์จะทำให้สามารถตัดข้อมูลบางส่วนออกไปได้โดยไม่มีผลต่อคุณภาพของภาพมากนัก ซึ่งถ้าภาพไม่ได้ถูกทำการแปลงแต่ยังคงอยู่ในรูปของจุดพิกเซลจะไม่สามารถตัดข้อมูลบางส่วนของภาพออกไปได้เช่นนี้เพราะแต่ละพิกเซลมีผลต่อการมองเห็นของภาพโดยรวม

3.2.2 การควอนไตเซชัน

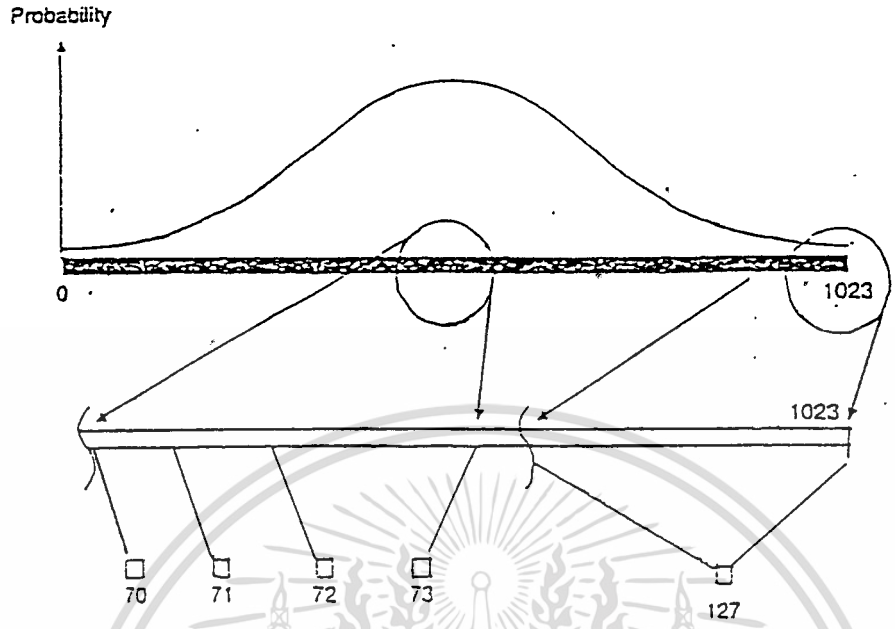
เวกเตอร์ควอนไตเซชัน หมายถึง การลดค่ารีโซลูชัน(resolution)ซึ่งใช้แทนค่าของชุดกลุ่มข้อมูล ซึ่งหลักการที่สำคัญก็คือการแทนกลุ่มค่าของชุดข้อมูลต้นฉบับเป็นค่าข้อมูลค่าเดี่ยวชุดใหม่ ซึ่งจะลดปริมาณข้อมูลลง ซึ่งตามความเป็นจริงแล้วกรรมวิธีในการจัดกลุ่มข้อมูลของชุดข้อมูลต้นฉบับเป็นกุญแจสำคัญในการลดปริมาณข้อมูลสูญหาย เวกเตอร์ควอนไตเซชันโดยคำจำกัดความแล้วจะเป็นเทคนิคการบีบอัดข้อมูลที่ยอมให้มีการสูญเสีย นั่นคือข้อมูลต้นฉบับบางค่าจะมีการสูญหาย ซึ่งถ้าจะอธิบายอย่างง่ายๆ ได้ดังนี้ เวกเตอร์ควอนไตเซชัน คือ กระบวนการที่นำค่าข้อมูลจากข้อมูลต้นฉบับมาจัดเป็นกลุ่มข้อมูลจากนั้นแทนค่าแต่ละกลุ่ม

ด้วยค่าเวกเตอร์ค่าหนึ่ง กระบวนการควอนไตเซชันเกิดขึ้นเมื่อมีการสร้างค่าข้อมูลชุดใหม่ขึ้นมาเพราะว่าค่าของข้อมูลทุกค่าที่แทนข้อมูลแต่ละกลุ่มนั้นไม่สามารถแยกแยะความแตกต่างของข้อมูลแต่ละค่าได้เพราะค่าของข้อมูลในกลุ่มนั้นจะถูกแทนที่ด้วยค่าค่าใหม่ค่าหนึ่ง ซึ่งจะมีค่าโดยประมาณใกล้เคียงกับค่าข้อมูลในกลุ่มทั้งหมด กระบวนการคล้ายคลึงกับการจัดระดับปัดค่าข้อมูล 1.1,1.2,1.3 ให้เป็นค่าจำนวนเต็มที่ใกล้เคียงที่สุด หลังจากผ่านกระบวนการจัดระดับแล้วค่าทั้ง 3 จะไม่สามารถแยกแยะแสดงความแตกต่างจากกันได้ การจัดข้อมูลเป็นกลุ่มขึ้นอยู่กับความเกี่ยวข้องสัมพันธ์กันระหว่างค่าข้อมูลและการกระจายค่าข้อมูลในทางสถิติ

ยกตัวอย่างง่ายๆ ก็คือ สมมุติว่าเรามีชุดข้อมูลซึ่งเป็นจำนวนเต็มซึ่งมีค่าอยู่ในช่วง 0 ถึง 1023 และสมมุติว่าเราจะแทนค่าชุดข้อมูลชุดนี้ด้วยค่าเพียง 128 ค่าเท่านั้น ค่าข้อมูลในช่วง 0 ถึง 1023 แทนเลขจำนวนเต็มขนาด 10 บิท และค่าข้อมูล 0 ถึง 127 แทนเลขจำนวนเต็มขนาด 7 บิท ทำให้ลดจำนวนข้อมูลลงได้ 3บิท ต่ออิลิเมนต์ ซึ่งการจัดกลุ่มแบบนี้จะกระทำแบบยูนิฟอร์มคือจะกระทำการลดค่ารีโซลูชันโดยการแทนค่าชุดกลุ่มข้อมูลอย่างต่อเนื่อง โดยจะแบ่งดีกรีในช่วงเท่าๆกัน ถ้าค่าความน่าจะเป็น(probabilities)ของข้อมูลเท่ากันแล้ววิธีนี้เป็นวิธีที่ดีที่สุดในการควอนไตเซชัน แต่ถ้าชุดข้อมูลมีค่าความน่าจะเป็นที่จะเกิดหรือมีความสำคัญไม่เท่ากันตลอดช่วงแล้ว ดีกรีในการควอนไตเซชันจะไม่เท่ากันโดยข้อมูลที่มีค่าความน่าจะเป็นสูงก็จะกระทำการจัดระดับแบบละเอียด แต่ค่าข้อมูลที่มีค่าความน่าจะเป็นต่ำก็จะกระทำการจัดระดับแบบหยาบโดยจะสามารถ ลดค่าความสำคัญของข้อมูลลงได้มากๆ



รูปที่ 3.5 การกระจายค่าความน่าจะเป็นแบบเท่ากันและเกี่ยวข้องกับการควอนไตซ์แบบเท่ากันทุกค่า



รูปที่ 3.6 การกระจายค่าความน่าจะเป็นแบบปกติและเกี่ยวข้องกับการควอนไตซ์แบบไม่เท่ากันทุกค่า

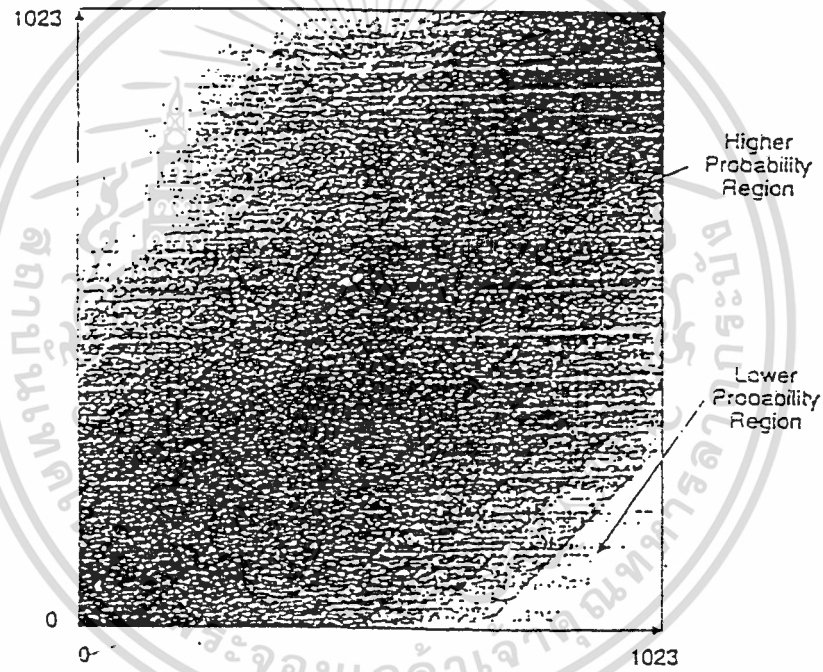
รูปที่ 3.5 แสดง ฮิสโตแกรมสำหรับค่าข้อมูลในช่วง 0 ถึง 1023 ซึ่งทุกค่าข้อมูลมีค่าความน่าจะเป็นเท่ากัน และยังแสดงการควอนไตเซชันแบบง่ายโดยการจัดข้อมูลชนิด 10 บิทเป็น 7 บิทด้วย รูปที่ 3.6 แสดงการควอนไตเซชันที่ค่าความน่าจะเป็นของชุดข้อมูลไม่เท่ากัน ในกรณีนี้เราจะแทนค่ากลุ่มข้อมูลที่มีค่าความน่าจะเป็นสูง หรือเกิดบ่อยๆ โดยจัดระดับแทนค่าให้มีความแม่นยำสูง ส่วนค่าข้อมูลที่มีค่าความน่าจะเป็นต่ำหรือแทบจะไม่เกิดแทนด้วยค่าที่ไม่ต้องมีความแม่นยำมากนัก

จะสังเกตเห็นได้ว่าการควอนไตเซชันจะกระทำโดยละเอียดในบริเวณที่มีค่าความน่าจะเป็นสูงๆ และจะหยาบขึ้นในบริเวณที่มีค่าความน่าจะเป็นต่ำกว่า ในกรณีนี้ไม่มีสมการทางคณิตศาสตร์ที่สามารถแสดงความสัมพันธ์ระหว่างข้อมูลต้นฉบับเดิมกับชุดข้อมูลที่ได้จากการควอนไตเซชันโดยชัดเจนได้ จากชุดข้อมูลเดิมแต่ละกลุ่มที่มีความสัมพันธ์กับค่าเดี่ยวๆในชุดค่าข้อมูลที่ผ่านการควอนไตซ์ เป็นเวกเตอร์ค่า ๆ หนึ่ง ตารางที่แสดงความสัมพันธ์ระหว่างค่าในการควอนไตซ์และค่าโดยประมาณที่ใกล้เคียงที่สุดกับค่าข้อมูลเดิมคือ หนังสือรหัส(code book) ซึ่งจะช่วยให้เราสามารถหาวิธีที่ดีที่สุดในการจัดกลุ่มข้อมูลจากข้อมูลเดิมเป็นเวกเตอร์ ซึ่งการเพิ่มประสิทธิภาพของเทคนิคในการจัดระดับ เวกเตอร์จะเป็นเวกเตอร์หลายมิติ ซึ่งจะนำไปสู่หลักการของความสัมพันธ์ระหว่างอิลิเมนต์ในชุดข้อมูล

ขยายความต่อจากตัวอย่างที่แล้ว สมมุติว่าค่าข้อมูลจากชุดข้อมูลเดิมถูกนำมาคิดเป็นคู่ๆ ดังนั้นแทนที่จะมีค่าที่เป็นไปได้เพียงแค่ 1024 ค่า จะทำให้ตอนนี้ค่าที่เป็นไปได้จะเพิ่มขึ้นมากกว่า 1 ล้านส่วนผสม ซึ่งเป็นตัวอย่างที่แสดงให้เห็นถึงการควอนไตซ์ของเวกเตอร์ 2 มิติ ซึ่งแม้ว่าจะไม่ได้มีทฤษฎีที่แน่นอนว่าควรจะเป็นการควอนไตซ์เวกเตอร์กี่มิติจึงจะดีแต่ในทางปฏิบัติแล้วก็จะมีข้อจำกัดอยู่บ้าง ยกตัวอย่างเช่น ในระบบ 3 มิติใช้ค่า 8

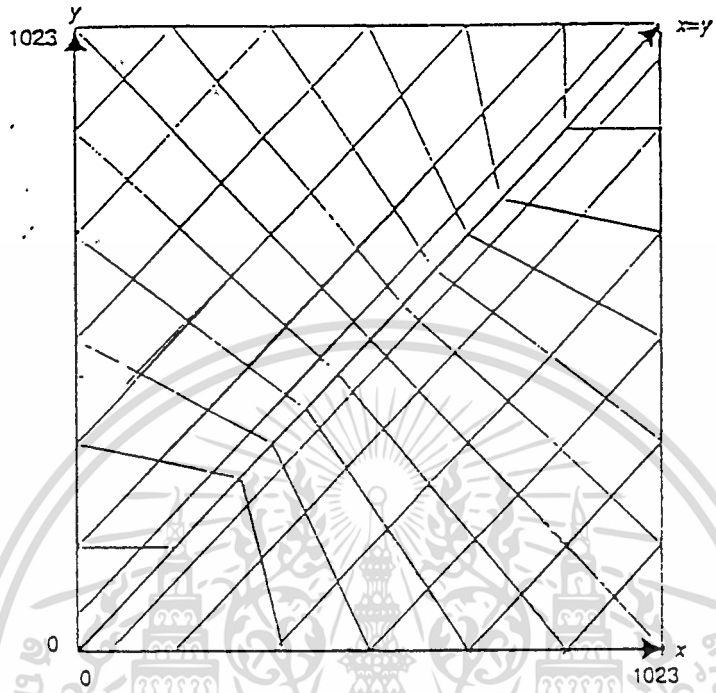
บิทจะทำให้ค่าที่เป็นไปได้อยู่ในช่วง 2^{8+8+8} หรือประมาณ 16 ล้าน คือทุกครั้งที่มีการเพิ่มมิติขึ้นมาความซับซ้อนของการวิเคราะห์ทางสถิติก็จะเพิ่มขึ้นเป็นเอ็กซ์โปเนนเชียล

เพื่อแสดงให้เห็นตัวอย่างของกรณีของระบบ 2 มิติ ลองพิจารณาตัวอย่างที่เกี่ยวข้องกับสตรีมของแซมเปิลดิจิตอลที่นำมาจากสัญญาณออดิโอเวฟฟอร์ม ดีกรีของความสัมพันธ์ระหว่างแซมเปิลใดๆไปยังแซมเปิลถัดไปจะเป็นฟังก์ชันของความถี่ในการแซมปลิง ผลต่างที่มากที่สุดจากแซมเปิลหนึ่งๆไปยังแซมเปิลถัดไปจะเกิดขึ้นภายในช่วงระหว่างคาบของความถี่สูงสุดภายในสเปคตรัมของเสียงออดิโอเท่านั้น ขณะที่ความถี่ของอินพุทลดลง ผลต่างระหว่างแซมเปิลที่อยู่ติดกันก็มีแนวโน้มที่จะลดลงด้วย ซึ่งแสดงว่าค่าความน่าจะเป็นของคู่แซมเปิลที่มีค่าเดลต้า(delta) น้อยๆจะสูงกว่าค่าของคู่แซมเปิลที่มีค่าเดลต้าผลต่างกว้างๆ ถ้าส่วนผสมที่เป็นไปได้ทั้งหมดของคู่แซมเปิลถูกพล็อตบนชาร์ต 2 มิติ โดยช่วงค่าของแซมเปิลในมิติหนึ่งอยู่บนแกน X และช่วงค่าของแซมเปิลอีกมิติหนึ่งอยู่บนแกน Y ซึ่งค่าความน่าจะเป็นมีแนวโน้มที่สูงในบริเวณที่ X=Y ดังรูปที่ 3.7



