Quality-aware frame skipping for MPEG-2 video based on inter-frame similarity

Hassan Bin Tariq Khan
hkn05002@student.mdh.se

Malik Khurram Anwar
kar05001@student.mdh.se

The Department of Computer Science and Electronics
Mälardalen University, Västerås, Sweden

Supervisor and Examiner: Damir Isovic
damir.isovic@mdh.se

MÄLARDALEN UNIVERSITY
# Table of contents

Table of contents

Acknowledgement

Abstract

Introduction

What is Video?

Video Stream Characteristics

| FRAME RATE | 8 |
| VIDEO RESOLUTION | 8 |
| INTERLACING | 8 |
| COLOR SPACE | 9 |
| BIT RATE (BR) | 9 |

Video Compression

| DEPARTMENT OF REDUNDANCY DEPARTMENT | 11 |
| RUN LENGTH CODING | 11 |
| VARIABLE LENGTH CODING | 11 |
| DIFFERENTIAL CODING | 11 |
| VECTOR CODING | 12 |
| QUANTIZATION | 12 |
| TRANSFORMS | 12 |
| TEMPORAL COMPRESSION | 12 |
| SCALING | 13 |
| BLOCKING AND REORDERING | 13 |
| MOTION COMPENSATION | 13 |
| CONSTANT/ VARIABLE BIT RATE ENCODING | 13 |

Video Compression Standards

| JPEG | 14 |
| MPEG | 14 |
| H.261/H.263 | 14 |
| DV | 14 |
| HDV | 15 |

Introduction to MPEG | 15 |
Experiments and Results 36

GRAPH OF TEST1: 37
GRAPH OF TEST2: 39
GRAPH OF TEST3: 41

Screen Shots of input and output files 43

Conclusion and Future Work 44

References 45

Appendix 46

CODE IN MPEG2EVENT (MACROBLOCK MODE) 46
CODE IN MPEG TRANSCODER OF MPEG_STREAM_CREATOR 52
CODE IN MPEG TRANSCODER OF FILE.C 58
OUTPUT OF TEST1 FROM MPEG2EVENT 60
OUTPUT OF TEST2 FROM MPEG2EVENT 61
OUTPUT OF TEST3 FROM MPEG2EVENT 62
Acknowledgement

Firstly we would like to thanks our beloved Allah (God) Who has given us enough knowledge to come to this stage and to contribute our efforts in a vast field of Information Technology.

Secondly, we would like to thanks to our supervisor Damir Isovic for his support and guidance during the entire period of our master’s thesis. He helped and encouraged us not only by suggestions but he also helped us by giving his worth time by explaining the examples related to our thesis.

We are really thankful to Department of Computer Science and Electronics for providing us special thesis room with latest computers which gave us opportunity to complete our final thesis successfully without any hurdle.

Lastly but not least we are really thankful to our parents, family and friends for moral support.

Hassan Bin Tariq Khan
Malik Khurram Anwar

September 26, 2006, Västerås, Sweden.
Abstract

Frame skipping is a method to adapt video streams to the fluctuations of available system resources. If frames are skipped properly, a high video quality can be provided while achieving good resource utilization. However, frame skipping needs appropriate assumptions about the video stream to be effective. Skipping the wrong frame at the wrong time can result in a noticeable disturbance in the played video stream.

This thesis extends the quality-aware frame skipping method for MPEG decoding under limited resources, developed by the research group that provided the thesis. Given that not all frames can be processed, it selects those which will provide the best picture quality while matching the available resources. However, the frame priority assignment algorithm of the previous method is based on structural approach, which means that it does not examine the contents of the frame while dropping frames. Therefore, the objective of this thesis was to develop an approach which considers information contained in a frame when selecting frames, and to apply it on the top of the previous method.

As a first step, we have performed a proper analysis of MPEG-2 video on sub-frame level to gain a deeper understanding of MPEG-2 coding. We have looked into the bitstream organization (i.e., slices, macroblocks and blocks), of diverse video streams to identify the most redundant picture elements which are to be skipped first upon overload situations. Then, we designed an algorithm that compares two frames on sub-frame level and decides which one is more important for the overall video quality as perceived by the end user. Finally, we have implemented the proposed method in a MPEG simulator and deployed it in a MPEG transcoder to obtain tailored MPEG-2 video streams that corresponds available system resources. Although we perform offline video transformation in this thesis, the work presented here can fairly easy be applied for online video transcoding at run-time.
Introduction

Advancement in computer, telecommunication and electronics has brought a wide vision in the term of multimedia. Multimedia is expanding with its features and advancement in its technology very rapidly. We are having more and more multimedia resources and applications daily on internet, our homes, cars, etc. Our normal TV broadcast is moving to digital era to have better results.

With this rapidly growth of multimedia, there are some problems too: most common are like transmitting data on internet, getting video broadcast on low resources systems and having better quality of video for web broadcast etc. Mostly these kinds of problems are because of size and complexity of the video.

Here we got term of compression. To minimize the size of a multimedia file, we need to compress the file before transmitting it and then need to decompress it before displaying it to the viewer. The hardware/software we use for compression is known as encoder while for decompression we use decoder.

Our purposed work in this paper is also about compression, but the compression of the totally processed video data, i.e., we decrease the load on the decoder by skipping frames upon overload situations. The main purpose of this thesis is to skip frames from an encoded video stream without disturbing the quality of video and get best video sequences to display after frame skipping. To get better approach for frame skipping, we have studied internal structure of frames deeply because we need to know all appropriate information about frames to select best one for skipping from the video stream.

We are also going to give introduction about video and its different characteristics. It’ll give better understanding of our work to whom who don’t have much information about video applications and quality etc. Then we’ll explain compression later in the paper with different methods of compression. After this basic information we’ll go to original problem and our work on that.
What is Video?

The word video or film is another name of moving pictures. In a video we display set of pictures or frames rapidly with a specific speed called Frame rate (frames per second). There are many other terms attached with this word such as recording and storing a video or capturing or transmitting etc. And we can view these videos then on television or our monitor screens. We are going to explain some characteristics of video to have its better understanding.

Video Stream Characteristics

Frame Rate

As mentioned above, frame rate means the number of frames per unit time of a video. There is a wide range of frame rate depending on the devices e.g. the frame rate for the latest professional cameras is almost 120 frames per second (fps) but it was just 6 or 8 fps in old mechanical cameras. There are different systems according to the frame rate on the face of globe i.e. phase alternation line (PAL), sequential color with memory (SECAM), National Television System(s) Committee (NTSC). PAL and SECAM are working on 25 fps while NTSC on 30 fps.

Video Resolution

Video resolution of a video represents size of that video and we measure it with Pixel (picture element) in 2D digital videos and for analog videos we calculate resolution with horizontal scan lines. Video resolution for the NTSC is 720/640 x 480i60 shows 720 or 640 pixels in each horizontal line and 480 horizontal lines, i means interlace and 60 fps for PAL and SECAM its 768/720 x 576i50. In latest high-definition televisions (HDTV) the video resolution is 1920×1080p60 where p stands for progressive.

For 3D-video the unit for measuring video resolution is Voxels (volume picture element). The most common resolution in use at the moment is 512 x 512 x 512 voxels. Higher resolution means high video quality.

Interlacing

Interlacing is a method to achieve good visual quality. In this scanning method, we give consecutive numbers to each horizontal scan lines and then divide them in two different fields; one is called odd field and other even field. This scanning technique will display one set of lines first e.g. odd field and on next scan it’ll display even field in the gaps developed by odd scan. But this can be cause of flickering.

The other scanning form is progressive scanning. In this type all horizontal line of a frame are included in scan.
Color space

There are different kinds of color model for videos and each model name describes the video color for specific device. The most popular color scheme is RGB (Red, Green and Blue). There are some other color models like YUV where Y is representing luminance while U and V is for difference in color. Another model is YIQ where I stand for in-phase and Q for quadrature. NTSC is using YIQ color model while PAL is using YUV.

Number of colors a pixel can represent depends on the number of bits per pixel (bpp).

Bit Rate (BR)

Numbers of bits that are processed per unit time are known as bit rate in computing. But more specifically in digital multimedia, we can say that the number of bits used in per unit time to represent a video or audio is known as bit rate. Higher bit rate will give higher video quality e.g. a VCD (Video Compact Disk) with the bit rate 1 Mbit/s has lower quality than a DVD (Digital Versatile Disk) with bit rate of 5 Mbits/s. There are two different types of bit rate.

- **Variable Bit Rate (VBR):** In any video section, there are different kinds of scenes. They can vary from very simple i.e. clear sky, to very complex i.e. a tree with a lot of branches and leaves. Therefore the number of bits required for a scene varies according to the scene complexity. The best way is to save bits from simple scenes and use them for complex one and that’s how a variable bit rate decoder works. Although process for calculating bit rate is very complex.

- **Constant/Fixed Bit Rate (CBR):** For some applications, we need to transmit data on a constant bit rate. Mostly broadcast mediums like cable or satellite etc have the limitation of fixed bit rate. Compressors for constant bit rate are not as efficient as variable bit rate encoders are, but still there are very high quality encoders provided by MPEG-2 systems for CBR.

VBR is much more efficient for achieving good quality. It gives constant good quality as it can change bit rate for complex scenes. On other hand CBR gives us variable quality of a video, as it has same bit rate for each n every scene of a video.

Video Compression

Video compression is a technique to reduce the size of a digital video signal in bytes e.g. if we talk about a specific video file on a computer, with the help of video compression the size of file will be reduced. This means that we need less number of bytes to store a big video file on hard
drive and we don’t need a high performance hard drive to play the video file without delays e.g. if we want to play a video file of 30MB/s then our hard drive should have at least same speed or more to eliminate stuttering. [10]

In Video Streaming, for instance broadcast channels over the air either cable or satellite, a compressed signal will take less bandwidth as compared to uncompressed signal. It means that with the help of compression in video stream we can have more channels in less broadcast space. A big advantage of video compression for the web users is that they can view good quality of web streaming video even with low system resources (internet connection/computer resources) but of course video compression need high system resources in terms of CPU usage and memory to decode the video file.

To view a compressed video file or stream, it needs to be decompressed to its original form. When we talk specifically about compressed video file, we need a program or software to decompress the file (can be the same program which did video compression). In broadcast video stream, usually the decompression takes place in the television receiver or set up box just before displaying it on television screen. In web video streaming, some player or software (Windows Media Player/Net Meeting etc) usually contains de-compressor or sometimes as a plug-in in the web browsers. Codec is a hardware component or a software program that does compression and decompression.

Computer users are already familiar with data file compression. WinZip and WinRar are good examples of data file compression. These programs take a data file as input and give output in a compressed file (e.g. .zip and .rar) with all the important information of the original file but in reduced size. Some of the compression techniques used by these programs are same as video compression. We can also use these programs for compressing video files. These programs must use lossless compression technique because these are not specifically designed to compress video files but for all kind of computer files. Lossless compression technique is a technique in which the de-compressor must be able to recreate all the contents of original file without any wastage of data.

In most video applications, we eliminate some picture information to get more compression. This is called lossy compression technique. The output which we get from this compression will be almost same like our original picture but it may not be exact replacement. That means lossy compression technique gives us more compressed file without bad quality if we do compression with awareness.

**Video Compression Trick**

The trick of video compression is to remove some stuff from a video that no one will notice or if it is noticeable then it doesn’t make big difference. Today, a lot of video compression techniques are in use to remove
redundant video information that’s why there are different standards of compression.

