การถอดรหัสข้อมูลMPEG

MPEG DECODER

โดย
นายกิติชัย แซ่ลิ้ม
นายจักรพงษ์ ศิริพลตั้น
นางสาวมณีรัตน์ ศรีอินทร์

ปริญญานิพนธ์เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

การถอดรหัสข้อมูลMPEG

MPEG DECODER

โดย

นายกิติชัย แซ่ลิ้ม 38014030

นายจักรพงษ์ ศิริพลตั้น 38014059

นางสาวมณีรัตน์ ศรีอินทร์ 38014385

อาจารย์ที่ปรึกษา

ผศ.ดร.สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

การถอดรหัสข้อมูลMPEG

MPEG DECODER

1. นายกิติชัย แซ่ลิ้ม เลขประจำตัว 38014030

2.นายจักรพงษ์ ศิริพลตั้น เลขประจำตัว 38014059

3. นางสาวมณีรัตน์ ศรีอินทร์ เลขประจำตัว 38014385

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(ผศ.ดร.สมศักดิ์ ชุมช่วย)

อาจารย์ที่ปรึกษา

ปริญญานิพนธ์ ปีการศึกษา 2541

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การถอดรหัสข้อมูลMPEG

ผู้จัดทำ

1. นายกิติชัย แซ่ลิ้ม เลขประจำตัว 38014030
2. นายจักรพงษ์ ศิริพลตั้น เลขประจำตัว 38014059
3. นางสาวมณีรัตน์ ศรีอินทร์ เลขประจำตัว 38014385

..... อาจารย์ที่ปรึกษา

(ผศ.ดร.สมศักดิ์ ชุมช่วย)

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี ซึ่งได้รับความช่วยเหลือดูแลจากผู้มีพระคุณต่างๆ ทางคณะผู้จัดทำจึงขอแสดงความขอบคุณทุกท่านที่ได้ให้ความช่วยเหลือแก่ผู้จัดทำ ซึ่งได้แก่ ผศ.ดร. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษาโครงงาน,รุ่นพี่ปริญญาโทที่อยู่ห้องร่วมทำงานด้วยกัน ซึ่งให้คำแนะนำแก่ผู้จัดทำ รวมทั้งให้ยืมใช้คอมพิวเตอร์,เพื่อนๆ ในภาค ที่ให้กำลังใจ และช่วยเหลืองานบางอย่าง,รุ่นน้องที่ชมรมว่ายน้ำที่ให้กำลังใจแก่ผู้จัดทำ ทางคณะผู้จัดทำจึงขอขอบคุณมา ณ ที่นี้

ผู้จัดทำ

การถอดรหัสข้อมูล MPEG

นายกิติชัย แซ่ลิ้ม
นายจักรพงษ์ ศิริพลตั้น
นางสาวมณีรัตน์ ศรีอินทร์
ปีการศึกษา 2541

บทคัดย่อ

ในปัจจุบันนี้มนุษย์มีความต้องการในงานทางด้านมัลติมีเดียมากขึ้น ดังนั้นจึงจำเป็นที่จะต้องมีการพัฒนาเทคนิคใหม่ๆขึ้นมา เพื่อตอบสนองความต้องการดังกล่าวให้มากที่สุด ในขณะเดียวกันรูปแบบและระเบียบวิธีการในการประมวลผลรหัสดิจิตอลก็เกิดขึ้นเช่นเดียวกัน โดยที่ผู้ผลิตได้พยายามคิดค้นหาวิธีการใหม่ๆขึ้นมาอย่างมากมาย เพื่อให้เทคโนโลยีของตนตอบสนองความต้องการของผู้บริโภคให้มากที่สุดเท่าที่จะเป็นไปได้ ดังนั้นจึงต้องมีการกำหนดมาตรฐานขึ้นมาเพื่อความเข้ากันได้ของระบบที่ต่างกัน มาตรฐานของ MPEG (Moving Picture Expert Group) จึงเป็นมาตรฐานที่เราให้ความสนใจอยู่ในขณะนี้

โครงงานนี้เป็นโครงงานที่สนับสนุนการถอดรหัสข้อมูลตามมาตรฐาน MPEG-1 ซึ่งพื้นฐานของมาตรฐานนี้พัฒนามาจากมาตรฐานการบีบอัดข้อมูลภาพนิ่งตามมาตรฐาน JPEG โดยจะประกอบไปด้วยการบีบอัดข้อมูลแบบสูญเสียและไม่สูญเสีย และยิ่งไปกว่านั้นในมาตรฐาน MPEG เราจะต้องเพิ่มกระบวนการการประมาณค่าการเคลื่อนที่และการทดแทนการเคลื่อนที่เข้าไปด้วย ซึ่งจะต้องมีการหาค่าเวคเตอร์การเคลื่อนที่ก่อน สำหรับโครงงานนี้เราใช้ภาษา C++ ในการถอดรหัสข้อมูล จากการเรียนรู้ในโครงงานของเรา สิ่งที่เรานำเสนอนั้นเป็นสิ่งที่ยอมรับได้เนื่องจากสิ่งใหม่ๆทั้งหลายที่เกิดขึ้น เช่น มาตรฐานและเทคนิคในการเขียนโปรแกรมได้รับการทดลองและพิสูจน์ให้เห็นจริงแล้ว

MPEG DECODER

Mr.Kitichai Saelim

Mr.Chakapong Siripolton

Miss Maneerat Sriin

1998

ABSTRACT

In recent years there is an increasing of the requirements in multimedia information. It is reasonable to develop the new techniques to support these requirements as much as possible. Simultaneously, it is appeared to be many new digital data processing formats and algorithm that the manufacturers try to get many new methods for their technology to support the requirements of the consumers as much as possible. So compatibility to the standard is necessary. A standard of MPEG (Moving Picture Expert Group) is one topic that we are interesting.

This project is to implement an MPEG-1 decoder. The basic for MPEG standard is developed from JPEG standard which is composed of lossy compression and lossless compression. In addition in MPEG standard, we must add the process for search motion compensation and motion estimation that we must search motion vector before. In this project we use C++ language for code implementation. In our studies, what we have obtained is quite accepted since a number of new things such as standard, and programming technique has been experienced.

สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 บทนำ

ในปัจจุบันคอมพิวเตอร์ได้มีการพัฒนาไปอย่างกว้างขวาง ทั้งในด้านฮาร์ดแวร์(Hardware) และด้านซอฟต์แวร์(Software) รวมทั้งเทคโนโลยีมัลติมีเดียก็ได้มีการพัฒนาอย่างกว้างมาก ในสมัยที่ยังไม่มีเทคโนโลยีการสร้างแผ่นซีดีรอม(CD-ROM) มนุษย์ก็คงได้แต่เพ้อฝันว่าจะสามารถดูภาพยนตร์ หรือฟังเพลงทางคอมพิวเตอร์ได้ จนกระทั่งมีคนคิดค้นแผ่นซีดีรอมขึ้นมา ซึ่งมีความจุถึง 600 เมกะไบต์(1ไบต์ มีค่าเท่ากับ 8บิท) และยิ่งในปัจจุบันได้มีแผ่นดีวีดี(DVD;Digital video disk) ซึ่งมีความจุประมาณ 2 ไบต์ไบต์(10^9 ไบต์) ทำให้มนุษย์ได้มีการคิดค้นวิธีการเก็บลงแผ่นซีดีรอมหรือดีวีดี ทำให้มนุษย์สามารถใช้งานคอมพิวเตอร์ได้มากขึ้น โดยเฉพาะงานทางด้านมัลติมีเดียนั้น ได้มีการแข่งขันกันอย่างมาก ได้มีการพัฒนาซอฟต์แวร์ในการบีบอัดข้อมูลให้ได้มากที่สุด เพื่อที่จะสามารถเก็บข้อมูล หรือรายละเอียดของภาพยนตร์และเสียงให้ได้มากที่สุด

มาตรฐานของ ISO-11172 Motion Picture Experts Group หรือ MPEG เป็นมาตรฐานในการบีบอัดข้อมูลวีดีโอเพื่อเก็บข้อมูลลงในแผ่นซีดีรอมหรือ เก็บลงในคอมพิวเตอร์ มาตรฐานนี้มีไว้เพื่อที่จะให้เราสามารถที่จะเก็บข้อมูลหรือเปิดข้อมูลวีดีโอในระบบที่แตกต่างกันได้ ซึ่งในปัจจุบันก็ได้มีการพัฒนามาตรฐาน MPEG เรื่อยๆ เช่น MPEG-1,MPEG-2 และ MPEG-3 ในโครงงานนี้ก็ได้ทำการเขียนโปรแกรมการถอดรหัสข้อมูลให้ตรงตามมาตรฐาน MPEG-1 เหตุที่เราเขียนโปรแกรมโดยใช้ข้อกำหนดมาตรฐาน MPEG-1 เพราะว่า มาตรฐานของ MPEG-2 และ MPEG-3 นั้นมีความสลับซับซ้อนในการทำงานมาก ทำให้ยุ่งยากต่อการออกแบบและคำนวณ

1.2 วัตถุประสงค์ของโครงงาน

โครงงานนี้เป็นการเขียนโปรแกรมการบีบอัดข้อมูลให้ตรงตามมาตรฐาน MPEG-1 โดยใช้ภาษา C++ ในการเขียนโปรแกรม โดยมีจุดประสงค์ดังนี้

1. ศึกษารูปแบบวิธีการแสดงภาพและวีดีโอ
2. ศึกษาวิธีการบีบอัดข้อมูลภาพและวีดีโอ
3. ศึกษาการเข้า/ถอดรหัส โดยวิธีต่างๆ ให้ตรงตามมาตรฐาน MPEG
4. เขียนโปรแกรมที่สามารถเก็บข้อมูลวีดีโอให้อยู่ในรูปแบบของ MPEG

1.3 ขอบเขตโครงงาน

ขอบเขตของโครงงานนี้คือการเขียนโปรแกรมที่สามารถถอดแฟ้มข้อมูลวีดีโอให้ได้ตามมาตรฐานของ MPEG-1 โดยใช้ภาษา C++ โดยจะแบ่งขอบเขตการทำงานตามภาคเรียนการศึกษาดังนี้

1.3.1 การศึกษาภาคเรียนที่ 1

- ศึกษาวิธีการแสดงภาพนิ่งและวีดีโอ,การบีบอัดข้อมูล,การเข้ารหัสและถอดรหัสข้อมูล, การบีบอัดข้อมูลแบบสูญเสียและไม่สูญเสีย , การแปลงข้อมูล DCT(Discrete Cosine Transform), การเข้ารหัสฮัฟแมน,การเข้ารหัสrun-length,การแปลงสี พิจารณาการบีบข้อมูลที่มีประสิทธิภาพดีที่สุด รวมทั้งศึกษาการทำงานแผนผังการทำงานของการบีบอัดข้อมูล ,การเข้ารหัสและถอดรหัสข้อมูล โดยวิธีการต่างๆ

- ศึกษาการหาเวคเตอร์การเคลื่อนที่(motion vector)ด้วยวิธีการต่างๆ รวมทั้งเข้าใจแผนผังการทำงานการหาเวคเตอร์การเคลื่อนที่ และพิจารณาว่าวิธีใดที่ให้ประสิทธิภาพที่ดีที่สุด และเข้าใจการทำงานระหว่างการทำงานของภาพนิ่งกับการทำงานของภาพเคลื่อนไหว

- ศึกษามาตรฐานของ JPEG(Joint Photography Experts Group ซึ่งเป็นมาตรฐานการบีบอัดข้อมูลของภาพนิ่ง และมาตรฐาน MPEG ซึ่งเป็นมาตรฐานการบีบอัดข้อมูลของภาพเคลื่อนไหวหรือวีดีโอ

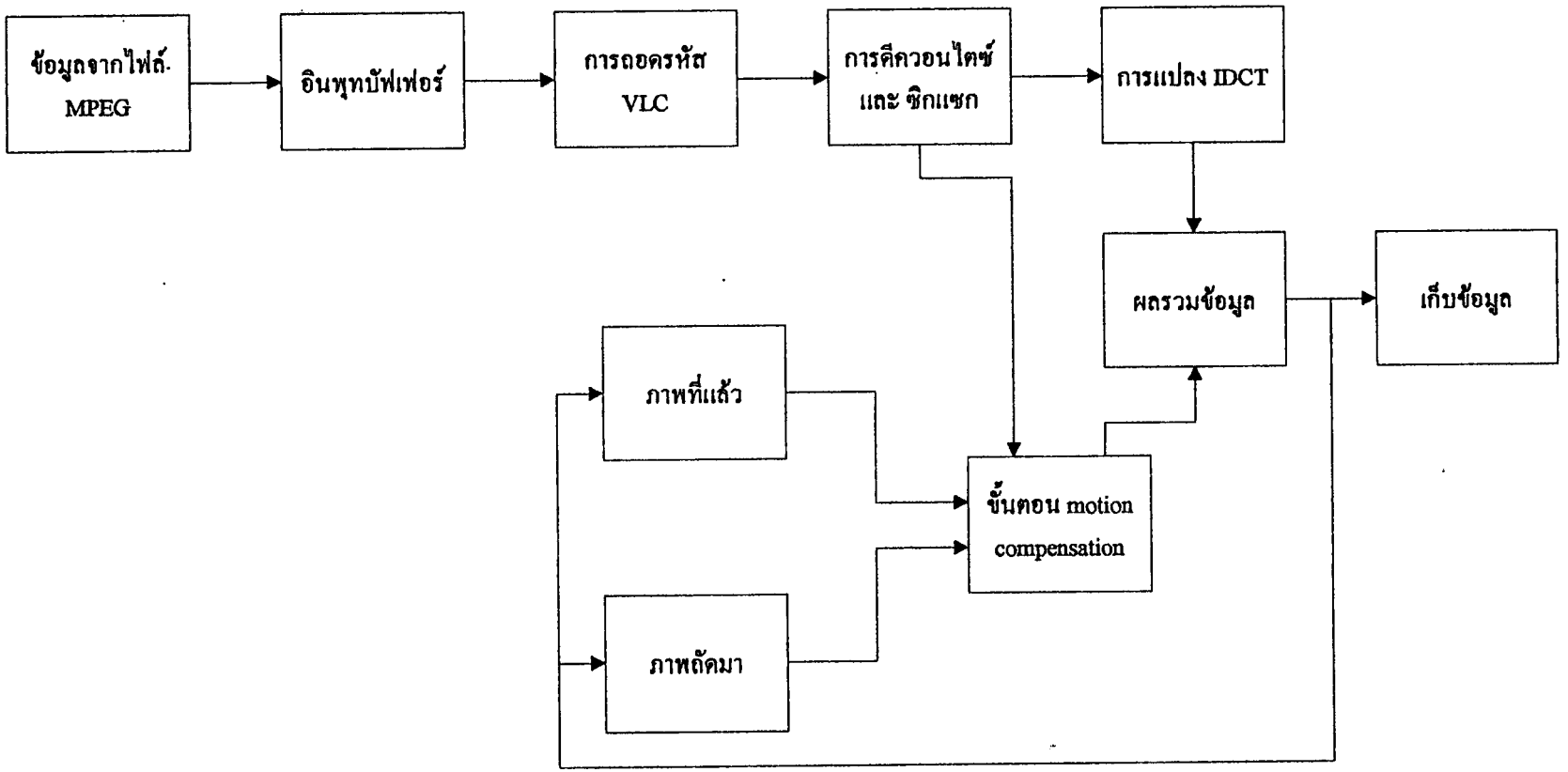
- ศึกษาภาษา C++ ซึ่งเป็นภาษาเราที่เราจะใช้เขียนโปรแกรมนี้ เหตุที่ใช้ภาษา C++ เพราะเป็นภาษาที่ใช้กันอยู่ทั่วไป และมีประสิทธิภาพบางอย่างที่ดีกว่า C หรือ ปาสคาล

1.3.2 การศึกษาภาคเรียนที่ 2

- ทำการเขียนโปรแกรมถอดข้อมูลของแฟ้มข้อมูล MPEG
- เขียนโปรแกรมการแสดงภาพที่เราได้ถอดข้อมูลมาแล้ว
- ทดลองการทำงานโปรแกรมที่เขียน

เราสามารถเขียนแผนภาพขั้นตอนการทำงานการถอดรหัสข้อมูลไฟล์ MPEG ได้ดังรูปที่1.1 ซึ่งรายละเอียด จะกล่าวในบทต่อไป

รูปที่ 1.1 แผนภาพการทำงานการถอดรหัสข้อมูลไฟล์ MPEG



1.4 ผลที่คาดว่าจะได้รับ

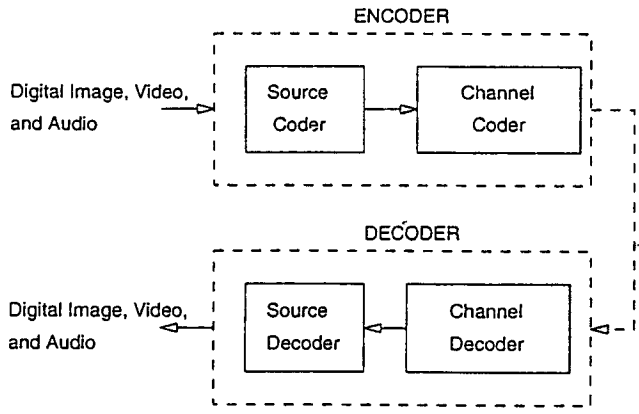
เราสามารถเขียนโปรแกรมถอดแฟ้มข้อมูลMPEG ได้โดยสามารถที่จะทำงานได้ตามมาตรฐานที่กำหนดไว้ และสามารถที่จะพัฒนาโปรแกรมขึ้นไปได้อีก และจากที่ทราบแล้วว่าJPEG เป็นพื้นฐานของการเขียนโปรแกรมการบีบอัดข้อมูล MPEG ทำให้เราก็สามารถที่จะเขียนโปรแกรมการบีบอัดข้อมูลภาพ

บทที่ 2
หลักการบีบอัดข้อมูลภาพ

2.1 บทนำ

การบีบอัดข้อมูล(Compression) คือ การทำให้ขนาดของข้อมูลดิจิตอลมีขนาดเล็ก เพื่อให้สามารถจัดการกับข้อมูลได้ง่ายสะดวกและรวดเร็ว ในงานที่ใช้ทางด้านมัลติมีเดียนั้นส่วนใหญ่แทบจะเป็นไปไม่ได้เลยที่จะสามารถทำงานได้โดยปราศจากขบวนการบีบอัดข้อมูล ปัจจุบันได้มีการพัฒนาทางด้านสถาปัตยกรรมการประมวลผลของภาพนิ่ง ภาพเคลื่อนไหวและเสียง โดยจัดการกับข้อมูลดิจิตอลเพื่อลดขนาดของข้อมูลโดยวิธีการบีบอัดข้อมูล ทำให้ได้มีการพัฒนาระบบมัลติมีเดียจำนวนมาก เพื่อตอบสนองความต้องการของตลาดผู้บริโภคที่มีขนาดใหญ่ ความแพร่หลายทางด้านการพัฒนาข้อมูลมัลติมีเดียดังกล่าว นำมาซึ่งมาตรฐานและวิธีการเข้ารหัสข้อมูล การเก็บข้อมูล การบีบอัดข้อมูล และการสื่อสารข้อมูลทำให้สามารถมีความเข้ากันได้ในทุกๆ ผลิตภัณฑ์ของผู้ผลิต

2.2 หลักการบีบอัดข้อมูล



รูปที่ 2.1 กระบวนการบีบอัดข้อมูล

หลักการพิจารณาข้อมูลดิจิตอลที่สามารถทำการบีบอัดข้อมูลได้

1. เมื่อพิจารณาสัญญาณส่วนเกิน(Redundancy) ในทางสถิติเราสามารถทำได้ดังนี้

- ข้อมูลภาพหรือ 1 เฟรม(Frame)ของข้อมูลวีดีโอ เมื่อเปรียบเทียบกับบริเวณข้างๆ ของภาพย่อมจะมีส่วนที่ซ้ำๆ กัน เรียกว่า Spatial correlation

- ข้อมูลที่มีหลายๆ ข้อมูล เช่น ภาพจากดาวเทียม ต้องมีการตรวจสอบท่ามกลางข้อมูลเหล่านั้น การเปรียบเทียบนี้เรียกว่า Spectral correlation

- ข้อมูลที่มีการเปลี่ยนแปลง เช่นข้อมูลจากวีดีโอ จะมีการเปรียบเทียบเฟรมหรือข้อมูลที่เวลาต่างกัน ซึ่งเรียกว่า Temporal correlation

2. มีการพิจารณา สัญญาณข้อมูล ซึ่งนอกเหนือจากจุดสิ่งที่มองเห็นได้และเข้าใจได้

3. บางข้อมูล มีแนวโน้มที่มีคุณลักษณะชั้นสูงที่เป็นส่วนเกินทั้งในด้านช่องว่าง(space) และเวลา (time)

จากรูปที่ 2.1 สิ่งสำคัญที่ต้องคำนึงถึงใน Encoder คือ ส่วนของ Source Coder

Source Coder จะแสดงกระบวนการบีบอัดข้อมูล ซึ่งกระบวนการดังกล่าวก็คือ การลดอัตราข้อมูลเข้า ทำให้อัตราข้อมูลเข้านั้นสามารถนำไปเก็บหรือทำการส่งผ่านข้อมูลได้ อัตราบิทที่เป็นเอาท์พุท ของ Encoder นี้สามารถวัดได้ในหน่วยบิทต่อวินาที สำหรับข้อมูลที่เป็นภาพนิ่งและภาพเคลื่อนไหว "จุดสี" จะเป็นส่วนสำคัญพื้นฐาน เพราะฉะนั้นหน่วยที่ใช้ในการวัดอัตราบิท จึงเป็นหน่วยบิทต่อพิกเซล(pixel)

อัตราการบีบอัดข้อมูล(compression ratio) หรือ Cr มักใช้แทน "อัตราบิท"

$$Cr = \text{ขนาดข้อมูลเข้า/ขนาดข้อมูลออก} \qquad (2.1)$$

ซึ่งสูตรดังกล่าวนี้บางครั้งก็ขึ้นอยู่กับชนิดของข้อมูลและวิธีการบีบอัดข้อมูลที่ใช้ เช่น ขนาดหมายถึง จำนวนบิทที่ต้องการแสดงบนภาพทั้งหมด สำหรับภาพเคลื่อนไหว คำว่า ขนาด จะหมายถึงจำนวนบิทที่ต้องการแสดงใน 1เฟรม

มีวิธีการบีบอัดข้อมูลหลายวิธีที่ไม่สามารถแสดงภาพแต่ละเฟรมของภาพเคลื่อนไหวได้ ดังนั้นคำว่า ขนาดนี้ อาจหมายถึงจำนวนบิทที่ต้องการแสดงในเวลา 1วินาที บนภาพเคลื่อนไหวก็ได้เช่นเดียวกัน แต่ในทางปฏิบัติ source coder มักจะขึ้นอยู่กับ the channel coder channel coder จะแปลงบิทที่ทำการบีบอัดข้อมูลแล้วไปเป็นสัญญาณที่เหมาะสมกับการเก็บหรือส่งผ่านข้อมูล ในระบบส่วนใหญ่ source coding และ channel coding จะมีกระบวนการที่แยกจากกัน แต่ในอนาคตข้างหน้านี้จะมีวิธีการรวมเอา 2 กระบวนการนี้เข้าด้วยกัน สำหรับ decoder ก็จะมีกระบวนการที่ตรงกันข้ามกับ encoder นั่นเอง โดยดูได้จากรูปที่ 2.1

ข้อจำกัดที่เป็นปัญหาในระบบการบีบอัดข้อมูล

1. ในการถอดรหัสข้อมูลข้อมูลจะถูกบังคับในเรื่องระดับคุณภาพของสัญญาณ ความซับซ้อนของการถอดรหัสที่ถูกบังคับนั้นส่งผลถึงทั้งการเข้ารหัสและการถอดรหัส

2. การเลื่อนของเวลาในการสื่อสาร(delay) เป็นการบังคับระยะเวลาตั้งแต่การเข้ารหัสจนถึงจนถึงการถอดรหัส

Application	Data Rate	
	Uncompressed	Compressed
Voice *8 ksamples/s, 8 bits/sample*	64 kbps	2 - 4 kbps
Slow-motion video (10 fps) *framesize 176 × 120, 8 bits/pixel*	5.07 Mbps	8 - 16 kbps
Audio conference *8 ksamples/s, 8 bits/sample*	64 kbps	16 - 64 kbps
Video conference (15 fps) *framesize 352 × 240, 8 bits/pixel*	30.41 Mbps	64 - 768 kbps
Digital audio (stereo) *44.1 ksamples/s, 16 bits/sample*	1.5 Mbps	128 - 1.5 Mbps
Video file transfer (15 fps) *framesize 352 × 240, 8 bits/pixel*	30.41 Mbps	384 kbps
Digital video on CD-ROM (30 fps) *framesize 352 × 240, 8 bits/pixel*	60.83 Mbps	1.5 - 4 Mbps
Broadcast video (30 fps) *framesize 720 × 480, 8 bits/pixel*	248.83 Mbps	3 - 8 Mbps
HDTV (59.94 fps) *framesize 1280 × 720, 8 bits/pixel*	1.33 Gbps	20 Mbps

ตารางที่ 2.1 การใช้งานการบีบอัดข้อมูล ภาพ วีดีโอ เสียง

2.2.1 วิธีการของการบีบอัดข้อมูล

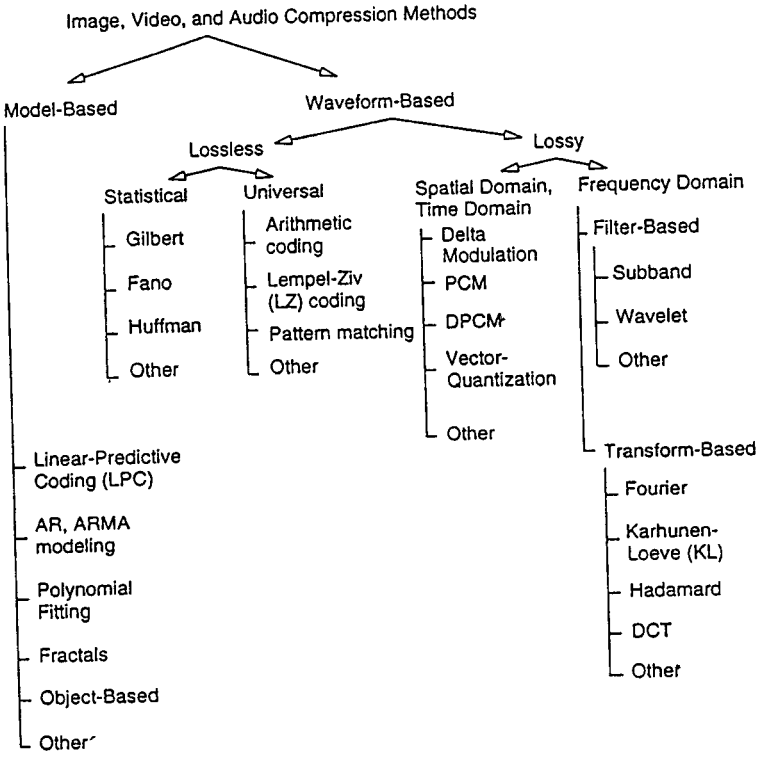
การบีบอัดข้อมูลแบบไม่สูญเสีย (lossless compression) เป็นการบีบอัดข้อมูลที่เมื่อทำการถอดรหัสแล้วข้อมูลที่สร้างขึ้นมาใหม่จะไม่มีการสูญเสียข้อมูลตรงส่วนใดๆเลย ในการบีบอัดข้อมูลแบบไม่สูญเสียนี้ สามารถแบ่งได้เป็น 3 วิธี คือ ประสิทธิภาพของการบีบอัดข้อมูล (coding efficiency) , ความซับซ้อนของการบีบอัดข้อมูล (coding complexity) ,หน่วยเวลาของการบีบอัดข้อมูล (coding delay)

- ประสิทธิภาพของการบีบอัดข้อมูล คือ การวัดอัตราบิทต่อวินาที(bps) ของการทำงานซึ่งจะถูกจำกัดด้วยปริมาณของข้อมูลจากแหล่งข้อมูล ในทางทฤษฎี " The entropy of a source X " จะ

ประกอบไปด้วยการนำค่า X มาวัดแบบสุ่ม(random) จากทฤษฎีของการบีบอัดข้อมูล source ที่มีขนาด entropy ใหญ่จะทำการบีบอัดข้อมูลได้ยากกว่า (ตัวอย่างเช่น สัญญาณรบกวนจะทำการบีบอัดข้อมูลได้ยาก)

- ความซับซ้อนของการบีบอัดข้อมูล เป็นสิ่งที่จำเป็นต่อการเข้ารหัสและการถอดรหัส เพื่อให้ได้การบีบอัดข้อมูลที่มีประสิทธิภาพสูงสุด ซึ่งอาจจะวัดจากความต้องการใช้หน่วยความจำหรือจำนวนคำสั่งของหน่วยคำนวณทางคณิตศาสตร์ (MOPS : Million Operation Per Second)

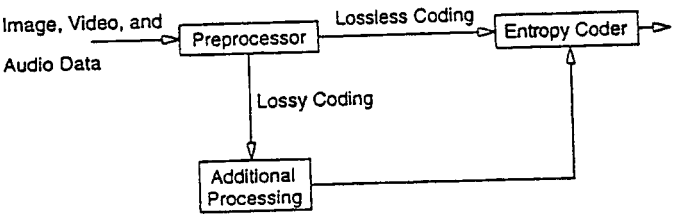
- การหน่วงเวลาของการบีบอัดข้อมูล ในกระบวนการบีบอัดข้อมูลที่ซับซ้อนมักจะนำไปสู่การเพิ่ม coding delays ขึ้นที่ encoder และ decoder เราสามารถลด coding delays ได้โดยการเพิ่มกำลังงานให้กับเครื่องจักรที่ใช้ในการคำนวณ แต่ในทางปฏิบัติเราอาจไม่สามารถควบคุมกำลังงานดังกล่าวได้ และในงานบางอย่างต้องการเวลาที่รวดเร็ว เช่น งานทางด้านระบบการสื่อสาร ดังนั้นเราต้องควบคุม coding delay นี้ให้ได้จึงจะทำให้งานนั้นประสบผลสำเร็จและมีประสิทธิภาพสูงสุด



รูปที่ 2.2 สาขาต่างของวิธีการบีบอัดข้อมูล

การบีบอัดข้อมูลแบบสูญเสีย (lossy compression) โดยปกติทั่วไปแล้วการทำงานกับข้อมูลภาพหรือวีดีโอนั้น ไม่จำเป็นจะต้องทำการเข้ารหัสและถอดรหัสแล้วให้ได้ข้อมูลเหมือนเดิมทุกอย่าง การสูญเสียข้อมูลไปบางส่วนไปที่ยอมรับได้ เราเรียกกระบวนการบีบอัดข้อมูลแบบนี้ว่า การบีบอัดข้อมูลแบบสูญเสีย(lossy compression) คุณภาพของสัญญาณ โดยส่วนใหญ่แล้วจะหมายถึงคุณสมบัติของสัญญาณที่เอาท์พุท

เพื่อที่มีประสิทธิภาพในการบีบข้อมูลได้สูง จึงต้องอาศัยการบีบอัดข้อมูลทั้งสองแบบ โดยสามารถเขียนแผนผังการทำงานในรูปแบบทั่วไปได้ดังนี้



รูปที่ 2.3 การทำงานร่วมกันของการบีบอัดข้อมูลแบบไม่สูญเสียกับแบบสูญเสีย

2.3 วิธีการและมาตรฐานของการบีบอัดข้อมูลแบบไม่สูญเสีย

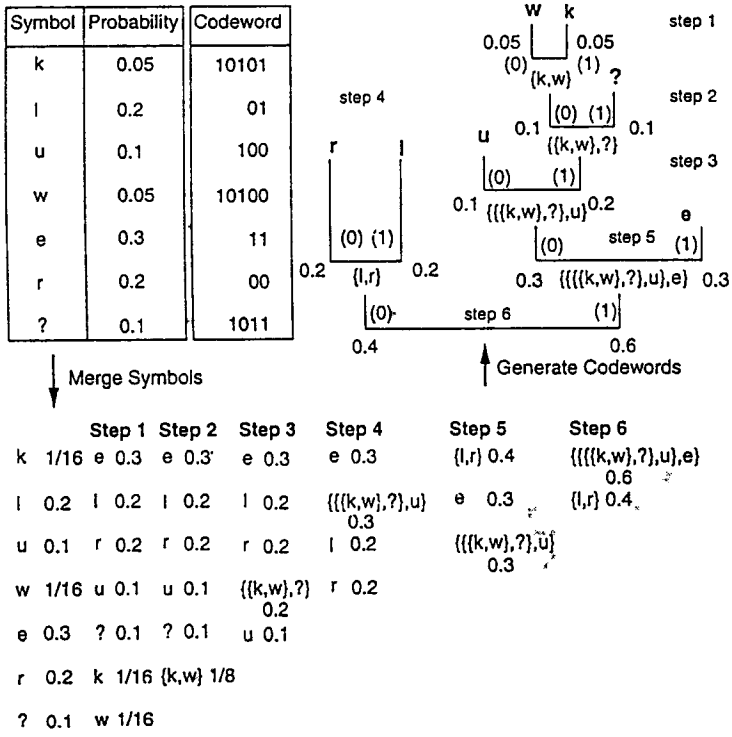
2.3.1 เทคนิคในการบีบอัดข้อมูล

ในการประมวลผลข้อมูลของการบีบอัดข้อมูลแบบไม่สูญเสีย จะถูกกำหนดให้ทำงานเป็น 8,16,32 หรือ 64 บิท แต่ถ้าขนาดข้อมูลของภาพเท่ากับ 12 บิทต่อพิกเซล ดังนั้นเราจะต้องทำการลดขนาดของข้อมูลลงให้กลายเป็น 8 บิทต่อพิกเซล แต่ยังคงความสามารถในการประมวลผลให้ได้อัตราการบีบอัดตามต้องการ นอกจากนั้นแทนที่เราจะประมวลผลทีละพิกเซล โดยทั่วไปแล้วเราควรจะจัดข้อมูลให้เป็นกลุ่มของพิกเซลแล้วทำการแบ่งแยกกลุ่ม โดยแต่ละกลุ่มก็จะเข้าบล็อกของ

มันเอง ซึ่งวิธีนี้จะทำให้การบีบอัดข้อมูลนั้นมีประสิทธิภาพมากขึ้น แต่จะทำให้การเข้ารหัสมีความซับซ้อนมากขึ้นเช่นกัน

2.3.2 การเข้ารหัสแบบฮัฟแมน(Huffman Encoder)

ในปี ค.ศ. 1952 ฮัฟแมนได้พัฒนาการบีบอัดข้อมูลแบบไม่สูญเสียมาเป็นกระบวนการที่รวมเอา Modeling และ Symbol-to-codeword mapping เข้าด้วยกัน ข้อมูลอินพุทจะถูกแบ่งเป็นลำดับของ symbols เพื่อสะดวกในกระบวนการโมเดล(Modeling process) ส่วนมากภาพและ การประยุกต์การบีบอัดวีดีโอ จะใช้ขนาดของตัวอักษรให้มาเป็น symbols โดยจะไม่เกิน 64000 symbols การเข้ารหัสแบบฮัฟแมน จะประกอบไปด้วยการปฏิบัติการที่ค่อยเป็นค่อยไปทีละขั้นดังนี้



รูปที่ 2.4 ตัวอย่างโครงสร้างของการเข้ารหัสฮัฟแมน

1. เรียงลำดับ symbols ตามความน่าจะเป็นของมัน โครงสร้างของการเข้ารหัสฮัฟแมน สิ่งแรกที่จำเป็นต้องทราบคือ ความถี่ของการเกิด symbols แต่ละตัว ซึ่งจะประมาณได้จากข้อมูลทดลองชุดหนึ่ง ซึ่งเป็นตัวแทนของข้อมูลที่ถูกบีบอัดในแบบไม่สูญเสีย(lossless)

2. คำนินการกับ 2 symbols ซึ่งมีความน่าจะเป็นน้อยที่สุด โดยใช้กระบวนการการหดตัว (Contraction process)

3. ทำซ้ำขั้นตอนข้างต้นจนกระทั่งเซตสุดท้ายมีเพียง 1สมาชิกเท่านั้น

คุณสมบัติของรหัสฮัฟแมน

กำหนดให้เอนโทรปีของ source S คือ

$$H(s) = \eta = \sum_i p_i \log_2 \frac{1}{p_i} \qquad (2.2)$$

โดยที่ค่าของเอนโทรปีจะแสดงถึงขอบเขตของความยาวเฉลี่ยของ codeword โดยที่เราจะพอใจกับค่าเอนโทรปีเมื่อ $\eta \leq l_{avg} \leq \eta+1$ โดยถือว่ามันได้เข้าใกล้สภาวะที่ดีที่สุด

ตารางรหัสฮัฟแมนที่แสดงไว้ดังรูปที่ 2.4 นั้นเรียกว่า วิธีการ bottom-up และในเร็วๆนี้ ได้มีการคิดค้น วิธีการ top-down ดังนั้นตารางรหัสฮัฟแมนที่แสดงดังรูปที่ 2.4 นั้นจะไม่ได้นำมาใช้งานจริง อย่างเช่นใน มาตรฐาน JPEG นั้นจะต้องใช้ รหัสฮัฟแมนควบคู่ไปกับ run-length-code ข้อเสียอย่างหนึ่งของการรหัสฮัฟแมนแบบธรรมดาคือ จะต้องใช้หน่วยความจำ(memory) มากในการเก็บค่าตารางดังกล่าว เนื่องจากว่าในการ decode จะต้องใช้ Look up table ซึ่งการที่มีความยาวไม่จำกัดของ codeword ทำให้ต้องใช้พื้นที่ในหน่วยความจำมาก

2.3.3 การถอดรหัสแบบฮัฟแมน

การเข้ารหัสฮัฟแมนจะเป็นไปในลักษณะตรงไปตรงมา โดยที่ตารางของ Symbol-to-codeword โดยมีวิธีการถอดรหัส 2วิธีดังนี้

1. การถอดรหัสข้อมูลอนุกรม(Bit-serial Decoding)เราจะทำสร้างโครงสร้างกิ่งสาขาของรหัสฮัฟแมนขึ้นใหม่โดยใช้ตาราง Symbols-to-codeword โดยกระทำตามขั้นตอนดังนี้

- อ่านสายบิตข้อมูลที่บีบไว้จนไปถึงสุดปลายสุดของกิ่งสาขา

- ในขณะที่แต่ละบิทถูกอ่านแล้วก็จะถูกทิ้งไว้ จนเมื่อถึงปลายสุดก็จะได้ Symbol ออกมา โดยเราจะทำซ้ำจนกระทั่งทุกอินพุทถูกใส่เข้าไป และเราจะพบว่าวิธีการนี้จะมีอัตราข้อมูลเข้า(Input bit rate) ที่คงที่ และจะมีอัตราข้อมูลออก(Output bit rate) ที่แปรค่าได้

2. การถอดรหัสโดยใช้ตาราง(Lookup-Table-Based Decoding) วิธีการนี้จะให้อัตราข้อมูลออกคงที่ โดยจะต้องทำการสร้าง Lookup-Table จากขั้นตอน Symbol-to-codeword mapping ขั้นตอนการสร้าง Lookup Table มีดังนี้

- ให้ Ci เป็น codeword ของ Si ถ้าให้ Ci มี li เราจะต้องสร้างแอดเดรสทั้งหมด L บิท โดยที่ li บิทแรกเป็น Ci และที่เหลือ L-li อาจจะเป็นหนึ่งหรือศูนย์ ดังนั้น Si จะมีแอดเดรสทั้งหมด 2^{L-li}

- ในแต่ละทางเข้า เราจะสร้างค่า(Si,li) 2 ชุด

การทำ Lookup-Table ทำให้การถอดรหัสค่อนข้างง่ายและรวดเร็ว โดยจะเห็นได้จากขั้นตอนดังนี้

- จากชุดข้อมูลอินพุท เราจะอ่าน L บิทเข้าไปยังบัฟเฟอร์(buffer)

