โปรแกรมแปลงและแสดงผลภาพเคลื่อนไหว

Conversion and Playback Program for Motion Video

โดย

นายกฤษดิพล ศรีสารคาม

นางสาวปรารถนา โล่ห์อำนวยกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

โปรแกรมแปลงและแสดงผลภาพเคลื่อนไหว

Conversion and Playback Program for Motion Video

โดย

นายกฤษติพล ศรีสารคาม 35104015

นางสาวปรารถนา โล่ห์อำนวยกุล 35104255

อาจารย์ที่ปรึกษา

ผศ.ดร.สุวิพล สิทธิชีวภาค

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538.

ปริญญานิพนธ์ปีการศึกษา 2538

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมแปลงและแสดงผลภาพเคลื่อนไหว

Conversion and Playback Program for Motion Video

ผู้จัดทำ

1. นายกฤษดิพล ศรีสารคาม 35104015
2. นางสาวปรารถนา โล่ห์อำนวยกุล 35104255

สุวิพล

(ผศ.ดร. สุวิพล สิทธิชีวภาค) อาจารย์ที่ปรึกษา

โปรแกรมแปลงและแสดงผลภาพเคลื่อนไหว

Conversion and Playback Program for Motion Video

โดย 1. นายกฤษติพล ศรีสารคาม 35104015
2. นางสาวปรารถนา โล่ห์อำนวยกุล 35104255

อาจารย์ที่ปรึกษา ผศ. ดร. สุวิพล สิทธิชีวภาค

บทคัดย่อ

โครงงานนี้จะศึกษาในส่วนของภาพเคลื่อนไหวโดยใช้รูปแบบ AVI ซึ่งเป็นรูปแบบของ Microsoft Video for Windows รูปแบบ AVI นั้นมีลักษณะการเก็บข้อมูลค่อนข้างซับซ้อน จึงนำมาแปลงให้อยู่ในรูปแบบอย่างง่าย - รูปแบบ VID โดยจัดเก็บให้มี header และ frame format ง่ายขึ้นและน้อยลง AVI ที่ศึกษามีรูปแบบเป็น 8 บิต color index ไม่มีส่วนของข้อมูลเสียง และได้เขียนโปรมแกรมเพื่อแสดงผลของรูปแบบ VID ที่ได้

ABSTRACT

This project studies about motion video in Audio Video Interleave (AVI) file format which is format of Microsoft Video for Windows. AVI format has the complicated data storage structures, so the conversion to empirical VIDEO (VID) file format is more appreciative. The data storage structures of VID format are easier and length shorter header and frame format than AVI format. AVI format is used in this project with 8 bit color index format with no audio data. In programming, the AVI to VID and VID VIEWER program are presented.

สารบัญ

กิตติกรรมประกาศ

เอกสารอ้างอิง

สารบัญรูปภาพ

หน้า

บทที่ 1

บทนำ

การนำเครื่องคอมพิวเตอร์มาช่วยในการสื่อสารข้อมูลเป็นเรื่องที่คุ้นเคยกันมากขึ้นในสังคมปัจจุบัน และก็คงจะเป็นที่ทราบกันอยู่แล้วว่าการที่ระบบคอมพิวเตอร์จะทำให้ผลงานมีคุณภาพและประสิทธิภาพดีขึ้น ได้มากน้อยเพียงใด ก็ขึ้นอยู่กับปัจจัยหลัก 2 ประการคือ ความเหมาะสมทางด้านฮาร์ดแวร์และซอฟต์แวร์ที่นำมาประยุกต์กับงานนั้น ๆ

การสื่อสารข้อมูลในปัจจุบัน มีลักษณะเป็นการสื่อสารข้อมูลแบบมัลติมีเดียมากขึ้น คือมีการส่งข้อมูลได้ทั้งทางด้านข้อความ,รูปภาพ,เสียงและภาพเคลื่อนไหว ในส่วนของภาพเคลื่อนไหวนั้นมีอยู่หลายรูปแบบ เช่น AVI, MPEG, MOV เป็นต้น

โครงงานนี้จะศึกษาในส่วนของภาพเคลื่อนไหวโดยใช้รูปแบบ AVI เป็นรูปแบบของ Microsoft Video for Windows ซึ่งเป็นที่รู้จักโดยทั่วไปและมีการใช้งานอย่างกว้างขวาง รูปแบบ AVI นั้นมีลักษณะการเก็บข้อมูลค่อนข้างซับซ้อน จึงนำมาแปลงให้อยู่ในรูปแบบอย่างง่ายก่อน โดยจัดเก็บให้มี header และ frame format ง่ายขึ้นและน้อยลง เพื่อให้สามารถจัดการกับการเข้าถึงข้อมูลได้ง่ายขึ้น รูปแบบอย่างง่ายนี้ใช้ชื่อว่า VID ย่อมาจาก Video

AVI ที่นำมาใช้ต้องมีรูปแบบเป็น 8 บิต color index 256 สี และไม่มีส่วนของข้อมูลเสียง โดยมีขนาดข้อมูล(resolution)เป็น 160x120 พิกเซล ส่วน VID มีรูปแบบเป็น 8 บิต 256 สี และโปรแกรมที่ใช้แสดงผลรูปแบบ VID นั้น สามารถเล่นไปข้างหน้า (play, fast forward) และเริ่มกลับมาเล่นใหม่ (rewind) ได้

บทที่ 2

AVI file format

AVI ย่อมาจาก Audio Video Interleave AVI file ใช้รูปแบบของ RIFF รูปแบบของ RIFF เป็นลักษณะการใช้ในรูปของ chunk chunk นั้นจะเป็นการแบ่งส่วน ไฟล์ โดยใช้ chunk header ข้อมูลทั้งหมดใน video clip จะอยู่ในส่วนของ RIFF "avi" chunk แต่ละ chunk จะนำด้วย 8 หรือ 12 ไบต์ มีลักษณะดังนี้ : 4 byte แรกจะบรรจุอักขระ ASCII 4 ตัว อักษรเหล่านี้ คือ "RIFF" สำหรับ chunk หลัก ตามด้วย 4 ไบต์ integer เป็นค่าแสดงความยาวของ chunk อักขระตัวอื่นอีก 4 ตัวที่อาจจะมีตามมา ใช้ในการแยกประเภทของ chunk ตัวอย่างเช่น RIFF AVI ส่วนแสดงลักษณะของ chunk เป็นรูปแบบของอักขระ 4 ไบต์ "RIFF" ตามด้วยความยาวของ chunk 4 ไบต์ ตามด้วยอักษร "AVI " RIFF AVI chunk จะประกอบไปด้วย หลาย ๆ "LIST" chunk และอาจจะตามด้วย index chunk ซึ่งใช้ " idx1" เป็นตัวแสดง โครงสร้างพื้นฐานแสดงดังข้างล่าง

RIFF AVI File Structure

```
"RIFF" <length> "AVI "
        "LIST" <length> "hdrl"
                "avih" <length>
                <Main AVI Header Data>
                "LIST" <length> "strl"
                        "strh" <length>
                        <Stream Header Data>
                        "strf" <length>
                        <Stream Format Information>
                        "strd" <length>
                        <Optional Extra Stream Data>
                        "strn" <length>
                        <Stream Name>
                "LIST" <length> "strl" (There may be any number of streams)
                        "strh" <length>
```

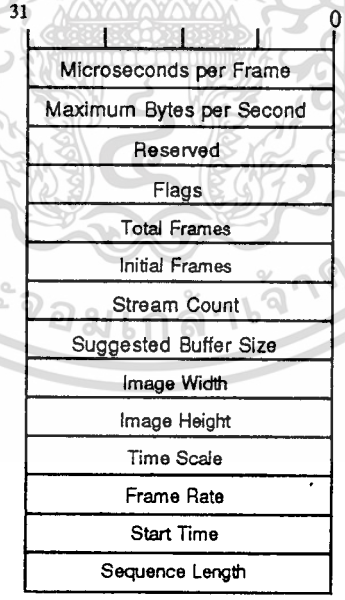
<Stream Header Data>
"strf" <length>
<Stream Format Information>
"strd" <length>
<Optional Extra Stream Data>
"strn" <length>
<Stream Name>
"LIST" <length> "movi"
{ Subchunk "LIST" <length> "rec "
"##dc" <length>
<Compressed Frame Data>
"##db" <length>
<Uncompressed Frame Data>
"##wb" <length>
<Audio Waveform Data>
"##pc" <length>
<Palette Data>
}
"##dc" <length>
<Compressed Frame Data>
"##db" <length>
<Uncompressed Frame Data>
"##wb" <length>
<Audio Waveform Data>
"##pc" <length>
<Palette Data>
"idx1" <length> (optional index data chunk)
<Index Data>

2.1 **Main AVI Header Chunk**

chunk แรก จะเป็น "hdrl" ประกอบด้วยข้อมูล header ของไฟล์ subchunk แรก ใน header เป็น "avih" chunk ซึ่งมีโครงสร้างข้อมูลบรรจุด้วย 14 ค่า double-word ดังนี้

```
typedef struct {
        DWORD       dwMicroSecPerFrame;
        DWORD       dwMaxBytesPerSec;
        DWORD       dwReserved1;
        DWORD       dwFlags;
        DWORD       dwTotalFrames;
        DWORD       dwInitialFrames;
        DWORD       dwStreams;
        DWORD       dwSuggestedBufferSize;
        DWORD       dwWidth;
        DWORD       dwHeight;
        DWORD       dwReserved[4];
} MainAVIHeader
```

โครงสร้างนี้แสดงในรูปที่ 2.1



รูปที่ 2.1 โครงสร้างของ Main AVI file header data organization.

Microseconds per frame

แสดงช่วงเวลาระหว่าง image frame ซึ่งเป็นค่าของ frame rate

Maximum bytes per second

แสดงอัตราข้อมูลที่จะรองรับการทำงานของ Video sequence ที่ full speed

flag field

มีส่วนประกอบ 4 ส่วนคือ

AVIF_HASINDEX	แสดงไฟล์ AVI มี "idx1" chunk ซึ่งมี Index อยู่ท้ายไฟล์ ใช้ในการดูลำดับของเฟรมในไฟล์
AVIF_MUSTUSEINDEX	แสดงว่าต้องใช้ Index ในการพิจารณาลำดับการแสดงข้อมูล
AVIF_ISINTERLEAVED	แสดงว่าไฟล์ AVI มีการแทรกไฟล์ระหว่าง audio และ video
AVIF_COPYRIGHTED	แสดงว่าไฟล์ AVI มีข้อมูลที่เป็นลิขสิทธ์ ถ้ามีการใช้แฟลกนี้ จะไม่อนุญาติให้ทำการสำเนาข้อมูล

Total frames

แสดงจำนวน frames ทั้งหมดที่มีอยู่ในไฟล์

Initial frame

รูปแบบจำนวนข้อมูล audio frame จะใช้ buffer ก่อนเริ่มเล่น Video sequence ให้สอดคล้องกับ audio sequence เพื่อเป็นการแน่ใจที่ว่า audio hardware มีข้อมูลอยู่ ขณะที่ video driver พยายามเริ่มทำงานทีหลัง

Stream

แสดงจำนวน data stream ในไฟล์ เช่น ไฟล์ จะมีทั้งข้อมูล audio และ video ซึ่งเกิดจาก stream ทั้ง 2 เป็น streams ที่แทรกอยู่ในไฟล์

Suggested buffer size

แสดงขนาดบัฟเฟอร์ว่าควร มีขนาดเท่าใด เพื่อให้แน่ใจว่าไม่จำเป็นต้องจองหน่วยความจำในระหว่างการเล่นกลับ ซึ่งจะทำให้ประสิทธิภาพการทำงานลดลง

Image hight และ width

กำหนดขนาดของภาพ ในหน่วย พิกเซล

Scale และ rate

เป็นการแสดง sample per second data rate ของไฟล์ ซึ่ง rate เป็นจำนวนของ sample และ scale ในหน่วยวินาที

Start และ length

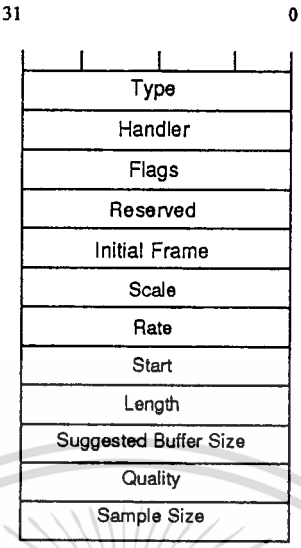
แสดงเวลาเริ่มต้น และความยาวของไฟล์ใน หน่วยที่กำหนดโดย rate และ scale

2.2 Stream Header

ต่อจาก header ไฟล์หลัก จะเป็นข้อมูลของ stream header ซึ่งแสดงด้วย LIST "strl" ทุกๆ stream header ประกอบด้วย 2 ส่วน คือ stream header chunk แสดงด้วย "strh" และ stream format chunk แสดงด้วย "strf" "strh" chunk มีโครงสร้างข้อมูล ประกอบด้วย 12 ค่า double word ดังนี้

```
typedef  struct {
        FOURCC      fccType;
        FOURCC      fccHandler;
        DWORD       dwFlags;
        DWORD       dwPriority;
        DWORD       dwInitialFrames;
        DWORD       dwScale;
        DWORD       dwRate;
        DWORD       dwStart;
        DWORD       dwLength;
        DWORD       dwSuggestedBufferSize;
        DWORD       dwQuality;
        DWORD       dwSampleSize;
} AVIStreamHeader;
```

โครงสร้างนี้แสดงในรูปที่ 2.2



รูปที่ 2.2 โครงสร้าง stream format header data

type เป็น 4 character code แสดงประเภทของ data stream จะใช้ "vids" เมื่อเป็น video stream, "auds" เมื่อเป็น audio stream หรือ "txts" เมื่อเป็น text stream

handler เป็น 4 character code ซึ่งกำหนดการ compress หรือ decompress ที่ใช้กับข้อมูล

quality แสดงคุณภาพของข้อมูล

sample size แสดงขนาดของ sample

ส่วนอื่น ๆ ที่มีชื่อเหมือนกับใน main file header จะมีความหมายเหมือนกัน โดยที่ข้อมูลใน main header ใช้กับ file ทั้งหมด ข้อมูลใน stream header จะใช้กับ stream เท่านั้น

"strf" chunk จะอยู่หลังจาก "strh" chunk และจะบอกขนาดและรูปแบบของ "strf" chunk โดยขึ้นกับชนิดของ stream ที่กล่าวถึง สำหรับ video stream ข้อมูลในส่วนนี้ประกอบด้วย โครงสร้าง BITMAPINFO

"strl" chunk อาจจะมี additional stream data chunk แสดงด้วย "strd" โดย chunk นี้จะบรรจุข้อมูลของ compression หรือ decompression driver ที่ใช้กับ stream และผ่านไปยัง driver โดยใช้ memory block ส่วน stream name chunk "strn" จะใช้ข้อความที่ปิดท้ายด้วยศูนย์ (zero terminated) อธิบาย stream โดยที่จะใช้ stream ใด ก็สามารถจัดการได้โดยใช้ชื่อของ stream นั้น

2.3 LIST "movi" chunk

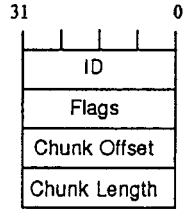
LIST "hdrl" chunk ตามด้วย LIST "movi" chunk ซึ่งประกอบด้วย video frame frame เดี่ยว ทุก ๆ frame จะมี 4 อักขระ 2 อักขระแรกนี้เป็น การเรียง stream ที่ถูกกำหนดใน LIST "hdrl" header chunk "00" คือ stream แรก "01" เป็น stream ที่สอง, 2 อักขระสุดท้ายเป็นเหมือน stream chunk, format ของข้อมูลใน chunk มีหลายรูปแบบ คือ ส่วน audio wave form chunk มี 2 อักขระ เป็น "wb", ส่วน uncompressed device independent bitmap (DIB) มี 2 อักขระเป็น "db", ถ้า frame data ถูก compress 2 อักขระนี้จะเป็น "dc" และถ้าข้อมูล เป็น palette 2 อักขระนี้จะเป็น "pc"

2.4 Index "idx1" chunk

ส่วนสุดท้ายที่ต่อจาก "movi" chunk เป็น option ของ "idx1" chunk มีความยาว 16 ไบต์ เป็น index ที่นำมาอ้างอิง video frame ใน LIST "movi" chunk โครงสร้างประกอบด้วยค่า 4 double word ดังนี้

```
typedef struct {
        DWORD       ckid;
        DWORD       dwFlags;
        DWORD       dwChunkOffset;
        DWORD       dwChunkLength;
} AVIINDEXENTRY;
```

โครงสร้างนี้แสดงในรูปที่ 2.3



รูปที่ 2.3 AVI file index chunk data format

ID field แสดงลักษณะสำคัญของ chunk

chunk offset field กำหนดตำแหน่งของ chunk ให้สอดคล้องกับ LIST "movi" chunk

chunk length field กำหนดความยาวของ video frame chunk ลบด้วย 8 bytes ที่ใช้สำหรับแสดงลักษณะของ RIFF

