Technical University of Braunschweig

Institute of Operation Systems and Computer Networks

Project on

Extraction of Annotated Presentations

Di, Pengfei

Supervised by

Zefir Kurtisi

August 2004, Braunschweig

Statement

I assure, that the following work is done by myself and with the help of listed literature.

August, 2004 Braunschweig Di, Pengfei

Abstract

In some E-learning projects, the lectures are recoded into videos and published online for the students to download. In IBR (Institut für Betriebssysteme und Rechnerverbund) of TU-Braunschweig, the animations of the slides used in lectures are recorded by "Camtasia Studio" [1] into videos for downloads with TSCC [2] codec (TechSmith Screen Capture Codec). Such codec can result in very compressed lossless videos, but since TSCC is a commercial codec and not available under Unix/Linux system, these TSCC coded videos are further encoded into MPEG4 [3] files with XviD [4] codec by reason of system-independency.

Since the real-time records of the slide include much duplicated information, for example, most slides are keeping idle for long time, the XviD codec is not the optimal way for slide video in respect of compression rate and encoding time. A new coding method is required. The goal of this student project is to develop a lossless visual codec as a potential alternative of MPEG4 codec for the slide-video with ensured quality and more compact file-size.

The project includes two parts: an Easy Extractor, which separates a slide-video into predefined frames, and an Easy Player, which reconstructs the source video. Comparing to XviD, Easy Extractor has have the advantages in both compression rate and encoding time. While comparing to TSCC, Easy Extractor has the advantage only in encoding time, not in compression rate. Nevertheless this project provides one method instead of MPEG4 codec to encode slide-video.

(1) "Camtasia Studio" is a commercial screen capture tool produced by TechSmith Corporation. http://www.techsmith.com

(2) "TSCC" is the lossless codec used in "Camtasia Studio". http://www.techsmith.com

(3) "MPEG4" is an ISO/IEC graphics and video lossy compression algorithm standard that is based on MPEG-1 and MPEG-2 and Apple QuickTime technology and developed by MPEG (Moving Picture Experts Group). http://www.mpeg.org/

(4) "XviD" is an ISO MPEG-4 compliant video codec. It's not a product, but an open source project, which is developed and maintained by lots of people from all over the world. It is available in both Windows system and Unix/Linux system. http://www.xvid.org

1 Introduction

1.1 Motivation

Along with the quick development of Internet, Remote Education and E-learning become possible and favorable. In many cases, the lectures are recorded into videos for downloading, so the students can take the lectures at home.

In IBR (Institut für Betriebssysteme und Rechnerverbund, TU-Braunschweig), as one part of the E-learning project, the lectures are also recorded into videos with a commercial screen capture tool “Camtasia Studio”. Since the visual codec TSCC used in “Camtasia Studio” is not an open source and not available under Unix/Linux system, the visual part of the video is later encoded to MPEG4 files with XviD codec for downloading. MEPG4 technology has significant advantage in the use of recording moving motions, and it is widely used in the encoding of movies, etc. But as to the current problem of ours, since our objects are static slides and rarely occurred animations and annotations made by professor during the lecture, the use of lossy MPEG4 codec designed for the active motions is not satisfactory in respect of compression rate and coding time. Therefore we intend to develop an alterative of MPEG4 codec that is best suitable for the coding of slide-videos.

This project would provide a simplex lossless codec to record and replay the slides as well as the animations and annotations on them. Some tests will also be included in this project to evaluate the codec’s performance.

1.2 MPEG Outline

This project uses some terms from MPEG technology. So it is necessary to introduce some basic knowledge of MPEG here.

MPEG is short for Moving Picture Experts Group, and pronounced m-peg, a working group of ISO. The term also refers to the family of digital video compression standards and file formats developed by the group.

MPEG achieves high compression rate by storing only the changes from one frame to another, instead of each entire frame. There are 3 main types of frames in MPEG: *I-frame* (intra frame or key frame), *P-frame* (predictive frame) and *B-frame* (bi-directional frame). *I-frame* is an independent frame of digital content that stores all of the data needed to display original scene, and it is also used for synchronization. *P-frames* come after I-frames (if there is no B-frames) and contain only the data that have changed from the preceding I-frame (such as color or content changes). As the name suggests, *B-frames* rely on the frames preceding and following them. B-frames stand between I-frames and P-frames, containing only the data that have changed from the preceding frame or are different from the data in the very next frame. The frames sequence is some like: IBBPBBPIBBPBBP…

Normally the I-frame is encoded using a technique called DCT (Discrete Cosine Transform), a common data compression method, as used in JPEG. Thus MPEG uses a type of lossy compression, since some data is removed. But the diminishment of data is generally imperceptible to the human eye.

There are three major MPEG standards: MPEG-1, MPEG-2 and MPEG-4.

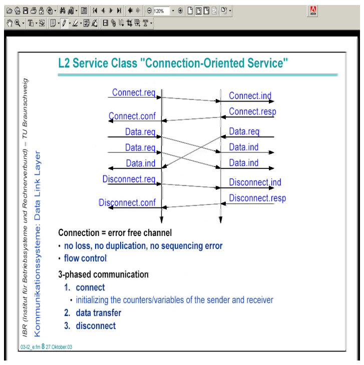
- The most common implementations of the MPEG-1 standard provide a video resolution of 352-by-240 at 30 frames per second (fps). MPEG-1 is used by VCD-ROMs. This produces video quality slightly below the quality of conventional VCR videos.

- MPEG-2 offers resolutions of 720x480 and 1280x720 at 60 fps, with full CD-quality audio. MPEG-2 is used by DVD-ROMs and HDTV (High Definition Television). MPEG-2 can compress a 2-hour video into a few gigabytes. While decompressing an MPEG-2 data stream requires only modest computing power, encoding video in MPEG-2 format requires significantly more processing power.
- MPEG-4 is a graphic and video compression algorithm standard based on MPEG-1 and MPEG-2 and Apple QuickTime technology. MPEG-4 audiovisual scenes are composed of hierarchical media objects, which consist of video, audio, text, graphic and 2-D and 3-D animation, etc. MPEG-4 was designed to transmit videos and images over a narrower bandwidth and can fit an entire DVD quality movie on a CD-R. MPEG-4 was standardized in October 1998 in the ISO/IEC document 14496.

1.3 Problem Description

In most cases, the slide-video has no complex motions, such as rotation, distortion or extension. Instead there are only 3 simple dynamic types: quiescence, animation/annotation, and new slide. Furthermore, the quiescence occupies the most time of the video, which consequently leads to vast duplicate frames in the video stream, while animations/annotations are usually very small.

Let's have a look at an example. One series of the slides of lecture is:

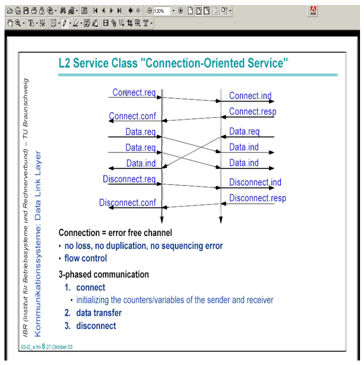


L2 Service Class "Connection-Oriented Service"

Connection = error free channel
- no loss, no duplication, no sequencing error
- flow control

3-phased communication
1. connect
   - initializing the counters/variables of the sender and receiver
2. data transfer
3. disconnect

10.png

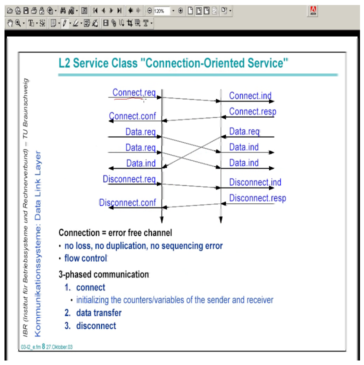


L2 Service Class "Connection-Oriented Service"

Connection = error free channel
- no loss, no duplication, no sequencing error
- flow control

3-phased communication
1. connect
   - initializing the counters/variables of the sender and receiver
2. data transfer
3. disconnect

11.png

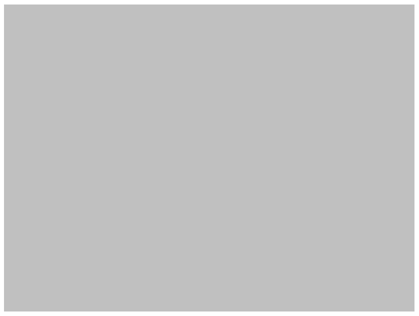


L2 Service Class "Connection-Oriented Service"

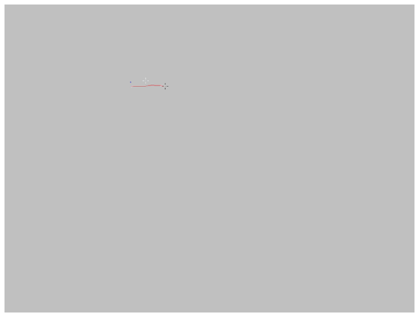
Connection = error free channel
- no loss, no duplication, no sequencing error
- flow control

3-phased communication
1. connect
   - initializing the counters/variables of the sender and receiver
2. data transfer
3. disconnect

12.png



Difference between 10 & 11



Difference between 11 & 12

Figure 1: example frames of slide video.