- เราจะใช้ค่า L เป็นแอดเดรสชี้ไปยัง Lookup-Table จะเห็นได้ค่า Symbol ออกมา

- จากนั้นจะทิ้งค่าเดิมแล้วอ่านค่าใหม่มายังบัฟเฟอร์

- ทำซ้ำจนกระทั่งได้ symbol ทั้งหมดออกมา

จากทั้ง 2วิธีที่กล่าวมานั้น วิธีถอดรหัสโดยเปิดตารางนั้นจะมีข้อได้เปรียบเทียบตรงที่ว่าสามารถถอดรหัสได้เร็วกว่าวิธีแบบข้อมูลอนุกรม แต่ก็ยังไม่นิยมใช้เพราะว่าต้องใช้หน่วยความจำเยอะดังที่ได้กล่าวมาแล้ว ดังนั้นจึงมีการคิดค้นการเข้ารหัสและถอดรหัสแบบอื่นๆ ขึ้นมาเพื่อให้ได้การบีบอัดข้อมูลให้ดีที่สุด

2.3.4 การเข้ารหัสแบบทำนายตัวเลข(Arithmetic coding)

เนื่องจากว่าการเข้ารหัสแบบฮัฟแมนในแต่ละวิธีนั้น ยังมีข้อจำกัดตรงที่ว่าต้องใช้จำนวนบิท อย่างน้อย 1บิทต่อ 1 symbol ทำให้การบีบอัดข้อมูลยังถูกจำกัดไว้อยู่ วิธีการเพิ่มคุณภาพการบีบอัดข้อมูลที่ดีอย่างหนึ่งนั้นก็คือ การเข้ารหัสแบบทำนายตัวเลข(Arithmetic coding) การเข้ารหัสฮัฟแมนยังมีข้อเสียอยู่อย่างหนึ่งคือ การปรับปรุงตารางรหัสนั้นทำได้ยาก กล่าวได้คือ เมื่อ symbol ที่ทางเข้ามีการเปลี่ยนแปลง เราจำเป็นที่จะต้องเขียนตารางรหัสฮัฟแมนใหม่ทั้งหมด ซึ่งจะเป็นวิธีการที่ยุ่งยากและเสียเวลา แต่การเข้ารหัสแบบทำนายตัวเลขนี้ไม่จำเป็นที่จะต้องเขียนใหม่ทั้งหมด

2.3.5 มาตรฐานการบีบอัดข้อมูลแบบไม่สูญเสีย

มาตรฐานการบีบอัดข้อมูลแบบไม่สูญเสียถูกกำหนดโดย International Telecommunication Union (ITU-T) โดยจะถูกกำหนดให้อยู่ในหมวดที่เรียกว่า CCITT ซึ่งจะมีดังนี้

1. มาตรฐานการบีบอัดข้อมูลแบบถ่ายสำเนา (Facsimile Compression Standards) อาศัยหลักการที่เลขไบนารี่มีการซ้ำกันหลายค่า โดยเราจะเปลี่ยนโดยใช้วิธีที่เรียกว่า Run-length coding

2. มาตรฐานการบีบอัดข้อมูลแบบJBIG Joint Binary Image Experts Group เป็นการบีบอัดข้อมูลที่พัฒนาจากวิธีการบีบอัดข้อมูลแบบฮาล์ฟโทน (Compress halftone) ที่มีประสิทธิภาพ หรือเรียกว่า ISO/IEC IS 11544 , ITU-T Rec. โดยประกอบด้วย modeler และ การเข้ารหัสแบบทำนายตัวเลข(Arithmetic coding) โดย modeler จะเป็นตัวประมาณค่าความน่าจะเป็นของ symbol เพื่อส่งค่าเข้าไปในขั้นตอนการเข้ารหัสแบบทำนายตัวเลข

3. มาตรฐานการบีบอัดข้อมูลแบบไม่สูญเสียแบบJPEG Joint Photography Experts Group ร่วมกับ ISO และ ITU-U เช่นเดียวกับแบบJBIG ต่างกันตรงที่การใช้วิธีการบีบอัดข้อมูลแบบ JPEG จะเป็นการแปลงแบบอิสระ โดยใช้การเข้ารหัสแบบผลต่าง(Differential coding) จากการทำนายค่าที่เหลืออยู่(prediction residuals) ดังแสดงไว้ในตารางที่ 2.2

Category	Prediction Residual
0	0
1	-1, 1
2	-3, -2, 2, 3
3	-7, .., -4, 4, .., 7
4	-15, .., -8, 8, .., 15
5	-31, .., -16, 16, .., 31
6	-63, .., -32, 32, .., 63
7	-127, .., -64, 64, .., 127
8	-255, .., -128, 128, .., 255
9	-511, .., -256, 256, .., 511
10	-1023, .., -512, 512, .., 1023
11	-2047, .., -1024, 1024, .., 2047
12	-4095, .., -2048, 2048, .., 4095
13	-8191, .., -4096, 4096, .., 8191
14	-16383, .., -8192, 8192, .., 16383
15	-32767, .., -16384, 16384, .., 32767
16	32768

ตารางที่ 2.2 Prediction residual categories for lossless-JPEG compression

2.4 หลักการบีบอัดข้อมูลแบบสูญเสีย

การบีบอัดข้อมูลแบบสูญเสียนั้นจะได้ข้อมูลกลับคืนมาไม่สมบูรณ์เหมือนตอนก่อนผ่านกระบวนการบีบอัดข้อมูล ในภาพนิ่งนั้นมีการพัฒนาการใช้การบีบอัดข้อมูลแบบสูญเสียกันอย่างแพร่หลาย

2.4.1. รูปแบบสำหรับที่ว่างเปรียบเทียบของภาพ

ข้อมูลที่เป็นภาพนิ่งมักจะมีที่ว่างที่ไม่จำเป็นซึ่งสายตามนุษย์ไม่สามารถสังเกตเห็นได้ ดังนั้นถ้ามีคนสังเกตการบีบอัดข้อมูลแบบสูญเสียของภาพนิ่งภาพหนึ่งตั้งแต่ต้นจนจบ จะเห็นได้ว่าการที่เราทำการลดจำนวนบิทของข้อมูลให้สั้นที่สัญญาณุดนั้น จะทำให้การบีบอัดข้อมูลมีประสิทธิภาพสูง โดยเราจะใช้ประโยชน์จากส่วนที่ว่างที่ซ้ำกันของภาพนั่นเอง

เราสามารถสร้างรูปแบบของสมการจากที่ว่างโดยให้สัมพันธ์กับพิกเซลเป็นฟังก์ชันแบบ 2 มิติ (2-D: two-dimensional) เรียกว่า Covariance function คือ

$$Cov_x(i,j) = \sigma^2 \exp(-\alpha\sqrt{i^2 + j^2}) \qquad (2.3)$$

โดยที่ σ^2 คือ ค่า variance ของภาพ

i,j คือ ระยะทางจากพิกเซลอ้างอิง

เมื่อ covariance function สามารถหาค่าได้ ถ้าเราให้

$$\rho_1 = E[X(i,j)\, X(i-1,j)] / E[X^2(i,j)] \qquad (2.4)$$

$$\rho_2 = E[X(i,j)\, X(i,j-1)] / E[X^2(i,j)] \qquad (2.5)$$

การเปรียบเทียบระหว่างพิกเซลในแนวนอนและแนวตั้ง ถ้าเราสมมติให้ $\rho_1 = \rho_2$ จากสมการ(2.4) และ (2.5) จะเห็นว่าไม่มีความแตกต่างกันของพิกเซลที่อยู่ใกล้กันเลยทั้งในแนวนอนและแนวตั้ง ซึ่งจะทำให้ $\exp-\alpha = \rho_1$ ภาพนิ่งโดยทั่วไปจะมีค่า ρ_1และρ_2 ประมาณ 0.95 จากการคำนวณธรรมดา ค่า iกับค่า j ที่เปลี่ยนแปลงไปแสดงให้เห็นว่า ถ้า $i,j > 8$ จะทำให้หาค่า covariance function ไม่ได้ ดังนั้นเราจำเป็นต้องกำหนดไม่ให้บล็อกของพิกเซล (block of pixels) มีค่าไม่เกิน 8 พิกเซล

จากหัวข้อที่ผ่านมา จะเห็นได้ว่าการบีบอัดข้อมูลแบบสูญเสียจะให้อัตราการบีบอัดที่ดีว่าการบีบอัดข้อมูลแบบไม่สูญเสีย แต่อย่างไรก็ตาม ในทางปฏิบัติการบีบอัดข้อมูลแบบสูญเสียจะมีเงื่อนไขระหว่างความสูญเสีย(D; distortion) และอัตราบิทของข้อมูลที่ถูกบีบอัดแล้ว(R) ด้วยซึ่งค่า "R" จะแสดงอยู่ในรูปของบิทต่อ encoder output symbol และ "D" จะถูกแสดงในรูปของ variance ของ encoder input ถ้าข้อมูลของ encoder input เป็น 8 ต่อพิกเซลอัตราการบีบอัดข้อมูลจะเป็น 8/R

ทฤษฎีของ อัตราการสูญเสีย(Rate-distortion) ให้ R_{min} คืออัตราสูญเสียที่น้อยที่สุด เพื่อให้ข้อมูลที่ถูกบีบอัดแล้ว สามารถถอดรหัสกลับมาภายในค่าความสูญเสียที่กำหนดคือ D การบีบอัดข้อมูลทุกวิธีจะมีความสัมพันธ์ของ D และ R เสมอสำหรับค่า D ค่าหนึ่ง Rate-distortion function R (D)

สามารถอธิบายให้มีค่าน้อยที่สุดที่อัตรา "R" จะเป็นไปได้ โดยจะต้องมีค่าความสูญเสีย D หรือน้อยกว่านี้ ถ้าR(D) จะเป็นอิสระกับการบีบอัดข้อมูลบางวิธี แต่จะขึ้นอยู่กับ stochastic model สำหรับอินพุทที่เป็นภาพ และการวัดความสูญเสียเท่านั้น รูปแบบทางคณิตศาสตร์สำหรับที่ว่างที่เป็นส่วนเกิน(spatial redundancy) จะต้องสนับสนุน 2 เป้าหมายคือ

1. มันต้องถูกใช้ในการออกแบบส่วนประกอบของระบบการบีบอัดภาพ

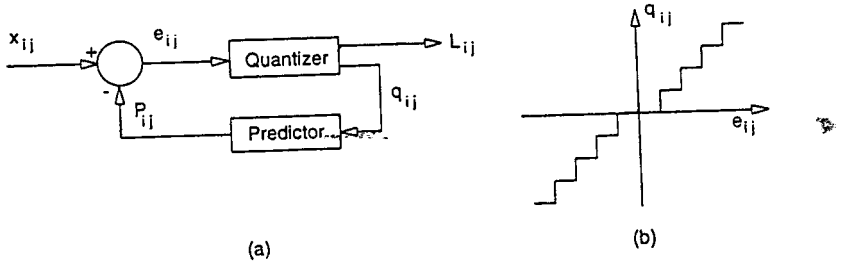
2. ถ้าเราให้ความผิดเพี้ยนระหว่างภาพจริงกับภาพหลังจากถูกทำการถอดรหัสแล้วเป็น D แล้วเราจะต้องสามารถหาอัตราเร็วข้อมูล(Bit rate) ที่น้อยที่สุด R(D) ได้ในทุกรูปแบบของการบีบอัดข้อมูล

2.4.2. การเข้ารหัสการบีบอัดข้อมูลแบบสูญเสีย(Lossy compression)

สำหรับข้อมูลที่เป็นภาพแล้ววิธีการบีบอัดข้อมูลมีอยู่ด้วยกัน 2ระดับคือ

1. sample-based coding รูปภาพที่ใช้ในการบีบอัดข้อมูลนั้นสามารถเป็นได้ทั้งแบบขอบเขตของที่ว่าง(spatial domain) และ ขอบเขตของความถี่(frequency domain) วิธีการบีบอัดข้อมูลแบบนี้จะมีรูปแบบง่ายๆ ดังเช่นการ Differential pulse code modulation(DPCM) ซึ่งแสดงไว้ดังรูปที่ 2.7 วิธีนี้คล้ายๆ กับการบีบอัดข้อมูลแบบไม่สูญเสียใน JPEG ต่างกันเพียงแค่ การบีบอัดข้อมูลในJPEG นั้น จะไม่ผ่านขั้นตอนการควอนไตซ์(Quantization)

จากรูปที่ 2.5(a) กำหนดให้ภาพตัวอย่างที่เป็นอินพุท มีค่า X_{ij} สัญญาณที่เหลือ e_{ij} จะถูกกำหนดโดยใช้สัญญาณที่ใช้ในการทำงาน p_{ij} ซึ่ง p_{ij} นั้นจะเป็นตัวถ่วงน้ำหนักผลรวมของพิกเซลที่ถูกถอดรหัสไปก่อนหน้านั้นแล้ว โดยพิกเซลดังกล่าวนี้จะอยู่ใกล้ๆ กับ X_{ij} ถ้าภาพตัวอย่างที่เป็นอินพุท คล้ายคลึงกับภาพที่แล้วมาก ค่า p_{ij} ก็จะเท่ากับค่า X_{ij} ซึ่งจะทำให้ค่า e_{ij} เล็กลง ถ้า e_{ij} ที่เล็กกว่าก็จะทำให้ได้การบีบอัดข้อมูลที่ดีกว่า รูปแบบของการควอนไตซ์ แสดงได้ดังรูปที่ 2.5(b) การควอนไตซ์จะทำให้ข้อมูลอินพุทที่เข้ามาหลายๆ ค่าออกมาเป็นเอาท์พุทค่าเดียว กระบวนการดังกล่าวนี้ไม่สามารถย้อนกลับได้ ซึ่งนี่ก็เป็นสาเหตุหลักที่ทำให้เกิดความผิดเพี้ยนของข้อมูลได้



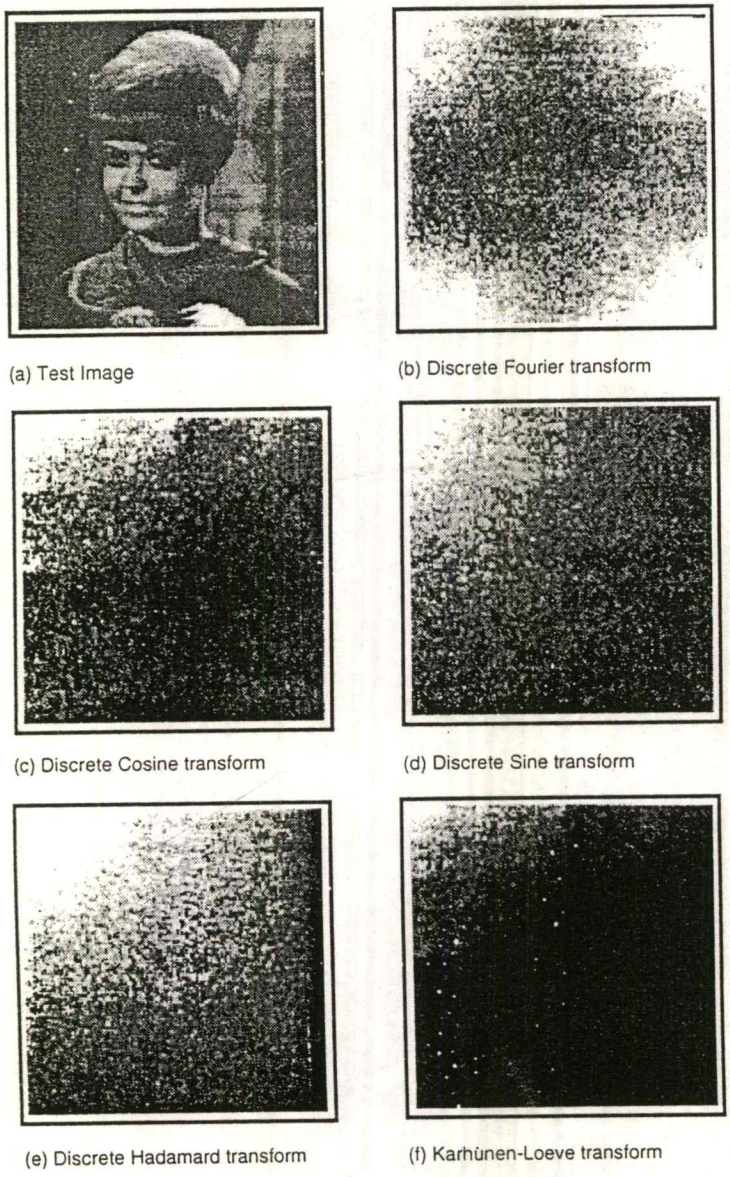
รูปที่ 2.5 การบีบอัดข้อมูลแบบ DPCM 2.5(a) Encoding Block diagram
2.5(b) กราฟแสดงความสัมพันธ์ระหว่างอินพุทกับเอาท์พุท

2. Block-Based coding จากทฤษฎีของอัตราการสูญเสีย(rate distortion) วิธีของ Source coding แสดงให้เห็นว่า ถ้าขนาดของ coding block เพิ่มขึ้น เราจะสามารถหา coding scheme ของ $R > R(D)$ ได้โดยไม่มีความสูญเสียเข้าใกล้ D หรืออีกนัยหนึ่งถ้า source ถูกเข้ารหัสแบบ Infinity large block เราก็สามารถหา block-coding scheme ที่มีอัตรา R(D) และมีความสูญเสีย D ได้เช่นกัน

จะเห็นได้ว่าการบีบอัดข้อมูลแบบ sample-based เช่น DPCM ไม่สามารถหาค่า R(D) ที่ต้องการได้ ดังนั้นเราจึงใช้วิธี Block-based coding แทน ซึ่งวิธีนี้จะทำให้ได้อัตราการบีบอัดข้อมูลที่ดีกว่า ถ้าเทียบที่ระดับความสูญเสียเดียวกันเทคนิคการบีบอัดข้อมูลเป็นจำนวนมากได้พัฒนาการทำงานมาเป็น Blockwise ซึ่งเทคนิคดังกล่าวนี้แบ่งออกได้เป็น 2 ประเภทคือ

- การเข้ารหัสในเชิงที่ว่าง(Spatial domain block coding) วิธีการนี้จะถูกแบ่งให้เป็นกลุ่มๆ ที่เรียกว่า บล็อก และบล็อกเหล่านี้จะถูกบีบอัดข้อมูลโดยให้อยู่ในรูป โดเมนที่ว่าง (spatial domain)

- transform-domain block coding วิธีการนี้พิกเซลจะถูกแบ่งออกเป็นกลุ่มที่เรียกว่า บล็อกเหมือนกัน เพียงแต่ว่าบล็อกเหล่านี้จะถูกแปลงไปเป็นอีกโดเมนอื่นๆ เช่น โดเมนความถี่



รูปที่ 2.6 เปรียบเทียบการแปลงข้อมูลด้วยวิธีการต่างๆ

วิธีการแปลง(transform) นั้นมีอยู่ด้วยกันหลายวิธี ดังรูปที่ 2.6 แต่วิธีที่มีประสิทธิในการบีบอัดข้อมูลดีที่สุด คือ Karhunen-Loeve transform (KLT) แต่ข้อเสียของวิธีคือ เราจะได้ภาพที่ไม่เป็นอิสระ เราจึงไม่นิยมมาใช้ เราจึงมาใช้วิธีอื่นที่ได้ภาพที่เป็นอิสระต่อกัน อย่างเช่น DFT,DCT,DST และ DHT และจากรูปที่ 2.6 นี้จะเห็นได้ว่าวิธี Discrete Cosine Transform (DCT) นั้นเป็นวิธีทีให้ประสิทธิภาพได้ดีที่สุดรองจากวิธี Karhunen-Loeve ทำให้เราใช้วิธีนี้ในการบีบอัดข้อมูลแบบสูญเสีย รวมถึงเราได้ใช้วิธี DCT นี้เป็นมาตรฐานในการบีบอัดข้อมูลทั้งในภาพนิ่งและในภาพเคลื่อนไหวด้วย

2.4.3. การเข้ารหัสโดย DCT

การเข้ารหัสแบบ DCT หรือ Discrete Cosine Transform เป็นพื้นฐานของมาตราการบีบอัดข้อมูลทั้งภาพนิ่งและภาพเคลื่อนไหวทั้งหมด พื้นฐานในการแปลงDCT การแปลงแบบDCT นั้นมีอยู่ด้วยกันหลายมีหลายประเภท เช่น 1-D DCT, 2-D DCT, 3-D DCT ซึ่งในแต่ละวิธีนี้ก็สามารถแบ่งได้อีกหลายวิธี แต่มาตรฐานของการบีบอัดข้อมูลแบบสูญเสียของ JPEG และ MPEG นั้นจะนิยมใช้ แบบ2-D DCT แต่ในทางทฤษฎีแล้วการแปลงDCT แบบ 3-D นั้นมีประสิทธิ์ภาพดีกว่าทั้ง 2 แบบ แต่เนื่องจากว่ามันมีรูปแบบการคำนวณที่สลับซับซ้อนมาก และยุ่งยากต่อออกแบบทำให้เราจึงไม่นิยมใช้กัน ดังนั้นในที่นี้เราจึงขอกล่าวเฉพาะการDCT แบบ 2-D เท่านั้น ซึ่งมีรูปแบบคือการแปลง เมตริกซ์ ขนาด N*N ของภาพ และในโดเมนที่ว่าง(spatial domain) ไปเป็น DCT domain สำหรับมาตรฐานการบีบอัดข้อมูลของภาพนิ่ง จะให้ N=8 เหตุผลที่เลือกขนาดของ บล็อกภาพ (image block) เป็น 8*8 พิกเซล ก็เพราะว่าที่ไม่ใช้เนื้อที่ในหน่วยความจำมากเกินไป และเป็นขนาดที่สายตามนุษย์สามารถสังเกตเห็นความแตกต่างได้ (ถ้าเราใช้ขนาดของเมตริกซ์ที่ใหญ่ 8*8 จะทำให้การบีบอัดข้อมูลดีขึ้น)

DCT นั้นมีคุณสมบัติเป็น Orthogonal transform ดังนั้นเราจึงสามารถเขียนอยู่ในรูปเมตริกซ์ได้ ดังนี้

$$\text{DCT} \quad Y = TXT^{t} \tag{2.6}$$

$$\text{IDCT} \quad Y = T^{t}XT \tag{2.7}$$

เราสามารถเขียนสมการการแปลง DCT จาก $X^{8*8DCT} \rightarrow Y$ ได้ดังนี้คือ

$$y_{kl} = \frac{C(k)C(l)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} x_{ij}\cos\left[\frac{(2i+1)k\pi}{16}\right]\cos\left[\frac{(2i+1)l\pi}{16}\right] \tag{2.8}$$

$$\text{โดยที่ } C(k) = \begin{cases} 1/2 & ;\text{ถ้า } k = 0 \\ 1 & ; k \neq 0 \end{cases} \tag{2.9}$$

หรืออาจเขียนให้อยู่ในรูปเมตริกซ์เวคเตอร์(Vector metrix) ได้เป็น

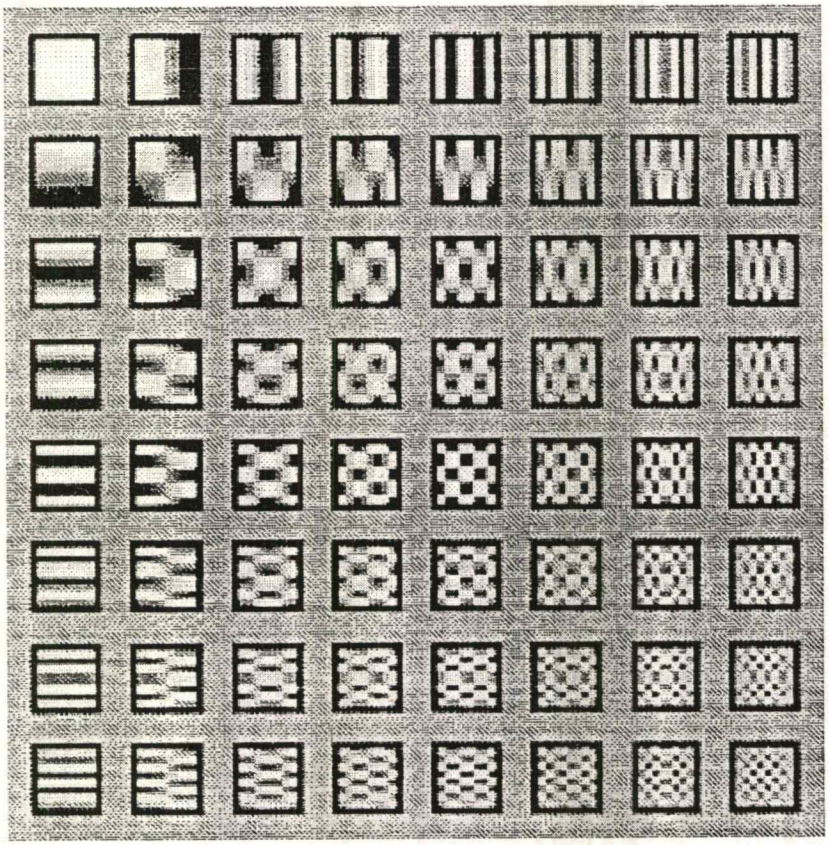
$$Y = TX \qquad (2.10)$$

โดยที่ $X = \{ X_{00}, X_{01}, \ldots, X_{07}, X_{10}, X_{11}, \ldots, X_{17}, \ldots, X_{70}, X_{71}, \ldots, X_{77} \}$

T = คือเมตริกซ์ขนาด 64*64 โดยค่าต่างๆ นั้นหาได้จากสมการDCT ในสมการ(2.8)

$Y = \{ Y_{00}, Y_{01}, \ldots, Y_{07}, Y_{10}, Y_{11}, \ldots, Y_{17}, \ldots, Y_{70}, Y_{71}, \ldots, Y_{77} \}$

การแปลง DCT จะสร้างอินพุทบล็อกขึ้นมาใหม่ เป็นอนุกรมของคลื่น(waveform) ซึ่งแต่ละบล็อกจะมีค่าเฉพาะของ spatial frequency ของตัวมันเอง ซึ่ง ผลรวมของ 64 คลื่นทั้งหมด นี้จะทำให้ได้ รูปแบบสมการพื้นฐานของ DCT ดังแสดงในรูปที่ 2.7 และจากรูปแสดงถึงกระบวนการการหาสัมประสิทธิ์ของแต่ละคลื่น(waveform) โดยที่สัมประสิทธิ์เหล่านี้จะทำให้เราสามารถแปลงข้อมูลกลับมาเป็น เมตริกซ์ X ขนาด 8*8 ได้อีก



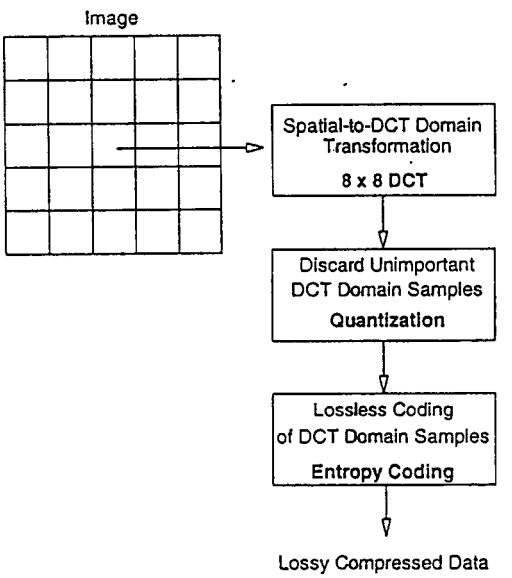
รูปที่ 2.7 รูปแบบพื้นฐานของการแปลง DCT

สำหรับ Inverse DCT เขียนเป็นสมการได้ดังนี้

$$x_{ij} = \frac{C(k)C(l)}{4} \sum_{i=0}^{7} \sum_{i=0}^{7} x_{ij} \cos\left[\frac{(2i+1)k\pi}{16}\right] \cos\left[\frac{(2i+1)k\pi}{16}\right] \qquad (2.11)$$

โดยที่ i,j = 0,1,...,7

พื้นฐานของระบบการเข้ารหัสแบบ DCT



รูปที่ 2.8 แผนภาพการทำงานของการแปลงภาพแบบ DCT

จากรูปที่ 2.8 แสดงให้เห็นถึงขั้นตอนการทำงานโดยทั่วไปของระบบ DCT ในส่วนของการเข้ารหัส(encoder) DCTจะทำการแปลงบล็อกขนาด 8*8 ในแต่ละบล็อก จากบล็อกX ให้กลายเป็น สัมประสิทธิ์ ของบล็อกY จากที่ได้กล่าวมาแล้วว่า ค่าสัมประสิทธิ์ในแต่ละอันเหล่านี้จะเป็นตัวถ่วงน้ำหนัก ที่มีความเกี่ยวข้องกับรูปแบบพื้นฐานของ DCT(DCT basic waveform) ที่มีลักษณะตรงกัน ในระบบการบีบอัดข้อมูลแบบสูญเสีย ตัวถ่วงน้ำหนักบางตัวจะถูกลบออก เพราะฉะนั้น รูปแบบสัญญาณ(waveform) ที่มีค่าตรงกันกับตัวถ่วงน้ำหนักนั้นก็จะไม่ถูกนำไปใช้ในการดีคอมเพรส (Decompress) อีก

จากกระบวนการที่มีการลบตัวถ่วงน้ำหนักบางตัวออก จะตรงกันกับขั้นตอนของการควอนไตซ์ ดังแสดงในรูปที่ 2.8 กระบวนการควอนไตซ์ เป็นกระบวนการที่ย้อนกลับไม่ได้ และเป็นขั้นตอนเดียวที่ทำให้มีการสูญเสียใน DCT coding scheme และหลังจากที่ข้อมูลผ่านการควอนไตซ์แล้วค่า Y_{ij} จะถูกบีบอัดข้อมูลแบบไม่สูญเสียโดยใช้เอนโทรปี(entropy coder)

จากสมการการแปลงDCT ในสมการ(2.8) ในการคำนวณบล็อกขนาด 8*8 นั้นเราต้องใช้ตัวเลขนั้นมาคูณกัน เท่ากับ 4096 ครั้ง และในรูปภาพ ภาพหนึ่งนั้นจะประกอบไปด้วยหลายๆ บล็อกทำให้ในการคำนวณแปลงค่าโดย DCT นั้นต้องใช้การคำนวณหลายๆ ครั้งทำให้เสียเวลาในการออกแบบและเสียเวลาในการคำนวณเป็นอย่างมาก ดังนั้นเราจึงจำเป็นต้องหาวิธีการที่สามารถคำนวณค่าต่างๆ ได้โดยใช้การคำนวณให้น้อยที่สุด เพื่อที่จะง่ายต่อการออกแบบ และใช้เวลาในการคำนวณได้รวดเร็วขึ้น วิธีการง่ายๆ ที่สามารถช่วยการคำนวณได้ นั้นก็คือ คำนวณ DCT แบบ 1-D ก่อนแล้วค่อยทำการแปลงจาก 1-D DCT ไปเป็น 2-D DCT อีกครั้งหนึ่งซึ่งจะช่วยลดการคำนวณจาก 4096 ครั้ง ไปเป็นคำนวณเพียง 1024 ครั้ง ซึ่งช่วยลดการคำนวณไปถึง 4 เท่า แต่ว่าการคำนวณแบบ 1-D DCT นี้ก็ยังใช้การคำนวณหลายครั้งอยู่ดี เราจึงต้องวิธีการที่ช่วยลดการคำนวณเหล่านี้ จึงได้มีวิธีที่เรียกว่าฟาสฟูร์เรียร์ (Fast fourier transform)[1] ขึ้นมาเพื่อใช้ในการลดการคำนวณ และเป็นที่นิยมกัน

บทที่ 3
หลักการบีบอัดข้อมูลภาพเคลื่อนไหว

3.1 บทนำ

จากบทที่แล้วเราได้กล่าวถึงทฤษฎีพื้นฐานของการบีบอัดข้อมูลแบบสูญเสียของภาพนิ่งมาบ้างแล้ว ซึ่งทฤษฎีพื้นฐานเหล่านั้นสามารถนำมาประยุกต์ใช้งานกับการบีบอัดข้อมูลภาพนิ่งที่ ต่อเนื่องกันหลายๆ ภาพ (ภาพวีดีโอหรือภาพเคลื่อนไหว) ได้เป็นอย่างดี โดยจะดำเนินการกับภาพนิ่งที่ต่อเนื่องกันทีละภาพอย่างเป็นลำดับเหมือนกับการดำเนินการกับภาพนิ่งเพียงภาพเดียว ตัวอย่างเช่น อัตราบิทของข้อมูลเอาท์พุทของ CD-ROM จะมีอัตราการส่งผ่านข้อมูลที่ 1.5 Mbits/s. ซึ่งในกรณีนี้เราต้องการอัตราการบีบอัดข้อมูลถึง 110:1 ดังนั้นในทางปฏิบัติ ถ้ากล่าวถึงการบีบอัดข้อมูลภาพนิ่งที่ต่อเนื่องกัน เราจะต้องทำการบีบอัดข้อมูลให้ได้มากถึง 4 เท่าของภาพนิ่ง จึงจะสามารถนำไปใช้งานได้

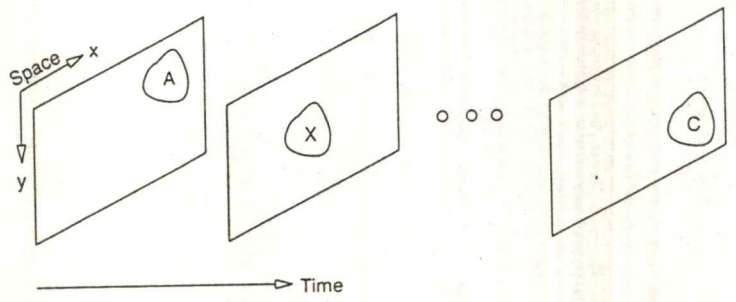
3.2 พื้นฐานของการเข้ารหัสข้อมูลที่เป็นภาพเคลื่อนไหว (Video Coding Basics)

3.2.1 รูปแบบของการลำดับภาพนิ่ง (Image Sequence Model)

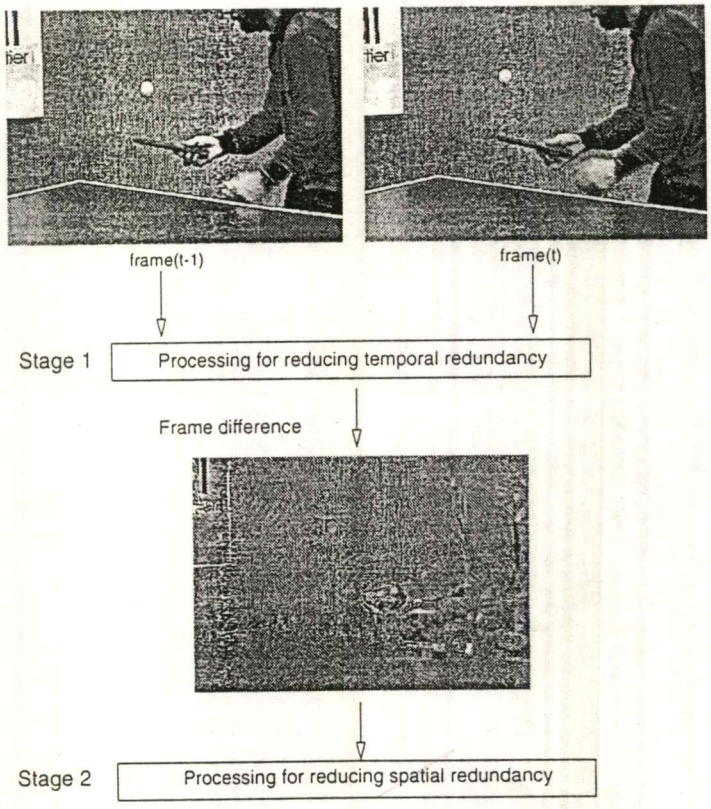
ในกรณีของภาพนิ่ง เราจะสังเกตได้ว่าข้อมูลที่เป็นภาพนิ่งนั้นมีแนวโน้มที่จะมีส่วนของที่ว่างที่ไม่จำเป็น (Spatial redundancy) อยู่เป็นจำนวนมาก ในขณะนี้ปัญหาที่สำคัญในการจับภาพ เคลื่อนไหวของวัตถุ 3 มิติที่เวลาใดๆ เป็นดังรูปที่ 3.1 จากรูปที่ 3.1 จะเห็นว่า ภาพแรกเราสามารถจับภาพของวัตถุได้ ซึ่งก็คือพื้นที่ A โดยภาพๆ นี้จะประกอบไปด้วยพิกเซลทั้งหมดของวัตถุ ถ้าวัตถุนี้เคลื่อนที่ เราก็จะได้ภาพนิ่งภาพถัดไป ซึ่งก็คือพื้นที่ X ดังนั้นเราจึงสามารถคาดเดาได้ว่าจะต้องมีการซ้ำกันของภาพในแต่ละเฟรมในบางพื้นที่เกิดขึ้น ที่เรียกว่า ที่ว่างที่จำเป็นที่เกิดขึ้นชั่วคราว (Temporal redundancy) เป้าหมายสำคัญของการบีบอัดข้อมูลภาพเคลื่อนไหวก็คือ การนำทั้ง ที่ว่างที่ไม่จำเป็น และ ที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราว มาใช้ประโยชน์เพื่อให้ได้การบีบอัดที่ดีที่สุด ส่วนใหญ่แล้วในระบบการเข้ารหัสภาพเคลื่อนไหว จะมีกระบวนการอยู่ 2 กระบวนการที่จะทำให้ได้การบีบอัดที่ดีที่สุด คือ

1. กระบวนการสำหรับการลดที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราว
2. กระบวนการสำหรับการลดที่ว่างที่ไม่จำเป็น

ดังแสดงในรูปที่ 3.2



รูปที่ 3.1 ความสัมพันธ์ของลำดับภาพนิ่ง



รูปที่ 3.2 แสดงกระบวนการการเข้ารหัสภาพเคลื่อนไหวแบบ 2 ขั้นตอน

3.2.2 การลดที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราว (Reducing Temporal Redundancy)

การประมวลผลในทางอุดมคติที่จะช่วยลดที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราวได้ก็คือ การเข้าถึงทุกๆ จุดของพิกเซลจากเฟรมหนึ่งไปยังอีกเฟรมหนึ่ง แต่การกระทำเช่นนี้ย่อมทำให้เกิดสัญญาณรบกวน (noise) ขึ้นด้วย ดังนั้นแทนที่เราจะทำทีละพิกเซล เราควรจะแบ่งข้อมูลให้มีขนาดพื้นที่ 16x16 ที่เรียกว่า มาโครบล็อก (macroblock) ซึ่งเป็นขนาดที่เหมาะสมสำหรับการเพิ่มประสิทธิภาพของการลดที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราว และทำให้ต้องการใช้พลังในการคำนวณในระดับที่พอเหมาะ

ถ้าเรามีเฟรม 2 เฟรมที่ติดกันดังรูปที่ 3.2 โดยสมมติให้เป็นเฟรม (t-1) และ เฟรม (t) ใน ขั้นแรกเราจะทำการแยกเฟรม (t) ออกเป็นพื้นที่ขนาด 16x16 (macroblock) โดยไม่ซ้อนทับกัน จากนั้นเราจะทำการหาว่าในแต่ละบล็อกขนาด 16x16 นั้น สอดคล้องกับพื้นที่จุดสีขนาด 16x16 ของเฟรม (t-1) หรือไม่ ซึ่งผลจากการดำเนินการนี้ กระบวนการการลดที่ว่างที่ไม่จำเป็นที่เกิดขึ้นชั่วคราว จะสร้างตัวแทนของเฟรม (t) ซึ่งมีเพียงส่วนที่เป็นความแตกต่างของทั้ง 2 เฟรมเท่านั้น ถ้าหากเฟรมทั้ง 2 เฟรมนั้นมีความเหมือนกันมาก (High degree of temporal redundancy) เฟรมที่แสดงความ แตกต่าง (Different frame) ก็จะได้ค่าที่ใกล้เคียง 0 เป็นจำนวนมาก ดังแสดงในรูปที่ 3.2 จะเห็นว่าภาพทั้ง 2 เฟรมมีความเหมือนกันมาก ทำให้เอาท์พุทที่ออกมาจากขั้นตอนที่ 1 (Stage 1) มีความสว่างน้อยกว่าภาพเดิมและทำให้ได้การบีบอัดที่ดีกว่า แต่ถ้าเฟรม (t) กับเฟรม (t-1) ต่างกันโดยสิ้นเชิง มันก็จะไม่สามารถหาพื้นที่ที่ซ้ำกันระหว่าง 2 เฟรมนั้นได้ ดังนั้นเราก็จะไม่ได้ประโยชน์จากวิธีการของขั้นตอนที่ 1 เลย ในระบบการบีบอัดข้อมูลภาพเคลื่อนไหวนั้น วิธีการบีบอัดที่ใช้ในการลดที่ว่างที่ไม่ จำเป็นที่เกิดขึ้นชั่วคราว เราจะเรียกว่า ตัวเข้ารหัสระหว่างเฟรม (Interframe coder)