รูปที่ 3.7 การกระจายของค่าความน่าจะเป็นแบบ 2 มิติเนื่องมาจากความสัมพันธ์ของแซมเปิล

โดยการแบ่งสเปซ 2 มิตินี้เป็นบริเวณ ซึ่งในแต่ละบริเวณจะมีมากกว่า 1 คู่ และสมมติสัญลักษณ์แทนค่าสำหรับแต่ละบริเวณ จำนวนสัญลักษณ์ทั้งหมดจะสามารถลดจาก 1 ล้านเป็นจำนวนสัญลักษณ์ที่น้อยกว่าได้ซึ่งสิ่งที่ควรรู้ก็คือ ว่าบริเวณเหนือเส้นไดอะโกนอล (Diagonal) แสดงว่า Y>X และบริเวณที่อยู่ใต้เส้นไดอะโกนอลแสดงว่า X>Y อย่างไรก็ตามเมื่อสตรีมข้อมูลได้รับการสร้างใหม่แต่ละเวกเตอร์จะสร้างคู่ของแซมเปิลที่เท่ากัน ซึ่งจะทำให้เกิดค่าแซมปลิงเออร์เรอร์(Sampling Error) ที่มีค่ามากในเอาท์พุทสตรีม รูปที่ 3.8 แสดงว่าแซมเปิลสเปซ 2 มิติจะแสดงค่าความน่าจะเป็นที่สูงกว่าของค่าผลต่างเพียงเล็กน้อยของแต่ละแซมเปิลได้อย่างไร



รูปที่ 3.8 การจัดแซมเปิลสเปซ 2 มิติสำหรับเวกเตอร์ควอนไตเซชัน

ซึ่งตัวอย่างที่แสดงให้ดูในที่นี้เป็นเป็นชุดของแซมเปิลสัญญาณออดิโอแต่จากหลักการเดียวกันนี้สามารถนำไปประยุกต์ใช้กับสัญญาณภาพได้เหมือนกัน เนื่องจากในช่วงความถี่ต่ำจะประกอบด้วยค่าที่เป็นค่าเฉลี่ยของข้อมูลซึ่งเป็นค่าที่มีความแม่นยำสูง เช่นถ้ากลุ่มของข้อมูลแบบ 4 พิกเซลที่ถูกนำมาควอนไตซ์ด้วยกันจะแทนชุดของการควอนไตซ์ แบบเวกเตอร์ 4 มิติ ซึ่งควรระวังที่จะสับสนระหว่างมิติของเวกเตอร์กับมิติของแซมเปิลสเปซ เราจะเห็นว่าสตรีมของสัญญาณออดิโอซึ่งเป็นแซมเปิลสเปซ 1 มิติ จะสามารถควอนไตซ์ได้โดยการใช้เวกเตอร์ 2 มิติ โดยการใช้ค่าแซมเปิลเป็นคู่ๆ การควอนไตซ์แซมเปิลสเปซ 2 มิติเช่นภาพ (natural image) ไม่สามารถบอกอะไรเกี่ยวกับมิติของเวกเตอร์ควอนไตเซชันได้ ซึ่งสิ่งที่จะบอกจำนวนมิติของเวกเตอร์ควอนไตเซชัน คือจำนวนแซมเปิลที่ไม่ต่อเนื่องซึ่งประกอบกันเป็นเวกเตอร์ ซึ่งไม่เกี่ยวข้องกับการจัดการทางลอจิกของแซมเปิลสเปซดั้งเดิม หรืออีกนัยหนึ่งก็คือจะมีผลกระทบกับคาแรคเตอริสติก (characteristics) ในทางสถิติซึ่งจะมีผลกระทบกับรูปแบบที่กลุ่มแซมเปิลจะถูกควอนไตซ์

ซึ่งการควอนไตเซชันในที่นี้เป็นเวกเตอร์ควอนไตเซชัน ลดค่าของข้อมูลภายในบล็อกข้อมูล DCT เพื่อจะลดจำนวนบิตที่ใช้ในการเก็บข้อมูลเหล่านี้ในขั้นตอนเข้ารหัส ต่อไป และจากการที่ภาพส่วนใหญ่มีการกระจายตัวของข้อมูลสีอยู่ในช่วงความถี่ต่ำ ๆ เจเปิค จึงให้ความสำคัญต่อข้อมูลในช่วงความถี่ต่ำมากกว่าข้อมูล

ในช่วงความถี่สูง ดังนั้นการ ควอนไตเซชัน คือ ความพยายามลดความสำคัญของข้อมูลในช่วงความถี่สูงโดยพยายามทำให้กลายเป็นค่าศูนย์ให้มากที่สุดนั่นเองโดยใช้บล็อกข้อมูลของควอนไตเซอร์ 8x8 ค่า หารบล็อกข้อมูลจาก DCT 8x8 ค่าดังสมการต่อไปนี้

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Q(u,v)}\right] \tag{3.4}$$

และมีสมการดิควอนไตเซชัน ดังนี้

$$\dot{T}(u,v) = \hat{T}(u,v)Q(u,v) \tag{3.5}$$

เมื่อ T(u,v) คือ ค่าสัมประสิทธิ์ DCT

$\hat{T}(u,v)$ คือ ค่าสัมประสิทธิ์ DCT เมื่อถูกควอนไตซ์

$\dot{T}(u,v)$ คือ ค่าสัมประสิทธิ์ DCT เมื่อดิควอนไตซ์

Q(u,v) คือ ค่าควอนไตเซอร์

และ *round* คือ การหาจำนวนเต็มที่มีค่าใกล้เคียงที่สุด

โดย $0 \le u \le 7$, $0 \le v \le 7$

ซึ่งผู้เขียนได้ออกแบบบล็อกข้อมูลควอนไตเซอร์ ที่ใช้ในโปรแกรมเป็นดังนี้

$$Q(8,8) = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 6 \\ 4 & 4 & 4 & 4 & 4 & 4 & 6 & 6 \\ 4 & 4 & 4 & 4 & 4 & 6 & 6 & 6 \\ 4 & 4 & 4 & 4 & 6 & 6 & 6 & 8 \\ 4 & 4 & 4 & 6 & 6 & 6 & 8 & 8 \\ 4 & 4 & 6 & 6 & 6 & 8 & 8 & 10 \\ 4 & 6 & 6 & 6 & 8 & 8 & 10 & 12 \end{bmatrix}$$

และได้ออกแบบให้โปรแกรมสามารถเปลี่ยนค่า Q(u,v) ภายในบล็อกได้หลายระดับ ด้วยการนำค่าของควอนไตเซอร์แฟคเตอร์ (quantizer_factor) (ตัวแปรที่ใช้ในโปรแกรมมีค่าระหว่าง 0 ถึง 100) มาเพิ่มให้กับควอนไตเซอร์ทั้ง 64 ตัวภายในบล็อก ซึ่งค่าของ Q.F.(quantizer_factor) นี้สามารถกำหนดผ่านทางคอมมานด์ไลน์ (command line)ได้เมื่อเรียกใช้งานโปรแกรม ดังนั้นสมการควอนไตเซชันที่ใช้ในโปรแกรมจะเป็นดังนี้

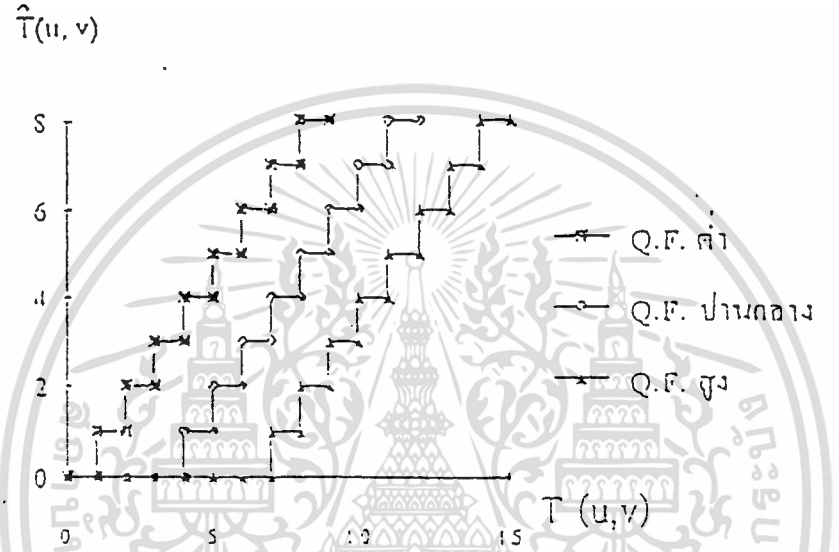
$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Q(u,v) + Q.F.}\right] \tag{3.6}$$

และมีสมการดิควอนไตเซชัน คือ

$$\dot{T}(u,v) = \hat{T}(u,v)\left[Q(u,v) + Q.F.\right] \qquad (3.7)$$

เมื่อ Q.F. คือ ค่าควอนไตเซอร์ แฟคเตอร์ (quantizer_factor)

และจากสมการ (3.6) สามารถเขียนเป็นกราฟความสัมพันธ์ระหว่างค่า T(u,v) กับ $\hat{T}(u,v)$ เมื่อเปลี่ยนค่า Q.F. ได้อย่างคร่าว ๆ ดังรูปที่ 3.4

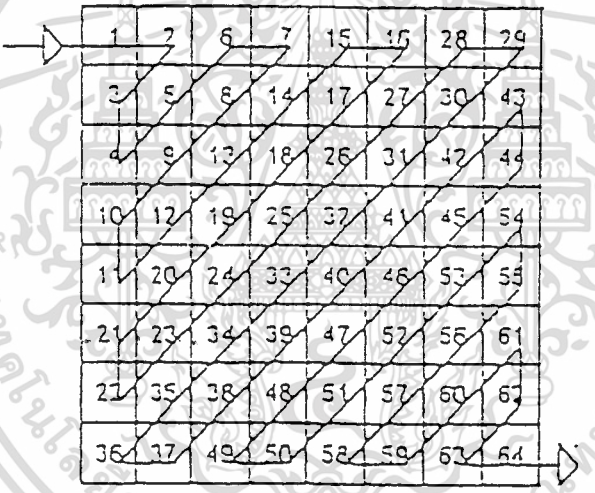


รูปที่ 3.9 กราฟแสดงความสัมพันธ์ระหว่าง T(u,v) กับ $\hat{T}(u,v)$ ที่มีค่า Q.F. ต่าง ๆ

จากกราฟจะเห็นว่าการเพิ่มค่า Q.F. จะมีผลทำให้มีจำนวนค่า $\hat{T}(u,v)$ ที่เป็นศูนย์เพิ่มมากขึ้น ซึ่งจำนวนของค่า $\hat{T}(u,v)$ ที่เป็นศูนย์ที่เพิ่มขึ้นนี้จะเป็นประโยชน์ต่อการบีบอัดข้อมูล เนื่องจากในขั้นตอนการเข้ารหัสค่า AC แบบฮัฟแม่น (Huffman) นั้นจะทำการเข้ารหัสข้อมูลเฉพาะค่าที่ไม่เป็นศูนย์ โดยข้อมูลที่เป็นศูนย์นั้นจะมีเทคนิคการเข้ารหัสรันเลนจ์ (runlength encoding) มาช่วยในการเข้ารหัส ดังนั้นถ้าจำนวนข้อมูลที่เป็นศูนย์มีมากขึ้นจะทำให้จำนวนข้อมูลที่ถูกเข้ารหัส (คือข้อมูลที่ไม่เป็นศูนย์) น้อยลง ซึ่งจะส่งผลให้จำนวนรหัส ที่ได้มีจำนวนน้อยลงด้วย และทำให้ได้อัตราการบีบอัดข้อมูลที่สูงขึ้น แต่ข้อมูล $\hat{T}(u,v)$ ที่ถูกควอนไตซ์เป็นศูนย์จะมีผลเสีย คือ เมื่อนำมาดิควอนไตซ์ตามสมการที่ (3.7) จะได้ค่าที่เป็นศูนย์คืนกลับมา ซึ่งมีความเป็นไปได้สูงที่มีค่าผิดพลาดจากค่าเดิมเป็นอย่างมาก อันจะมีผลทำให้คุณภาพของภาพที่ได้คืนมาแย่ลงมาก

3.2.3 การอ่านข้อมูลแบบซิกแซก

การอ่านข้อมูลแบบซิกแซก เป็นการอ่านข้อมูลที่ได้จากการควอนไตเซชันซึ่งอยู่ในรูปแบบของบล็อกขนาด 8 แถว 8 คอลัมน์ ให้อยู่ในรูปของชุดข้อมูลที่เรียงต่อกันไป โดยลักษณะของการอ่านข้อมูลแบบซิกแซกนี้มีลักษณะดังรูปที่ 3.10 ซึ่งการอ่านข้อมูลในลักษณะนี้ก็เพื่อให้สอดคล้องกับข้อมูลที่ได้จากการแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มและการควอนไตเซชันเพราะในการแปลงแบบดิสครีตโคซายน์ทรานสฟอร์มนั้นผลที่ได้จากการแปลงจะมีการเรียงค่าส่วนประกอบทางความถี่เพิ่มขึ้นตามแนวเส้นทแยงมุมของบล็อกขนาด 8 แถว 8 หลัก ดังได้กล่าวไว้แล้วในเรื่องการแปลงแบบดิสครีตโคซายน์ทรานสฟอร์ม และผลที่ได้จากการแปลงนี้เมื่อนำมาทำการควอนไตเซชันค่าของข้อมูลก็จะลดลงตามแนวเส้นทแยงมุมของบล็อกเช่นเดียวกันและในช่วงความถี่สูงๆค่าจะถูกลดลงเป็นศูนย์มาก ดังนั้นการอ่านข้อมูลแบบซิกแซกก็มีความพยายามที่จะนำค่าเป็นค่าศูนย์มาเรียงให้ติดกันเพื่อประโยชน์ในการบีบอัดข้อมูลในขั้นตอนการเข้ารหัสต่อไป



รูปที่3.10 แสดงลำดับการอ่านข้อมูลแบบซิกแซก

3.2.4 การเข้ารหัสข้อมูล

กระบวนการสุดท้ายของเจเปิก คือการเข้ารหัสข้อมูลที่ผ่านการควอนไตซ์แล้ว ด้วยกระบวนการ 3 ขั้นตอน

1. การเก็บค่าสัมประสิทธิ์กระแสตรงของบล็อกพิกเซล [8x8] บล็อกแรกก่อน แล้วทำการเปลี่ยนค่าสัมประสิทธิ์กระแสตรงที่จุด (0,0) ของบล็อกต่อมาเป็นค่าที่สัมพัทธ์กับค่าแรกหรือเป็นค่าผลต่างนั่นเอง แล้วจึงทำการเข้ารหัสข้อมูลผลต่างนี้แทน เพื่อประโยชน์คือสามารถเข้ารหัสโดยใช้ค่าผลต่างที่ถือว่าน้อยด้วยจำนวนบิทที่น้อยกว่าการเข้ารหัสค่าจริง (เพราะถือว่าบล็อกของพิกเซลที่อยู่ใกล้กันจะมีส่วนที่แตกต่างกันน้อยมากและสัมประสิทธิ์กระแสตรงก็ได้รวมข้อมูลที่สำคัญของภาพไว้แล้ว)