In Figure 1 we see that there is almost no difference between these three successive pictures, only little change occurs in the 12th picture. So our focus is to record the little change in that frame as well as the quiescence of the other two frames.

Although MPEG4 codec can compresses efficiently high-quality movies, considering the following reasons, the MPEG4 encoding method is not the optimal choice for reproducing the slide-video:

◆ The codec of MPEG is lossy.

A lossless codec is sometimes preferred; because it can 100% reproduce the real scene of the original movie. To get high-quality picture, the file-size of DCT based pictures will become very large. Therefore using lossless formats like PNG-file (Portable Network Graphics) is a better choice.

◆ The encoding and decoding of MPEG4 are too complex.

Due to the simpleness of the slide video, an easily understandable encode & decode method is more preferred of the developers to find the optimal solution of slide-videos.

◆ The encoding time of high-fps video is not satisfactory.

The encoding time of a 90-minutes slide-video with 15 fps takes more than an hour, which is somehow unendurable.

◆ The compression rate of MPEG4 codec is not as good as we had thought.

For example, a 2MB TSCC encoded video may become 20MB after XviD codec, which is 10 times larger than the original.

Due to the reasons mentioned above, we intend to develop a new program to deal with slide videos, which will overcome these drawbacks and ensure better performance than MPEG4 video codec.

1.4 Basic Idea

As mentioned in last paragraph, we consider that the slide videos have only 3 types of motions: quiescent, some added annotation, a new slide, named as *0-frame, P-frame,* and *I-frame* respectively. The basic idea of this project is to separate the video streams into these 3 types of frames. The process includes: separating the video stream into images in memory, comparing every 2 adjacent images pixel-by-pixel, determining the frame type of 2nd frame, recording the different pixels and writing them as separated files on hard disk.

The frame types defined here are similar with those used in MPEG technology. We say that if there are no or just very few different pixels (maybe just some noise) between 2 images, or the amount of different pixels is less than the threshold of new P-frame, we consider the 2 images as same one, i.e. there is no annotation or animation occurred. We call the 2nd images as *0-frame*, e.g. the “11.png” in Figure 1.

If there do exist some different pixels, whose amount is larger than the threshold of new P-frame but less than the threshold of new I-frame, we consider the 2nd image is based on the first one. What should be done is to record the differences from the 2nd image. And we call these differences in 2nd images as *P-frame*. For example, in Figure 1, we can consider 12th frame as P-frame:

12.png = 11.png + (some red points)

If there are too many different pixels, whose amount exceeds the threshold of new I-frame, we should consider them as 2 independent images, and save the 2nd image as *I-frame.* The types of each frame are recorded into a separate file, named as “INF” file.

A program, named as *Easy Extractor*, will perform the preceding works. On the contrary, to replay these separated I-frame (slide) and P-frame (annotations and animations), an *Easy Player* will also be developed. From the "INF" file the player gets the information to read and show I-frames, then reads and draws the P-frames following the previous I-frame, until next I-frame.

1.5 Related Works

In this project, the first task is to separate a AVI video into images. There are quite a few tools that can separate the video into single pictures on the hard disk, like "ImageMagick"[4], "Virtual Dub"[5], etc. But such operation exhausts huge disk space and time. So normally it is not allowed to do in this way, considering that one 90 minutes video may cost hundreds of Gigabytes on hard disk.

Started on November 2002, Corvus V Corax developed a project "Motiontrack"[6] to detect the motion within an image. The idea is to compare the brightness of every pixel of two pictures, then use a filter to enhance the result, if there are some sectors whose brightness are larger than the others, there must be some motions occurred. This project could be a reference of our task to compare 2 images.

The project "Motiontrack" is based on the "libgd 1.0"[7], which performs quite well with the 256 color pictures or gray pictures, but does not support 16- or 24-color image. So the performance of "Motiontrack" is limited.

We may consult the ideas of the projects mentioned above to develop a new tool that can compare the frames in memory and can also deal with 24-color image.

1.6 Development Draft

Easy Extractor uses VC++ 6.0 as the development GUI, and "vfw32.lib" (Video For Windows library) as the main method to open and read the frame-stream from an AVI file. By using "vfw32.lib", arbitrary frame from the AVI file can be obtained and stored into memory. "cximage.lib"[1], an open C++ image processing and conversion library, is used to compare every two frames and then save them into different predefined frames (I-frame or P-frame). The reason to use "cximage.lib" is that it is open source, it is C++ based, and it has ability to deal almost all type of pictures.

When dealing with lossy videos, which are encoded by some lossy codecs, there will be some unavoidable distortions to the original scenes. These distortions can be considered as random noise. To avoid such random noise, filter can be used during comparing. The details will be interpreted in chapter 2.

To increase the encoding speed for the lossless source, "probe" is introduced. Originally the Easy Extractor compares every 2 adjacent frames to determine the frame's type, but for high-fps source, the idle scenes repeat for many times, we can compare the frame with the one e.g. 10-frames later. If there is no change occurred between them, we say there is no change within all these 10 frames. That means all these 10 frames are 0-frames. The "probe" machine can increase the encoding speed about 10 times or even more.

The Easy Player will also use VC++ 6.0 as the development GUI, and "cximage.lib" as the tool to read and draw the I- and P-frames. The interface of the player will include 3 functional buttons, responding "start", "pause" and "stop" commands, as well as a slide bar, responding "search within stream" command.

To evaluate the performance of the extractor and player with TSCC and XviD codecs, some small evaluation tools are developed together.

2 Some Concepts

There are some important terms that are frequently used in this project, such as Distance, Difference, and Filter. This chapter will describe in detail how they are defined, what are the uses of them.

2.1 Distance

Distance indicates how different 2 colors are. It determines whether 2 pixels could be considered as same one. Here we defined the Distance by absolute value:

$$\mathbf{D = |R1-R2| + |G1-G2| + |B1-B2|}$$

D	---	the distance between 2 pixels
R1, R2	---	the value of red in RGB format, (0~255)
G1, G2	---	the value of green in RGB format, (0~255)
B1, B2	---	the value of blue in RGB format, (0~255)

We can set a minimal value of the distance (threshold of different pixel), because if the distance is so small that the diminishment is generally imperceptible to the human eye, we can of course ignore it. In this project the default threshold of different pixel is 20. That means:

```
If (D<=20)   // the distance of 2 pixels are very small.
   D=0;      //ignore it.
```

In some other projects or literatures, the space distance is always used:

$$\mathbf{D^2 = (R1-R2)^2 + (G1-G2)^2 + (B1-B2)^2}$$

This definition describes the exact distance in 3-D space. But because the processing time of multiplication is exponentially larger than that of addition, to save the encoding time, here we just use the simply absolute distance instead of the space distance. The error between absolute distance and space distance reaches the maximal value when the R, G, B values are equal with each other.

$$\textbf{Max Error} = \sqrt{3}\ \ \mathbf{|R1-R2|} \qquad \text{when } R1=G1=B1 \text{ and } R2=G2=B2$$

By comparing with the threshold, Distance is used to determine whether 2 pixels can be considered as the same one or not. Since the threshold is set manually and experimentally, it is not a serious problem which type of Distance is used. In practice, the difference of using 2 definitions can be ignored.

2.2 Difference

The Difference of two images is defined as the amount of their different pixels.

$$\mathbf{Diff = \{ (i,j) \mid (D_{(i,j)} > 0) \}}$$

Diff	---	The difference of 2 images
D	---	The distance of 2 pixels
i,j	---	The coordinates in the images

The Difference between 2 images is used to judge which type the next frame is, I-frame, P-frame, or 0-frame. 2 thresholds are used: threshold of a new I-frame and threshold of a new P-frame. The source code is:

```
if (Diff > NewIFrameThreshold)
    Return -1;                // too many different points; It is I-frame
else if (Diff <= NewPFrameThreshold)
    Return 0;                 //almost no different points. It is 0-frame.
Else
    Return Diff;              //there exist some different points on the 2nd image, it is P-frame.
```

2.3 Filter

Filter is always used to suppress either the high frequencies of the images, i.e. the random noise or the low frequencies, i.e. enhancing the edges of the images.

The corresponding process in the real domain is to convolve the input image f (i,j) with the filter function h (i,j). This can be written as:

$$g(i,j) = h(i,j) \odot f(i,j)$$

The mathematical operation is identical to the multiplication in the frequency space, but the results of the digital implementations vary, since we have to approximate the filter function with a discrete and finite mask (kernel).

The discrete convolution can be defined as a `shift and multiply' operation, where we shift the kernel over the image and multiply its value with the corresponding pixel values of the image. For a squared kernel with size M× M, we can calculate the output image with the following formula:

$$g(i,j) = \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \sum_{n=-\frac{M}{2}}^{\frac{M}{2}} h(m,n) f(i-m, j-m)$$

Various standard kernels exist for specific applications, where the size and the form of the mask determine the characteristics of the operation. The masks for two examples, the mean and the Laplacian operator, can be seen in Figure 2.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Mean

0	-1	0
-1	4	-1
0	-1	0

Laplacian

Figure 2: Convolution kernel for a mean filter and one form of the discrete Laplacian.

The Mean filter is always used to smooth the image; otherwise the Laplacian filter is always used to enhance the difference.