3.2.3 การลดที่ว่างที่ไม่จำเป็น (Reducing Spatial redundancy)

หลังจากขั้นตอนที่ 1 เราจะได้เฟรมที่แสดงถึงความแตกต่างของ 2 เฟรมที่ติดกัน ดังนั้นเราจะสามารถหาประโยชน์จากพิกเซลที่เหมือนกันเหล่านั้นได้ เพื่อให้ได้ผลของการ บีบอัดที่เป็นตัวแทนของเฟรม (t) ซึ่งกระบวนการนี้จะแสดงในขั้นตอนที่ 2 (Stage 2) ดังรูปที่ 3.2 มาตรฐานการเข้ารหัสภาพเคลื่อนไหวจะใช้การเข้ารหัสแบบ DCT เพื่อลดที่ว่างที่ไม่จำเป็น ในระบบการบีบอัดข้อมูลภาพเคลื่อนไหวนั้น วิธีการที่ใช้ในการลดที่ว่างที่ไม่จำเป็น เราจะเรียกว่า ตัวเข้ารหัสภายในเฟรม (Intraframe coder) การรวมเอาการเข้ารหัสระหว่างเฟรม (Interframe coding) และ การเข้ารหัสภายในเฟรม(Intraframe coding) เข้าด้วยกันดังแสดงในรูปที่ 3.2 นั้น เราเรียกว่า วิธีการเข้ารหัสแบบไฮบริด (ภายในเฟรม / ระหว่างเฟรม) [a hybrid (intraframe / interframe) coding method]

3.2.4 การชดเชยการเคลื่อนที่ (Motion Compensation)

จากกระบวนการที่เปรียบเทียบเฟรม 2 เฟรมที่ติดกันเพื่อหาความแตกต่างของ 2 เฟรม เราอ้างถึงการทำนายที่ต้องใช้การชดเชยการเคลื่อนที่ เราสามารถอธิบายการชดเชยการเคลื่อนที่ได้ว่าเป็นกระบวนการที่ชดเชยการขจัดของวัตถุที่เคลื่อนไหวจากเฟรมหนึ่งไปยังอีกเฟรมหนึ่ง ในทางปฏิบัติการชดเชยการเคลื่อนที่จะถูกดำเนินการด้วยการประมาณค่าการเคลื่อนที่ (Motion Estimation) ซึ่งเป็นการระบวนการที่หาพิกเซลที่สอดคล้องกันระหว่างเฟรมที่ต้องการ โดยสามารถเขียนสมการการชดเชยการเคลื่อนที่ได้เป็น

$$e(x,y,t) = I(x,y,t) - I(x-u,y-v,t-1) \qquad (3.1)$$

โดยที่ I(x,y,t) คือ ค่าของพิกเซลที่ตำแหน่ง (x,y) ในเฟรม (t)

I(x-u,y-v,t-1) คือ ค่าของพิกเซลที่สอดคล้องกับ I(x,y,t) ที่ตำแหน่ง (x-u,y-v) ในเฟรม (t-1)

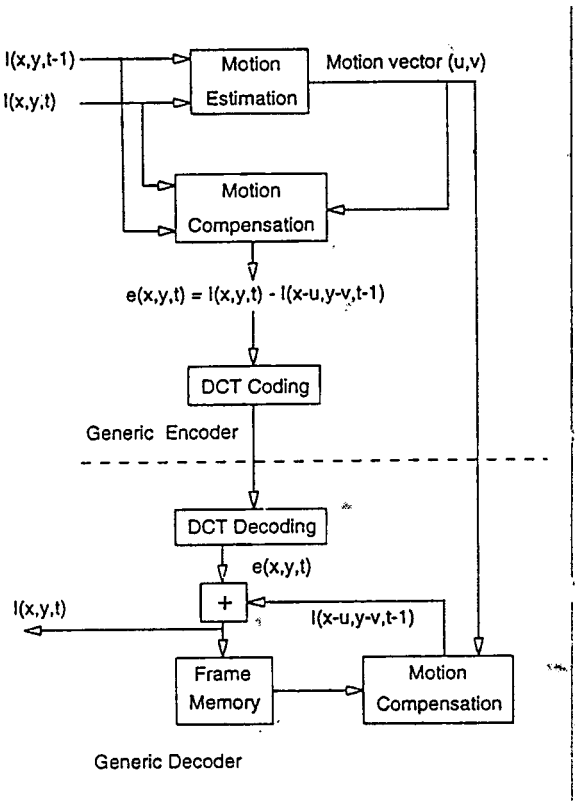
จากกระบวนการนี้จะได้เอาท์พุทที่เป็นความสัมพันธ์ของการเคลื่อนที่ของ 1 บล็อกจากเฟรมหนึ่งไปยังอีกเฟรมหนึ่ง ในรูปเวคเตอร์ (u,v) จากรูปที่ 3.3 แสดงบล็อกไดอะแกร็มของการเข้ารหัสแบบไฮบริด (hybrid coding) สังเกตได้ว่า การชดเชยการเคลื่อนที่ จะมีอยู่ทั้งในตัวเข้ารหัสและตัวถอดรหัส แต่การประมาณค่าการเคลื่อนที่นั้นจะมีอยู่ในตัวเข้ารหัสเท่านั้น ในทางปฏิบัติ ตัวเข้ารหัส (coder) จะต้องรวมบล็อกของการทำงานของหน่วยความจำของบัฟเฟอร์ (buffer memory), กระบวนการที่เพิ่มเติมก่อนกระบวนการจริง (additional preprocessing) และกระบวนการหลังกระบวนการจริง (postprocessing) เอาไว้ด้วย ซึ่งไม่ได้แสดงไว้ในรูปนี้

เมื่อเรารวมการทำงานของการเข้ารหัสระหว่างเฟรมกับการเข้ารหัสภายในเฟรมเข้าด้วยกันแล้ว จะทำให้อัตราการบีบอัดดีกว่าการทำงานเพียงลำพังกระบวนการเดียว

3.3 การทำนายสำหรับการชดเชยการเคลื่อนที่ (Motion Compensated prediction)

จากที่ได้กล่าวมาแล้ว วิธีการเข้ารหัสแบบไฮบริด (ระหว่างเฟรม / ภายในเฟรม) นั้น จะทำให้การบีบอัดข้อมูลภาพเคลื่อนไหวมีประสิทธิภาพมากขึ้น การเข้ารหัสภาพเคลื่อนไหวที่ใช้พื้นฐานของวิธีนี้แสดงให้เห็นดังรูปที่ 3.3 สิ่งที่สำคัญที่สุดอย่างหนึ่งในการเข้ารหัสระหว่างเฟรมก็คือ กระบวนการการประมาณค่าการเคลื่อนที่นั่นเอง

จากรูปที่ 3.4 แสดงให้เห็นถึงปัญหาของกระบวนการการประมาณค่าการเคลื่อนที่ในมาตรฐานการเข้ารหัสภาพเคลื่อนไหว จากรูปเราให้ภาพอ้างอิงมา 1 ภาพ รวมทั้งมาโครบล็อกขนาด NxM ของภาพปัจจุบันมาด้วย การประมาณค่าการเคลื่อนที่จะเป็นตัวกำหนดบล็อกขนาด NxM ที่อยู่ในภาพอ้างอิงซึ่งสอดคล้องกัน (match) กับลักษณะของมาโครบล็อกของภาพปัจจุบันมากที่สุด โดยที่ภาพปัจจุบันเราให้เป็นภาพนิ่งหรือเฟรมที่เวลา t ส่วนที่ภาพอ้างอิงเราจะกำหนดให้เป็นภาพนิ่งหรือ เฟรมทั้งเวลาที่เป็นอดีตคือ t-n สำหรับการประมาณค่าการเคลื่อนที่แบบไปข้างหน้า (forward motion estimation) และเวลาที่เป็นอนาคต t+k สำหรับการประมาณค่าการเคลื่อนที่ที่ย้อนกลับ (backward motion estimation) ในกรณีนอกเหนือจากนี้ เรขาคณิตของบล็อกที่สอดคล้องกัน (matching block) ในภาพอ้างอิงไม่จำเป็นต้องมีรูปทรงเหมือนบล็อกที่อยู่ในปัจจุบัน แต่อย่างไรก็ตามในมาตรฐานของการเข้ารหัสภาพเคลื่อนไหว เพียงแค่ใช้รูปทรงสี่เหลี่ยมเป็นเรขาคณิตของบล็อกที่สอดคล้องกันก็เพียงพอแล้ว

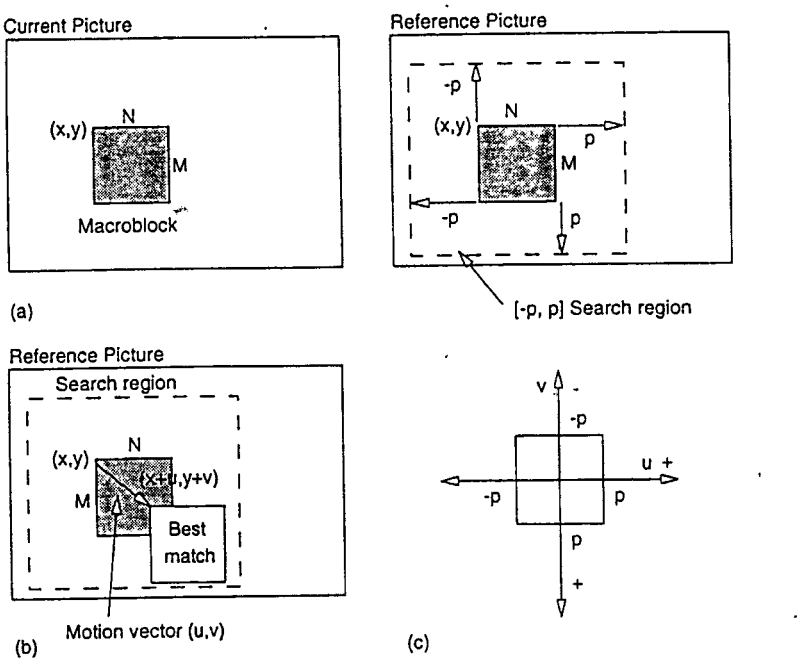


รูปที่ 3.3 การทำงานทั่วไปของตัวเข้ารหัสและตัวถอดรหัสภาพเคลื่อนไหวแบบไฮบริด

ตำแหน่งของพื้นที่มาโครบล็อกมักจะถูกกำหนดให้เป็นคู่ลำดับ(x,y) ซึ่งเป็นตำแหน่งที่มุมซ้ายด้านบนของพื้นที่ ในทางอุดมคติเราต้องการที่จะค้นหา (search) ภาพอ้างอิงทั้งภาพ เพื่อที่จะได้บล็อกที่สอดคล้องกันมากที่สุด แต่ในทางปฏิบัติเราไม่สามารถทำได้ ดังนั้นเราจึงจำกัดการค้นหา ภายในพื้นที่ [-p,p] รอบๆ จุดกำเนิดของมาโครบล็อกในภาพปัจจุบันเท่านั้น

กำหนดให้ (x+u,y+v) เป็นตำแหน่งของบล็อกที่สอดคล้องมากที่สุดของภาพอ้างอิง ดังแสดงในรูปที่ 3.4(b) เวคเตอร์จากตำแหน่ง (x,y) ถึงตำแหน่ง (x+u,y+v) หมายถึงเวคเตอร์ของการเคลื่อนที่ (motion vector) ที่บอกตำแหน่งของมาโครบล็อกซึ่งก็คือ ตำแหน่ง (x,y) นั่นเอง เรามักจะสมมติให้ตำแหน่ง (x,y) มีค่าเป็น (0,0) ดังนั้นเวคเตอร์ของการเคลื่อนที่จึงถูกกำหนอให้เป็น (u,v)

ในมาตรฐานการเข้ารหัสภาพเคลื่อนไหว N=M=16 จากรูปที่ 3.4(a) สำหรับพื้นที่ของการค้นหาจะเห็นว่า $-p \le u \le p$ และ $-p \le v \le p$



รูปที่ 3.4 กระบวนการของการประมาณค่าการเคลื่อนที่

3.4 อัลกอริทึมของการประมาณค่าการเคลื่อนที่

มีวิธีการหลายวิธีที่ใช้ในการค้นหาการประมาณค่าของการเคลื่อนที่ แต่ในมาตรฐานการเข้ารหัสภาพเคลื่อนไหวนั้นไม่ได้มีข้อกำหนดที่ตายตัวว่าให้ใช้วิธีใด ดังนั้นเราจึงสามารถนำวิธีการใดไปประยุกต์ใช้ให้ตรงตามความต้องการก็ได้ โดยทั่วไปการค้นหาการประมาณค่าการเคลื่อนที่วิธีการต่างๆ สามารถสรุปได้ดังนี้

1. วิธีการค้นหาแบบเต็มรูปแบบ (Full - Search Method)

2. วิธีการค้นหาแบบลอการิทึมสองมิติ (Two - Dimensional Logarithmic Search)

3. วิธีการค้นหาแบบลำดับชั้นคู่ขนานหนึ่งมิติ (Parallel Hierarchical One - Dimensional Search (PHODS))

4. วิธีการประมาณค่าการเคลื่อนที่แบบลำดับชั้น (Hierarchical Motion Estimation)

3.4.1 วิธีการค้นหาแบบเต็มรูปแบบ (Full - Search Method)

วิธีที่ง่ายที่สุดในการค้นหาเวคเตอร์ของการเคลื่อนที่สำหรับแต่ละมาโคร-บล็อกก็คือ การคำนวณหา MAE(i,j) ของแต่ละตำแหน่งในพื้นที่ของการค้นหา (search space) ซึ่งเรียกว่า อัลกอริทึมของการค้นหาแบบเต็มรูปแบบ แต่วิธีนี้มีการคำนวณที่ยุ่งยากมาก แต่ก็ได้รับการรับรองว่าจะสามารถหาค่า MAE ที่น้อยที่สุดได้ ดังนั้นเราจึงหาวิธีปรับปรุงและพัฒนาวิธีการใหม่ๆ ขึ้นมาเพื่อลดความยุ่งยากในการคำนวณดังกล่าว แต่ก็ยังไม่สามารถรับรองได้ว่าจะได้ค่า MAE ที่น้อยที่สุดหรือไม่

3.4.2 วิธีการค้นหาแบบลอการิทึมสองมิติ (Two - Dimensional Logarithmic Search)

วิธีการค้นหาแบบนี้คล้ายกับการค้นหาแบบไบนารี่มาก ขั้นแรกเราจะแบ่งพื้นที่ที่ต้องการค้นหา [-p,p] ออกเป็น 2 ส่วน ส่วนแรกจะอยู่ภายในพื้นที่รูปสี่เหลี่ยม $[\frac{-p}{2},\frac{p}{2}]$ (ที่ความละเอียดเป็นจำนวนเต็มของจุดสี) ส่วนที่ 2 จะอยู่ภายนอกพื้นที่ดังกล่าว นอกจากนั้นแทนที่เราจะค้นหาภายในพื้นที่ $[\frac{-p}{2},\frac{p}{2}]$ ทั้งหมด เราจะคำนวณหาการทำงาน (function) ของ MAE สำหรับ 9 ตำแหน่งเท่านั้น กล่าวคือ ที่ตำแหน่ง (0,0) และอีก 8 จุดที่สำคัญภายในพื้นที่ $[\frac{-p}{2},\frac{p}{2}]$ ซึ่งถ้าระยะทางระหว่างจุดเหล่านี้คือ d_1 เราจะสามารถหา MAE ที่มีค่าน้อยที่สุดได้จากการคำนวณ MAE ที่จุด (0,0) , $(0,d_1)$, $(0,-d_1)$, $(-d_1,0)$, $(d_1,0)$, (d_1,d_1) , $(d_1,-d_1)$, $(-d_1,d_1)$ และ $(-d_1,-d_1)$ โดยระยะ d_1 จะถูกกำหนดเป็น

$$d_1 = 2^{k-1} \qquad (3.2)$$

โดยที่ $k = \lceil \log_2 p \rceil$

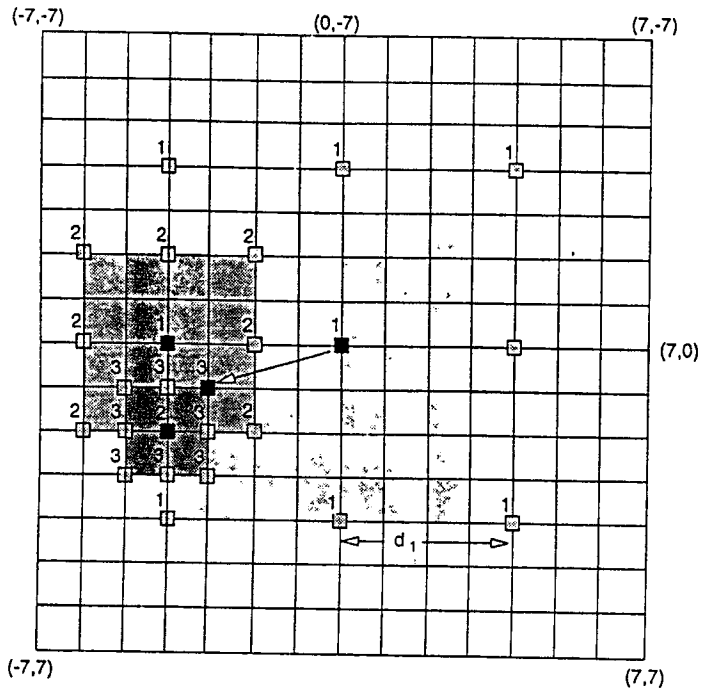
ตัวอย่างเช่น สำหรับค่า $p = 7$, $k = 3$ และ $d_1 = 4$ เราจะใช้ตำแหน่งที่สอดคล้องกันมากที่สุดเป็นจุดกึ่งกลาง แล้วเราจะทำการค้นหาพื้นที่ที่สอดคล้องกันมากที่สุดต่อไปภายใน 8 จุดที่ถูกกำหนดโดยระยะ $d_2 = d_1 / 2$ เราจะทำการคำนวณและค้นหาไปจนกว่าจะครบการค้นหาครั้งที่ k ซึ่ง 8 จุดภายในพื้นที่ที่ค้นหานั้นจะถูกกำหนดให้ระยะเป็น 1 หลังจากที่ 8 จุดนั้นได้ถูกกำหนดแล้ว เราก็จะได้ค่า MAE ที่น้อยที่สุดนั่นเอง

จากรูปที่ 3.5 แสดงให้เห็นกระบวนการการค้นหาแบบลอการิทึมสองมิติที่นิยมใช้กันมาก ซึ่งเรียกว่า การค้นหาแบบ 3 ขั้นตอน (three - step search หรือ TSS) ซึ่งเราให้ k=3 และ p=7 ในการประยุกต์ใช้งานสำหรับข้อมูลภาพเคลื่อนไหวนั้นค่า p=7 ก็เพียงพอแล้ว

การค้นหาที่แบ่งออกเป็น 3 ขั้นตอนดังกล่าว แสดงให้เห็นโดยตัวเลขที่กำกับเป็น 1,2 และ 3 ในรูปที่ 3.5 นั่นเอง ซึ่งรายละเอียดมีดังต่อไปนี้

- ขั้นแรก เราจะเริ่มที่ตำแหน่ง (0,0) เป็นจุดกึ่งกลาง จากนั้นเราจะคำนวณหา 8 จุดที่เหลือ ซึ่งแสดงได้ดังรูป ตรงตำแหน่งที่มีเลขกำกับว่า 1 ระยะระหว่างจุดเหล่านั้น คือ d_1 และสามารถสันนิษฐานได้ว่า ค่า MAE ที่น้อยที่สุดภายในพื้นที่ที่ค้นหาคือ (-4,0)

- ขั้นที่สอง เราจะใช้ตำแหน่ง (-4,0) เป็นจุดกึ่งกลางในการค้นหา 8 จุดที่เหลือนั้นต่อไป โดยมีเลขกำกับเป็นตัวเลข 2 ดังรูป ซึ่งกำหนดให้ระยะระหว่างจุดเหล่านั้นคือ $d_2 = d_1 / 2$ และสามารถสันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดอยู่ที่ตำแหน่ง (-4,2)



รูปที่ 3.5 ตัวอย่างของวิธีการค้นหาแบบ 3 ขั้นตอน (three - step search หรือ TSS)

- ขั้นที่สาม เราจะใช้ตำแหน่ง (-4,2) เป็นจุดกึ่งกลางในการค้นหา 8 จุดที่เหลือเช่นกัน โดยระยะระหว่างจุดเหล่านั้นคือ $d_3 = 1$ และเราสามารถสันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดอยู่ที่ตำแหน่ง (-3,1) กระบวนการค้นหาจะสิ้นสุดที่จุดๆ นี้ เพราะการแบ่งพื้นที่ของการค้นหาไม่สามารถกระทำได้อีกต่อไปแล้ว และเราจะได้เวคเตอร์ของการเคลื่อนที่ที่ถูกกำหนดเป็นคู่ลำดับ (-3,1)

3.4.3 วิธีการค้นหาแบบลำดับชั้นคู่ขนานหนึ่งมิติ (Parallel Hierarchical One - Dimensional Search หรือ PHODS)

สิ่งที่แตกต่างจากวิธีการค้นหาแบบลอการิทึมก็คือ การค้นหาแบบนี้สามารถทำได้โดยอิสระภายใน 2 มิติ โดยมีอัลกอริทึม ดังนี้

1. สำหรับพื้นที่ค้นหา [-p,p] จากรูปที่ 3.4 ให้ $S = 2^{\lfloor \log_2 p \rfloor}$ และให้จุดกำเนิดของพื้นที่ที่ต้องการค้นหาคือ (0,0) สิ่งที่ควรตระหนักถึงคือ จุดกำเนิดดังกล่าวนั้นเสมือนจุด (di,dj)

2. ในขณะเดียวกัน เราจะคำนวณ

- แกน i ที่น้อยที่สุด กล่าวคือ ระหว่างจุด 3 จุด ต่อไปนี้ (di-S,dj) , (di,dj) , (di+S,dj) เราจะหาตำแหน่งที่ให้ค่า MAE ที่น้อยที่สุด โดยเราจะทำให้ di อยู่ในคู่ลำดับของ i ของตำแหน่งนี้

- แกน j ที่น้อยที่สุด กล่าวคือ ระหว่างจุด 3 จุด ต่อไปนี้ (di,dj-S) , (di,dj) , (di,dj+S) เราจะหาตำแหน่งที่ให้ค่า MAE ที่น้อยที่สุด โดยเราจะทำให้ dj อยู่ในคู่ลำดับของ j ของตำแหน่งนี้ และให้ $S = S / 2$

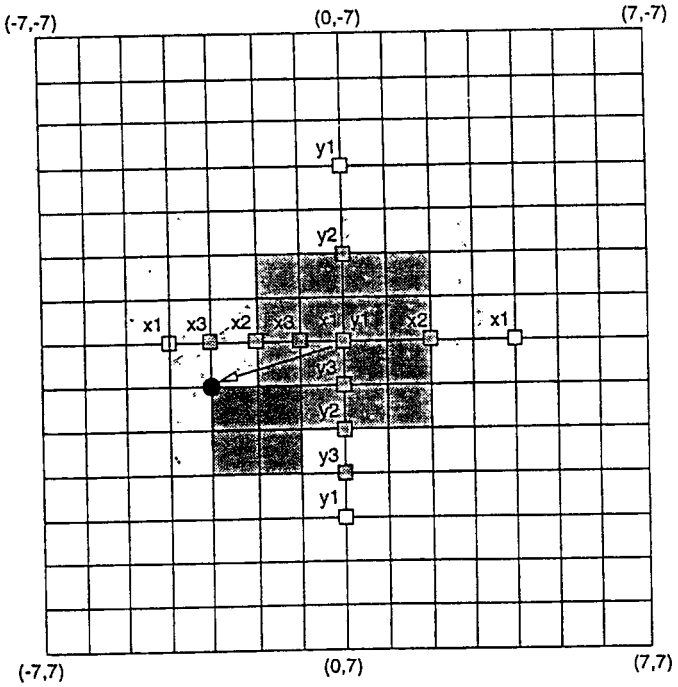
ทำซ้ำขั้นตอนที่สอง จนกระทั่งค่า S = 0 ค่า (di,dj) ที่ได้จะเป็นเวคเตอร์ของการเคลื่อนที่ที่ทำให้ได้ค่ามาโครบล็อกที่สอดคล้องที่สุดในภาพปัจจุบัน

กระบวนการที่แสดงในรูปที่ 3.6 เราจะให้ค่า p = 7 ซึ่งเราอาจจะต้องการเปรียบเทียบกับกระบวนการการค้นหาแบบที่แล้วในรูปที่ 3.5 ได้เช่นกัน

เริ่มต้นที่ค่า S = 4

1. ขั้นแรกเราต้องหาแกน i ที่น้อยที่สุดก่อน โดยการคำนวณค่า MAE ของ 3 จุดดังแสดงได้ด้วยสัญลักษณ์ x_1 ในรูปที่ 3.6 สันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดถูกกำหนดด้วยค่า i = 0 ในขณะเดียวกันเราก็ต้องคำนวณหาแกน j ที่น้อยที่สุดเช่นกัน ซึ่งในรูปที่ 3.6 เราจะแสดงด้วยสัญลักษณ์ y_1 และสันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดถูกกำหนดด้วยค่า j = 0 ดังนั้นจุดกำเนิดใหม่จะอยู่ที่ (0,0)

2. ระยะห่างระหว่างจุดถูกลดลงให้เหลือ 2 ค่าแกน i ที่น้อยที่สุดจะถูกกำหนดจากจุดกึ่งกลางใหม่ที่หาได้จากขั้นตอนที่แล้ว โดยจะแสดงได้ด้วยสัญลักษณ์ x_2 ดังรูปที่ 3.6 สันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดอยู่ที่ค่า i = -2 และสำหรับการหาค่าแกน j ที่น้อยที่สุดนั้น เราจะแสดงด้วยสัญลักษณ์ y_2 และสันนิษฐานได้ว่า ค่า MAE ที่น้อยที่สุดอยู่ที่ค่า j = 2 ดังนั้นจุดกำเนิดของการค้นหาใหม่อยู่ที่ตำแหน่ง (-2,2)



รูปที่ 3.6 ตัวอย่างของวิธีการค้นหาแบบ PHODS

3. สำหรับขั้นตอนนี้ ค่า S = 1 และค่า MAE ที่น้อยที่สุดตามแนวแกน i จะแสดงด้วยสัญลักษณ์ x_3 และในทำนองเดียวกัน ค่า MAE ที่น้อยที่สุดตามแนวแกน j ก็จะถูกแสดงด้วยสัญลักษณ์ y_3 เช่นเดียวกัน ดังนั้นเราสามารถสันนิษฐานได้ว่า ค่า MAE ที่น้อยที่สุดอยู่ที่ค่า (-3,1)

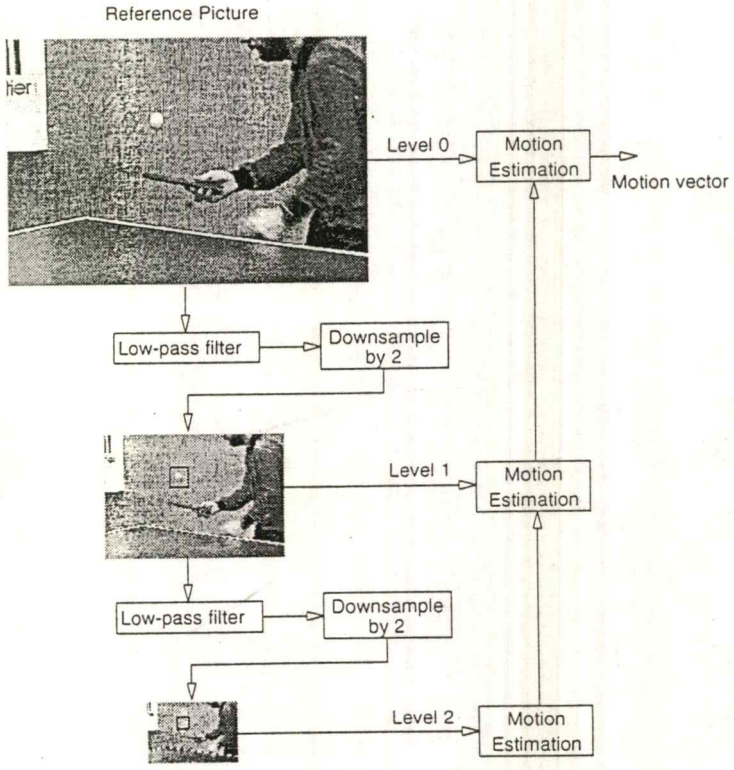
ด้วยเหตุผลที่ว่าค่า S=1 คือค่าเริ่มต้นของขั้นตอน การที่จะลดหรือแบ่งพื้นที่จึงไม่สามารถกระทำได้ เราจึงสรุปได้ว่า ค่าMAE ที่น้อยที่สุดอยู่ที่ตำแหน่ง (-3,1) และดังนั้นเวคเตอร์ของการเคลื่อนที่ที่ต้องการสำหรับมาโครบล็อกที่สอดคล้องที่สุดในภาพปัจจุบัน ก็คือ ค่าของตำแหน่งนี้นั่นเอง

PHODS มีข้อดีกว่า TSS อยู่ 1 ประการคือ

1. มีการทำงานที่เป็นระเบียบ เพราะการค้นหาตำแหน่งที่ต้องการนั้นจะกระทำไปตามแนวแกน i และ j เสมอ

3.4.4 วิธีการประมาณค่าการเคลื่อนที่แบบลำดับชั้น (Hierarchical Motion Estimation)

วิธีการประมาณค่าการเคลื่อนที่หลายวิธีได้นำการค้นหาตำแหน่งที่น้อยลงมารวมกันกับการคำนวณหาพิกเซลที่น้อยลง โดยใช้การคำนวณหาMAEซึ่งเราจะเรียกวิธีการนี้ว่า วิธีค้นหาแบบลำดับชั้น วิธีการนี้มีกระบวนการที่ทำได้หลายแบบ รูปที่3.8นี้เป็นเพียงการทำงานแบบหนึ่งของวิธีนี้เท่านั้น



รูปที่ 3.7 วิธีการทั่วไปของการค้นหาเวคเตอร์ของการเคลื่อนที่ในวิธีการค้นหาแบบลำดับชั้น

กระบวนการค้นหาเวคเตอร์ของการเคลื่อนที่ มีขั้นตอนดังต่อไปนี้

1. เราจะกำหนดให้ภาพปัจจุบันและภาพอ้างอิงมีค่าความละเอียดต่ำหลายค่า ดังรูปที่ 3.7 แสดงให้เห็นถึงภาพอ้างอิงที่มีค่าความละเอียดต่ำ 2 ค่า คือที่ ระดับที่ 1 (level 1) และ ระดับที่ 2 (level

2) ให้เราสันนิษฐานว่า มาโครบล็อกในภาพปัจจุบันมีตำแหน่งอยู่ที่ (x,y) ดังนั้นในระดับที่ 1 และระดับที่ 2 มาโครบล็อกที่มีความคล้ายคลึงกัน จะอยู่ที่ตำแหน่ง $(\frac{x}{2},\frac{y}{2})$ และ $(\frac{x}{4},\frac{y}{4})$ ตามลำดับ และให้เราสันนิษฐานว่าในระดับที่ 0 (level 0) มาโครบล็อกจะมีขนาด 16x16 และมีเวคเตอร์ของการเคลื่อนที่อยู่ในช่วงพิกเซลที่ ±p

2. เราจะเริ่มค้นหาเวคเตอร์ของการเคลื่อนที่ที่ระดับที่ 2 เป็นอันดับแรก เราจะใช้พารามิเตอร์ของการค้นหาเป็น $\frac{p}{4}$ แทนที่จะใช้ p ดังนั้นวิธีการหาเวคเตอร์ของการเคลื่อนที่ในระดับที่ 2 จึงสามารถใช้ได้ทั้งวิธีของการค้นหาแบบเต็มรูปแบบและการค้นหาแบบลอการิทึม ซึ่งทั้ง 2 วิธีการนี้จะใช้มาโครบล็อกขนาด 4x4 , ใช้พารามิเตอร์ในการค้นหาคือ $\frac{p}{4}$ และจุดเริ่มต้นของการค้นหาอยู่ที่ตำแหน่ง $(\frac{x}{4},\frac{y}{4})$ ทำให้เราสามารถสันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดอยู่ที่ตำแหน่ง (u_2,v_2)

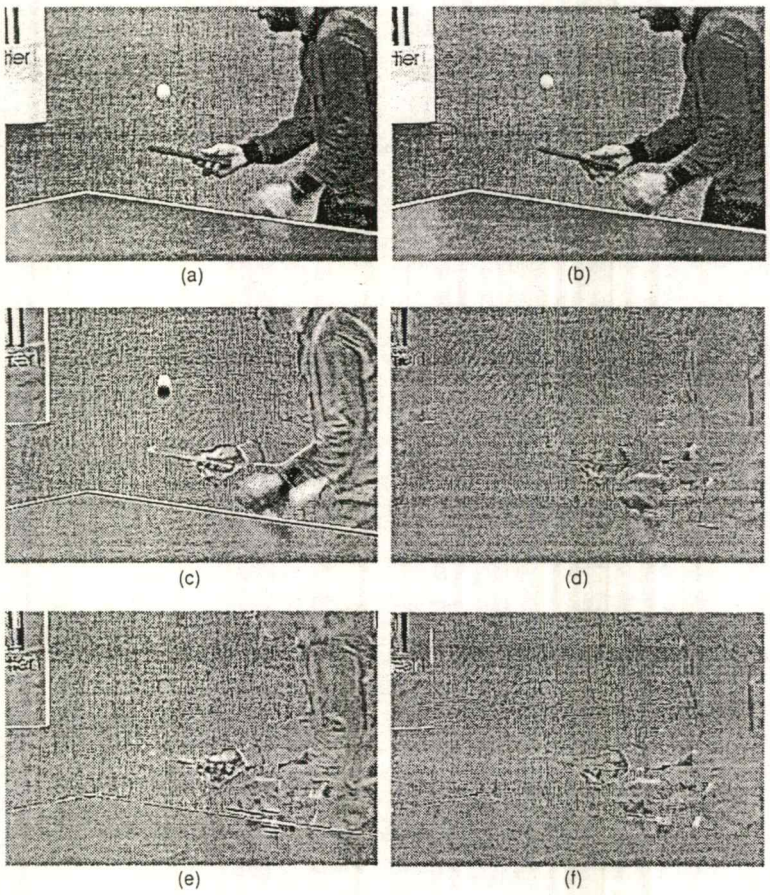
3. ที่ระดับที่ 1 เราจะสามารถค้นหาเวคเตอร์ของการเคลื่อนที่ได้โดยใช้มาโครบล็อกขนาด 8x8 ซึ่งจุดเริ่มต้นการค้นหาอยู่ที่ $(\frac{x}{2}+2u_2,\frac{y}{2}+2v_2)$ และพื้นที่ของการค้นหาอยู่ที่ตำแหน่ง [-1,1] รอบๆ จุดเริ่มต้น และสามารถสันนิษฐานได้ว่าค่า MAE ที่น้อยที่สุดอยู่ที่ตำแหน่ง (u_1,v_1)

4. ที่ระดับที่ 0 เรากำหนดให้จุดเริ่มต้นของการค้นหาอยู่ที่ $(x+2u_1 , y+2v_1)$ และพื้นที่ของการค้นหาอยู่ที่ตำแหน่ง [-1,1] รอบๆ จุดเริ่มต้น ซึ่งเราจะใช้มาโครบล็อกขนาด 16x16 โดยใช้วิธีการค้นหาแบบใดก็ได้ ซึ่งก็รวมทั้งวิธีการค้นหาแบบเต็มรูปแบบด้วย ค่า MAE ที่น้อยที่สุดที่ได้นั้นจะเป็นค่าเวคเตอร์ของการเคลื่อนที่สุดท้ายที่เอาท์พุท

การประมาณค่าการเคลื่อนที่ของวิธีแบบลำดับชั้นนี้จะช่วยลดความยุ่งยากเมื่อเปรียบเทียบกับวิธีการค้นหาแบบเต็มรูปแบบลง 29.89 GOPS ซึ่งแสดงให้เห็นว่าวิธีการนี้เป็นวิธีที่มีประสิทธิภาพมาก แต่อย่างไรก็ตามวิธีการนี้ยังต้องการการเก็บข้อมูลที่ดีพอ ยิ่งไปกว่านั้นเราอาจจะได้เวคเตอร์ของการเคลื่อนที่ที่ไม่เที่ยงตรงนักโดยเฉพาะในบริเวณที่เป็นวัตถุขนาดเล็ก เพราะวิธีการนี้เราต้องเริ่มจากภาพที่มีความละเอียดต่ำก่อน ทำให้บริเวณที่เป็นวัตถุขนาดเล็กอาจถูกละเลยไป แต่ในทางกลับกัน การที่เรากำหนดให้ภาพเริ่มแรกมีค่าความละเอียดต่ำจะเป็นผลทำให้ไม่เกิดสัญญาณรบกวนด้วยเช่นกัน

Search Method	Operations per Macroblock	Operations for pictures 720 x 480 at 30 fps	
		$p = 15$	$p = 7$
Full-search	$(2p+1)^2NM3$	29.89 GOPS	6.99 GOPS
Logarithmic	$(8\lceil log_2 p\rceil + 1)NM3$	1.02 GOPS	777.60 MOPS
PHODS	$(4\lceil log_2 p\rceil + 1)NM3$	528.76 MOPS	404.35 MOPS
Hierarchical	$[(2\lceil \frac{p}{4}\rceil + 1)^2 + 180]\ \frac{NM}{16}3$	507.38 MOPS	398.52 MOPS

ตารางที่ 3.1 เปรียบเทียบคุณสมบัติของการค้นหาแบบวิธีต่างๆ



รูปที่ 3.8 การเปรียบเทียบการค้นหาการประมาณค่าการเคลื่อนที่วิธีการต่างๆ
(a) ภาพอ้างอิง : (b) ภาพปัจจุบัน : (c) ภาพที่แสดงให้เห็นความแตกต่างระหว่างเฟรมแบบง่ายๆ :
(d) วิธีการค้นหาแบบเต็มรูปแบบ : (e) วิธีการค้นหาแบบลอการิทึม : (f) วิธีการค้นหาแบบ 3 ขั้นตอน

จากรูปที่ 3.8 แสดงให้เห็นถึงวิธีการค้นหาการประมาณค่าการเคลื่อนที่วิธีการต่างๆ โดยใช้ค่า p = 15 และ มาโครบล็อกขนาด 16x16

ในรูปที่ 3.8 (c) ถึง 3.8 (f) แสดงให้เห็นถึงการหาการทำนายความผิดพลาดของการชดเชยการเคลื่อนที่ ถ้าการประมาณการเคลื่อนที่ที่หามาได้นั้นถูกต้อง เราก็จะได้ภาพที่แสดงความแตกต่างเป็นสีดำสนิท

จากรูปที่ 3.8(c) แสดงให้เห็นถึงการทำนายความผิดพลาด ในกรณีของเวคเตอร์ของการเคลื่อนที่ที่มีค่าเป็นศูนย์ (zero - valued motion vector) วิธีการค้นหาแบบเต็มรูปแบบจะเป็นวิธีการที่ดีที่สุดในการเปรียบเทียบความแตกต่างระหว่างภาพปัจจุบันและภาพอ้างอิง ส่วนวิธีการค้นหาแบบลอการิทึมนั้นจะเป็นวิธีการที่แย่ที่สุด เนื่องมาจากการที่มีค่า MAE ที่น้อยที่สุดที่ไม่ใช่ค่า MAE ที่น้อย

ที่สุดที่แท้จริงนั่นเอง การที่เราใช้ตัวกรองความถี่ต่ำผ่าน (low pass filter) จะช่วยลดการมีค่า MAE ที่น้อยที่สุดหลายค่าได้

จากรูปที่ 3.8 (f) แสดงให้เห็นว่าวิธีการการค้นหาแบบ 3 ขั้นตอนนี้ มีความใกล้เคียงกับวิธีการค้นหาแบบเต็มรูปแบบมาก