2. การจัดลำดับสัมประสิทธิ์กระแสสลับเป็นลำดับแบบซิกแซกเพื่อให้ค่าศูนย์ ที่เกิดขึ้นจากกระบวนการควอนไตเซชันมีความต่อเนื่องกันยาวๆ ทำให้สามารถเข้ารหัสได้ง่าย

3. เลือกทำการเข้ารหัสสัมประสิทธิ์กระแสสลับที่จัดลำดับแล้ว

3.1 ด้วยวิธีความต่อเนื่องของข้อมูล RLE (Run Length Encoding) ซึ่งข้อดีของวิธีนี้คือถ้าค่าศูนย์ ที่เรียงติดกันมีความต่อเนื่องกันยาวๆ มาก จะทำให้สามารถเข้ารหัสได้ง่าย สั้นลง

3.2 การเข้ารหัสแบบเอนโทรปี (Entropy Encoding) ซึ่งอาจจะทำได้โดยวิธีฮัฟแมน หรือการประมวลผลทางคณิตศาสตร์(Arithmatic) ก็ได้ ซึ่งในที่นี้จะใช้การเข้ารหัสแบบฮัฟแมน

การเข้ารหัสข้อมูลแบบฮัฟแมน

การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นเป็นการเข้ารหัสเฉพาะข้อมูลที่มีค่าไม่เป็นศูนย์ ดังนั้นจากการพยายามลดค่าของข้อมูลให้เป็นศูนย์มากๆในขั้นตอนของการควอนไตเซชันและการอ่านข้อมูลแบบซิกแซกที่พยายามทำให้ค่าที่เป็นศูนย์มาอยู่เรียงกันนั้นจึงมีประโยชน์ต่อการเข้ารหัสข้อมูลแบบฮัฟแมนมาก เพราะจะทำให้ข้อมูลที่ต้องทำการเข้ารหัสมีจำนวนน้อยลง การเข้ารหัสข้อมูลแบบฮัฟแมนนั้นจะแบ่งการเข้ารหัสข้อมูลเป็นสองส่วนคือการเข้ารหัสข้อมูลของค่า DC และการเข้ารหัสข้อมูลของค่า AC

การเข้ารหัสค่า DC ของแถวข้อมูล

ในการเข้ารหัสค่า DC ของแถวข้อมูลนั้น (ข้อมูลตัวแรกในแถวซึ่งคิดค่าสเปกตรัมสัมประสิทธิ์กระแสตรงที่จุด (0,0) ของบล็อกพิกเซลขนาด 8X8) จะถูกเข้ารหัสเฉพาะค่าความแตกต่างระหว่างค่า DC ของแถวข้อมูลปัจจุบัน กับค่า DC ของแถวข้อมูลซึ่งเป็นสีชนิดเดียวกัน ที่ถูกเข้ารหัสแถวล่าสุดที่ผ่านไป เช่น เมื่อกำลังทำการเข้ารหัส ค่า DC ของแถวข้อมูล C_6 อยู่ซึ่งมีค่าเท่ากับ 40 จะทำการหาค่าความแตกต่างกัน ค่า DC ของแถวข้อมูล C_0 ที่ถูกเข้ารหัสผ่านไปแล้วแถวล่าสุด (สมมติว่ามีค่าเท่ากับ 15) ดังนั้นค่าที่จะถูกนำมาเข้ารหัสคิด 25 (40-15) ซึ่งการเข้ารหัสข้อมูลค่า DC จะมีการเข้ารหัสสองส่วนคือ

1. SIZE หมายถึง จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูล (ผลต่าง) ซึ่งจะหาจากตารางฮัฟแมน

2. AMPLITUDE หมายถึง ค่าขนาดของผลต่าง

ดังนั้นรหัสที่เข้า คือ (SIZE) (AMPILTUDE)

ในตารางฮัฟแมนจะเก็บรหัสความยาวต่างๆ ที่ใช้ในการแทนข้อมูลตามค่า Category ของข้อมูลนั้น ซึ่งการหาค่า Category ของข้อมูลสามารถหาได้จากตาราง JPEG Coefficient Coding Categories ซึ่งแสดงไว้ในตารางที่ 3.1 เมื่อได้ค่าCategory แล้วก็สามารถหารหัสข้อมูลได้จากตารางค่า DC ของฮัฟแมน(ในภาคผนวก)โดยการเทียบหาตามค่า Category ที่ได้จากตาราง 3.1

Range	DC Difference Category	AC Category
0	0	N/A
-1,1	1	1
-3,-2,2,3	2	2
-7,..,-4,4,..,7	3	3
-15,..,-8,8,..,15	4	4
-31,..,-16.16,..,313	5	5
-63,..,-32,..,63	6	6
-127,..,-64,..,127	7	7
-255,..,-128,128,..,255	8	8
-511,..,-256,511,..,256	9	9
-1023,..,-512,512-,..,1023	A	A
-2047,..,-1024.1027,..,2047	B	B
-4095,..,-2048,2048,..,4095	C	C
-8191,..,-4096,4096,..,-8191	D	D
-16382,..,-8192,8192,..,16383	E	E
-32767,..,-16384,19384,..,32767	F	F

ตารางที่ 3.1 ตารางแสดงค่า Categories ของข้อมูล

จะเห็นว่ารหัสที่ได้จากตารางที่ 3.1 นั้นใช้เป็นตัวบอกช่วงของข้อมูลที่ถูกเข้ารหัสเท่านั้นซึ่งจะได้รหัสตัวที่หนึ่งของข้อมูลออกมา ส่วนการระบุค่าของข้อมูลนั้นจำเป็นจะใช้รหัสตัวที่สองเป็นตัวระบุ โดยรหัสตัวที่สองนี้จะหามาจากค่าของข้อมูลโดยตรงโดยจะมีจำนวนบิตเท่ากับค่า Category ของข้อมูลที่นำมาเข้ารหัส นั่นเอง และจะเอาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

การเข้ารหัสค่า AC ของแถวข้อมูล

เนื่องจากการจัดเรียงข้อมูลแบบซิกแซก นั้น จะทำให้ค่าศูนย์ที่เกิดขึ้นจากการควอนไตซ์อยู่เรียงกันอย่างต่อเนื่องยาว ๆ ทำให้สามารถเข้ารหัสได้ง่าย และยิ่งมีศูนย์ที่ต่อเนื่องกันมากก็จะสามารถเข้ารหัสด้วยจำนวนที่น้อยลงได้มาก และโดยมากค่าศูนย์จะต่อเนื่องกันอยู่ในส่วนท้าย ๆ ของแถวข้อมูล

การเข้ารหัสค่า AC (ข้อมูลตั้งแต่ตัวที่ 2 ถึงตัวที่64 ในแถวข้อมูล) จะต่างจากในการเข้ารหัสค่า DC คือ จะทำการเข้ารหัสค่าของข้อมูลโดยตรง (ไม่เข้ารหัสเฉพาะค่าความแตกต่างเหมือนใน DC) และการเข้ารหัสค่า AC นั้นจะทำการเข้ารหัสเฉพาะค่า AC ที่มีค่าไม่เป็นศูนย์เท่านั้น ส่วนค่า AC ที่เป็นศูนย์นั้น JPEG จะอาศัยการเข้ารหัสแบบ run-length มาช่วยในการเข้ารหัสดังนี้คือในการเข้ารหัสค่า AC ที่ไม่เป็นศูนย์แต่ละตัวจะเน้นจำนวนข้อมูล AC ที่เป็นศูนย์ซึ่งอยู่ติดกันด้านหน้าของข้อมูลที่จะเข้ารหัสนั้นโดยถือเป็นค่า Run ของข้อมูลในการเข้ารหัส เช่น ค่า AC ภายในแถวเป็น 5 8 0 0 9 7 0 0 0 0 4 _ _ _ จะทำการเข้ารหัสเฉพาะข้อมูลที่ไม่เป็นศูนย์และมีค่า Run ของข้อมูลแต่ละตัวดังนี้คือ 5 (Run = 0) , 8 (Run = 0) , 9 (Run - 2) , 7 (Run = 0), 4 (Run = 4) ..... โดยมีการเข้ารหัสข้อมูลดังกล่าวคล้ายใน DC แต่จะใช้รหัส 3 ตัว แทนข้อมูลแต่ละตัวดังนี้

1.RUNLENGTH คือ จำนวนค่า 0 ที่ต่อเนื่องกันก่อนหน้าข้อมูลตัวที่จะเข้ารหัส

2. SIZE คือ จำนวนบิทที่จะต้องใช้ในการใส่ค่าข้อมูลที่ไม่เท่ากับ 0 ซึ่งหาจากตารางฮัฟแมน

3. AMPLITUDE คือ ค่าแอมปลิจูดหรือค่าของข้อมูลที่ไม่เท่ากับ 0 นั่นเอง

ดังนั้นรหัสคือ (RUNLENGTH,SIZE) (AMPLITUDE)

<u>หมายเหตุ</u>

มีเงื่อนไขพิเศษ คือ ถ้ามี 0 ต่อเนื่องกันมากกว่า 16 ตัว สมมุติว่าเป็น 22 ตัว จะได้รหัสเป็น

(15,0) (6,4) (13)

(15,0) ความหมายคือ มี 0 ต่อเนื่องกัน 16 ตัว (เป็นรูปแบบที่กำหนดตายตัว)

(6,4)(13) ความหมายคือ มี 0 อีก 6 ตัว

ใช้ 4 บิทในการแทนค่าข้อมูลที่มีค่าเท่ากับ 13

ซึ่งตารางที่ใช้ในการดูค่าจำนวนบิทที่ใช้ต่างๆกัน เรียกว่าเป็นรหัสแบบความยาวของรหัสไม่คงที่หรือแวริเอเบิลเลนจ์โคด (Variable Length Code) เนื่องจากข้อมูลที่นำมาเข้ารหัสจะมีค่า Run เข้ามาเกี่ยวข้องดังนั้นในตารางค่า AC ของฮัฟแมน จะต่างจากในตารางค่า DC ของฮัฟแมน คือจะมีการเปลี่ยนแปลงจาก Category ใน DC เป็น Run/Category นั่นคือการหารหัสข้อมูลจากตารางค่า AC ของฮัฟแมน จะต้องทราบค่า Run และ Category ของข้อมูลที่นำมาเข้ารหัส ซึ่งการหาค่า Category ของข้อมูล AC จะเทียบหาจากตาราง JPEG Coefficient Coding Categories ตามตารางที่ 3.1 เช่นเดียวกับใน DC เมื่อทราบค่า Run และ Category ของข้อมูลแล้วก็จะสามารถหา รหัสข้อมูลจากตารางค่า AC ของฮัฟแมนได้โดยเทียบตามค่า Run/Category

และภายในตารางค่า AC ของฮัฟแมนจะมีรหัสพิเศษ 2 ตัวคือ รหัสเมื่อค่า Run/Category เท่ากับ 0/0 ใช้ในกรณีเมื่อทำการเข้ารหัสข้อมูลภายในแถวจนกระทั่งเหลือแต่ข้อมูลที่เป็นศูนย์เพียงอย่างเดียวก็จะใช้รหัสนี้เป็นตัวบอกตัวถอดรหัสว่าข้อมูลที่เหลือทั้งหมดภายในแถวมีค่าเป็นศูนย์ และ รหัสอีกตัวหนึ่งเมื่อ Run/Category เท่ากับ F/0 ใช้เมื่อมีข้อมูลที่มีค่าศูนย์อยู่ติดกัน 16 ตัวภายในแถว โดยยังมีข้อมูลที่ไม่เป็นศูนย์ที่ยังไม่ถูกเข้ารหัสเหลืออยู่ในแถวอีก ก็จะใช้รหัสตัวนี้แทน ข้อมูลที่มีค่าศูนย์ 16 ตัว ดังกล่าว

จะเห็นได้ว่ารหัสข้อมูลที่ได้จากตารางค่า AC ของฮัฟแมนนั้นใช้ในการบอกจำนวนศูนย์ (คือค่า Run) ที่อยู่ด้านหน้าของข้อมูลนั้นและช่วงของค่าข้อมูล(คือค่า Category) ที่นำมาเข้ารหัสดังนั้นจะต้องมีรหัสตัวที่สอง

สำหรับใช้ในการระบุค่าของข้อมูล ซึ่งการหารหัสตัวที่สองนี้ใช้วิธีเดียวกันกับใน DC คือเอามาจาก LSB จำนวน Category บิตของข้อมูลที่นำมาเข้ารหัสและจะเอามาจากค่า 2's complement ถ้าข้อมูลมีค่าเป็นลบ

ตัวอย่างการบีบอัดข้อมูลภาพขนาด 8×8 บล็อก

ภาพต้นฉบับขนาด 8×8 บล็อก

140	144	147	140	139	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

ค่าของข้อมูลถูกลบด้วย 128

12	16	19	12	11	27	51	47
16	24	12	19	12	20	39	51
24	27	8	39	35	34	24	44
40	17	28	32	24	27	8	32
34	20	28	20	12	8	19	34
19	39	12	27	27	12	8	34
8	28	-5	39	34	16	12	19
20	27	8	227	24	19	19	8

ค่าของข้อมูลหลังผ่านขั้นตอนของ FDCT

47	-5	4	-2	6	-2	-3	-5
5	-9	7	-2	-3	3	3	1
-3	-6	-1	2	-5	1	-3	0
-2	-2	4	-4	-2	0	-1	1
-1	3	2	0	-2	3	3	2
1	-1	-5	1	1	-1	0	-1
2	0	0	1	0	-1	0	0
0	-2	-1	0	0	1	-1	0

ค่าของข้อมูลหลังผ่านกระบวนการควอนไตเซชั่น

46	-4	4	-2	6	-2	-3	-5
5	-9	7	-2	-3	3	3	1
-3	-6	0	2	-5	1	-3	0
-2	-1	4	-4	-2	0	-1	1
-1	2	2	0	-2	3	3	2
1	-1	-5	1	1	-1	0	-1
2	0	0	1	0	-1	0	0
0	-1	0	0	0	1	-1	0

ข้อมูลหลังผ่านกระบวนการจัดเรียงข้อมูลแบบซิกแซก

46,-4,5,-3,-9,4,-2,7,-6,-2,-1,-1,0,-2,6,-2,-3,2,4,2,1,2,-1,2,-4,-5,3,-3,-5,3,1,-2,0,-5,0,0,-1,0,1,-2,0,-3,1,0,-1,3,1,1,0,0,0,
-1,0,1,2,0,-1,0,1,0,-1,0,-1,0

ข้อมูลที่ถูกจัดเตรียมเพื่อเข้ารหัส

(6)(46), (0,3),(-4), (0,3)(5), (0,2)(-3), (0,4)(-9), (0,3)(4), (0,2)(-2), (0,3) (7), (0,3)(-6), (0,2)(-2), (0,1)(-1),
(0,1)(-1),(1,2)(-2), (0,3)(6), (0,2)(-2), (0,2)(-3), (0,2)(2), (0,3)(4), (0,2)(2), (0,1)(1), (0,2)(2), (0,1)(-1), (0,2)(2),
(0,3)(-4), (0,3)(-5), (0,2)(3), (0,2)(-3), (0,3)(-5), (0,2)(3), (0,1)(1), (0,2)(-2), (1,3)(-5), (2,1)(-1), (1,1)(1), (0,2)(-2),
(1,2)(-3), (0,1)(1), (1,1)(-1), (0,2)(3), (0,1)(1), (0,1)(1), (3,1)(-1), (1,1)(1), (0,2)(2), (1,1)(-1), (1,1)(1), (1,1)(-1),
(1,1)(-1),(0,0)