2.3.1 Mean filter

The main idea of mean filtering is to replace each pixel value with the mean value of its adjacent pixels.

$$g(i,j)=\sum_{m=-\frac{M}{2}}^{\frac{M}{2}}\sum_{n=-\frac{M}{2}}^{\frac{M}{2}}\frac{1}{M*M}f(i-m,j-m)$$

The convolution kernels of Mean filter with mask range M=3 and with M=5 are like the following 2 tables:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

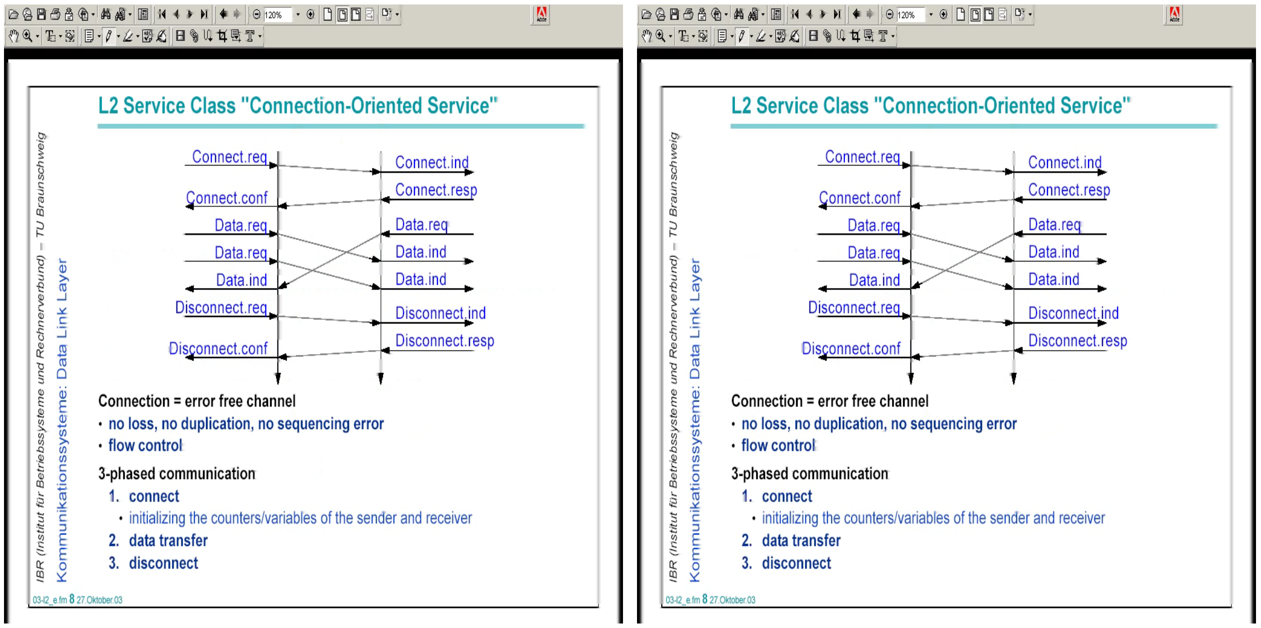
M=3

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

M=5

Figure 3: Convolution kernel of mean filters with mask range M=3 and M=5

Let’s have a look at the following 2 pictures: (5.png & 6.png)



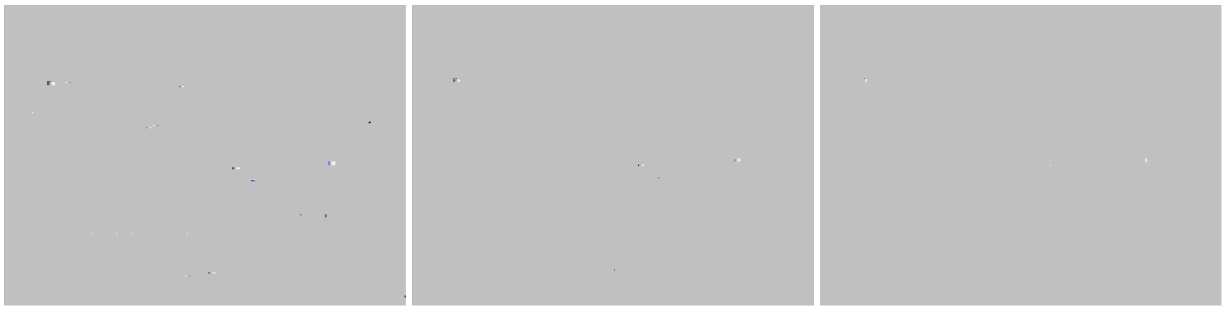
L2 Service Class "Connection-Oriented Service"

Connection = error free channel
- no loss, no duplication, no sequencing error
- flow control

3-phased communication
1. connect
   - initializing the counters/variables of the sender and receiver
2. data transfer
3. disconnect

Figure 4: alike pictures for mean filter and median filter.

The differences between above 2 images are printed in the following Figure 5 (I). Here we can only find some noise. (II) and (III) in Figure 5 are the differences after filters.



(I) the noise of picture (II) after 3×3 Mean filter (III) after 5×5 Mean filter

Figure 5: the noise before and after filtering

According to Figure 5, we can easily tell that these 2 images are the same. The differences between them are only some noise, which can be reduced by some filters. As seen in Figure 4 (II) and (III). The noise is enormously decreased after 3×3 Mean filter or 5×5 Mean filter.

From this comparing we can see that by using Mean filter the noise can be reduced effectively, therefore it can help us to determine whether 2 images are the same one without the infection of noise.

2.3.2 Median Filter

The main idea of median filtering is to replace the value of the pixel with the median of its neighbors.
For example, the 3×3 convolution kernel of median filter is:

123	125	146	125	124
123	125	130	132	129
119	120	99	136	109
110	120	130	131	140
120	103	126	127	119

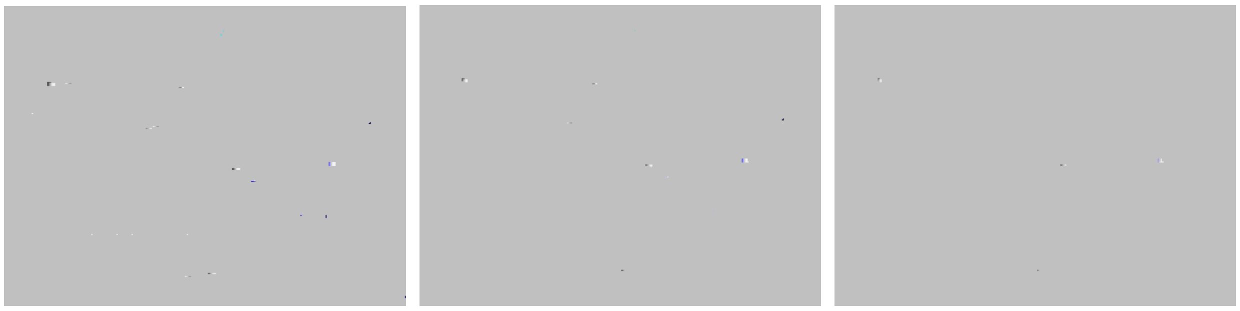
The neighborhood values:

125,130,132,120,99,136,120,130,131

Median value: 130

Figure 6: median filter method

Like mean filter, median filter can also reduce the noise efficiently. As seen in the following Figure 7:



(I) the noise of picture (II) after 3×3 Median filter (III) after 5×5 Median filter

Figure 7: noise before and after filtering

The effect of median filter is a little bit less than mean filter, with about 65% efficiency of mean filter. (This data is obtained by enumerating the different pixels after filtering.) In fact, median filter is not suggested due to another serious problem: imprecision. This problem will be detailed in chapter 5.

2.3.3 Summary

Mean filter and Median filter are two filtering methods always used to reduce noise in an image. In this project both methods will be tested with different sized kernels, and get the performance of each type of these filters.

3 Development of Software

There are totally four programs included in this project: extractor, player, player searching time tester, normal AVI file searching time tester. This chapter will only specify the first two, extractor and player. The two testers are only used in evaluating the performance of extractor and player.

3.1 Development Environment

The environment of development is:

- Operation system: Windows XP
- Developing tool: Microsoft Visual C++6.0
- basic libraries used: vfw32.lib, png.lib, cximage.lib

vfw32.lib:	Windows Multimedia SDK
png.lib:	library for reading and writing PNG image. [8]
cximage.lib	library for dealing with normal image, based on png.lib [4]

3.2 Implementation of Extractor

The main task of extractor includes: opening an AVI file and getting its file information，reading the video frame stream from the memory, providing an interface to configure the encoding process, comparing every 2 adjacent frames of the original video, and saving the frame as predefined frames as well as the information file. The executable file of extractor can be found as “\extractor\debug\extractor.exe” in appended CD.

3.2.1 Interface

1. Main interface

We can open an AVI file by clicking submenu <open> in menu <file>, or just clicking the shortcut of 📂, then do some configurations in the menu <control>, finally start the extracting process by the shortcut ▶.