**Department of Redundancy Department**

If the file contains repeated data then it is obvious that we can remove the repeated data without impacting on the rest of the file. For example, there is always a letter ‘U’ after the letter ‘Q’ in English language. So it means we can compress an English text file by eliminating all ‘U’ letters that are followed by the letter ‘Q’. While de-compressing process decompressor will put the letter ‘U’ after all the letters ‘Q’. But there is an exception if the word finishes with letter ‘Q’ e.g. “Iraq”. To grip this situation we need to improve our algorithm with some important checks. This technique is lossless [10].

**Run Length Coding**

If we have a picture of same solid color then it would be more fast way to say that the whole picture is Blue than to send the same pixel value. Normally we don’t check entire video frames that are only of one color. Normally animations and graphics do have large areas which are same. Run length coding have advantage, it sends the pixel value and the length of the run to the compressor. For instance we have value 34/56/83 for a RGB pixel and it’ll remain same for next 200 pixels then we’ll just send the value and run of the pixel. This technique is also lossless [10].

**Variable Length Coding**

Codes are the actual values of the data that makes a “pixel”. In original video signal, for one pixel we use 3 Bytes to represent Red, Green and Blue values of that specific pixel. But sometimes one color appears more than other colors. For this situation we need to invent a coding system in which one byte could be use for frequent colors and more bytes for non-frequent colors. If there are more frequent pixels with one byte code than non-frequent pixels with four or more bytes codes then the total number of bytes in the file would be less [10].

**Differential Coding**

In a common picture, the value of pixel doesn’t change much from one pixel to the next pixel. So in this technique we send the difference between the values of adjacent pixels e.g. if the value of next pixel is a little difference of Red or Green or Blue color from the previous one, then we’ll just send this difference to the compressor for that next pixel. So in this case we’ll be sending a small difference if the adjacent pixels don’t differ much. This technique can be used in lossless video compression if we use small codes for small differences and large codes for larger differences.
Vector Coding

In some cases data patterns happen more frequently than others. In this trick the algorithm gets the common pattern by making a smaller code. For example, there is a specific object like chair that’s repeating in the video frame. Then we can just send message ‘draw a chair’ rather than sending same pixel values again and again [10].

Quantization

Digitizing a video signal means to sample Red, Green and Blue signal levels at regular intervals and then convert them into digital numbers. Precision means accuracy. More precision of a pixel means more accuracy of a pixel e.g. if the green value of a pixel is 7 then its mean it is less précised. To make it more precise we need a value 7.354 but to get more precision we require more data bits.

Quantization is a reverse process of precision. To decrease the number of bits we discard the precision. For instance, in a un-compress video signal we need 8 data bits to represent one color value in RGB. So we have a series of value from 0 to 255 for each color (Red, Green and Blue). By using quantization we can use less data bits to represent one color value e.g. 4 or 5 data bits. It means we don't need 3 full bytes now to represent one pixel. Quantization is a lossy compression technique [10].

Transforms

Some compression techniques are based on transforming the pixels of a picture into a new form that is easier to compress e.g. color-space transformation (converts Red, Green and Blue pixels values into luminance and color difference that’s Y, U and V). Human eye is more aware to brightness (luminance) rather than color details that are why we can remove some detail data in color difference.

One more complex transform technique is Discrete Cosine Transform abbreviated as DCT. This mathematical algorithm changes RGB values into the matching set of RGB frequencies. We can use some of the compression techniques we explained already for RGB frequencies to get better compression.

Temporal Compression

A video contains a series of still frames e.g. NTSC video consist of 30 frames per second. This rate is enough to make fool a human’s eye. The techniques explained above are not only comparing the values the adjacent pixel but also the neighboring temporal pixels. Temporal pixel is a pixel on the same position but in the next or previous frame [10].
Scaling

Scaling is a compression technique in which we shrink the picture that also reduces the size of picture in bytes. The de-compressor takes a smaller version of picture and decompresses it in the original size of the picture. This is a lossy compression technique that’s why we lose the quality of the picture after resizing it [10].

Blocking and Reordering

Scanning of a video frame start from top to bottom and left to right means from top-left corner to right-bottom. We have explained some compression tricks in which we get advantage of consecutive pixels (Run Length Encoding and Differential Coding). Sometimes, if we change the scanning order of the pixels then we can have better results.

Some algorithms of compression divide the picture into small blocks of pixels i.e. 8 pixels broad and 8 pixels tall. In blocking compression technique the algorithm analyzes the RGB values in these small blocks because the adjacent pixels have similar values [10].

Motion compensation

This is another type of temporal compression technique. If we take an example of a moving car in front of a static background then we’ll send the same pixel value of car each time and save some bytes.

Constant/Variable Bit Rate Encoding

Bit rate means frames per unit time. Some compression techniques work really well for one type of video, which is they reduce more size in bytes but they don’t work for other kind of video. Constant Bit Rate (CBR) encoding is an encoding method that varies the level of quality to ensure a constant bit rate throughout the file. Variable bit rate encoding method make sure the high quality video throughout the encoding process. If the bit rate alters from shot to shot then it is called variable bit rate coding. It will be fine if we are compressing this file for our own computer but if this file will be shown on another computer through network then it will be a problem. For the files available for video streaming, constant bit rate is the best choice where the data is constant regardless of how the original video looks alike. The video compression quality is related to the bit rate, so variable bit rate is constant quality level and constant bit rate is variable quality level.

Video Compression Standards

The techniques used above can use themselves to compress a video file or they can work with other techniques to compress the video file to
more reduced size. Some popular compression standards are explained below.

**JPEG**

The JPEG (Joint Photographic Experts Group) standard was designed for the still pictures. It is a combination of Quantization, Coding Techniques and Discrete Cosine Transform. There is a stipulation that most people use lossless JPEG compression in one of the lossy modes. To use JPEG in video then simply take each video frame and treat it as a single picture. It is called "Motion JPEG" or "M-JPEG" when we use JPEG in this way. JPEG is variable bit rate and lossy compression technique but the amount of compression can be changed to remain the bit rate approximately constant.

**MPEG**

The MPEG (Moving Pictures Experts Group) is a set of standards for video compression and storage of moving video. Mostly videos that are available on www (World Wide Web) are with this MPEG format. It used motion compensation techniques that are used in JPEG. The three major released from MPEG are MPEG-1, MPEG-2 and MPEG-4. The MPEG-4 is more advance and complex as compared to MPEG-1. The MPEG-1 could only show the frame size of 320*240 pixels and used for Web and CD-ROM. MPEG-2 is more specific for satellite or cable transmission as it is designed for broadcasting purpose. MPEG-2 standard is also used in mastering DVD. MPEG-4 is the enhanced version of old MPEG standards that’s why it gives better compression results but the algorithm for MPEG-4 is much more complex than MPEG-1 and MPEG-2. MPEG has a variable bit rate with lossy compression technique. These standards are mostly in use for entertainment multimedia with high bit rate i.e. broadcast or movies etc.

**H.261/H.263**

These compression standards were given by the ITU (International Telecommunication Union) and used specifically for visual collaboration (videoconferencing). It means that these standards are working on comparatively at low bit rate than MPEG standards. H.261 was designed to transform data on ISDN (Integrated Services Digital Network) lines. This was the first digital video coding standard, all other standards like MPEG or H.263 were enhanced forms of this standard. H.263 compression standard is enhance form of H.261 and can be used for videoconferencing on different data rate i.e. not specific for ISDN. But as it can work also on ISDN lines too so it is a suitable replacement for previous standard (H.261) of the series.

**DV**

Sony and Panasonic gave a new compression standard specifically for consumer camcorders, called Digital Video (DV) compression standard. The main advantage of camcorders developed by this standard was that they had computer ports e.g. Fire Wire. With the help of these ports users can copy the compressed file directly from camcorder to computer. DV
and M-JPEG are same in design. DV is lossy compression with constant bit rate. But we can use DV standard only for NTSC and PAL videos.

**HDV**

HDV (High Definition Video) is using MPEG compression at constant bit rate. It is a high definition camcorder compression.

**Introduction to MPEG**

**MPEG Idea and Standard**

Moving Pictures Expert Groups (MPEG) is a standard for Digital Audio and digital Video and it is a part of International Standards Organization abbreviated as ISO. The primary task of MPEG was to make a format to playback audio and video files in real time from a compact disk (CD). When the time passed the demands increased and along with CD the Digital Versatile Disk (DVD) also needs to be supported by MPEG as transmission media like networks and satellites. These uses are covered by many collections of standards. The famous standards are MPEG-1, MPEG-2, MPEG-4 and MPEG-7. Each standard has different levels which gives support to special applications in its own way.

**MPEG-2 Digital Video Specifications**

MPEG-2 video belongs to ISO/IEC³ standard that concentrates on the syntax of an enclosed video bitstream. It includes some parameter like resolutions, picture sizes and bit rates and how they decoded to re-develop the picture. The implementation of the encoder and decoder does not defined in MPEG-2, but only they can accept the MPEG-2 bitstream. Due to this designers can develop best decoding and encoding methods while keeping in mind about compatibility. MPEG-2 has a big range of possibilities so that all applications can not use all the features. The Australian Pay TV operator, Galaxy, installed a 10-channel digital satellite DTH structure.

We are going to give some examples of the companies that are using the MPEG-2 digital video compression. In October 1995 a 20-channel digital TV was launched in Belgium, The Netherlands, Luxemburg, Scandinavia, the Middle East and Africa by a company named as Nethold’s Multichoice. Panasonic, Pace and Phillips supplied 1 million MPEG-2 set-top decoders for this system. Echostar’s DISH Television Network considered presenting 150 television channels. The Canadian DTH operator known as Express Vu intended to start on 100-channel MPEG-2 service.

**The Worldwide MPEG-2 Standard**

The bit stream is standardized that is why MPEG-2 allows different MPEG-2 compatible devices. The encoding process to generate the bitstream depends on the designer of the compressor. Therefore, all devices will not give the same video quality i.e. bit rate. So there will be different
products with different prices so user has to choose the equipment according to his/her needs.

Profiles and Levels

Description

The MPEG team did a great job for international standardization of MPEG-2. The explanation of audio and video compression, the multiplexing structure need for combing the audio, video and timing was successfully covered in ISO/IEC documents 13818 (-1 to -4).

MPEG-2 video is a group of systems, each system have a different degree of compatibility and commonality. The MPEG-2 has four levels to be coded. The range is from Limited Definition i.e. Video Cassette Recorder (VCR) to full High Definition Television abbreviated as HDTV, each have different bit rates. MPEG-2 has different ‘profiles’ according to these source formats. Each profile has a number of compression tools that combines together to make a coding system. Different profiles have different compression tools.

Profiles

MPEG-2 system has currently five profiles. Each profile is more advance and has more tools than the previous profile. This means that each new profile can do more things than the previous one, more complex to make than the previous one and of course it is expensive to make and costs more to the consumer.

Description of the five profiles

- **Simple profile** has the fewest tools. It has the basic tool kit for MPEG-2 encoding. It is predicted and intra frame decoding and encoding YUV (Y is the signal with the components luminance and U, V is color difference) 4:2:0 color sub sampling.
- The profile followed by the simple profile is known as **Main Profile**. It contains all the tools from simple profile plus one more i.e. bi-directional prediction. It gives maximum quality for the same bit rate as compared to simple profile. But it will be expensive on Integrated Circuit surface area. The decoder of main profile will decode the pictures encoded by both simple and main profile. There is an advanced form of main profile known as Main Profile Professional Level or MPEG 422. But it is not official. This Main Profile Professional Level has the ability of using line-sequential color difference signals (4:2:2), but not the higher profile scalable tools.
- The following two profiles after the Main profile are **SNR (Signal to Noise Ratio) Scalable Profile** and the **Spatially Scalable Profile**. The new tools that are added in these profiles are able to make partition of coded video data into a base layer and one or more top-up signals. Noise (SNR Scalability) or the resolution (Spatial Scalability) can be improved by the top-up signals. There may have some interesting uses
of these scaleable systems. The layer on the lowest level can be coded in a healthy way so it can give broadcast to wider area, or it can provide good service in bad receiving end conditions. It will be too expensive to use these profiles in receiver complexity. Both profiles are not supported by the Digital Video Broadcasting (DVB). YUV component video is the input for this system.