บทที่ 4

มาตรฐาน MPEG

4.1บทนำ

ความก้าวหน้าของเทคโนโลยีทางด้านภาพเคลื่อนไหว ในระบบดิจิตอลและการเก็บรักษาภาพเคลื่อนไหวในระบบดิจิตอล นำไปสู่การนำภาพเคลื่อนไหวในระบบดิจิตอลไปประยุกต์ใช้งานทางด้านมัลติมีเดีย ในปี 1988 ได้มีการตื่นตัวที่จะสร้างรูปแบบทั่วไปของการเข้ารหัสและเก็บข้อมูลภาพเคลื่อนไหวในระบบดิจิตอล มาตรฐานองค์กรนานาชาติ (ISO) ได้ก่อตั้ง MPEG (Moving Picture Expert Group) ขึ้นมาเพื่อพัฒนามาตรฐานการเข้ารหัสข้อมูลภาพเคลื่อนไหวรวมเข้ากับข้อมูลเสียงบนสื่อเก็บข้อมูลดิจิตอล MPEG รู้จักกันอย่างเป็นทางการในชื่อ ISO - IEC/JTC1 SC29/WG11 ซึ่งการทำงานครั้งแรกได้เสร็จสิ้นเมื่อปี 1991 ด้วยการพัฒนามาตรฐาน ISO 11172 ซึ่งเป็นการเข้ารหัสของภาพเคลื่อนไหวโดยทำงานร่วมกับระบบเสียง สำหรับการเก็บรักษาข้อมูลบนสื่อเก็บข้อมูลดิจิตอลที่มีการส่งผ่านข้อมูลที่ 1.5 เมกะบิทต่อวินาที มาตรฐานแรกนี้รู้จักกันในชื่อว่า MPEG-1

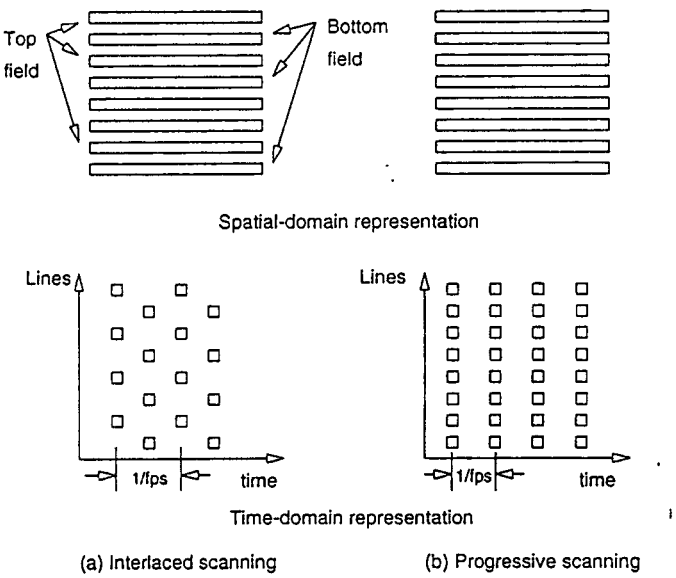
ในปี 1990 MPEG ได้เริ่มงานที่ 2 นั่นก็คือการพัฒนา MPEG-1 ออกไปอีก โดยให้มีความยืดหยุ่นของรูปแบบข้อมูลเข้ามากขึ้น อัตราการส่งผ่านข้อมูลดีขึ้น และลดความผิดพลาดในการ ถอดรหัสลง โดยให้ชื่อว่า มาตรฐาน ISO 13818 ซึ่งเป็นการเข้ารหัสทั่วไปของภาพเคลื่อนไหวโดยทำงานร่วมกับระบบเสียง หรือ MPEG-2 และได้เริ่มงานใหม่โดยการพัฒนามาตรฐานการเข้ารหัสแบบใหม่ขึ้นซึ่งจะทำให้ได้อัตราบิท (bit rate) ที่ต่ำมาก โดยให้ชื่อว่า MPEG-4 สำหรับมาตรฐานนี้คาดว่าจะเสร็จภายในปี 1998 มาตรฐาน MPEG ทั้งหมดจะประยุกต์ใช้งานได้อย่างอิสระ โดยมันจะกำหนดในเรื่องวิธีการเข้ารหัสบิทสตรีมและกระบวนการถอดรหัสเท่านั้น ไม่ได้กำหนดวิธีการทำงานของการ เข้ารหัสไว้ด้วย จึงทำให้เป็นวิธีที่ยืดหยุ่นเพียงพอสำหรับผู้ขายแต่ละรายให้สามารถพัฒนาแต่ส่วนของระบบของตนเองให้ดีที่สุด มาตรฐาน MPEG ถูกตีพิมพ์ออกเป็น 4 ส่วน คือ ระบบ (systems) , ภาพเคลื่อนไหว (video) , เสียง (audio) และการทดสอบโครงสร้าง (conformance testing) ในบทนี้เราจะกล่าวถึงเฉพาะในส่วนของภาพเคลื่อนไหวเท่านั้น

4.2 มาตรฐานภาพเคลื่อนไหวใน MPEG-1

4.2.1 พื้นฐานเบื้องต้น

อัลกอริทึมของการเข้ารหัสในมาตรฐานนี้ เป็นการบีบอัดข้อมูลแบบสูญเสียที่สามารถประยุกต์ใช้งานได้กับรูปแบบของข้อมูลเข้าหลายรูปแบบ แต่อย่างไรก็ตาม มันจะเหมาะสำหรับอุปกรณ์ที่มีอัตราบิทประมาณ 1.5 เมกะบิทต่อวินาที มากกว่า (อย่างเช่น CD-ROM) ในมาตรฐาน MPEG-1 จะใช้คำว่า "ภาพ (picture)" แทนคำว่า "เฟรม (frame)" เพราะมันไม่สามารถรับข้อมูลแบบสลับ (interlace) ได้

ในระบบของภาพเคลื่อนไหวที่สามารถสลับได้นั้น แต่ละเฟรมจะถูกแบ่งออกเป็น 2 ฟิลด์ (field) คือ ฟิลด์ที่อยู่ส่วนบน (top field) และฟิลด์ที่อยู่ส่วนล่าง (bottom field) ภาพในหนึ่งเฟรมที่สามารถสลับได้นี้ เส้นสแกนของทั้ง 2 ฟิลด์จะแทรกกันอยู่ดังรูปที่ 4.1 (a) ส่วนในระบบภาพเคลื่อนไหวที่ไม่สามารถสลับได้นั้นจะต่างกันตรงที่มันจะสแกนต่อเนื่องไปจากมุมซ้ายบนไปจนถึงด้านล่างของเฟรมโดยไม่มีการแบ่งเป็นฟิลด์เลย



รูปที่ 4.1 แสดงให้เห็นการสแกนสลับกันและการสแกนแบบเรียงลำดับ

กุญแจสำคัญที่ต้องการตลอดการพัฒนาระบบมาตรฐาน MPEG-1 มีอยู่ 2 ประการคือ

1. การที่มีอัตราการบีบอัดที่สูง (High compression)
2. ความสามารถในการเข้าถึงข้อมูลแบบสุ่ม (Random Access Capability)

การเข้ารหัสภายในเฟรมเพียงอย่างเดียวจะสามารถบรรลุจุดประสงค์ได้เพียงความสามารถในการเข้าถึงข้อมูลแบบสุ่มได้เท่านั้น ดังนั้นถ้าเราต้องการบรรลุจุดประสงค์ทั้ง 2 ประการ เราจะต้องใช้เทคนิคของการเข้ารหัสภายในเฟรมและการเข้ารหัสระหว่างเฟรมร่วมกัน ยิ่งไปกว่านั้น ถ้าเราต้องการปรับปรุงอัตราการบีบอัดข้อมูลให้ดีขึ้น เราจำเป็นต้องใช้การเข้ารหัสทั้งแบบการทำนาย (predictive) และการสอดแทรก (interporative) ด้วย

การทำงานของการบีบอัดข้อมูลใน MPEG จะประกอบด้วย

1. การลดอัตราบิทในส่วนของที่ว่าง (spatial domain) และในส่วนของที่ว่างชั่วคราว (temporal domain) ที่เกิดขึ้นของทั้งองค์ประกอบแสงและองค์ประกอบสี

2. การอธิบายการทำงานของ DCT โดยใช้บล็อกเป็นพื้นฐาน (Block - based DCT) สำหรับการเข้ารหัสระหว่างเฟรมและการเข้ารหัสภายในเฟรม

3. การอธิบายการทำงานของการชดเชยการเคลื่อนที่โดยใช้บล็อกเป็นพื้นฐาน (Block - based motion compensation) สำหรับการทำนายระหว่างเฟรมและการสอดแทรกระหว่างเฟรม

4. การเข้ารหัสฮัฟแมนสำหรับการบีบอัดข้อมูลแบบสูญเสียของเวคเตอร์ของการเคลื่อนที่และสัมประสิทธิ์ DCT ที่ถูกควอนไตซ์ (quantize) แล้ว

รูปแบบของแหล่งกำเนิดของข้อมูลเข้า (the source input format หรือ SIF)

ใน MPEG-1 กำหนดให้ภาพมีขนาดสูงสุดได้ 4095x4095 พิกเซล อย่างไรก็ตามการประยุกต์ใช้งานเป็นจำนวนมากที่ใช้มาตรฐาน MPEG-1 ในการบีบอัดข้อมูลภาพเคลื่อนไหวจะเหมาะสำหรับรูปแบบของภาพเคลื่อนไหวแบบ SIF ซึ่งได้มาจากรูปแบบของ CCIR601 สำหรับภาพเคลื่อนไหวในโทรทัศน์ระบบดิจิตอล จากรูปแบบของCCIR601 แหล่งกำเนิดของภาพเคลื่อนไหวที่เป็นสีจะประกอบด้วย 3 ส่วนคือ 1 องค์ประกอบแสง (Y) และ 2 องค์ประกอบสี (Cb , Cr)

CCIR601 มี 2 ทางเลือกสำหรับค่าความละเอียดที่ต่างกันคือ

1. สำหรับโทรทัศน์ระบบ NTSC จะใช้การสแกน 525 เส้นต่อเฟรม ที่อัตราเร็ว 60 เฟรมต่อวินาที เฟรมขององค์ประกอบแสงมีขนาด 720x480 พิกเซล และแต่ละเฟรมขององค์ประกอบสีมีขนาด 360x480 พิกเซล หรืออาจกล่าวได้ว่าเป็นรูปแบบของการแบ่งข้อมูลเป็นส่วนย่อยๆ (subsampling) ด้วยอัตราส่วน 4:2:2

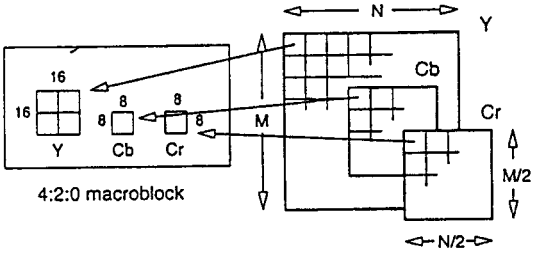
2. สำหรับโทรทัศน์ระบบ PAL จะใช้เส้นสแกน 625 เส้นต่อเฟรม ที่อัตราเร็ว 50 เฟรมต่อวินาที เฟรมขององค์ประกอบแสงมีขนาด 720x576 พิกเซล และแต่ละเฟรมขององค์ประกอบสีมีขนาด 360x576 พิกเซล

เพื่อที่จะทำให้ได้อัตราการบีบอัดที่ต่ำถึง 1.5 เมกะบิทต่อวินาที MPEG-1 จำเป็นต้องกำหนดรูปแบบของแหล่งกำเนิดของข้อมูลเข้า (SIF) ซึ่งลำดับของ SIF มีความละเอียดขององค์

ประกอบแสงที่ 360x240 พิกเซลต่อภาพ และมีอัตราเร็ว 30 ภาพต่อวินาที หรือที่ 360x288 พิกเซลต่อภาพ และมีอัตราเร็ว 25 ภาพต่อวินาที

ทั้ง 2 กรณีนี้ความละเอียดขององค์ประกอบสีจะเป็นครึ่งหนึ่งของความละเอียดขององค์ประกอบแสง ทั้งในแนวนอนและแนวตั้ง หรืออาจกล่าวได้ว่าเป็นรูปแบบของการแบ่งข้อมูลเป็นส่วนย่อยๆ ที่มีอัตราส่วน 4:2:0 ภาพของ SIF สามารถได้มาง่ายๆ จากเฟรมของ CCIR601 โดยใช้การกรองและการแบ่งข้อมูลเป็นส่วนย่อยๆ

ใน MPEG-1 องค์ประกอบ YCbCr จะแทรก (interleave) กันอยู่เสมอ โดยมาโครบล็อกจะถูกกำหนดให้เป็นเสมือนหน่วยของรหัสที่เล็กที่สุด และประกอบด้วย 4 มาโครบล็อกที่มีขนาด 8x8 สำหรับองค์ประกอบแสง (Y) และ 1 มาโครบล็อกขนาด 8x8 สำหรับแต่ละองค์ประกอบสี (Cb , Cr) ดังแสดงในรูปที่ 4.2 โดยขนาดสูงสุดของมาโครบล็อกคือ 16x16



รูปที่ 4.2 ข้อกำหนดของมาโครบล็อกในมาตรฐาน MPEG-1

ภาพแต่ละภาพจะถูกแบ่งออกเป็นอนุกรมของมาโครบล็อกจากซ้ายไปขวา และจากบนลงล่าง ซึ่งภาพแต่ละภาพจะต้องมีความละเอียดทั้งในแนวนอนและแนวตั้งเป็นจำนวนเท่าของ 16 ถ้าหากว่าไม่เป็นไปตามข้อตกลงนี้ ตัวเข้ารหัสจะทำให้ค่าพิกเซลเป็น 0 แล้วนำค่าไปใส่ไว้ทางขวามือหรือด้านล่างของแต่ละภาพ ซึ่งค่าเหล่านี้จะถูกตัดทิ้งโดยตัวถอดรหัสในภายหลัง

MPEG-1 จะให้ความยืดหยุ่นในการนำไปใช้งานได้เป็นอย่างดี แต่อย่างไรก็ตามก็ไม่สามารถคาดหวังได้ว่า ตัวถอดรหัสทั้งหมดของ MPEG จะสามารถสนับสนุนการเข้ารหัสทุกรูปแบบได้ ดังนั้น MPEG-1 จึงกำหนด ข้อจำกัดของพารามิเตอร์ของบิทสตรีม (constrained parameters bit stream) ขึ้น ซึ่งก็คือ บิทสตรีมที่ตัวถอดรหัสทั้งหมดของ MPEG จะสามารถสนับสนุนได้ จากตารางที่ 4.2 แสดงให้เห็นถึงพารามิเตอร์ดังกล่าว แต่อย่างไรก็ตามพารามิเตอร์เหล่านี้ก็ยังไม่เป็นที่ต้องการในมาตรฐาน

จากข้อจำกัดในตารางที่ 4.2 จะพบว่าอัตราพิกเซล (pixel rate) 396x25 มาโครบล็อกต่อวินาทีจะเป็นตัวจำกัดขนาดของภาพให้มีขนาดประมาณ 352x288 พิกเซล ซึ่งเป็นพื้นที่ของภาพนิ่ง SIF ที่ 25 เฮิรตซ์

Picture Rate (Hz)	30	25
CCIR 601		
Y	720 × 480	720 × 576
Cb, Cr	360 × 480	360 × 576
SIF		
Y	360 × 240	360 × 288
Cb, Cr	180 × 120	180 × 144
Significant Pixel Area for SIF		
Y	352 × 240	352 × 288
Cb, Cr	176 × 120	176 × 144

ตารางที่ 4.1 แสดงขนาดของภาพสำหรับระบบ CCIR601,SIF และพิกเซลในพื้นที่ที่ใช้งาน

Coding Parameter	Maximum Value
Horizontal picture size	768 pixels
Vertical picture size	576 lines
Macroblocks	396
Pixel rate	396 × 25 macroblocks/s
Picture rate	30 pictures/s
Range of motion vectors	+/- 64 pixels (half-pixel resolution)
Size of input buffer	327,680 bits
Bit rate	1,856 kbits/s

ตารางที่ 4.2 ข้อจำกัดของพารามิเตอร์ของบิทสตรีม

5.2.2 ชนิดของภาพ (Picture Types)

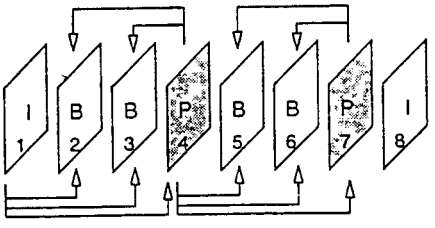
การทำงานพื้นฐานบางอย่างบนภาพเคลื่อนไหว (video stream) ซึ่งรวมไปถึงการแก้ไข (editing) , การเข้าถึงข้อมูลแบบสุ่ม (random access) และการค้นหา ข้อมูล โดยการทำงานเหล่านี้มักจะขัดแย้งกับการเพิ่มประสิทธิภาพของการบีบอัดข้อมูล MPEG-1 จึงแบ่งภาพออกเป็น 4 ชนิด เพื่อให้มีความยืดหยุ่นกันระหว่างประสิทธิภาพของการเข้ารหัสและการเข้าถึงข้อมูลแบบสุ่ม

1. ภาพที่เกิดจากเงื่อนไขภายใน หรือภาพ I (Intra- picture หรือ I-pictures) เป็นภาพที่ถูกบีบอัดโดยใช้การเข้ารหัสแบบภายในเฟรม ซึ่งไม่ต้องอ้างอิงกับภาพอื่นในบิทสตรีม มันจะให้การเข้าถึงข้อมูลแบบสุ่มที่รวดเร็ว (fast random access) แต่จะให้อัตราการบีบอัดในระดับปานกลางเท่านั้น ซึ่งจะคล้ายคลึงกับการประยุกต์ใช้งานของการบีบอัดข้อมูลแบบสูญเสียของ JPEG บนภาพเดี่ยว

2. ภาพที่ได้มาจากการทำนาย หรือภาพ P (Predicted pictures หรือ P-pictures) เป็นภาพที่ถูกเข้ารหัสโดยวิธีการทำนายสำหรับการชดเชยการเคลื่อนที่ จากภาพ P หรือภาพ I ภาพก่อนๆ ซึ่งภาพ P จะมีอัตราการบีบอัดที่ดีกว่าภาพ I และมันยังสามารถใช้เป็นจุดอ้างอิงในกระบวนการการชดเชยการเคลื่อนที่ได้อีกด้วย

3. ภาพที่ได้มาจากการทำนายแบบสองทิศทาง หรือภาพ B (Bidirectionally predicted pictures หรือ B-pictures) เป็นภาพที่ให้อัตราการบีบอัดดีที่สุด และเป็นภาพที่ถูกเข้ารหัสโดยวิธีการทำนายสำหรับการชดเชยการเคลื่อนที่ จากทั้ง ภาพ I (ที่เพิ่งผ่านไป และ/หรือ ที่กำลังจะเกิดขึ้น) หรือภาพ P เพราะภาพ B ไม่สามารถใช้ในการทำนายภาพB หรือภาพ P ใดๆ ได้ จึงทำให้มันเป็นภาพที่บิดเบือนได้ง่ายและมีอัตราการบีบอัดที่สูงกว่าภาพ I หรือภาพ P

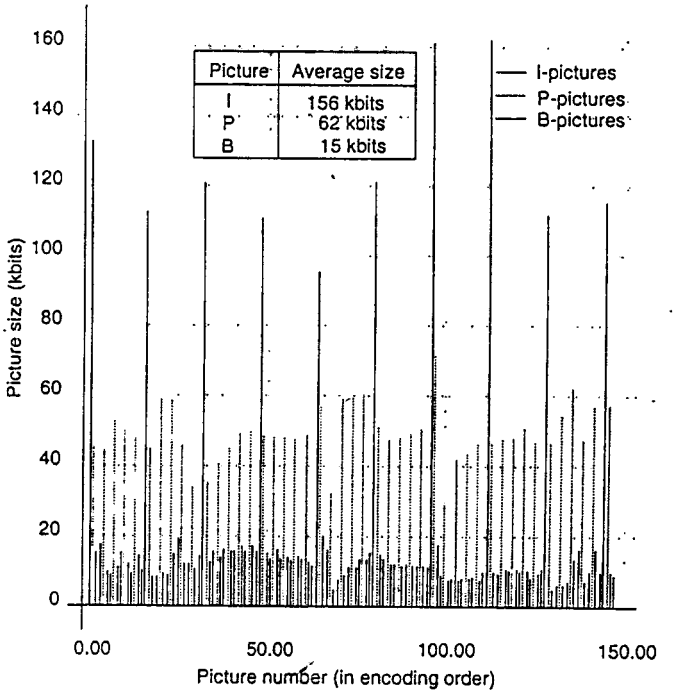
4. ภาพที่ได้จากการเข้ารหัส DC หรือภาพ D (DC - coded pictures หรือ D-pictures) มีลักษณะคล้ายกับภาพ I แต่ต่างกันตรงที่ภาพ D จะแสดงเฉพาะในส่วนของสัมประสิทธิ์ DC เท่านั้น ภาพ D ไม่สามารถนำมารวมกับภาพชนิดอื่นๆ ได้ อีกทั้งยังไม่สามารถใช้ใน MPEG-2 ได้อีกด้วย แต่จะสามารถใช้ได้โดยตรงกับการค้นหาแบบรวดเร็ว (fast searches)



รูปที่ 4.3 ตัวอย่างของความเกี่ยวเนื่องกันระหว่างภาพ I , ภาพ P และภาพ B ในลำดับของภาพเคลื่อนไหว

จากรูปที่ 4.3 แสดงถึงความสัมพันธ์ระหว่างภาพหลัก 3 ชนิดในอนุกรมภาพเคลื่อนไหว โดยภาพ p_1 และ p_8 เป็นภาพ I , p_4 และ p_7 เป็นภาพ P และที่เหลือเป็นภาพ B ในตัวอย่างนี้ p_5 จะถูกเข้ารหัสโดยวิธีการทำนายสำหรับการชดเชยการเคลื่อนที่จาก p_1 และ p_4 บิทสตรีมใน MPEG ไม่จำเป็นต้องใช้ทั้งภาพ B และภาพ P ก็ได้ ส่วนขนาดของภาพ I , P และ B นั้นโดยปกติในอนุกรมของภาพเคลื่อนไหวจะถูกเข้ารหัสที่ความละเอียดของ SIF ซึ่งก็คือ 1.15 เมกะบิทต่อวินาทีนั่นเอง ดังแสดงในรูปที่ 4.4

ภาพ B ต้องการจำนวนบิทที่น้อยกว่าภาพ I และภาพ P การเพิ่มจำนวนของภาพ B ระหว่างภาพ I กับภาพ P นั้นอาจจะไม่ทำให้อัตราการบีบอัดข้อมูลสูงขึ้น เนื่องจากว่ามันจะทำให้เกิดความแตกต่างกันระหว่างภาพ I หรือ ภาพ P ที่ติดกันได้



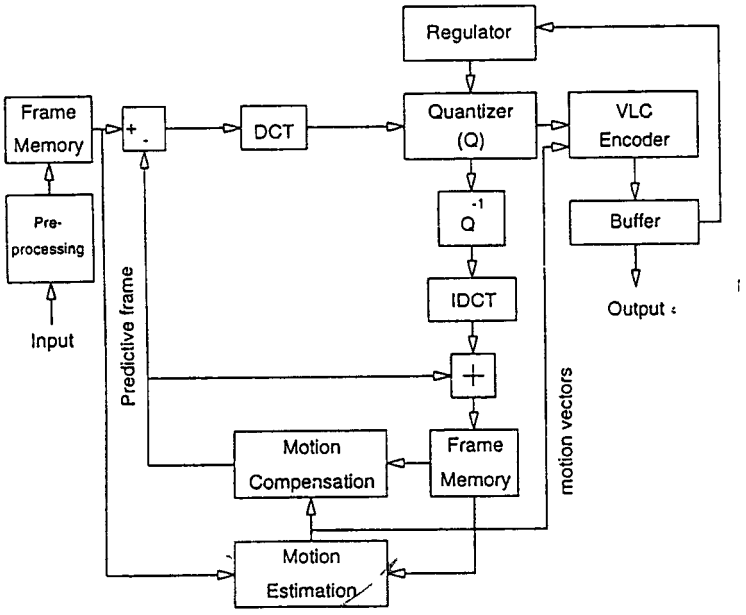
รูปที่ 4.4 ตัวอย่างของการกระจายบิทระหว่างภาพ I,P และ B ในการเข้ารหัสบิทสตรีมของ MPEG-1 โดยมีลำดับของภาพที่เข้ารหัสเป็นดังนี้ IPBBPBBPBBPBBPBB

4.2.3 ตัวเข้ารหัสภาพเคลื่อนไหว (Video Encoder)

เนื่องจากในมาตรฐาน MPEG ไม่ได้กำหนดกระบวนการการเข้ารหัสไว้ แต่ได้กำหนดวิธีการเข้ารหัสบิทสตรีม และกระบวนการการถอดรหัสไว้เท่านั้น จากรูปที่ 4.5 แสดงการทำงานที่จำเป็นต่อการเข้ารหัสในมาตรฐาน MPEG

- กระบวนการก่อนกระบวนการจริง (Preprocessing)

การเข้ารหัสมักเริ่มด้วยกระบวนการเบื้องต้นบางอย่าง ซึ่งอาจจะหมายถึงการเปลี่ยนสีเป็นแบบ YCbCr , การเปลี่ยนภาพแบบที่สามารถสลับได้เป็นแบบเรียงลำดับ , การกรองเบื้องต้น (prefiltering) และการแบ่งข้อมูลออกเป็นส่วนย่อยๆ ซึ่งกระบวนการเหล่านี้ไม่ได้เป็นส่วนสำคัญเท่าไรนักในมาตรฐาน MPEG



รูปที่ 4.5 บล็อกไดอะแกร็มแสดงการทำงานของตัวเข้ารหัสในมาตรฐาน MPEG

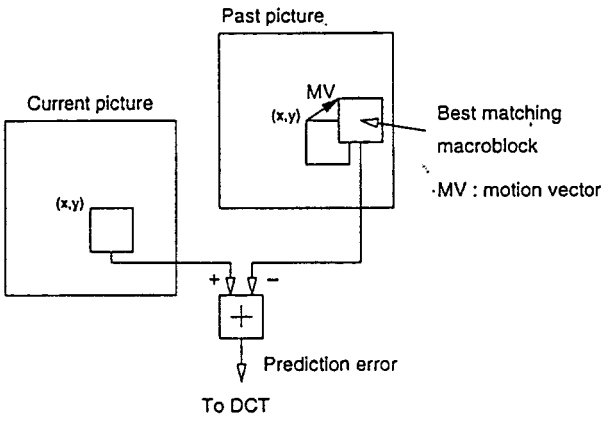
- การประมาณการเคลื่อนที่และการชดเชยการเคลื่อนที่

หลังจากการทำงานของกระบวนการก่อนกระบวนการจริงแล้ว ตัวเข้ารหัสจะเลือกชนิดของภาพที่เป็นอินพุท ซึ่งถ้าเป็นภาพ I เราก็ไม่จำเป็นต้องทำกระบวนการเหล่านี้เลย

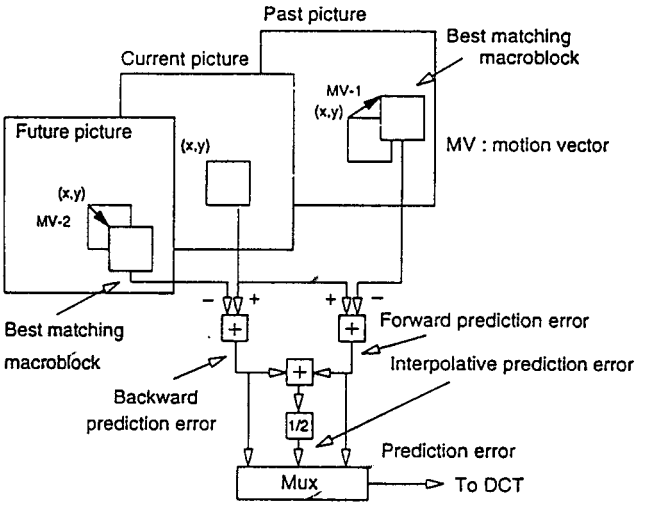
จากรูปที่ 4.5 แต่ละมาโครบล็อกจะเข้ารหัสแบบ DCT จากนั้นสัมประสิทธิ์ DCT จะถูกควอนไตซ์ แล้วผ่านเข้าสู่กระบวนการของการเข้ารหัสแบบ VLC (variable - length coder) ต่อจากนั้นจะนำไปเก็บไว้ในบัฟเฟอร์ทางด้านเอาท์พุท ภายในแต่ละมาโครบล็อกกระบวนการต่างๆ จะต้องทำงานกับข้อมูลขนาด 8x8 ในส่วนของการประยุกต์ใช้งาน เราต้องการอัตราบิทที่คงที่ ตัวปรับค่าความต่างศักย์ที่เอาท์พุทของบัฟเฟอร์ (Buffer regulator) จะเป็นตัวปรับการควอนไตซ์เมตริกซ์เพื่อที่จะทำให้อัตราการบีบอัดของบิทสตรีมคงที่ สำหรับงานที่ต้องการเปลี่ยนอัตราบิท ตัวปรับค่าความต่างศักย์ที่เอาท์พุทก็ไม่ต้องทำงาน สำหรับกระบวนการควอนไตซ์จะเหมือนกับการควอนไตซ์ผกผัน (Inverse quantized หรือ Q^{-1}) จากนั้นเราจะแปลงข้อมูลที่ได้มาจากกระบวนการการควอนไตซ์ผกผันให้อยู่ในส่วนของที่ว่าง โดยใช้ IDCT การทำงานนี้จะจำลองพฤติกรรมของตัวถอดรหัสและให้ภาพที่เหมือนกับภาพที่ผ่านการเข้ารหัสมาแล้ว ซึ่งจะถูกเห็นโดยตัวถอดรหัส ภาพดังกล่าวจะถูกเก็บไว้ใน หน่วยความจำชั่วคราวและจะถูกใช้ในการทำนายรหัสในอนาคต ผลจากการทำงานในส่วนนี้ทำให้ตัวเข้ารหัสเป็นตัวบอกคุณภาพของการส่งผ่านข้อมูลภาพนิ่ง เพื่อที่จะได้ภาพที่ไม่แตกต่างไปจากสัญญาณเริ่มต้นเท่าไรนัก เนื่องจากว่า VLC เป็นการทำงานแบบไม่สูญเสีย ดังนั้นจึงไม่ต้องมีการป้อนสัญญาณกลับ (feedback) ในส่วนของ VLC

ถ้าข้อมูลเข้าถูกเข้ารหัสเป็นภาพ P หรือภาพ B ตัวเข้ารหัสจะไม่เข้ารหัสภาพเป็นมาโครบล็อกโดยตรง แต่มันจะเข้ารหัสแบบทำนายความผิดพลาดแทน ในMPEG เราเรียกการทำงานแบบนี้ว่า การเข้ารหัสแบบการทำนายระหว่างเฟรม (Interframe predictive coding) สำหรับภาพ P กระบวนการนี้จะอธิบายได้ด้วยรูปที่ 4.6

สำหรับภาพ B กระบวนการการประมาณค่าการเคลื่อนที่จะถูกกระทำ 2 ครั้ง ครั้งแรกสำหรับภาพในอดีตและครั้งที่ 2 สำหรับภาพในอนาคต เพื่อให้ได้เวคเตอร์ของการเคลื่อนที่ 2 ตัว ตัวเข้ารหัสสามารถกำหนดรูปแบบของมาโครบล็อกของการทำนายความผิดพลาดได้ทั้งจากมาโครบล็อกที่มีอยู่แล้ว และจากค่าเฉลี่ยของมัน ซึ่งกระบวนการนี้แสดงรายละเอียดได้ดังรูปที่ 4.7 ใน MPEG จะเรียกการทำงานแบบนี้ว่า การเข้ารหัสแบบสอดแทรกระหว่างเฟรม (Interframe interpolative coding) การทำนายความผิดพลาดจะถูกเข้ารหัสโดยใช้การอธิบายการทำงานของ DCT โดยใช้บล็อกเป็นพื้นฐาน สัมประสิทธิ์ DCT ที่ถูกควอนไตซ์แล้วของการทำนายความผิดพลาดดังกล่าว รวมทั้งเวคเตอร์ของการเคลื่อนที่ จะถูกรวมสัญญาณ (multiplex) และเข้ารหัสโดยกระบวนการของ VLC ใน MPEG จะแตกต่างกับ JPEG ตรงที่ MPEG จะใช้ตัวเข้ารหัสแบบฮัฟแมนสำหรับ VLC เท่านั้น



รูปที่ 4.6 กระบวนการที่ดำเนินไปข้างหน้าของการชดเชยการเคลื่อนที่



รูปที่ 4.7 กระบวนการการชดเชยการเคลื่อนที่แบบ 2 ทิศทาง

เนื่องจากเป็นการทำนายแบบ 2 ทิศทาง การจัดเรียงภาพอินพุทใหม่จึงจำเป็นต้องกระทำที่ตัวเข้ารหัส เพื่อที่จะส่งภาพให้กับตัวถอดรหัสเป็นลำดับได้อย่างถูกต้อง ตัวอย่างเช่น ในรูปที่ 4.3

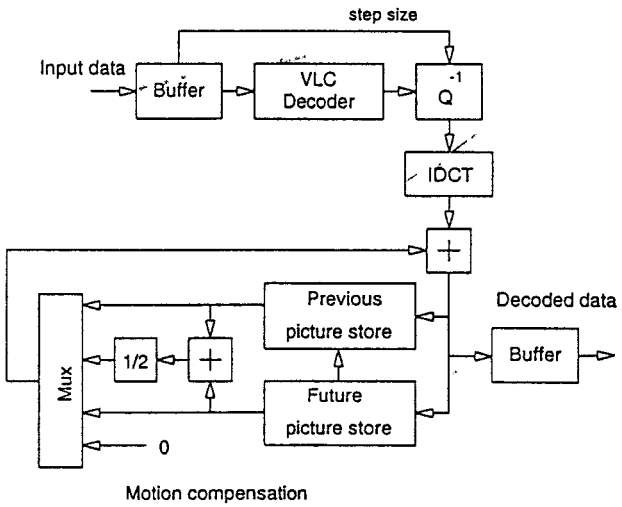
เนื่องจาด p_2 เป็นภาพ B ที่ขึ้นอยู่กับ p_1 และ p_4 ซึ่งมันจะสามารถเข้ารหัสได้หลังจากที่ p_4 เข้ามาแล้วเท่านั้น หลังจากที่มีการจัดเรียงภาพอินพุทแล้ว ลำดับของการเข้ารหัสที่ถูกต้องก็คือ p_1 , p_4 , p_2 , p_3 , p_7 , p_5 , p_6 และ p_8 ซึ่งลำดับดังกล่าวจะให้ข้อมูลที่จำเป็นต่อการถอดรหัสทั้งหมดแก่ตัวถอดรหัส อย่างไรก็ตาม ตัวถอดรหัสจำเป็นต้องมีการจัดเรียงข้อมูลใหม่เช่นกัน เพื่อที่จะให้ได้ภาพที่เป็นลำดับ

4.2.4 ตัวถอดรหัสภาพเคลื่อนไหว (Video Decoder)

จากรูปที่ 4.8 แสดงบล็อกไดอะแกร็มของการทำงานของตัวถอดรหัสภาพเคลื่อนไหวในมาตรฐาน MPEG ซึ่งวงจรของตัวถอดรหัสภาพเคลื่อนไหวจะมีลักษณะคล้ายคลึงกับวงจรในส่วนของการป้อนกลับของตัวเข้ารหัสสัญญาณภาพเคลื่อนไหวนั่นเอง

ขั้นแรกภาพที่เข้ามาจะถูกถอดรหัสแบบฮัฟแมนและชนิดของภาพจะถูกกำหนดจากข้อมูลของส่วนหัวของสัญญาณ (header) สำหรับแต่ละมาโครบล็อกสัมประสิทธิ์จะถูกควอนไตซ์ แล้วแปลงกลับไปสู่ส่วนของที่ว่างโดย IDCT และลำดับของการเข้ารหัสคือ p_1 , p_4 , p_2 , p_3 , p_7 , p_5 , p_6 และ p_8

เพื่อให้บรรลุจุดประสงค์ดังกล่าว เราจะสมมติให้มาโครบล็อกทั้งหมดในภาพ P ถูกเข้ารหัสเป็นมาโครบล็อก P (P-macroblocks) และมาโครบล็อกทั้งหมดในภาพ B จะถูกเข้ารหัสเป็น มาโครบล็อก B (B-macroblocks) MPEG-1 จะให้ความยืดหยุ่นในการเข้ารหัสมาโครบล็อกที่ดีกว่า



รูปที่ 4.8 บล็อกไดอะแกร็มแสดงการทำงานของตัวถอดรหัสภาพเคลื่อนไหวในมาตรฐาน MPEG

กระบวนการการถอดรหัสมีขั้นตอนดังต่อไปนี้

1. ภาพ p_1 (ภาพ I) ไม่มีการทำกระบวนการการชดเชยการเคลื่อนที่ เพราะฉะนั้นอินพุทที่เข้ามาที่ตัวรวมสัญญาณในรูปที่ 4.8 จะถูกกำหนดเป็น 0 เมื่อทำการคำนวณ IDCT แล้ว เราจะเก็บเอาท์พุทไว้ในบัฟเฟอร์ที่ใช้แสดงผลและในบล็อกที่ใช้เก็บภาพในอดีต (Previous picture store)

2. ภาพ p_4 (ภาพ P) แต่ละมาโครบล็อกจะถูกคำนวณ IDCT และเข้ากระบวนการการชดเชยการเคลื่อนที่ โดยจะบวกพื้นที่ที่ถูกชี้โดยเวคเตอร์ของการเคลื่อนที่ในบล็อกที่ใช้เก็บภาพในอดีต กับเอาท์พุทของ IDCT ภาพที่สร้างขึ้นมาใหม่นี้จะถูกเก็บไว้ในบล็อกที่ใช้เก็บภาพในอนาคต (future picture store)

3. ภาพ p_2 (ภาพ B) คำนวณ IDCT เสร็จแล้วจะดำเนินกระบวนการการชดเชยการเคลื่อนที่แบบ 2 ทิศทาง ซึ่งใช้เวคเตอร์ของการเคลื่อนที่ 2 ตัว เพื่อเข้าถึงค่าของพิกเซลในบริเวณที่ตรงกับพื้นที่ที่ใช้เก็บภาพในอดีตและภาพในอนาคต และคำนวณหาการทำนายสำหรับ p_2 จากนั้นเราจะรวม เอาท์พุทของ IDCT กับผลของการทำนายที่ได้

เนื่องจาก p_2 เป็นภาพที่ต่อจาก p_1ในลำดับภาพ p_2จึงสามารถแสดงได้เลยในขณะนั้น สังเกตได้ว่าภาพ B ไม่จำเป็นต้องเก็บในพื้นที่ที่ใช้เก็บภาพ เพราะมันไม่ต้องนำมาใช้ในการทำนายลำดับของภาพต่อๆไป

4. ภาพ p_3 (ภาพ B) ทำแบบเดียวกับกระบวนการที่ถอดรหัสภาพ p_2 ออกมา หลังจากที่แสดงภาพ p_3 แล้ว ภาพ p_4 ก็จะถูกแสดงผลต่อไป

5. ภาพ p_7 (ภาพ P) ทำแบบเดียวกับกระบวนการที่ถอดรหัสภาพ p_4 แต่อย่างไรก็ตามภาพที่ถูกสร้างขึ้นมาใหม่จำเป็นที่จะต้องเก็บรักษาไว้ในที่ที่เก็บภาพในอดีต ซึ่ง p_1 ก็ถูกเก็บไว้ก่อนแล้ว

6. ภาพ p_5 , p_6 (ภาพ B) ทำซ้ำเหมือนกับขั้นตอนที่ 3 การทำนายสำหรับ ภาพ p_5 , p_6 จะใช้ p_4 , p_7 หลังจากที่ p_5 และ p_6 ถูกถอดรหัสแล้ว เราจะแสดงภาพ p_4 ก่อน แล้วจึงตามด้วยภาพ p_5 , p_6 และ p_7 ตามลำดับ