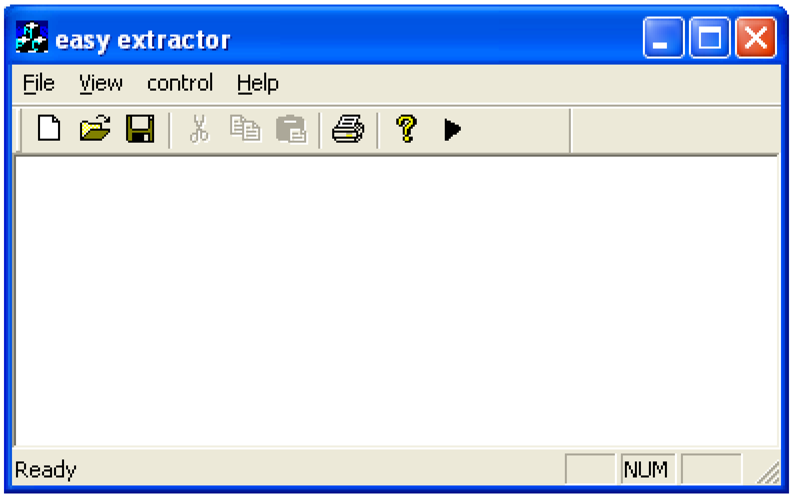


Figure 8: Interface of extractor

The menu <control> contains two submenus. One is <control->filter>, where we can set the type of filter and some thresholds for distance and difference. The other is <control->output>, where we set the types of output files and some properties during comparing.

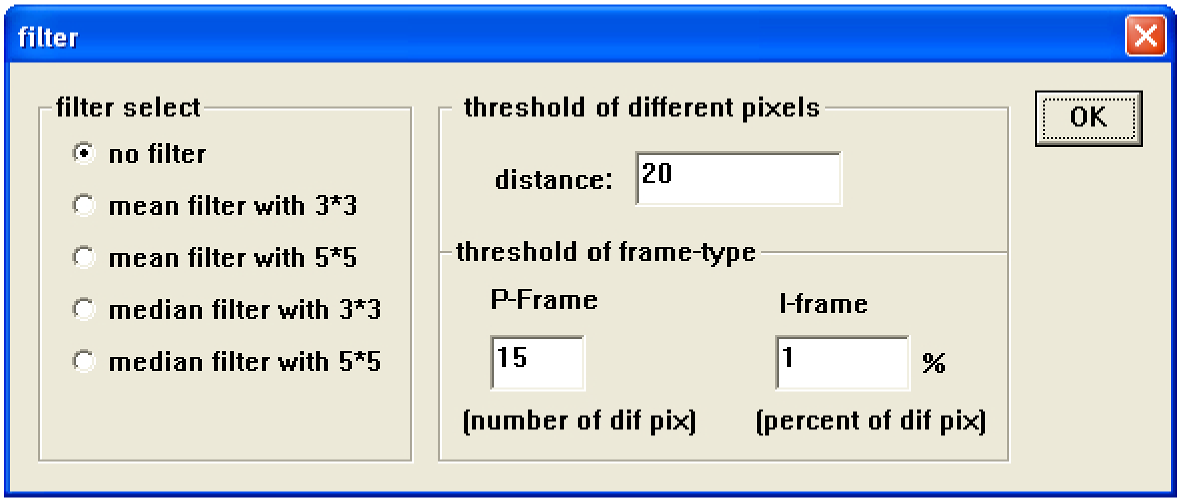
2. Interface of submenu <control->filter>:

The left part is to set the type and range of the filter. The right part is to set the 3 thresholds:

- DiffPixelThreshold, threshold of different pixels, which determines whether the colors of 2 pixels can be considered as the same one or not.
- NewPFrameThreshold, threshold of a new P-frame, to determine the type of frame, 0-frame or P-frame.

◆ NewIFrameThreshold, threshold of a new I-frame, to determine the type of frame, P-frame or I-frame.

Note that NewPFrameThreshold is absolute while NewIFrameThreshold is relative, as seen in Figure 9.



filter select
[✓ no filter] mean filter with 3*3 / mean filter with 5*5 / median filter with 3*3 / median filter with 5*5

threshold of different pixels
distance: 20

threshold of frame-type
P-Frame: 15 (number of dif pix)
I-frame: 1 % (percent of dif pix)

Figure 9: dialog of filter

For example, Figure 9 above implicates:

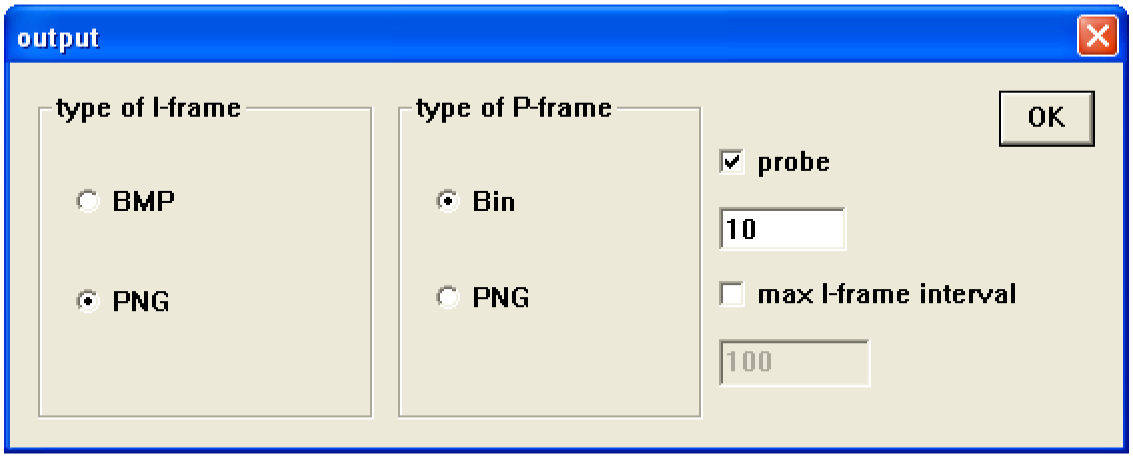
```
If (Distance of 2 pixel < 20)
    Distance=0;

Switch (Difference)     // the number of different points

    CASE: Diff < 15      CASE: 15<Diff< 1% of image    CASE: Diff> 1% of image
        Return: 0-frame;     Return: P-frame;              Return: I-frame;
```

Specifically the thresholds are chosen by experience. Normally the threshold of I-frame should be 1% ~ 10%, the threshold of P-frame should be less than 40, and the threshold of different pixels should be less than 30.

3. Interface of submenu <control->output>:



type of I-frame: BMP [✓ PNG]

type of P-frame: [✓ Bin] PNG

[✓ probe] 10

max I-frame interval 100

Figure 10: dialog of output

From this dialog plate we can select the outputting type of I-frames and P-frames. In Figure 10, the “max I-frame interval” is used for synchronization of reconstruction. The probe means that every ‘probe’ number of frames we make an exploration. If the frame, e.g. which is 10-frames later, is the same as the current one, we consider all the frames between them are 0-frames. The utilization of probe can increase the extracting speed greatly.

Specifically, the type of I-frame is strongly suggested to be PNG, since PNG file stores all the information of BMP file with much reduced file-size. The type of P-frame is suggested to be “Bin”, in respect of searching time, which will be detailed in chapter 4. The probe interval is depended on source video, large probe interval responds long slide idle-time. Normally the value is suggested to be 10 but should not be larger than 20. The max I-frame interval is only preferred with “PNG” as P-frame format. Its value should not be larger than 100, otherwise it will result in very long searching time.

3.2.2 Output Files

There are 3 types of output files: INF, I-frame file, and P-frame file.

- I-frames can be stored as BMP or PNG file, and PNG is preferred.
- P-frames can be stored as Bin or PNG file, and Bin is preferred.
- INF file contains the information of the original AVI file, as well as the information of I-frame and P-frame.

BMP/PNG files:

BMP files are used only to represent I-frames, while PNG files can be used to represent either I-frames or P-frames. The file name indicates the sequence number of the frame. For example, “15.png” is the 15th frame in the original AVI file. The frame type is indicated in INF file. In the dialog of <control->output>, the max interval of I-frame can be set to force I-frames.

Bin files:

Bin file is the default format of P-frame, which stores the differences to the last frame. For example, 17.bin means that it is a P-frame and is the annotation based on 16th frame, which can be 16.png, 16.bin or 16.bmp.

The structure of “Bin” file is like a table, every line represents one different pixel. As in the following example:

short x	short y	BYTE blue	BYTE green	BYTE red
123	36	FF	00	C9
236	90	C4	2F	22
…	…	…	…	…
-1(end flag)	-1	00	00	00

Table 1: format of Bin file (17.bin)

This file above indicates that the pixel (123,36), (236,90) … in the 17th frame should be changed into (FF,00,C9), (C4,2F,22), the other pixels which are not in this list should be maintained just as in 16th frame.

INF file:

INF file contains the information of original AVI file, the file-type of I-frame and P-frame, as well as the type of every frame: I-frame, P-frame, or 0-frame.