ข้อมูลที่ถูกเข้ารหัสแล้ว

1110101110, 100011, 100101, 0100, 10110110, 100100, 0101, 100111, 100001, 0101, 000, 000, 11100101,
100110, 0100, 0110, 100110, 0110,001, 0110, 000, 0110, 100001, 100010, 0111, 0100, 100010, 0111, 001,
0101, 1111001010, 110110, 11001, 0101, 11100100, 001, 11000, 0111, 001, 001, 1110100, 11001, 0110, 11001,
11000, 11000, 1010

บทที่ 4

การทดลองและผลการทดลอง

การทดลองจะทดลองใช้โปรแกรมบีบอัดข้อมูลภาพบิทแมพชนิด 4,8 และ 24 บิท/พิกเซลที่ขนาดของภาพต่าง ๆ กัน โดยใช้ค่า Q. F. ต่างๆกันในการบีบอัดโดยใช้ค่า Q. F. ในการบีบอัดเท่ากับ 0, 50 และ 100 เพื่อศึกษาอัตราการบีบอัดข้อมูลและคุณภาพของข้อมูลที่ได้จากการบีบอัด ซึ่งจะได้ผลการทดลองดังแสดงในตารางที่ 4.1

รูปภาพ	ชนิด(bits/pixel)	กว้างxยาว (pixel)	Q.F.	Compression ratio
GIRL	Grayscale(8)	256X256	0	66%
			50	93%
			100	95%
ผีเสื้อ	Grayscale(8)	250x250	0	57%
			50	88%
			100	92%
พริก	4	250x250	0	-93%
			50	39%
			100	67%
ผู้หญิง	4	300X300	0	-89%
			50	41%
			100	66%
LENA	8	256X256	0	49%
			50	91%
			100	95%
ดอกไม้	8	250X250	0	47%
			50	88%
			100	93%
LEO	24	246X325	0	84%
			50	96%
			100	98%
BABY	24	512X384	0	86%
			50	97%
			100	98%

ตารางที่ 4.1 แสดงผลการทดสอบการบีบอัดภาพบิทแมพ

แสดงตัวอย่างภาพที่ใช้ในการทดลองเป็นภาพชนิด 8 บิท/พิกเซล

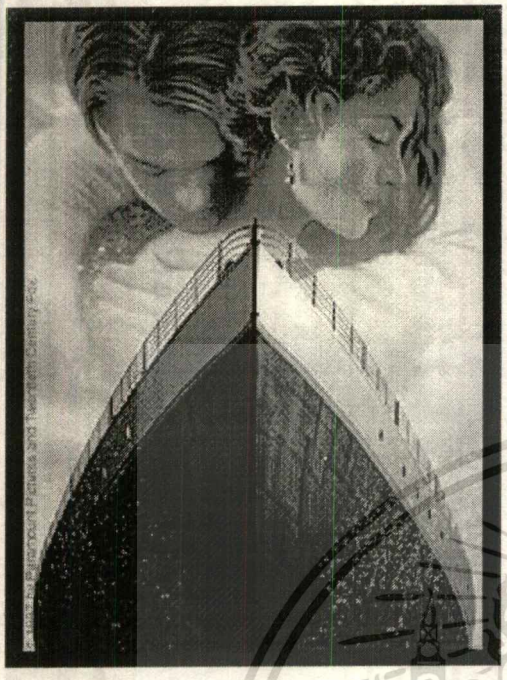
ภาพต้นฉบับของ Lena

ภาพ Lena เมื่อถูกบีบอัดโดยใช้ Q.F เท่ากับ0

ภาพ Lena เมื่อถูกบีบอัดโดยใช้ Q.F เท่ากับ 50

ภาพ Lena เมื่อถูกบีบอัดโดยใช้ Q.F เท่ากับ 100

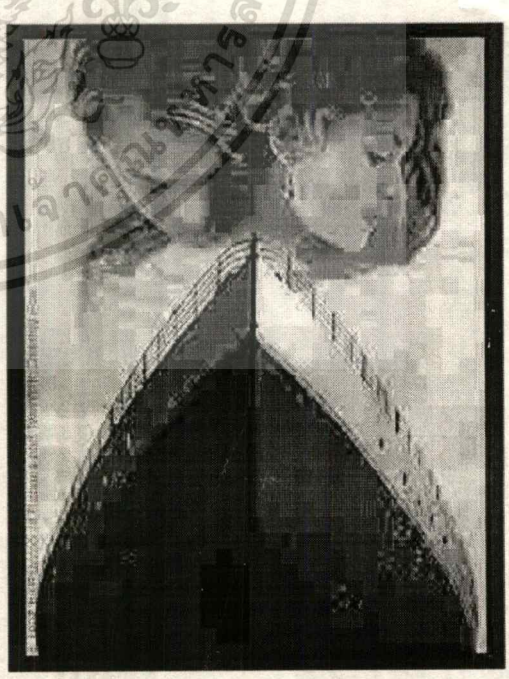
แสดงภาพตัวอย่างที่ใช้ในการทดลองเป็นภาพชนิด 24 บิท/พิกเซล



รูปภาพต้นฉบับของ TITANIC

รูป TITANIC เมื่อถูกบีบอัดโดยใช้ค่า Q.F.เท่ากับ0

รูป TITANIC เมื่อถูกบีบอัดโดยใช้ค่า Q.F.เท่ากับ 50



รูป TITANIC เมื่อถูกบีบอัดโดยใช้ค่า Q.F.เท่ากับ 100

แสดงการเปรียบเทียบภาพ Bitmap ต้นฉบับกับภาพ Bitmap ที่ได้ทำการแปลงกลับหลังจากได้รับการบีบอัดแล้ว

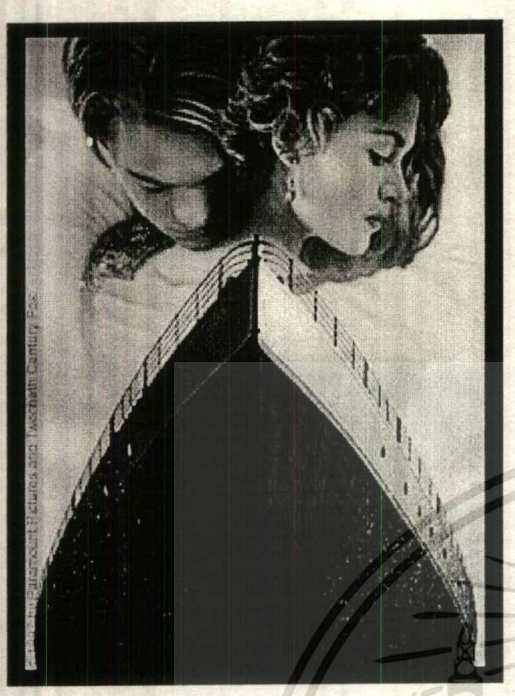
ภาพต้นฉบับ

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 0

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 50

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 100

แสดงการเปรียบเทียบภาพ Bitmap ต้นฉบับกับภาพ Bitmap ที่ได้ทำการแปลงกลับหลังจากได้รับการบีบอัดแล้ว



ภาพต้นฉบับ

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 0

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 50

ภาพที่ได้จากการแปลงกลับหลังจากถูกบีบอัด
โดยใช้ค่า Q.F เท่ากับ 100

บทที่ 5

สรุปผลการทดลอง

จากตารางและภาพที่ได้จากการทดลองในบทที่ 4 เมื่อเราพิจารณาจากอัตราส่วนการบีบอัดข้อมูล (Compression Ratio) กับชนิดของภาพที่นำมาบีบอัดจะเห็นได้ว่า

-ในการบีบอัดข้อมูลภาพบิทแมพชนิด 4 บิท/พิกเซล จะให้อัตราส่วนการบีบอัดข้อมูลที่แย่ คือ ขนาดข้อมูลที่ได้จากการบีบอัดอาจมีขนาดใหญ่กว่าเดิมหรือไม่ก็จะได้อัตราส่วนการบีบอัดข้อมูลที่ต่ำ

-ในการบีบอัดข้อมูลภาพบิทแมพชนิด 8 บิท/พิกเซล อัตราส่วนการบีบอัดข้อมูลที่ได้จะอยู่ในเกณฑ์ดี คือ ประมาณ 20 % ถึง 49% เมื่อบีบอัดด้วยค่า Q.F. เท่ากับ 0 โดยขึ้นกับขนาดของภาพที่นำมาบีบอัด และสามารถบีบอัดได้ในอัตราสูงขึ้นเป็นอัตราส่วนสูงถึง 81% ถึง 95% เมื่อบีบอัดด้วยค่า Q.F. เท่ากับ 50 และ 100 แต่ถ้าบีบอัดด้วยค่า Q.F. ที่สูงมากเกินไปจะทำให้คุณภาพของภาพที่ได้แย่ลงมาก

-สำหรับการบีบอัดข้อมูลภาพชนิดเกรย์สเกล (8 บิท) อัตราส่วนการบีบอัดข้อมูลที่ได้จะอยู่ในเกณฑ์ดี และมีอัตราส่วนการบีบอัดสูงกว่าการบีบอัดภาพชนิด 8 บิท/พิกเซล และ ชนิด 4 บิท/พิกเซล

-ส่วนการบีบอัดภาพชนิด 24 บิท/พิกเซล นั้น จะสามารถลดอัตราส่วนการบีบอัดข้อมูลได้สูงมาก คือ ประมาณ 84% เมื่อใช้เพียงค่า Q. F. เท่ากับ 0 และจะสามารถบีบอัดข้อมูลได้สูงถึง 98% เมื่อบีบอัดด้วยค่า Q. F.เท่ากับ 100

ซึ่งเมื่อพิจารณาจากอัตราส่วนการบีบอัดข้อมูลและคุณภาพของภาพที่ได้กับค่า Q.F. ที่ใช้จะเห็นว่า เมื่อใช้ค่า Q.F. ที่สูงขึ้นในการบีบอัดข้อมูลภาพจะทำให้อัตราส่วนการบีบอัดข้อมูลภาพที่ได้สูงขึ้นแต่คุณภาพของภาพที่ได้ก็จะแย่ลงตามไปด้วย

สรุปได้ว่าการบีบอัดข้อมูลภาพด้วยวิธีการ JPEG ชนิดซีเควนเชียลเบสไลน์ มีความเหมาะสมสำหรับการบีบอัดข้อมูลภาพบิทแมพชนิด 24 บิท/พิกเซลมากที่สุด และยังสามารถใช้ได้ดีในการบีบอัดภาพชนิด 8 บิท/พิกเซล ด้วย เนื่องจากจะให้อัตราส่วนการบีบอัดข้อมูลที่ดี และสามารถบีบอัดข้อมูลได้สูงขึ้นเมื่อใช้ค่า Q.F.ที่สูงขึ้นคือยอมสูญเสียคุณภาพของภาพมากขึ้น แต่ไม่เหมาะสำหรับการบีบอัดภาพแบบ 1,4 บิท/พิกเซล เนื่องจากให้อัตราส่วนการบีบอัดข้อมูลที่ไม่ดี และคุณภาพของภาพที่ได้ก็แย่ลงมากขึ้นอยู่กับค่าควอนไตเซชัน คือถ้าค่าควอนไตเซชันสูงขึ้น อัตราส่วนการบีบอัดข้อมูลที่ได้ก็จะสูงขึ้นแต่คุณภาพของภาพที่ได้ก็จะแย่ลงตามลำดับ