จากขั้นตอนทั้งหมดนี้ถือว่าเป็นการถอดรหัสภาพหนึ่งกลุ่มได้เสร็จสิ้นสมบูรณ์ สำหรับภาพในกลุ่มต่อไป จะเริ่มต้นจาก p_8 แล้วดำเนินการถอดรหัสเหมือนเดิม จากผังการทำงานจะมีภาพมากที่สุด 3 ภาพที่ถูกเก็บไว้พร้อมกันในชั่วขณะหนึ่ง มาตรฐาน MPEG-1 จึงต้องการหน่วยความจำอย่างน้อยครึ่งเมกะไบต์ (Mbyte) ในการแสดงความละเอียดแบบ SIF ซึ่งลำดับของการแสดงภาพจะถูกกำหนดเวลาไว้ในระบบของเลเยอร์ (layer)

4.2.5 โครงสร้างในการเข้ารหัสของบิทสตรีมที่เป็นภาพเคลื่อนไหว (Structure of the Coded Video Bit stream)

MPEG-1 กำหนดให้บิทสตรีมของภาพเคลื่อนไหวมีการเข้ารหัสแบบลำดับชั้น ซึ่งมีการทำงานทั้งหมด 6 ชั้น คือ (1) ชั้นของการลำดับภาพ (a sequence layer) , (2) ชั้นของกลุ่มของรูปภาพ (a group of pictures layer หรือ GOP) , (3) ชั้นของรูปภาพ (a picture layer) , (4) ชั้นของสัญญาณไลด์ (a slice layer) , (5) ชั้นของมาโครบล็อก (a macroblock layer) , (6) ชั้นของบล็อก (a block layer) ดังแสดงได้ดังรูปที่ 4.9

(1) ชั้นของการลำดับภาพ เป็นชั้นสูงสุดของการเข้ารหัส ซึ่งจะประกอบด้วยลำดับของส่วนหัวของสัญญาณ แล้วตามด้วยกลุ่มของรูปภาพหนึ่งกลุ่มหรือมากกว่านั้น โดยมันจะจบลงด้วย รหัสของการสิ้นสุดลำดับภาพ ข้อมูลที่ถูกบรรจุอยู่ในลำดับของส่วนหัวของสัญญาณนั้น จะเป็นขนาดทั้งในแนวนอนและแนวตั้งของรูปภาพแต่ละภาพ , มาตราส่วนของพิกเซล (pixel aspect ratio) , อัตราเร็วภาพต่อวินาที , อัตราบิทในหน่วย 400 บิทต่อวินาที และขนาดที่เล็กที่สุดของบัฟเฟอร์ที่ตัวถอดรหัสต้องการ (ในหน่วย 2048 ไบต์) บิทเดี่ยวๆ ก็สามารถทำตามกระบวนการนี้ได้ถึงแม้ว่าตัวเข้ารหัสจะไม่พบพารามิเตอร์ที่ต้องการทั้งหมดในบิทสตรีมก็ตาม

(2) ชั้นของกลุ่มของรูปภาพ คือภาพจำนวนหนึ่งที่แสดงผลต่อเนื่องกันเป็นลำดับซึ่งจะต้องมีภาพ I อย่างน้อย 1 ภาพเป็นส่วนประกอบ ชั้นนี้สามารถเริ่มต้นด้วยภาพ I หรือ ภาพ P ก็ได้และจะจบด้วยภาพ I หรือ ภาพ P ก็ได้เช่นกัน ถ้าภาพแรกเป็นภาพ I หรือ ภาพ B ที่ไม่ขึ้นอยู่กับภาพของกลุ่มของรูปภาพก่อนหน้านี้ ดังนั้นกลุ่มของรูปภาพนี้จะสามารถเข้ารหัสและแสดงผลได้อย่างเป็นอิสระจากกลุ่มอื่น ซึ่งเราจะเรียกว่า กลุ่มของรูปภาพที่เป็นวงปิด (closed GOP) โดยที่ส่วนหัวของสัญญาณของกลุ่มของรูปภาพนี้จะประกอบไปด้วย กำหนดเวลาของข้อมูล (timing information) , ตัวบ่งบอกสถานะ (flag) ของกลุ่มของรูปภาพที่เป็นวงปิด , ส่วนขยาย และข้อมูลของผู้ใช้ จากรูปที่ 4.3 แสดงให้เห็นกลุ่มของรูปภาพที่เป็นวงปิด ที่กลุ่มของรูปภาพมีขนาดเท่ากับ 7

(3) ชั้นของรูปภาพ จะเป็นตัวกำหนดข้อมูลของการเข้ารหัสสำหรับแต่ละภาพ เพื่อให้สามารถเรียงลำดับภาพ P และภาพ B ได้อย่างถูกต้อง โดยที่ส่วนหัวของสัญญาณของชั้นนี้จะให้ตัวเลขอ้างอิงสำหรับอธิบายลำดับของภาพที่จะแสดง นอกจากนี้ส่วนหัวของสัญญาณยังจะให้ข้อมูลเกี่ยวกับชนิดของภาพ , การเข้าจังหวะกันได้ (synchronization) , ความละเอียดของภาพ และขนาดของเวคเตอร์ของการเคลื่อนที่

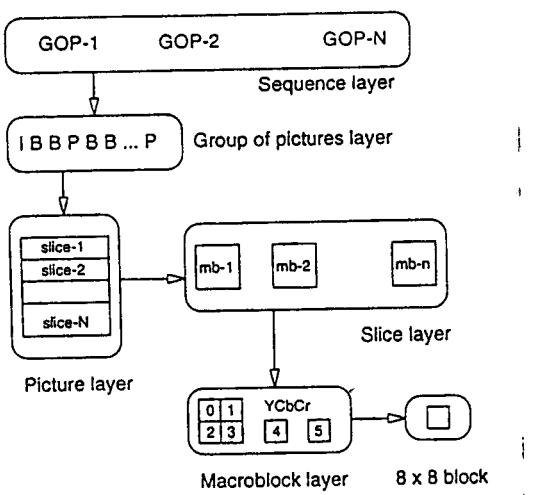
(4) ชั้นของสัญญาณไลด์ โดยขนาดของชั้นนี้สามารถที่จะใหญ่ถึงภาพทั้งภาพหรือเล็กแค่หนึ่งมาโครบล็อกก็ได้ ในกรณีที่เกิดความเสียหายของข้อมูล ตัวถอดรหัสจะสามารถทำการสร้างข้อมูลขึ้นใหม่ โดยใช้ข้อมูลจากส่วนหัวของสัญญาณ ตัวอย่างเช่น การส่งผ่านข้อมูลเกิดการผิดพลาด

ตัวถอดรหัสจะลดขนาดของสัญญาณไลด์ลงให้มีขนาดไม่เต็มภาพ ในส่วนหัวของสัญญาณนั้นจะบรรจุข้อมูลตำแหน่งของมันในภาพ และตัวที่กำหนดการวัดขนาดของตัวควอนไตซ์ (quantizer scale factor) ที่มีค่าระหว่าง 1 ถึง 31 เพื่อที่จะใช้ตัวถอดรหัสในการลดการควอนไตซ์ (dequantize) สัมประสิทธิ์ DCT ลงได้ ความพิเศษของตัวที่กำหนดการวัดขนาดของตัวควอนไตซ์ คือจะทำให้ตัวเข้ารหัสสามารถแสดงอัตราการปรับค่าความต่างศักย์ที่เอาท์พุทของชั้นนี้ได้

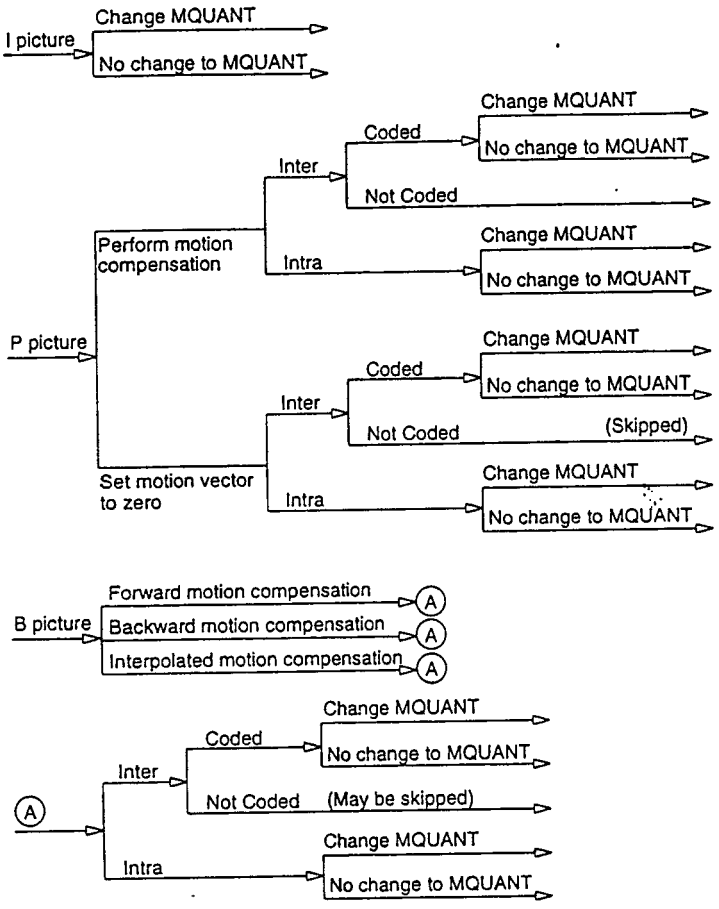
(5) ชั้นของมาโครบล็อกและ (6) ชั้นของบล็อก ในสไลด์ข้อมูลจะถูกแบ่งออกเป็นมาโครบล็อกซึ่งที่ส่วนหัวของสัญญาณของมาโครบล็อกจะบอกชนิดของมาโครบล็อก , ตำแหน่งของข้อมูล , รหัสของเวคเตอร์ของการเคลื่อนที่ทั้งในแนวนอนและแนวตั้ง รวมไปถึงสามารถบอกได้ว่าบล็อกไหนภายในหนึ่งมาโครบล็อกที่มีการเข้ารหัสและส่งผ่านข้อมูลอย่างแท้จริง

4.2.6 การเข้ารหัสของข้อมูลที่เป็นมาโครบล็อก (Macroblock Coding)

จากที่ได้กล่าวมาแล้ว ใน MPEG จะมีภาพหลักอยู่ 3 ชนิด คือภาพ I , ภาพ P และภาพ B แต่อย่างไรก็ตามสำหรับภาพแต่ละภาพนั้น มาโครบล็อกจะถูกเข้ารหัสแตกต่างกันไป ซึ่งสามารถแสดงเป็นแผนภูมิสาขาได้ดังรูปที่ 4.10



รูปที่ 4.9 สรุปความสัมพันธ์ของชั้นต่างๆ ในการเข้ารหัสภาพเคลื่อนไหวในมาตรฐาน MPEG-1



รูปที่ 4.10 แผนภูมิสาขาแสดงการเข้ารหัสข้อมูลที่เป็นมาโครบล็อกของภาพ I , P และB

บทที่5

รูปแบบรหัสข้อมูลวิดีโอตามมาตรฐาน MPEG-1

5.1 บทนำ

จากการทำงานของระบบประมวลผลภาพเคลื่อนไหวตามมาตรฐาน MPEG ในบทที่ 4 นั้น เป็นเพียงโครงสร้างการทำงานของระบบเท่านั้น แต่ข้อมูลจริงๆนั้นถูกกำหนดไว้ใน ISO/IEC 11172-2 ชุดของ Information technology – Coding of moving picture and associated audio for digital storage media at up to about 1.5 Mbit/s ซึ่งจะแบ่งออกเป็น 4 ส่วนคือ Systems, Video, Audio และ Compliance testing ซึ่งจะทำให้ผู้ออกแบบระบบการประมวลผลภาพเคลื่อนไหว สามารถพัฒนาบนมาตรฐานเดียวกัน

5.2 รหัสเริ่มต้น (Start code)

รหัสเริ่มต้น คือ รหัสที่ใช้เป็นรหัสที่บอกการเริ่มของกระบวนการหรือชั้นใดชั้นหนึ่งซึ่งใน มาตรฐาน MPEGได้กำหนดไว้ดังในตารางที่ 5.1

ชื่อ	เลขฐาน 16
Picture_start_code	00000100
Slice_start_codes	00000101-000001AF
Reserved	000001B0, 000001B1
User_data_start_code	000001B2
Sequence_header_code	000001B3
Sequence_error_code	000001B4
Extenstion_start_code	000001B5
Reserved	000001B6
Sequence_end_code	000001B7
Group_start_code	000001B8
System_start_codes	000001B9-000001FF

ตารางที่ 5.1 รหัสเริ่มต้นที่กำหนดไว้ใน ISO/IEC 11172-1

5.3 ชั้นของข้อมูล

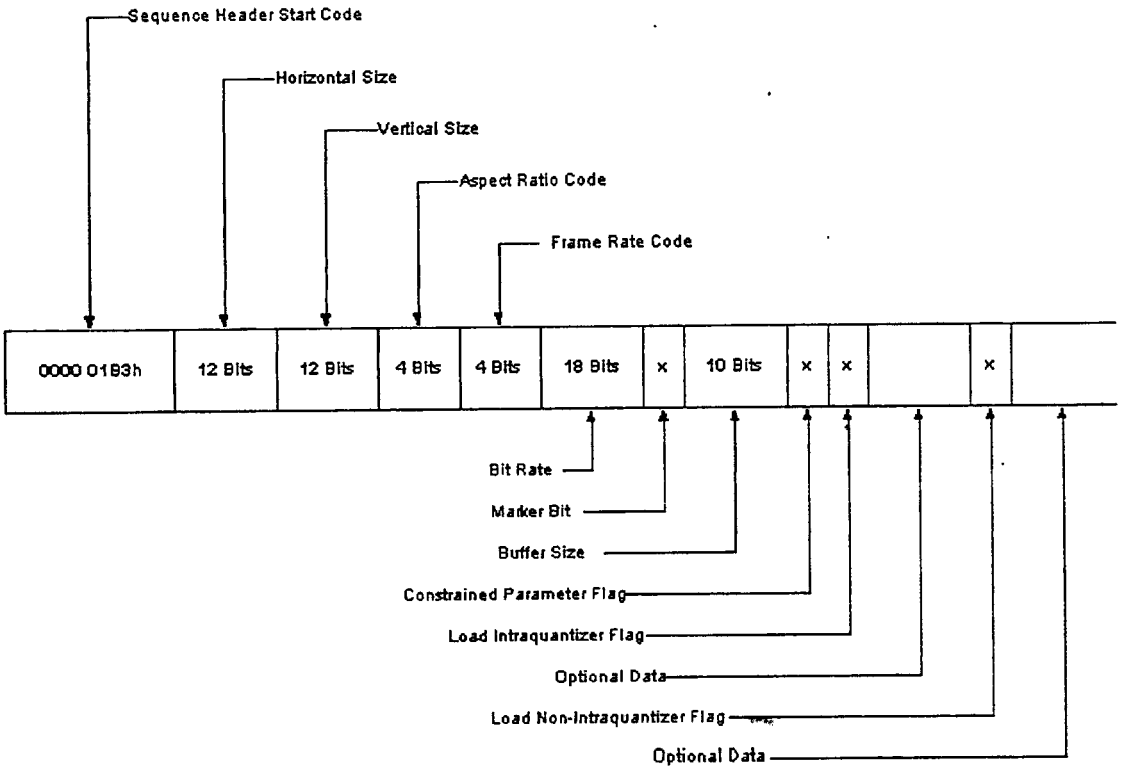
เนื่องจากข้อมูลที่ใช้ในแฟ้มข้อมูล MPEG มีจำนวนมาก ดังนั้นทางมาตราฐาน MPEG ได้ทำการแบ่งข้อมูลออกเป็นชั้นต่างๆ ตามหน้าที่การทำงานเพื่อให้สามารถตรวจสอบได้ง่าย มีทั้งหมด 6 ชั้นคือ

5.3.1 ชั้น Video

ตามมาตราฐาน ISO11172-2 ข้อมูลที่เราพบอันแรกจะเป็น ข้อมูลของชั้น Video ซึ่งมีรูปแบบการทำงานดังนี้

Syntax	จำนวนบิท
Video_sequence() {	
Next_start_code()	
do{	
Sequence_header()	
Do{	
Group_of_pictures()	
} while (nextbits() == group_start_code)	
} while (nextbits() == sequence_header_code)	
Sequence_end_code	32
}	

ในชั้น Video นี้จะมี กลุ่มข้อมูลที่สำคัญคือ sequence header ซึ่งเป็นการเริ่มต้นของข้อมูลของวีดีโอ มีรูปแบบของข้อมูลดังรูปที่ 5.1



รูปที่ 5.1 รูปแบบของ MPEG Sequence header

จากรูปที่ 5.1 เราสามารถเขียนโปรแกรมการทำงานได้ดังนี้

Syntax	จำนวนบิท
Sequence_header() {	
Sequence_header_code	32
Horizontal_size	12
Vertical_size	12
Pel_aspect_ratio	4
picture_rate	4
bit_rate	18
marker_bit	1
vbv_buffer_size	10

```
        Constrained_parameter_flag                                  1
        load_intra_quantizer_matrix                                 1
        if (load_intra_quantizer_matrix)
                Intra_quantizer_matrix[ ]                           8*64
        load_non_intra_quantizer_matrix                             1
        if (load_non_intra_quantizer_matrix)
                Non_intra_quantizer_matrix [ ]                      8*64
        if (nextbits() == extenstion_start_code) {
                Extenstion_start_code                               32
                While (nextbits()!= "0000 0000 0000 0000 0000 0001""){
                        sequence_extenstion_data                    8
                }
                Next_start_code()
        }
        if (nextbits() == extenstion_start_code){
                User_data_start_code                                32
                While (nextbits()!= "0000 0000 0000 0000 0000 0001'){
                        user_data                                   8
                 }
                 Next_start_code()
        }
}
```

โดยชื่อแต่ละอย่างมีความหมายดังนี้

sequence_header_code เป็นค่าคงที่ค่าหนึ่งมีค่าดังตารางที่ 5.1 เป็นตัวบอกว่าเริ่มต้นของ sequence header

horizontal_size คือ ความกว้างของส่วนที่แสดงผลได้ของ Y ในหน่วยพิกเซล

vertical_size คือ ความสูงของส่วนที่แสดงผลได้ของ Y หน่วยพิกเซล

pel_aspect_ratio คือ ค่าที่แสดงอัตราส่วนของความสูงต่อความกว้างอ่านค่าได้ตามตารางที่ 5.2

Pel_aspect_ratio	สูง/กว้าง	ตัวอย่าง
0000	ห้าม	
0001	1.0000	VGA etc.
0010	0.6735	
0011	0.7031	16:9, 525 line
0100	0.7615	
0101	0.8055	CCIR601, 625 line
0110	0.8437	
0111	0.8935	
1000	0.9157	
1001	0.9815	
1010	1.0255	
1011	1.0697	
1100	1.0950	
1101	1.1575	CCIR601, 525 line
1110	1.2015	
1111	Reserved	

ตารางที่ 5.2 รหัสข้อมูลของ pel_aspect_ratio

picture_rate คืออัตราของภาพต่อนาที

Picture rate	ภาพต่อวินาที
0000	ห้าม
0001	23.976
0010	24
0011	25

0100	29.97
0101	30
0110	50
0111	59.94
1000	60
.....	Reserved
1111	Reserved

ตารางที่ 5.3 รหัสข้อมูลของ picture_rate

bit_rate คือ ค่าจำนวนเต็มซึ่งกำหนดอัตราความเร็วของบิตในหน่วยของ 400 bits/s (ห้ามเป็น 0)

marker_bit คือ บิตจำนวนหนึ่งบิตถูกเซตให้เป็น "1"

vbv_buffer_size คือ ค่าจำนวนเต็มขนาด 10 บิต ใช้กำหนดขนาดของ Video Buffering Verifier โดยค่าขนาดของ VBV ต่ำสุดที่ยังสามารถใช้ถอดรหัสได้คือ 16*1024*vbv_buffer_size

constrained_parameters_flag เป็นแฟลกที่ถ้าถูกเซตเป็น "1" ค่าต่างๆจะถูกบังคับให้เป็น

horizontal_size <= 768 pels

vertical_size <= 576 pels

(horizontal_size+15)/16)*((vertical_size+15)/16)< = 396

((horizontal_size+15/16)*((vertical_size+15)/16)*picture_rate <= 396

picture_rate <= 30 pictures/s

forward_f_code <= 4

backward_f_code <= 4

load_intra_quantizer_matrix ถ้าเป็น 1 ค่าที่ตามมาจะเป็น intra_quantizer_matrix ถ้าถูกเซตเป็น 0 จะใช้ค่า quantizer_matrix แบบ default ตามรูปเมตริกข้างล่าง และจะใช้ไปจนกว่าจะถึง sequence ถัดไป

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

intra_quantizer_matrix คือ ข้อมูลจำนวน 64 กลุ่มของบิตจำนวน 8 บิต แบบจำนวนเต็มไม่มีเครื่องหมาย โดยจะสามารถใส่ลงในเมตริกในแบบ zigzag scanning order (ห้ามเป็น 0) ค่าของ 8 บิตแรกต้องเป็น 8

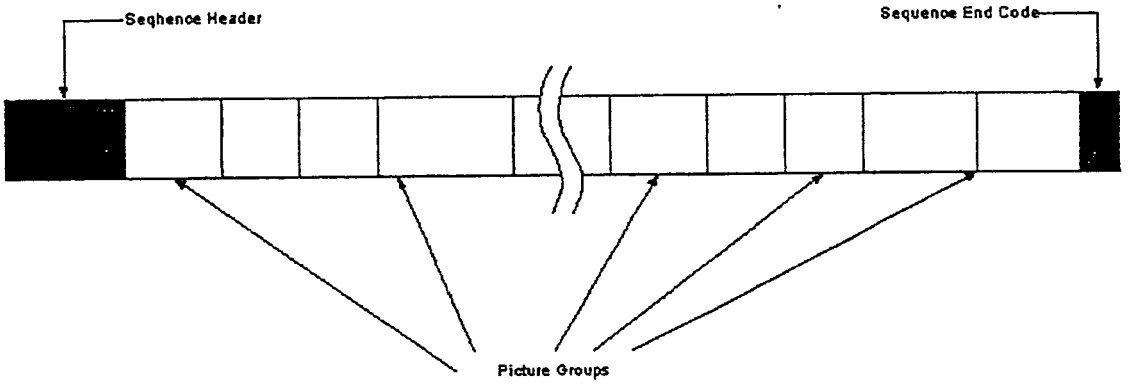
load_non_intra_quantizer_matrix ถ้าเป็น 1 ค่าที่ตามมาจะเป็น non_intra_quantizer_matrix ถ้าถูกเซตเป็น 0 จะใช้ค่า quantizer_matrix แบบ default ตามรูปเมตริกข้างล่าง และจะใช้ไปจนกว่าจะถึง sequence ถัดไป

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

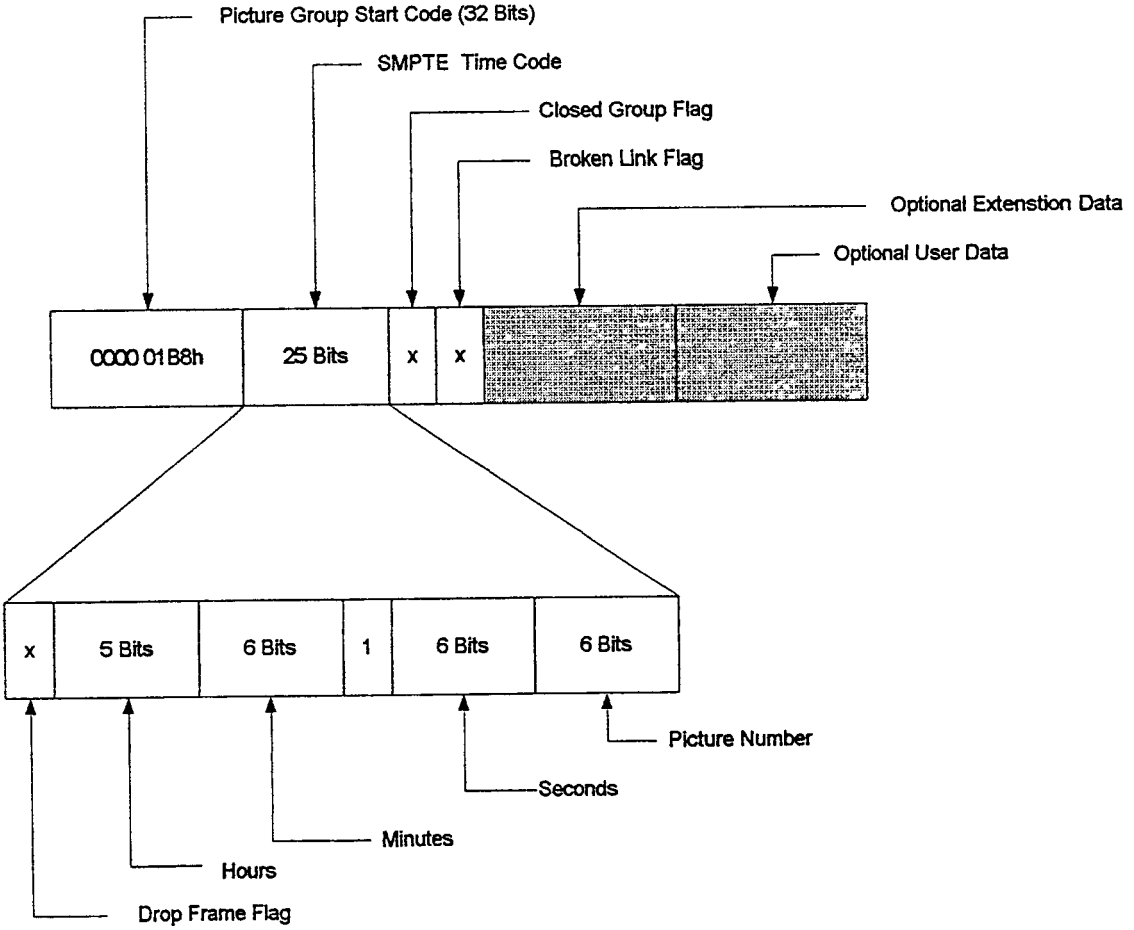
non_intra_quantizer_matrix เป็นข้อมูลจำนวน 64 กลุ่มของบิตจำนวน 8 บิต แบบจำนวนเต็มไม่มีเครื่องหมาย โดยจะสามารถใส่ลงในเมตริกในแบบ zigzag scanning order (ห้ามมีค่าใดเป็น 0)

5.3.2 ชั้น Group of pictures

ชั้นของกลุ่มของภาพ(Group of picture) จะมีอยู่คิดกับชั้น Sequence header ซึ่งอาจจะมีกลุ่มของภาพกลุ่มเดียวหรือมากกว่านั้นก็ได้ ดังรูปที่ 5.2



รูปที่ 5.2 รูปแบบลำดับข้อมูลของ MPEG



รูปที่ 5.3 รูปแบบของหัวของกลุ่มของภาพ(Header group of picture)

ในแต่ละกลุ่มของภาพ จะต้องมีหัว(header) เป็นตัวขึ้นต้นซึ่งมีลักษณะดังรูปที่ 5.3 และจากรูปที่ 5.3 เราสามารถเขียนโปรแกรมการทำงานได้ดังนี้

Syntax	จำนวนบิท
Group_of_pictures(){	
Group_start_code()	32
Time_code	25
Closed_gop	1
Broken_link	1
Next_start_code ()	
if(nextbits()==extension_stat_code) {	
Extension_start_code	32
While(nextbits()!='0000 0000 0000 0000 0000 0001') {	
group_extension_data	8
}	
Next_start_code ()	
}	
if(nextbits()==user_data_start_code) {	
User_data_start_code	32
While(nextbit() !='0000 0000 0000 0000 0000 0001') {	
User_data	8
}	
Next_start_code()	
}	
do {	
Picture()	
}while (nextbits()==picture_stxrt_code)	
}	

Group_of_picture เป็นค่าที่บอกการเริ่มต้นของชั้นนี้ มีค่าดังตารางที่ 5.1

time_code คือ ข้อมูลจำนวน 25 บิต ซึ่งจะถูกตามด้วย drop_frame_flag, time_code_hours, time_code_minutes, marker_bit, time_code_seconds และ time_code_pictures ซึ่งค่าเหล่านี้เป็นค่าที่สอดคล้องกับมาตรฐาน IEC ในเรื่อง "time and control codes for video tape recorders" ค่า drop_frame_flag จะเป็น "1" เมื่อ picture rate = 29.97 Hz เท่านั้น การนับจำนวนภาพจะตัดภาพที่ 0 และ 1 ของตอนเริ่มนาที ยกเว้นนาทีที่ 0, 10 ,20, 30, 40, 50 แต่ถ้า drop_frame_flag ถูกเซ็ตเป็น 0 มันจะปัดให้ picture rate เท่ากับจำนวนเต็มค่าที่ใกล้ที่สุด

Time_code	range of picture	Bits	
Drop_frame_flag		1	
Time_code_hours	0-23	5	uimsbf
Time_code_minutes	0-59	6	uimsbf
Marker_bit	1	1	"1"
Time_code_seconds	0-59	6	uimsbf
Time_code_pictures	0-59	6	uimsbf

ตารางที่ 5.4 รหัสข้อมูลหลัง time_code

close_group ถ้าถูกเซตเป็น "1" แสดงว่า GOP นี้ถูกเข้ารหัสโดยไม่ได้ใช้ การชี้ของ motion vector จาก GOP ก่อนหน้านั้น

broken_link ควรจะถูกเซตเป็น "0" เสมอ แต่ถ้าเซตเป็นหนึ่งจะแสดงว่าไม่สามารถถอดรหัส B-pictures ได้เนื่องมาจากภาพที่จะใช้ทำนายชนิด I หรือ P ไม่มีอยู่

5.3.3 ชั้น Picture

ชั้นของpicture จะอยู่ถัดจาก หัวของกลุ่มของภาพ(Header group of picture) ซึ่งจะมีอยู่ด้วยกันหลายภาพดังรูปที่ 5.4

Picture_start_code เป็นค่าที่บอกการเริ่มต้นของชั้นpicture

temporal_reference เป็นค่าที่ใช้บอกจำนวนของภาพ โดยจะเพิ่มขึ้นหนึ่งครั้งต่อหนึ่งภาพในแบบ modulo 1024 ภาพแรกของ GOP ให้ค่าเป็น 0

picture_coding_type เป็นค่าที่บอกชนิดของภาพดังตารางที่ 5.5

picture_coding_type	ชนิดของภาพ
000	Forbidden
001	intra-coded (I)
010	predictive-coded (P)
011	Bidirectionally-predictive-coded (B)
100	dc intra-coded (D)
101	Reserved
....	....
11	Reserved

ตารางที่ 5.5 รหัสข้อมูลของ picture_coding_type

vbv_delay เป็นค่าจำนวนเต็มแบบไม่มีเครื่องหมายขนาด 16 บิต สำหรับการทำงานแบบบิตเรตคงที่ มันจะถูกใช้สำหรับตั้งค่าเริ่มต้น บัฟเฟอร์ของตัวถอดรหัส เมื่อเริ่มการถอดรหัสภาพเพื่อให้บัฟเฟอร์ไม่เกิดการ Overflow หรือ Underflow โดยจะทำการคำนวณเวลาที่จะต้องเติม VBV buffer หลังจากที่ว่างในช่วงเริ่มต้นตั้งให้บิตเรตเท่ากับ R เพื่อให้สามารถแก้ไขได้ในทันทีก่อนที่ภาพขณะนั้นถูกลบไปจากบัฟเฟอร์

ค่าของ vbv_delay จะเป็นค่าของช่วงเวลาของ clock 90 kHz เพื่อรอหลังจากรับไบต์สุดท้ายของ picture_start_code โดยสามารถคำนวณได้จาก

$vbv\_delay_n = 90000 * B_n / R$

$n > 0$

B_n = VBV occupancy ในหน่วย bits

R = บิตเรตในหน่วย bits/s

สำหรับการทำงานแบบบิตเรตไม่คงที่ vbv_delay จะถูกตั้งให้เป็นค่า $FFFF_H$

full_pel_forward_picture ถ้าเป็น 1 แสดงว่าค่าของ forward motion vector เป็นจำนวนเต็ม

forward_f_code เป็นตัวที่นำไปใช้ในการหา forward motion vector เป็นจำนวนเต็มค่า 1 ถึง 7

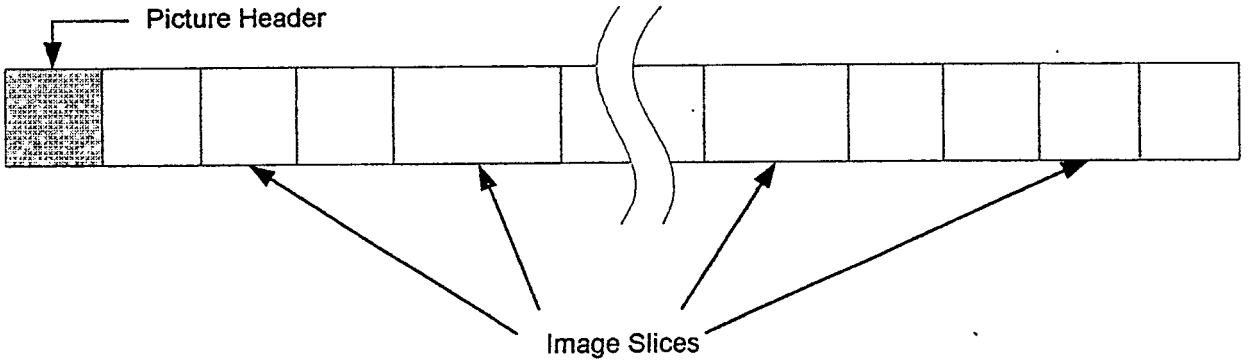
full_pel_backward_picture ถ้าเป็น 1 แสดงว่าค่าของ backward motion vector เป็นจำนวนเต็ม

backward_f_code เป็นตัวที่นำไปใช้ในการหา backward motion vector เป็นจำนวนเต็มค่า 1 ถึง 7

extra_bit_picture ถ้าเป็น 1 แสดงว่าจะมีข้อมูล extra information ตามมา

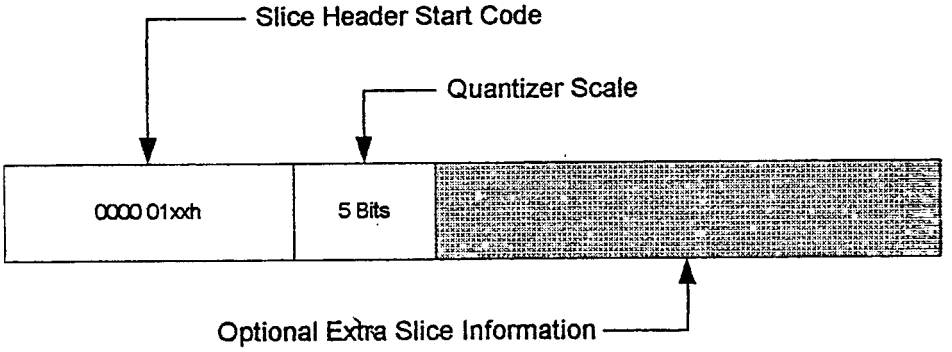
5.3.4 ชั้น Slice

ชั้นของ Slice จะอยู่ติดกับ หัวของภาพ(picture header) ซึ่งจะมีอยู่ด้วยกันหลาย slice ดังรูปที่ 5.6



รูปที่ 5.6 รูปแบบของชั้น picture

แต่ละชั้นของslice จะประกอบด้วยหัวของสไลด์(Slice header) ซึ่งมีลักษณะดังรูปที่ 5.7



รูปที่ 5.7 รูปแบบของหัวสไลด์(Slice header)

จากรูปที่ 5.7 เราสามารถเขียนโปรแกรมการทำงานได้ดังนี้

Syntax	No. of bits
slice(){	
Slice_start_ code()	32
Quantizer_scale	5
While (nextbits() == '1'){	
extra_bit_slice	1
extra_information_slice	8
}	
Extra_bit_slice	1
Do{	
macroblock()	
} while (nextbits() != '0000 0000 0000 0000 0000 0000'	
Next_start_code()	
}	

slice_start_code เป็นข้อมูล 32 บิต แสดงการเริ่มต้นของชั้น slice โดยมี 24 บิตแรกเป็น 000001H และ 8 บิตหลังเป็นตำแหน่งของ slice ค่าตั้งแต่ 01H-AFH

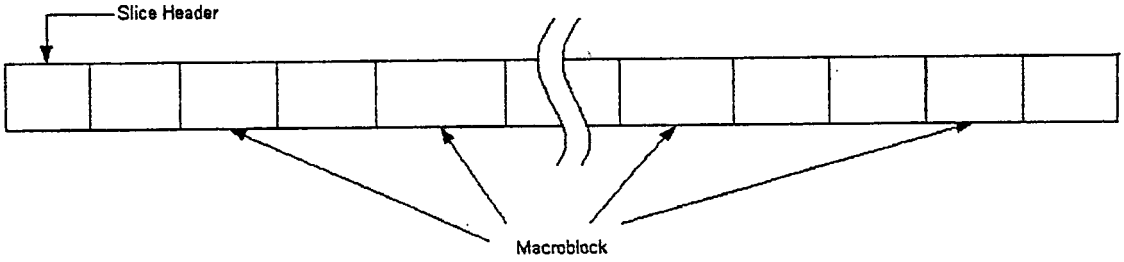
slice_vertical_position เป็นค่า 8 บิตสุดท้ายของ slice_start_code โดยที่ slice แรกมีค่าเป็น 1 อาจมีหลาย slice ที่มีค่าเดียวกัน ซึ่งแสดงว่าขนาดของ slice ไม่จำเป็นต้องเท่ากัน อาจจะเริ่มที่ตำแหน่ง vertical หนึ่ง และไปจบที่อีก vertical หนึ่ง แต่ที่แน่นอนคือ macroblock แรกของภาพจะอยู่ใน slice แรก และ macroblock สุดท้ายของภาพจะอยู่ใน slice สุดท้าย

quantizer_scale เป็นค่าจำนวนเต็ม ตั้งแต่ 1-31 สำหรับคูณกับค่า DCT coefficient ที่ได้จาก stream ข้อมูลค่านี้จะถูกใช้จนกว่าจะรับมาได้ใหม่

extra_bit_slice ถ้าเป็น 1 จะมีข้อมูล extra_information_slice ตามมา

5.3.5 ชั้น Macroblock

ชั้นของมาโครบล็อก จะอยู่ถัดจาก หัวสไลด์ ซึ่งมีอยู่ด้วยกันหลายมาโครบล็อกด้วยกันมีลักษณะดังรูปที่ 5.8



รูปที่ 5.8 รูปแบบของชั้นสไลด์

ในชั้นของมาโครบล็อกนั้นจะไม่มีส่วนของหัวอยู่จะมีแต่ข้อมูลบางอย่างดังที่ได้แสดงในโปรแกรมข้างล่าง