The file structure is like the following table:

```
{   AVIFILEINFO  avi_info;          //information of original AVI file
    Char         I_frame_type[4]   //normally "png"
    Char         P_frame_type[4]   //normally "bin"
    Byte         TypeOfFrame       //1st frame's type
    Byte         TypeOfFrame       //2nd frame's type
    Byte         TypeOfFrame       //3rd frame's type
    ......
    Byte         TypeOfFrame       //last frame's type
}
```

Table 2: format of INF file

- Avi_info records all the information of original AVI video.
- I_frame_type is the file type of the I-frame, which can be "png", or "bmp". Normally it is "png".
- P_frame_type is the file type of the P-frame, which can be "bin", or "png". "bin" is strongly recommended.
- TypeOfFrame indicates the type of every frame, which can be "2", "1" or "0". "2" indicates an I-frame, "1" indicates a P-frame, "0" indicates a 0-frame.

3.2.3 Process

The flow chart of extractor is shown in the following Figure 11:

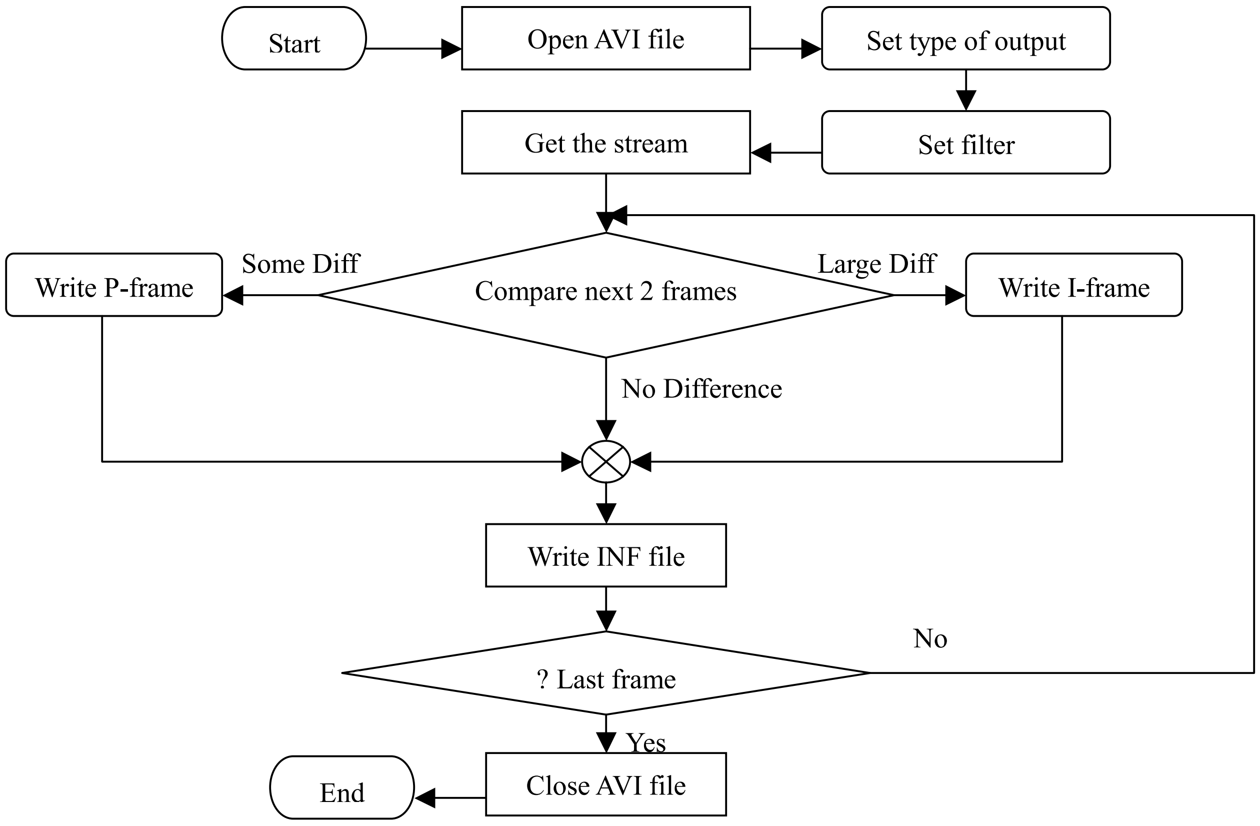


Figure 11: flow of extractor

3.2.4 CMyAvi Class

Here the class CMyAavi is introduced, because this class achieves most of the functionalities of this project. The main functionalities of this class include: initiating, comparing, and recording. Its members and functions are:

```
Public:   //construction & destruction
CMyAvi();                                       //Construct a CMyavi object
virtual ~CMyAvi();                              //destroy a CMyavi object

Public:   //methods
unsigned short GetHeight();                     //get the height of the AVI video's scene.
unsigned short GetWidth();                      //get the width of the AVI video's scene.
AVIFILEINFO* GetAviFileInfo();                  //get the information of AVI-file.
BITMAPINFOHEADER* GetBMPInfo();                 //get the information of the BMP images

void SetFilterRange(BYTE value);                //set the filter range
void SetFilterType(BYTE value);                 //set the filter type
void SetPixelThreshold(int value);              //set the threshold of different pixel
void SetPFrameType(BYTE value);                 //set the P-frame's type, PNG or Bin
void SetIFrameType(BYTE value);                 //set the I-frame's type, PNG or BMP
void SetIFrameInterval(int);                    //set the maximal I-frame interval
void SetProbeInterval(int);                     //set the interval of probe
void SetPFrameThreshold(unsigned short);        //set the threshold of P-frame
void SetIFrameThresdRelative(float);            //set threshold of I-frame, it's relative value

void AviInit(CString string);                   //open an AVI file according to the name,
                                                and get the information of its stream.
BOOL AviProcess(CWnd *pFrame);                  //main process to comparing every 2
                                                frames, recording the difference.

Private: //methods
void WriteBmpFile(int num);                     //write I-frame as BMP file
void WriteBinFile_P(long NumOfDiff);            //write P-frame as Bin file
void WritePngFile(int num);                     // write I-frame as PNG file
void WritePngFile_P(int num);                   //write P-frame as PNG file
void WriteJpegFile(int num);                    //write I-frame as Jpeg file
void WriteJpegFile_P(int num);                  //write P-frame as Jpeg file

void InitCompare();                             //initiate the comparing process, write the
                                                first frame as I-frame.
long Compare2Frame(int lframe, int lframe2);    //compare 2 frames. Pixel by pixel
long Compare2Frame_24(int lframe, int lframe2);//optimized function for 24-bit color video.
BYTE GetMedianValue(BYTE *a, BYTE num);         //get the median value of an array.
```

```
Private:  //members
CFile m_fileInf;                                  //file point to the INF file.
CFile m_fileBin;                                  //file point to the Bin file.

unsigned short m_wWidth;                          //the width of video scene.
unsigned short m_wHeight;                         //the height of video scene

long   m_lIFrameThreshold;                        //the threshold of I-frame
float  m_fIFrameThresdRelative;                   //the relative value of I-frame threshold
short  m_wPFrameThreshold;                        //the threshold of P-frame
short  m_nPixelThreshold;                         //the threshold of different pixel

int    m_iProbeInterval;                          //the interval of probe.
int    m_iMaxIFrameInterval;                      //the maximal interval of I-frame.
BYTE   m_byRunningProbeFlag;                      //identify whether the probe is running.

BYTE   m_byPFrameType;                            //the type of P-frame, PNG or Bin.
BYTE   m_byIFrameType;                            //the type of I-frame. BMP or PNG.
BYTE   m_byFilterRange;                           //the filter range
BYTE   m_byFilterType;                            //the filter type: mean filter, median filter
                                                  or no filter.

CxImage  m_Ximage1,m_Ximage2;                     //the 2 images, which are compared.
MyPixel *m_pTempPixel;                            //temp array to store the different pixels
CxImage *m_XimageP;                               //the image storing the difference, used
                                                  when P-frame is PNG or JPEG.

PAVIFILE          m_pAviFile;                     //point to the Avi file.
AVIFILEINFO      *m_pAviFileInfo;                 //the point to the information of Avi-file
PAVISTREAM        m_pAviStream;                   //point to the video stream
AVISTREAMINFO     m_AviStreamInfo;                //the information of video stream
PGETFRAME         m_pGetFrame, m_pGetFrame2;      //point to the video frame
BITMAPINFOHEADER *m_lpBMPInfoHeader;              //point to 1st BMP's header information.
BITMAPINFOHEADER *m_lpBMPInfoHeader2;             //point to 2nd BMP's header information.
LPBYTE            m_pBmpData, m_pBmpData2;        //point to the data pare of BMP image
```

3.3 Implementation of Player

The main process of player includes: opening an INF file, getting the information of the original AVI file as well as the file types of I-frame and P-frame, showing the I-frames and P-frames on scene one by one via a timer, seeking within the stream, and resizing the scene to video's original size.

Seeking within the stream includes 2 processes: searching and redrawing. Searching means looking for the I-frame preceding the preferred frame. Redrawing means drawing that I-frame and the following P-frames until this searched frame. When the player is in redrawing state, the video is paused.
The executable file of player can be found as "\player\debug\test4.exe" in appended CD.