สำหรับการแทรกไฟล์ (คือ ไฟล์ ประกอบด้วย audio และ video streams) audio และ video frame แต่ละเฟรมจะถูกรวมเข้าไปใน record chunk ใช้ส่วนของ "rec" แสดงการ synchronize ระหว่าง video กับ audio

2.5 Data Chunk อื่น ๆ

ถ้าต้องการจัดเรียงข้อมูลในไฟล์ AVI สามารถจะเพิ่ม 'JUNK' chunk เข้าไปได้ (chunk นี้เป็นมาตรฐานของ RIFF type) โดยเวลาอ่าน chunk นี้จะข้ามข้อมูลใน chunk นี้ไป 'JUNK' chunk มีรูปแบบดังนี้

AVI Padding 'JUNK'
Byte data[]

เช่นเดียวกับ RIFF file อื่น ๆ ขณะทำการอ่านไฟล์ AVI จะทำการข้าม non-AVI chunk ที่ไม่รู้จักไปได้ ซึ่งจะทำการอ่านและเขียนไฟล์ AVI โดยจะเก็บเป็น non-AVI chunk

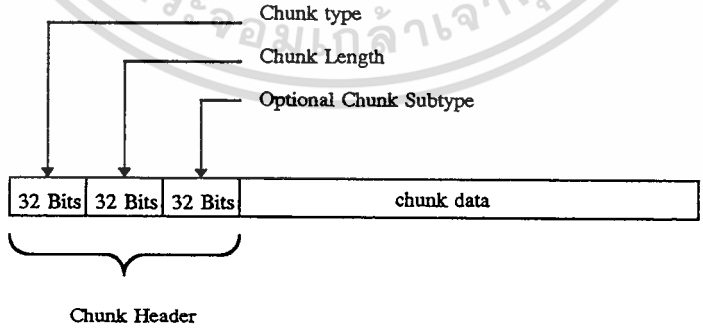
บทที่ 3

Motion Video

จุดประสงค์ของโครงงาน นี้คือการ process video บน PC, video sequences ทั้งหมดจะใช้งานจากโครงงาน นี้ เป็นการจับภาพ โดยใช้ Video for Windows ซึ่งมีแต่ข้อมูลภาพ ไม่มีข้อมูลเสียง โปรแกรมที่ใช้ทำการแปลงข้อมูลจาก Video for Windows format เป็นรูปแบบอย่างง่าย ซึ่งรูปแบบนี้ สามารถทำการ compress ได้ แต่โปรแกรมแปลงจะไม่สามารถแปลง compressed Video for Windows file ได้โดยตรง แต่ก็ยังมี compression formats ที่สามารถทำได้โดยการใช้ compression หรือ decompression driver ที่เหมาะสม, uncompressed file จะใช้ในรูปแบบของ 24 bit , 16 bit หรือ 15 bit RGB หรือ 8 bit-per-pixel color index format ซึ่งใช้เทียบกับตารางสี (CLUT-color lookup table) โปรแกรมการแปลงจะสนับสนุน 8 bit color index format, ขบวนการแปลง 8 bit color index format จะมีการเปลี่ยนแปลงเล็กน้อย โดยจะตัดข้อมูล file header ส่วนใหญ่ และทำการจัด frame format ให้อยู่ในรูปแบบอย่างง่าย

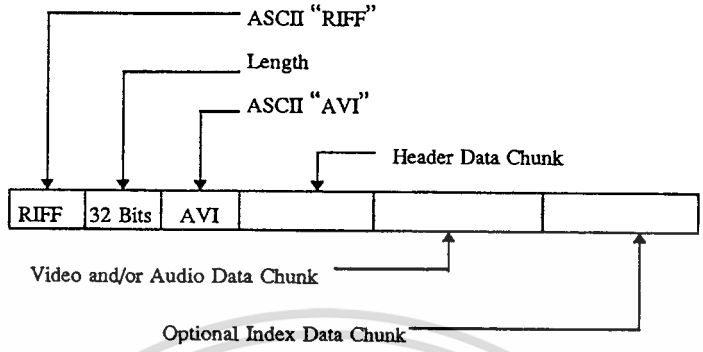
3.1 8 bit AVI file

AVI file เป็นรูปแบบที่อยู่ในรูปไฟล์ชนิด RIFF file ซึ่งเป็นรูปแบบที่ทำขึ้นเพื่อรองรับข้อมูล multimedia ที่สร้างขึ้นมาจากแนวความคิดในการใช้ Data stream แบบขนาน แต่ยังมีข้อจำกัดของจำนวน stream ที่อยู่ใน file header อย่างไรก็ตาม รูปแบบของไฟล์ RIFF จะถูกแบ่ง ให้อยู่ในหน่วยที่รู้จักในรูปของ chunks แต่ละ chunk จะอยู่ในรูปแบบอย่างง่าย ซึ่งแสดงในรูปที่ 3.1



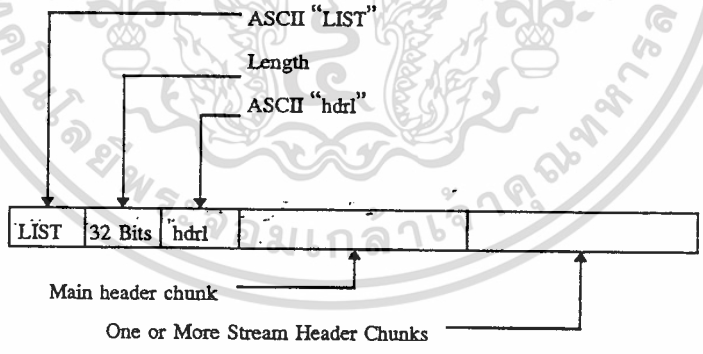
รูปที่ 3.1 ส่วนประกอบของ Microsoft file chunk format.

ความยาวของ chunk จะกำหนดเป็นจำนวน bytes ใน length chunk และตามด้วย Option chunk subtype , AVI file จะทำการเก็บเป็น RIFF chunk ดังแสดงในรูปที่ 3.2



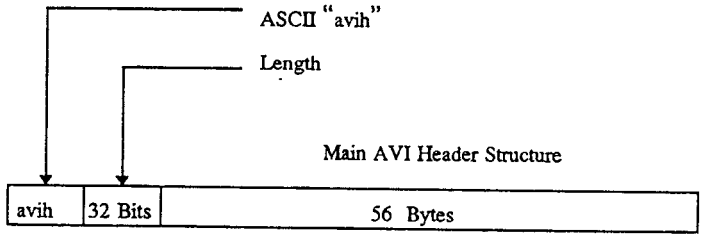
รูปที่ 3.2 แสดง RIFF AVI file format

data segment ของ RIFF chunk จะถูกแบ่งย่อย ๆ เป็น 2 หรือ 3 subsection , section แรกจะเป็น header chunk ซึ่งแสดงในรูปที่ 3.3



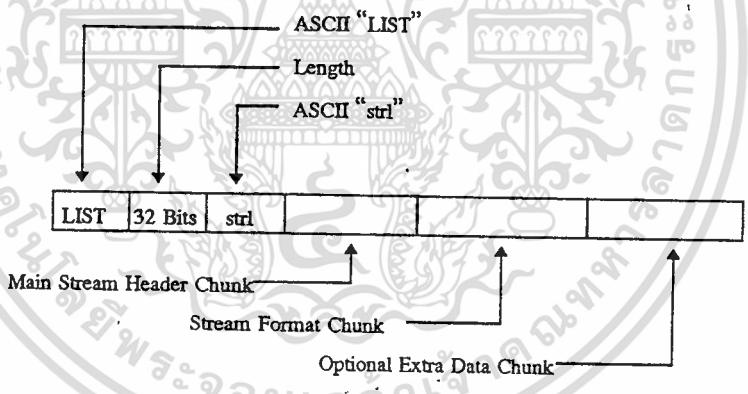
รูปที่ 3.3 รูปแบบของ RIFF AVI header

ในส่วนข้อมูลของ header chunk จะถูกแบ่งย่อยในรูปแบบ main header chunk ตามด้วย 1 stream header หรือมากกว่า รูปแบบของ main header chunk จะแสดงในรูปที่ 3.4



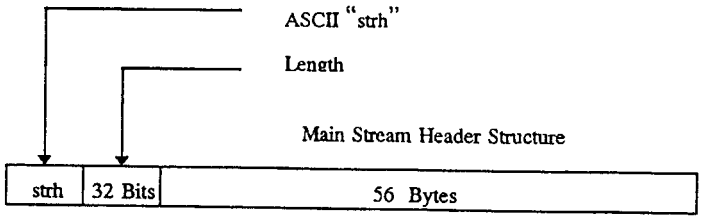
รูปที่ 3.4 รูปแบบของ RIFF AVI main header chunk

จะเห็นได้ว่านี่คือ logical level แรกที่แบ่งย่อย ไม่มี optional chunk subtype เพราะข้อมูลใน chunk จะเป็นรูปแบบที่แน่นอน ต่อจาก main header chunk จะเป็น 1 stream header chunk หรือมากกว่าขึ้นอยู่กับว่า มี data stream อยู่ในไฟล์เป็นจำนวนเท่าใด แต่ละ stream header จะมีรูปแบบดังแสดงในรูปที่ 3.5



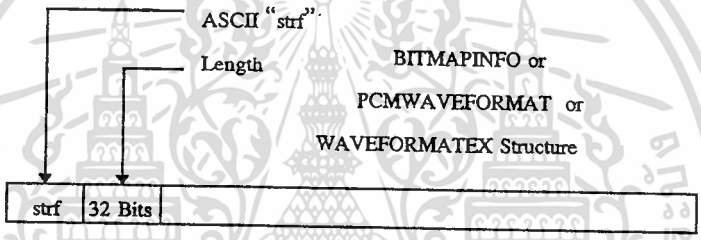
รูปที่ 3.5 รูปแบบของ stream header chunk

Stream header chunk ถูกแบ่งเป็น 2 หรือ 3 subchunk ส่วนแรกคือ main stream header chunk ที่มีลักษณะคล้าย format ของ main AVI header chunk ส่วนที่ 2 จะเป็น stream format chunk ซึ่งจะกำหนดรูปแบบข้อมูลใน stream ส่วนที่ 3 จะเป็น optional chunk ซึ่งมีข้อมูลพิเศษซึ่งใช้โดยตัวถอดรหัส(decoder) โดยรูปแบบที่แน่นอนของข้อมูลจะไม่ถูกกำหนดอยู่ในการกำหนดลักษณะ(specification) ของ AVI โครงสร้างของ main stream header chunk แสดงในรูปที่ 3.6



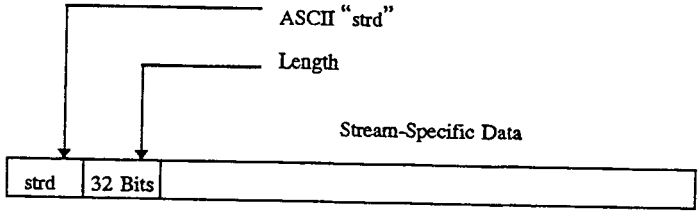
รูปที่ 3.6 รูปแบบของ Main stream header chunk

ต่อจาก header chunk จะตามด้วย stream format chunk ประกอบด้วยโครงสร้างข้อมูลซึ่งกำหนดภายใต้ Windows เป็นรูปแบบของ stream ที่มีพารามิเตอร์แน่นอนใช้ในการกำหนดการเล่นกลับ รูปแบบของ chunk จะแสดงในรูปที่ 3.7



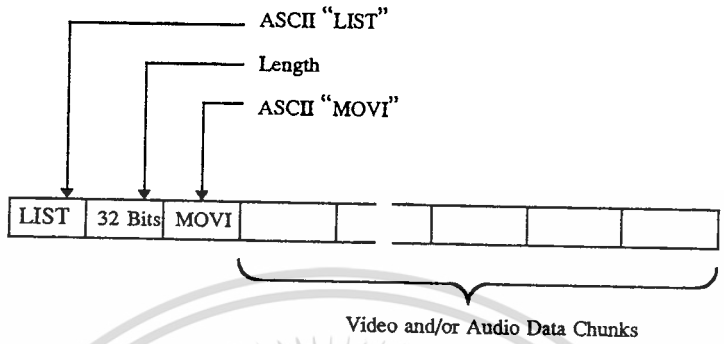
รูปที่ 3.7 รูปแบบของ stream format definition chunk

โครงสร้างภายใน data segment ของ chunk นี้ถูกกำหนดใน Windows SDK โครงสร้าง BITMAPINFO จะใช้สำหรับ image stream และ 2 โครงสร้าง WAVEFORMAT ถูกใช้สำหรับ audio streams รูปแบบของ optional stream specific data chunk แสดงในรูปที่ 3.8



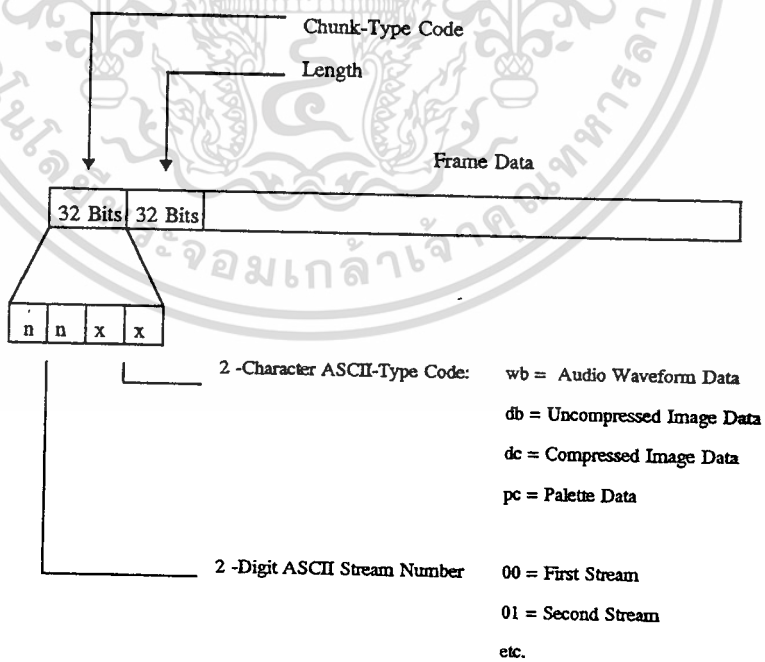
รูปที่ 3.8 รูปแบบของ Optional stream data chunk

Application จะผ่านข้อมูลยาวๆ นี้มายัง format specific driver ภายใต้ Video for Windows main header section ของ AVI file เป็น chunk ที่ประกอบด้วย audio หรือ video data สำหรับ ไฟล์ ที่สร้างขึ้น รูปที่ 3.9 แสดง chunk ถูกเรียบเรียงอย่างไร ปกติ chunk header จะตามด้วย 1 subordinate chunk หรือมากกว่า



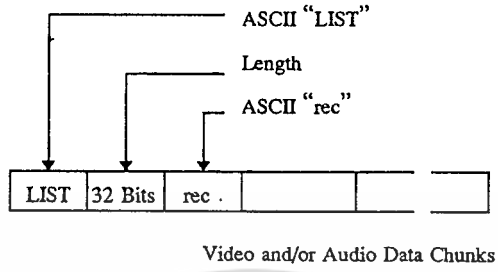
รูปที่ 3.9 AVI file data chunk containing the actual video frames

หนึ่งใน subordinate chunk เหล่านี้ จะใช้สำหรับแต่ละ frame ใน video sequence ถ้ามีหนึ่ง data stream ในไฟล์เท่านั้น เหมือนกับในกรณีของ video ที่ปราศจาก audio โดยที่โครงสร้างของ chunk header อย่างง่ายนี้จะตามด้วย image data ดังแสดงในรูปที่ 3.10



รูปที่ 3.10 รูปแบบของ frame chunk

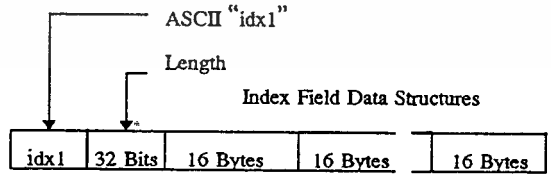
chunk type code กำหนดชนิดของข้อมูลที่บรรจุใน chunk ถ้ามีหลาย stream ในไฟล์ แต่ละ subordinate chunk เหล่านี้อาจจะถูกแบ่งเพิ่มมากขึ้น รูปแบบของ Subordinate chunk แสดงในรูปที่ 3.11