Syntax	จำนวนบิท
Macroblock(){	
while (nextbits() == '0000 0001 111')	
Macroblock_stuffing	11
while (nextbits() == '0000 0001 000')	
Macroblick_escape	11
Macroblock_address_increment	1-11
Macroblock_type	1-6
If (macroblock_quant)	
Quantizer_scale	5
If (macroblock_motion_forward){	
Motion_horizontal_forward_code	1-11
if ((forward_f !=1) &&	
(motion_horizontal_forward_code != 0)	
)	
Motion_horizontal_forward_r	1-6

```
        Motion_vertical_forward_code                                    1-11
        if ( ( forward_f != 1) &&
                        (motion_vertical_forward_code != 0) )
            Motion_vertical_forward_r                                   1-6
    }
    If (macroblock_motion_backward){
        Motion_horizontal_backward_code                                 1-11
        if ( ( backward_f != 1) &&
                        (motion_horizontal_backward_code != 0))
            Motion_horizontal_backward_r                                1-6
        Motion_vertical_back_code                                       1-11
        if ( ( backward_f != 1 ) &&
                        (motion_vertical_backward_code != 0) )
            Motion_vertical_forward_r                                   1-6
    }
    If (macroblock_pattern)
        code_block_pattern                                              3-9
    For ( I=0 ; I<6 ; I++)
        block( I )
    If (picture_coding_type == 4)
        end_of_macroblock                                               1
}
```

macroblock_stuffing เป็นค่าคงที่ "0000 0001 111" ใส่โดย encoder เพื่อเพิ่มบิตเรต

macroblock_escape เป็นค่าคงที่ "0000 0001 000" เพื่อเป็นการบอกว่าเป็นกรณี macroblock address increment มีค่าเกิน 33 โดยจะนำ 33 มาคูณด้วยจำนวนของ macroblock_escape แล้วบวกเข้ากับค่าที่ให้มา

macroblock_address_increment เป็นค่า vlc บอกค่าความแตกต่างระหว่างตำแหน่งของมันกับแมคโครบล็อกก่อนหน้า

macroblock_type เป็นค่าแสดงชนิดของ macroblock

macroblock_scale เป็นค่าเดียวกับที่อยู่ในชั้น slice จะเปลี่ยนเมื่อได้รับค่าใหม่เข้ามา

motion_horizontal_forward_code เป็นค่า vlc ที่บอกว่ามี motion_horizontal_forward_r อยู่ใน stream หรือไม่

motion_horizontal_forward_r เป็นจำนวนเต็มไม่มีเครื่องหมายใช้สำหรับการหา forward motion vector

motion_vertical_forward_code เป็นค่า vlc ที่บอกว่ามี motion_vertical_forward_r อยู่ใน stream หรือไม่

motion_vertical_forward_r เป็นค่าจำนวนเต็มไม่มีเครื่องหมายใช้สำหรับการหา forward motion vector

motion_horizontal_backward_code เป็นค่า vlc ที่บอกว่ามี motion_horizontal_backward_r อยู่ใน stream หรือไม่

motion_horizontal_backward_r เป็นค่าจำนวนเต็มไม่มีเครื่องหมายใช้สำหรับการหา backward motion vector

motion_vertical_backward_code เป็นค่า vlc ที่บอกว่ามี motion_vertical_backward_r อยู่ใน stream หรือไม่

motion_vertical_backward_r เป็นค่าจำนวนเต็มไม่มีเครื่องหมายใช้สำหรับการหา backward motion vector

code_block_pattern เป็นค่า vlc บอกว่ามีการรับ block ตำแหน่งใดเข้ามาหรือว่าเป็น skipped bloock

5.3.6 ชั้น Block

จากที่ได้ทราบมาแล้วว่า หนึ่งมาโครบล็อกจะประกอบไปด้วย 6 บล็อกโดย 4บล็อกแรกเป็นของ ความสว่าง(luminance:Y) และ บล็อกถัดมาเป็นของ Cr ถัดมาเป็นของ Cb เราสามารถเขียนการทำงานได้ดังนี้

Syntax	จำนวนบิท
Block(i){	
if (pattern_code[i]){	
if (macroblock_intra){	
if (i < 4){	
Dct_dc_size_luminance	2-7

```
            if ( dc_size luminance != 0 )
                dct_dc_differential                          1-8
            }
            Else {
                dct_dc_size_chrominance                      2-8
            }
        }
        else {
            Dct_coeff_first                                  2-28
        }
        if ( picture_coding_type != 4 ){
            While ( nextbits() != '10' )
                dct_coeff_next                               3-28
            End_of block                                     2
        }
    }
}
```

dct_dc_size_luminance เป็นจำนวนบิตของค่า dct_dc_differential ของ component Y ที่ตามมา

dct_dc_size_chrominance เป็นจำนวนบิตของค่า dct_dc_differential ของ component cb และ cr ที่จะตามมา

dct_dc_differential เป็นค่า vlc จะไม่มีใน stream ข้อมูลถ้า dct_dc_size ทั้งสองเป็น 0

dct_coeff_first เป็นค่า vlc ที่เป็น coefficient ตัวแรกของ block

dct_dc_next เป็นค่า vlc ของ coefficient ตัวถัดไปของ block

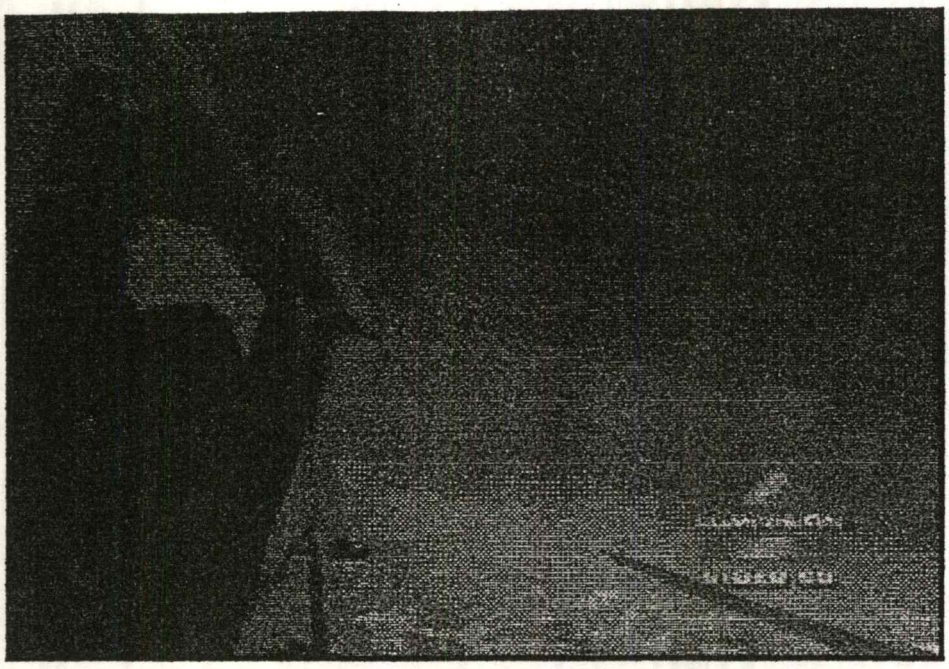
บทที่ 6
ผลการทดลองและสรุป

6.1 การรันโปรแกรม

ในความเป็นจริงแล้วเราสามารถเขียนโปรแกรมฟังก์ชั่นที่เกี่ยวข้องกับกราฟิกส์ ที่สามารถคอมไพล์ในวินโดว์95 หรือวินโดว์32บิทได้ แต่ในที่นี้ผู้จัดทำยังไม่มีความรู้ทางด้านนี้มากพอ เนื่องจากต้องใช้เวลาในการศึกษามากพอสมควร

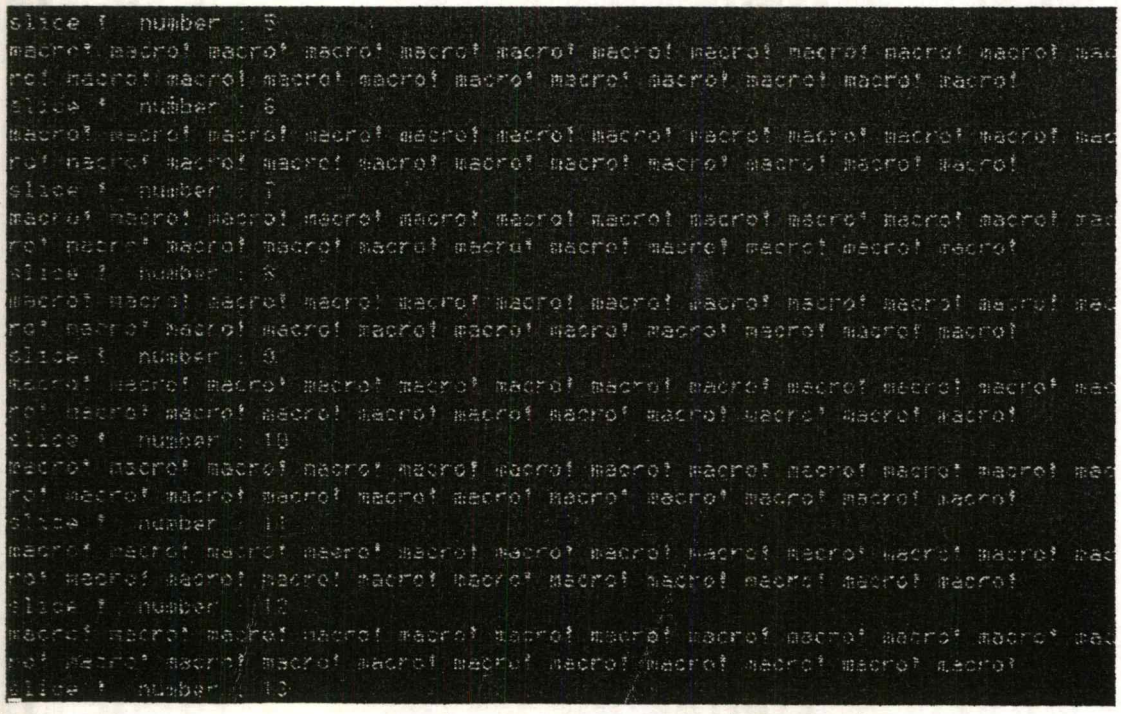
6.1.1 โปรแกรมถอดรหัสข้อมูล MPEG (DECODE.EXE)

โปรแกรมนี้เป็นโปรแกรมที่ทำการอ่านไฟล์ .MPG (video stream ISO11172-2 เท่านั้น) แล้วทำการถอดรหัส จนเป็นภาพในแต่ละเฟรม เรียงต่อกัน แล้วแสดงภาพตามลำดับที่ถูกต้องออกมา (ไม่ใช่ตามลำดับของข้อมูลใน สตรีม ไฟล์) โดยในส่วนของโปรแกรมได้ใช้ การคำนวณสมการหาค่า IDCT โดยวิธีการ Matrix IDCT 1 มิติ 2 ครั้ง (การวนรอบการทำงาน 1,096 ครั้ง ต่อ บล็อค 8x8 1 บล็อค) แทนการใช้สมการ IDCT 2 มิติ โดยตรง (ซึ่งต้องใช้การวนรอบการทำงานถึง 4,096 ครั้ง ต่อ บล็อค 8x8 1 บล็อค) ทำให้การทำงานในส่วนนี้เร็วขึ้น แต่ถึงอย่างไรก็ตาม ความเร็วในส่วนนี้ก็ยังคงไม่พอกับความต้องการที่จะทำให้การถอดรหัสภาพเป็นไปอย่างต่อเนื่อง ซึ่งส่วนนี้ต้องปรับปรุงกันต่อไป ซึ่งโดยทั่วๆ ไป มักนิยมนำวิธีการทาง DSP มาใช้ในกระบวนการนี้ ซึ่งจะทำให้ความเร็วเพิ่มขึ้นอีก หลายๆ เท่าตัวมาก ซึ่งทางผู้จัดทำโครงงานเองยังไม่สามารถทำได้ แต่ถึงอย่างไรก็ตาม ทางผู้จัดทำโครงงานได้ทำทางเลือกอื่นไว้เพื่อแก้ไขในส่วนของความต่อเนื่องของภาพ โดยเมื่อทำการรันโปรแกรม ถ้าพิมพ์ –s เพิ่มเข้าไป โปรแกรมจะทำการเก็บข้อมูลของภาพที่ถอดรหัสแล้วมาเก็บไว้ในไฟล์ OUTPUT.RAW (เป็นการเก็บข้อมูลภาพแบบ 16บิต) แล้วใช้โปรแกรม PLAY.EXE เพื่อทำการแสดงภาพ ที่เก็บไว้ นอกจากนั้นเรายังอาจใช้ ทางเลือกอื่นๆ ได้อีก เช่น –t เพื่อทำการแสดงการถอดรหัสในแบบ เท็กซ์โหมด แทน กราฟิกส์โหมด หรือ –m เพื่อแสดงการถอดรหัสในแบบ เท็กซ์โหมด แบบย่อ , -r เป็นการสร้าง ไฟล์ ที่รายงานการจับเวลาในการถอดรหัสข้อมูลภาพในแต่ละเฟรม (REPORT.RPT) เมื่อเราทำการรันโปรแกรมแล้วจะมีลักษณะดังรูปที่ 6.1 จากรูปที่ 6.1 โปรแกรมจะทำการอ่านข้อมูลและถอดรหัสแล้วจึงแสดงภาพในลำดับที่ถูกต้องออกมา



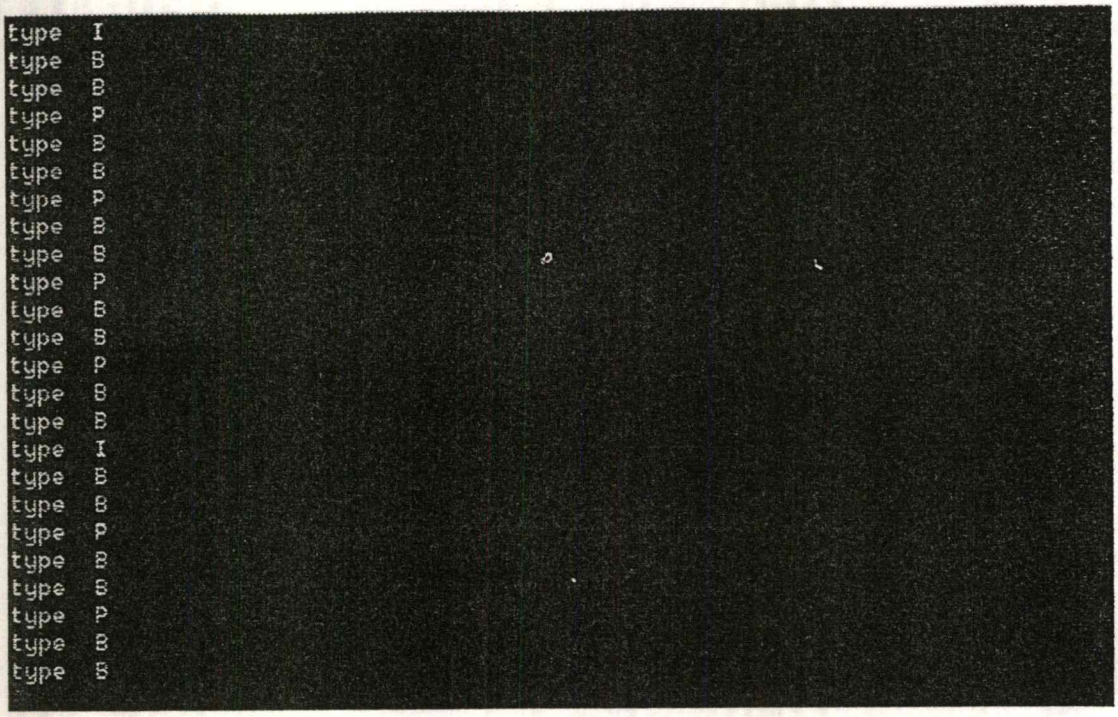
รูป 6.1 ลักษณะหน้าจอของโปรแกรมโดยไม่ใช้พารามิเตอร์ –t หรือ –m(ภาพจากเฟรมที่ 1)

ในรูปที่ 6.2 เป็นการรันโปรแกรมโดยใช้ พารามิเตอร์ –t ซึ่งโปรแกรมจะทำการอ่านข้อมูลและถอดรหัส แล้วบอกว่าขณะนั้นกำลังอ่านภาพชนิดใด และ กำลังถอดรหัสที่ส่วนใดของภาพ (sequence, group_of_picture, picture, slice, macroblock)



รูป 6.2 ลักษณะหน้าจอของโปรแกรมโดยใช้พารามิเตอร์ -t

ในรูปที่ 6.3 เป็นการรันโปรแกรมโดยใช้ พารามิเตอร์ –m ซึ่งโปรแกรมจะทำการอ่านข้อมูลและถอดรหัส แล้วบอกว่าขณะนั้นกำลังอ่านภาพชนิดใดเท่านั้น



รูป 6.3 ลักษณะหน้าจอของโปรแกรมโดยใช้พารามิเตอร์ -m

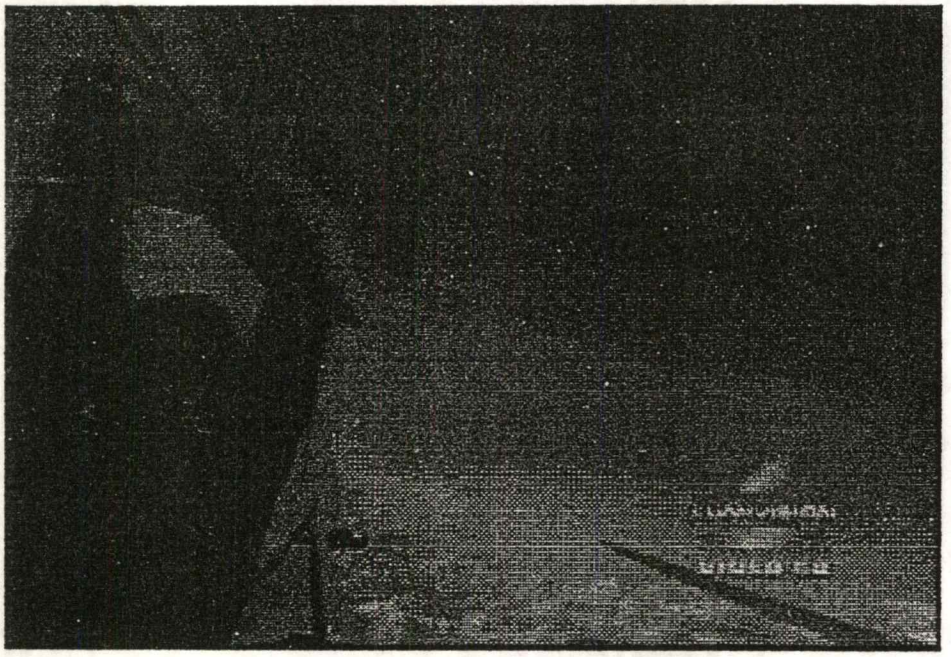
6.1.2 โปรแกรมแสดงภาพ (PLAY.EXE) ไฟล์ .RAW

โปรแกรมนี้คือโปรแกรมที่อ่านไฟล์ .RAW มาแล้วแสดงเป็นภาพต่อๆ กัน ซึ่งไฟล์ .RAW นี้จะเก็บค่าสี 3 สี คือ แดง เขียว น้ำเงิน 1 พิกเซล เรียงกันไปหลายๆ พิกเซลจนเป็นภาพ 1 ภาพแล้วเรียงต่อกันไปเป็นหลายๆ ภาพ(ใน 1 พิเซลจะมีข้อมูลของค่าสี : สีแดง 5 บิต, สีเขียว 6 บิต, สีน้ำเงิน 5 บิต : ซึ่งเป็นข้อมูลที่ใช้ในการแสดงภาพในโหมด 111H)

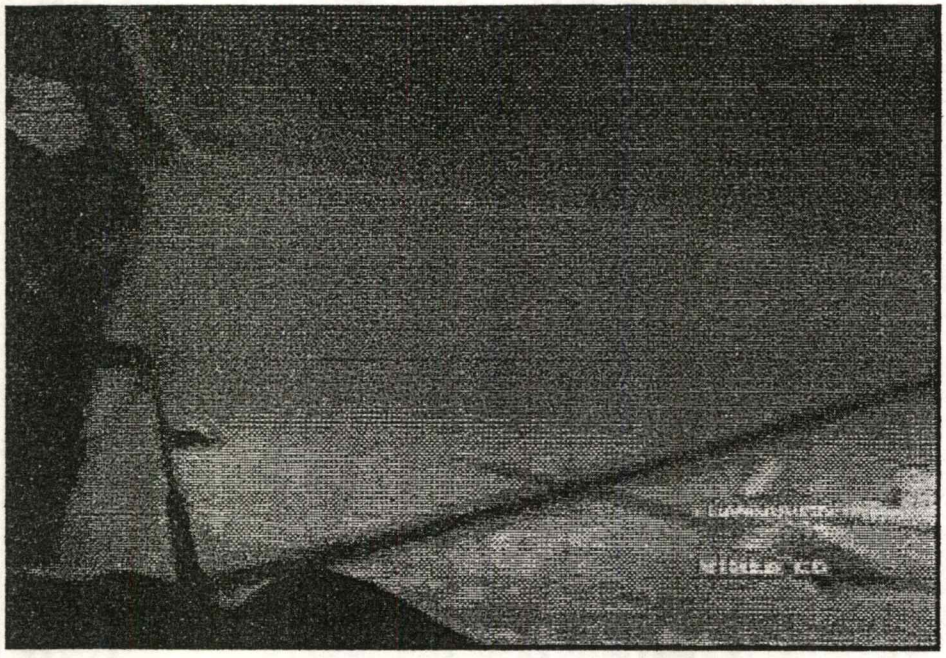
6.2 ผลการทดลอง

6.2.1 ผลการแสดงภาพเมื่อรันโปรแกรมถอดรหัสข้อมูล MPEG (DECODE.EXE)

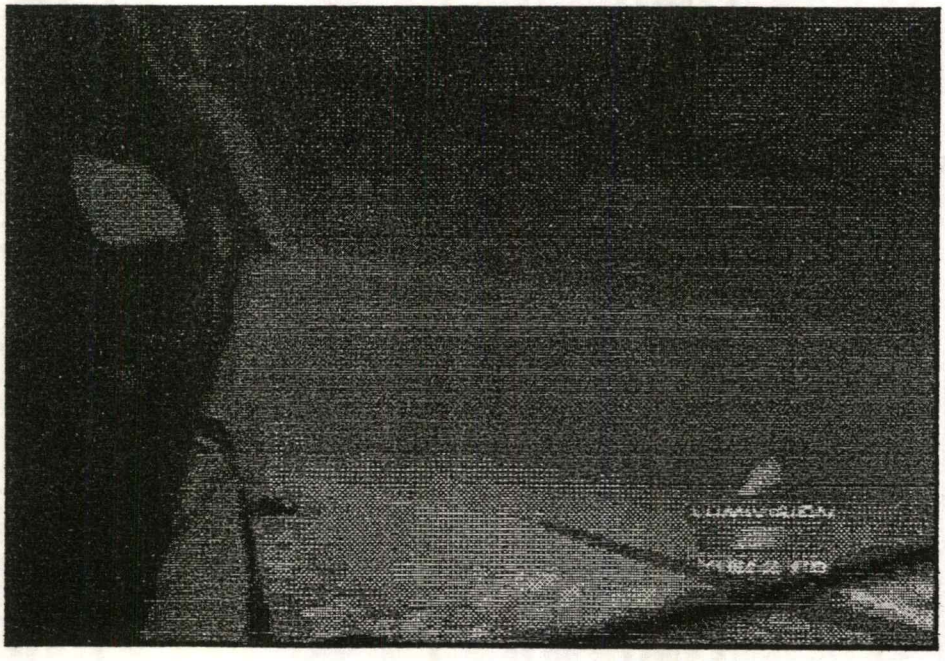
ผลของการรันโปรแกรมมีลักษณะดังรูปที่ 6.4 ,6.5 และ 6.6 โดยรูปที่ 6.4 จะเป็นภาพชนิด I รูปที่ 6.5 เป็นภาพชนิด P และ รูปที่ 6.6 เป็นภาพชนิด B ซึ่งทุกภาพมีขนาดเท่ากับของจริงคือ 352*240 พิกเซล ส่วนรูปที่ 6.7 จะเป็นภาพที่ถูกย่อมีขนาด 200*136 (ใช้โปรแกรม Photoshop) จำนวน 20ภาพตามลำดับ จากผลการทดลองจะเห็นว่าบางภาพอาจไม่สมบูรณ์ ซึ่งอาจเกิดจากขั้นตอน การคำนวณค่า IDCT เกิดการผิดพลาด หรืออาจเกิดจาก การหาค่า บล็อค แบบมี motion vector ขนาด 8x8 ได้ไม่ถูกต้อง ทำให้ค่าสีที่อ่านได้ไม่ตรงตามความเป็นจริง



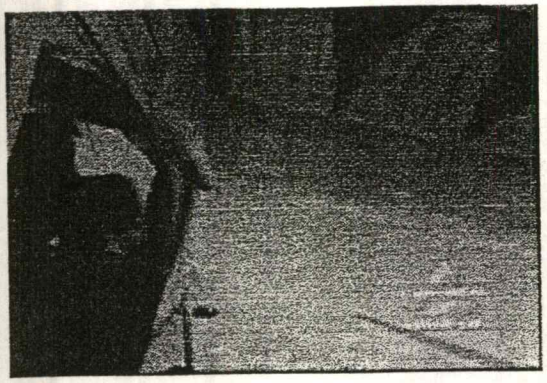
รูปที่ 6.4 ภาพชนิด I(ภาพจากเฟรมที่3)



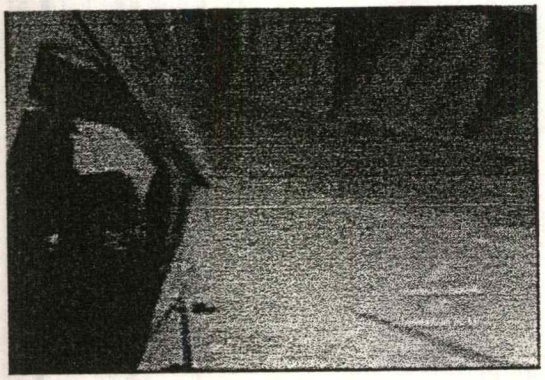
รูปที่ 6.5 ภาพชนิด P(ภาพจากเฟรมที่ 6)



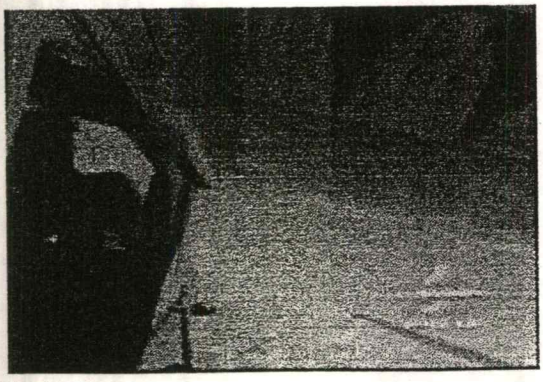
รูปที่ 6.6 ภาพชนิด B(ภาพจากเฟรมที่ 4)



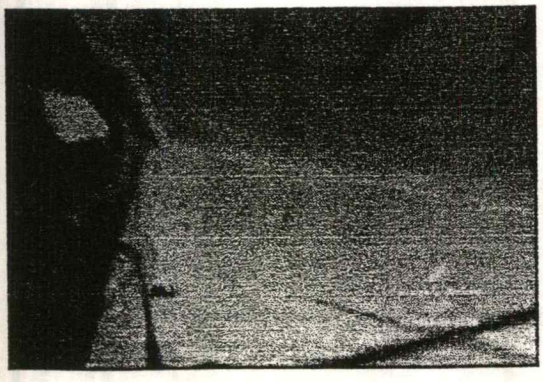
รูปที่ 6.7 เฟรมที่1 ภาพชนิด B



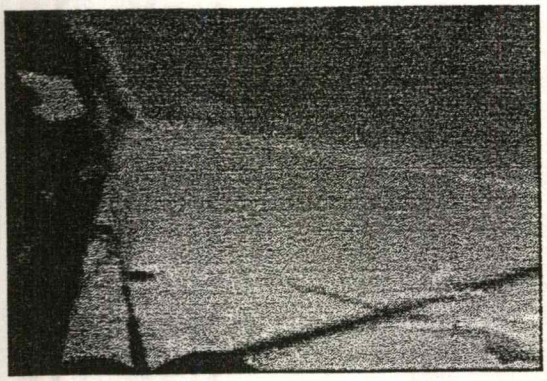
รูปที่ 6.7 เฟรมที่2 ภาพชนิด B



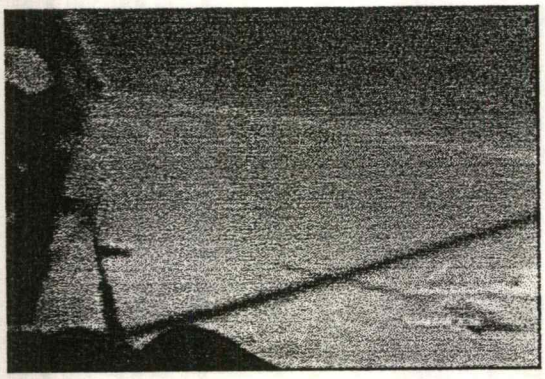
รูปที่ 6.7 เฟรมที่3 ภาพชนิด I



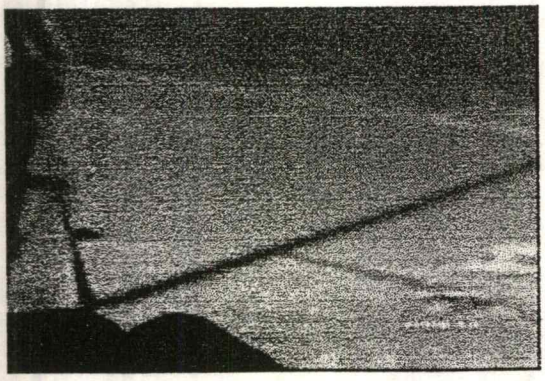
รูปที่ 6.7 เฟรมที่4 ภาพชนิด B



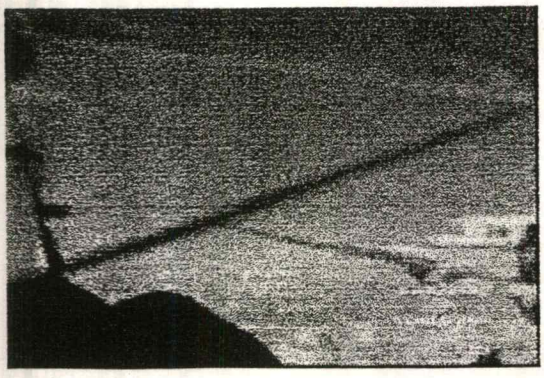
รูปที่ 6.7 เฟรมที่5 ภาพชนิด B



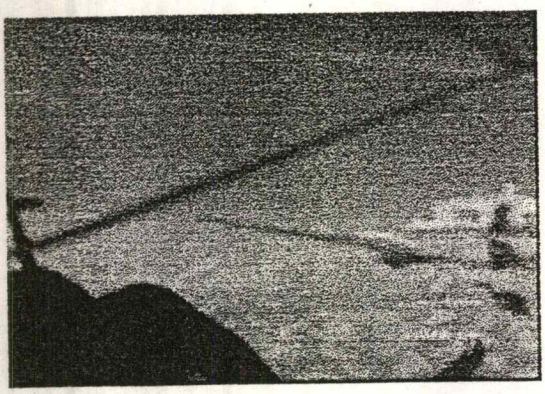
รูปที่ 6.7 เฟรมที่6 ภาพชนิด P



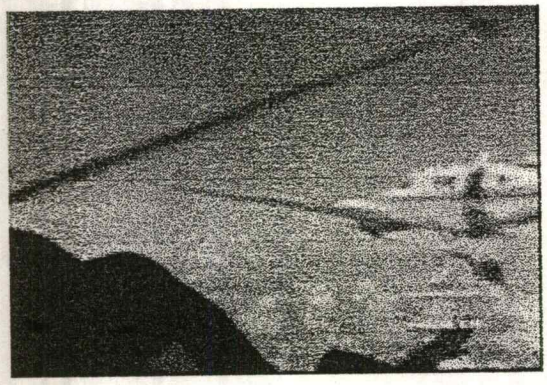
รูปที่ 6.7 เฟรมที่7 ภาพชนิด B



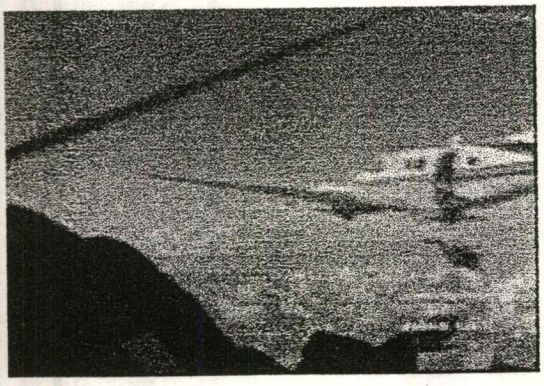
รูปที่ 6.7 เฟรมที่8 ภาพชนิด B



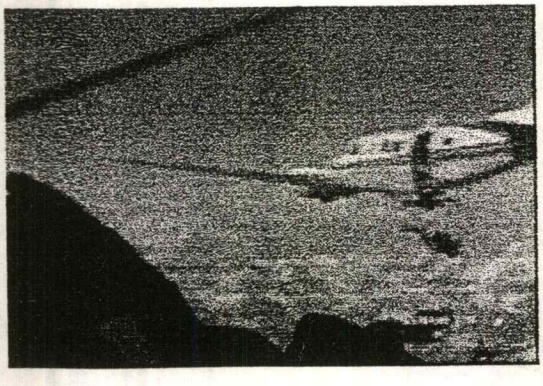
รูปที่ 6.7 เฟรมที่9 ภาพชนิด P



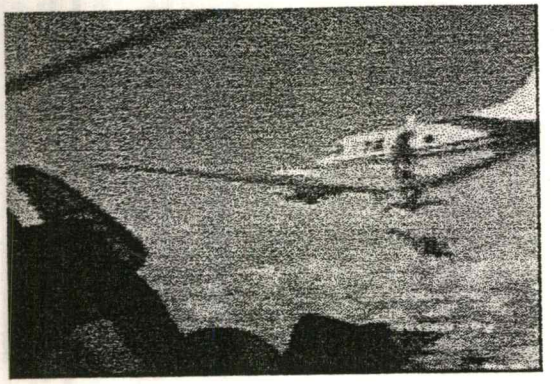
รูปที่ 6.7 เฟรมที่10 ภาพชนิด B



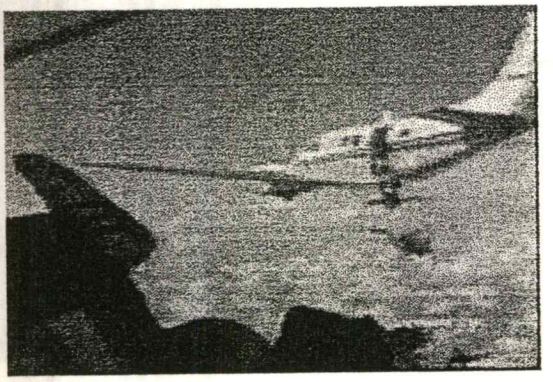
รูปที่ 6.7 เฟรมที่11 ภาพชนิด B



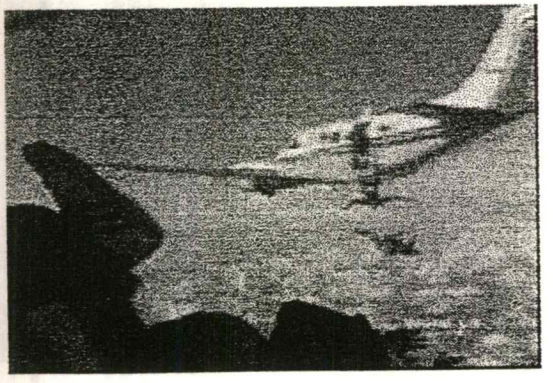
รูปที่ 6.7 เฟรมที่12 ภาพชนิด P



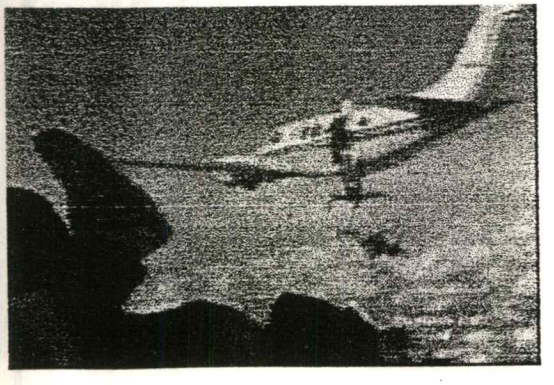
รูปที่ 6.7 เฟรมที่13 ภาพชนิด B



รูปที่ 6.7 เฟรมที่14 ภาพชนิด B



รูปที่ 6.7 เฟรมที่15 ภาพชนิด P



รูปที่ 6.7 เฟรมที่16 ภาพชนิด B

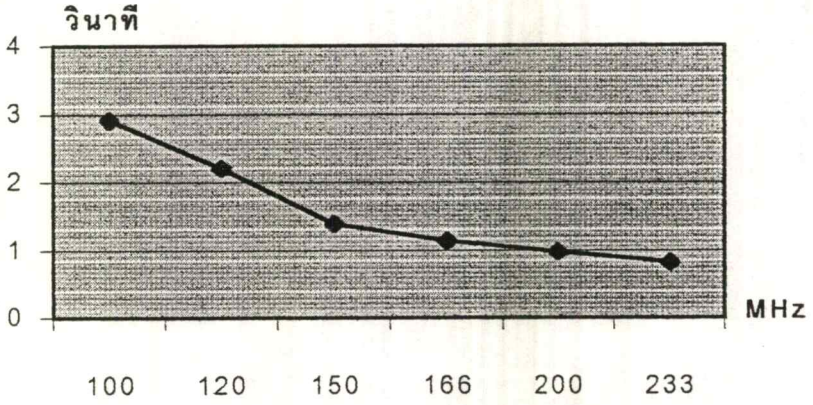
6.2.2 ค่าเวลาในการถอดรหัสภาพแต่ละชนิด

ในการที่จะวัดความเร็วการถอดรหัสภาพของโปรแกรมนั้น ในที่นี้เราได้ทำการวัดโดยให้โปรแกรมทำการถอดรหัสไฟล์ TEST.MPG (ขนาด 352 x 240) บนเครื่องคอมพิวเตอร์ที่มีความเร็วของตัวไมโครโปรเซสเซอร์ต่างๆ กัน และจับเวลาการถอดรหัสภาพแต่ละชนิดหลายๆ ภาพ แล้วนำภาพแต่ละชนิดมาหาค่าเฉลี่ย ได้ค่าดังต่อไปนี้

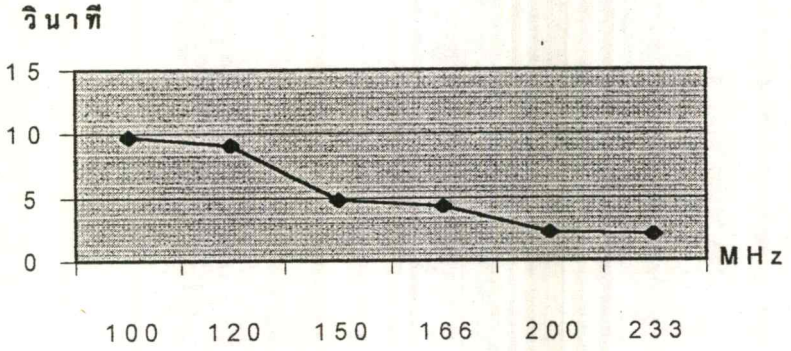
ความเร็วCPU(MHz)	ภาพชนิด I(วินาที)	ภาพชนิด P(วินาที)	ภาพชนิด B(วินาที)
เพนเทียม 100	2.915	9.812	16.112
เพนเทียม 120	2.205	9.135	15583
เพนเทียม 150	1.381	4.772	7.826
เพนเทียม 166	1.137	4.273	7.312
เพนเทียม 200	0.973	2.304	3.449
เพนเทียม 233	0.83	2.04	3.015

ตารางที่ 6.1 ค่าเวลาในการถอดรหัสภาพแต่ละชนิดเปรียบเทียบกับความเร็วซีพียู

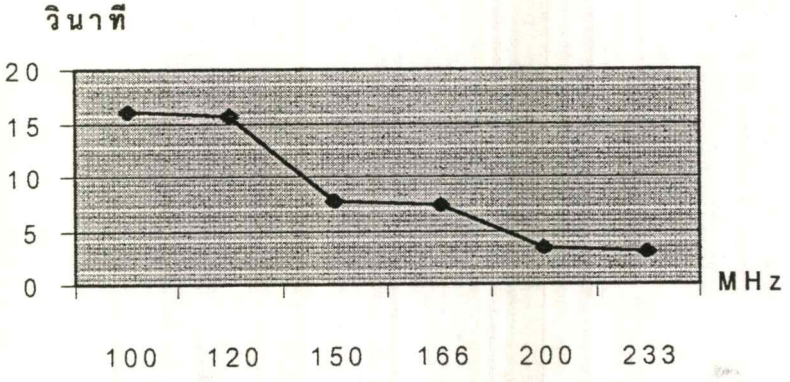
ซึ่งนำมาแสดงเป็นกราฟได้ ดังนี้



รูปที่ 6.8 กราฟแสดงค่าเวลาในการถอดรหัสภาพชนิด I



รูปที่ 6.9 กราฟแสดงค่าเวลาในการถอดรหัสภาพชนิด P



รูปที่ 6.10 กราฟแสดงค่าเวลาในการถอดรหัสภาพชนิด B

6.3 สรุป

โปรแกรมที่ผู้จัดทำได้เขียนนั้นยังมีข้อผิดพลาดอยู่หลายส่วนด้วยกัน เช่นขั้นตอนการแปลง IDCT (Inverse Discrete Cosine Transform) ยังช้าอยู่มาก รวมทั้งขั้นตอนการถอดรหัสในยังมีข้อผิดพลาดทำให้ภาพที่ออกมาไม่สมบูรณ์ทุกภาพ ซึ่งควรที่จะมีการแก้ไขต่อไปให้ถูกต้อง