3.3.1 Interface

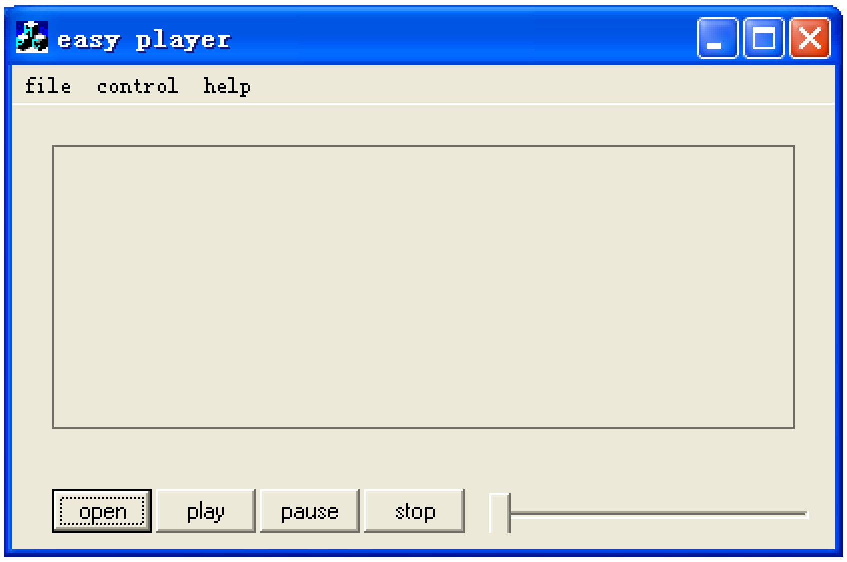


Figure 12: interface of player

We can open an AVI file by clicking submenu <open> in the menu <file>, or just clicking the button <open>. The menu <control> is used for some management during playing the frames, e.g. the submenu <play>, <pause>, or <stop>. Such functions can also be done easily by clicking the buttons down. Additionally videos can be played according to its original size. By dragging the slide bar, we can also seek within the stream.

Easy player now supports BMP/PNG files as I-frame, and Bin/PNG files as P-frame. To replay the slide video, we need to open an INF file, which is by default located in the directory "pictures". The directory "pictures" is created by extractor, and located in the same directory of the original AVI file.

3.3.2 Process Flow

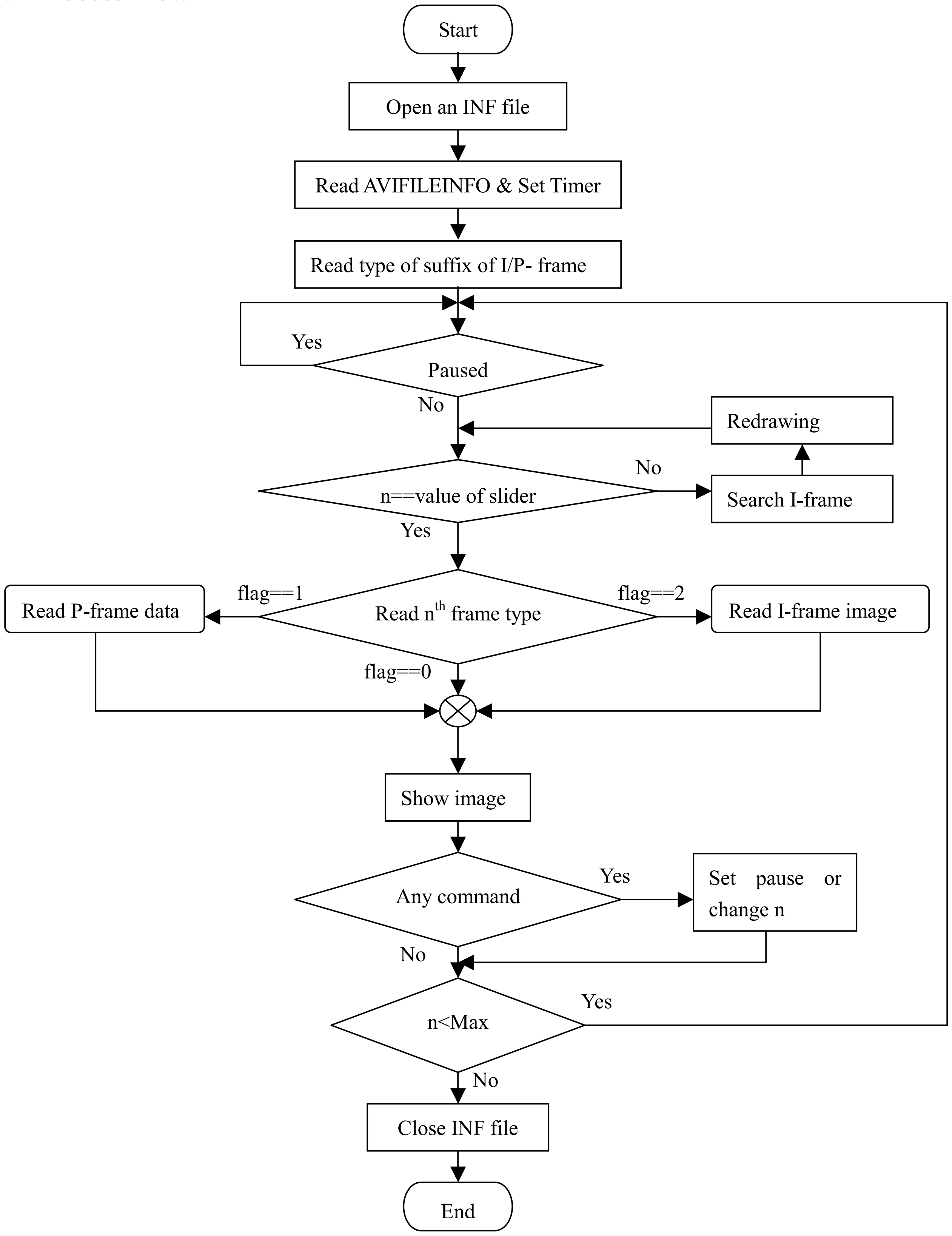


Figure 13: flow of player

3.3.3 Members and Functions

```
public:        //construction
CTest4Dlg(CWnd* pParent = NULL);                 // standard constructor

protected:     //methods
void OnTimer(UINT nIDEvent);                     //a timer to read and show frames
void ReadIFrame();                               //read I-frame
void ReadPFrame();                               //read P-frame
void Read0Frame();                               //read 0-frame
void ShowImage();                                //display the final image
void SliderChange();                             //when the slider is drawn
void Mix2Pic(CxImage *image1, CxImage image2);//mix 2 images

virtual BOOL OnInitDialog();                            //the initiation of dialog
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);     //system's function
afx_msg void OnPaint();                          //system's function
afx_msg HCURSOR OnQueryDragIcon();               //system's function

afx_msg void OnBUTTONopen();                     //open a video via an Inf file
afx_msg void OnBUTTONplay();                     //play the video
afx_msg void OnBUTTONpause();                    //pause the video
afx_msg void OnBUTTONstop();                     //stop the video
afx_msg void OnSize(UINT nType, int x, int y);//when the window is resized
afx_msg void OnOriginSize();                     //play the video on its original scene size.
afx_msg void OnOpen();                           //open a video via an Inf file
afx_msg void OnPause();                          //pause the video
afx_msg void OnPlay();                           //play the video
afx_msg void OnStop();                           //stop the video

protected:     //members
CButton   m_ButtonStop;                          //used to stop video stream
CButton   m_ButtonPlay;                          //used to play video stream
CButton   m_ButtonPause;                         //used to pause video stream
CButton   m_ButtonOpen;                          //used to open video stream
CSliderCtrl   m_Slider;                          //used to seek with video stream
CListBox  m_ListForDebug;                        //listbox, used only for debugging

byte   m_byInfOpenedFlag;                  //a flag to tell whether the Inf file is in open state
byte   m_byBeginFlag;                      //a flag to indicate the process in just in begin state
byte   m_byRedrawingFlag;                  //a flag to tell whether it is in redrawing state
byte   m_byPauseFlag;                      //a flag of paused

CFile m_fileInf;                                 //INF file.
AVIFILEINFO m_AviFileInfo;                       //the information of original AVI-file
```

```
char m_strIFrameType[4],m_strPFrameType[4];     //the type of I-frame and P-frame
DWORD m_dwFrameNum;                             //the sequence number of frame
CxImage m_Ximage;                               //the image finally shown on the scene
```

4 Testing and Evaluation

The performance of a codec is normally based on its encoding speed and encoded file-size. Therefore to get the performance within Extractor, XviD 1.0.1 and TSCC (TechSmith Screen Capture Codec), these two performances are tested for each method. In addition, the Subjective visual quality for the extractor and searching time for player are tested also.

The testing results are obtained by checking the property of the files displayed by Windows system, such as file size, file created time and last modified time. The encoding processes of XviD and TSCC are performed with "Virtual Dub". [5]

These tests are all performed by my personal computer. The results will differ according to the different test platform.