รูปที่ 3.11 รูปแบบของ Frame record chunk

chunk นี้จะถูกแบ่งย่อยออกเป็น subordinate chunks เท่ากับจำนวนของ stream ที่ใช้งาน ในขณะนั้น ถ้ามีจำนวนหลาย stream ไม่จำเป็นที่จะเริ่มที่จุดเดียวกันทั้งหมดในไฟล์ ตัวอย่างเช่น ถ้ามี video stream และ audio stream อาจจะเป็น frame เฉพาะของ video data ใน ไฟล์ ก่อนที่จะเริ่ม audio stream

ในกรณีนี้จะมีหลาย record chunks ดังที่แสดงในรูปที่ 3.11 ซึ่งประกอบด้วย ส่วน video เท่านั้นต่อจากอนุกรมของ record chunk ซึ่งมีทั้งส่วน video และ audio การกระทำนี้จะใช้ buffer ของ video frame เพื่อที่จะได้สะดวกในการ synchronize ระหว่าง audio และ video เมื่อ record chunk ที่ถูกใช้ในไฟล์ subordinate chunk จะมีรูปแบบดังแสดงในรูปที่ 3.10 chunk สุดท้ายในไฟล์ ถ้ามีจะเป็น option index chunk แสดงในรูปที่ 3.12



รูปที่ 3.12 รูปแบบของ Optional Index chunk

index chunk นี้จะไม่ถูกแบ่งย่อยเป็น subordinate chunk ในส่วนของข้อมูลประกอบด้วยอนุกรมของโครงสร้างข้อมูล(แต่ละความยาว 16 bytes) สำหรับทุก ๆ frame ใน main data chunk index chunk จะใช้ในการเข้าถึงข้อมูลแบบ random ซึ่งเป็นประโยชน์สำหรับการเล่นกลับหลัง หรือการแก้ไขลำดับ video

ถ้าเป็นข้อมูลวิดีโอ uncompressed เป็นแบบ 24 bit RGB, ทุก ๆ pixel จะแทนด้วย 3 ไบต์ คือสีแดง สีเขียว และสีน้ำเงิน โดยการอ่านข้อมูลจะเริ่มจากมุมบนซ้ายของรูปภาพ จะสแกนจาก ซ้ายไปขวาและบนลงล่าง, จำนวน bytes ในรูปภาพ จะเป็น 3 เท่าของความสูงคูณด้วยความกว้าง , สำหรับ 8 bit pixels แต่ละ pixel จะใช้ 1 ไบต์ ในไฟล์ ตามด้วยลำดับเหมือนกับใน 24 bit ซึ่งเรียกว่า เป็น device independent bitmaps (DIBs)

3.2 โครงสร้างของ BITMAPINFO

โครงสร้างข้อมูลของ BITMAPINFO ถูกกำหนดโดย Microsoft Windows ซึ่งใช้ในการ compressed และ decompressed ภาพ Bitmaps โครงสร้างจะมีลักษณะดังนี้

```
typedef struct tagBitMAPINFO {
        BITMAPINFOHEADER  bmiHeader;      // BITMAPINFOHEADER data
        RGBQUAD           bmiColors[];    // Color table
} BITMAPINFO;
```

โครงสร้างข้อมูล **BITMAPINFOHEADER** ใช้กับ bmiHeader ใช้ผ่านข่าวสารเกี่ยวกับรูปแบบของ bitmaps ว่า compressed หรือ decompressed มีโครงสร้างดังนี้

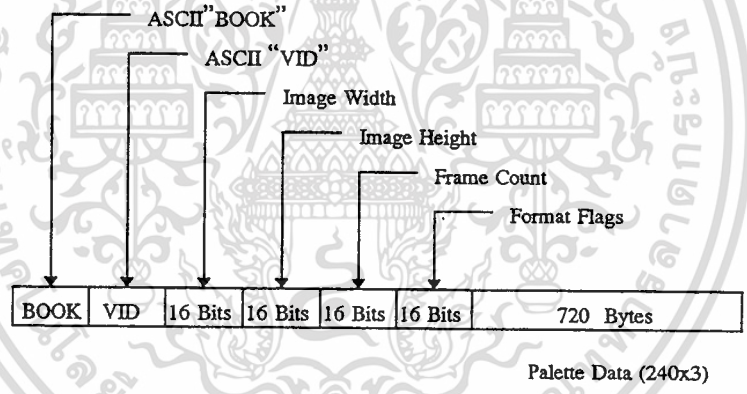
```
typedef struct tagBITMAPINFOHEADER  {
        DWORD     biSize;           // Structure size
        LONG      biWidth;          // Width of the bitmap
        LONG      biHeight;         // Height of the bitmap
        WORD      biPlanes;         // Planes of the target device
        WORD      biBitCount;       // Number of bits per pixel
        DWORD     biCompression     // Compression type
        DWORD     biSizeImage;      // Bytes contained in the image
```

```
LONG        biXPelsPerMeter;    // Horizontal resolution
LONG        biYPelsPerMeter;    // Vertical resolution
DWORD       biClrUsed;          // Number of color indexes
DWORD       biClrImportant;     // Number of important colors
} BITMAPINFOHEADER;
```

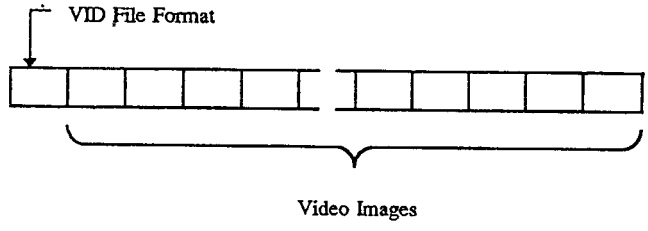
RGBQUAD ใช้แสดงค่าตารางสี โดยผ่านตัวแปร bmiColors

3.3 VID File format

VID file format ถูกสร้างเพื่อให้สนับสนุนอนุกรมของรูปภาพ ที่ใช้ใน motion video รูปแบบจะเป็นรูปแบบง่าย ๆ ประกอบด้วยโครงสร้างของ file header เล็ก ๆ ตามด้วยอนุกรมของ รูปภาพ รูปแบบของ file header แสดงในรูปที่ 3.13 และ ในรูปที่ 3.14 แสดงส่วนประกอบในไฟล์



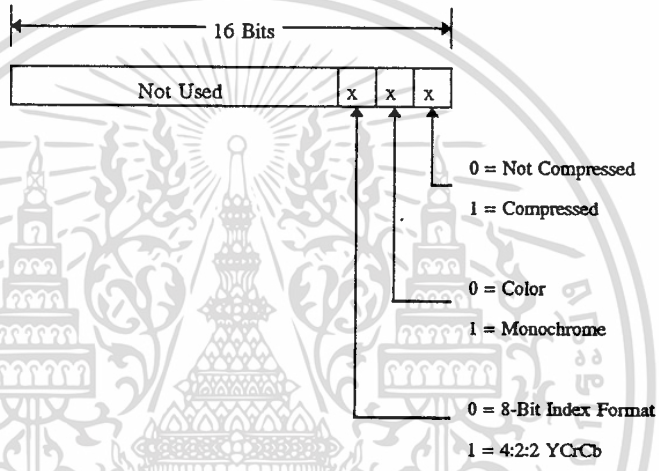
รูปที่ 3.13 แสดง VID file header



รูปที่ 3.14 แสดง Overall VID file

ต่อจากส่วน file header จะเป็นรูปภาพซึ่งถูกเก็บติดต่อกันไปเป็นลำดับ สำหรับไฟล์วิดีโอที่ไม่ได้ compress ขอบเขตระหว่างรูปภาพแต่ละภาพ จะหาได้จากการคำนวนขนาดของรูปภาพ เช่น 160x120 video ซึ่งใช้ 8 bit color lookup format จะมีอนุกรมของเฟรม ที่ประกอบด้วย 19,200 ไบต์

format flags ใน file header ประกอบด้วยบิตแสดงสถานะ 3 บิต ใช้ในการบอกว่าถูก compressed หรือ uncompressed , เป็นสี หรือขาวดำ และอยู่ในรูปแบบ 8 bit หรือในรูปแบบ 4:2:2 YCrCb format การจัดเรียง flag bits แสดงในรูปที่ 3.15



รูปที่ 3.15 Flag bit assignment in VID file header

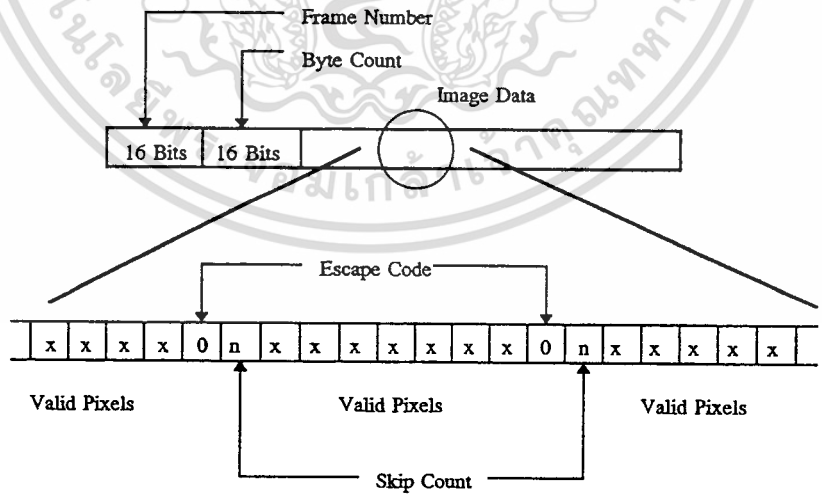
VID file สามารถทำการ compressed ถ้าเก็บในรูป 8 bit color index format การ compressed สามารถเล่นกลับได้เหมือนกับตอนไม่ compressed ในส่วนของการเล่นกลับของ compressed file จะมีความเรียบของภาพมากกว่า รูปแบบขาวดำ(monochrome) แต่ละ pixel จะแสดงการส่องสว่างเท่านั้น มีช่วงระหว่าง 16 ถึง 256 รูปแบบ YCrCb จะถูกใช้ในโปรแกรม JPEG เพื่อที่จะ compress และ decompress ภาพแบบ full color ซึ่งรูปแบบนี้สามารถเล่นกลับใช้โปรแกรม VIEWER โดยจะแสดงเฉพาะค่าการส่องสว่างของข้อมูลเท่านั้น(ขาวดำ)

3.4 รูปแบบของ Compressed Color Index

compression อัลกอลิธึม ที่ใช้กับ color index format คือการตัดลำดับภาพที่ไม่ใช้ในไฟล์ออก การตัดออกจะหมายความว่า ถ้าค่าในพิกเซลเหมือนภาพที่แสดงขณะนั้น ภาพก่อนหน้านั้นจะไม่ถูกเก็บลงในไฟล์ ภาพก่อนหน้าอีกภาพจะถูกใช้แทน threshold levelกำหนดให้พิกเซลเปรียบเทียบกับค่า RGB มากกว่า Index ของมัน

การลดลงของ noise signal threshold ถูกเซตที่ระดับต่ำ ความแตกต่างในพิกเซลถูกสมมติไม่เข้ากัน วิเคราะห์แต่ละแถกของพิกเซลรูปภาพ จะพบกลุ่มพิกเซลที่ไม่เปลี่ยนแปลงไปจากภาพก่อน กลุ่มเหล่านี้จะถูกเก็บเป็น escape code และค่าที่นับแสดงจาก viewer program ว่า จำนวนพิกเซลที่ข้ามไปก่อนทำการเก็บพิกเซลที่ใช้ได้ถัดไป 16 colors ใน VGA palette ถูกสงวนไว้สำหรับ standard VGA color หมายถึง พิกเซลรูปภาพที่ใช้ได้ค่าค่า index เป็น 16 หรือมากกว่าศูนย์จะถูกใช้เป็น escape code เมื่อค่าศูนย์ถูกตรวจพบ ไบต์ถัดไปแสดงจำนวนพิกเซลที่ถูกข้ามก่อนทำการเก็บพิกเซลถัดไป หมายค่าวาขนาดของเฟรมรูปภาพในไฟล์จะไม่ยาวเกินกว่าที่กำหนด 2-16 บิต Integer เก็บขณะเริ่มต้นเฟรมรูปภาพใน compressed file ดังแสดงในรูปที่ 3.16

จำนวนเฟรมที่ใช้ในการรักษา frame synchronization ขณะ decompress viewer program รู้จำนวนเฟรมถัดไปที่จะทำการเล่น ดังนั้น ถ้าถูกตรวจไม่พบ ทำให้รู้ว่ามี error เกิดขึ้นหรือ input file ที่เข้ามาไม่ถูกต้อง บล๊อกความยาว แสดงจำนวนไบต์ที่อ่านในเฟรมปัจจุบันก่อนการ decompress