ภาคผนวก

```
// ================================================
//  Compression by JPEG
// ================================================

#include <io.h>
#include <dir.h>
#include <dos.h>
#include <math.h>
#include <alloc.h>
#include <conio.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdio.h>
#include <direct.h>
#include <stdlib.h>
#include <string.h>
#include <sys\stat.h>
#include "A:\WORK1.H"

extern int   CheckType(char* );
extern char  *ChangeName(char* ,char* );
extern void  FilePath(char* ,char* ,char* ,char* );
extern char  **AllocateArray2CHAR(int,int);
extern BYTE  **AllocateArray2BYTE(int,int);
extern float **AllocateArray2FLOAT(int,int);
extern BYTE  ***AllocateArray3BYTE(int,int,int);
extern void  FreeArray2BYTE(BYTE** ,int);
extern void  FreeArray2FLOAT(float** ,int);
extern void  FreeArray3BYTE(BYTE*** ,int,int);
extern int   QuantizerFactor(char* );
extern void  WriteJFIFheader(BYTE* ,BYTE* ,JFIFHEADER*
            ,BITMAPINFOHEADER* ,FILE* );
extern void  RGBtoYCbCrTable(BITMAPCOLORTABLE* ,YCbCrCOLORTABLE* ,int);
extern void  RGBtoYCbCr(BYTE*** ,float** ,float** ,float** ,int);
extern void  ReadPixels(int,int,long,long,WORD,int,BYTE*** ,FILE* ,char* );
extern void  CalculateCoeffDCT(float** );
extern void  FDCT_Quantize_ZigZag(float** ,int* ,float** ,BYTE* );
extern void  Initilize(BYTE** ,int);
extern int   GetCategory(int);
extern long  PackBits(WORD,int,BYTE** ,long);
extern long  HuffmanEncode(int* ,int* ,BYTE** ,long);
extern long  WriteCode(int,BYTE** ,long,FILE* ,char* );

//------------------------------
void main(int argc,char* argv[])
{
  ffblk             ffblk;          // DOS file control block
  FILE              *fi;            // file pointer to input file
  FILE              *fo;            // file pointer to output file
  char              ext[5];         // extension oflist of i/p file
  char              file[9];        // name of list of input file
  char              drive[3];       // drive of list of input file
  char              dir[80];        // directory of listof i/p file
  char              input_path[80]; // full path of inputfile
```

```
char                output_path[80];    // full path of outputfile
char                *output_name;       // output file name
char                **input_name;       // input file name
char                samp[3];
int                 open_flag;          // open file name flag
int                 compress_flag;      // compression flag
int                 fi_handle;          // handle of input file
int                 fo_handle;          // handle of output file
int                 overwrite_flag=0;   // overwrite exist file flag
int                 color_number;       // number of colors in colortable
int                 plane;              // number of planein pic_buffer
int                 index1=0,index2=0;  // index counter
int                 row,col;            // row and column counter
int                 row_shift;          // row shift counter
int                 col_shift;          // column shift counter
int                 block_row;          // block row counter
int                 block_col;          // block column counter
int                 quantizer_factor=0;// quantization factor
int                 Y_DC,Cb_DC,Cr_DC;   // dc coefficient storage
int                 mem_row;            // row of output buffer
long                byte_row;           // number of byte which padded
                                        // between row
long                byte_end;           // number of unused byte at end
                                        // of inputfile
long                inputfile_size;     // size of Bitmap input file
long                outputfile_size;    // size of JFIF output file
long                x_size,y_size;      // size of image in col and row
long                pic_size;           // picture size
long                bit_count;          // Encoded bit count
long                bit_limit;          // number of bits which output
                                        // buffer can hold
DWORD               mem_free;           // unused memory in bytes
float               **Y,**Cb,**Cr;      // Y,Cb,Cr buffer
float               **DCT_coeff;        // DCT coefficients buffer
float               **pixel_buffer;     // pixel buffer
BYTE                **output_buffer;    // output buffer
BYTE                ***pic_buffer;      // input image buffer
int                 zigzag_data[64];    // zigzag data
BITMAPCOLORTABLE    *RGB_table;         // RGB table color buffer
YCbCrCOLORTABLE     *YCbCr_table;       // YCbCr table color buffer
BITMAPMAINHEADER    mainheader;         // Bitmap file mainheader
BITMAPINFOHEADER    infoheader;         // Bitmap file infoheader
JFIFHEADER          jfifheader;         // JFIF file header
BYTE                luma_quantizer[64]={ 4, 4, 4, 4, 4, 4, 4 ,4
                                         4, 4, 4, 4, 4, 4, 4, 6,
                                         4, 4, 4, 4, 4, 4, 6, 6,
                                         4, 4, 4, 4, 4, 6, 6, 6,
                                         4, 4, 4, 4, 6, 6, 6, 8,
                                         4, 4, 4, 6, 6, 6, 8, 8,
                                         4, 4, 6, 6, 6, 8, 8,12,
                                         4, 6, 6, 6, 8, 8,12,15
                                        };
```

```
   BYTE                chroma_quantizer[64]={ 4, 4, 4, 4, 4, 4, 4, 4,
                                              4, 4, 4, 4, 4, 4, 4, 6,
                                              4, 4, 4, 4, 4, 4, 6, 6,
                                              4, 4, 4, 4, 4, 6, 6, 6,
                                              4, 4, 4, 4, 6, 6, 6, 8,
                                              4, 4, 4, 6, 6, 6, 8, 8,
                                              4, 4, 6, 6, 6, 8, 8,12,
                                              4, 6, 6, 6, 8, 8,12,15
                                             };

  printf("\n JPEG Image Compression Utility.");
  if(argc<2 || argc>3)
  {
   printf("\n (for compress 1,4,8 and 24 bits/pixel Windows Bitmap file to
          JFIF file) \n");
   printf("\n Usage : JPEG  <inputfile(s)> [option]");
   printf("\n List of inputfile(s) (maximum 200 files) may use wildcard ?
          and *");
   printf("\n Outputfile's name is same as input filename, but extension
          .jpg");
   printf("\n Option");
   printf("\n /q:xx   define xx factor for quantization : 0 to 100");
   printf("\n 0   : highest quality of compressed picture,lowest compression
             ratio");
   printf("\n 100: lowest quality of compressed picture,highest compression
             ratio");
   printf("\n default factor: 0 when not define \n");
   exit(0);
  }
  printf("\n");
  if(argc==3)
  {
  if(argv[2][0]!='/'||(argv[2][1]!='q'&&argv[2][1]!='Q')||    argv[2][2]!=':')
     {
      printf("\n ERROR : Unknow option '%s'\n",argv[2]);
      exit(0);
     }
    if(argv[2][3]!='0' || argv[2][4]!='\0')
    quantizer_factor=QuantizerFactor(argv[2]);
   }
   printf("\n what type");
   gets(samp);
     //----Calculate quantizer when user define quantization factor ---//
  if(quantizer_factor>0)
  {
   for(row=0;row<8;row++)
   for(col=0;col<8;col++)
   {
   luma_quantizer[row*8+col]+=quantizer_factor;
   chroma_quantizer[row*8+col]+=quantizer_factor;
   }
  }
  input_name=AllocateArray2CHAR(200,13);
  open_flag=fnsplit(argv[1],drive,dir,file,ext);
```

```
if(dir[0]=='.')
 {
  printf("\n ERROR: You should use full path of inputfile\n");
  exit(0);
 }
open_flag=findfirst(argv[1],&ffblk,0);
if(open_flag==(-1))
 {
  printf("\n ERROR: Can't open list of '%s'.\n",argv[1]);
  exit(0);
 }
strcpy(input_name[index1],ffblk.ff_name);
while(open_flag==0 && index1<=199)
 {
  index1+=1;
  open_flag=findnext(&ffblk);
  strcpy(input_name[index1],ffblk.ff_name);
 }
compress_flag=0;
if(index1==200) printf("\n Can process maximum 200 file once \n");
printf("\n Quantizer factor(0-100): %d",quantizer_factor);
while(index2<index1)
 {
  FilePath(input_path,drive,dir,input_name[index2]);
  fi_handle=open(input_path,O_BINARY,S_IREAD);
  fi=fdopen(fi_handle,"rb");
  output_name=ChangeName(input_name[index2],"JPG");
  FilePath(output_path,drive,dir,output_name);
  printf("\n %s => %s",strlwr(input_path),strlwr(output_path));
  fread(&mainheader,sizeof(BITMAPMAINHEADER),1,fi);
  if(CheckType(mainheader.file_id)!=0)
  {
   printf(", isn't BITMAP file.");
   compress_flag=1;
  }
  fread(&infoheader,sizeof(BITMAPINFOHEADER),1,fi);
  if(infoheader.compression!=0 && compress_flag==0)
  {
   printf(", this Bitmap was compressed by RLE (I can't read).");
   compress_flag=1;
  }

              //----- open outputfile -----//
 if(strcmp(strlwr(output_path),strlwr(input_path))!=0 && compress_flag==0)
  {
   fo_handle=open(output_path,O_CREAT|O_EXCL|O_BINARY,S_IWRITE);
   fo=fdopen(fo_handle,"wb");
   if(fo==NULL)
    {
     if(overwrite_flag!='a')
     {
     printf(", already exist.Overwrite? [y/n/a]  ");
     do {
          printf("\b");
```

```
            overwrite_flag=getche();
            overwrite_flag=tolower(overwrite_flag);
          }
        while(overwrite_flag!='y' && overwrite_flag!='a' &&
            overwrite_flag!='n');
        }
        if(overwrite_flag=='y' || overwrite_flag=='a' )
        {
         remove(output_path);
         fo_handle=open(output_path,O_CREAT|O_BINARY,S_IWRITE);
         fo=fdopen(fo_handle,"wb");
        }
        else compress_flag=1;
       }
       inputfile_size=filelength(fi_handle);
    }
    else
     {
      if(compress_flag==0)
      printf(", \"%s\" has extension .jpg alrealy",strlwr(output_name));
      compress_flag=1;
     }

                //----- calculate factor of reading inputfile -----//
  if(compress_flag==0)
  {
   x_size=infoheader.width;
   y_size=infoheader.height;
   if(infoheader.bitcount==1) // Bitmap type 1 bits/pixel
   {
    plane=1;
    pic_size=((x_size+7L)/8L)*y_size;
    if(infoheader.clrused==0L) color_number=2;
    else color_number=(int)infoheader.clrused;
    if(infoheader.image_size==0L)
    byte_row=(inputfile_size-pic_size-(4L*(long)color_number)-54L)
              /y_size;
    else byte_row=((long)infoheader.image_size-pic_size)/y_size;
   }
  if(infoheader.bitcount==4) // Bitmap type 4 bits/pixel
   {
    plane=1;
    pic_size=((x_size+1L)/2L)*y_size;
    if(infoheader.clrused==0L) color_number=16;
    else color_number=(int)infoheader.clrused;
    if(infoheader.image_size==0L)
   byte_row=(inputfile_size-pic_size-(4L*(long)color_number)-54L)
              /y_size;
    else
    byte_row=((long)infoheader.image_size-pic_size)/y_size;
   }

  if(infoheader.bitcount==8) // Bitmap type 8 bits/pixel
   {
```

```
  plane=1;
  pic_size=x_size*y_size;
  if(infoheader.clrused==0L) color_number=256;
  else color_number=(int)infoheader.clrused;
  if(infoheader.image_size==0L)
  byte_row=(inputfile_size-pic_size-(4L*(long)color_number)- 54L)/y_size;
  else byte_row=((long)infoheader.image_size-pic_size)/y_size;
 }
 if(infoheader.bitcount==24) // Bitmap type 24 bits/pixel
 {
  plane=3;
  pic_size=3L*x_size*y_size;
  if(infoheader.image_size==0L)
  {
  if((inputfile_size-pic_size-54L)%y_size==0L)
  infoheader.clrused=0L;
  byte_row=(inputfile_size-pic_size-(4L*(long)infoheader.clrused)-54L)
           /y_size;
  byte_end=(inputfile_size-pic_size-(4L*(long)infoheader.clrused)-54L)
          -(y_size*byte_row);
 }
 else
 {
   byte_row=((long)infoheader.image_size-pic_size)/y_size;
   byte_end=inputfile_size-pic_size-(4L*(long)infoheader.clrused)-54L
           -(y_size*byte_row);
 }
 }
 printf("\n");
 printf("Width = %d\n",x_size);
 printf("Height = %d\n",y_size);
 printf("Byte type = %d bits/pixel \n",infoheader.bitcount);
 printf("Colour in picture %d\n",infoheader.clrused);
 getch();

                  //----- allocate buffer -----//
 DCT_coeff     = AllocateArray2FLOAT(8,8);
 pixel_buffer  = AllocateArray2FLOAT(8,8);
 if(plane==1) // colortable Bitmap (1,4,8 bits/pixel)
  {
   RGB_table   = (BITMAPCOLORTABLE*)
   malloc((color_number)*sizeof(BITMAPCOLORTABLE));
   YCbCr_table = (YCbCrCOLORTABLE *)
   malloc((color_number)*sizeof(YCbCrCOLORTABLE));
   pic_buffer  = AllocateArray3BYTE(1,(int)y_size+16,(int)x_size+24);
  }
 if(plane==3) // truecolor Bitmap (24 bits/pixel)
  {
   pic_buffer  = AllocateArray3BYTE(3,16,(int)x_size+16);
   Y =AllocateArray2FLOAT(16,(int)x_size+16);
   Cb=AllocateArray2FLOAT(16,(int)x_size+16);
   Cr=AllocateArray2FLOAT(16,(int)x_size+16);
  }
```

```
if((plane==1 && (RGB_table ==NULL || YCbCr_table ==NULL)) ||
   (plane==3 && (Y==NULL || Cb==NULL || Cr==NULL)) ||
    pic_buffer==NULL || pixel_buffer==NULL || DCT_coeff==NULL)
{
 printf("\n ERROR : Could not allocate sufficient buffer storage.\n");
 remove(output_path);
 exit(0);
}

      //----- read pixels of colortable Bitmap and zero padding -----//
if(plane==1)
 {
  fread(RGB_table,sizeof(BITMAPCOLORTABLE),color_number,fi);
  RGBtoYCbCrTable(RGB_table,YCbCr_table,color_number);
  ReadPixels((int)y_size,(int)x_size,byte_row,byte_end,
  infoheader.bitcount,0,pic_buffer,fi,output_path);


      //----- make image full with MCU(16x16) -----//
if((x_size%16L)!=0L)
 {
  for(row=0;row<y_size;row++)
  for(col=0;col<16;col++)
  pic_buffer[0][row][(int)x_size+col]=pic_buffer[0][row][(int)x_size-1];
  x_size+=16L;
 }
if((y_size%16L)!=0L)
 {
  for(row=0;row<16;row++)
  for(col=0;col<x_size;col++)
  pic_buffer[0][(int)y_size+row][col]=pic_buffer[0][(int)y_size-1][col];
  y_size+=16L;
 }
}
if(plane==3) // truecolor Bitmap
  {
    if((x_size%16L)!=0L) x_size+=16L;
    if((y_size%16L)!=0L) y_size+=16L;
  }


            //----- calculate and allocate output buffer -----//
 mem_free=coreleft();
 mem_row=(int)(mem_free/10000L)-1;
 bit_limit=(((long)mem_row*10000L)-2000L)*8L;
 output_buffer = AllocateArray2BYTE(mem_row,10000);
 if(output_buffer==NULL || mem_row==0)
  {
   printf("\n ERROR : Could not allocate sufficient buffer storage.\n");
   remove(output_path);
   exit(0);
  }
```

```
        //----- compression process -----//
if(samp[0]=='1')
{
 printf("\n\n     ");
 bit_count=0L;
 row_shift=0;
 Y_DC=0; Cb_DC=0; Cr_DC=0;
 Initilize(output_buffer,mem_row);
 CalculateCoeffDCT(DCT_coeff);

WriteJFIFheader(luma_quantizer,chroma_quantizer,&jfifheader,&infoheader,fo);
for(block_row=0;block_row<(int)y_size/16;block_row++)
 {
  col_shift=0;
  if(plane==3) // read pixel in truecolor Bitmap
  {
   ReadPixels((int)infoheader.height,(int)infoheader.width,byte_row,
              byte_end,infoheader.bitcount,block_row,
   pic_buffer,fi,output_path);
   RGBtoYCbCr(pic_buffer,Y,Cb,Cr,(int)x_size);
 }
for(block_col=0;block_col<(int)x_size/16;block_col++)
   {

            //----- process Y pixels with subsampling H:1 V:1 -----//
                      //----- upper left block -----//
 for(row=0;row<8;row++)
 for(col=0;col<8;col++)
  {
   if(plane==1) // colortable Bitmap
   pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift]
      [col+col_shift]].Y;
   else // truecolor Bitmap
   pixel_buffer[row][col]=Y[row][col+col_shift];
  }
 FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
 bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);

                    //----- upper right block -----//
 for(row=0;row<8;row++)
 for(col=0;col<8;col++)
 {
  if(plane==1)
  pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift]
      [col+col_shift+8]].Y;
  else
  pixel_buffer[row][col]=Y[row][col+col_shift+8];
 }
 FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
 bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);

                    //----- lower left block -----//
 for(row=0;row<8;row++)
 for(col=0;col<8;col++)
```

```
    {
       if(plane==1)
       pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift+8]
                                  [col+col_shift]].Y;
       else
       pixel_buffer[row][col]=Y[row+8][col+col_shift];
    }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);

                    //------ lower right block -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
    {
       if(plane==1)
       pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift+8]
                                  [col+col_shift+8]].Y;
       else
       pixel_buffer[row][col]=Y[row+8][col+col_shift+8];
     }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);


              //----- process Cb pixels with subsampling H:2 V:2 -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
    {
       if(plane==1) // colortable Bitmap
       pixel_buffer[row][col]=(YCbCr_table[pic_buffer[0][2*row+row_shift]
                         [2*col+col_shift]].Cb+YCbCr_table[pic_buffer[0]
                         [2*row+row_shift][2*col+col_shift+1]].Cb
                         +YCbCr_table[pic_buffer[0][2*row+row_shift+1]
                         [2*col+col_shift]].Cb+YCbCr_table[pic_buffer[0]
                         [2*row+row_shift+1][2*col+col_shift+1]].Cb)/4.00;
       else // truecolor Bitmap
       pixel_buffer[row][col]=((Cb[2*row][2*col+col_shift]+Cb[2*row]
                                  [2*col+col_shift+1]+Cb[2*row+1]
                                  [2*col+col_shift]+Cb[2*row+1]
                                  [2*col+col_shift+1])/4.00);
     }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,chroma_quantizer);
  bit_count=HuffmanEncode(&Cb_DC,zigzag_data,output_buffer,bit_count);


              //----- process Cr pixels with subsampling H:2 V:2 -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {
    if(plane==1) // colortable Bitmap
    pixel_buffer[row][col]=(YCbCr_table[pic_buffer[0][2*row+row_shift]
                          +col_shift]].Cr+YCbCr_table[pic_buffer[0]
                          row+row_shift][2*col+col_shift+1]].Cr
                          +YCbCr_table[pic_buffer[0][2*row+row_shift+1]
```

```
                        [2*col+col_shift]].Cr+YCbCr_table[pic_buffer[0]
                        [2*row+row_shift+1][2*col+col_shift+1]].Cr)/4.00;
   else // truecolor Bitmap
   pixel_buffer[row][col]=((Cr[2*row][2*col+col_shift]+Cr[2*row]
                        [2*col+col_shift+1]+Cr[2*row+1][2*col+col_shift]
                        +Cr[2*row+1][2*col+col_shift+1])/4.00);
  }
 FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,chroma_quantizer);
 bit_count=HuffmanEncode(&Cr_DC,zigzag_data,output_buffer,bit_count);


                    //----- write data in buffer to outputfile -----//
  if(bit_count>bit_limit)
  bit_count=WriteCode(mem_row,output_buffer,bit_count,fo,output_path);
  col_shift+=16;
  printf("\b\b\b\b%3.0f\%",((float)block_row+((float)block_col/
        ((float)x_size/16)))*100/((float)y_size/16));
    }
  row_shift+=16;
   }
  }
  if(samp[0]=='2')
  {
   printf("\n\n     ");
   bit_count=0L;
   row_shift=0;
   Y_DC=0; Cb_DC=0; Cr_DC=0;
   Initilize(output_buffer,mem_row);
   CalculateCoeffDCT(DCT_coeff);
  WriteJFIFheader(luma_quantizer,chroma_quantizer,&jfifheader,&infoheader,fo);
  for(block_row=0;block_row<(int)y_size/16;block_row++)
  {
    col_shift=0;
    if(plane==3) // read pixel in truecolor Bitmap
   {
  ReadPixels((int)infoheader.height,(int)infoheader.width,byte_row,byte_end
              ,infoheader.bitcount,block_row,pic_buffer,fi,output_path);
  RGBtoYCbCr(pic_buffer,Y,Cb,Cr,(int)x_size);
   }
  for(block_col=0;block_col<(int)x_size/16;block_col++)
   {
              //----- process Y pixels with subsampling H:1 V:1 -----//
                    //----- upper left block -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {
     if(plane==1) // colortable Bitmap
     pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift]
                           [col+col_shift]].Y;
     else // truecolor Bitmap
     pixel_buffer[row][col]=Y[row][col+col_shift];
  }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);
```

```
                    //----- upper right block -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {
     if(plane==1)
     pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift]
                    [col+col_shift+8]].Y;
     else
     pixel_buffer[row][col]=Y[row][col+col_shift+8];
  }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);

                    //----- lower left block -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {
     if(plane==1)
     pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift+8]
                    [col+col_shift]].Y;
     else
     pixel_buffer[row][col]=Y[row+8][col+col_shift];
  }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);

                    //------ lower right block -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {  if(plane==1)
     pixel_buffer[row][col]=YCbCr_table[pic_buffer[0][row+row_shift+8]
                    [col+col_shift+8]].Y;
     else
     pixel_buffer[row][col]=Y[row+8][col+col_shift+8];
  }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,luma_quantizer);
  bit_count=HuffmanEncode(&Y_DC,zigzag_data,output_buffer,bit_count);


              //----- process Cb pixels with subsampling H:2 V:1 -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {  if(plane==1) // colortable Bitmap
     pixel_buffer[row][col]=(YCbCr_table[pic_buffer[0][2*row+row_shift]
                    [2*col+col_shift]].Cb+YCbCr_table[pic_buffer[0]
                    [2*row+row_shift][2*col+col_shift+1]].Cb)/2.00;
     else // truecolor Bitmap
     pixel_buffer[row][col]=((Cb[2*row][2*col+col_shift]+Cb[2*row]
                    [2*col+col_shift+1])/2.00);
  }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,chroma_quantizer);
  bit_count=HuffmanEncode(&Cb_DC,zigzag_data,output_buffer,bit_count);
```

```
        //----- process Cr pixels with subsampling H:2 V:1 -----//
  for(row=0;row<8;row++)
  for(col=0;col<8;col++)
  {
     if(plane==1) // colortable Bitmap
     pixel_buffer[row][col]=(YCbCr_table[pic_buffer[0][2*row+row_shift]
                    [2*col+col_shift]].Cr+YCbCr_table[pic_buffer[0]
                    [2*row+row_shift][2*col+col_shift+1]].Cr)/2.00;
     else // truecolor Bitmap
     pixel_buffer[row][col]=((Cr[2*row][2*col+col_shift]+Cr[2*row]
                    [2*col+col_shift+1])/2.00);
   }
  FDCT_Quantize_ZigZag(pixel_buffer,zigzag_data,DCT_coeff,chroma_quantizer);
  bit_count=HuffmanEncode(&Cr_DC,zigzag_data,output_buffer,bit_count);


                    //----- write data in buffer to outputfile -----//
  if(bit_count>bit_limit)
  bit_count=WriteCode(mem_row,output_buffer,bit_count,fo,output_path);
  col_shift+=16;
  printf("\b\b\b\b%3.0f\%",((float)block_row+((float)block_col/
              ((float)x_size/16)))*100/((float)y_size/16));
    }
  row_shift+=16;
  }
  }
  if((bit_count%8)!=0)
  bit_count=PackBits(0xFFFF,8-(int)(bit_count%8L),output_buffer,bit_count);
  bit_count=WriteCode(mem_row,output_buffer,bit_count,fo,output_path);
  printf("\n\n     ");


                     //----- end of image marker -----//
  if(putw(0xD9FF,fo)!=(int)0xD9FF)
  {
  printf("\n\n ERROR : Disk full \n");
  remove(output_path);
  exit(0);
  }
  outputfile_size=filelength(fo_handle)+2;
  printf("\b\b\b\b(%2d%%), done.",(int)((1-((double)
  outputfile_size/(double)inputfile_size))*100));
                     //----- free memory of buffer -----//
  if(plane==1)
  {
  free(RGB_table);
  free(YCbCr_table);
  if((y_size%16L)!=0L) y_size-=16L;
  FreeArray3BYTE(pic_buffer,1,(int)y_size+16);
  }
  if(plane==3)
   {
    FreeArray3BYTE(pic_buffer,3,16);
    FreeArray2FLOAT(Y,16);
```

```
    FreeArray2FLOAT(Cb,16);
    FreeArray2FLOAT(Cr,16);
   }
    FreeArray2FLOAT(DCT_coeff,8);
    FreeArray2FLOAT(pixel_buffer,8);
    FreeArray2BYTE(output_buffer,mem_row);
   fclose(fi);
   fclose(fo);
   }
   compress_flag=0;
   index2+=1;
   }
  printf("\n");
  }
```

```
//=====================
// JPEG_MEM.CPP
//=====================

#include <stdlib.h>
#include <A:\WORK1.H>

//-------------------------------------------------
// Allocate buffer for 2-dimensional array of CHAR
//-------------------------------------------------
char **AllocateArray2CHAR(int row,int col)
{
 char **buffer;
 int index,free1;
 buffer=(char **)malloc(row*sizeof(char *));
 if(buffer==NULL) return(NULL);
 for(index=0;index<row;index++)
 {
  buffer[index]=(char*)malloc(col*sizeof(char));
  if(buffer[index]==NULL) //free memory when cound not allocate buffer
   {
    for(free1=0;free1<index;free1++) free(buffer[free1]);
    free(buffer);
    return(NULL);
   }
 }
 return(buffer);
}

 //-------------------------------------------------
 // Allocate buffer for 2-dimensional array of BYTE
 //-------------------------------------------------
BYTE **AllocateArray2BYTE(int row,int col)
{
 BYTE **buffer;
 int index,free1;
 buffer=(BYTE **)malloc(row*sizeof(BYTE *));
 if(buffer==NULL) return(NULL);
 for(index=0;index<row;index++)
 {
 buffer[index]=(BYTE *)malloc(col*sizeof(BYTE));
 if(buffer[index]==NULL) // free memory when cound not allocate buffer
  {
   for(free1=0;free1<index;free1++) free(buffer[free1]);
   free(buffer);
   return(NULL);
  }
 }
 return(buffer);
}

 //-------------------------------------------------
 // Allocate buffer for 2-dimensional array of FLOAT
 //-------------------------------------------------
 float **AllocateArray2FLOAT(int row,int col)
 {
  float **buffer;
  int index,free1;
```

```
 buffer=(float **)malloc(row*sizeof(float *));
 if(buffer==NULL) return(NULL);
 for(index=0;index<row;index++)
 {
  buffer[index]=(float *)malloc(col*sizeof(float));
  if(buffer[index]==NULL) // free memory when could not allocate buffer
   {
    for(free1=0;free1<index;free1++) free(buffer[free1]);
    free(buffer);
    return(NULL);
   }
  }
 return(buffer);
}

//--------------------------------------------------
// Allocate buffer for 3-dimensional array of BYTE
//--------------------------------------------------

BYTE ***AllocateArray3BYTE(int plane,int row,int col)
  {
  BYTE ***buffer;
  int x,y;
  int free1,free2;

  buffer=(BYTE ***)malloc(plane*sizeof(BYTE** ));
  if(buffer==NULL) return(NULL);
  for(x=0;x<plane;x++)
    {
    buffer[x]=(BYTE **)malloc(row*sizeof(BYTE* ));
    if(buffer[x]==NULL)
      {
      for(free1=0;free1<x;free1++) free(buffer[x]);
      free(buffer);
      return(NULL);
      }
    for(y=0;y<row;y++)
      {
      buffer[x][y]=(BYTE *)malloc(col*sizeof(BYTE));
      if(buffer[x][y]==NULL)
        {
        for(free2=0;free2<y;free2++) free(buffer[x][free2]);
        free(buffer[x]);
        for(free1=0;free1<(x-1);free1++)
        for(free2=0;free2<row;free2++)
          {
          free(buffer[free1][free2]);
          free(buffer[free1]);
          }
        free(buffer);
        return(NULL);
        }
      }
    }
  return(buffer);
  }
```

```
//-----------------------------------------------------
// Free memory of 2-dimensional array of BYTE buffer
//-----------------------------------------------------
void FreeArray2BYTE(BYTE** buffer,int row)
  {
  int index;

  for(index=0;index<row;index++)
  free(buffer[index]);
  free(buffer);
  }

//-----------------------------------------------------
// Free memory from 2-dimensional array of FLOAT buffer