The platform of these tests:

- CPU: Celeron 1.7G
- Operation system: Windows XP
- Memory: 256M
- Hard disk: 7200r/s 2M cache

The configuration of XviD codec:

- Target quantizer: 2.00
- Maximum I-Frame interval: 400
- No B-VOPs
- Cartoon Mode

The configuration of Extractor (BIN):

- I-frame: PNG
- I-frame threshold: 1%
- no maximal I-frame interval
- P-frame: BIN
- probe:10
- no filter

The configuration of Extractor (PNG):

- I-frame: PNG
- I-frame threshold: 10%
- no maximal I-frame interval
- P-frame :PNG
- probe:10
- no filter

4.1 Extractor

There are totally 5 tests for extractor, first two are on lossless videos with little motion, other two are on lossless videos with some motion, and the last one is on lossy video with little motion. All these videos are records of the lectures in IBR, which are downloadable from IBR's webpage.

Little motion here means almost no movement in the source video, except the movement of mouse. Some motion means that there exist some obvious movements in the scene of source video, such as a sentence emerging from down to center of the slide.

Lossless video means that the source video is encoded by a lossless codec, while lossy video means the visual part of source video is encoded by a lossy codec.

4.1.1 Test1 (few motions)

File name:	tscc_mm-kap00.avi
Decompressor:	TechSmith Screen Capture Codec
Playing time:	1:24:43 with 1 fps.
Length:	5083 frames/ with 1 key-frame
Type of image:	1028*768 24-bit color
Size of file:	2.45 M
Link:	http://www.ibr.cs.tu-bs.de/users/kurtisi/public/slide_videos/tscc_mm-kap00.avi
Comment:	one lecture of “Multimedia-System” by Prof. L. Wolf

	TSCC	XviD	Extractor (BIN)	Extractor (PNG)
Frames/key-frames	5083/1	5083/15	159/75	159/75
Size of files (MB)	2.45	8.81	4.65	4.91
Exhausted time (m)	2.4	5.7	1.8	2.1

Table 3: test1 of extractor

4.1.2 Test2 (few motions)

File name:	tscc_mk-kap00.avi
Decompressor:	TechSmith Screen Capture Codec
Playing time:	1:38:34 with 1 fps.
Length:	5914 frames/ with 1 key-frame
Type of image:	1028*768 24-bit color
Size of file:	2.78 M
Link:	http://www.ibr.cs.tu-bs.de/users/kurtisi/public/slide_videos/tscc_mk-kap00.avi
Comment:	one lecture of “Mobile Communication” by Prof. L. Wolf

	source	XviD	Extractor (BIN)	Extractor (PNG)
Frames/key-frames	5914/1	5914/34	265/101	265/98
Size of files (MB)	2.78	10.8	4.52	5.17
Exhausted time (m)	2.5	5.35	2.35	3.5

Table 4: test2 of extractor

4.1.3 Test3 (some motions)

File name:	tscc_ea-kap00.avi
Decompressor:	TechSmith Screen Capture Codec
Playing time:	45:25 with 15 fps.
Length:	40887 frames/ with 1 key-frame
Type of image:	1028*768 24-bit color
Size of file:	2.65 M
Link:	http://www.ibr.cs.tu-bs.de/users/kurtisi/public/slide_videos/tscc_ea-kap00.avi
Comment:	one lecture of “Enterprise Application” by Prof. S. Fischer.

	source	XviD	Extractor (BIN)	Extractor (PNG)
Frames/key-frames	40887/1	40887/111	937/110	937/58
Size of files (MB)	2.65	24.1	7.22	7.33
Exhausted time (m)	18.4	38.8	9.3	18.5

Table 5: test3 of extractor

4.1.4 Test4 (some motions)

File name:	tscc_bsn-kap00.avi
Decompressor:	TechSmith Screen Capture Codec
Playing time:	29:41 with 15 fps.
Length:	26725 frames/ with 15 key-frame
Type of image:	1028*768 24-bit color
Size of file:	2.18 M
Link:	http://www.ibr.cs.tu-bs.de/users/kurtisi/public/slide_videos/tscc_bsn-kap00.avi
Comment:	one lecture of “Operation System and Network” by Prof. S. Fischer.

	source	XviD	Extractor (BIN)	Extractor (PNG)
Frames/key-frames	26725/1	26725/69	705/77	705/38
Size of files (MB)	2.18	17.9	6.54	5.79
Exhausted time (m)	12.0	25.8	5.7	12.6

Table 6: test4 of extractor

The relative average performances of these 3 testing codecs with lossless sources are shown in following figure:

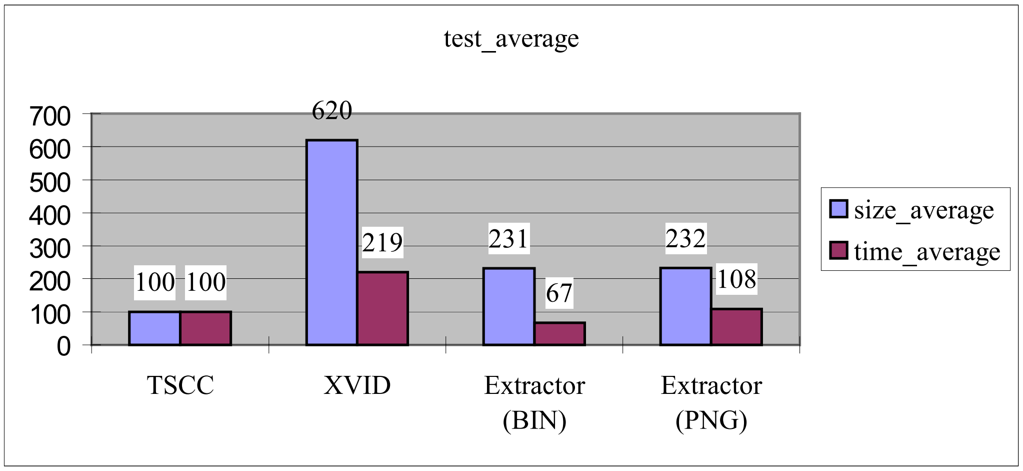


Figure 14: Average performances of testing codecs with lossless sources

4.1.5 Test5 (lossy source)

Filename:	ks-ws0304-03.avi
Decompressor:	XviD 0.9x codec
Playing time:	1:07:08
Length:	4028 / with 55 key-frames
Type of image:	1024*768 YUV

Size of file / video part:	35.4M / 20.0M
Link:	ftp://ftp.ibr.cs.tu-bs.de/lehre/ws0304/ks/ks-ws0304-03.avi
Comment:	one lecture of "Communication System" by Prof. L. Wolf

	source	XviD (1.0.1)	TSCC	Extractor (BIN)	3×3 Mean filter	5×5 Mean filter	3×3 median filter	5×5 Median filter
Size (MB)	20.2	10.0(*)	49.2	21.6	20.2	19.9	19.5	12.9
Time (m)	-	5.0	9.3	15	15	15	15	24
Subjective visual quality	Good	Good	Good	Good	Good	Good	Bad	Bad

(*): due to the improvement of coding efficiency between XviD 0.9x and 1.0.1 version

Table 5: test3 of extractor with lossy source

The relative average performances of these 3 testing codecs with lossy source are shown in following figure:

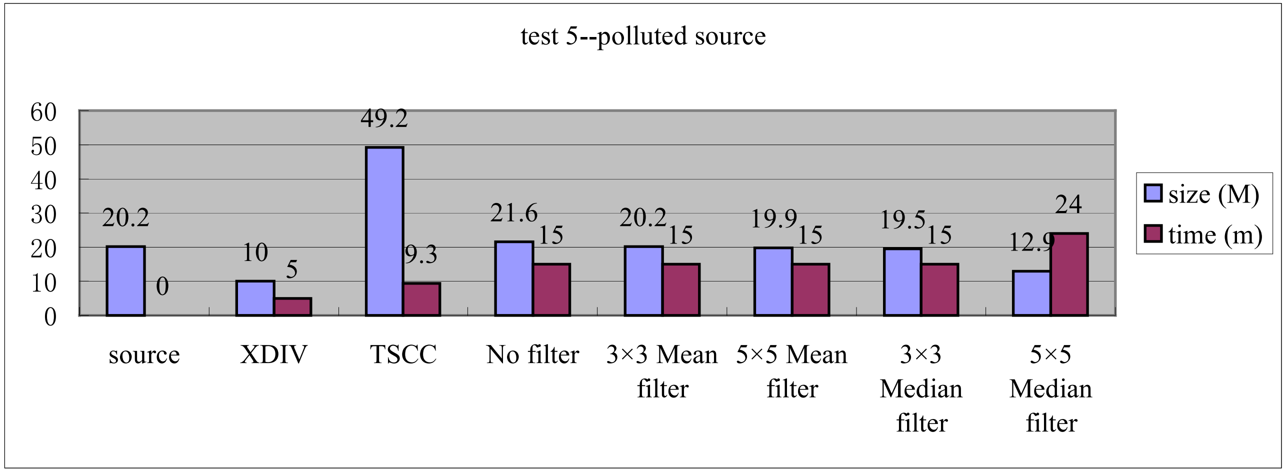


Figure 15: test of extractor with lossy source

4.1.6 Summary

Lossless source:

1. Compression

For the lossless sources, the size of the files resulted from extractor is only 30%~40% of the MPEG4 file with XviD codec. But relative to TSCC coded file, it becomes 1~2 times larger. The maximal interval of I-frame in XviD is one possible reason of its largeness. In extractor the maximal interval of I-frame is dynamic, or there is no maximal interval of I-frames at all. In short, the file-size is: TSCC : XviD : extractor=10 : 62 : 23

2. Encoding time

The encoding speed of extractor is fastest relative to XviD and TSCC. The encoding time is only 2/3 of TSCC and 1/3 of XviD. In short, the encoding time is: TSCC : XviD : extractor=10 : 22 : 7

3. Visual quality

Since extractor uses lossless encoding method, it can 100% reconstruct the source, as TSCC does. Although XviD is a lossy codec, the quality of the images is good enough.