- **High Profile** is the last profile. This profile contains all the tools from previous profiles and line-simultaneous color-difference signals coding. The High Profile is a super system that works on aberrant applications that have no restriction on bit rate.

  \( I \) frames are known as intra frames.
  \( B \) frames are known as Bi-directional frames.
  \( P \) frames are known as Predicted frames.
  The detail information of these three types of frames will be explained later in the report.

Below is the table that is showing the tools profiles have.

<table>
<thead>
<tr>
<th>Tool Profile</th>
<th>I-Frames</th>
<th>P-Frames</th>
<th>B-Frames</th>
<th>4:2:2</th>
<th>SNR Scalable</th>
<th>Spatially Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main</td>
<td>x</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>422*</td>
<td>x</td>
<td>X</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNR Scalable</td>
<td>x</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Spatially Scalable</td>
<td>x</td>
<td>X</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>x</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table1</th>
<th>MPEG-2 Profiles and Coding Tool Functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Refinement of the Main Profile</td>
<td></td>
</tr>
</tbody>
</table>

**Level**

**Description of a level**

The level of the MPEG means the physical parameters like resolutions, picture sized and bit rates. MPEG-2 has four levels that are low level, Main level, High level and High 1440 level. The sampling limit of MPEG-2 video Main level and Main profile is ITU-R\textsuperscript{10} 601 parameters i.e. NTSC and PAL. The Levels limits encoding parameters such as sample rates, buffer size, frame dimensions, coded bit rates etc. and Profiles limit syntax i.e. algorithms. When Video Main Profile and Main Level combine together (MP@ML) they maintain difficulty within current technical limits. Most cable and satellites systems widely used the combination of MP@ML but different grouping is achievable to go well with other applications.
**Level according to quality**

Levels are linked with the video signal’s source format that provides a variety of impending behavior, from limited definition to high definition.

- **ITU-R Recommendation** (International Telecommunication Union - Recommendation) BT. 601 input frame is full in Main Level.
- **ITU-R Recommendation** BT.601’s definition of picture is four time than the input format of the Low Level.
- High Level contain 1920 samples/line High Definition format.
- High-1440 contain 1440 samples/line High Definition format.

<table>
<thead>
<tr>
<th>Level</th>
<th>Low</th>
<th>Main</th>
<th>High 1440</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Size (PAL/NTSC)</td>
<td>352x288</td>
<td>720x576</td>
<td>1440x1152</td>
<td>1920x1152</td>
</tr>
<tr>
<td></td>
<td>352x240</td>
<td>720x480</td>
<td>1440x1080</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Maximum Bitrate</td>
<td>4 Mb/s</td>
<td>15 Mb/s</td>
<td>60 Mb/s</td>
<td>80 Mb/s</td>
</tr>
<tr>
<td>Significance</td>
<td>CIF, Consumer tape equiv.</td>
<td>ITU-R 601, Studio TV</td>
<td>4x601 consumer HDTV</td>
<td>Prod. smpte</td>
</tr>
</tbody>
</table>

*Table2. The four Levels with maximum bit rates for each level*

Every system that covered the whole lot from computer compressed data rates of fewer than 4 MBits/Sec, through usual TV at 10 to 15 MBits/Sec and High Definition Television in use at up to 80 MBits/Sec is really a globe standard similar to MPEG-2. MPEG-2 also offers elasticity within all levels.

**Practical usage of Levels and Profiles**

*Typical Main Level bit rates for common applications*

The quality of the current video distribution formats can easily gettable if MPEG-2 video is stored on suitable medium. Perfect quality still is gettable at a low bit rate [2]. An overview of current distribution formats and bit rates are shown in the table below. MPEG-2 is coded to ML@MP with IPB frames.

<table>
<thead>
<tr>
<th>Coded rate (IPB)</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 MBit/Sec</td>
<td>Equal to VHS</td>
</tr>
<tr>
<td>4 MBit/Sec</td>
<td>PAL Broadcast quality</td>
</tr>
<tr>
<td>10 MBit/Sec</td>
<td>DVD quality</td>
</tr>
<tr>
<td>15 MBit/Sec</td>
<td>Equal to DV quality</td>
</tr>
</tbody>
</table>

*Table3. Comparison of MPEG-2 bit rates and common video distribution formats*

*Typical picture size and application*
MPEG-2 describes a number of picture sizes to go well with a number of dissimilar applications. It means MPEG-2 video is still friendly and well-matched with today’s video formats.

<table>
<thead>
<tr>
<th>NTSC</th>
<th>PAL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>352x240</td>
<td>352x288</td>
<td>SIF, Video Games and CD White Book Movies</td>
</tr>
<tr>
<td>352x480</td>
<td>352x576</td>
<td>Half Horizontal Resolution (VHS)</td>
</tr>
<tr>
<td>544x480</td>
<td>544x576</td>
<td>D2, Band Limited Broadcast and Laserdisc</td>
</tr>
<tr>
<td>640x480</td>
<td>---</td>
<td>Square Pixel NTSC</td>
</tr>
<tr>
<td>720x480</td>
<td>720x576</td>
<td>D1 and ITU-R 601</td>
</tr>
</tbody>
</table>

Table 4. MPEG-2 supports NTSC and PAL resolutions

**Video bitstream in MPEG-2**

The video bit stream is made up of dissimilar things such as blocks of pixels, macroblocks (MB), picture, Group of Pictures (GOP) and video sequences, as follows

- Pixel
- Block
- Macroblock
- Slice
- Picture
- Group of pictures
- Video Sequences

**Pixel**

Pixel is the shortcut of picture element which is a single point in an image. There is a misconception that pixel is a small dot or square but it’s really not the case see figure 1. A pixel is an abstract sample. Value of a pixel can vary according to the color system i.e. RGB color systems represents a pixel with three values of Red, Green and Blue colors.

![Pixel](image1)

**Figure 1.** A little square is not a pixel. This image shows substitute traditions of rebuilding an picture from a set of pixel values, by dots, lines, or smooth filtering. [1]
**Block**

A Block is a lowest-level entity in an encoded stream of a video. A block can be constructed with 8 x 8 pixels (8 pixels horizontally and 8 pixels vertically). The block is finished by an end of block code in the stream. Blocks can be coded with different luminance and chrominance sequence. Most common used sequence is with 4 luminance coded blocks followed by 2 chrominance coded blocks.

There are four blocks of luminance in the 4:2:0 macroblock format. There is one block for C_b and one block for C_r. Same like 4:2:0, 4:2:2 contains four blocks of luminance but two for C_b and two for C_r.

Again, four luminance blocks for 4:4:4, four for C_b and four for C_r as shown in figure 2.

![Figure 2 Macroblock Structures](image)

The figure show 8 blocks for 4:2:2 which makes 8x8x8=512 pixels.

**Macroblock**

A Macroblock structure can be defined with 16 x 16 pixels or 2 x 2 blocks. We have a macroblock header that with addition of other blocks create a macroblock. This macroblock header contains the all control information of the macroblock i.e. motion vector, spatial address and quantizer step size etc. See the structure of Macroblock in the figure 3.

![Figure 3. Structure of Macroblock](image)

The number of macroblocks in a frame depends on the picture resolution e.g. a frame with resolution 296 x 192 will have 192 macroblocks. Numbers of macroblocks in a frame should be a whole number, not a decimal value like 192.5 etc. so it shows the horizontal and vertically sizes of a frame should be some multiple of 16. If there is not a multiple of 16 in
some frame then compressor should add some padding pixels to the bottom or right of the frame. There are some different kinds of macroblocks.

- **I-macroblocks or Intra-macroblocks** are coded by themselves same as I-Pictures. I-picture contains all I-macroblocks in that. But I-macroblocks can be found in P and B frames too. Reason of the I-macroblocks presence in P and B frame can be that the motion prediction in previous or forward frame is not sufficient successful. So macroblocks can code on themselves.

- **P-macroblocks or predicted-macroblocks** use forward prediction. From name it looks like that P-macroblocks can be found in just P-pictures but its not the case. P-macroblocks also could be present in the B-pictures. Each P-macroblock carries a value of motion vector that informs about the difference between the current macroblock to be decoded and the macroblock at the similar location in the preceding frame.

- **B-macroblocks or bi-directional macroblocks** are predicted from previous and forward both directions. B-frame try to use all B-macroblocks but there are some cases when macroblock is using just forward prediction. In that case that block should be coded as P-macroblock. If the macroblock with a lot of new information to be coded then it'll be coded as I-macroblock. B and P blocks are also known as Inter-blocks.

- **Skipped macroblocks:** These macroblocks are not coded actually. We can define a skipped macroblock as if the similar positioned macroblock in the previous frame is same like the current macroblock or have a little change then the same macroblock can be used with no information need to transmit through that macroblock.

**Slice**

Slice header with some macroblocks in a sequence create a slice. Slices are very important in a video stream because they are the smallest entity in a stream from which we can attain synchronization. Every slice has a distinctive start code which can be coded in slice header and this code or number can’t be duplicated in stream anywhere. Usually a slice consist all macroblocks in a row. But it’s not necessary. There are many cases where we have many slices in a row or sometimes slice consisting just one macroblock. See the structure of slice in the figure 4.

![Slice](image)

**Figure4. Structure of a slice [4]**
**Picture**

A Picture is a set of consecutive slices preceded by picture header. In the picture case, each picture header has a unique code and control information about that picture for decoding e.g. type of picture (I, B and P). The figure 5 is showing the structure of the picture.

![Picture header](image)

**Figure 5. Structure of picture [4]**

There are three main different kinds of pictures.

- **I-pictures or Intra-pictures.** These pictures are coded by themselves. It's mean they are totally independent from other pictures. I-pictures don’t give good compression because the information they keep is important but a very big advantage of using intra pictures is that we can access video randomly at dissimilar points in a stream. The random access feature is very important when we are using some storage medium. We need that for forwarding fast or searching some specific scene in a video sequence therefore I-pictures can be viewed as start of sequence. There should be two I-pictures in one second display of a video stream according to the standard requirements.

- **P-pictures or Predicted pictures.** These pictures are forward predicted. These pictures get information from previous I or P pictures. The information current P-picture take from other pictures is used to reconstruct the picture and it's in the form of motion compensation. P-pictures give better compression than I-pictures. So we can see that I and P-pictures can be used as a reference for other pictures but they are classed with B-pictures as inter coded frames.

- **B-pictures or Bi-directional pictures.** As the name shows, these pictures are predicted from both ways mean they can take information from previous and/or forward references pictures. The main thing here to note is that B-frames are not used as prediction of any other preceding or upcoming frame. B-pictures give the maximum compression in a video sequence.

There is another type of pictures, known as D-pictures. These types of pictures are used for browsing purpose on very low bit rate. But we will not discuss them anymore as they are not in common use.

**Group of Pictures**

We have already given information about three types of frames or pictures I, B and P. These 3 frames are transmitted with a specific sequence in video in shape of a group, known as group of pictures (GOP). The figure 6 is showing the structure of GOP.
There are two different types of GOP.

- **Open GOP.** GOP always begins with an I-frame. Mostly the GOP ends with I-frame too, called open GOP. The last I-frame in the group uses the first I-frame of the next group for referencing. The cause for doing this is to get random access of video.