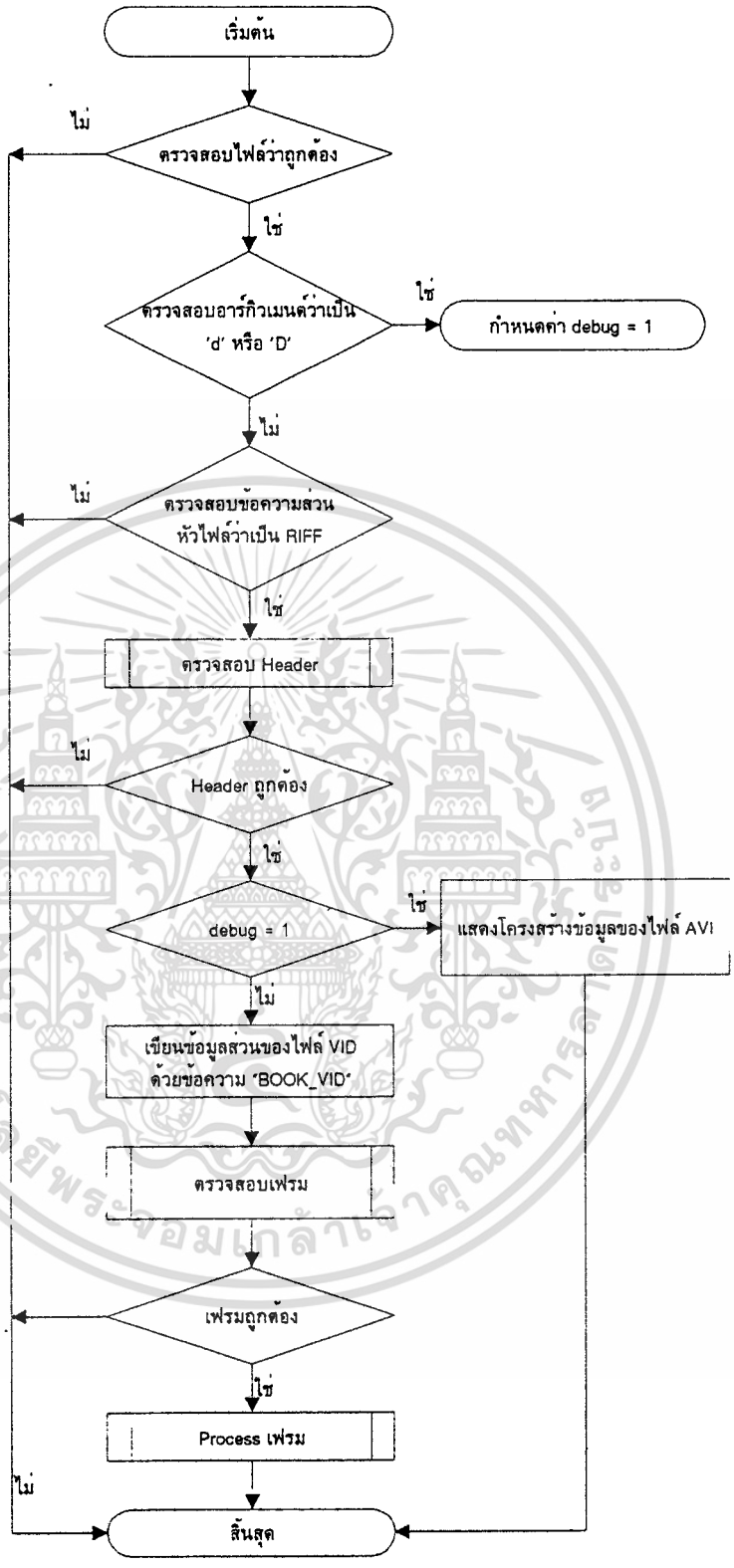


รูปที่ 3.16 แสดง 8-bit-per-pixel compression ใน VID file

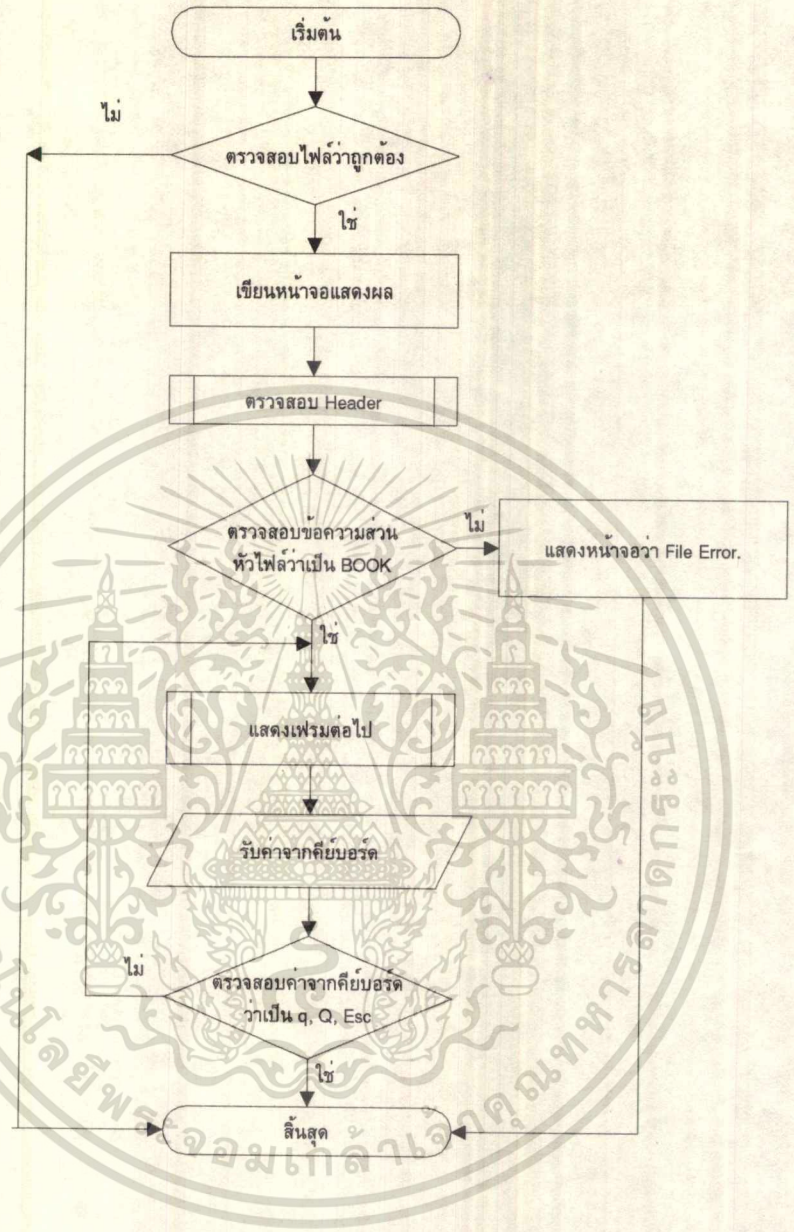
บทที่ 4

ผังงานและแนวความคิดในการออกแบบ

การทำงานของโปรแกรม AVI2VID.EXE และ VID_VIEW.EXE มีขั้นตอนในการทำงานดังแสดงในผังงานที่ 1 และ 2



ผังงานที่ 1 แสดงการทำงานของโปรแกรม AVI2VID.EXE



ผังงานที่ 2 แสดงการทำงานของโปรแกรม VID_VIEW.EXE

บทที่ 5

การทดลอง

ขั้นตอนการทดลอง

5.1 แสดงข้อมูลโครงสร้างของไฟล์ AVI ด้วยโปรแกรม AVI2VID.EXE โดยใช้คำสั่ง AVI2VID ตามด้วยไฟล์ต้นฉบับคือ test.avi ไฟล์ปลายทางคือ test.vid และพารามิเตอร์ D (ใช้ในการแสดงข้อมูลโครงสร้างของไฟล์)

5.2 แปลงไฟล์ AVI เป็น VID ด้วยโปรแกรม AVI2VID.EXE โดยใช้คำสั่ง AVI2VID ตามด้วยไฟล์ต้นฉบับคือ test.avi ไฟล์ปลายทางคือ test.vid

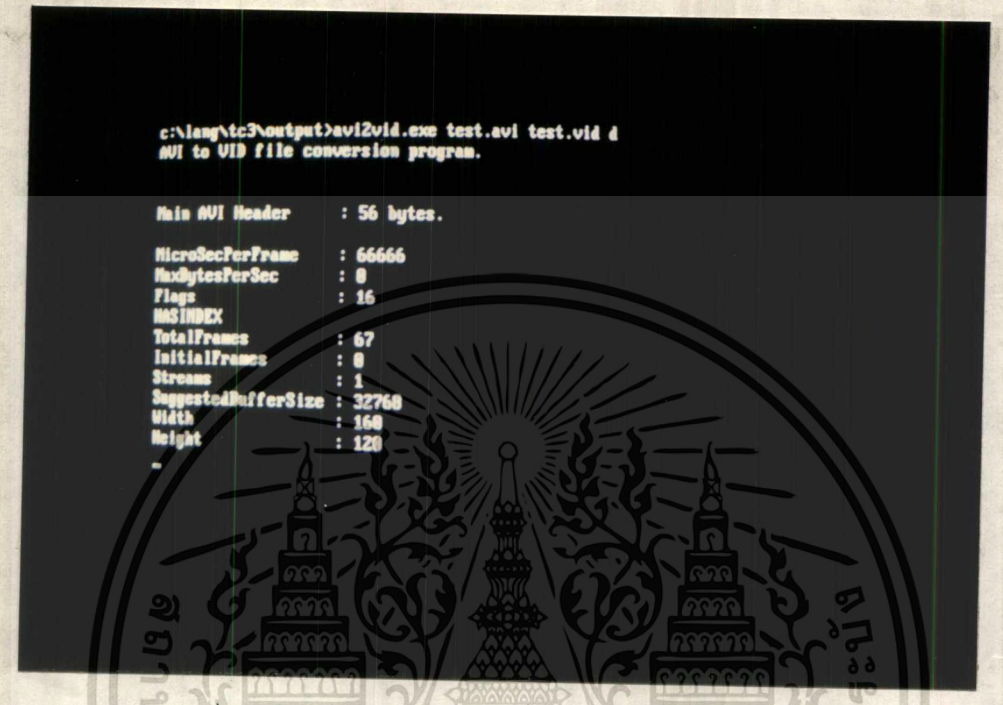
5.3 นำไฟล์ VID มาแสดง ด้วยโปรแกรม VID_VIEW.EXE โดยใช้คำสั่ง VID_VIEW ตามด้วยไฟล์ต้นฉบับคือ test.vid

5.4 นำไฟล์ VID มาแสดง ด้วยโปรแกรม VID_VIEW.EXE โดยใช้คำสั่ง VID_VIEW ตามด้วยไฟล์ต้นฉบับคือ tennis.vid

บทที่ 6

ผลการทดลอง

6.1 จากการทดลองที่ 5.1 แสดงผลได้ดังนี้



รูปที่ 6.1 แสดงการแสดงโครงสร้างข้อมูลของไฟล์ test.avi

โดยมีผลเป็นโครงสร้างข้อมูลของไฟล์ test.avi ดังนี้

AVI to VID file conversion program.

Main AVI Header	: 56 bytes.
MicroSecPerFrame	: 66666
MaxBytesPerSec	: 0
Flags	: 16
HASINDEX	
TotalFrames	: 67
InitialFrames	: 0
Streams	: 1
SuggestedBufferSize	: 32768
Width	: 160
Height	: 120
AVI Stream Header	: 56 bytes.
Type	: vidsDIB
Handler	: DIB
Flags	: 0

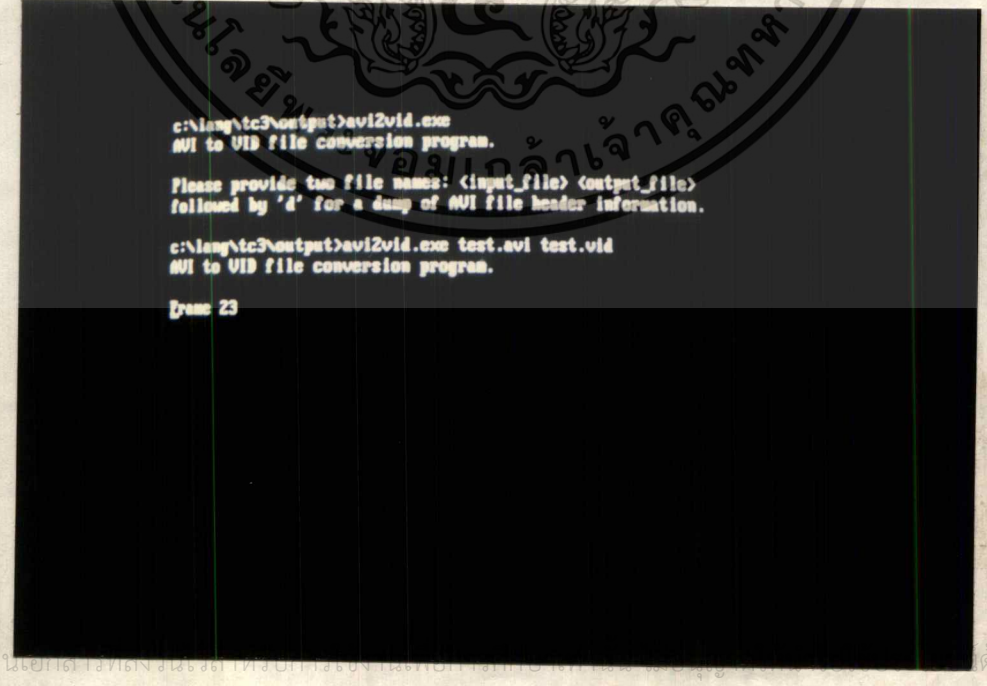
Priority	: 0
InitialFrames	: 0
Scale	: 1
Rate	: 15
Start	: 0
Length	: 67
SuggestedBufferSize	: 19200
Quality	: 0
SampleSize	: 0

Stream Format Header: 1064 bytes.

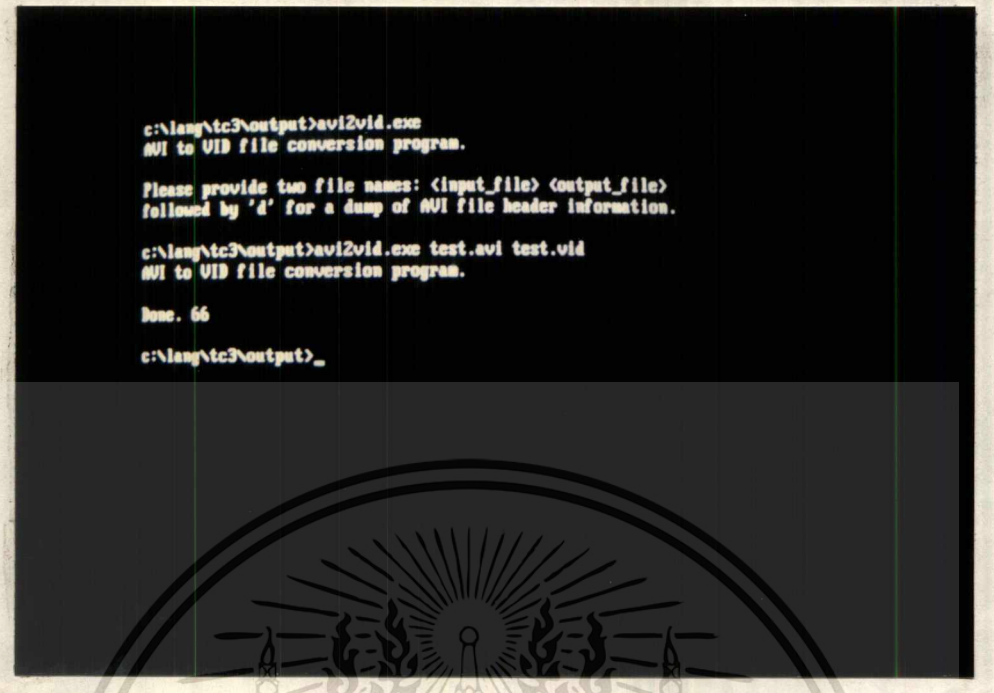
Size	: 40
Width	: 160
Height	: 120
Planes	: 1
BitCount	: 8
Compression	: 0
SizeImage	: 19200
XPelsPerMeter	: 0
YPelsPerMeter	: 0
ClrUsed	: 256
ClrImportant	: 0

Done.

6.2 จากการทดลองที่ 5.2 แสดงผลได้ดังนี้



รูปที่ 6.2 แสดงขณะทำการแปลงไฟล์ test.avi ไปเป็น test.vid

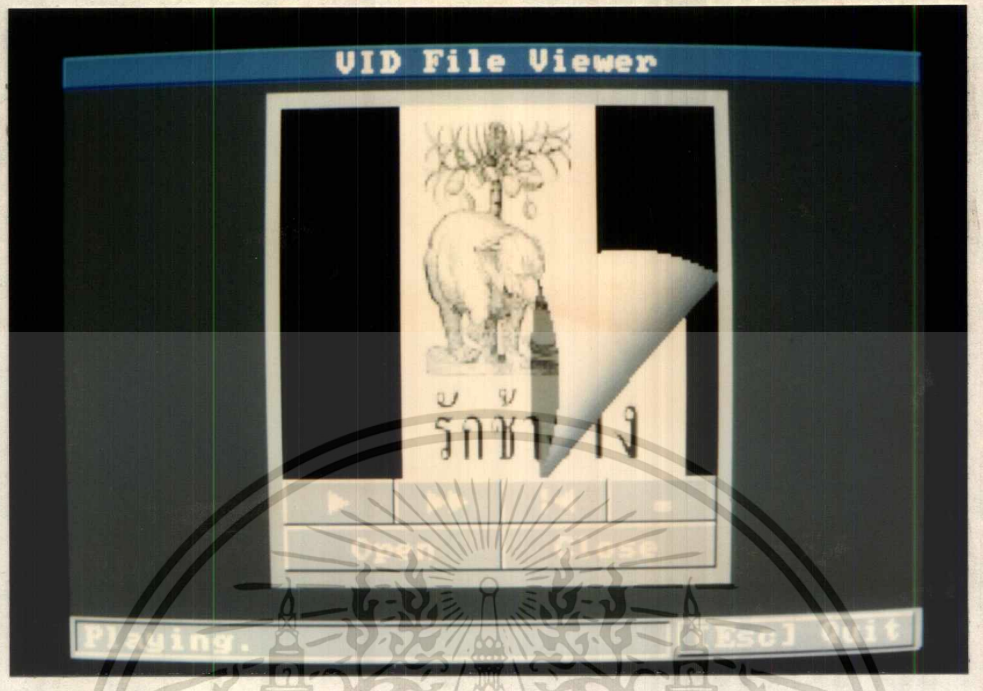


รูปที่ 6.3 แสดงขณะทำการแปลงไฟล์ test.avi ไปเป็น test.vid เสร็จเรียบร้อยแล้ว

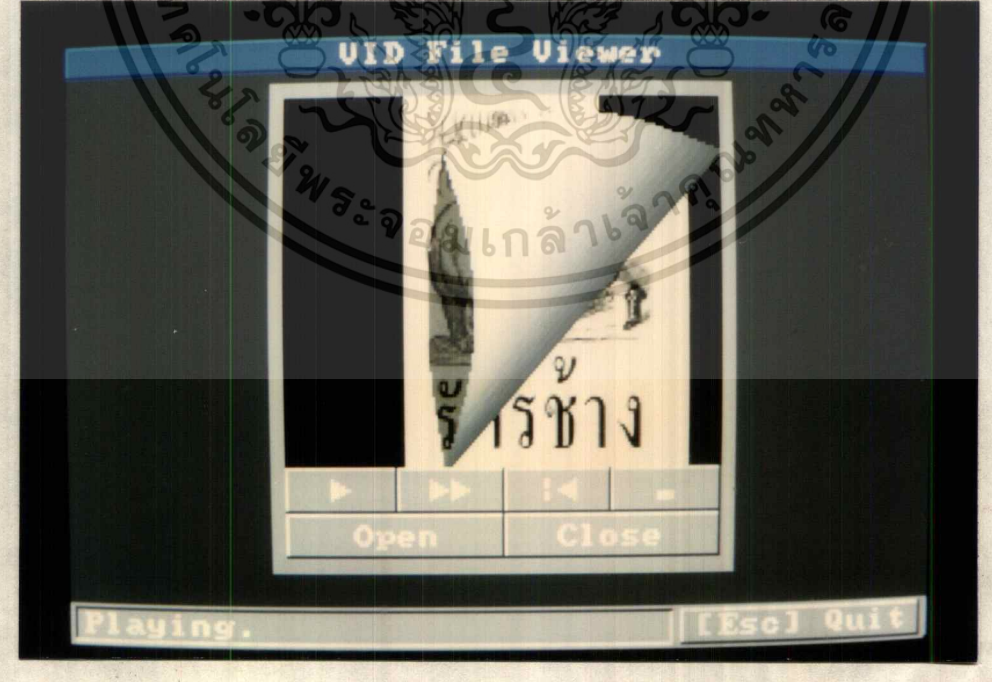
จากรูปที่ 6.3 จะเป็นได้ว่าเมื่อขณะทำการแปลงไฟล์เสร็จเรียบร้อยแล้วจะมีเฟรมทั้งหมด 67 เฟรมโดยที่เฟรมที่ 67 จะแสดงว่าคำว่า Done.