//-----------------------------------------------------
void FreeArray2FLOAT(float** buffer,int row)
  {
  int index;

  for(index=0;index<row;index++)
  free(buffer[index]);
  free(buffer);
  }

//-----------------------------------------------------
// Free memory from 3-dimensional array of BYTE buffer
//-----------------------------------------------------
void FreeArray3BYTE(BYTE*** buffer,int plane,int row)
  {
  int index1,index2;

  for(index1=0;index1<plane;index1++)
    {
    for(index2=0;index2<row;index2++)
    free(buffer[index1][index2]);
    free(buffer[index1]);
    }
  free(buffer);
  }
```

```
//================================
// JPEG.H
//================================

#define  BYTE     unsigned char
#define  WORD     unsigned int
#define  DWORD    unsigned long
#define  ROUND(A) ((A<0.00)?(int)(A-0.5f):(int)(A+0.5f))

//------------------------
//      Bitmap File
//------------------------
typedef struct                  // Bitmap main header
  {
  char       file_id[2];        // must be ASCII text "BM"
  DWORD      size_file;         // size of the file
  long       reserved;          // reserved for future use
  long       offset;            // offset from the Bitmap File Header
  } BITMAPMAINHEADER;

typedef struct                  // Bitmap infomation header
{
  DWORD      size;              // size of the struct (must be 40 bytes)
  long       width;             // bitmap width
  long       height;            // bitmap height
  WORD       plane;             // color planes (must be set to 1)
  WORD       bitcount;          // bits per pixel
  DWORD      compression;       // 0: No compression; 1: run length (8
                 // bits per pixel); 2: run length (4 bits per pixel)
  DWORD      image_size;        // size of image in bytes
  long       xpelsmeter;        // horizontal resolution in pixels/meter
  long       ypelsmeter;        // vertical resolution in pixels/meter
  DWORD      clrused;           // count of colors
  DWORD      clrimportant;      // how many matter?
  } BITMAPINFOHEADER;

//------------------------
//   Tables of colors
//------------------------
typedef struct                          // table of RGB color
  {
  BYTE       blue;                      // value of blue color
  BYTE       green;                     // value of green color
  BYTE       red;                       // value of red color
  BYTE       reserved;                  // reserved for future use
  } BITMAPCOLORTABLE;

typedef struct                          // table of YCbCr color
  {
  float       Y;                        // value of Y
  float       Cb;                       // value of Cb
  float       Cr;                       // value of Cr
  } YCbCrCOLORTABLE;

//------------------------
//      JFIF File
//------------------------
typedef struct                  // JFIF header
```

```
{
WORD        soi;                // start of image: Hex FFD8
WORD        app;                // application segment 0: Hex FFE0
WORD        length;             // length includes these 2 bytes
                                // but not the previous 2
char        name_id[5];         // "JFIF": Hex 4A46494600
WORD        version;            // currently 1.01: Hex 0101
BYTE        units;              // 0: no units; 1: inch; 2: cm
WORD        xdensity;           // horizontal pixel density
WORD        ydensity;           // vertical pixel density
BYTE        xthumbnail;         // thumbnail horizontal pixel count
BYTE        ythumbnail;         // thumbnail vertical pixel count
} JFIFHEADER;
```

```
//=========================================================================
// JPEG_SUB.CPP
//=========================================================================

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "jpeg.h"

//------------------------
// Check type of inputfile
//------------------------
int CheckType(char *s)       // check file_id in header.
  {                          // Is it "BM"(Bitmap)?
  if(s[0]=='B' && s[1]=='M') return(0);
  return(1);
  }

//----------------------------------
// Change input file name * to *.sign
//----------------------------------
char* ChangeName(char* oldname,char* sign)
  {
  int a,b;
  char* newname;

  a=b=0;
  newname=(char *)malloc(13*sizeof(char));
  while(a<13 && (newname[a]=oldname[a])!='\0') a+=1;
  for(;a<13;a++)
    {
    oldname[a]='\0';
    newname[a]='\0';
    }
  while(newname[b]!='.' && b<a) b+=1;
if(b!=a || newname[b]=='\0')
{
newname[b]='.';newname[b+1]=sign[0];
newname[b+2]=sign[1];newname[b+3]=sign[2];
newname[b+4]='\0';
}
return(newname);
}

 //------------------------
 // Make full path of file
 //------------------------
 void FilePath(char* path,char* drive,char* dir,char* inputname)
   {
   int index1=0,index2=0;

   if(drive[0]!='\0')
     {
     path[0]=drive[0];
     path[1]=drive[1];
     index1+=2;
     }
```

```
    if(dir[0]!='\0')
    while(dir[index1-2]!='\0')
      {
      path[index1]=dir[index2];
      index1+=1; index2+=1;
      }
    index2=0;
    while(inputname[index2]!='\0')
      {
      path[index1]=inputname[index2];
      index1+=1; index2+=1;
      }
    path[index1]='\0';
    }

  //---------------------------------
  // Determind factor of quantization
  //---------------------------------
  int QuantizerFactor(char* option)
    {
    int factor;

    factor=atoi(&option[3]);
    if((factor<1 || factor >100) && factor!=0)
      {
      printf("\n Fail factor: %d (factor can be 0 to 100).\n",factor);
      exit(0);
      }
    if(factor==0)
      {
      printf("\n ERROR: I don't understand '%s'.\n",option);
      exit(0);
        }
    return(factor);
    };

  //-----------------------------------------
  // Convert RGB colortable to YCbCr colortable
  //-----------------------------------------
  void RGBtoYCbCrTable(BITMAPCOLORTABLE* RGB,YCbCrCOLORTABLE* YCbCr,
                       int number)
    {
    int index;
    float Y,Cb,Cr;

    // Y  =  0.29900 * R + 0.58700 * G + 0.11400 * B
    // Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B + 128.00000
    // Cr =  0.50000 * R - 0.41869 * G - 0.08131 * B + 128.00000
    for(index=0;index<number;index++)
      {

Y=(0.29900*(float)(RGB[index].red))+(0.58700*(float)(RGB[index].green))
      +(0.11400*(float)(RGB[index].blue));
if(Y<0.00) Y=0.00;        // saturate Y value (0.0<=Y<=255.0)
if(Y>255.00) Y=255.00;
Y-=128.00;                // level shifting by -128 before do Forward DCT
YCbCr[index].Y=Y;
```

```
Cb=((-0.16874)*(float)(RGB[index].red))
   (0.33126*(float)(RGB[index].green))+
   (0.50000*(float)(RGB[index].blue))+128.00000;
if(Cb<0.00)   Cb=0.00;   // Saturate Cr value (0.0<=Cr<=255.0)
if(Cb>255.00) Cb=255.00;
 Cb-=128.00;          // level shifting by -128 before do Forward DCT
YCbCr[index].Cb=Cb;
Cr=(0.50000*(float)(RGB[index].red))-  (0.41869*(float)
      (RGB[index].green))-(0.08131*(float)
      (RGB[index].blue))+128.00000;
if(Cr<0.00)   Cr=0.00;   // saturate Cb value (0.0<=Cb<=255.0)
if(Cr>255.00) Cr=255.00;
Cr-=128.00;          // level shifting by -128 before do Forward DCT
YCbCr[index].Cr=Cr;
}
}

 //---------------------
 // Convert RGB to YCbCr
 //---------------------
 void RGBtoYCbCr(BYTE*** buffer,float** Y,float** Cb,
      float** Cr,int width)
{
int row,col;
for(row=0;row<16;row++)
for(col=0;col<width;col++)
{
 Y[row][col]=(0.29900*(float)buffer[2][row][col])
                +(0.58700*(float)buffer[1][row][col])
                +(0.11400*(float)buffer[0][row][col]);
 if(Y[row][col]<0.00)   Y[row][col]=0.00;
 if(Y[row][col]>255.00) Y[row][col]=255.00;
 Y[row][col]-=128.00;   // level shifting by -128 before do Forward DCT

 Cb[row][col]=((-0.16874)*(float)buffer[2][row][col])
            -(0.33126*(float)buffer[1][row][col])
            +(0.50000*(float)buffer[0][row][col])+128.00000;
 if(Cb[row][col]<0.00)   Cb[row][col]=0.00;
 if(Cb[row][col]>255.00) Cb[row][col]=255.00;
 Cb[row][col]-=128.00;
 Cr[row][col]=(0.50000*(float)buffer[2][row][col])
              -(0.41869*(float)buffer[1][row][col])
              -(0.08131*(float)buffer[0][row][col])+128.00000;
 if(Cr[row][col]<0.00)   Cr[row][col]=0.00;
 if(Cr[row][col]>255.00) Cr[row][col]=255.00;
 Cr[row][col]-=128.00;
 }
}

 //-----------------------------------------------------
 // Read pixels form Bitmap file to store in picture buffer
 //-----------------------------------------------------
 void ReadPixels(int height,int width,long byte_row,long byte_end,
            WORD bit_pixel,int block_row,BYTE*** buffer,FILE*
            inputfile,char* outputfile)
 {
 int row,col;
 static long read_row;
```

```
    if(bit_pixel==1)
      {
      BYTE* tmp;
      tmp=(BYTE *)malloc((width+7)/8);
      for(row=0;row<height;row++)
        {
        fread(tmp,sizeof(BYTE),(width+7)/8,inputfile);
        fseek(inputfile,byte_row,SEEK_CUR);
        for(col=0;col<(width+7)/8;col++)
          {
          buffer[0][height-row-1][8*col]   =(tmp[col]&0x80)>>7;
          buffer[0][height-row-1][8*col+1]=(tmp[col]&0x40)>>6;
          buffer[0][height-row-1][8*col+2]=(tmp[col]&0x20)>>5;
          buffer[0][height-row-1][8*col+3]=(tmp[col]&0x10)>>4;
          buffer[0][height-row-1][8*col+4]=(tmp[col]&0x08)>>3;
          buffer[0][height-row-1][8*col+5]=(tmp[col]&0x04)>>2;
          buffer[0][height-row-1][8*col+6]=(tmp[col]&0x02)>>1;
          buffer[0][height-row-1][8*col+7]=tmp[col]&0x01;
          }
        }
    free(tmp);
     }

    if(bit_pixel==4)
    {
     BYTE* tmp;
     tmp=(BYTE *)malloc((width+1)/2);
     if(tmp==NULL)
     {
     printf(" ERROR : Could not allocate sufficient buffer storage.\n\n");
     remove(outputfile);
     exit(0);
     }
     for(row=0;row<height;row++)
     {
      fread(tmp,sizeof(BYTE),(width+1)/2,inputfile);
      fseek(inputfile,byte_row,SEEK_CUR);
      for(col=0;col<(width+1)/2;col++)
      {
       buffer[0][height-row-1][2*col]=(tmp[col]&0xF0)>>4;
       buffer[0][height-row-1][2*col+1]=tmp[col]&0x0F;
      }
     }
     free(tmp);
    }
    if(bit_pixel==8)
    {
     for(row=0;row<height;row++)
     {
      fread(buffer[0][height-row-1],sizeof(BYTE),width,inputfile);
      fseek(inputfile,byte_row,SEEK_CUR);
     }
    }
    if(bit_pixel==24)
    {
     BYTE* tmp;
     tmp=(BYTE *)malloc(3*width);
```

```
if(block_row==0) read_row=0L;
if(tmp==NULL)
{
printf(" ERROR : Could not allocate sufficient buffer storage.\n\n");
remove(outputfile);
exit(0);
}
for(row=0;row<16;row++)
{
read_row+=1L;
if((int)read_row<=height)
{
fseek(inputfile,-byte_end-(byte_row*read_row)
     -(3L*(long)width*read_row),SEEK_END);
fread(tmp,sizeof(BYTE),3*width,inputfile);
for(col=0;col<width;col++)
 {
  buffer[0][row][col]=tmp[3*col];
  buffer[1][row][col]=tmp[3*col+1];
  buffer[2][row][col]=tmp[3*col+2];
 }
}
else
{
 int tmp=1;
 for(col=0;col<width;col++)
 {
  buffer[0][row][col]=buffer[0][row-tmp][col];
  buffer[1][row][col]=buffer[1][row-tmp][col];
  buffer[2][row][col]=buffer[2][row-tmp][col];
 }
  tmp+=1;
}
if((width%16)!=0)
for(col=0;col<16;col++)
 {
  buffer[0][row][width+col]=buffer[0][row][width-1];
  buffer[1][row][width+col]=buffer[1][row][width-1];
  buffer[2][row][width+col]=buffer[2][row][width-1];
 }
 }
free(tmp);
}
}

 //---------------------------
 // Calculate DCT coefficients
 //---------------------------
 void CalculateCoeffDCT(float** coeff)
   {
      int row,col;
   double f1,f2,tmp;

   f1=M_PI/16.00000;  // M_PI = 3.14159265358979323846
   for(row=0;row<8;row++)
     {
      f2=(double)row*f1;
     for(col=0;col<8;col++)
```

```
        {
        tmp=cos((double)(2.00*col+1.00)*f2);
        coeff[row][col]=(float)tmp;
        }
      }
    }

  //---------------------------------------
  // Calculate a forward DCT on an 8x8 block
  //---------------------------------------
  void FDCT_Quantize_ZigZag(float** input,int* output,
                    float** dct_coeff,BYTE* quantizer)
{
int x,y,coeff;
float tmp1[8][8];
float tmp2[8][8];
static int order[64]={
                       0, 1, 5, 6,14,15,27,28,
                       2, 4, 7,13,16,26,29,42,
                       3, 8,12,17,25,30,41,43,
                       9,11,18,24,31,40,44,53,
                      10,19,23,32,39,45,52,54,
                      20,22,33,38,46,51,55,60,
                      21,34,37,47,50,56,59,61,
                      35,36,48,49,57,58,62,63
                      };

    //----- Do 1-dimemsional row DCTs -----//
for(y=0;y<8;y++)
for(coeff=0;coeff<8;coeff++)
{
 tmp1[y][coeff]=0.0;
 for(x=0;x<8;x++)
 {
  tmp1[y][coeff]+=((float)input[y][x])*dct_coeff[coeff][x];
 }
}
       //----- Do 1-dimensional column DCTs -----//
for(x=0;x<8;x++)
for(coeff=0;coeff<8;coeff++)
{
 tmp2[coeff][x]=0.0;
 for(y=0;y<8;y++)
 {
  tmp2[coeff][x]+=tmp1[y][x]*dct_coeff[coeff][y];
 }
}
       //----- Divide by coefficient -----//
for(y=0;y<8;y++)
{
for(x=0;x<8;x++)
 {
  if(x==0) tmp2[y][x]/=sqrt(2.00);
  if(y==0) tmp2[y][x]/=sqrt(2.00);
  tmp2[y][x]/=4.00;
  if(tmp2[0][0]>32767.00)
     tmp2[0][0]=32767.00;     // saturate value
  if(tmp2[0][0]<(-32767.00)) tmp2[0][0]=(-32767.00);
```

```
        //----- Quanitization process -----//
  tmp2[y][x]/=(double)quantizer[(y*8)+x];
        //----- Zigzag process -----//
  output[order[(y*8)+x]]=ROUND(tmp2[y][x]);
   }
  }
 }

 //-------------------------------------
 // Fing a magnitude category of number
 //-------------------------------------
 int GetCategory(int number)
      {
      int absolute=abs(number);
      int cat=11;
      int bit=0x0400;
      if(absolute==0) return(0);
      while((absolute&bit)==0)
        {
        cat-=1;
        bit>>=1;
        }
      return(cat);
      }

 //-----------------------------------------------------------------
 // Store a variable-length Huffman code or magnitude in a buffer
 //-----------------------------------------------------------------
 long PackBits(WORD symbol,int length,BYTE **output,long bit_count)
 {
  long output_bytes=bit_count/8L;
  int  bits_used=(int)(bit_count-8L*output_bytes);
  int  bits_remaining=8-bits_used;
  int  row=(int)(output_bytes/10000L);
  int  col=(int)(output_bytes-(long)row*10000L);
  int  upper_bits;
  int  lower_bits;
  BYTE upper_byte;
  BYTE lower_byte;

  upper_byte=(BYTE)(symbol>>8);
  lower_byte=(BYTE)(symbol&0xFF);
  lower_bits=0;
  upper_bits=length;
  if(upper_bits>8)
  {
  lower_bits=upper_bits-8;
  upper_bits=8;
  }
  output[row][col]|=(upper_byte>>bits_used);  // store upper bits
  bits_used+=upper_bits;
  if(bits_used>=8)
  {
   bits_used-=8;
   if((output[row][col]==0xFF))
     {
     col+=1;
```

```
      if(col==10000) { row+=1; col=0; }
    output[row][col]=0x00;
    bit_count+=8L;
   }
    col+=1;
   if(col==10000) { row+=1; col=0; }
   output[row][col]|=(upper_byte<<bits_remaining);
   bits_remaining=8-bits_used;
   }
if(lower_bits>0)                    // store lower bits
   {
   output[row][col]|=(lower_byte>>bits_used);
   bits_used+=lower_bits;
   if(bits_used>=8)
   {
    if((output[row][col]==0xFF))
     {
       col+=1;
       if(col==10000) { row+=1; col=0; }
         output[row][col]=0x00;
         bit_count+=8L;
     }
  bits_used-=8;
  col+=1;
  if(col==10000) { row+=1; col=0; }
  output[row][col]|=(lower_byte<<bits_remaining);
  if((output[row][col]==0xFF))
   {
    col+=1;
    if(col==10000) { row+=1; col=0; }
       output[row][col]=0x00;
       bit_count+=8L;
   }
  }
}
bit_count+=(long)length;
return(bit_count);
}

  //-------------------------------------------------------
  // Encode a block of DCT coefficients using Huffman coding
  //-------------------------------------------------------
  long HuffmanEncode(int* dc,int *input,BYTE **output,long bit_count)
       {
       static int AC_len[256]=
         {
              4, 2, 2, 3, 4, 5, 6, 7,10,16,16, 0, 0, 0, 0, 0,
              0, 4, 6, 7, 9,11,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 5, 8,10,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 6, 9,11,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 6,10,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 7,10,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 7,11,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 8,11,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 8,15,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 9,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 9,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
              0, 9,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
```

```
            0,10,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
            0,11,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
            0,12,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0,
           12,16,16,16,16,16,16,16,16,16,16, 0, 0, 0, 0, 0
      };
    static WORD AC_code[256]=
      {
      0xA000,0x0000,0x4000,0x8000,0xB000,0xD000,0xE000,0xF000,
      0xFD80,0xFF82,0xFF83,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xC000,0xE400,0xF200,0xFB00,0xFEC0,0xFF84,0xFF85,
      0xFF86,0xFF87,0xFF88,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xD800,0xF800,0xFDC0,0xFF89,0xFF8A,0xFF8B,0xFF8C,
      0xFF8D,0xFF8E,0xFF8F,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xE800,0xFB80,0xFEE0,0xFF90,0xFF91,0xFF92,0xFF93,
      0xFF94,0xFF95,0xFF96,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xEC00,0xFE00,0xFF97,0xFF98,0xFF99,0xFF9A,0xFF9B,
      0xFF9C,0xFF9D,0xFF9E,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xF400,0xFE40,0xFF9F,0xFFA0,0xFFA1,0xFFA2,0xFFA3,
      0xFFA4,0xFFA5,0xFFA6,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xF600,0xFF00,0xFFA7,0xFFA8,0xFFA9,0xFFAA,0xFFAB,
      0xFFAC,0xFFAD,0xFFAE,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xF900,0xFF20,0xFFAF,0xFFB0,0xFFB1,0xFFB2,0xFFB3,
      0xFFB4,0xFFB5,0xFFB6,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFA00,0xFF80,0xFFB7,0xFFB8,0xFFB9,0xFFBA,0xFFBB,
      0xFFBC,0xFFBD,0xFFBE,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFC00,0xFFBF,0xFFC0,0xFFC1,0xFFC2,0xFFC3,0xFFC4,
      0xFFC5,0xFFC6,0xFFC7,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFC80,0xFFC8,0xFFC9,0xFFCA,0xFFCB,0xFFCC,0xFFCD,
      0xFFCE,0xFFCF,0xFFD0,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFD00,0xFFD1,0xFFD2,0xFFD3,0xFFD4,0xFFD5,0xFFD6,
      0xFFD7,0xFFD8,0xFFD9,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFE80,0xFFDA,0xFFDB,0xFFDC,0xFFDD,0xFFDE,0xFFDF,
      0xFFE0,0xFFE1,0xFFE2,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFF40,0xFFE3,0xFFE4,0xFFE5,0xFFE6,0xFFE7,0xFFE8,
      0xFFE9,0xFFEA,0xFFEB,0x0000,0x0000,0x0000,0x0000,0x0000,
      0x0000,0xFF60,0xFFEC,0xFFED,0xFFEE,0xFFEF,0xFFF0,0xFFF1,
      0xFFF2,0xFFF3,0xFFF4,0x0000,0x0000,0x0000,0x0000,0x0000,
      0xFF70,0xFFF5,0xFFF6,0xFFF7,0xFFF8,0xFFF9,0xFFFA,0xFFFB,
      0xFFFC,0xFFFD,0xFFFE,0x0000,0x0000,0x0000,0x0000,0x0000
      };
satic WORD DC_code[12]=
      {
      0x4000,0x6000,0x8000,0x0000,0xA000,0xC000,
      0xE000,0xF000,0xF800,0xFC00,0xFE00,0xFF00
      };
    static WORD bit_mask[12]=
      {
      0x0000,0x0001,0x0003,0x0007,0x000F,0x001F,
      0x003F,0x007F,0x00FF,0x01FF,0x03FF,0x07FF
      };
       int src=0;                  // source index
       int end;                    // end of buffer pointer
       int run;                    // run length
       int cat;                    // category
       int mag;                    // magnitude

       cat=GetCategory(input[src]-(*dc));
       bit_count=PackBits(DC_code[cat],DC_len[cat],output,bit_count);
```

```
        mag=input[src]-(*dc);
        *dc=input[src];


        src+=1;
        if(mag<0) mag-=1;
        mag&=bit_mask[cat];
        mag=mag<<(16-cat);
        bit_count=PackBits(mag,cat,output,bit_count);
        end=63;
        while(end>=1 && input[end]==0) end-=1;
        while(src<=end)
          {
          run=0;
          while(src<end && input[src]==0 && run<16)
               {
               run+=1;
               src+=1;
               }
          if(run==16) cat=0x00F0;
          else
               {
               cat=GetCategory(input[src]);
               cat|=((run&0x000F)<<4);
               }
          bit_count=PackBits(AC_code[cat],AC_len[cat],output,bit_count);

          if(cat!=0x00F0)
               {
               cat&=0x000F;
               mag=input[src];
               src+=1;
               if(mag<0) mag-=1;
               mag&=bit_mask[cat];
               mag=mag<<(16-cat);
               bit_count=PackBits(mag,cat,output,bit_count);
               }
          }
        if(src!=64)
  bit_count=PackBits(AC_code[0],AC_len[0],output,bit_count);
        return(bit_count);
        }


    //---------------------------------------
    // Initilize all of output buffer to zero
    //---------------------------------------
    void Initilize(BYTE** output,int mem_row)
        {
        int row,col;

        for(row=0;row<mem_row;row++)
        for(col=0;col<10000;col++)
        output[row][col]=0;
        }
```

```
//-------------------------------------
// Write compressed codes to output file
//-------------------------------------
long WriteCode(int mem_row,BYTE** output,long bit_count,
               FILE* outputfile,char* filename)
  {
BYTE tmp;
int  index=0;
long bytes;
int  bits_used,row_output,col_output;
bytes=bit_count/8L;
bits_used=(int)(bit_count-8L*bytes);
row_output=(int)(bytes/10000L);
col_output=(int)(bytes-(long)row_output*10000L);


for(index=0;index<row_output;index++)
if(fwrite(output[index],sizeof(BYTE),10000,outputfile)!=10000)
     {
     printf("\n\n ERROR2 : Disk full \n");
     remove(filename);
     exit(0);
     }

if(fwrite(output[index],sizeof(BYTE),col_output,outputfile)!=col_output)
     {
     printf("\n\n ERROR3 : Disk full \n");
     remove(filename);
     exit(0);
     }
   tmp=output[row_output][col_output];
   Initilize(output,mem_row);
   output[0][0]=tmp;
   bit_count=(long)bits_used;
   return(bit_count);
   }


//------------------------
// Write data of JFIF data
//------------------------
void WriteJFIFheader(BYTE* luma,BYTE* chroma,JFIFHEADER* header,
     BITMAPINFOHEADER* bmpinfo,FILE* outputfile)
   {
   int marker[2];
   BYTE tmp[6];

         //----- Write JFIF Header -----//
header->name_id[0]='J';
header->name_id[1]='F'; header->name_id[2]='I';
header->name_id[3]='F'; header->name_id[4]='\0';
header->soi=0xD8FF;header->app=0xE0FF;header->length=4096;
header->version=0x0101;header->units='\0';header->xdensity=0;
header->ydensity=0;header->xthumbnail=0;header->ythumbnail=0;
fwrite(header,sizeof(JFIFHEADER),1,outputfile);
```

```
        //----- Write Quantization Table ------//
marker[0]=0xDBFF;     // quantization table marker
marker[1]=132<<8;     // quantization table length
fwrite(marker,sizeof(int),2,outputfile);
putc(0x00,outputfile); // table precision,identifier of luma_quantizer
fwrite(luma,sizeof(BYTE),64,outputfile);    // luma_quantizer
putc(0x01,outputfile); // table precision,identifier of chroma_quantizer
fwrite(chroma,sizeof(BYTE),64,outputfile); // chroma_quantizer


        //----- Write Frame Header ------//
marker[0]=0xC0FF;     // start of frame
marker[1]=17<<8;      // frame header length
fwrite(marker,sizeof(int),2,outputfile);
putc(0x08,outputfile);          // sample precision
marker[0]=(int)bmpinfo->height;
marker[1]=(int)bmpinfo->width;
tmp[0]=(BYTE)(marker[0]>>8);  // number of lines
tmp[1]=(BYTE)(marker[0]&0x00FF);
tmp[2]=(BYTE)(marker[1]>>8);  // sample per line
tmp[3]=(BYTE)(marker[1]&0x00FF);
fwrite(tmp,sizeof(BYTE),4,outputfile);
putc(0x03,outputfile);          // number of image components
tmp[0]=0x01;  // Y's identifier
tmp[1]=0x22;  // horizontal and vertical samping factor h:1 v:1
tmp[2]=0x00;  // quantization table selector (use luma_quantizer)
fwrite(tmp,sizeof(BYTE),3,outputfile);
tmp[0]=0x02;  // Cb's identifier
tmp[1]=0x11;  // horizontal and vertical samping factor h:1 v:1
tmp[2]=0x01;  // quantization table selector (use chroma_quantizer)
fwrite(tmp,sizeof(BYTE),3,outputfile);
tmp[0]=0x03;  // Cr's identifier
tmp[1]=0x11;  // horizontal and vertical samping factor h:1 v:1
tmp[2]=0x01;  // quantization table selector (use chroma_quantizer)
fwrite(tmp,sizeof(BYTE),3,outputfile);


        //----- Write Huffcode Table -----//
            // DC Huffman Table define  //
marker[0]=0xC4FF;     // define Huffman table marker
marker[1]=0x1F00;     // Huffman table definition length
fwrite(marker,sizeof(int),2,outputfile);
putc(0x00,outputfile); // Huffman table DC:0,identifier:0
BYTE DC_bitlength[16] = { 0,1,5,1,1,1,1,1,1,0,0,0,0,0,0,0 };
BYTE DC_code[12] = {
                  0x0003,0x0000,0x0001,0x0002,0x0004,0x0005,
                  0x0006,0x0007,0x0008,0x0009,0x000A,0x000B
                   };
fwrite(DC_bitlength,sizeof(BYTE),16,outputfile);
fwrite(DC_code,sizeof(BYTE),12,outputfile);


         // AC Huffman Table define
marker[0]=0xC4FF;     // define Huffman table marker
marker[1]=0xB500;     // Huffman table definition length
fwrite(marker,sizeof(int),2,outputfile);
putc(0x10,outputfile); // Huffman table AC:1,identifier:0
BYTE AC_bitlength[16] = { 0,2,1,3,2,4,4,3,5,5,5,2,0,0,1,125 };
```

```
BYTE AC_code[162] =
    {
    0x01,0x02,0x03,0x00,0x04,0x11,0x05,0x21,0x06,
    0x12,0x31,0x41,0x07,0x13,0x51,0x61,0x22,0x71,
    0x81,0x14,0x32,0x91,0xA1,0xB1,0x08,0x23,0x42,
    0x52,0xC1,0x15,0x33,0x62,0x72,0xD1,0xE1,0xF0,
    0x82,0x09,0x0A,0x16,0x17,0x18,0x19,0x1A,0x24,
    0x25,0x26,0x27,0x28,0x29,0x2A,0x34,0x35,0x36,
    0x37,0x38,0x39,0x3A,0x43,0x44,0x45,0x46,0x47,
    0x48,0x49,0x4A,0x53,0x54,0x55,0x56,0x57,0x58,
    0x59,0x5A,0x63,0x64,0x65,0x66,0x67,0x68,0x69,
    0x6A,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A,
    0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x92,
    0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0xA2,
    0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xB2,
    0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xC2,
    0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xD2,
    0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xE2,
    0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xF1,
    0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA
    };
    fwrite(AC_bitlength,sizeof(BYTE),16,outputfile);
    fwrite(AC_code,sizeof(BYTE),162,outputfile);

    //----- Write Scan Header Fields -----//
    marker[0]=0xDAFF;     // start of scan marker
    marker[1]=0x0C00;     // scan header length
    fwrite(marker,sizeof(int),2,outputfile);
    putc(0x03,outputfile); // number of image components
    tmp[0]=0x01; // scan component selector to Y scan pixels
    tmp[1]=0x00; // DC,AC coding table selector DC=0 AC=0
    tmp[2]=0x02; // scan component selector to Cb scan pixels
    tmp[3]=0x00; // DC,AC coding table selector DC=0 AC=0
    tmp[4]=0x03; // scan component selector to Cr scan pixels
    tmp[5]=0x00; // DC,AC coding table selector DC=0 AC=0
    fwrite(tmp,sizeof(BYTE),6,outputfile);
    tmp[0]=0x00; // start of spectral selection: 0
    tmp[1]=0x3F; // end of spectral selection: 63
    tmp[2]=0x00; // successive approximation bit high:0 and low:0
    fwrite(tmp,sizeof(BYTE),3,outputfile);
}
```