- **Close GOP.** This is another form of GOP which starts also with the I-frame but by definition it ends with a P-frame. As it ends with a P-frame, it doesn’t have any prediction link with the next group of picture.

Well this was introduction to start and end of GOP and types of GOP according to that. Now we discuss about the whole structure of a GOP. There are B and P frames between the starting and ending frame of a picture, there is no I-frame in-between first and last frame. Number of frames within a GOP can be different. It depends for which video you are going to design GOP or what application you use for the video.

### Length for Distribution Purpose

The length of a GOP for a movie that contains very complex scenes or fast motion scenes is mostly in the range of 8-24 frames. Most commonly used length for movies is 12 frames for 50 Hz systems and for 60 Hz system its 16 frames per group. Grouping of pictures is not essential in MPEG-2 but it’s necessary in DVD for achieving time base for Society of Motion Picture and Television Engineers (SMPTE). MPEG-2 bit stream without GOP can be accessed directly by sequence header at a specific time. This is also known as IBP GOP length as all 3 different types of frames are used in this grouping.

```
I B B P B B P B B P B B
```

*Figure 7. Structure and size of a typical GOP for distribution purpose video streams*

### Length for Editing Purpose

Group of pictures used for editing purpose are containing just I and P frames, there is no B-frame in this GOP. This is also known as IP GOP. The length of an IP GOP non-linear postproduction varies from 3 to 4 frames. This length of pictures starts same as other lengths with I-frame having 2 or 3 P-frames bounded with that. Any addition P-frame will not effect in decreasing the data rate.

```
I P P P
```

*Figure 8. Structure and size of a typical GOP for editing purpose*

23
Some smart encoders detect the change in successive frames within a video. If the change is a solid change then encoder assumes that there is a new scene now in the movie and force to create a new GOP from there. So it stops previous GOP there and a start new one with I-frame.

**Sequence**

Finally GOPs are grouped together preceded by sequence header to make a video sequence. See figure 9 for sequence.

![Figure 9. Structure of Sequence](image)

Note here that we can get a sequence just by grouping some pictures too; mean having GOP is not necessary to construct a sequence. Sequence header has the basic information for decoding video stream. Most important things that can be in a sequence header are like size of the picture and frame rate etc. we need to have sequences periodically in a video stream for better decoding because they have very important information. In most real-time applications sequences have unlimited length but its not case for every application. There can have a limited length too, mean sequence can have end.

The figure 10 is giving general view of the structure of video.

![Figure 10. Structure of Video](image)

**Frame Skipping in MPEG-2**

The main aim of frame skipping is to speed up the decoder that can result as increase the display latency. But unfortunately we don’t get totally refined results of video after frame skipping. Because when
we skip a whole frame then we also lost some amount of data that we need. So the display latency increases by a whole frame period in this case. Anyways still frame skipping is an effective method to display our video on low resources systems [7].

It’s not easy to skip frames from a given video stream. For this reason you must have information about the start and the end of the frame. And then need to throw away the details of just specific frame without losing any information from previous or forward frames. We have two different types of frame skipping [7].

- **Reactive frame skipping**: This frame skip can take place on or later than missing a deadline to re-establish the frame count stability. If the deadline is missed then we have two options. First is to abort the late frame that is almost decoded or other option is to complete this frame and skip the later frame in the stream [7].

- **Preventive frame skipping**: This type of frame skipping increases the display latency. In this type of frame skipping, decision to skip the frame takes place in the beginning of a frame. [7].

We’ll describe a little more detail of MPEG-2. When we want to skip frames from a video sequence then we need frame similarity there. For example if we have a pair of frames then we must know first that if frames are in that pair, are similar with each other or not? Mean if there is no similarity between both frames then skipping one of them can disturb the video quality a lot. To check frame similarity most known method is DCT. We’ll describe it in our paper to have its understanding.

**The encoding process**

The conformist verified that you cannot compress a video until and unless you throw something away from it. Fortunately a human visual system is not capable of gripping all of the substances existing in a moving image. Thus, by using compression techniques that selectively eliminate some information from the moving image, the human eye cannot notice and inspiring results can be obtained.

There are two techniques of encoding of video information named as Spatial and Temporal Compression. **Spatial compression** concerns with the investigation of the repeated data in the picture i.e. the data with the same frequency which is not noticeable for the human eye. **Temporal compression** deals with the encoding of the dissimilarity of the consecutive frames. For example if a car is moving in front of the static background then we do not need to encode the static background again and again. We will just encode the first whole picture until the object (car) moves. Afterward, only that part of the picture require to be encoded that is moving. The remaining part of the picture does not need to be encoded because it is the same like the first picture.
The way by which is decided that how much movement is appeared among two consecutive pictures is named as motion estimation prediction. The parts of the picture which needs to be thrown away are identified in the motion compensated prediction by giving the information attained from motion estimation prediction. It means picture cannot be deliberate in separation. The new picture builds from the forecast from the preceding picture and can be used to calculate the subsequently picture.

All the three pictures (I, P and B) offer a good prospect of redundancy. Slight compression used to encode I pictures. Motion compensation is used to eliminate temporally unneeded information from P and B pictures. B pictures provide a large amount of compression.

The bit allocations are shown in the table below.

<table>
<thead>
<tr>
<th>Picture type</th>
<th>Bit Allocation 30HZ SIF @ 1.15 MBit/sec</th>
<th>Bit Allocation 30 HZ ITU-R 601@4 MBit/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>150 Kbit</td>
<td>400 Kbit</td>
</tr>
<tr>
<td>P</td>
<td>50 Kbit</td>
<td>200 Kbit</td>
</tr>
<tr>
<td>B</td>
<td>20 Kbit</td>
<td>80 Kbit</td>
</tr>
</tbody>
</table>

Table5. Standard Test Sequence with an I-Frame distance of 15 and a P-Frame distance of 3 [2]

Details of compression

We use DCT to achieve the spatial compression. DCT converts the picture information in frequency domain for encoding purpose. Then the frequencies with ignoring amplitude are removed as they contain just some redundant information and round all other frequencies available from the picture into the standard values. Human eye can’t feel contrast difference at high frequencies, so these high frequencies are useless too and we can
remove them. We can use run length coding compression technique here to get more compression using DCT frequencies. We try to find regular pattern coming in the picture frequencies, if there are some same type of pattern, we can represent them with by some other shorter pattern. It actually works to obtain some extra compression. The one more compression technique used in MPEG-2 compressor is MCP (Motion Compensated Prediction). MCP is used to remove unnecessary temporal information from a video that is not changing from one frame to other frame, as we know the scenes or frames in a picture don’t change much in small period of time. So it means MCP encoding of a frame is dependant on other frames close to it.

**Why we need local Decoder?**

The interpreter of motion compensation requires a signal on which the prediction is based. This signal will be postponed within the predictor. The predictor should be in a loop which must be simulated in the distant decoder.

It’s a prediction on a signal which is also presented in the distant decoder that is why we need to generate a *locally decoded* signal in the encoder. This is completed in the *local decoder*, that consists of blocks to rollback the encoding phases of quantizer and DCT, chased by an adder which receives the decoded prediction error and insert it back in to a correctly postponed edition of the forecast to construct a locally decoded signal. [4]

**MPEG-2 video processing**

There are three stages of playing a MPEG video stream that are input, decoding and display. Different jobs complete these actions, which are separated by input buffer and a set of frame buffer. See figure 12. [5]

![Figure12. MPEG tasks and buffers [5]](image)

The input task takes the input of MPEG video stream. It puts the encoded stream of the video in the input buffer. The input action is very usual so it is only dogged by the permanent rate. If we talk more general then input action can be bad because of an uneven resource such as internet. [5]

The duty of the *decoding task* is to decode the date and puts it into the *frame buffer space*. If the buffer space is enough to store these decoded frames then it works more fine. [5]
The co-processor performs the display task. It depends in IO. Is also relates to the refresh rate of the screen. Once the display task started it always needs the frames to be shown on the screen.

**Similarity of Frames**

**Previous Approaches**

Different algorithms are used to check the similarity between frames e.g. DCT (Discrete Cosine Transform). Most of the algorithms are accomplished on the pixel domain. The minimal measures receive the addition of absolute pixel wise luminance dissimilarity between 2 frames of image. [6]

**DCT (Discrete Cosine Transform)**

The discrete cosine transform (DCT) is Fourier linked transform like the discrete Fourier transform but is only uses the real numbers. The DCT is used for the processing of image and signal. It is used specially in lossy data compression.

The human eyes reveal a few unique nesses that are broken by MPEG video compression. One of these is that huge substances are greatly more visible as compare to detail within them. In other words, little spatial frequency in sequence is much more visible as compare to high spatial frequency information.

MPEG video compression rejects little high spatial frequency information - the information which is less noticeable to a human eye. The primary stair in this progression is to translate a stationary image into the frequency domain. The DCT makes this transformation.

The total frame is divided into blocks of 8x8 pixels. The DCT algorithm changes the spatial information inside the block into the frequency domain. Later than the alteration, the peak left value of the block symbolizes the DC level of the block. The value right away to the right of this represents short frequency horizontal information. Value on the peak right symbolizes high frequency horizontal information. Likewise, the bottom left value symbolizes high frequency vertical information.

The subsequent figure number 14 demonstrate a 4x4 block of pixels and the resulting DCT values. Values in the DCT output matrices array is from 0 to 15.
<table>
<thead>
<tr>
<th>Pixels</th>
<th>DCT Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Block of grey pixels" /></td>
<td>8 0 0 0</td>
<td>A block of grey pixels. The DC value of the DCT output symbolizes the normal brightness. All the other values are zero.</td>
</tr>
<tr>
<td><img src="image" alt="Low frequency horizontal module" /></td>
<td>8 15 0 0</td>
<td>Low frequency horizontal module. DC value of the DCT symbolizes the normal intensity of all pixels. The ‘15’ represents the low frequency horizontal module.</td>
</tr>
<tr>
<td><img src="image" alt="A high frequency vertical component" /></td>
<td>8 0 0 0</td>
<td>A high frequency vertical component makes a high value in the bottom left corner.</td>
</tr>
<tr>
<td><img src="image" alt="Diagonal line has high frequency information" /></td>
<td>8 0 0 0</td>
<td>Diagonal line has high frequency information in horizontal and vertical orders, generating high values in the bottom right corner of the DCT production matrix.</td>
</tr>
</tbody>
</table>

**Figure 13.** 4x4 blocks of pixel and DCT values [11]
Our DCT transformed values holds a precise illustration of original macroblock. By applying an inverse DCT on the values we get back our original pixels. Our DCT amount produced at this time detained as high accuracy (e.g. floating point) values. We apply a method called quantization to decrease the accuracy of the values. Quantization only means storing the value by means of a discrete number of bits, removing the smallest amount of important information. By using this knowledge that the high spatial frequency information is less noticeable to the eye as compare to low frequency we can quantize the high frequency portions using fewer bits. It is essential that the DC component is precisely represented. [11]

In the above example 4 bits are used (values in the array 0 to 15) to symbolize the DCT matrix. By keeping in mind that human eye cannot find out high frequency information as perfectly as low frequency information, the number of bits can be changed that we quantize every entry in the matrix. The DC component must be exactly symbolized, but the number of bits can be decreased necessary for other cells. An example is shown of how many bits could be allocated for each call in the DCT matrix. Figure 14:

![DCT Matrix](image)

The original matrix had 16 calls with 4 bits per cell, giving a total of 64 bits. The quantized matrix has a total of:

\[(4 \times 1) + (3 \times 4) + (2 \times 7) + (1 \times 4) = 4 + 12 + 14 + 4 = 34 \text{ bits.}\]

A cutback is nearly 50 percent. The number of bits is verified in real MPEG decoder that DCT matrix values are coded to on every frame.