6.3 จากการทดลองที่ 5.3 แสดงผลได้ดังนี้

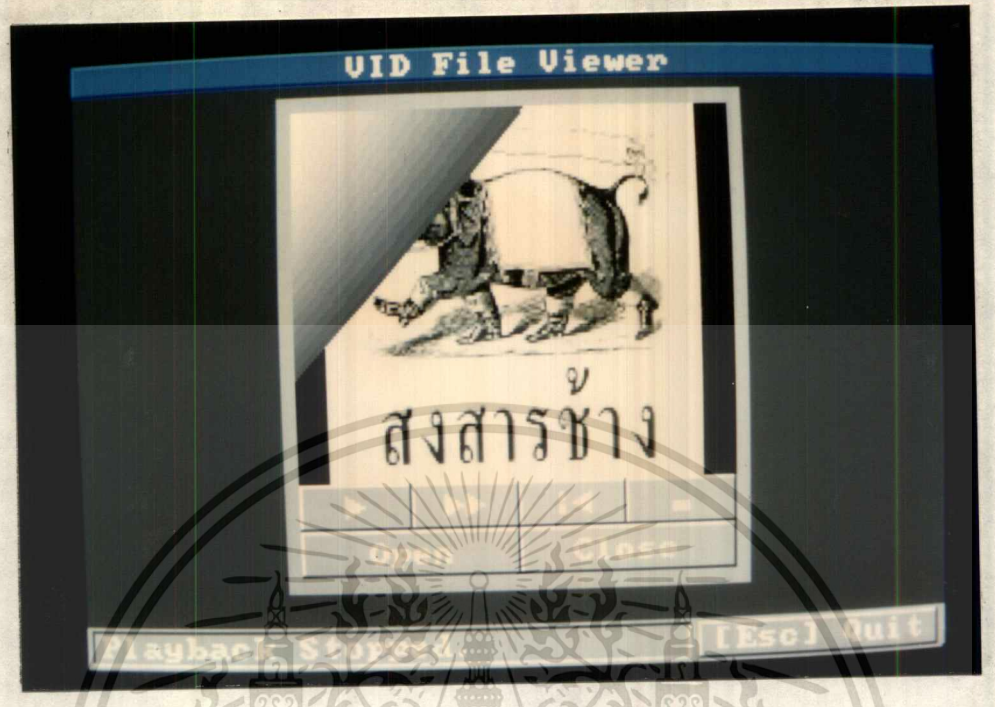
รูปที่ 6.4 แสดงขณะทำการเปิดไฟล์ต้นฉบับคือ test.vid



รูปที่ 6.5 แสดงขณะทำการเล่นไฟล์ test.vid

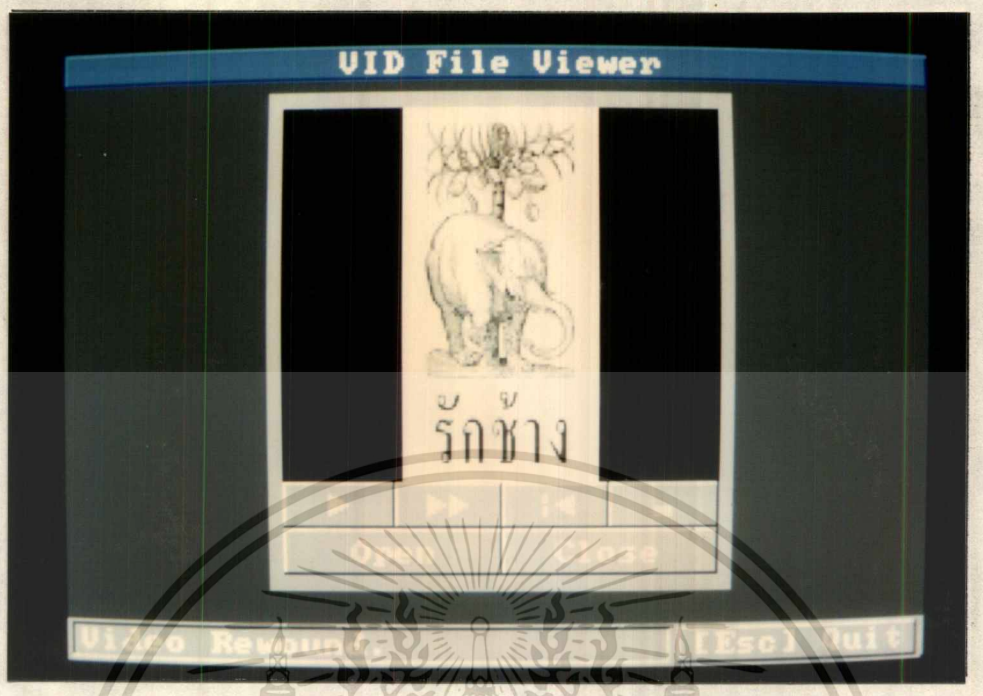


รูปที่ 6.6 แสดงขณะทำการเล่นไฟล์ test.vid



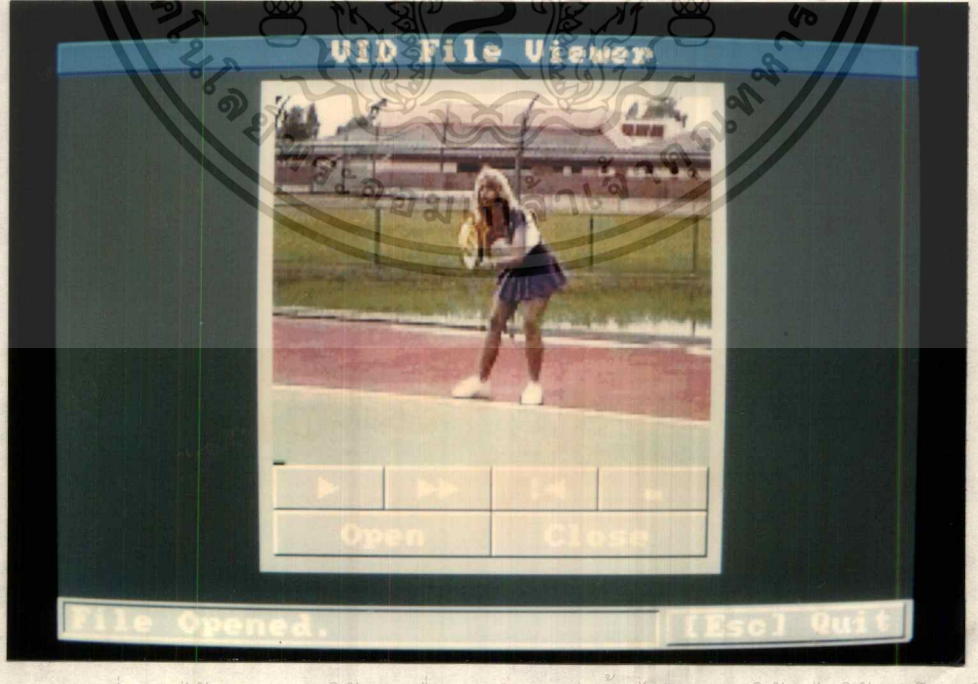
รูปที่ 6.7 แสดงขณะหยุดเล่นไฟล์ test.vid

รูปที่ 6.8 แสดงขณะการเล่นไฟล์ test.vid สิ้นสุดลง



รูปที่ 6.9 แสดงเมื่อทำการกลับมาเริ่มต้นใหม่ที่เฟรมแรกของไฟล์ test.vid

6.4 จากการทดลอง 5.4 แสดงผลได้ดังนี้



รูปที่ 6.10 แสดงขณะทำการเปิดไฟล์ต้นฉบับคือ tennis.vid

รูปที่ 6.11 แสดงขณะทำการเล่นไฟล์ tennis.vid

รูปที่ 6.12 แสดงขณะทำการเล่นไฟล์ tennis.vid

รูปที่ 6.13 แสดงขณะการเล่นไฟล์ tennis.vid สิ้นสุดลง

บทที่ 7

สรุปและวิเคราะห์ผลการทดลอง

จากผลการทดลองที่ 6.1และ 6.2 จะเห็นได้ว่า รูปแบบ AVI นั้นมีโครงสร้างของข้อมูลค่อนข้างซับซ้อนประกอบไปด้วย Header หลายส่วนเช่น Main Header, Stream Header และStream Format Header กว่าจะเข้าถึงข้อมูลของภาพได้ก็ต้องผ่าน Header หลายส่วน ซึ่งเป็นยุ่งยาก จึงได้มีการแปลงให้อยู่ในรูปแบบ VID ที่มีโครงสร้างที่ง่ายเพื่อให้เข้าถึงข้อมูลภาพได้ง่ายขึ้นและมีขนาดของไฟล์เล็กลง ทำให้ประหยัดเนื้อที่ในการเก็บเนื่องจากไม่มีโครงสร้างที่ซับซ้อนมากมาย

จากผลการทดลองที่ 6.3 และ 6.4 จะเห็นได้ว่า รูปแบบ VID ที่ทำการแปลงแล้วนั้นสามารถแสดงผลได้ดีพอ ๆ กับรูปแบบ AVI ต้นฉบับ จึงน่าจะเป็นการดีกว่าที่จะเก็บให้อยู่ในรูปแบบ VID แต่ในความเป็นจริง รูปแบบ VID ที่สร้างขึ้นมาใหม่นี้ยังไม่เป็นมาตรฐานและเป็นที่ยอมรับกันอย่างกว้างขวางเมื่อเทียบกับรูปแบบของ AVI เดิม ดังนั้นรูปแบบ VID ที่สร้างขึ้นนี้ จึงสร้างขึ้นเพื่อใช้ในการศึกษาถึงลักษณะการเก็บข้อมูลทางดิจิตอลวิดีโอและอาจจะนำมาพัฒนาต่อไปให้มีประสิทธิภาพสูงขึ้นจนเป็นที่ยอมรับอย่างกว้างขวางได้

บทที่ 8

แนวทางพัฒนาต่อ

เนื่องจากไฟล์ AVI ที่นำมาใช้นั้นมีข้อจำกัดในการใช้งานอยู่คือ ต้องเป็น 8 บิต 256 สี มีขนาดภาพ(resolution) 160x120 พิกเซล และไม่มีส่วนของข้อมูลเสียง จึงทำให้ใช้งานไม่ได้กว้างขวางเท่าที่ควร น่าจะพัฒนาให้สามารถรองรับได้กับไฟล์ที่มีขนาดใหญ่ขึ้น มีสีมากขึ้นและสามารถเล่นข้อมูลเสียงในไฟล์ได้

ส่วนไฟล์ VID ที่ได้นั้นมีขนาดใกล้เคียงกับไฟล์ AVI ต้นฉบับ ดังนั้นจึงน่าที่จะมีการบีบอัดข้อมูลเพื่อให้ไฟล์มีขนาดเล็กลง สะดวกในการจัดเก็บถ้าไฟล์เกิดมีขนาดใหญ่มาก ๆ และโปรแกรมที่ใช้ในการแสดงภาพของไฟล์ VID ก็ควรที่จะปรับปรุงให้มีรูปแบบในการติดต่อกับผู้ใช้ที่ดีขึ้นและมีฟังก์ชันในการใช้งานมากขึ้น