Table 6.16 JPEG Default AC Code (Luminance)

Run/ Category	Base Code	Length	Run/ Category	Base Code	Length
0/0	1010 (= EOB)	4			
0/1	00	3	8/1	11111010	9
0/2	01	4	8/2	1111111111000000	17
0/3	100	6	8/3	1111111110110111	19
0/4	1011	8	8/4	1111111110111000	20
0/5	11010	10	8/5	1111111110111001	21
0/6	111000	12	8/6	1111111110111010	22
0/7	1111000	14	8/7	1111111110111011	23
0/8	1111110110	18	8/8	1111111110111100	24
0/9	1111111110000010	25	8/9	1111111110111101	25
0/A	1111111110000011	26	8/A	1111111110111110	26
1/1	1100	5	9/1	111111000	10
1/2	111001	8	9/2	1111111110111111	18
1/3	1111001	10	9/3	1111111111000000	19
1/4	111110110	13	9/4	1111111111000001	20
1/5	11111110110	16	9/5	1111111111000010	21
1/6	1111111110000100	22	9/6	1111111111000011	22
1/7	1111111110000101	23	9/7	1111111111000100	23
1/8	1111111110000110	24	9/8	1111111111000101	24
1/9	1111111110000111	25	9/9	1111111111000110	25
1/A	1111111110001000	26	9/A	1111111111000111	26
2/1	11011	6	A/1	111111001	10
2/2	11111000	10	A/2	1111111111001000	18
2/3	1111110111	13	A/3	1111111111001001	19
2/4	1111111110001001	20	A/4	1111111111001010	20
2/5	1111111110001010	21	A/5	1111111111001011	21
2/6	1111111110001011	22	A/6	1111111111001100	22
2/7	1111111110001100	23	A/7	1111111111001101	23
2/8	1111111110001101	24	A/8	1111111111001110	24
2/9	1111111110001110	25	A/9	1111111111001111	25
2/A	1111111110001111	26	A/A	1111111111010000	26
3/1	111010	7	B/1	111111010	10
3/2	111110111	11	B/2	1111111111010001	18
3/3	11111110111	14	B/3	1111111111010010	19
3/4	1111111110010000	20	B/4	1111111111010011	20
3/5	1111111110010001	21	B/5	1111111111010100	21
3/6	1111111110010010	22	B/6	1111111111010101	22
3/7	1111111110010011	23	B/7	1111111111010110	23
3/8	1111111110010100	24	B/8	1111111111010111	24
3/9	1111111110010101	25	B/9	1111111111011000	25
3/A	1111111110010110	26	B/A	1111111111011001	26