**Modified Huffman Coding**

Unchanged tables are used to achieve Huffman coding in Modified Huffman coding. This method is used for encoding of DCT output to reduce the number of bits required. The root of Huffman encoding is that encoded signs are a changeable number of bits. Regularly used symbols eat fewer bits; less frequently used symbols eat more bits. The result is a reduction in the bit requirements [11].
Frame Similarity Algorithm

The first thing that should be in knowledge is that it is not a really good idea to skip P or I frame from a video sequence because they are reference frames. Skipping P or I frame from a given bit stream can cause a noticeable disturbance in video quality. So skipping order for a video sequence should be B-frame, P-frame and then I-frame. So we are considering our all work on B-frames and if decoder will find any P or I frame in the sequence, it'll decode that frame without comparing them with other frames. Information above about DCT is good enough to understand concept of DCT. Now before representing our algorithm about which frame to skip from two similar frames, we would like to tell you one algorithm to get similar frames. There are many different algorithms to get similar frames but the one we are going to explain is following the DCT approach [11]. In this specific algorithm the frame similarity will be checked at macroblocks level. The frame will be divided in 8x8 pixel size blocks first. Then DCT will be implemented at each block to get DCT coefficient. Number of blocks in a frame depends on the resolution of frame. So when we’ll get DCT coefficients from the both frames, similarity can be checked with several methods but in this algorithm related person did it on macroblock level [9].

Here is the explanation of the given algorithm. Assume there are two frames A and B in a video sequence that we have right now in the algorithm to check similarity. The first thing will be to get the macroblocks matrices for each frame. Number of macroblocks in a frame will be M1...Mn where ‘n’ is representing the number of macroblocks in a frame.

Function Similarity(A,B)
{
    MBCount=0 ; /* Number of matching macroblocks */
    For i = 1 to N /* N is the number of Macroblocks */
    {
        if (Macrosimilar(AMi, BMi)) MBCount++;
    }
    Similarity(A,B) = MBCount / N ;
    return Similarity(A,B) ;
}

/* thresh : Threshold value for two blocks to be considered similar
   NumMatchingBlock : Number of similar blocks within a Macroblock
   Blockthresh: Threshold for two Macroblocks to be considered similar */
Function Macrosimilar(MA,MB) /*Macroblock Similarity Calculation */
{
    NumMatchingBlock=0;
    for(i=1;i<=2;i++)
    for(j=1;j<=2;j++)
        if ( abs(MA[i][j] - MB[i][j]) < thresh) NumMatchingBlock++;
    if (NumMatchingBlock > BlockThresh) return true;
    else return false;
}

31
Algorithm above [9] is to get similarity between two frames at macroblock levels. The main thing in this is threshold. Its value can be from one to four according to the writers who gave this algorithm. One (1) means the most lenient conditions for similarity while four (4) means to get similarity at very restrict level.

After getting similar frames from a sequence, we need to skip one frame from the pair of similar frames. There can be many conditions to check within the frame-internal structure before skipping the frame. But our algorithm is checking three things from them, those are most important for us. First thing is number of intra-macroblocks, second is number of skipped macroblocks and at the last position of the frame in the stream.

**Basics of our algorithm**

We combined different research works and made our own algorithm to drop a specific and appropriate frame. Dropping all frames in MPEG stream is not sufficient so that it will make the video quality very poor. There are different criteria to decide the importance of frame keeping and skipping [7]. Our algorithm takes two similar B frames and check the deep contents of the frames and makes decision which frame to skip. Previous work of checking similarity between two frames will be used in our algorithm so we will not concentrate on checking the frames similarity. The most significant things in our algorithm are explained below:

- **Frame type**: According to this criterion the most important frame in the entire GOP is I frame because all other frames depends on this frame. So there is no reason to skip this frame. If we drop this frame then the entire GOP will be disturbed so it is not possible to decode following B and P frame. B frames are the least important, then P frames and most important are I frames. If this criterion is applied only then first we will skip B frames then P and then I [7]. According to our algorithm we are not considering I and P frames to skip. In our algorithm we only consider B frames to skip. So firstly we will see the type of frame in our proposed algorithm.

- Secondly we will count the number of *intra macroblocks* in the two similar B frames. If the difference of *intra macroblocks* between two frames exceeds from a specific limit then our algorithm will drop the frame with less number of *intra macroblocks*.

- After comparing the difference of *intra macroblocks* in two frames if number of *intra macroblocks* does not exceed the specific limit our algorithm then checks the number of *skipped macroblocks* in two frames. Again if the difference of *skipped macroblocks* in frames exceeds a specific limit, the frame with more *skipped macroblocks* will be dropped.

- If the difference of *skipped macroblocks* in two frames does not exceed a specific number then algorithm will again compare the *intra macroblocks* and discard the frame with less *intra macroblocks* without checking how much difference between *intra macroblocks* of two frames have.
• At the last if there is no difference in number of intra macroblocks in both frames then according to our algorithm the later frame in the sequence will be skipped.

Here we can also check the size of two frames and keep the frame with more size but it needs more system resources so we skip this part. If we have more system resources then we can also check the size of two frames at the end and decide which frame to skip.

**Pseudo Code of proposed algorithm**

Similarity function is already implemented by many people that is why we are not considering it here. We are taking two frames "A" and "B" and assume they are already similar. Now we will go in deep details through our Algorithm to check which frame should be skipped from "A" and "B". Frame "A" and "B" both are "B-Frames" type which are known as "Bi-Directional Frames".

```
NIMBA ----> Number of Intra Macroblocks in A
NIMBB ----> Number of Intra Macroblocks in B
SMBA ----> Skipped Macroblocks in A
SMBB ----> Skipped Macroblocks in B
```

The pseudo code of proposed algorithm is described below:

```
if  ((NIMBA > NIMBB) && (NIMBA - NIMBB => 2))
    skip B
else if ((NIMBB > NIMBA) && (NIMBB - NIMBA => 2))
    skip A
else
{
    if      (( SMBA < SMBB) && (SMBB - SMBA => 10))
        skip B
    else if ((SMBA > SMBB) && (SMBA - SMBB => 10))
        skip A
    else
    {
        if (NIMBA < NIMBB)
            skip A
        else
            skip B
    }
}
```
Implementation

MPEG2event

We implemented our algorithm in Mpeg2Event. Mpeg2Event has option for embedding user code so we modified the code. It is a library of C# which made possible to make the prototypes of analysis tools of MPEG2 [8]. The parsing speed of MPEG2event is not very high as compared to other MPEG-2 decoding libraries but it has wide range of coding elements enclosed in MPEG-2 bitstream.

When we parse the video stream, the library automatically creates and prints an occurrence for each coding element bumped into. MPEG2event only work for MPEG-2 video streams [8].

Sub-Frame Analysis

We have implemented a program in C# which examines the MPEG-2 video stream on junior-frame level and gets the number of macroblocks with their sizes. We calculated the size of each frame I, B and P and also the number of intra, forward, backward, forward backward and skipped macroblocks in each frame. We also calculated the sum of all the macroblocks. The result of output file is shown in figure 15.

The details of output files are below:

- **First** column shows the type of frame.
- **Second** column shows the size of the frame.
- **Third** column shows the total number of coded macroblocks in specific frame.
• Fourth column shows the number of intra macroblocks in the related frame.
• Fifth column is showing the number of forward macroblocks.
• Sixth column has total number of backward macroblocks in that frame.
• In seventh column we showed total number of forward backward macroblocks.
• Eighth column is showing the total number of intra blocks for I frames and difference of total number of macroblocks and intra macroblocks for P and B frames.
• The total number of skipped macroblocks is shown in column number nine.
• The tenth column is giving information about which frame to skip. “1” for the frame which has to be skipped and “0” for the frame which has to be kept.

C# is used to implement the algorithm. The implemented source code takes video bitstream as input and gives an output of a text file with total number of macroblocks in I, B and P frames, their sizes, number of intra macroblocks, forward macroblocks, backward macroblocks and forward backward macroblocks. It also gives the total number of skipped macroblocks and a column which shows the value “1” to the frame which has to be skipped and “0” to the frame which has to be kept.

Now after getting the output files we used this output in other tools named as MPEG Transcoder and Peggy Tracer to create video streams with skipped frames.

MPEG transcoder

It is written in C language by Damir Isovic. It is command line based tool and our algorithm will be checked in this tool. It has the ability to take the video bitstream and amount of on hand resources as input and then gives the modified video stream. It gives the assured output depending on the available resources. The output obtains from MPEG transcoder can be played and completely well-matched with any decoders such as Windows Media Player.

MPEG transcoder also replicates the different algorithms of frame selection. It has a modular design so that frame selection algorithms can easily be added. The best feature of this tool is that it can be used both offline and online. Online means taking the input from a network and offline means taking the input of stored video stream such i.e. from hard drive.

Peggy Tracer

This application is implemented by Christian Hultman and Patrick Samuelsson. Peggy Tracer is using to analyze the MPEG video stream frame by frame. It is implemented in C language and based on libmpeg2 library.

It has a GUI (Graphical User Interface) which shows the different information of the video bit stream such as frame rate, bit rate, resolution,
compression type and also the frame related information such as number of frame in GOP. Figure16.

![Screenshot of Peggy Tracer](image)

Figure16. Screenshot of Peggy Tracer

**Experiments and Results**

We tested out algorithm on three different streams. The output of those experiments is shown in appendix. We also analyzed and considered the distance between two dropped frames. If the number of intra macroblocks and number of skipped frames are equal in both frames then we will drop the frame which comes afterwards. To increase the distance between two dropped frames makes a big difference in the video quality. It is very important to have a smooth video. We performed different experiments and the results of three experiments are shown in the form of graphs. We got great results from MPEG Transcoder which took input of a text file with skipped frames which we generated from MPEG2EVENT and generated video stream with skipped frames. There is almost no quality difference between the original and the output video file.
Graph of test 1:

The graph of first test video is shown in figure number 17.

Figure 17. Number of different macroblocks in first test video
The summary of $B$ frames is shown in figure number 18.

**Figure 18.** Summary of $B$ type frames
Graph of test2:

The graph of second test video is shown in figure 19.

Figure 19. Number of different macroblocks
The summary of B type frames is shown in figure 20.

**Figure 20.** Summary of B type frames
Graph of test3:
The graph of second test video is shown in figure 21.

Figure 21. Number of different macroblocks
The summary of B type frames is shown in figure 22.

Figure 22. Summary of B type frames
Screen Shots of input and output files

Many video files were tested in MPEG transcoder and lastly the input and output files were seen and amazingly there was nearly no difference in quality and big difference in size. The size of the video streams with skipped frames was reduced nearly 25% to 30%. Two screen shots are given below. One is from original video stream and the other one is from the video stream with frames skipped. See figures 23 and 24.

Figure 23. Screen shot of original video
Conclusion and Future Work

In this thesis we proposed an algorithm which is used to skip MPEG-2 video frames when system resources are not adequate to process all the frames. Unlike many other frame skipping approaches base proposed in the literature that are based on the structural approach to frame skipping, i.e., frame type, frame size, etc, our method compare frames on sub-frame level by examining the contents of the frames. It identifies the redundant picture elements, such as skipped macroblocks, and decides which frames are the most important for the overall video quality as perceived by the end user. Simulation results underline the effectiveness of our approach.