ภาคผนวก ก

แสดงโปรแกรม AVI2VID.CPP

โปรแกรมหลักในการแปลงจาก AVI เป็น VID

```
//=========================================================
//
// AVI2VID.CPP
//
//=========================================================

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#include "avi_riff.h"

//---------------------------------------------------------
// Global program variables
//---------------------------------------------------------
MainAVIHeader           main_header;            // AVI main header
AVIStreamHeader         stream_header;          // AVI stream header
BitMapInfoHeader        stream_format;          // AVI DIB stream format
RGBQuad                 palette[256];           // VGA palette data
vid                     vid_header;             // Output file header
BYTE                    local_palette[240][3];  // Local palette storage
BYTE                    *frame_buffer;          // Frame buffer pointer
FILE                    *output;                // File output buffer pointer
int                     stream_number;          // Stream number storage
int                     frame;                  // Frame counter

//---------------------------------------------------------
int CMP4(char *s1, char *s2)// Compare a pair of 4-char tags
 {
 if(s1[0]==s2[0] && s1[1]==s2[1] && s1[2]==s2[2] && s1[3]==s2[3])return(1);
 return(0);
 }

//---------------------------------------------------------
void ExitError(int n)
 {
 switch(n)
   {
   case 0:printf("ERROR -- This is not a RIFF file.\n"); break;
   case 1:printf("ERROR -- This is not an AVI file.\n"); break;
   case 2:printf("ERROR -- Invalid file format.\n");     break;
   case 3:printf("ERROR -- Invalid header length.\n");   break;
   case 4:printf("ERROR -- File must have DIB frame format.\n"); break;
   case 5:printf("ERROR -- Can only read 8-bit pixels.\n");  break;
   case 6:printf("ERROR -- Insufficient frame buffer memory.\n");break;
```

```
  default:printf("PROGRAM ERROR -- Undefined error number.\n");
  }
 exit(0);
 }

//-------------------------------------------------------
// Convert  input AVI frame to output VID frame
//-------------------------------------------------------
long ProcessFrame(FILE *stream, TagT *RIFF_tag)
 {
 char          chunk_type[4];          // RIFF type ID
 long          bytes_read;             // Main AVI header block length
 long          block_length;           // Main AVI header block length
 BYTE          line_buffer[1024];      // Temporary line buffer
 BYTE          R,G,B;                  // RGB storage
 WORD          i;                      // General counter

 chunk_type[0]='0';              // Construct stream chunk ID
 chunk_type[1]='0';
 chunk_type[2]='d';
 chunk_type[3]='b';

 bytes_read=RIFF_tag->chunk_size;
 printf("Frame %d%c",frame,0x0d);

 fread(frame_buffer,bytes_read,1,stream);
 if(CMP4(RIFF_tag->chunk_type,chunk_type))
  {
  int x_size=vid_header.x_size;
  int y_size=vid_header.y_size;

  if(frame==0)fwrite(&vid_header,sizeof(vid),1,output);

  for(int y=0;y<y_size/2;y++)             // Un-invert DIB image
   {
   memmove(line_buffer,&frame_buffer[bytes_read-((y+1)*x_size)],x_size);
   memmove(&frame_buffer[bytes_read-((y+1)*x_size)],&frame_buffer[y*x_size],x_size);
   memmove(&frame_buffer[y*x_size],line_buffer,x_size);
   }

  for(int i=0;i<bytes_read;i++)           // Adjust pixel base value
   {
   frame_buffer[i]=frame_buffer[i]+16;
   }
  fwrite(frame_buffer,bytes_read,1,output);
```

```
    frame++;
    }
  return(bytes_read);
  }

//-----------------------------------------------------------
// Process a 'rec' chunk
//-----------------------------------------------------------
long ReadRecord(FILE *stream, long chunk_size)
  {
  TagT          RIFF_tag;           // RIFF file tag structure
  long          bytes;              // Main AVI header block length
  long          bytes_read;         // Main AVI header block length
  long          block_length;       // Main AVI header block length
  long          header_length;      // Main AVI header length

  block_length=chunk_size-ID_SIZE;
  bytes_read=0;

  while(block_length>0L)
    {
    fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);      // Get chunk tag
    block_length-=TAG_SIZE;
    bytes_read+=TAG_SIZE;

    bytes=ProcessFrame(stream,&RIFF_tag);
    block_length-=bytes;
    bytes_read+=bytes;
    }
  return(bytes_read);
  }

//-----------------------------------------------------------
// Scan through a LIST 'movi' chunk, and processing frames
//-----------------------------------------------------------
int ReadFrames(FILE *stream)
  {
  TagJ          JUNK_tag;           // JUNK chunk tag structure
  TagT          RIFF_tag;           // RIFF file tag structure
  char          RIFF_type[4];       // RIFF type ID
  long          bytes;              // Main AVI header block length
  long          bytes_read;         // Main AVI header block length
  long          block_length;       // Main AVI header block length
  long          header_length;      // Main AVI header length
```

```
 fread(&JUNK_tag.JUNK_type,TAG_SIZE,1,stream);          // Get 'JUNK' chunk tag
 if(!CMP4(JUNK_tag.JUNK_type,"JUNK"))ExitError(2);
 fseek(stream,JUNK_tag.JUNK_size,1);

 fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);         // Get chunk tag
 fread(&RIFF_type,ID_SIZE,1,stream);
 if(!CMP4(RIFF_tag.chunk_type,"LIST"))ExitError(2);
 if(!CMP4(RIFF_type,"movi"))ExitError(2);
 block_length=RIFF_tag.chunk_size-ID_SIZE;
 bytes_read=TAG_SIZE+ID_SIZE;

 frame=0;
 while(frame<(int)main_header.TotalFrames)
  {
  fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);
  block_length-=TAG_SIZE;
  bytes_read+=TAG_SIZE;

  if(CMP4(RIFF_tag.chunk_type,"LIST"))
    {
    fread(&RIFF_type,ID_SIZE,1,stream);
    if(!CMP4(RIFF_type,"rec "))ExitError(2);
    block_length-=ID_SIZE;
    bytes_read+=ID_SIZE;
    bytes=ReadRecord(stream,RIFF_tag.chunk_size);
    }
  else
    {
    bytes=ProcessFrame(stream,&RIFF_tag);
    }
  block_length-=bytes;
  bytes_read+=bytes;
  }
 return(0);
 }

//-------------------------------------------------------
// Read a stream format chunk from the AVI file
//-------------------------------------------------------
long ReadStreamFormat(FILE *stream, int debug)
 {
 TagT            RIFF_tag;              // RIFF file tag structure
 char            RIFF_type[4];          // RIFF type ID
 long            bytes_read;            // Stream header block length
 long            format_length;         // Stream format length
```

```
fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);          // Get block tag
if(!CMP4(RIFF_tag.chunk_type,'strf'))ExitError(2);
bytes_read=TAG_SIZE;

format_length=RIFF_tag.chunk_size;                      // Save length
bytes_read+=format_length;
fread(&stream_format,DIB_HDR_SIZE,1,stream);            // Read format data

if(debug)
 {
 printf('\n');
 printf('Stream Format Header: %ld bytes.\n\n',format_length);
 printf('Size          : %ld\n',stream_format.Size);
 printf('Width         : %ld\n',stream_format.Width);
 printf('Height        : %ld\n',stream_format.Height);
 printf('Planes        : %d \n',stream_format.Planes);
 printf('BitCount      : %d \n',stream_format.BitCount);
 printf('Compression   : %ld\n',stream_format.Compression);
 printf('SizeImage     : %ld\n',stream_format.SizeImage);
 printf('XPelsPerMeter : %ld\n',stream_format.XPelsPerMeter);
 printf('YPelsPerMeter : %ld\n',stream_format.YPelsPerMeter);
 printf('ClrUsed       : %ld\n',stream_format.ClrUsed);
 printf('ClrImportant  : %ld\n',stream_format.ClrImportant);
 getch();
 }

if(stream_format.BitCount!=8)ExitError(5);
format_length-=(long)DIB_HDR_SIZE;
fread(&palette[0],format_length,1,stream);              // Read palette data
return(bytes_read);
}

//-------------------------------------------------------------
// Read a stream header chunk from the AVI file
//-------------------------------------------------------------
long ReadStreamHeader(FILE *stream, int debug, int n)
 {
 TagT            RIFF_tag;          // RIFF file tag structure
 char            RIFF_type[4];      // RIFF type ID
 long            block_length;      // Stream header block length
 long            bytes_read;        // Stream header block length
 long            header_length;     // Stream header length
 AVIStreamHeader local_header;      // Local header data buffer

 fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);          // Get header block tag
```

```
fread(&RIFF_type,ID_SIZE,1,stream);                    // Get stream length tag
if(!CMP4(RIFF_tag.chunk_type,'LIST'))ExitError(2);
if(!CMP4(RIFF_type,'strl'))ExitError(2);
bytes_read=TAG_SIZE+ID_SIZE;
block_length=RIFF_tag.chunk_size-ID_SIZE;              // Save header length

fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);         // Get stream header tag
if(!CMP4(RIFF_tag.chunk_type,'strh'))ExitError(2);
header_length=RIFF_tag.chunk_size;                     // Save header length
bytes_read+=TAG_SIZE;
block_length-=TAG_SIZE;
if(header_length!=STR_HDR_SIZE)ExitError(3);
fread(&local_header,STR_HDR_SIZE,1,stream);            // Read header data
block_length-=header_length;
bytes_read+=header_length;

if(debug)
 {
 printf('\n');
 printf('AVI Stream Header  : %ld bytes.\n\n',header_length);
 printf('Type               : %s\n',local_header.Type);
 printf('Handler            : %s\n',local_header.Handler);
 printf('Flags              : %ld\n',local_header.Flags);
 if(local_header.Flags&RIFF_HASINDEX)    printf(' HASINDEX\n');
 if(local_header.Flags&RIFF_MUSTUSEINDEX) printf(' MUSTUSEINDEX\n');
 if(local_header.Flags&RIFF_ISINTERLEAVED)printf(' ISINTERLEAVED\n');
 if(local_header.Flags&RIFF_COPYRIGHTED) printf(' COPYRIGHTED\n');
 printf('Priority           : %ld\n',local_header.Priority);
 printf('InitialFrames      : %ld\n',local_header.InitialFrames);
 printf('Scale              : %ld\n',local_header.Scale);
 printf('Rate               : %ld\n',local_header.Rate);
 printf('Start              : %ld\n',local_header.Start);
 printf('Length             : %ld\n',local_header.Length);
 printf('SuggestedBufferSize : %ld\n',local_header.SuggestedBufferSize);
 printf('Quality            : %ld\n',local_header.Quality);
 printf('SampleSize         : %ld\n',local_header.SampleSize);
 getch();
 }

if(CMP4(local_header.Type,'vids') && CMP4(local_header.Handler,'DIB '))
 {
 memcpy(&stream_header,&local_header,STR_HDR_SIZE);    // Save this header
 long bytes=ReadStreamFormat(stream,debug);            // Get format data
 block_length-=bytes;
```

```
  bytes_read+=bytes;
  stream_number=n;
  }
 if(block_length>0L)
  {
  fseek(stream,block_length,SEEK_CUR);
  bytes_read+=block_length;
  }
 return(bytes_read);
 }

//-------------------------------------------------------
// Read a main AVI header chunk from the AVI file
//-------------------------------------------------------
int ReadHeader(FILE *stream, int debug)
 {
 TagT            RIFF_tag;              // RIFF file tag structure
 char            RIFF_type[4];          // RIFF type ID
 long            bytes_read;            // Main AVI header block length
 long            block_length;          // Main AVI header block length
 long            header_length;         // Main AVI header length

 fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);      // Get 'LIST....hdrl'
 fread(&RIFF_type,ID_SIZE,1,stream);
 if(ICMP4(RIFF_tag.chunk_type,'LIST'))ExitError(2);
 if(ICMP4(RIFF_type,'hdrl'))ExitError(2);
 bytes_read=TAG_SIZE+ID_SIZE;
 block_length=RIFF_tag.chunk_size-ID_SIZE;          // Account for 'hdrl'

 fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);      // Get header tag
 if(ICMP4(RIFF_tag.chunk_type,'avih'))ExitError(2);
 bytes_read+=TAG_SIZE;
 block_length-=TAG_SIZE;
 header_length=RIFF_tag.chunk_size;                 // Save header length
 if(header_length!=MAIN_HDR_SIZE)ExitError(3);
 fread(&main_header,MAIN_HDR_SIZE,1,stream);        // Read header data
 bytes_read+=header_length;
 block_length-=header_length;

 if(debug)
  {
  printf("\n");
  printf("Main AVI Header     : %ld bytes.\n\n",header_length);
  printf("MicroSecPerFrame    : %ld\n",main_header.MicroSecPerFrame);
  printf("MaxBytesPerSec      : %ld\n",main_header.MaxBytesPerSec);
```

```
  printf("Flags             : %ld\n",main_header.Flags);
  if(main_header.Flags&RIFF_HASINDEX)     printf("  HASINDEX\n");
  if(main_header.Flags&RIFF_MUSTUSEINDEX) printf("  MUSTUSEINDEX\n");
  if(main_header.Flags&RIFF_ISINTERLEAVED)printf("  ISINTERLEAVED\n");
  if(main_header.Flags&RIFF_COPYRIGHTED)  printf("  COPYRIGHTED\n");
  printf("TotalFrames       : %ld\n",main_header.TotalFrames);
  printf("InitialFrames     : %ld\n",main_header.InitialFrames);
  printf("Streams           : %ld\n",main_header.Streams);
  printf("SuggestedBufferSize : %ld\n",main_header.SuggestedBufferSize);
  printf("Width             : %ld\n",main_header.Width);
  printf("Height            : %ld\n",main_header.Height);
  getch();
  }

 for(int i=0;i<main_header.Streams;i++)                  // Read the stream headers
  {
  long bytes=ReadStreamHeader(stream,debug,i);
  block_length-=bytes;
  bytes_read+=bytes;
  }
 if(!CMP4(stream_header.Type,"vids") || !CMP4(stream_header.Handler,"DIB "))
  {
  ExitError(4);                                          // Validate AVI file type
  }
 if(block_length>0L)                                     // Seek past any junk data
  {
  fseek(stream,block_length,SEEK_CUR);
  bytes_read+=block_length;
  }
 return(0);
 }

//-------------------------------------------------------
// Process a RIFF AVI file chunk
//-------------------------------------------------------
int ReadRIFF(FILE *stream,int debug)
 {
 TagT            RIFF_tag;              // RIFF file tag structure
 char            RIFF_type[4];          // RIFF type ID
 long            main_header_length;    // Main RIFF header length

 fread(&RIFF_tag.chunk_type,TAG_SIZE,1,stream);
 fread(&RIFF_type,ID_SIZE,1,stream);
 if(!CMP4(RIFF_tag.chunk_type,"RIFF"))ExitError(0);
 if(!CMP4(RIFF_type,"AVI "))ExitError(1);                // Verify this is AVI file
```

```
ReadHeader(stream,debug);                          // Read the file header

frame_buffer=(BYTE *)malloc(stream_format.SizeImage);
if(frame_buffer==NULL)ExitError(6);

strcpy(vid_header.file_ID,"BOOK_VID");             // Set file ID
vid_header.compression=0;                          // Write uncompressed file

vid_header.frames=(int)main_header.TotalFrames;
vid_header.x_size=(int)main_header.Width;
vid_header.y_size=(int)main_header.Height;

for(int i=0;i<stream_format.ClrUsed;i++)           // Use palette from AVI file
  {
  vid_header.palette[i][0]=(int)palette[i].Red>>2;
  vid_header.palette[i][1]=(int)palette[i].Green>>2;
  vid_header.palette[i][2]=(int)palette[i].Blue>>2;
  }

if(debug==0)ReadFrames(stream);
return(0);
}

//-------------------------------------------------------------
// Main Program
//-------------------------------------------------------------
void main(int argc, char* argv[])
 {
 FILE              *fi,*fo;
 int               debug=0;

 printf("AVI to VID file conversion program.\n\n");
 if(argc<3)                                        // Validate command line parameters
   {
   printf("Please provide two file names: <input_file> <output_file>\n");
   printf("followed by 'd' for a dump of AVI file header information. \n");
   printf(" \n");
   exit(0);
   }
 fi=fopen(argv[1],"rb");
 fo=fopen(argv[2],"wb");
 if(fi==NULL)
   {
   printf("Could not open input file %s.\n",argv[1]);
```

```
  exit(0);
  }
if(fo==NULL)
  {
  printf("Could not open output file %s.\n",argv[2]);
  exit(0);
  }
output=fo;                              // Use global output pointer

if(argc==4 && argv[3][0]=='d')debug=1;  // Set debug flag
if(argc==4 && argv[3][0]=='D')debug=1;

ReadRIFF(fi,debug);

fclose(fi);
fclose(fo);
printf("Done.\n");
}                                       // **** End of Main Program ****
```

ภาคผนวก ข

แสดงโปรแกรม AVIRIFF.H
เป็นโครงสร้างข้อมูลและค่าคงที่ที่ใช้ในโปรแกรม AVI2VID.EXE

```
//=========================================================
//
// AVI_RIFF.H
//
//---------------------------------------------------------
// Data Structures and constants for AVI2VID.EXE
//---------------------------------------------------------

#define BYTE            unsigned char           // 1 byte
#define WORD            unsigned int            // 2 bytes
#define DWORD           unsigned long           // 4 bytes

#define ESC_KEY         27
#define IMAGE_SIZE      19200                   // 160x120 pixels
#define TAG_SIZE        sizeof(TagT)            //  8 bytes
#define MAIN_HDR_SIZE   sizeof(MainAVIHeader)   // 56 bytes
#define STR_HDR_SIZE    sizeof(AVIStreamHeader) // 56 bytes
#define DIB_HDR_SIZE    sizeof(BitMapInfoHeader) // 40 bytes
#define ID_SIZE         4
#define HDR_SIZE        12

#define COMPRESSED      1
#define MONOCHROME      2
#define YCRCB           4
#define COMPRESSED_X    6
#define MONOCHROME_X    5
#define YCRCB_X         3

// flags for use in <dwFlags> in AVIFileHdr
#define RIFF_HASINDEX           0x00000010      // Index at end of file
#define RIFF_MUSTUSEINDEX       0x00000020
#define RIFF_ISINTERLEAVED      0x00000100
#define RIFF_COPYRIGHTED        0x00020000

// flags for use in <dwFlags> in AVIStreamHdr
#define AVI_DISABLED            0x00000001      // User rendering control
#define AVI_VIDEO_PALCHANGES    0x00010000      // Embedded palette changes


typedef struct          // RIFF tag structure
 {
 char           chunk_type[4];
 long           chunk_size;
 } TagT;
```

```
typedef struct
 {
 char            JUNK_type[4];          // JUNK chunk tag
 long            JUNK_size;             // JUNK type ID
 } TagJ;

typedef struct          // Main AVI File Header
 {
 DWORD           MicroSecPerFrame;      // frame display rate (or 0L)
 DWORD           MaxBytesPerSec;        // max. transfer rate
 DWORD           Reserved1;
 DWORD           Flags;                 // RIFF flags
 DWORD           TotalFrames;           // # frames in file
 DWORD           InitialFrames;         // For interleaved files
 DWORD           Streams;               // Stream count
 DWORD           SuggestedBufferSize;   // Chunk buffer size
 DWORD           Width;                 // Frame Width
 DWORD           Height;                // Frame Height
 DWORD           Reserved[4];           // Scale, Rate, Start Time, Sequence Length
 } MainAVIHeader;

typedef struct          // Stream header
 {
 char            Type[4];               // Should be 'vids'
 char            Handler[4];            // Driver code
 DWORD           Flags;                 // Contains AVI_* flags
 DWORD           Priority;
 DWORD           InitialFrames;
 DWORD           Scale;
 DWORD           Rate;                  // dwRate/dwScale == samples/second
 DWORD           Start;
 DWORD           Length;
 DWORD           SuggestedBufferSize;
 DWORD           Quality;
 DWORD           SampleSize;
 DWORD           Reserved[2];           // Supplements fullfilled up to 56 bytes
 } AVIStreamHeader;

typedef struct          // Bitmap Information Header
 {
 DWORD           Size;                  // Size of this structure
 long            Width;                 // Bitmap width
 long            Height;                // Bitmap Height
 WORD            Planes;                // Bitmap bit planes
 WORD            BitCount;              // Bits per pixel
 DWORD           Compression;           // Compression type
```

```
DWORD       SizeImage;          // # of bytes in image
long        XPelsPerMeter;      // H resolution
long        YPelsPerMeter;      // V resolution
DWORD       ClrUsed;            // # of palette entrys used; 0=all
DWORD       ClrImportant;       // # of important entrys
} BitMapInfoHeader;

typedef struct          // RGB Quad Structure
{
BYTE        Blue;
BYTE        Green;
BYTE        Red;
BYTE        Reserved;
} RGBQuad;

typedef struct          // Bitmap Structure for Stream Format Header
{
BitMapInfoHeader    Header;
RGBQuad             Colors[256];
} BitMapInfo;

typedef struct          // VID file header structure
{
char        file_ID[8];
int         x_size;
int         y_size;
int         frames;
int         compression;
int         palette[240][3];
} vid;
```