Table 6.16 Continued

Run/ Category	*Base Code*	*Length*	*Run/ Category*	*Base Code*	*Length*
4/1	111011	7	C/1	1111111010	11
4/2	1111111000	12	C/2	1111111111011010	18
4/3	1111111110010111	19	C/3	1111111111011011	19
4/4	1111111110011000	20	C/4	1111111111011100	20
4/5	1111111110011001	21	C/5	1111111111011101	21
4/6	1111111110011010	22	C/6	1111111111011110	22
4/7	1111111110011011	23	C/7	1111111111011111	23
4/8	1111111110011100	24	C/8	1111111111100000	24
4/9	1111111110011101	25	C/9	1111111111100001	25
4/A	1111111110011110	26	C/A	1111111111100010	26
5/1	1111010	8	D/1	11111111010	12
5/2	1111111001	12	D/2	1111111111100011	18
5/3	1111111110011111	19	D/3	1111111111100100	19
5/4	1111111110100000	20	D/4	1111111111100101	20
5/5	1111111110100001	21	D/5	1111111111100110	21
5/6	1111111110100010	22	D/6	1111111111100111	22
5/7	1111111110100011	23	D/7	1111111111101000	23
5/8	1111111110100100	24	D/8	1111111111101001	24
5/9	1111111110100101	25	D/9	1111111111101010	25
5/A	1111111110100110	26	D/A	1111111111101011	26
6/1	1111011	8	E/1	111111110110	13
6/2	11111111000	13	E/2	1111111111101100	18
6/3	1111111110100111	19	E/3	1111111111101101	19
6/4	1111111110101000	20	E/4	1111111111101110	20
6/5	1111111110101001	21	E/5	1111111111101111	21
6/6	1111111110101010	22	E/6	1111111111110000	22
6/7	1111111110101011	23	E/7	1111111111110001	23
6/8	1111111110101100	24	E/8	1111111111110010	24
6/9	1111111110101101	25	E/9	1111111111110011	25
6/A	1111111110101110	26	E/A	1111111111110100	26
7/1	11111001	9	F/0	111111110111	12
7/2	11111111001	13	F/1	1111111111110101	17
7/3	1111111110101111	19	F/2	1111111111110110	18
7/4	1111111110110000	20	F/3	1111111111110111	19
7/5	1111111110110001	21	F/4	1111111111111000	20
7/6	1111111110110010	22	F/5	1111111111111001	21
7/7	1111111110110011	23	F/6	1111111111111010	22
7/8	1111111110110100	24	F/7	1111111111111011	23
7/9	1111111110110101	25	F/8	1111111111111100	24
7/A	1111111110110110	26	F/9	1111111111111101	25
			F/A	1111111111111110	26

บรรณานุกรม

[1] Rafael C. Gonzales and Richard E. Woods, "Digital Image Processing", Addison-Wesley Publishing Company, 1992

[2] Phillip E Mattison, "Practical Digital Video with Programming Example in C", John Wiley & Son, Inc., 1994

[3] Nelson M., "The Data Compression Book", MAT publishing Company, Inc., 1991

[4] C.Wayne Brown and Barry J. Shepherd,"Graphics File Format", Manning Publications Cio., 1995

[5] ชัยรัตน์ ฤทธิรงค์, "การบีบอัดข้อมูลทางการแพทย์โดยการ modulate แบบบ ADPCM", วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิตสาขาวิศวกรรมไฟฟ้า, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2539

[6] ธนา ศรีแสง, "ระบบควบคุมการติดตามอัตโนมัติโดยข้อมูลภาพ", วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต, สาขาวิศวกรรมไฟฟ้า, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2535