The work presented in this thesis can be extended in several ways. For example, other MPEG formats can be used, such as MPEG-4 that is becoming more and more de-facto standard for video streaming through different networks such as Internet or in-home entertainment networks. Furthermore, the work presented here can fairly easy be applied for online video transcoding at run-time, making it more usable for online streaming applications.
References

4. MPEG Video (Mike Knee, Snell & Wilcox)
5. Damir Isovic, Gerhard Fohler, Liesbeth Steffens, Real Time issues of MPEG-2 play out in resource constraint systems
7. Damir Isovic, Gerhard Fohler, Liesbeth Steffens, Timing Constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions
Appendix

Code in Mpeg2Event (Macroblock Mode)

using System;
using System.IO;
using MPEG2Event;
using MPEG2Event.Video;

namespace MacroblockMode
{
    public class MacroblockMode
    {
        static int total_bitsize=0, total_mb=0,ptype;
        static int mb_intra=0,mb_only_fw=0,mb_only_bw=0,mb_fw_bw=0,
        mb_sum=0;
        static int
        cumulative_B_intra=0,cumulative_B_fw=0,cumulative_B_bw=0,cumulative_B_fw_bw=0;
        static int
        cumulative_P_intra=0,cumulative_P_fw=0,cumulative_P_bw=0,cumulative_P_fw_bw=0;
        static int
        cumulative_I_intra=0,cumulative_I_fw=0,cumulative_I_bw=0,cumulative_I_fw_bw=0;
        static int cumulative_mb_I=0,cumulative_mb_P=0,cumulative_mb_B=0;

        public static void Main(String[] args)
        {
            BitStream bs;
            string stream_file_name,out_file_name="OUTPUT/";
            StreamWriter out_file;
            long skip_total_p = 0;
            long skip_total_b = 0;
            try
            {
                //stream_file_name = args[0];
                stream_file_name = "shrek.m2v";
                bs = new BitStream(File.OpenRead("c:/movie/"+stream_file_name));
            }
            catch(FileNotFoundException e)
            {
                System.Console.WriteLine("Input file not
found: " + e.FileName);
                System.Console.WriteLine("nUsage:\n");
                System.Console.WriteLine("MacroblockMode
<file_name>\n\n");
                return;
            }
            out_file_name += "output_" + stream_file_name + ".txt";
            out_file = new StreamWriter(out_file_name + ";

            VideoParser vp = new VideoParser(bs);
            Macroblock.handlers += new Macroblock.Handler(get_macroblocks);
            PictureHeader.handlers += new PictureHeader.Handler(get_picture_type);
            try
            {
                System.Console.WriteLine("Fr\tSize\t#mb\t#mb_int\t#mb_fw\t#mb_bw\t#mb_fw_bw\t(#control sum)\t#skip_mb\t#skiped");
                System.Console.WriteLine("-------------------------------------------");
                System.Console.WriteLine("-------------------------------------------");
            }
            out_file.WriteLine("Fr\tSize\t#mb\t#mb_int\t#mb_fw\t#mb_bw\t#mb_fw_bw\t(#control sum)\t#skip_mb\t#skiped");
        }
out_file.WriteLine("-------------------
-------------------------------------------------------------------------------");
int mb_sum1 = 0;
int mt = 0;
mydata data_old = null;
while (!vp.EOF)
  //for(int i=0;i<34;i++)  //ust first 3
  GoS for testing purposes
    {
      mt++;
      if(mt==20)
        {
          break;
        }
      char type = 'A';
      int skiped = 0;
      vp.parsePicture();
      if (ptype ==
PictureCodingType.I)
        { 
          System.Console.Write("I");
          type = 'I';
          cumulative_mb_I+=total_mb;
        }
      else if (ptype ==
PictureCodingType.P)
        { 
          System.Console.Write("P");
          type = 'P';
          cumulative_mb_P+=total_mb;
        }
      else if (ptype ==
PictureCodingType.B)
        { 
          System.Console.Write("B");
          type = 'B';
          cumulative_mb_B+=total_mb;
        }
      mb_sum = mb_intra +
      mb_only_fw + mb_only_bw + mb_fw_bw;
      mb_sum1 = mb_sum;
      if(ptype ==
      PictureCodingType.I)
        { 
          mb_sum = mb_intra +
          mb_sum1 = mb_sum;
        }
      else
        { 
          skiped = mb_sum1 - mb_sum;
        }
      if(ptype ==
      PictureCodingType.P)
        { 
          skip_total_p+=skiped;
        }
      else
        { 
          skip_total_b+=skiped;
        }
      System.Console.WriteLine("\t" + total_bitsize + "\t" + total_mb + "\t" +
      mb_intra + "\t" + mb_only_fw + "\t" + mb_only_bw + "\t" + mb_fw_bw + "\t\t" + mb_sum + "
      (diff: " + (total_mb-mb_sum) + ")" + "\t" + skiped);
mydata data = new
mydata();

mb_sum;

if(data_old==null)
{
    data_old = data;
}
else
{
    int diff = data_old.mb_int-data.mb_int;
    if(diff>0 & diff>=10)
    {
        skipped=1;
    }
    else
    {
        if(diff<0 & diff<-10)
        {
            skipped_old=1;
        }
        else
        {
            diff = data_old.skip_mb - data.skip_mb;
            if(diff>0 & diff>=10)
            {
                skipped_old=1;
            }
            else
            {
                if(diff<0 & diff<-10)
                {
                    skipped=1;
                }
                else
                {
                    if(data.mb>data_old.mb_int)
                    {
                    }
                }
            }
        }
    }
}
skipped=1;
}
else
if(data_old.mb>data.mb_int)
{
    skipped_old=1;
}
else
{
    skipped=1;
}
}
}
out_file.WriteLine(data_old.fr + "\t" + data_old.size + "\t" + data_old.mb + "\t" + data_old.mb_int + "\t" + data_old.mb_fw + "\t" + data_old.mb_bw + "\t" + data_old.mb_fw_bw + "\t" + data_old.mb + "\t" + data_old.mb_int + "\t" + data_old.mb_fw + "\t" + data_old.mb_bw + "\t" + data_old.mb_fw_bw + "\t" + data_old.mb + (diff:" + data_old.control + ")" + "\t" + data_old.skip_mb + "\t" + skipped_old);
out_file.WriteLine(data.fr + "\t" + data.size + "\t" + data.mb + "\t" + data.mb_int + "\t" + data.mb_fw + "\t" + data.mb_bw + "\t" + data.mb_fw_bw + "\t" + data.mb + (diff:" + data.control + ")" + "\t" + data.skip_mb);
data_old = null;
}
else
{
}
out_file.WriteLine(data.fr + "\t" + data.size + "\t" + data.mb + "\t" + data.mb_int + "\t" + data.mb_fw + "\t" + data.mb_bw + "\t" + data.mb_fw_bw + "\t" + data.mb + (diff:" + data.control + ")" + "\t" + data.skip_mb);
}
catch (OutOfBitsException e)
{
    // Ran out of bits.
    // Print warning or do whatever you want to do when you run out of bits.
}
System.Console.WriteLine("--------------------
" + cumulative_mb_I);
System.Console.WriteLine("I frames:");
System.Console.WriteLine("total_mb\t"+cumulative_mb_I);
System.Console.WriteLine("cum_int\t"+cumulative_I_intra);
System.Console.WriteLine("cum_fwd\t"+cumulative_I_fw);
System.Console.WriteLine("cum_bwd\t"+cumulative_I_bw);
System.Console.WriteLine("cum_bth\t"+cumulative_I_fw_bw);
System.Console.WriteLine("\n");
System.Console.WriteLine("P frames:");
System.Console.WriteLine("total_mb\t"+cumulative_mb_P);
System.Console.WriteLine("cum_int\t"+cumulative_P_intra);
System.Console.WriteLine("cum_fwd\t"+cumulative_P_fw);
System.Console.WriteLine("cum_bwd\t"+cumulative_P_bw);
System.Console.WriteLine("cum_bth\t"+cumulative_P_fw_bw);
System.Console.WriteLine("\n");
System.Console.WriteLine("B frames:");
System.Console.WriteLine("total_mb\t"+cumulative_mb_B);
System.Console.WriteLine("cum_int\t"+cumulative_B_intra);
System.Console.WriteLine("cum_fwd\t"+cumulative_B_fw);
System.Console.WriteLine("cum_bwd\t"+cumulative_B_bw);
System.Console.WriteLine("cum_bth\t"+cumulative_B_fw_bw);

out_file.WriteLine("---------------------------------


");
out_file.WriteLine("I frames:");
out_file.WriteLine("total_mb\t"+cumulative_mb_I);
out_file.WriteLine("cum_int\t"+cumulative_I_intra);
out_file.WriteLine("cum_fwd\t"+cumulative_I_fw);
out_file.WriteLine("cum_bwd\t"+cumulative_I_bw);
out_file.WriteLine("cum_bth\t"+cumulative_I_fw_bw);
out_file.WriteLine("cum_skp\t0");
out_file.WriteLine("\n");
out_file.WriteLine("P frames:");
out_file.WriteLine("total_mb\t"+cumulative_mb_P);
out_file.WriteLine("cum_int\t"+cumulative_P_intra);
out_file.WriteLine("cum_fwd\t"+cumulative_P_fw);
out_file.WriteLine("cum_bwd\t"+cumulative_P_bw);
out_file.WriteLine("cum_bth\t"+cumulative_P_fw_bw);
out_file.WriteLine("cum_skp\t" + skip_total_p);
out_file.WriteLine("\n");
out_file.WriteLine("B frames:");
out_file.WriteLine("total_mb\t"+cumulative_mb_B);
out_file.WriteLine("cum_int\t"+cumulative_B_intra);
out_file.WriteLine("cum_fwd\t"+cumulative_B_fw);
out_file.WriteLine("cum_bwd\t"+cumulative_B_bw);
out_file.WriteLine("cum_bth\t"+cumulative_B_fw_bw);
out_file.WriteLine("cum_skp\t" + skip_total_b);
out_file.WriteLine("totalnumber = \t"+total_num);

System.Console.WriteLine("\nDone!");
out_file.Close();
}
static void get_macroblocks(BitStream bs, Macroblock mb) {
    total_bitsize += mb.Length;
    total_mb++;
    if (mb.Mode.Intra) {
        mb_intra++;
        if (ptype == PictureCodingType.I)
{  
cumulative_I_intra ++;
}
else if (ptype == PictureCodingType.P)  
{  
cumulative_P_intra ++;
}
else if (ptype == PictureCodingType.B)  
{  
cumulative_B_intra ++;
}
}
if(mb.Mode.MotionFwdSet && !mb.Mode.MotionBwdSet)  
{  
mb_only_fw++;  
if (ptype == PictureCodingType.I)  
{  
cumulative_I_fw ++;
}
else if (ptype == PictureCodingType.P)  
{  
cumulative_P_fw ++;
}
else if (ptype == PictureCodingType.B)  
{  
cumulative_B_fw ++;
}
}
else if(mb.Mode.MotionBwdSet && !mb.Mode.MotionFwdSet)  
{  
mb_only_bw++;  
if (ptype == PictureCodingType.I)  
{  
cumulative_I_bw ++;
}
else if (ptype == PictureCodingType.P)  
{  
cumulative_P_bw ++;
}
else if (ptype == PictureCodingType.B)  
{  
cumulative_B_bw ++;
}
}
else if(mb.Mode.MotionBwdSet && mb.Mode.MotionFwdSet)  
{  
mb-fw_bw++;  
if (ptype == PictureCodingType.I)  
{  
cumulative_I-fw_bw ++;
}
else if (ptype == PictureCodingType.P)  
{  
cumulative_P-fw_bw ++;
}
else if (ptype == PictureCodingType.B)  
{  
cumulative_B-fw_bw ++;
}
}
}

static void get_picture_type(BitStream bs, PictureHeader phdr)  
{  
ptype = phdr.PictCodingType;
}
public class mydata  
{  
public char fr = 'I';  
public int size,mib,m_int,m_fwmib,m_bw,mb-fw-bw,control,skip_mib,skipped;
}  