ภาคผนวก ค

แสดงโปรแกรม VID_VIEW.CPP

โปรแกรมหลักในการแสดงภาพจากไฟล์รูปแบบ VID

```
//=============================================================================
// VID_VIEW.CPP
//=============================================================================

#define   APP_TITLE         "VID File Viewer"

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#include "vid.h"

extern "C" void SetVmode(int);
extern "C" void WriteBlock(int,int,int,int,unsigned char far *);
extern "C" void DrawBlock(int,int,int,int,int);
extern "C" void DrawASCII(int,int,int,int);
extern "C" void WriteEntry(int,int,int,int);

header_t  input;                  // File header input structure
DWORD     f_file_pos;             // File seek pointer
DWORD     f_start_pos;            // File seek pointer
FILE      *fi;                    // File stream pointer
long      previous_time;          // Time of last frame display
int       f_count;                // Number of files in list
int       f_play;                 // File play flag
int       f_play_fast;            // File play fast forward flag
int       f_frame;                // Video frame counter
int       x_size,y_size;          // Video frame size
int       frame_count;            // Video frame count
int       compression;            // Compression flag
int       block_size;             // Compressed block size
int       block_number;           // Compressed block count
int       headerOK;               // Return flag
int       x_pos,y_pos;            // Position of String

BYTE      image_buffer[IMAGE_SIZE];
BYTE      data_buffer[IMAGE_SIZE];

//-----------------------------------------------------------------
// Draw String
//-----------------------------------------------------------------
void DrawString(int x_pos,int y_pos,char *string,int color)
 {
 while(*string!=0 && x_pos<312)
```

```
    {
    DrawASCII(x_pos,y_pos,(int)*string,color);
    x_pos+=8;
    string++;
    }
  }

 //-------------------------------------------------------------
 // Draw Program Screen
 //-------------------------------------------------------------
 void SignOn(void)
  {
  DrawBlock(0,0,320,200,DARKGRAY);                    // Gray background
  DrawBlock(0,0,320,10,LIGHTBLUE);                    // Title Bar
  DrawString(160-strlen(APP_TITLE)*4,2,APP_TITLE,WHITE);   // Title

  DrawBlock(0,186,320,14,LIGHTGRAY);                  // Message Bar
  DrawBlock(2,198,220,1,WHITE);
  DrawBlock(222,187,1,12,WHITE);
  DrawBlock(2,187,221,1,DARKGRAY);
  DrawBlock(2,187,1,12,DARKGRAY);
  }

 //-------------------------------------------------------------
 // Create Control Button
 //-------------------------------------------------------------
 void CreateButton(int x_pos,int y_pos,int x_size,int y_size, char *label)
  {
  DrawBlock(x_pos,y_pos,x_size,y_size,LIGHTGRAY);
  DrawBlock(x_pos,y_pos+y_size,x_size+1,1,DARKGRAY);
  DrawBlock(x_pos+x_size,y_pos,1,y_size,DARKGRAY);
  DrawBlock(x_pos,y_pos,x_size,1,WHITE);
  DrawBlock(x_pos,y_pos,1,y_size,WHITE);
  DrawString(1+x_pos+x_size/2-strlen(label)*4,1+y_pos+y_size/2-4,label,WHITE);
  }

 //-------------------------------------------------------------
 // Display a message in the message bar
 //-------------------------------------------------------------
 void UserMessage(char *message)              // Display a message on the screen
  {
  DrawBlock(3,189,218,10,LIGHTGRAY);
  DrawString(6,189,message,WHITE);
  DrawBlock(2,198,220,1,WHITE);
  }
```

```
//-----------------------------------------------------------------
// Compare 4 characters
//-----------------------------------------------------------------
int Compare4C(char *s1, char *s2)              // Compare a pair of 4-char tags
 {
 if(s1[0]==s2[0] && s1[1]==s2[1] && s1[2]==s2[2] && s1[3]==s2[3])return(1);
 return(0);
 }

//-----------------------------------------------------------------
// Display the next frame in a video sequence
//-----------------------------------------------------------------
void ShowNextFrame()                    // Show the next frame in a video sequence
 {
 int     buffer_p;          // Pointer to compressed data buffer
 int     image_p;           // Pointer to decompressed image frame
 BYTE    pixel;             // Pixel storage
 BYTE    row;               // Pixel row counter

 if((compression&1)==1 && f_frame>0)          // 8-bit color index, compressed
  {
  fread(&block_number,sizeof(int),1,fi);
  if(block_number!=f_frame)
   {
   f_play=FALSE;
   f_play_fast=FALSE;
   f_frame=0;
   }
  else
   {
   fread(&block_size,sizeof(int),1,fi);
   fread(data_buffer,block_size,1,fi);
   buffer_p=0;
   image_p=0;
   while(image_p<19200)
        {
        pixel=data_buffer[buffer_p++];
        if(pixel==0)image_p+=(int)data_buffer[buffer_p++];
        else image_buffer[image_p++]=pixel;
        }
   }
  WriteBlock(SCREEN_POS,x_size,y_size,(char far *)image_buffer);
  f_frame++;
  }
 else
```

```
  {
  fread(image_buffer,19200,1,fi);      // Read image
  }
 WriteBlock(SCREEN_POS.x_size,y_size,(char far *)image_buffer);
 f_frame++;

 if(f_play_fast && f_frame<frame_count-1)
  {
  fseek(fi,19200,SEEK_CUR);
  fseek(fi,19200,SEEK_CUR);
  f_frame+=2;
  }

 if(f_frame>=frame_count)UserMessage("End of Video.");
 }

//---------------------------------------------------------------------
// Read the header information from a VID file
//---------------------------------------------------------------------
int ReadHeaderInfo()                  // Read video information file header
 {
 fpos_t   filepos;                    // File position pointer
 int      i;
 int      Red,Green,Blue;

 fread(input.buffer,sizeof(vid_t),1,fi);
 if(!Compare4C(input.vid_header.file_ID,"BOOK"))return(0);

for(i=0;i<240;i++)  // Set palette for this video
  {
  Red=  (int)input.vid_header.palette[i][0];
  Green=(int)input.vid_header.palette[i][1];
  Blue= (int)input.vid_header.palette[i][2];
  WriteEntry(i+16,Red,Green,Blue);
  }
 x_size=input.vid_header.x_size;
 y_size=input.vid_header.y_size;
 frame_count=input.vid_header.frames;
 compression=input.vid_header.compression;

 fgetpos(fi,&filepos);
 f_start_pos=(DWORD)filepos;          // Set start position for rewind
 f_frame=0;
 return(1);
 }
```

```
//---------------------------------------------------------------
// Rewind the video sequence to the first frame
//---------------------------------------------------------------

void Rewind()                   // User Pressed the REWIND button
 {
 f_frame=0;
 f_play=FALSE;
 f_play_fast=FALSE;
 fseek(fi,f_start_pos,SEEK_SET);
 ShowNextFrame();
 UserMessage("Video Rewound.");
 }

//---------------------------------------------------------------
// Play the video sequence
//---------------------------------------------------------------
void Play()                     // User Pressed the PLAY button
 {
 char ch;

 previous_time=0;
 while(f_play && f_frame<frame_count)
  {
  long current_time=clock(); // Clock tick 18.2 per sec.
  if(f_play_fast || current_time>previous_time)
    {
    previous_time=current_time;
    UserMessage("Playing Video.");
    ShowNextFrame();
    if(kbhit())
      {
      ch = getch();
      switch(ch)
        {
        case 'p': {
                  f_play=TRUE;
                  f_play_fast=FALSE;
                  Play();
                  } break;
        case 'P': {
                  f_play=TRUE;
                  f_play_fast=FALSE;
                  Play();
                  } break;
```

```
      case 'f': {
                f_play=TRUE;
                f_play_fast=TRUE;
                Play();
                } break;
      case 'F': {
                f_play=TRUE;
                f_play_fast=TRUE;
                Play();
                } break;
      case 'r': {
                f_play=FALSE;
                f_play_fast=FALSE;
                Rewind();
                } break;
      case 'R': {
                f_play=FALSE;
                f_play_fast=FALSE;
                Rewind();
                } break;
      case 's': {
                f_play=FALSE;
                UserMessage("Play Stopped.");
                } break;
      case 'S': {
                f_play=FALSE;
                UserMessage("Play Stopped.");
                } break;
      case 'q': ch=ESC_KEY; break;
      case 'Q': ch=ESC_KEY; break;
      default: ;
      }
     }
    }
   }
  }

//*************************************************************
void main(int argc, char* argv[])          // Start of Main Program
 {
 char  ch;

 if(argc!=2)                               // Validate command line parameters
  {
  printf("VID file Viewer Program.\n\n");
```

```
  printf("Please provide the input VID file name: <SRC_FILE.VID>\n");
  printf("Syntax : VID_PLAY  <SRC_FILE.VID>\n\n");
  exit(0);
  }

 fi=fopen(argv[1],"rb");

 if(fi==NULL)
  {
  printf("VID file Viewer Program.\n\n");
  printf("Could not open input file \"%s\".\n\n",argv[1]);
  printf("Please provide the input VID file name: <SRC_FILE.VID>\n");
  printf("Syntax : VID_PLAY  <SRC_FILE.VID>\n\n");
  exit(0);
  }

 SetVmode(0x13);
 SignOn();

 DrawBlock(FRAME_POS,FRAME_SIZE,LIGHTGRAY);      // Control focus area
 DrawBlock(SCREEN_POS,SCREEN_SIZE,BLACK);        // Screen area

 CreateButton(POS_STOP,SIZE_CTRL,"STOP");
 CreateButton(POS_REW,SIZE_CTRL,"REW");
 CreateButton(POS_FF,SIZE_CTRL,"FF");
 CreateButton(POS_PLAY,SIZE_CTRL,"PLAY");
 CreateButton(POS_QUIT,SIZE_QUIT,"[Esc] Quit");

 headerOK=ReadHeaderInfo();
 if(!headerOK)
  {
  DrawString(LIST_BAR_XPOS,LIST_BAR_YPOS,"File Error",WHITE);
  getch();
  SetVmode(3);
  exit(0);
  }
 else
  {
  ShowNextFrame();                    // Show frame 1 of selected file
  UserMessage("File Opened.");
  }

 f_play=FALSE;                        // Initialize control flags
 f_play_fast=FALSE;
```

```
do
 {
 ch=getch();
 switch(ch)
  {
  case 'p': {
                    f_play=TRUE;
                    f_play_fast=FALSE;
                    Play();
                    } break;
  case 'P': {
                    f_play=TRUE;
                    f_play_fast=FALSE;
                    Play();
                    } break;
  case 'f': {
                    f_play=TRUE;
                    f_play_fast=TRUE;
                    Play();
                    } break;
  case 'F': {
                    f_play=TRUE;
                    f_play_fast=TRUE;
                    Play();
                    } break;
  case 'r': {
                    f_play=FALSE;
                    f_play_fast=FALSE;
                    Rewind();
                    } break;
  case 'R': {
                    f_play=FALSE;
                    f_play_fast=FALSE;
                    Rewind();
                    } break;
  case 'q': ch=ESC_KEY; break;
  case 'Q': ch=ESC_KEY; break;
  default: ;
  }
 } while(ch!=ESC_KEY);

SetVmode(3);
fclose(fi);
}                                   // End of Main Program
```

ภาคผนวก ง

แสดงโปรแกรม VID.H

เป็นโครงสร้างข้อมูลและค่าคงที่ที่ใช้ในโปรแกรม VID_VIEW.EXE

```
//=================================================================
// VID.H
//-----------------------------------------------------------------
// Data Structure and Constants for VID_VIEW.EXE
//-----------------------------------------------------------------

#define BYTE        unsigned char
#define WORD        unsigned int
#define DWORD       unsigned long

#define ESC_KEY             27
#define IMAGE_SIZE          19200
#define TRUE                1
#define FALSE               0

// Screen positions for playback window
#define FRAME_POS           75,15
#define FRAME_SIZE          170,145
#define SCREEN_POS          80,20
#define SCREEN_SIZE         160,120
#define POS_PLAY            80,140
#define POS_FF              120,140
#define POS_REW             160,140
#define POS_STOP            200,140
#define POS_QUIT            226,187
#define SIZE_CTRL           39,14
#define SIZE_QUIT           90,11
#define LIST_BAR_XPOS       90
#define LIST_BAR_YPOS       30

// File header structure
typedef struct
 {
 char               file_ID[8];
 int                x_size;
 int                y_size;
 int                frames;
 int                compression;
 int                palette[240][3];
 } vid_t;

typedef union
 {
 vid_t              vid_header;
 unsigned char      buffer[sizeof(vid_t)];
 } header_t;
```