ข้อผิดพลาดอีกประการหนึ่งคือ เป็นความยุ่งยาก,ล้าสมัย และล่าช้า ในการที่จะเขียนโปรแกรมที่ต้องการความเร็วสูง, เนื้อที่หน่วยความจำจำนวนมาก, และการแสดงผลทางกราฟิกส์ในระดับความละเอียดสูง ในระบบปฏิบัติการแบบ DOS ที่ในปัจจุบันไม่ค่อยมีผู้ใช้แล้ว อันเนื่องมาจากได้มีการพัฒนาระบบปฏิบัติการเป็นแบบ GUI(Graphics User Interface) ทั้งยังมีความเร็วสูงอีกด้วย เนื่องจาก การสนับสนุนการทำงานส่งผ่านข้อมูลแบบ 32 บิต เพราะฉะนั้นในการที่จะพัฒนาโปรแกรมต่อไปเราจำเป็นที่จะต้องศึกษาการเขียนโปรแกรมภาษาC++ ทางด้านกราฟิกส์ที่คอมไพล์ผ่านวินโดว์ 32บิทหรือวินโดว์ 95,98 ให้มากขึ้น

ภาคผนวก ก

(normative)

ตารางแสดงค่า Variable length code

บทนำ

Annex นี้มีรายละเอียดเกี่ยวกับตารางแสดงค่า variable length code สำหรับการกำหนดตำแหน่งของมาโครบล็อก (macroblock addressing), การแสดงชนิดของมาโครบล็อก (macroblock type), การแสดงรูปแบบของมาโครบล็อก (macroblock pattern), เวคเตอร์ของการเคลื่อนไหว (motion vectors) และสัมประสิทธิ์ของ DCT (DCT coefficients)

ก.1 การกำหนดตำแหน่งของมาโครบล็อก

macroblock_address_increment VLC code	increment value
1	1
011	2
010	3
0011	4
0010	5
0001 1	6
0001 0	7
0000 111	8
0000 110	9
0000 1011	10
00001010	11
00001001	12
0000 1000	13

macroblock_address_increment VLC code	increment value
0000 0111	14
0000 0110	15
0000 0101 11	16
0000 0101 10	17
0000 0101 01	18
0000 0101 00	19
0000 0100 11	20
0000 0100 10	21
0000 0100 011	22
0000 0100 010	23
0000 0100 001	24
0000 0100 000	25
0000 0011 111	26
0000 0011 110	27
0000 0011 101	28
0000 0011 100	29
0000 0011 011	30
0000 0011 010	31
0000 0011 001	32
0000 0011 000	33
0000 0001 111	macroblock_stuffing
0000 0001 000	macroblock_escape

ตารางที่ ก.1 Variable length code สำหรับ macroblock_address_increment

ก.2 การแสดงชนิดของมาโครบล็อก

คุณสมบัติของมาโครบล็อกจะถูกกำหนดด้วยการแสดงชนิดของมาโครบล็อกของ VLC ดังตารางต่อไปนี้

Macroblock_type VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	0	0	0	1
01	1	0	0	0	1

ตารางที่ ก.2a. VLC สำหรับ macroblock_type ในภาพชนิด I

Macroblock_type VLC code	Macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	1	0	1	0
01	0	0	0	1	0
001	0	1	0	0	0
00011	0	0	0	0	1
00010	1	1	0	1	0
00001	1	0	0	1	0
000001	1	0	0	0	1

ตารางที่ ก.2b. VLC สำหรับ macroblock_type ในภาพชนิด P

macroblock_type VLC code	Macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
10	0	1	1	0	0
11	0	1	1	1	0
010	0	0	1	0	0
011	0	0	1	1	0
0010	0	1	0	0	0
0011	0	1	0	1	0
00011	0	0	0	0	1
00010	1	1	1	1	0
000011	1	1	0	1	0
000010	1	0	1	0	0
000001	1	0	0	0	1

ตารางที่ ก.2c. VLC สำหรับ macroblock_type ในภาพชนิด B

macroblock_type VLC code	Macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra
1	0	0	0	0	1

ตารางที่ ก.2d. VLC สำหรับ macroblock_type ในภาพชนิด D

ก.3 การแสดงรูปแบบของมาโครบล็อก

code_block_pattern VLC code	cbp
111	60
1101	4
1100	8
1011	16
1010	32
1001 1	12
1001 0	48
1000 1	20
1000 0	40
0111 1	28
0111 0	44
0110 1	52
0110 0	56
0101 1	1
0101 0	61
0100 1	2
0100 0	62
001111	24
0011 10	36
0011 01	3

code_block_pattern VLC code	cbp
0011 00	63
0010 111	5
0010 110	9
0010 101	17
0010 100	33
0010 011	6
0010 010	10
0010 001	18
0010 000	34
0001 1111	7
0001 1110	11
0001 1101	19
0001 1100	35
0001 1011	13
0001 1010	49
0001 1001	21
0001 1000	41
0001 0111	14
0001 0110	50
0001 0101	22
0001 0100	42
0001 0011	15

code_block_pattern VLC code	cbp
0001 0010	51
0001 0001	23
00010000	43
0000 1111	25
0000 1110	37
0000 1101	26
0000 1100	38
0000 1011	29
0000 1010	45
0000 1001	53
0000 1000	57
0000 0111	30
0000 0110	46
0000 0101	54
0000 0100	58
0000 0011 1	31
0000 0011 0	47
0000 0010 1	55
0000 0010 0	59
0000 0001 1	27
0000 0001 0	39

ตารางที่ ก.3 VLC สำหรับ coded_block_pattern

ก.4 เวคเตอร์ของการเคลื่อนไหว

Motion VLC code	code
0000 0011 001	-16
0000 0011 011	-15
0000 0011 101	-14
0000 0011 111	-13
0000 0100 001	-12
0000 0100 011	-11
0000 0100 11	-10
0000 0101 01	-9
0000 0101 11	-8
0000 0111	-7
0000 1001	-6
0000 1011	-5
0000 111	-4
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2
0001 0	3
0000 110	4
0000 1010	5
0000 1000	6
0000 0110	7
0000 0101 10	8

motion VLC code	code
0000 0101 00	9
0000 0100 10	10
0000 0100 010	11
0000 0100 000	12
0000 0011 110	13
0000 0011 100	14
0000 0011 010	15
0000 0011 000	16

ตารางที่ ก.4 VLC สำหรับ motion_horizontal_forward_code, motion_vertical_forward_code, motion_horizontal_backward_code และ motion_vertical_backward_code

ก.5 สัมประสิทธิ์ของ DCT

VLC code	dct_dc_size_luminance
100	0
00	1
01	2
101	3
110	4
1110	5
11110	6
111110	7
1111110	8

ตารางที่ ก.5a VLC สำหรับ dct_dc_size_luminance

VLC code	dct_dc_size_luminance
00	0
01	1
10	2
110	3
1110	4
11110	5
111110	6
1111110	7
11111110	8

ตารางที่ ก.5b VLC สำหรับ dct_dc_size_chrominance

dct_coeff_first and dct_coeff_next variable length code (NOTE1)	run	level
10	end_of_block	
1 s (NOTE2)	0	1
11 s (NOTE3)	0	1
011 s	1	1
0100 s	0	2
0101 s	2	1
0010 1 s	0	3
0011 1 s	3	1
0011 0 s	4	1

dct_coeff_first and dct_coeff_next variable length code (NOTE1)	run	level
0001 10 s	1	2
0001 11 s	5	1
0001 01 s	6	1
0001 00 s	7	1
0000 110 s	0	4
0000 100 s	2	2
0000 111 s	8	1
0000 101 s	9	1
0000 01	escape	
0010 0110 s	0	5
0010 0001 s	0	6
0010 0101 s	1	3
0010 0100 s	3	2
0010 0111 s	10	1
0010 0011 s	11	1
0010 0010 s	12	1
0010 0000 s	13	1
0000 0010 10 s	0	7
0000 0011 00 s	1	4
0000 0010 11 s	2	3
0000 0011 11 s	4	2
0000 0010 01 s	5	2
0000 0011 10 s	14	1
0000 0011 01 s	15	1
0000 0010 00 s	16	1

NOTES

1. The list bit 's' denotes the sign of the level , '0' for positive, '1' for negative.
2. This code shall be used for dct_coeff_first.
3. This code shall be used for dct_coeff_next

ตารางที่ ก.5c. VLC สำหรับ dct_coeff_first และ dct_coeff_next

dct_coeff_first and dct_coeff_next variable length code (NOTE)	run	level
0000 0001 1101 s	0	8
0000 0001 1000 s	0	9
0000 0001 0011 s	0	10
0000 0001 0000 s	0	11
0000 0001 1011 s	1	5
0000 0001 0100 s	2	4
0000 0001 1100 s	3	3
0000 0001 0010 s	4	3
0000 0001 1110 s	6	2
0000 0001 0101 s	7	2
0000 0001 0001 s	8	2
0000 0001 1111 s	17	1
0000 0001 1010 s	18	1
0000 0001 1001 s	19	1
0000 0001 0111 s	20	1
0000 0001 0110 s	21	1
0000 0000 1101 0 s	0	12
0000 0000 1100 1 s	0	13

dct_coeff_first and dct_coeff_next variable length code (NOTE)	Run	level
0000 0000 1100 0 s	0	14
0000 0000 1011 1 s	0	15
0000 0000 1011 0 s	1	6
0000 0000 1010 1 s	1	7
0000 0000 1010 0 s	2	5
0000 0000 1001 1 s	3	4
0000 0000 1001 0 s	5	3
0000 0000 1000 1 s	9	2
0000 0000 1000 0 s	10	2
0000 0000 1111 1 s	22	1
0000 0000 1111 0 s	23	1
0000 0000 1110 1 s	24	1
0000 0000 1110 0 s	25	1
0000 0000 1101 1 s	26	1
NOTE – The last bit 's' denotes the sign of the level, '0' for positive, '1' for negative		

ตารางที่ ก.5d. VLC สำหรับ dct_coeff_first และ dct_coeff_next

Dct_coeff_first and dct_coeff_next Variable length code (NOTE)	run	level
0000 0000 0111 11 s	0	16
0000 0000 0111 10 s	0	17
0000 0000 0111 01 s	0	18
0000 0000 0111 00 s	0	19
0000 0000 0110 11 s	0	20
0000 0000 0110 10 s	0	21
0000 0000 0110 01 s	0	22
0000 0000 0110 00 s	0	23
0000 0000 0101 11 s	0	24
0000 0000 0101 10 s	0	25
0000 0000 0101 01 s	0	26
0000 0000 0101 00 s	0	27
0000 0000 0100 11 s	0	28
0000 0000 0100 10 s	0	29
0000 0000 0100 01 s	0	30
0000 0000 0100 00 s	0	31
0000 0000 0011 000 s	0	32
0000 0000 0010 111 s	0	33
0000 0000 0010 110 s	0	34
0000 0000 0010 101 s	0	35
0000 0000 0010 100 s	0	36
0000 0000 0010 011 s	0	37
0000 0000 0010 010 s	0	38
0000 0000 0010 001 s	0	39
0000 0000 0010 000 s	0	40

Dct_coeff_first and dct_coeff_next Variable length code (NOTE)	Run	level
0000 0000 0011 111 s	1	8
0000 0000 0011 110 s	1	9
0000 0000 0011 101 s	1	10
0000 0000 0011 100 s	1	11
0000 0000 0011 011 s	1	12
0000 0000 0011 010 s	1	13
0000 0000 0011 001 s	1	14
0000 0000 0001 0011 s	1	15
0000 0000 0001 0010 s	1	16
0000 0000 0001 0001 s	1	17
0000 0000 0001 0000 s	1	18
0000 0000 0001 0100 s	6	3
0000 0000 0001 1010 s	11	2
0000 0000 0001 1001 s	12	2
0000 0000 0001 1000 s	13	2
0000 0000 0001 0111 s	14	2
0000 0000 0001 0110 s	15	2
0000 0000 0001 0101 s	16	2
0000 0000 0001 1111 s	27	1
0000 0000 0001 1110 s	28	1
0000 0000 0001 1101 s	29	1
0000 0000 0001 1100 s	30	1
0000 0000 0001 1011 s	31	1
NOTE – The last bit 's' denotes the sign of the level. '0' for positive, '1' for negative		

ตารางที่ ก.5e. VLC สำหรับ dct_coeff_first และ dct_coeff_next (concluded)

fixed length code	run
0000 00	0
0000 01	1
0000 10	2
…	…
…	…
…	…
…	…
…	…
…	…
1111 11	63

Fixed length code	level
Forbidden	-256
1000 0000 0000 0001	-255
1000 0000 0000 0010	-254
…	
1000 0000 0111 1111	-129
1000 0000 1000 0000	-128
1000 0001	-127
1000 0010	-126
…	…
1111 1110	-2
1111 1111	-1
Forbidden	0
0000 0001	1
…	…
0111 1111	127

Fixed length code	level
0000 0000 1000 0000	128
0000 0000 1000 0001	129
...	
0000 0000 1111 1111	255

ตารางที่ ก.5f. การเข้ารหัสของ run และ level ที่ทำตาม an escape code ทั้ง 14-bit fixed length code (-127 <= level <= 127) และ 22-bit fixed length code (-255 <= level <= -128, 128 <= level <= 255)

บรรณานุกรม

[1] Kluwer Academic Publisher,"IMAGE AND VIDEO COMPRESSION STANDARD Algorithms and Architetures",1995.

[2] ISO/1EC, "INTERNATIONAL STANDARD ISO/IEC 11172-1 Information technology Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s Part 1:System",1993

[3] ISO/1EC, "INTERNATIONAL STANDARD ISO/IEC 11172-2 Information technology Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s Part 2:Video",1993

[4] Phillip E. Mattison,"Practical Digital Video With Programming Examples In C", John Wiley & Sons Ins,1994

[5] Rexe Bradford,"Real-time Animation Toolkit in C++",John Wiley & Sons Ins

[6] Loren Heiny,"Advanced Graphics Programming Using C/C++",John Wiley & Sons Ins

[7] นายชาญวิทย์ แสงอรุณบริสุทธิ์, "การศึกษาการถอดรหัสมาตรฐานเอ็มเพ็ค", 2540

[8] นายธันวา ศรีประโมง,"การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม พิมพ์ครั้งที่ 4, โครงการตำราวิชาการ มหาวิทยาลัยเทคโนโลยีมหานคร