}
Code in MPEG Transcoder of MPEG_stream_creator

/************************************************************
File:   MPEG_stream_creator.c
Description: Reads an MPEG stream and creates another
MPEG stream with some frames skipped
History:  - 2003.05.13 - file created, Damir Isovich
**************************************************************************/

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

/************************* MPEG_in properties definitions ******/
#define PICTURE_START_CODE   0x00 // 0 (dec)
#define SLICE_MIN_START      0x01 //
#define SLICE_MAX_START      0xAF // 175
#define USER_DATA_START_CODE 0xB2 // 178
#define SEQUENCE_HEADER_START_CODE 0xB3 // 179
#define SEQUENCE_ERROR_START_CODE 0xB4 // 180
#define EXTENSION_START_CODE  0xB5 // 181
#define SEQUENCE_END_CODE    0xB7 // 183
#define GOP_START_CODE   0xB8 // 184

/************************* Frame types definitions ************/
#define I_FRAME 1
#define P_FRAME 2
#define B_FRAME 3
#define ERROR -1

/************************* Misc definitios ********************/
#define TRUE 1
#define FALSE 0
#define CONTINUE 0
#define END 1

/************************* Global variables *******************/
int width, height, skip = FALSE;
long current_frame_size = 0, VBV_delay, bitrate;
double fps;
char *aspect_ratio_str;

int frame_delay = 0;

int picture_header_byte3, picture_header_byte4;
int picture_header_byte5, picture_header_byte6;
int picture_header_byte7;

FILE *out;     //text file
FILE *MPEG_in, *MPEG_out; //MPEG-2 files

/**************************** Get stream info from sequence header (SH)****************************
width = entire byte4 and the first half of byte5
height = the second half of byte5 and the entire 6
aspect = the first half of byte7
fps = the second half of byte7
bitrate= byte8, byte9 and two first bits of byte10

(see MPEG 2 ISO/IEC 12818 for more info on headers)
void getStreamInfo() {

    long SH_byte4, SH_byte5, SH_byte6, SH_byte7, SH_byte8, SH_byte9, SH_byte10;
    int aspect_ratio_code, frame_rate_code;
    int a, b, c; // temp variables

    // get relevant bytes from the sequence header
    SH_byte4 = getc(MPEG_in);
    SH_byte5 = getc(MPEG_in);
    SH_byte6 = getc(MPEG_in);
    SH_byte7 = getc(MPEG_in);
    SH_byte8 = getc(MPEG_in);
    SH_byte9 = getc(MPEG_in);
    SH_byte10 = getc(MPEG_in);

    // extract width and height from the first three bytes
    width = (SH_byte4 << 4) | (SH_byte5 >> 4);
    height = ((SH_byte5 & 0x0f) << 8) | SH_byte6;

    // extract aspect ratio and fps codes from byte 7
    aspect_ratio_code = (SH_byte7 >> 4) & 0x0f;
    frame_rate_code = SH_byte7 & 0x0f;

    // translate the codes
    switch(aspect_ratio_code) {
        case 0x01: aspect_ratio_str = "1:1"; break;
        case 0x02: aspect_ratio_str = "3:4"; break;
        case 0x03: aspect_ratio_str = "9:16"; break;
        case 0x04: aspect_ratio_str = "1:2.21"; break;
        default: aspect_ratio_str = "unknown"; break;
    }
    switch(frame_rate_code) {
        case 0x01: fps = 23.976; break;
        case 0x02: fps = 24; break;
        case 0x03: fps = 25; break;
        case 0x04: fps = 29.970; break;
        case 0x05: fps = 30; break;
        case 0x06: fps = 50; break;
        case 0x07: fps = 59.940; break;
        case 0x08: fps = 60; break;
        default: fps = 0; break;
    }

    // extract bitrate
    // (need some help variables for better readability)
    a = SH_byte8 << 8;
    b = SH_byte9;
    c = SH_byte10 >> 6;

    bitrate = ((a | b) << 2) | c;
    bitrate *= 400; // bitrate is measured in units of 400 bits/sec

    // copy all bytes to the output MPEG stream
    putc(SH_byte4, MPEG_out);
    putc(SH_byte5, MPEG_out);
    putc(SH_byte6, MPEG_out);
    putc(SH_byte7, MPEG_out);
    putc(SH_byte8, MPEG_out);
    putc(SH_byte9, MPEG_out);
    putc(SH_byte10, MPEG_out);
}

DESC: Get frame type from the picture header (PH)
frame type = three bits in byte 5
******************************************************************************************/
int PH_byte4, PH_byte5, PH_byte6, PH_byte7, frame_type;
int a,b,c;  // temp variables

// get relevant bytes from the picture header
// needed to determine frame type
PH_byte4 = getc(MPEG_in);
PH_byte5 = getc(MPEG_in);

// extract frame type from byte5
frame_type = (PH_byte5 >> 3) & 7;

// get relevant bytes from the picture header
// needed to determine VBV delay of the frame
PH_byte6 = getc(MPEG_in);
PH_byte7 = getc(MPEG_in);

// extract VBV delay
// (need some help variables for better readability)
a = (PH_byte5 & 0x07) << 8;
b = PH_byte6;
c = PH_byte7 >> 3;
VBV_delay =  ((a | b)<<5) | c;

// Do not write bytes directly to the output
// MPEG stream since we don’t know yet whether the frame
// will be skipped or not. Save those bytes and write them
// later to the output stream if the frame is not skipped
picture_header_byte4 = PH_byte4;
picture_header_byte5 = PH_byte5;
picture_header_byte6 = PH_byte6;
picture_header_byte7 = PH_byte7;

switch (frame_type) {
    case I_FRAME: return I_FRAME;
    case P_FRAME: return P_FRAME;
    case B_FRAME: return B_FRAME;
    default: return ERROR;
}

/**************************************************************
DESC: Parses the MPEG_in sequentially until a start code is
found (a start code begins with 0x000001).
**************************************************************
void nextStartCode () {
    long counter=0;
    long byte;
    while(1) {
        byte = getc(MPEG_in);
       putc(byte,MPEG_out);
        switch(byte) {
            case EOF:  return;
            case 0x00: counter++; break;
            case 0x01: if (counter >= 2) return;
            default:   counter=0;
        }
    }
}

/**************************************************************
DESC: Parse MPEG input stream until next picture start code.
Copy each byte to the output MPEG stream.
**************************************************************
int nextPictureHeader() {
    long counter=0;
    long byte;
while(1) {
    byte = getc(MPEG_in);
    putc(byte, MPEG_out);

    switch(byte) {
    case EOF:  return END;
    case 0x00: counter++; break;
    case 0x01:
        if (counter >= 2) {
            byte = getc(MPEG_in);
            if (byte == PICTURE_START_CODE) {
                picture_header_byte3 = byte;
                picture start code,
                to write it to the output MPEG stream
                is not skipped
                write to MPEG_out directly since it may
                that we want to skip the frame that starts here
                we don't want to write its header)
                picture_header_byte3 = byte;
                // Save
                // we need
                // if frame
                // (do not
                // happen
                // and hence
                picture_header_byte3 = byte;
                // the
                // initiate
                return byte;
                }
                putc(byte, MPEG_out);
            break;
            default:
                counter=0;
                current_frame_size += 8; /* add 8 bits
*
            }
        }
    } /*
    }
}

/***************************************************************************/
DESC: Parse MPEG_in until next picture start code WITHOUT
      copying bytes to the output MPEG stream. The effect is
      the current frame is not written to the output MPEG
      stream, i.e., it is skipped.
***************************************************************************/
int skipFrame () {
    long counter=0;
    long byte;
    while(1) {
        byte = getc(MPEG_in);
        switch(byte) {
        case EOF: return END;
        case 0x00: counter++; break;
        case 0x01:
            if (counter >= 2) {
                byte = getc(MPEG_in);
                if (byte == PICTURE_START_CODE) {
                    picture_header_byte3 = byte;
                    picture start code,
                    to write it to the output MPEG stream
                    is not skipped
                    write to MPEG_out directly since it may
                    that we want to skip the frame that starts here
                    we don't want to write its header)
                    picture_header_byte3 = byte;
                    // Save
                    // we need
                    // if frame
                    // (do not
                    // happen
                    // and hence
                    picture_header_byte3 = byte;
                    // the
                    // initiate
                    return byte;
                    }
                    putc(byte, MPEG_out);
                break;
                default:
                    counter=0;
                    current_frame_size += 8; /* add 8 bits
                */
            }
        }
    }
}
if (byte == PICTURE_START_CODE) {
    // the current frame is skipped ==> 
    // increase the display delay of the next frame
    frame_delay++;
    return byte;
}

default:
    counter = 0;
    current_frame_size += 8; /* add 8 bits */
*/
}

/****************************************************************************
DESC: Here it starts....
IN: MPEG file name
****************************************************************************/
main(short argc, char *argv[]) {

    char in_MPEG_filename[100], out_TXT_filename[100] = "c:\output\output.txt",
    out_MPEG_filename[100] = "c:\output\output.m2v",
    *frame_type_str;
    int finish = FALSE;
    int frameType;
    int byte;
    int num = 0;
    // check the input parameters
    /*if (argc < 2)
       printf("Wrong numbers of arguments. \n Usage: MPEG_stream_creator <file name>\n\n" );
       return 0; */
    strcpy(in_MPEG_filename, "c:\test5.m2v");
    //in_MPEG_filename = "";
    if ((MPEG_in = fopen(in_MPEG_filename, "rb")) == NULL) {
        printf("Error: Cannot open input MPEG file \\
%s.\n", in_MPEG_filename);
        return 0;
    }

    // get output file names (from the input file name)
    /*strcat(out_MPEG_filename, argv[1]);
    strcat(out_TXT_filename, argv[1]);
    strcat(out_TXT_filename, ".txt");*/

    // try to create output files
    if ((MPEG_out = fopen(out_MPEG_filename, "wb")) == NULL) {
        printf("Error: Cannot create output MPEG file \\
%s.\n", out_MPEG_filename);
        return 0;
    }
    if ((out = fopen(out_TXT_filename, "wt")) == NULL) {
        printf("Error: Cannot create output text file \\
%s.\n", out_TXT_filename);
        return 0;
    }

    // Find first sequence header in the input stream
    nextStartCode();

    byte = getc(MPEG_in);
   putc(byte, MPEG_out);
if (byte != SEQUENCE_HEADER_START_CODE) {
    printf("Error: %s is not a valid MPEG video MPEG_in. SEQUENCE_HEADER_START_CODE not found!\n"),
} else{
    // get stream resolution, aspect ratio and fps-rate from seq. header
    getStreamInfo();
    printf("aspect ratio: \t %s\n",aspect_ratio_str);
    printf("fps: \t\t %f\n",fps);
    printf("bitrate: \t %d bits/sec\n",bitrate);
}