ภาคผนวก จ

แสดงโปรแกรม VID_FUNC.ASM
เป็นฟังก์ชันที่ใช้ในโปรแกรม VID_FUNC.ASM

```
                TITLE  'Functions call for VID_VIEW.EXE'
                .MODEL  small
                .CODE

;=================================================================
;
; Name:         SetVmode 'Set Video Mode'
;
; Function:     Call IBM PC ROM BIOS to set a video display mode.
;
; Caller:  Turbo C:
;
;               void SetVmode(n);
;
;               int n;                  /* video mode */
;
;=================================================================


FLAGS           EQU     byte ptr ds:[10h] ; (in Video Display Data Area)

CGAflag         EQU     00100000b       ; bits for FLAGS
MDAflag         EQU     00110000b

                PUBLIC  _SetVmode
_SetVmode       PROC    NEAR
                ARG     v_mode:WORD

                push    bp              ; preserve caller registers
                mov     bp,sp
                push    ds

                mov     ax,40h
                mov     ds,ax           ; DS -> Display Data Area
                mov     bl,CGAflag      ; BL := Detect CGA
                mov     ax,[v_mode]     ; AL := Mode number
                mov     ah,al           ; Monochrome?
                and     ah,7
                cmp     ah,7
                jne     L00             ; Mode not 7 or 0Fh
                mov     bl,MDAflag      ; Set MDA

L00:            and     FLAGS,11001111b
                or      FLAGS,bl
                xor     ah,ah           ; AH := 0 (INT 10h function number)
                push    bp
```

```
                int     10h             ; Set the video mode
                pop     bp
                pop     ds
                mov     sp,bp
                pop     bp
                ret

_SetVmode       ENDP

;=====================================================================
;
; Name:         DrawBlock 'Draw a Block of Pixels in 256 Color Mode'
;
; Function:     Copy an Image in 320x200 Graphics Mode Frame Buffer
;
; Caller:  Turbo C:
;
;               void DrawBlock(x,y,width,height,color);
;
;               int x;                  /* X Location - Bytes */
;               int y;                  /* Y Location - Lines */
;               int width;              /* Width - Pixels */
;               int height;             /* Height - Pixels */
;               int color;              /* Color Index 0-255 */
;
;=====================================================================

                PUBLIC  _DrawBlock
_DrawBlock      PROC    NEAR
                ARG     x_loc:WORD,y_loc:WORD,x_size:WORD,y_size:WORD,color:WORD

                push    bp              ; preserve caller registers
                mov     bp,sp
                push    ds
                push    es
                push    si
                push    di
                push    ax
                push    bx
                push    cx
                push    dx

                mov     ax,320
                mul     [y_loc]
```

```
                add     ax,[x_loc]          ; Calculate Frame Buffer Start Address
                mov     di,ax               ; DI --> block start
                mov     ax,0a000h
                mov     es,ax               ; Set frame buffer segment (es:di)

; routine for byte-aligned bit blocks

                mov     ax,[x_size]
                mov     cx,ax
                mov     ax,[y_size]
                mov     bx,ax
                mov     ax,[color]
                cld                         ; set direction forward
L10:            push    di                  ; preserve Destination
                push    cx                  ; preserve width

L11:            mov     es:[di],al          ; update display memory
                inc     di                  ; increment output pointer
                loop    L11

                pop     cx
                pop     di
                add     di,320              ; ES:DI -> next pixel row
                dec     bx                  ; Decrement row count
                jnz     L10                 ; loop down pixel rows

                pop     dx
                pop     cx
                pop     bx
                pop     ax
                pop     di                  ; restore registers and exit
                pop     si
                pop     es
                pop     ds
                mov     sp,bp
                pop     bp
                ret

_DrawBlock      ENDP

;=====================================================================
;
; Name:         WriteBlock 'Display Block of Pixels in 256 Color Mode'
;
; Function:     Copy an Image in 320x200 Graphics Mode Frame Buffer
```

```
;
; Caller:   Turbo C:
;
;                   void WriteBlock(x,y,width,height,src);
;
;                   int x;                  /* X Location - Bytes */
;                   int y;                  /* Y Location - Lines */
;                   int width;              /* Width - Pixels */
;                   int height;             /* Height - Pixels */
;                   unsigned char far *src  /* Pointer to Source Block */
;
;=================================================================

                    PUBLIC  _WriteBlock
_WriteBlock         PROC    NEAR
                    ARG     x_loc:WORD,y_loc:WORD,x_size:WORD,y_size:WORD,src:FAR PTR BYTE

                    push    bp              ; preserve caller registers
                    mov     bp,sp
                    push    ds
                    push    es
                    push    si
                    push    di
                    push    ax
                    push    bx
                    push    cx
                    push    dx

                    mov     ax,320
                    mul     [y_loc]
                    add     ax,[x_loc]      ; Calculate Frame Buffer Start Address
                    mov     di,ax           ; DI --> block start
                    mov     ax,0a000h
                    mov     es,ax           ; Set frame buffer segment (es:di)
                    lds     si,[src]        ; Set source pointer (ds:si)

                    mov     ax,[x_size]
                    mov     cx,ax
                    mov     ax,[y_size]
                    mov     bx,ax
                    cld                     ; set direction forward
L20:                push    di              ; preserve Destination
                    push    cx              ; preserve width
```

```
L21:            lodsb                   ; AL := pixel @DS:SI
                mov     es:[di],al      ; update display memory
                inc     di              ; increment output pointer
                loop    L21

                pop     cx
                pop     di
                add     di,320          ; ES:DI -> next pixel row
                dec     bx              ; Decrement row count
                jnz     L20             ; loop down pixel rows

                pop     dx
                pop     cx
                pop     bx
                pop     ax
                pop     di              ; restore registers and exit
                pop     si
                pop     es
                pop     ds
                mov     sp,bp
                pop     bp
                ret

_WriteBlock     ENDP


;=====================================================================
;
; Name:         DrawASCII 'Draw an ASCII character in 256 Color Mode'
;
; Function:     Draw an ASCII character in 256 Color Mode
;
; Caller:   Turbo C:
;
;               void DrawASCII(x,y,char,color);
;
;               int x;                  /* X Location - Bytes */
;               int y;                  /* Y Location - Lines */
;               int char;               /* Character Index 0-127 */
;               int color;              /* Color Index 0-255 */
;
;=====================================================================

                PUBLIC  _DrawASCII
_DrawASCII      PROC    NEAR
```

```
        ARG     x_loc:WORD,y_loc:WORD,char:WORD,color:WORD

        push    bp              ; preserve caller registers
        mov     bp,sp
        push    ds
        push    es
        push    si
        push    di
        push    ax
        push    bx
        push    cx
        push    dx

        mov     ax,320
        mul     [y_loc]
        add     ax,[x_loc]      ; Calculate Frame Buffer Start Address
        mov     di,ax           ; DI --> block start
        mov     ax,0a000h
        mov     es,ax           ; Set frame buffer segment (es:di)

        mov     si,OFFSET _VideoFont
        mov     ax,SEG _VideoFont
        mov     ds,ax           ; DS:SI --> Font Table
        mov     ax,[char]
        and     ax,7fh
        shl     ax,1            ; Calculate font offset
        shl     ax,1
        shl     ax,1
        add     si,ax

; routine for byte-aligned bit blocks

        mov     ax,8
        mov     bx,ax
        mov     cx,ax
        mov     ax,[color]
        xchg    ah,al
        mov     al,[si]         ; Get bit map row
        cld                     ; set direction forward
L30:    push    di              ; preserve Destination
        push    cx              ; preserve width
        push    ax              ; preserve color/bitmap
```

```
L31:    nd      l,80h
        jz      32
        xchg    ah,al
        mov     es:[di],al          ; Update display memory
        xchg    ah,al
L32:    op      x

        shl     al,1
        push    ax
        inc     di                  ; increment output pointer
        loop    31

        pop     ax
        pop     cx
        pop     di
        inc     si
        mov     al,[si]             ; Get next bit map row
        add     di,320              ; ES:DI -> next pixel row
        dec     bx                  ; Decrement row count
        jnz     L30                 ; loop down pixel rows

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     di                  ; restore registers and exit
        pop     si
        pop     es
        pop     ds
        mov     sp,bp
        pop     bp
        ret

_VideoFont      label byte

;       These are the bit-mapped characters for graphics mode.

        db      000h,000h,000h,000h,000h,000h,000h,000h ; 00 - nul
        db      07eh,081h,0a5h,081h,0bdh,099h,081h,07eh ; 01 - ^A
        db      07eh,0ffh,0dbh,0ffh,0c3h,0e7h,0ffh,07eh ; 02 - ^B
        db      06ch,0feh,0feh,0feh,07ch,038h,010h,000h ; 03 - ^C
        db      010h,038h,07ch,0feh,07ch,038h,010h,000h ; 04 - ^D
        db      038h,07ch,038h,0feh,0feh,07ch,038h,07ch ; 05 - ^E
        db      010h,010h,038h,07ch,0feh,07ch,038h,07ch ; 06 - ^F
        db      000h,000h,018h,03ch,03ch,018h,000h,000h ; 07 - ^G
```

```
db      0ffh,0ffh,0e7h,0c3h,0c3h,0e7h,0ffh,0ffh ; 08 - ^H
db      000h,03ch,066h,042h,042h,066h,03ch,000h ; 09 - ^I
db      0ffh,0c3h,099h,0bdh,0bdh,099h,0c3h,0ffh ; 0A - ^J
db      00fh,007h,00fh,07dh,0cch,0cch,0cch,078h ; 0B - ^K
db      03ch,066h,066h,066h,03ch,018h,07eh,018h ; 0C - ^L
db      03fh,033h,03fh,030h,030h,070h,0f0h,0e0h ; 0D - ^M
db      07fh,063h,07fh,063h,06eh,067h,0e6h,0c0h ; 0E - ^N
db      099h,05ah,03ch,0e7h,0e7h,03ch,05ah,099h ; 0F - ^O
db      080h,0e0h,0f8h,0feh,0f8h,0e0h,080h,000h ; 10 - ^P
db      002h,00eh,03eh,0feh,03eh,00eh,002h,000h ; 11 - ^Q
db      018h,03ch,07eh,018h,018h,07eh,03ch,018h ; 12 - ^R
db      066h,066h,066h,066h,066h,000h,066h,000h ; 13 - ^S
db      07fh,0dbh,0dbh,07bh,01bh,01bh,01bh,000h ; 14 - ^T
db      03eh,063h,038h,06ch,06ch,038h,0cch,078h ; 15 - ^U
db      000h,000h,000h,000h,07eh,07eh,07eh,000h ; 16 - ^V
db      018h,03ch,07eh,018h,07eh,03ch,018h,0ffh ; 17 - ^W
db      018h,03ch,07eh,018h,018h,018h,018h,000h ; 18 - ^X
db      018h,018h,018h,018h,07eh,03ch,018h,000h ; 19 - ^Y
db      000h,018h,00ch,0feh,00ch,018h,000h,000h ; 1A - ^Z
db      000h,030h,060h,0feh,060h,030h,000h,000h ; 1B - ^[
db      000h,000h,0c0h,0c0h,0c0h,0feh,000h,000h ; 1C - ^\
db      000h,024h,066h,0ffh,066h,024h,000h,000h ; 1D - ^]
db      000h,018h,03ch,07eh,0ffh,0ffh,000h,000h ; 1E - ^6
db      000h,0ffh,0ffh,07eh,03ch,018h,000h,000h ; 1F - ^-
db      000h,000h,000h,000h,000h,000h,000h,000h ; 20 - spc
db      030h,078h,078h,030h,030h,000h,030h,000h ; 21 - !
db      06ch,06ch,06ch,000h,000h,000h,000h,000h ; 22 - "
db      06ch,06ch,0feh,06ch,0feh,06ch,06ch,000h ; 23 - #
db      030h,07ch,0c0h,078h,00ch,0f8h,030h,000h ; 24 - $
db      000h,0c6h,0cch,018h,030h,066h,0c6h,000h ; 25 - %
db      038h,06ch,038h,076h,0dch,0cch,076h,000h ; 26 - &
db      060h,060h,0c0h,000h,000h,000h,000h,000h ; 27 - '
db      018h,030h,060h,060h,060h,030h,018h,000h ; 28 - (
db      060h,030h,018h,018h,018h,030h,060h,000h ; 29 - )
db      000h,066h,03ch,0ffh,03ch,066h,000h,000h ; 2A - *
db      000h,030h,030h,0fch,030h,030h,000h,000h ; 2B - +
db      000h,000h,000h,000h,000h,030h,030h,060h ; 2C - ,
db      000h,000h,000h,0fch,000h,000h,000h,000h ; 2D - -
db      000h,000h,000h,000h,000h,030h,030h,000h ; 2E - .
db      006h,00ch,018h,030h,060h,0c0h,080h,000h ; 2F - /
db      07ch,0c6h,0ceh,0deh,0f6h,0e6h,07ch,000h ; 30 - 0
db      030h,070h,030h,030h,030h,030h,0fch,000h ; 31 - 1
db      078h,0cch,00ch,038h,060h,0cch,0fch,000h ; 32 - 2
db      078h,0cch,00ch,038h,00ch,0cch,078h,000h ; 33 - 3
db      01ch,03ch,06ch,0cch,0feh,00ch,01eh,000h ; 34 - 4
```

```
db      0fch,0c0h,0f8h,00ch,00ch,0cch,078h,000h ; 35 - 5
db      038h,060h,0c0h,0f8h,0cch,0cch,078h,000h ; 36 - 6
db      0fch,0cch,00ch,018h,030h,030h,030h,000h ; 37 - 7
db      078h,0cch,0cch,078h,0cch,0cch,078h,000h ; 38 - 8
db      078h,0cch,0cch,07ch,00ch,018h,070h,000h ; 39 - 9
db      000h,030h,030h,000h,000h,030h,030h,000h ; 3A - :
db      000h,030h,030h,000h,000h,030h,030h,060h ; 3B - ;
db      018h,030h,060h,0c0h,060h,030h,018h,000h ; 3C - <
db      000h,000h,0fch,000h,000h,0fch,000h,000h ; 3D - =
db      060h,030h,018h,00ch,018h,030h,060h,000h ; 3E - >
db      078h,0cch,00ch,018h,030h,000h,030h,000h ; 3F - ?
db      07ch,0c6h,0deh,0deh,0deh,0c0h,078h,000h ; 40 - @
db      030h,078h,0cch,0cch,0fch,0cch,0cch,000h ; 41 - A
db      0fch,066h,066h,07ch,066h,066h,0fch,000h ; 42 - B
db      03ch,066h,0c0h,0c0h,0c0h,066h,03ch,000h ; 43 - C
db      0f8h,06ch,066h,066h,066h,06ch,0f8h,000h ; 44 - D
db      0feh,062h,068h,078h,068h,062h,0feh,000h ; 45 - E
db      0feh,062h,068h,078h,068h,060h,0f0h,000h ; 46 - F
db      03ch,066h,0c0h,0c0h,0ceh,066h,03eh,000h ; 47 - G
db      0cch,0cch,0cch,0fch,0cch,0cch,0cch,000h ; 48 - H
db      078h,030h,030h,030h,030h,030h,078h,000h ; 49 - I
db      01eh,00ch,00ch,00ch,0cch,0cch,078h,000h ; 4A - J
db      0e6h,066h,06ch,078h,06ch,066h,0e6h,000h ; 4B - K
db      0f0h,060h,060h,060h,062h,066h,0feh,000h ; 4C - L
db      0c6h,0eeh,0feh,0feh,0d6h,0c6h,0c6h,000h ; 4D - M
db      0c6h,0e6h,0f6h,0deh,0ceh,0c6h,0c6h,000h ; 4E - N
db      038h,06ch,0c6h,0c6h,0c6h,06ch,038h,000h ; 4F - O
db      0fch,066h,066h,07ch,060h,060h,0f0h,000h ; 50 - P
db      078h,0cch,0cch,0cch,0dch,078h,01ch,000h ; 51 - Q
db      0fch,066h,066h,07ch,06ch,066h,0e6h,000h ; 52 - R
db      078h,0cch,0e0h,070h,01ch,0cch,078h,000h ; 53 - S
db      0fch,0b4h,030h,030h,030h,030h,078h,000h ; 54 - T
db      0cch,0cch,0cch,0cch,0cch,0cch,0fch,000h ; 55 - U
db      0cch,0cch,0cch,0cch,0cch,078h,030h,000h ; 56 - V
db      0c6h,0c6h,0c6h,0d6h,0feh,0eeh,0c6h,000h ; 57 - W
db      0c6h,0c6h,06ch,038h,038h,06ch,0c6h,000h ; 58 - X
db      0cch,0cch,0cch,078h,030h,030h,078h,000h ; 59 - Y
db      0feh,0c6h,08ch,018h,032h,066h,0feh,000h ; 5A - Z
db      078h,060h,060h,060h,060h,060h,078h,000h ; 5B - [
db      0c0h,060h,030h,018h,00ch,006h,002h,000h ; 5C - \
db      078h,018h,018h,018h,018h,018h,078h,000h ; 5D - ]
db      010h,038h,06ch,0c6h,000h,000h,000h,000h ; 5E - ^
db      000h,000h,000h,000h,000h,000h,000h,0ffh ; 5F - _
db      030h,030h,018h,000h,000h,000h,000h,000h ; 60 - `
db      000h,000h,078h,00ch,07ch,0cch,076h,000h ; 61 - a
```

```
        db      0e0h,060h,060h,07ch,066h,066h,0dch,000h ; 62 - b
        db      000h,000h,078h,0cch,0c0h,0cch,078h,000h ; 63 - c
        db      01ch,00ch,00ch,07ch,0cch,0cch,076h,000h ; 64 - d
        db      000h,000h,078h,0cch,0fch,0c0h,078h,000h ; 65 - e
        db      038h,06ch,060h,0f0h,060h,060h,0f0h,000h ; 66 - f
        db      000h,000h,076h,0cch,0cch,07ch,00ch,0f8h ; 67 - g
        db      0e0h,060h,06ch,076h,066h,066h,0e6h,000h ; 68 - h
        db      030h,000h,070h,030h,030h,030h,078h,000h ; 69 - i
        db      00ch,000h,00ch,00ch,00ch,0cch,0cch,078h ; 6A - j
        db      0e0h,060h,066h,06ch,078h,06ch,0e6h,000h ; 6B - k
        db      070h,030h,030h,030h,030h,030h,078h,000h ; 6C - l
        db      000h,000h,0cch,0feh,0feh,0d6h,0c6h,000h ; 6D - m
        db      000h,000h,0f8h,0cch,0cch,0cch,0cch,000h ; 6E - n
        db      000h,000h,078h,0cch,0cch,0cch,078h,000h ; 6F - o
        db      000h,000h,0dch,066h,066h,07ch,060h,0f0h ; 70 - p
        db      000h,000h,076h,0cch,0cch,07ch,00ch,01eh ; 71 - q
        db      000h,000h,0dch,076h,066h,060h,0f0h,000h ; 72 - r
        db      000h,000h,07ch,0c0h,078h,00ch,0f8h,000h ; 73 - s
        db      010h,030h,07ch,030h,030h,034h,018h,000h ; 74 - t
        db      000h,000h,0cch,0cch,0cch,0cch,076h,000h ; 75 - u
        db      000h,000h,0cch,0cch,0cch,078h,030h,000h ; 76 - v
        db      000h,000h,0c6h,0d6h,0feh,0feh,06ch,000h ; 77 - w
        db      000h,000h,0c6h,06ch,038h,06ch,0c6h,000h ; 78 - x
        db      000h,000h,0cch,0cch,0cch,07ch,00ch,0f8h ; 79 - y
        db      000h,000h,0fch,098h,030h,064h,0fch,000h ; 7A - z
        db      01ch,030h,030h,0e0h,030h,030h,01ch,000h ; 7B - {
        db      018h,018h,018h,000h,018h,018h,018h,000h ; 7C - |
        db      0e0h,030h,030h,01ch,030h,030h,0e0h,000h ; 7D - }
        db      076h,0dch,000h,000h,000h,000h,000h,000h ; 7E - ~
        db      000h,010h,038h,06ch,0c6h,0c6h,0feh,000h ; 7F -

_DrawASCII      ENDP

PelMask     EQU     3c6h
PelWritePort EQU    3c8h
PelData     EQU     3c9h

;=========================================================================
;
; Name:             WriteEntry
;
; Function:         Copy Data Into 256-Color Palette
;
; Caller:   Turbo C:
;
```

```
;                   void WriteEntry(windex,r,g,b);
;
;                   int windex;                 /* Palette Index */
;                   int r,g,b;                  /* Palette Data */
;
;=====================================================================

                    PUBLIC  _WriteEntry
_WriteEntry         PROC    NEAR
                    ARG     windex:WORD,rdata:WORD,gdata:WORD,bdata:WORD

                    push    bp                  ; preserve caller registers
                    mov     bp,sp
                    push    ds
                    push    es
                    push    si
                    push    di
                    push    ax
                    push    bx
                    push    cx
                    push    dx

                    mov     dx,PelMask
                    mov     al,0ffh
                    out     dx,al               ; Enable all bit planes
                    mov     dx,PelWritePort
                    mov     ax,[windex]
                    out     dx,al               ; Point to color @windex
                    mov     dx,PelData
                    cli
                    mov     ax,[rdata]
                    out     dx,al               ; Load palette data
                    mov     ax,[gdata]
                    out     dx,al               ; Load palette data
                    mov     ax,[bdata]
                    out     dx,al               ; Load palette data
                    sti

                    pop     dx
                    pop     cx
                    pop     bx
                    pop     ax
                    pop     di                  ; restore registers and exit
                    pop     si
                    pop     es
```

```
                pop     ds
                mov     sp,bp
                pop     bp
                ret

_WriteEntry     ENDP

                END
```

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์สุวิพล สิทธิชีวภาค ที่ช่วยให้คำปรึกษาแนะนำ ตลอดจนให้การช่วยเหลือด้วยดีเรื่อยมา

ขอขอบคุณพี่ ๆ และ เพื่อน ๆ ที่ให้ยืมอุปกรณ์ในการทำโครงงาน ให้คำปรึกษาแนะนำ และให้กำลังใจตลอดมา

เอกสารอ้างอิง

1. Phillip E. Mattison , " Practical Digital Video Programming with Examples in C ", John Wiley & Sons, Inc.,1994.

2. Microsoft Corperation., " Help File from Video for Windows Development Kit ".