โปรแกรมแปลงไฟล์ MPEG เป็น RAW

```
/*              MPEG decode.C – Project */
#define BUFSIZE 8192
#define PI 3.14159265358979323846
#define picture_start_code       0x100
#define user_data_start_code     0x1B2
#define sequence_header_code     0x1B3
#define sequence_error_code      0x1B4
#define extension_start_code     0x1B5
#define sequence_end_code        0x1B7
#define group_start_code         0x1B8
#define iso_11172_end_code       0x1B9
#define pack_start_code          0x1BA
#define system_header_start_code 0x1BB

#include "Dct.h"
#include "vlc.h"
FILE *mpgfile;
FILE *testfile;
char stopchar;
void iso11172_stream(void);
void pack(void);
void system_header(void);
void packet(void);

void video_sequence(void);
void sequence_header(void);
void group_of_pictures(void);
void picture(void);
void slice(void);
void macroblock(void);
void block(int a);

void Initialize(void);
void InitialBuffer(void);
void InverseDCT(int input[8][8] , int *output[8][8]);
double C[8][8] , Ct [8][8];

int slice_start_code(void);
void macroblock_type(void);

long sub_nextkeyword(int a);
long nextkeyword(int a);
long nextbit(int a);
long nextbits(int a);
int shift_left(int a,int b);
int shift_right(int a,int b);
void Type_of_stream(void);
char buffer(void);
```

```
void next_start_code(void);
void sub_next_start_code(void);
void sub_pack(void);
void sub_packet(void);

char sub_buffer(void);

int vlc_macroblock_address_increment(void);
int vlc_motion_code(void);
int vlc_code_block_pattern(void);
int vlc_dct_dc_size_luminance(void);
int vlc_dct_dc_size_chrominance(void);
void vlc_dct_coeff(int *run_level,int first);

void dequantize_ipicture(int block_number);
void dequantize_nonipicture(int block_number);
void keep_picture(void);
void block_search(int block_number);
void motion_rebuild(void);

int pattern_code[6];
int packet_length,packet_data_length,packet_data_byte;
int stream_id,reserved_stream=188,private_stream_1=189,padding_stream=190,private_stream_2=191;
int macroblock_address_increment,cbp,macroblock_address=-1;
int forward_f_code,forward_r_size,forward_f,full_pel_forward_vector;
int backward_f_code,backward_r_size,backward_f,full_pel_backward_vector;
int          motion_horizontal_forward_code,motion_vertical_forward_code;
int          motion_horizontal_backward_code,motion_vertical_backward_code;
int          motion_horizontal_forward_r,motion_vertical_forward_r;
int          motion_horizontal_backward_r,motion_vertical_backward_r;
int past_intra_address,dct_dc_y_past,dct_dc_cb_past,dct_dc_cr_past;
int          picture_coding_type;
int recon_right_for,recon_down_for;
int recon_right_back,recon_down_back;
int recon_right_for_prev=0,recon_down_for_prev=0;
int recon_right_back_prev=0,recon_down_back_prev=0;
int past_picture[3][240][352],next_picture[3][240][352];
int recent_picture[3][240][352],pel[8][8];
int picture_out[15][22][6][8][8],block_out[8][8];
int block_idct_out[8][8],data_out[64];

int video_stream_flag=0,stream_jump_flag=0;
int close_gop_flag,coeff_first_flag,recon_ppic_flag=0;

char bit_out;
long byte_no=1;
int start_code[4],quantizer_scale;
int data_code[8],data_byte;
```

```
int macroblock_number=0,slice_number=0,picture_number=0,GOP_number=0;
char buffer_code[64];
int video_buffer[3][81920],buffer_length[3];
int write_mem=0,read_mem,mem_point,bit_amout;
int bit_test=0,test_space=0;
int ipicture_number=0,ppicture_number=0,bpicture_number=0;

int macroblock_quant,macroblock_motion_forward;
int macroblock_motion_backward,macroblock_pattern,macroblock_intra;

int intra_quantizer_matrix[8][8]=
{
                    {8,16,19,22,26,27,29,34},
                    {16,16,22,24,27,29,34,37},
                    {19,22,26,27,29,34,34,38},
                    {22,22,26,27,29,34,37,40},
                    {22,26,27,29,32,35,40,48},
                    {26,27,29,32,35,40,48,58},
                    {26,27,29,34,38,46,56,69},
                    {27,29,35,38,46,56,69,83}
};

int non_intra_quantizer_matrix[8][8]=
{
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16},
                    {16,16,16,16,16,16,16,16}

};

int zzscan[8][8]=
{
                    {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}

};
```

```
void main(void)
{
 char *FileBuf;
        int k,i,l,j;
        mpgfile=fopen("e:\\mpeg\\Test.mpg","rb");
 if (mpgfile==NULL)
  {
  printf (" Can't open file ! \n");
  getch();
  exit(1);
  }

        testfile=fopen("Ptest.raw","wb");
 if (testfile==NULL)
  {
  printf (" Can't open file ! \n");
  getch();
  exit(1);
  }

 if ((FileBuf = (char *)malloc(BUFSIZE))!=NULL)
   setvbuf(mpgfile,FileBuf,_IOFBF,BUFSIZE);
 else
   printf("Can't Allocate memory!! \n");


        byte_no=1;
 bit_out=1;
 Initialize();
 InitialBuffer();


        iso11172_stream();

 //     printf("\nByte number = %d\n",byte_no);
        stopchar=getch();

}


void iso11172_stream(void)
{

        do
        {
                pack();
                if(nextbit(32)==439)
                        nextkeyword(32);
```

```
        sub_nextkeyword(1);
        sub_nextkeyword(1);
        sub_nextkeyword(1);
        sub_nextkeyword(1);
        sub_nextkeyword(5);
        sub_nextkeyword(8);

        while(nextbit(1)==1)
        {
                sub_nextkeyword(8);
                if(sub_nextkeyword(2)!=3)
                {
                        printf("header error");
                        stopchar=getch();
                }

                sub_nextkeyword(1);
                sub_nextkeyword(13);
        }
}


void packet(void)
{
        int i,j;
  long k;
        char ch;

        if(sub_nextkeyword(24)!=1)
        {
                printf("packet_start_code_prefix  ERROR!! \n");
                stopchar=getch();
        }

        stream_id=sub_nextkeyword(8);
        packet_length=sub_nextkeyword(16);

        Type_of_stream();


        k=byte_no;

        if(stream_id!=private_stream_2)
        {
                while(nextbit(8)==255)
                        sub_nextkeyword(8);
                if(nextbit(2)==1)
                {
```

```
        }while(nextbit(32)==442);
 // where 's the iso_11172_end_code ??
}


void pack(void)
{
        if(sub_nextkeyword(32)!=442)                                                    //pack_start_code
        {
                printf("pack_start_code  ERROR !!!\n");
                stopchar=getch();
        }

        if(sub_nextkeyword(4)!=2)
        {
                printf("pack error\n");
                stopchar=getch();
        }

        sub_nextkeyword(3);
        sub_nextkeyword(1);
        sub_nextkeyword(15);
        sub_nextkeyword(1);
        sub_nextkeyword(15);
        sub_nextkeyword(1);
        sub_nextkeyword(1);
        sub_nextkeyword(22);
        sub_nextkeyword(1);

        if(nextbit(32)==443)
                system_header();

        while((nextbit(24)==1)&&(nextbit(32)!=442)&&(nextbit(32)!=441)&&(nextbit(32)!=439))
                packet();

}


void system_header(void)
{

        sub_nextkeyword(32);
        sub_nextkeyword(16);
        sub_nextkeyword(1);
        sub_nextkeyword(22);
        sub_nextkeyword(1);
        sub_nextkeyword(6);
        sub_nextkeyword(1);
```

```
                sub_nextkeyword(2);
                sub_nextkeyword(1);
                sub_nextkeyword(13);
        }
        if(nextbit(4)==2)
        {
                sub_nextkeyword(4);
                sub_nextkeyword(3);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
        }
        else if(nextbit(4)==3)
        {
                sub_nextkeyword(4);
                sub_nextkeyword(3);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);

                if(sub_nextkeyword(4)!=1)
                {
                        printf("Packet ERROR");
                        stopchar=getch();
                }

                sub_nextkeyword(3);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
        }
        else
        {
                if(sub_nextkeyword(8)!=15)
                {
                        printf("Packet ERROR !!");
                        stopchar=getch();
                }

        }
}
```

```
        k=byte_no-k;

        packet_data_length=packet_length-k;
        packet_data_byte=-1;

        if(video_stream_flag)
        {

                for(i=0;i<packet_data_length;i++)
                {

                        if(i<8)
                        {
                                for(j=0;j<8;j++)
                                        video_buffer[write_mem][i*8+j]=buffer_code[i*8+j];
                        }
                        else
                        {
                                ch=fgetc(mpgfile);
    byte_no++;
                                for(j=0;j<8;j++)
                                {
                                        k=ch&1;
                                        video_buffer[write_mem][8*(i+1)-(j+1)]=k;
                                        ch>>=1;
                                }
                        }

                }

InitialBuffer();

                buffer_length[write_mem]=packet_data_length;
                write_mem++;
                video_stream_flag=0;
                if(write_mem==2)
                {
                        read_mem=0;
                        mem_point=0;
                        bit_amout=buffer_length[read_mem]*8;
                        video_sequence();
                }
        }
        else
        {
                for(i=0;i<packet_data_length;i++)
                        sub_nextkeyword(8);
        }
```

```
}

void video_sequence(void)
{

        next_start_code();
        do
        {
                sequence_header();
                do
                {
                        group_of_pictures();
                        GOP_number++;
                        picture_number=1;
                }while(nextbits(32)==group_start_code);
        }while(nextbits(32)==sequence_header_code);
        nextkeyword(32);

}


void sequence_header(void)
{
        if(nextkeyword(32)!=0x1b3)
        {
                printf("/n sequence_header code ERROR !! /n");
                stopchar=getch();
        }

        nextkeyword(12);
        nextkeyword(12);
        nextkeyword(4);
        nextkeyword(4);
        nextkeyword(18);
        nextkeyword(1);
        nextkeyword(10);
        nextkeyword(1);
        if(nextkeyword(1)==1)
                nextkeyword(8*64);
        if(nextkeyword(1)==1)
                nextkeyword(8*64);

        next_start_code();
        if(nextbits(32)==extension_start_code)
        {
                nextkeyword(32);
```

```
                while(nextbits(24)!=1)
                        nextkeyword(8);
                next_start_code();
        }

        if(nextbits(32)==user_data_start_code)
        {
                nextkeyword(32);
                while(nextbits(24)!=1)
                        nextkeyword(8);
                next_start_code();
        }

}

void group_of_pictures(void)
{

        if(nextkeyword(32)!=group_start_code)
        {
                printf("/n group_start_code ERROR !! /n");
                stopchar=getch();
        }

  if (GOP_number==9)
   {
   fclose(testfile);
   exit(1);
   }
        printf("\nstart of GOP %d",GOP_number);
//      stopchar=getch();

        nextkeyword(25);
        close_gop_flag=nextkeyword(1);
        nextkeyword(1);

        next_start_code();

        if(nextbits(32)==extension_start_code)
        {
                nextkeyword(32);
                while(nextbits(24)!=1)
                        nextkeyword(8);
                next_start_code();
        }

        if(nextbits(32)==user_data_start_code)
```

```
        {
                nextkeyword(32);
                while(nextbits(24)!=1)
                        nextkeyword(8);
                next_start_code();
        }

        do
        {
                picture();
//              printf("Number of slice = %d\n",slice_number);
                slice_number=0;
                picture_number++;

        }while(nextbits(32)==picture_start_code);

}


void picture(void)
{

  char *pictype[5]= { "error","I","P","B","D" };
        int k,l,m,n;

        if(nextkeyword(32)!=picture_start_code)
        {
                printf("/n picture_start_code ERROR !! /n");
                stopchar=getch();
        }

        printf("\n\nstart of picture %d  of  GOP ! %d\n",picture_number,GOP_number);
        nextkeyword(10);
        picture_coding_type=nextkeyword(3);
        printf("picture 's type %s\n",pictype[picture_coding_type]);
        nextkeyword(16);
//      stopchar=getch();

        if((picture_coding_type==2)&&(recon_ppic_flag))
        {
                for(m=0;m<240;m++)
                {
                        for(n=0;n<352;n++)
                        {
                                k=floor(m/2);
                                l=floor(n/2);
                                past_picture[0][m][n]=next_picture[0][m][n];
                                past_picture[1][k][l]=next_picture[1][k][l];
```

```
                                past_picture[2][k][l]=next_picture[2][k][l];
                        }
                }
}


if((picture_coding_type==2)||(picture_coding_type==3))
{
        full_pel_forward_vector=nextkeyword(1);
        forward_f_code=nextkeyword(3);
        forward_r_size=forward_f_code-1;
        forward_f=1<<forward_r_size;
}

if(picture_coding_type==3)
{
        full_pel_backward_vector=nextkeyword(1);
        backward_f_code=nextkeyword(3);
        backward_r_size=backward_f_code-1;
        backward_f=1<<backward_r_size;
}

while(nextbits(1)==1)
{
        nextkeyword(1);
        nextkeyword(8);
}

nextkeyword(1);
next_start_code();


if(nextbits(32)==extension_start_code)
{
        nextkeyword(32);
        while(nextbits(24)!=1)
                nextkeyword(8);
        next_start_code();
}

if(nextbits(32)==user_data_start_code)
{
        nextkeyword(32);
        while(nextbits(24)!=1)
                nextkeyword(8);
        next_start_code();
}
```

```
do
{
        slice();
        slice_number++;
}while(slice_start_code());

if(picture_coding_type==3)
        bpicture_number++;

keep_picture();

if(picture_coding_type==1)
{
        for(m=0;m<240;m++)
        {
                for(n=0;n<352;n++)
                {
                        k=floor(m/2);
                        l=floor(n/2);
                        past_picture[0][m][n]=recent_picture[0][m][n];
                        past_picture[1][k][l]=recent_picture[1][k][l];
                        past_picture[2][k][l]=recent_picture[2][k][l];
                }
        }
}
else if(picture_coding_type==2)
{
        for(m=0;m<240;m++)
        {
                for(n=0;n<352;n++)
                {
                        k=floor(m/2);
                        l=floor(n/2);
                        next_picture[0][m][n]=recent_picture[0][m][n];
                        next_picture[1][k][l]=recent_picture[1][k][l];
                        next_picture[2][k][l]=recent_picture[2][k][l];
                        recon_ppic_flag=1;
                }
        }
}

}

void slice(void)
```

```
{
        int k;

        past_intra_address=-2;
        dct_dc_y_past=128*8;
        dct_dc_cr_past=128*8;
        dct_dc_cb_past=128*8;
        recon_right_for_prev=0;
        recon_down_for_prev=0;
        recon_right_back_prev=0;
        recon_down_back_prev=0;


        if (nextkeyword(24)!= 1)
  {
   printf("slice start code error!");
   stopchar = getch();
  }
  k = nextkeyword(8);
//      printf("\nslice start code = %0x \nslice_vertical_positions = %d \n",0x100|k,k);
//      stopchar=getch();

        quantizer_scale=nextkeyword(5);
        while(nextbits(1)==1)
        {
                nextkeyword(1);
                nextkeyword(8);
        }
        nextkeyword(1);
        macroblock_number=0;

        do
        {
                cbp=0;
                macroblock();
                macroblock_number++;

        }while(nextbits(23)!=0);
//      printf("Number of macroblock is %d \n",macroblock_number);
        next_start_code();

}


void macroblock(void)
{
        int i,j,k,l,m,n;
```

```
while(nextbits(11)==15)
        nextkeyword(11);

while(nextbits(11)==8)
        nextkeyword(11);

macroblock_address_increment=vlc_macroblock_address_increment();

if(macroblock_address_increment>1)
{
        if(picture_coding_type==2)
        {
                recon_right_for=0;
                recon_down_for=0;
                m=macroblock_address_increment-1;
                for(n=0;n<m;n++)
                {
                        for(i=0;i<6;i++)
                        {
                                block_search(i);
                                for(k=0;k<8;k++)
                                {
                                        for(l=0;l<8;l++)
                                                picture_out[slice_number][macroblock_number][i][k][l]=pel
[k][l];
                                }
                        }
                        macroblock_number++;
                }
        }
        else if(picture_coding_type==3)
        {
                m=macroblock_address_increment-1;
                for(n=0;n<m;n++)
                {
                        for(i=0;i<6;i++)
                        {
                                block_search(i);
                                for(k=0;k<8;k++)
                                {
                                        for(l=0;l<8;l++)
                                                picture_out[slice_number][macroblock_number][i][k][l]=pel
[k][l];
                                }
                        }
                        macroblock_number++;
                }
        }
```

```
            else
            {
                        printf("\n  skip macroblock ERROR !! \n");
                        stopchar=getch();

            }
}


macroblock_address=macroblock_address+macroblock_address_increment;
macroblock_type();


if(!macroblock_intra)
{
             dct_dc_y_past=128*8;
             dct_dc_cr_past=128*8;
             dct_dc_cb_past=128*8;
}


 if(macroblock_quant)
              quantizer_scale=nextkeyword(5);

 if(macroblock_motion_forward)
 {
              motion_horizontal_forward_code=vlc_motion_code();
              if((forward_f!=1)&&(motion_horizontal_forward_code!=0))
                        motion_horizontal_forward_r=nextkeyword(forward_r_size);

              motion_vertical_forward_code=vlc_motion_code();
              if((forward_f!=1)&&(motion_vertical_forward_code!=0))
                        motion_vertical_forward_r=nextkeyword(forward_r_size);
  }


  if(macroblock_motion_backward)
  {
              motion_horizontal_backward_code=vlc_motion_code();
              if((backward_f!=1)&&(motion_horizontal_backward_code!=0))
                        motion_horizontal_backward_r=nextkeyword(backward_r_size);

              motion_vertical_backward_code=vlc_motion_code();
              if((backward_f!=1)&&(motion_vertical_backward_code!=0))
                        motion_vertical_backward_r=nextkeyword(backward_r_size);
  }
```

```
if(picture_coding_type==2)
{
        if((!macroblock_motion_forward)||(macroblock_address_increment>1))
        {
                recon_right_for_prev=0;
                recon_down_for_prev=0;
        }

        if((!macroblock_motion_backward)||(macroblock_address_increment>1))
        {
                recon_right_back_prev=0;
                recon_down_back_prev=0;
        }
}

if(picture_coding_type==3)
{
        if(macroblock_intra)
        {
                recon_right_for_prev=0;
                recon_down_for_prev=0;
                recon_right_back_prev=0;
                recon_down_back_prev=0;
        }
}

if(!macroblock_intra)
        motion_rebuild();

if(macroblock_pattern)
        cbp=vlc_code_block_pattern();
for (i=0;i<6;i++)
{
pattern_code[i]=0;
if (cbp&(1<<(5-i))) pattern_code[i]=1;
if (macroblock_intra)pattern_code[i]=1;
}

for(i=0;i<6;i++)
{
        for(k=0;k<64;k++)
                data_out[k]=0;
        for(k=0;k<8;k++)
        {
                for(l=0;l<8;l++)
                        block_out[k][l]=0;
```

```
        }

        block(i);

        if(macroblock_intra)
                InverseDCT(block_out,&block_idct_out);
        else
        {
                InverseDCT(block_out,&block_idct_out);
                block_search(i);
                for(k=0;k<8;k++)
                {
                        for(l=0;l<8;l++)
                                block_idct_out[k][l]=block_idct_out[k][l]+pel[k][l];
                }
        }

        for(k=0;k<8;k++)
        {
                for(l=0;l<8;l++)
                {
                        j=block_idct_out[k][l];
                        if(j<0)
                                j=0;
                        //limit to range 0-255
                                else if(j>255)
                                        j=255;
                                picture_out[slice_number][macroblock_number][i][k][l]=j;
                }
        }

    }

    if(macroblock_intra)
            past_intra_address=macroblock_address;

    if(picture_coding_type==4)
            nextkeyword(1);

}

void block(int block_number)
{
  short dc_size;
```

```
unsigned int dct_dc_differential;
int data_out_number,run_level[2];

        if(pattern_code[block_number])
        {
                data_out_number=0;
                if(macroblock_intra)
                {
                        if(block_number<4)
                        {
                                dc_size=(short)vlc_dct_dc_size_luminance();

                                if(dc_size!=0)
                                {
dct_dc_differential = nextkeyword(dc_size);
                                        if( dct_dc_differential & (1<<(dc_size-1)))
  data_out[0]=dct_dc_differential;
else
                                                data_out[0]=((-1)<<(dc_size))|(dct_dc_differential+1);

                                        data_out_number++;
                                }
                                else
                                        data_out_number++;
                        }
                        else
                        {
                                dc_size = (short)vlc_dct_dc_size_chrominance();
                                if(dc_size!=0)
{
 dct_dc_differential = nextkeyword(dc_size);
                                        if( dct_dc_differential & (1<<(dc_size-1)))
   data_out[0]=dct_dc_differential;
 else
                                                data_out[0]=((-1)<<(dc_size))|(dct_dc_differential+1);

                                        data_out_number++;
}
                                 else
                                        data_out_number++;
                        }
                }
                else
                {
                        coeff_first_flag=1;
                        vlc_dct_coeff(&run_level,1);
                        data_out[run_level[0]]= run_level[1];
                        data_out_number = run_level[0]+1;
```

```
                coeff_first_flag=0;
        }

        if(picture_coding_type!=4)
        {
                while(nextbits(2)!=2)
                {
                        vlc_dct_coeff(&run_level,0);
                        data_out[data_out_number+run_level[0]]=run_level[1];
                        data_out_number=data_out_number+run_level[0]+1;
                }

                if(data_out_number>64)
                {
                        printf("BLOCK DATA OVER !!\n");
                        stopchar=getch();
                }

                if(nextkeyword(2)!=2)
                {
                        printf("End of block ERROR !!\n");
                        stopchar=getch();
                }

        }


        if(macroblock_intra)
                dequantize_ipicture(block_number);
        else
                dequantize_nonipicture(block_number);

    }

}


int vlc_macroblock_address_increment(void)
{

    if(nextkeyword(1))
            return 1;
    else if(nextkeyword(1))
    {
            if(nextkeyword(1))
                    return 2;
            else
```

```
                return 3;
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
                return 4;
        else
                return 5;
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
                return 6;
        else
                return 7;
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 8;
                else
                        return 9;
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 10;
                else
                        return 11;
        }
        else
        {
                if(nextkeyword(1))
                        return 12;
                else
                        return 13;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 14;
                else
                        return 15;
```

```
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 16;
                        else
                                return 17;
                }
                else
                {
                        if(nextkeyword(1))
                                return 18;
                        else
                                return 19;
                }
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 20;
                else
                        return 21;
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 22;
                else
                        return 23;
        }
        else
        {
                if(nextkeyword(1))
                        return 24;
                else
                        return 25;
        }

}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
```

```
                {
                        if(nextkeyword(1))
                                return 26;
                        else
                                return 27;
                }
                else
                {
                        if(nextkeyword(1))
                                return 28;
                        else
                                return 29;
                }
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 30;
                else
                        return 31;
        }
        else
        {
                if(nextkeyword(1))
                        return 32;
                else
                        return 33;
        }
    }
    else
    {
        printf("vlc_macroblock_address_increment code ERROR!!\n");
        printf("At GOP>%d Picture>%d Slice>%d
Macroblock>%d",GOP_number,picture_number,slice_number,macroblock_number);
        stopchar=getch();
    }
  }
  else
  {
        printf("vlc_macroblock_address_increment OVERZERO ERROR!!\n");
        printf("At GOP>%d Picture>%d Slice>%d
Macroblock>%d",GOP_number,picture_number,slice_number,macroblock_number);
        stopchar=getch();
  }

}
```

```
int vlc_motion_code(void)
{
        if(nextkeyword(1))
                return 0;
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return -1;
                else
                        return 1;
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return -2;
                else
                        return 2;
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return -3;
                else
                        return 3;
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return -4;
                        else
                                return 4;
                }
                else if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return -5;
                        else
                                return 5;
                }
                else
                {
                        if(nextkeyword(1))
                                return -6;
                        else
                                return 6;
```

```
        }
    }
    else if(nextkeyword(1))
    {
        if(nextkeyword(1))
        {
            if(nextkeyword(1))
                return -7;
            else
                return 7;
        }
        else if(nextkeyword(1))
        {
            if(nextkeyword(1))
            {
                if(nextkeyword(1))
                    return -8;
                else
                    return 8;
            }
            else
            {
                if(nextkeyword(1))
                    return -9;
                else
                    return 9;
            }
        }
        else if(nextkeyword(1))
        {
            if(nextkeyword(1))
                return -10;
            else
                return 10;
        }
        else if(nextkeyword(1))
        {
            if(nextkeyword(1))
                return -11;
            else
                return 11;
        }
        else
        {
            if(nextkeyword(1))
                return -12;
            else
                return 12;
```

```
                }

        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                        {
                                if(nextkeyword(1))
                                {
                                        if(nextkeyword(1))
                                        {
                                                if(nextkeyword(1))
                                                        return -13;
                                                else
                                                        return 13;
                                        }
                                        else
                                        {
                                                if(nextkeyword(1))
                                                        return -14;
                                                else
                                                        return 14;
                                        }
                                }
                                else
                                {
                                        if(nextkeyword(1))
                                        {
                                                if(nextkeyword(1))
                                                        return -15;
                                                else
                                                        return 15;
                                        }
                                        else
                                        {
                                                if(nextkeyword(1))
                                                        return -16;
                                                else
                                                        return 16;
                                        }
                                }
                        }
                }
        }
        else
        {
                printf("\n vlc_motion_code OVERZERO ERROR!!\n");
                printf("At GOP>%d Picture>%d Slice>%d
\Macroblock>%d",GOP_number,picture_number,slice_number,macroblock_number);
```

```
                stopchar=getch();
        }

}


int vlc_code_block_pattern(void)
{

        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 60;
                        else
                        {
                                if(nextkeyword(1))
                                        return 4;
                                else
                                        return 8;
                        }
                }
                else if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 16;
                        else
                                return 32;
                }
                else if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 12;
                        else
                                return 48;
                }
                else
                {
                        if(nextkeyword(1))
                                return 20;
                        else
                                return 40;
                }
        }
        else if(nextkeyword(1))
        {
```

```
if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 28;
                else
                        return 44;
        }
        else
        {
                if(nextkeyword(1))
                        return 52;
                else
                        return 56;
        }
  }
  else if(nextkeyword(1))
  {
        if(nextkeyword(1))
                return 1;
        else
                return 61;
  }
  else
  {
        if(nextkeyword(1))
                return 2;
        else
                return 62;
  }
}
else if(nextkeyword(1))
{
   if(nextkeyword(1))
   {
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 24;
                else
                        return 36;
        }
        else
        {
                if(nextkeyword(1))
                        return 3;
                else
```

```
                                        return 63;
                        }
                }
                else if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                        {
                                if(nextkeyword(1))
                                        return 5;
                                else
                                        return 9;
                        }
                        else
                        {
                                if(nextkeyword(1))
                                        return 17;
                                else
                                        return 33;
                        }
                }
                else if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 6;
                        else
                                return 10;
                }
                else
                {
                        if(nextkeyword(1))
                                return 18;
                        else
                                return 34;
                }
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                        {
                                if(nextkeyword(1))
                                {
                                        if(nextkeyword(1))
                                                return 7;
                                        else
                                                return 11;
                                }
```

```
                else
                {
                        if(nextkeyword(1))
                                return 19;
                        else
                                return 35;
                }
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 13;
                else
                        return 49;
        }
        else
        {
                if(nextkeyword(1))
                        return 21;
                else
                        return 41;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 14;
                else
                        return 50;
        }
        else
        {
                if(nextkeyword(1))
                        return 22;
                else
                        return 42;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
                return 15;
        else
                return 51;
}
else
```

```
        {
                if(nextkeyword(1))
                        return 23;
                else
                        return 43;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                {
                        if(nextkeyword(1))
                                return 25;
                        else
                                return 37;
                }
                else
                {
                        if(nextkeyword(1))
                                return 26;
                        else
                                return 38;
                }
        }
        else if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 29;
                else
                        return 45;
        }
        else
        {
                if(nextkeyword(1))
                        return 53;
                else
                        return 57;
        }

}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 30;
```

```
                else
                        return 46;
        }
        else
        {
                if(nextkeyword(1))
                        return 54;
                else
                        return 58;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1))
                        return 31;
                else
                        return 47;
        }
        else
        {
                if(nextkeyword(1))
                        return 55;
                else
                        return 59;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
                return 27;
        else
                return 39;
}

}


int vlc_dct_dc_size_luminance(void)
{

        if(nextkeyword(1)==0)
        {
                if(nextkeyword(1)==0)
                        return 1;
```

```
                else
                        return 2;
        }
        else
        {
                if(nextkeyword(1)==0)
                {
                        if(nextkeyword(1)==0)
                                return 0;
                        else
                                return 3;
                }
                else
                {
                        if(nextkeyword(1)==0)
                                return 4;
                        else
                        {
                                if(nextkeyword(1)==0)
                                        return 5;
                                else
                                {
                                        if(nextkeyword(1)==0)
                                                return 6;
                                        else
                                        {
                                                if(nextkeyword(1)==0)
                                                        return 7;
                                                else
                                                {
                                                        if(nextkeyword(1)==0)
                                                                return 8;
                                                        else
                                                        {
                                                                printf("dct_dc_size_luminance ERROR \n");
                                                                printf("At GOP>%d  Picture>%d  Slice>%d
Macroblock>%d",GOP_number,picture_number,slice_number,macroblock_number);
                                                                stopchar=getch();
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

//              k=packet_data_length*8-(packet_data_byte*8+bit_out-1);
```

```
}

int vlc_dct_dc_size_chrominance(void)
{

        if(nextkeyword(1)==0)
        {
                if(nextkeyword(1)==0)
                        return 0;
                else
                        return 1;
        }
        else
        {
                if(nextkeyword(1)==0)
                        return 2;
                else
                {
                        if(nextkeyword(1)==0)
                                return 3;
                        else
                        {
                                if(nextkeyword(1)==0)
                                        return 4;
                                else
                                {
                                        if(nextkeyword(1)==0)
                                                return 5;
                                        else
                                        {
                                                if(nextkeyword(1)==0)
                                                        return 6;
                                                else
                                                {
                                                        if(nextkeyword(1)==0)
                                                                return 7;
                                                        else
                                                        {
                                                                if(nextkeyword(1)==0)
                                                                        return 8;
                                                                else
                                                                {
                                                                        printf("dct_dc_size_chrominance
ERROR \n");
                                                                        printf("At GOP>%d Picture>%d
Slice>%d Macroblock>%d",GOP_number,picture_number,slice_number,macroblock_number);
```

```
                                                                    stopchar=getch();
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
}

void vlc_dct_coeff(int *run_level,int first)
{
  DCTtab *tab;
  unsigned int code;
  int sign;

    code = nextbits(16);

    if (code>=16384)
      {
      if (first)
        tab = &DCTtabfirst[(code>>12)-4];
      else
        tab = &DCTtabnext[(code>>12)-4];
      }
    else if (code>=1024)
      tab = &DCTtab0[(code>>8)-4];
    else if (code>=512)
      tab = &DCTtab1[(code>>6)-8];
    else if (code>=256)
      tab = &DCTtab2[(code>>4)-16];
    else if (code>=128)
      tab = &DCTtab3[(code>>3)-16];
    else if (code>=64)
      tab = &DCTtab4[(code>>2)-16];
    else if (code>=32)
      tab = &DCTtab5[(code>>1)-16];
    else if (code>=16)
      tab = &DCTtab6[code-16];
    else
      printf("invalid Huffman code \n");
```

```
  nextkeyword(tab->length);

if (tab->run==64) printf(" nextbits function error! ");


if (tab->run==65) /* escape */
 {
  run_level[0] = nextkeyword(6);

  run_level[1] = nextkeyword(8);
  if (run_level[1]==0)
   run_level[1] = nextkeyword(8);
  else if (run_level[1]==128)
   run_level[1] = nextkeyword(8) - 256;
  else if (run_level[1]>128)
   run_level[1] -= 256;
 }
 else
 {
  sign = nextkeyword(1);
  (sign == 0) ? (sign = 1) : (sign=-1);
  run_level[0]= (tab->run);
  run_level[1] = sign*(tab->level);

 }

}




int slice_start_code(void)
{

        if(nextbits(32)==0x101)
                return 1;
        else if(nextbits(32)==0x102)
                return 1;
        else if(nextbits(32)==0x103)
                return 1;
        else if(nextbits(32)==0x104)
                return 1;
```

```
        else if(nextbits(32)==0x105)
                return 1;
        else if(nextbits(32)==0x106)
                return 1;
        else if(nextbits(32)==0x107)
                return 1;
        else if(nextbits(32)==0x108)
                return 1;
        else if(nextbits(32)==0x109)
                return 1;
        else if(nextbits(32)==0x10a)
                return 1;
        else if(nextbits(32)==0x10b)
                return 1;
        else if(nextbits(32)==0x10c)
                return 1;
        else if(nextbits(32)==0x10d)
                return 1;
        else if(nextbits(32)==0x10e)
                return 1;
        else if(nextbits(32)==0x10f)
                return 1;
        else if(nextbits(32)==0x110)
                return 1;
        else if(nextbits(32)==0x111)
                return 1;
        else if(nextbits(32)==0x112)
                return 1;
        else if(nextbits(32)==0x113)
                return 1;
        else if(nextbits(32)==0x114)
                return 1;
        else if(nextbits(32)==0x115)
                return 1;
        else
                return 0;
}

void macroblock_type(void)
{
        if(picture_coding_type==1)
        {
                if(nextkeyword(1))
                {
```

```
                macroblock_quant=0;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=1;
        }
        else if(nextkeyword(1))
        {
                macroblock_quant=1;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=1;
        }
        else
        {
                printf("Macroblock type 1 ERROR !!\n");
                stopchar=getch();
        }

}
else if(picture_coding_type==2)
{
        if(nextkeyword(1))
        {
                macroblock_quant=0;
                macroblock_motion_forward=1;
                macroblock_motion_backward=0;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
        else if(nextkeyword(1))
        {
                macroblock_quant=0;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
        else if(nextkeyword(1))
        {
                macroblock_quant=0;
                macroblock_motion_forward=1;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=0;
        }
        else if(nextkeyword(1))
```

```
        {
                if(nextkeyword(1))
                {
                        macroblock_quant=0;
                        macroblock_motion_forward=0;
                        macroblock_motion_backward=0;
                        macroblock_pattern=0;
                        macroblock_intra=1;
                }
                else
                {
                        macroblock_quant=1;
                        macroblock_motion_forward=1;
                        macroblock_motion_backward=0;
                        macroblock_pattern=1;
                        macroblock_intra=0;
                }
        }
        else if(nextkeyword(1))
        {
                macroblock_quant=1;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
        else if(nextkeyword(1))
        {
                macroblock_quant=1;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=1;
        }
        else
        {
                printf("Macroblock type 2 ERROR !!\n");
                stopchar=getch();
        }

}
else if(picture_coding_type==3)
{
        if(nextkeyword(1))
        {
                if(nextkeyword(1)==0)
                {
                        macroblock_quant=0;
```

```
                macroblock_motion_forward=1;
                macroblock_motion_backward=1;
                macroblock_pattern=0;
                macroblock_intra=0;
        }
        else
        {
                macroblock_quant=0;
                macroblock_motion_forward=1;
                macroblock_motion_backward=1;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1)==0)
        {
                macroblock_quant=0;
                macroblock_motion_forward=0;
                macroblock_motion_backward=1;
                macroblock_pattern=0;
                macroblock_intra=0;
        }
        else
        {
                macroblock_quant=0;
                macroblock_motion_forward=0;
                macroblock_motion_backward=1;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1)==0)
        {
                macroblock_quant=0;
                macroblock_motion_forward=1;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=0;
        }
        else
        {
                macroblock_quant=0;
                macroblock_motion_forward=1;
                macroblock_motion_backward=0;
```

```
                macroblock_pattern=1;
                macroblock_intra=0;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                macroblock_quant=0;
                macroblock_motion_forward=0;
                macroblock_motion_backward=0;
                macroblock_pattern=0;
                macroblock_intra=1;
        }
        else
        {
                macroblock_quant=1;
                macroblock_motion_forward=1;
                macroblock_motion_backward=1;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
}
else if(nextkeyword(1))
{
        if(nextkeyword(1))
        {
                macroblock_quant=1;
                macroblock_motion_forward=1;
                macroblock_motion_backward=0;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
        else
        {
                macroblock_quant=1;
                macroblock_motion_forward=0;
                macroblock_motion_backward=1;
                macroblock_pattern=1;
                macroblock_intra=0;
        }
}
else if(nextkeyword(1))
{
        macroblock_quant=1;
        macroblock_motion_forward=0;
        macroblock_motion_backward=0;
        macroblock_pattern=0;
```

```
                macroblock_intra=1;
            }
            else
            {
                printf("Macroblock type 3 ERROR !!\n");
                stopchar=getch();
            }
        }
        else
        {
            printf("Macroblock type ERROR !!\n");
            stopchar=getch();
        }
}

long nextkeyword(int a)
{
    int i,l;
  unsigned long k;
    k=0;
    for(i=0;i<a;i++)
    {
        l=buffer();
        k<<=1;
        k=k+l;
    }
    return (signed)k;
}

long nextbit(int a)
{
    int i,l;
  unsigned long k;
    k=0;
    for(i=0;i<a;i++)
    {
        l=buffer_code[i];
        k<<=1;
        k=k+l;
    }
```

```
        return (signed)k;
}


void next_start_code(void)
{
        while((mem_point%8)!=0)
                buffer();

        while(nextbits(24)!=1)
                nextkeyword(8);

}

void sub_next_start_code(void)
{
        while(bit_out!=9)
                sub_buffer();

        while(nextbit(24)!=1)
                sub_nextkeyword(8);

}


long nextbits(int a)
{
        int i,l,m,n;
  unsigned long k;
        k=0;
        m=read_mem;
        for(i=mem_point;i<(mem_point+a);i++)
        {
                if(i<bit_amout)
                        n=i;
                else
                {
                        if(i==bit_amout)
                        {
                                n=0;
                                if(read_mem==0)
                                        m=1;
                                else if(read_mem==1)
                                        m=2;
                                else
                                        m=0;
```

```
                }
                else
                        n++;
        }
        |=video_buffer[m][n];
        k<<=1;
        k=k+l;
    }
    return (signed)k;
}


char buffer(void)
{
    char out;

    out=video_buffer[read_mem][mem_point];
    mem_point++;



    if(mem_point==bit_amout)
    {
//          printf("\n\n  ************** This is start of PACKET ************\n");
            video_stream_flag=0;
            sub_pack();

            mem_point=0;
            if(read_mem==0)
                    read_mem=1;
            else if(read_mem==1)
                    read_mem=2;
            else
                    read_mem=0;

            bit_amout=buffer_length[read_mem]*8;

      }

      return out;

}


void Type_of_stream(void)
{
```

```
        int k,l;

//      printf("\nstream_id = %0x\n",stream_id);

        if(stream_id==188)
                {}
        else if(stream_id==189)
                {}
        else if(stream_id==190)
                {}
        else if(stream_id==191)
                {}
        else
        {
                l=stream_id;
                k=l>>5;
                if(k==6)
    {}
                else
                {
                        k=l>>4;
                        if(k==14)
                        {
                                video_stream_flag=1;
//                              printf("This is video stream number: %d\n",(l & 0x0f));
                        }
                        else
                        {
                                printf("Type of stream ERROR or RESERVE\n");
                                stopchar=getch();
                        }

                }
        }

}


void InitialBuffer(void)
{
  int i,j,k;
  unsigned ch;

  for(i=0;i<57;i+=8)
                {
                        ch=fgetc(mpgfile);
                        for(j=8+i;j>i;j--)
                        {
```

```
                                        k=ch&1;
                                        buffer_code[j-1]=k;
                                        ch>>=1;
                                }
    }
  bit_out = 1;

}


long sub_nextkeyword(int a)
{
        int i,l;
  unsigned long k;
        k=0;
        for(i=0;i<a;i++)
        {
                l=sub_buffer();
                k<<=1;
                k=k+l;
        }
        return (signed)k;
}



char sub_buffer(void)
{
        int i,j,k;
        char ch,out;

        out=buffer_code[0];
        for(i=0;i<63;i++)
        {
                buffer_code[i]=buffer_code[i+1];
        }
        bit_out++;

    if(bit_out==9)
        {
                ch=fgetc(mpgfile);
                for(i=64;i>56;i--)
                {
                        k=ch&1;
                        buffer_code[i-1]=k;
                        ch>>=1;
                }
                bit_out=1;
```

```
                byte_no++;
        }

        return out;

}

void sub_pack(void)
{

loop1:

        if((nextbit(24)==1)&&(nextbit(32)!=442))
                goto loop2;

        if(sub_nextkeyword(32)!=442)                                    //pack_start_code
        {
                printf("pack_start_code  ERROR !!!\n");
                stopchar=getch();
        }

        if(sub_nextkeyword(4)!=2)
        {
                printf("pack error\n");
                stopchar=getch();
        }

        sub_nextkeyword(3);
        sub_nextkeyword(1);
        sub_nextkeyword(15);
        sub_nextkeyword(1);
        sub_nextkeyword(15);
        sub_nextkeyword(1);
        sub_nextkeyword(1);
        sub_nextkeyword(22);
        sub_nextkeyword(1);

        if(nextbit(32)==443)
                system_header();

loop2:

        sub_packet();
```

```
        if(stream_jump_flag)
        {
                stream_jump_flag=0;
                goto loop1;
        }

}

void sub_packet(void)
{
        int i,j,k;
        char ch;

        if(sub_nextkeyword(24)!=1)
        {
                printf("packet_start_code_prefix  ERROR!! \n");
                stopchar=getch();
        }

        stream_id=sub_nextkeyword(8);
        packet_length=sub_nextkeyword(16);

        Type_of_stream();
//      printf("Packet length = %d\n\n",packet_length);

        k=byte_no;

        if(stream_id!=private_stream_2)
        {
                while(nextbit(8)==255)
                        sub_nextkeyword(8);
                if(nextbit(2)==1)
                {
                        sub_nextkeyword(2);
                        sub_nextkeyword(1);
                        sub_nextkeyword(13);
                }
                if(nextbit(4)==2)
                {
                        sub_nextkeyword(4);
                        sub_nextkeyword(3);
                        sub_nextkeyword(1);
                        sub_nextkeyword(15);
                        sub_nextkeyword(1);
                        sub_nextkeyword(15);
```

```
                sub_nextkeyword(1);
        }
        else if(nextbit(4)==3)
        {
                sub_nextkeyword(4);
                sub_nextkeyword(3);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                if(sub_nextkeyword(4)!=1)
                {
                        printf("Packet ERROR");
                        stopchar=getch();
                }
                sub_nextkeyword(3);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
                sub_nextkeyword(15);
                sub_nextkeyword(1);
         }
         else
         {
                 if(sub_nextkeyword(8)!=15)
                 {
                         printf("Packet ERROR !!");
                         stopchar=getch();
                 }

         }
}

k=byte_no-k;
packet_data_length=packet_length-k;
packet_data_byte=0;

if(video_stream_flag)
{
         for(i=0;i<packet_data_length;i++)
         {

                 if(i<8)
                 {
                         for(j=0;j<8;j++)
                                 video_buffer[write_mem][i*8+j]=buffer_code[i*8+j];
                 }
```

```
                else
                {
                        ch=fgetc(mpgfile);
    byte_no++;
                        for(j=0;j<8;j++)
                        {
                                k=ch&1;
                                video_buffer[write_mem][8*(i+1)-(j+1)]=k;
                                ch>>=1;
                        }
                }
        }
InitialBuffer();
        buffer_length[write_mem]=packet_data_length;
        if(write_mem==0)
                write_mem=1;
        else if(write_mem==1)
                write_mem=2;
        else
                write_mem=0;
    }
    else
    {
        stream_jump_flag=1;
        for(i=0;i<packet_data_length;i++)
                sub_nextkeyword(8);
    }
}
void dequantize_ipicture(int block_number)
{
    int i,j,k;
    if(block_number==0)
    {
        for(i=0;i<8;i++)
        {
                for(j=0;j<8;j++)
```

```
                {
                        k=zzscan[i][j];
                        block_out[i][j]=(2*data_out[k]*quantizer_scale*intra_quantizer_matrix[i][j])/16;

                        if((block_out[i][j]&1)==0)
                                block_out[i][j]=block_out[i][j]-Sign(block_out[i][j]);
                        if(block_out[i][j]>2047)
                                block_out[i][j]=2047;
                        if(block_out[i][j]<-2047)
                                block_out[i][j]=-2048;
                }
        }
        block_out[0][0]=data_out[0]*8;
        if((macroblock_address-past_intra_address)>1)
                block_out[0][0]=128*8+block_out[0][0];
        else
                block_out[0][0]=dct_dc_y_past+block_out[0][0];
        dct_dc_y_past=block_out[0][0];
}
else if((block_number>0)&&(block_number<4))
{
        for(i=0;i<8;i++)
        {
                for(j=0;j<8;j++)
                {
                        k=zzscan[i][j];
                        block_out[i][j]=(2*data_out[k]*quantizer_scale*intra_quantizer_matrix[i][j])/16;

                        if((block_out[i][j]&1)==0)
                                block_out[i][j]=block_out[i][j]-Sign(block_out[i][j]);
                        if(block_out[i][j]>2047)
                                block_out[i][j]=2047;
                        if(block_out[i][j]<-2048)
                                block_out[i][j]=-2048;
                }
        }
        block_out[0][0]=dct_dc_y_past+(data_out[0]*8);
        dct_dc_y_past=block_out[0][0];
}
else if(block_number==4)
{
        for(i=0;i<8;i++)
        {
                for(j=0;j<8;j++)
                {
                        k=zzscan[i][j];
                        block_out[i][j]=(2*data_out[k]*quantizer_scale*intra_quantizer_matrix[i][j])/16;
```

```
                                        if((block_out[i][j]&1)==0)
                                                block_out[i][j]=block_out[i][j]-Sign(block_out[i][j]);
                                        if(block_out[i][j]>2047)
                                                block_out[i][j]=2047;
                                        if(block_out[i][j]<-2047)
                                                block_out[i][j]=-2048;
                                }
                        }
                        block_out[0][0]=data_out[0]*8;
                        if((macroblock_address-past_intra_address)>1)
                                block_out[0][0]=128*8+block_out[0][0];
                        else
                                block_out[0][0]=dct_dc_cb_past+block_out[0][0];
                        dct_dc_cb_past=block_out[0][0];
                }
                else if(block_number==5)
                {
                        for(i=0;i<8;i++)
                        {
                                for(j=0;j<8;j++)
                                {
                                        k=zzscan[i][j];
                                        block_out[i][j]=(2*data_out[k]*quantizer_scale*intra_quantizer_matrix[i][j])/16;

                                        if((block_out[i][j]&1)==0)
                                                block_out[i][j]=block_out[i][j]-Sign(block_out[i][j]);
                                        if(block_out[i][j]>2047)
                                                block_out[i][j]=2047;
                                        if(block_out[i][j]<-2047)
                                                block_out[i][j]=-2048;
                                }
                        }
                        block_out[0][0]=data_out[0]*8;
                        if((macroblock_address-past_intra_address)>1)
                                block_out[0][0]=128*8+block_out[0][0];
                        else
                                block_out[0][0]=dct_dc_cr_past+block_out[0][0];
                        dct_dc_cr_past=block_out[0][0];
                }
}


void dequantize_nonipicture(int block_number)
{
        int m,n,i;

        for(m=0;m<8;m++)
        {
```

```
                for(n=0;n<8;n++)
                {
                        i=zzscan[m][n];
                        block_out[m][n]=((2*data_out[i]+Sign(data_out[i]))*quantizer_scale*non_intra_quantizer_matrix[m][n])/16;
                        if((block_out[m][n]&1)==0)
                                block_out[m][n]=block_out[m][n]-Sign(block_out[m][n]);
                        if(block_out[m][n]>2047)
                                block_out[m][n]=2047;
                        if(block_out[m][n]<-2048)
                                block_out[m][n]=-2048;
                        if(data_out[i]==0)
                                block_out[m][n]=0;
                }
        }

}


void block_search(int block_number)
{
        int i,j,k;
        int right_half_for,down_half_for;
        int right_half_back,down_half_back;
        int right_for,down_for;
        int right_back,down_back;
        int b_row,b_column,sub_sampling;
        int pel_for[8][8],pel_back[8][8];


        if(block_number<4)
        {
                right_for=recon_right_for>>1;
                down_for=recon_down_for>>1;
                right_half_for=recon_right_for-(2*right_for);
                down_half_for=recon_down_for-(2*down_for);

                right_back=recon_right_back>>1;
                down_back=recon_down_back>>1;
                right_half_back=recon_right_back-(2*right_back);
                down_half_back=recon_down_back-(2*down_back);

        }
        else
        {
                right_for=(recon_right_for/2)>>1;
                down_for=(recon_down_for/2)>>1;
                right_half_for=recon_right_for/2-(2*right_for);
```

```
            down_half_for=recon_down_for-(2*down_for);

            right_back=(recon_right_back/2)>>1;
            down_back=(recon_down_back/2)>>1;
            right_half_back=recon_right_back/2-(2*right_back);
            down_half_back=recon_down_back/2-(2*down_back);
}


sub_sampling=16;

if(block_number<4)
            k=0;
else if(block_number==4)
            k=1;
else
            k=2;


if(block_number==0)
{
            b_row=0;
            b_column=0;
}
else if(block_number==1)
{
            b_row=1;
            b_column=0;
}
else if(block_number==2)
{
            b_row=0;
            b_column=1;
}
else if(block_number==3)
{
            b_row=1;
            b_column=1;
}
else
{
            b_row=0;
            b_column=0;
            sub_sampling=8;
}


for(i=0;i<8;i++)
```

```
{
    for(j=0;j<8;j++)
    {
        if((!right_half_for)&&(!down_half_for))
            pel_for[i][j]=past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for]
[macroblock_number*sub_sampling+b_row*8+j+right_for];

        if((!right_half_for)&&(down_half_for))
            pel_for[i][j]=floor((past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for]
[macroblock_number*sub_sampling+b_row*8+j+right_for]+past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for+1][macroblock_numb
er*sub_sampling
+b_row*8+j+right_for])/2+0.5);

        if((right_half_for)&&(!down_half_for))
            pel_for[i][j]=floor((past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for]
[macroblock_number*sub_sampling+b_row*8+j+right_for]+past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for][macroblock_number*
sub_sampling+
            b_row*8+j+right_for+1])/2+0.5);

        if((right_half_for)&&(down_half_for))
            pel_for[i][j]=floor((past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for]
[macroblock_number*sub_sampling+b_row*8+j+right_for]+past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for+1][macroblock_numb
er*sub_sampling
+b_row*8+j+right_for]+past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for][macroblock_number*sub_sampling+b_row*8+j+right_for
+1]+past_picture[k][slice_number*sub_sampling+b_column*8+i+down_for+1][macroblock_number*sub_sampling+b_row*8+j+
right_for+1])/4+0.5);
    }
}


if(picture_coding_type==3)
{
    for(i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
        {
            if((!right_half_back)&&(!down_half_back))

pel_back[i][j]=next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*sub_sampling+b_row*8+j+rig
ht_back];

            if((!right_half_back)&&(down_half_back))

pel_back[i][j]=floor((next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*sub_sampling+b_row*8
+j+right_back]+next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back+1][macroblock_number*
sub_sampling+b_row*8+j+right_back])/2+0.5);
```

```
                                        if((right_half_back)&&(!down_half_back))

            pel_back[i][j]=floor((next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*sub_sampling+b_row*8
+j+right_back]+next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*
sub_sampling+b_row*8+j+right_back+1])/2+0.5);

                                        if((right_half_back)&&(down_half_back))

            pel_back[i][j]=floor((next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*sub_sampling+b_row*8
+j+right_back]+next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back+1][macroblock_number*
sub_sampling+b_row*8+j+right_back]+next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back][macroblock_number*sub_sampling+b_r
ow*8+j+right_back+1]+next_picture[k][slice_number*sub_sampling+b_column*8+i+down_back+1][macroblock_number*
sub_sampling+b_row*8+j+right_back+1])/4+0.5);

                                }
                        }

            }


            if(picture_coding_type==2)
            {
                        for(i=0;i<8;i++)
                        {
                                    for(j=0;j<8;j++)
                                                pel[i][j]=pel_for[i][j];
                        }

            }
            else
            {
                        if((macroblock_motion_forward)&&(macroblock_motion_backward))
                        {
                                    for(i=0;i<8;i++)
                                    {
                                                for(j=0;j<8;j++)
                                                            pel[i][j]=floor((pel_for[i][j]+pel_back[i][j])/2+0.5);
                                    }
                        }
                        else if(macroblock_motion_forward)
                        {
                                    for(i=0;i<8;i++)
                                    {
                                                for(j=0;j<8;j++)
                                                            pel[i][j]=pel_for[i][j];
                                    }
                        }
                        else
```

```
                {
                        for(i=0;i<8;i++)
                        {
                                for(j=0;j<8;j++)
                                        pel[i][j]=pel_back[i][j];
                        }
                }
        }

}


void motion_rebuild(void)
{

        int complement_horizontal_forward_r,complement_vertical_forward_r;
        int complement_horizontal_backward_r,complement_vertical_backward_r;
        int right_little,right_big,down_little,down_big;
        int max,min;
        int new_vector;



        if(macroblock_motion_forward)
        {
                if((forward_f==1)||(motion_horizontal_forward_code==0))
                        complement_horizontal_forward_r=0;
                else
                        complement_horizontal_forward_r=forward_f-1-motion_horizontal_forward_r;

                if((forward_f==1)||(motion_vertical_forward_code==0))
                        complement_vertical_forward_r=0;
                else
                        complement_vertical_forward_r=forward_f-1-motion_vertical_forward_r;

                right_little=motion_horizontal_forward_code*forward_f;

                if(right_little==0)
                        right_big=0;
                else
                {
                        if(right_little>0)
                        {
                                right_little=right_little-complement_horizontal_forward_r;
                                right_big=right_little-(32*forward_f);
                        }
                        else
```

```
                {
                        right_little=right_little+complement_horizontal_forward_r;
                        right_big=right_little+(32*forward_f);
                }
        }

        down_little=motion_vertical_forward_code*forward_f;
        if(down_little==0)
                down_big=0;
        else
        {
                if(down_little>0)
                {
                        down_little=down_little-complement_vertical_forward_r;
                        down_big=down_little-(32*forward_f);
                }
                else
                {
                        down_little=down_little+complement_vertical_forward_r;
                        down_big=down_little+(32*forward_f);
                }
        }


        max=(16*forward_f)-1;
        min=(-16*forward_f);

        new_vector=recon_right_for_prev+right_little;
        if((new_vector<=max)&&(new_vector>=min))
                recon_right_for=recon_right_for_prev+right_little;
        else
                recon_right_for=recon_right_for_prev+right_big;
        recon_right_for_prev=recon_right_for;
        if(full_pel_forward_vector)
                recon_right_for=recon_right_for<<1;

        new_vector=recon_down_for_prev+down_little;
        if((new_vector<=max)&&(new_vector>=min))
                recon_down_for=recon_down_for_prev+down_little;
        else
                recon_down_for=recon_down_for_prev+down_big;
        recon_down_for_prev=recon_down_for;
        if(full_pel_forward_vector)
                recon_down_for=recon_down_for<<1;
}
else
{
        if(picture_coding_type==2)
```

```
                down_big=0;
        else
        {
                if(down_little>0)
                {
                        down_little=down_little-complement_vertical_backward_r;
                        down_big=down_little-(32*backward_f);
                }
                else
                {
                        down_little=down_little+complement_vertical_backward_r;
                        down_big=down_little+(32*backward_f);
                }
        }

        max=(16*backward_f)-1;
        min=(-16*backward_f);

        new_vector=recon_right_back_prev+right_little;
        if((new_vector<=max)&&(new_vector>=min))
                recon_right_back=recon_right_back_prev+right_little;
        else
                recon_right_back=recon_right_back_prev+right_big;

        recon_right_back_prev=recon_right_back;
        if(full_pel_backward_vector)
                recon_right_back=recon_right_back<<1;

        new_vector=recon_down_back_prev+down_little;
        if((new_vector<=max)&&(new_vector>=min))
                recon_down_back=recon_down_back_prev+down_little;
        else
                recon_down_back=recon_down_back_prev+down_big;
        recon_down_back_prev=recon_down_back;
        if(full_pel_backward_vector)
                recon_down_back=recon_down_back<<1;

}
else
{
        recon_right_back=recon_right_back_prev;
        recon_down_back=recon_down_back_prev;
}

}
```

```
void keep_picture(void)
{
        int i,j,k,l,m,n;
        int         cb_macroblock[22][16][16],cr_macroblock[22][16][16];
        int y_slice_block[16][352],cb_slice_block[16][352],cr_slice_block[16][352];
        int r_slice_block[16][353],g_slice_block[16][352],b_slice_block[16][352];
        char ch;

        for(i=0;i<15;i++)
        {
                for(l=0;l<2;l++)
                {
                        for(j=0;j<22;j++)
                        {
                                for(k=0;k<2;k++)
                                {
                                        for(m=0;m<8;m++)
                                        {
                                                for(n=0;n<8;n++)
                                                        y_slice_block[m+l*8][n+k*8+j*16]=picture_out[i][j][k+l*2]
[m][n];
                                        }
                                }
                        }

                        for(j=0;j<22;j++)
                        {
                                for(m=0;m<8;m++)
                                {
                                        for(n=0;n<8;n++)
                                        {
                                                cb_macroblock[j][m*2][n*2]=picture_out[i][j][4][m][n];
                                                cb_macroblock[j][m*2][n*2+1]=picture_out[i][j][4][m][n];
                                                cb_macroblock[j][m*2+1][n*2]=picture_out[i][j][4][m][n];
                                                cb_macroblock[j][m*2+1][n*2+1]=picture_out[i][j][4][m][n];

                                                cr_macroblock[j][m*2][n*2]=picture_out[i][j][5][m][n];
                                                cr_macroblock[j][m*2][n*2+1]=picture_out[i][j][5][m][n];
                                                cr_macroblock[j][m*2+1][n*2]=picture_out[i][j][5][m][n];
                                                cr_macroblock[j][m*2+1][n*2+1]=picture_out[i][j][5][m][n];
                                        }
                                }
                        }
```

```
for(j=0;j<22;j++)
{
        for(m=0;m<16;m++)
        {
                for(n=0;n<16;n++)
                {
                        cb_slice_block[m][n+j*16]=cb_macroblock[j][m][n]-128;
                        cr_slice_block[m][n+j*16]=cr_macroblock[j][m][n]-128;
                }

        }
}


for(m=0;m<16;m++)
{
        for(n=0;n<352;n++)
        {
                k=floor((16*i+m)/2);
                l=floor(n/2);
                recent_picture[0][16*i+m][n]=y_slice_block[m][n];
                recent_picture[1][k][l]=cb_slice_block[m][n]+128;
                recent_picture[2][k][l]=cr_slice_block[m][n]+128;
        }
}


for(m=0;m<16;m++)
{
        for(n=0;n<352;n++)
        {
                r_slice_block[m][n]=y_slice_block[m][n]+cr_slice_block[m][n]*1.402;
                g_slice_block[m][n]=y_slice_block[m][n]+cb_slice_block[m][n]*(-0.344)+cr_slice_block[m][n]*(-
0.714);
                b_slice_block[m][n]=y_slice_block[m][n]+cb_slice_block[m][n]*1.772;

                if(r_slice_block[m][n]>255)
                        r_slice_block[m][n]=255;
                if(g_slice_block[m][n]>255)
                        g_slice_block[m][n]=255;
                if(b_slice_block[m][n]>255)
                        b_slice_block[m][n]=255;
                if(r_slice_block[m][n]<0)
                        r_slice_block[m][n]=0;
```

```
                                        if(g_slice_block[m][n]<0)
                                                g_slice_block[m][n]=0;
                                        if(b_slice_block[m][n]<0)
                                                b_slice_block[m][n]=0;

                                }
                        }
//                      if(bpicture_number==8)
//                      {
                                for(m=0;m<16;m++)
                                {
                                        for(n=0;n<352;n++)
                                        {

                                                ch=r_slice_block[m][n];
                                                fputc(ch,testfile);

                                                ch=g_slice_block[m][n];
                                                fputc(ch,testfile);

                                                ch=b_slice_block[m][n];
                                                fputc(ch,testfile);

                                        }
                                }

                }

        /*if(bpicture_number==80)
          {
                fclose(testfile);
           printf(" close file !!!! ");
           getch();
           } */
        }


int shift_right(int a,int b)
{
                int i,j,k,l,code[8];

                i=0;j=0;k=0;l=0;
                for (i=0;i<8;i++)
                {
                        j=a&1;
```

```
                code[i]=j;
                a>>=1;
        }

        for(i=b;i>0;i--)
        {
                j=code[i-1];
                k<<=1;
                k=k+j;
        }
        return k;
}


int shift_left(int a,int b)
{
        int i,j,k,l,code[8];

        i=0;j=0;k=0;l=0;
        for (i=0;i<8;i++)
        {
                j=a&1;
                code[i]=j;
                a>>=1;
        }

        for(i=0;i<b;i++)
        {
                j=code[7-i];
                k<<=1;
                k=k+j;
        }
        return k;
}



 /* metrix IDCT */

/**********************************
*    Initialize          *
**********************************/

void Initialize()
{
   int i,j;
```

```
  for ( j=0 ; j<8 ; j++ )
  {
   C[0][j] = 1.0/sqrt(8);
   Ct[j][0] = C[0][j];
  }

  for ( i=1 ; i<8 ; i++ )
  {
   for ( j=0 ; j<8 ; j++ )
   {
    C[i][j] = sqrt(0.25)*cos((2*j+1)*i*PI/16);
    Ct[j][i] = C[i][j];
   }
  }


}



/*******************************
*   Matrix InverseDCT      *
*******************************/

void  InverseDCT(int input[8][8], int *output[8][8])

{
   double temp[8][8];
   double temp1;
   int i;
   int j;
   int k;


   /* MatrixMultiple( temp , input , C ); */

   for ( i=0; i<8; i++ )
   {
    for ( j=0; j<8; j++ )
    {
     temp[i][j]= 0.0;
     for (k=0; k<8 ; k++)
      temp[i][j] += input[i][k] * C[k][j];
    }
   }
```

```
/* MatrixMultiply( output , Ct , temp ) */

for ( i=0 ; i<8 ; i++)
{
 for (j=0 ;j<8 ; j++)
 {
  temp1 = 0.0;
  for (k=0 ; k<8 ; k++)
   temp1 += Ct[i][k] * temp [k][j];
   output[i][j] = (int)floor( temp1+ 0.5);

 }
}
}
```

โปรแกรมแสดงภาพ

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <malloc.h>
#include <math.h>

#define Line 23
#define BUFSIZE 8192
#define Loop 3

int ScrnWd =640;
int Row,Column,PicFm,ScrnSeg =0;
unsigned char Temp[2];
FILE *Rf;
unsigned int Data;
unsigned int *Color;
char FileName[60];
long FileSize;
int Frames,PicWd,PicHt,i;

void PutPixel(int X, int Y);
void SetDisplay(void);


main()
{
   char * FileBuf;

   clrscr();
   printf("Enter 16 BITS .RAW File name > ");
   scanf("%s",FileName);
   if ((Rf=fopen(FileName,"rb"))==NULL)
     {
      printf(" Can't open MPEG file !!");
      exit(1);
     }

   FileBuf = (char *)malloc(BUFSIZE);
   setvbuf(Rf,FileBuf,_IOFBF,BUFSIZE);

   fseek(Rf,0,SEEK_END);
   FileSize = ftell(Rf);

   fseek(Rf,8,SEEK_SET);
   fread(&Temp[0],2,1,Rf);
   PicWd = (Temp[0]<<8) | Temp[1];
            fread(&Temp[0],2,1,Rf);
            PicHt = (Temp[0]<<8) | Temp[1];
   clrscr();
            printf(" Picture Width : %d \n Picture Height : %d \n",PicWd,PicHt);
```

```
printf(" size : %ld \n",FileSize);

Frames = (int)((FileSize-32)/((long)PicWd*(long)PicHt*2));
printf(" Frames : %d \n",Frames);
getch();

fseek(Rf,32,SEEK_SET);
SetDisplay();

if((Color=(unsigned int *)malloc(PicWd*PicHt*2))==NULL)
  exit(1);

for (i=0; i<Loop; i++)
{

fseek(Rf,32,SEEK_SET);
for  (PicFm=0 ;PicFm< Frames ;PicFm++)
  {
   for (Row=0;Row< PicHt ; Row++)
   {
    if (( Row % Line )== 0 )
      if (( Row + Line) > PicHt)
        fread(Color,PicWd*(PicHt-Row)*2,1,Rf);
      else
        fread(Color,PicWd*Line*2,1,Rf);

    for  (Column=0;Column< PicWd ; Column++)
      PutPixel(Column,Row);
   }
  }
}
  getch();
  free(Color);
  free(FileBuf);
  fclose(Rf);
 return;
}


void SetDisplay(void)
{
union REGS Regs;
   Regs.x.ax = 0x4f02;
            Regs.x.bx = 0x0111;
            int86(0x10,&Regs,&Regs);
}


void PutPixel(int X, int Y)
{
            unsigned long Pixel;
```

```
        unsigned char WriteSeg;
        unsigned int far *ScreenPtr;
union REGS Regs;

        Pixel = ( X + Y*(unsigned long)ScrnWd )*2;
        WriteSeg = Pixel / 65536L;
        if (WriteSeg != ScrnSeg)
        {
  Regs.x.ax = 0x4f05;
  Regs.x.bx = 0x0000;
         Regs.x.dx = WriteSeg;
         int86(0x10,&Regs,&Regs);
         ScrnSeg = WriteSeg;
        }
        FP_SEG(ScreenPtr) = 0xA000;
        FP_OFF(ScreenPtr) = (unsigned int)(Pixel & 0xFFFF);
        *ScreenPtr = *Color;
  Color--;
}
```