// parse the input stream until the first picture is found
while(!finish)
{

    byte = nextPictureHeader();
    // parse the rest
    while(!finish)
    {

        if(byte == PICTURE_START_CODE){
            frameType = getFrameType();
            if(frameType == I_FRAME){
                frame_type_str = "\nI ";
                skip = FALSE;
            } else if(frameType == P_FRAME){
                frame_type_str = "P ";
                skip = FALSE;
            } else if(frameType == B_FRAME){
                frame_type_str = "B";
                if(skipthis(num)==0)
                    skip = FALSE;
                else
                    skip = TRUE;
            }
            if(skip == TRUE){
                printf("-");
                byte = skipFrame();
            } else{
                printf("%s",frame_type_str);
                // delay the frame depending on how many frames are skipped before the last displayed
                // frame and this one
                //if(frame_delay != 0){
                    picture_header_byte5 = 0xf8;
                    // picture_header_byte6 = 0x00;
                    // picture_header_byte7 = 0x00;
                //}
                write the 'getFrameType'
                // to the output MPEG stream
                putc(picture_header_byte3,MPEG_out);
                putc(picture_header_byte4,MPEG_out);
                putc(picture_header_byte5,MPEG_out);
                putc(picture_header_byte6,MPEG_out);
                putc(picture_header_byte7,MPEG_out);
                byte = nextPictureHeader();
            }
        }
    }
}
if(byte == END || byte == ERROR){
    printf("\n\nFINISH\n");
    finish = TRUE;
}
num++;

fclose(out);
fclose(MPEG_out);
return 0;
}

Code in MPEG Transcoder of file.c

/***************************************************************************/
FILE:   MPEG_stream_creator.c
Description: Reads an MPEG stream and creates another MPEG stream with some frames skipped
History:  - 2003.05.13 - file created, Damir Isovic
/***************************************************************************/

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

int skipthis(int rnum)
{
    FILE *fr;
    char line[300];
    int mynum = 0;
    int res = 0;
    int len =0;
    if((fr = fopen("c:\test5.txt", "r")) != NULL)
    {
        while(fgets(line, 300, fr) != NULL)
        {
            len = strlen(line);
            if(mynum==rnum+2)
            {
                if(line[len-2]=='1')
                {
                    res = 1;
                }
                break;
            }
            mynum++;
        }
    fclose(fr);
    }
    else
    {
        printf("Error Opening File.\n");
    }
    return res;
}

int main()
{
    FILE *fr;
    char line[300];
    int mynum = 0;
    int res = 0;
    int len =0;
    if((fr = fopen("c:\test4.txt", "r")) != NULL)
    {
        while(fgets(line, 300, fr) != NULL)
        {
            len = strlen(line);
            if(mynum==rnum+2)
            {
                if(line[len-2]=='1')
                {
                    res = 1;
                }
                break;
            }
            mynum++;
        }
    fclose(fr);
    }
    else
    {
        printf("Error Opening File.\n");
    }
    return res;
}
while(fgets(line, 300, fr) != NULL) {
    printf(line);
    printf("========\n");
}
else {
    printf("Error Opening File.\n");
}
Output of test1 from MPEG2EVENT

<table>
<thead>
<tr>
<th>FF</th>
<th>Size</th>
<th>#DB</th>
<th>#DB_Tot</th>
<th>#DB_Fe</th>
<th>#DB_Len</th>
<th>#DB_Fe_Len</th>
<th>(#Control_sum)</th>
<th>#Skip_DB</th>
<th>#Skiped</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>38841</td>
<td>1340</td>
<td>1350</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1350 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>36415</td>
<td>1183</td>
<td>46</td>
<td>0</td>
<td>1123</td>
<td>0</td>
<td>1123 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>31578</td>
<td>1094</td>
<td>88</td>
<td>0</td>
<td>88</td>
<td>0</td>
<td>88 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>35618</td>
<td>1147</td>
<td>17</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>17 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>49137</td>
<td>1217</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>37936</td>
<td>1122</td>
<td>111</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>111 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>89545</td>
<td>1179</td>
<td>77</td>
<td>0</td>
<td>77</td>
<td>0</td>
<td>77 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>12107</td>
<td>1140</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>9 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>71500</td>
<td>1092</td>
<td>73</td>
<td>0</td>
<td>73</td>
<td>0</td>
<td>73 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>12413</td>
<td>1127</td>
<td>17</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>17 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>41883</td>
<td>1127</td>
<td>17</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>17 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>28562</td>
<td>1150</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>15 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>248055</td>
<td>1310</td>
<td>1150</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1150 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>12576</td>
<td>1240</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>15 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>484960</td>
<td>1147</td>
<td>152</td>
<td>244</td>
<td>234</td>
<td>537</td>
<td>357 (diff 0)</td>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>473793</td>
<td>1196</td>
<td>374</td>
<td>408</td>
<td>0</td>
<td>0</td>
<td>408 (diff 0)</td>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>132190</td>
<td>1140</td>
<td>142</td>
<td>88</td>
<td>13</td>
<td>79</td>
<td>129 (diff 0)</td>
<td>219</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>152474</td>
<td>1155</td>
<td>2</td>
<td>88</td>
<td>1023</td>
<td>49</td>
<td>1035 (diff 0)</td>
<td>181</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>113015</td>
<td>1136</td>
<td>2</td>
<td>125</td>
<td>125</td>
<td>590</td>
<td>590 (diff 0)</td>
<td>966</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>130386</td>
<td>1127</td>
<td>2</td>
<td>126</td>
<td>123</td>
<td>777</td>
<td>777 (diff 0)</td>
<td>221</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>131183</td>
<td>1146</td>
<td>0</td>
<td>519</td>
<td>136</td>
<td>721</td>
<td>721 (diff 0)</td>
<td>534</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>35015</td>
<td>1091</td>
<td>5</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89 (diff 0)</td>
<td>519</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>134901</td>
<td>1116</td>
<td>2</td>
<td>189</td>
<td>120</td>
<td>309</td>
<td>309 (diff 0)</td>
<td>309</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>134846</td>
<td>1137</td>
<td>2</td>
<td>101</td>
<td>148</td>
<td>806</td>
<td>806 (diff 0)</td>
<td>213</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>470965</td>
<td>1310</td>
<td>155</td>
<td>252</td>
<td>258</td>
<td>853</td>
<td>853 (diff 0)</td>
<td>239</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>120729</td>
<td>1134</td>
<td>0</td>
<td>169</td>
<td>111</td>
<td>854</td>
<td>854 (diff 0)</td>
<td>218</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>249393</td>
<td>1100</td>
<td>9</td>
<td>911</td>
<td>0</td>
<td>0</td>
<td>911 (diff 0)</td>
<td>420</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>135988</td>
<td>1111</td>
<td>1</td>
<td>157</td>
<td>122</td>
<td>839</td>
<td>839 (diff 0)</td>
<td>239</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>138998</td>
<td>1116</td>
<td>1</td>
<td>116</td>
<td>173</td>
<td>844</td>
<td>844 (diff 0)</td>
<td>764</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>266865</td>
<td>1092</td>
<td>115</td>
<td>105</td>
<td>0</td>
<td>0</td>
<td>105 (diff 0)</td>
<td>322</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>115210</td>
<td>1178</td>
<td>6</td>
<td>155</td>
<td>81</td>
<td>876</td>
<td>876 (diff 0)</td>
<td>222</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>436390</td>
<td>1180</td>
<td>836</td>
<td>481</td>
<td>0</td>
<td>0</td>
<td>836 (diff 0)</td>
<td>259</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>187032</td>
<td>1146</td>
<td>9</td>
<td>68</td>
<td>1090</td>
<td>19</td>
<td>1099 (diff 0)</td>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>172892</td>
<td>1143</td>
<td>0</td>
<td>69</td>
<td>1935</td>
<td>21</td>
<td>1937 (diff 0)</td>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>441728</td>
<td>1330</td>
<td>1550</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1550 (diff 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>147687</td>
<td>1141</td>
<td>0</td>
<td>182</td>
<td>210</td>
<td>709</td>
<td>709 (diff 0)</td>
<td>209</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>149875</td>
<td>1147</td>
<td>0</td>
<td>181</td>
<td>217</td>
<td>785</td>
<td>785 (diff 0)</td>
<td>202</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>144970</td>
<td>1144</td>
<td>0</td>
<td>186</td>
<td>188</td>
<td>755</td>
<td>755 (diff 0)</td>
<td>219</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>149892</td>
<td>1140</td>
<td>1</td>
<td>143</td>
<td>130</td>
<td>860</td>
<td>860 (diff 0)</td>
<td>318</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>315472</td>
<td>1091</td>
<td>5</td>
<td>514</td>
<td>0</td>
<td>0</td>
<td>514 (diff 0)</td>
<td>431</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>149878</td>
<td>1143</td>
<td>0</td>
<td>101</td>
<td>198</td>
<td>621</td>
<td>621 (diff 0)</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>49954</td>
<td>1145</td>
<td>0</td>
<td>185</td>
<td>323</td>
<td>637</td>
<td>637 (diff 0)</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>289960</td>
<td>1192</td>
<td>11</td>
<td>996</td>
<td>0</td>
<td>0</td>
<td>996 (diff 0)</td>
<td>348</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>140892</td>
<td>1148</td>
<td>1</td>
<td>328</td>
<td>0</td>
<td>0</td>
<td>328 (diff 0)</td>
<td>202</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>141274</td>
<td>1150</td>
<td>15</td>
<td>159</td>
<td>147</td>
<td>793</td>
<td>793 (diff 0)</td>
<td>197</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>156754</td>
<td>1137</td>
<td>0</td>
<td>128</td>
<td>81</td>
<td>228</td>
<td>228 (diff 0)</td>
<td>213</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>297008</td>
<td>1096</td>
<td>774</td>
<td>315</td>
<td>0</td>
<td>0</td>
<td>1096 (diff 0)</td>
<td>281</td>
<td>1</td>
</tr>
</tbody>
</table>
Output of test2 from MPEG2EVENT

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>7151</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>7150</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>4973</td>
<td>83</td>
<td>0</td>
<td>0</td>
<td>83</td>
<td>83</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>14690</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>3940</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>O</td>
<td>12433</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>121</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>3281</td>
<td>87</td>
<td>0</td>
<td>0</td>
<td>87</td>
<td>87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>6162</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>3940</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>12963</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>34696</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>10020</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>6707</td>
<td>99</td>
<td>0</td>
<td>0</td>
<td>99</td>
<td>99</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>10020</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>14035</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>81</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>13899</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>86</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>17658</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>13351</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>81</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>2407</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>17758</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>6358</td>
<td>83</td>
<td>0</td>
<td>0</td>
<td>83</td>
<td>83</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>14705</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>92</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>5482</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>11885</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>4669</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>86</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>6031</td>
<td>78</td>
<td>0</td>
<td>0</td>
<td>78</td>
<td>78</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>20512</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>l</td>
<td>7151</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>6797</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>n</td>
<td>10774</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>108</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>o</td>
<td>12184</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>24870</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q</td>
<td>12894</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r</td>
<td>12895</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>12896</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>7449</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>u</td>
<td>11856</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>v</td>
<td>4698</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w</td>
<td>8162</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>21884</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>21885</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>z</td>
<td>21886</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Output of test3 from MPEG2EVENT

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Disk</th>
<th>#MB_Int</th>
<th>#MB_Fw</th>
<th>#MB_Bm</th>
<th>#MB_Fw_Bm</th>
<th>#MB_Ctrl</th>
<th>#MB_Diff</th>
<th>#MB_SKIP</th>
<th>#MB_SKIPed</th>
</tr>
</thead>
<tbody>
<tr>
<td>81877</td>
<td>1620</td>
<td>1620</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1620</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>39813</td>
<td>1192</td>
<td>1192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1192</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>11247</td>
<td>1218</td>
<td>1218</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1218</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>61562</td>
<td>1380</td>
<td>1380</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1380</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>14954</td>
<td>1141</td>
<td>1141</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1141</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>49819</td>
<td>1190</td>
<td>1190</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1190</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>31182</td>
<td>1349</td>
<td>1349</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1349</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>38999</td>
<td>1249</td>
<td>1249</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1249</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>102174</td>
<td>1120</td>
<td>1120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1120</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
<tr>
<td>47224</td>
<td>1220</td>
<td>1220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1220</td>
<td>DIFF(0)</td>
<td>0</td>
<td>1590</td>
<td>0</td>
</tr>
</tbody>
</table>

62