The best codec for lossless sources is TSCC in respect of compression, or easy extractor in respect of encoding time. XviD performs worst in both aspects.

Lossy source:

1. Compression

The size of the files resulted from extractor is only 40% of that from TSCC, but almost 200% of that from XviD. In short, the file-size is: TSCC: XviD: extractor=5: 1: 2

2. Encoding time

The encoding speed of extractor is the slowest one within the 3 codecs. Its encoding time is 2 times larger than that of XviD, and 1 time larger than TSCC. In short, the encoding time is: TSCC: XviD: extractor=2: 1: 3

3. Visual quality

The visual quality of extractor without filter is always very good, because it uses lossless encoding method, and the pictures' quality with mean filter is also acceptable, but with median filter, is not. Median filter confuses the annotation with noise, where the annotations are too thin to distinguish from noise. So the median filter is always not suggested.

The best codec for lossy sources is XviD in both respects. The TSCC performs badly in respect to compression rate, and easy extractor performs badly in encoding time.

4.2 Player

There are three tests of player focusing on the searching time, which is obtained by embedded functions in source code. By drawing the slider bar into 8 arbitrary positions, the exhausted searching time will be displayed. The executable file for calculating searching time of player is "\PlayerSearchingTime\Debug\test4.exe ".

The searching time of TSCC and XviD is based on "vfw32.lib" of VC++ 6.0. It uses "AVIStreamGetFrame()" function to get the arbitrary frames, and "clock()" function to calculate the time exhausted. The executable program for calculating searching time for AVI file is "\AviSearchingTime\Debug\AviSearchingTime.exe ".

Test1, the sources are the results from test1 of extractor, the searching time: (ms)

	1	2	3	4	5	6	7	8	average
Extractor (BIN)	141	141	125	125	141	125	125	125	131
Extractor (PNG)	140	140	3203	109	125	1672	140	125	706
TSCC	172	140	141	313	187	78	578	266	234
XviD	297	500	375	406	422	125	454	469	381

Table 6: test1 of player

Test2, the sources are the results from test3 of extractor, the searching time: (ms)

	1	2	3	4	5	6	7	8	average
Extractor (BIN)	109	125	140	203	109	94	344	234	170
Extractor (PNG)	515	6250	500	17609	19844	9281	6641	484	7641
TSCC	172	234	187	297	344	93	766	250	293
XviD	156	297	266	187	140	250	266	94	207

Table 7: test2 of player

Test3, the sources are the results from test5 of extractor, the searching time: (ms)

	1	2	3	4	5	6	7	8	average
Extractor (BIN)	219	156	172	235	297	203	188	203	209
Extractor (PNG)	15125	6985	1016	625	1032	4032	594	597	3751
TSCC	1469	844	766	1032	829	94	2344	1000	1047
XviD	390	328	359	313	422	125	625	547	389

Table 8: test3 of player

The average searching time of each codec is shown in the following Figure 16:

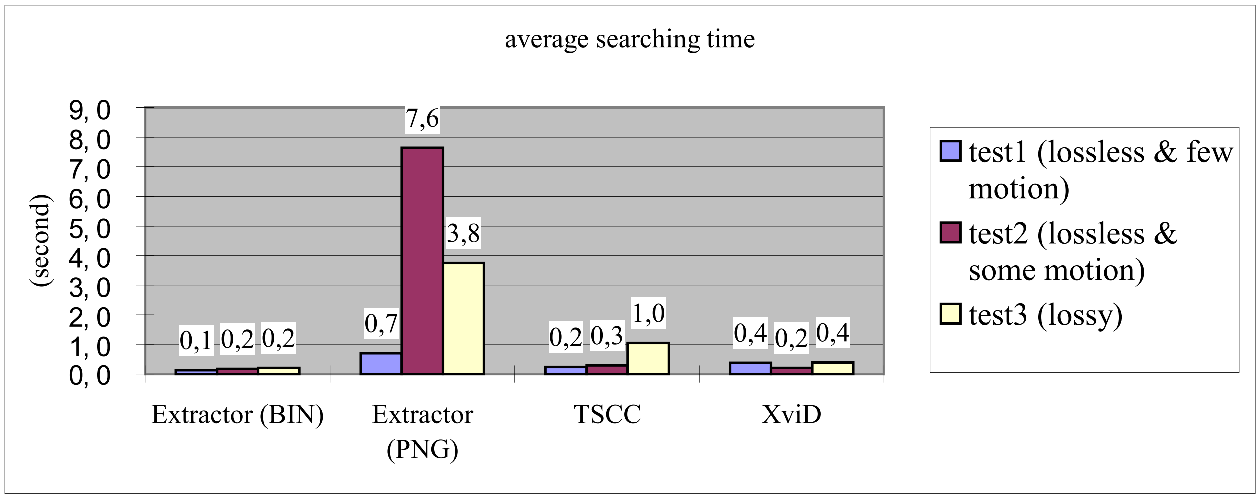


Figure 16: average searching time of each codec

From Figure 16, we can see that the average player searching time for “Extractor(BIN)” is about 200 ms, which is better than other codecs. And this result should be satisfactory for normal application.

The average player searching time for “Extractor(PNG)” is the largest one in every test. The searching time may cost several seconds, which is surely unendurable. The problem exists in the long processing time to decode dozens or hundreds of P-frames with PNG format succeeding an I-frame. To solve this problem, maximal I-frame interval could be used to force more I-frames. But this will in turn increase the file-size.

The searching time of TSCC is satisfactory only for lossless video, but not lossy video. And the XviD’s searching time is satisfactory for each case.

5 Existing Problems and Possible Solutions

Because this project is a simplex version, it has quite a few problems and questions. For example, whether the basic idea is correct or precise enough? Why does the compression rate cannot match TSCC? This chapter could be a consult for the future development.

5.1 Compression

In last chapter, we find that the compression rate cannot match TSCC, due to the fact that the easy extractor is insensitive to the movements of sectors in the image.

As mentioned in chapter 1, we just consider there are only 3 types of motions: quiescence, annotation, new I-frame. Although they are the main types of motions in the slide-video, there does exist some one we had ignored: movement. When some movements occur, the extractor just considers them as annotations, which result in a mass of similar P-frames. Although the appearance of movement is not very frequently, but that do result in large duplicated frames, as seen in Figure 17. This is the main reason that the compression rate of extractor cannot match TSCC.

One possible solution is to add a new type frame, movement frame, into the existent structure.

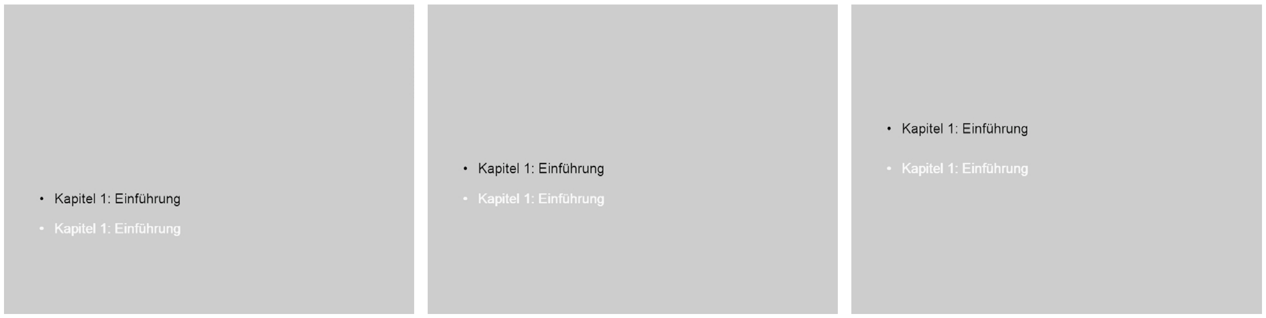


Figure 17: similar P-frames resulted from movement.

For the P-frame written as Bin file, it has also many redundancies. For example, one Bin file is:

X	2	2	2	2	6	6	6	9
Y	1	2	3	4	9	56	58	45
(R,G,B)	(FF,0,0)	(FF,0,0)	(FF,0,0)	(FF,0,0)	(FF,0,0)	(FC,34,D3)	(FF,0,0)	(12,36,F1)

Table 9: example of a P-frame file

The table above is a sparse matrix, and we have many methods to compress it, like “Compressed Sparse Row” to compress the X-coordinate, or using Huffman method to compress the color value, due to the high reiteration rate of the RGB value, which means the annotations made by professor are always in one color.

5.2 Extracting Time

The extracting speed is quite fast for the lossless source, which is about 3 times faster than XviD. It is because no noise exists in the original frames; therefore probe can be used during encoding. For the lossy source, the difference exists almost every 2 adjacent frames, so there is no chance to apply probe in process. To increase the speed for lossy source, it is necessary to do some optimization in the algorithm.

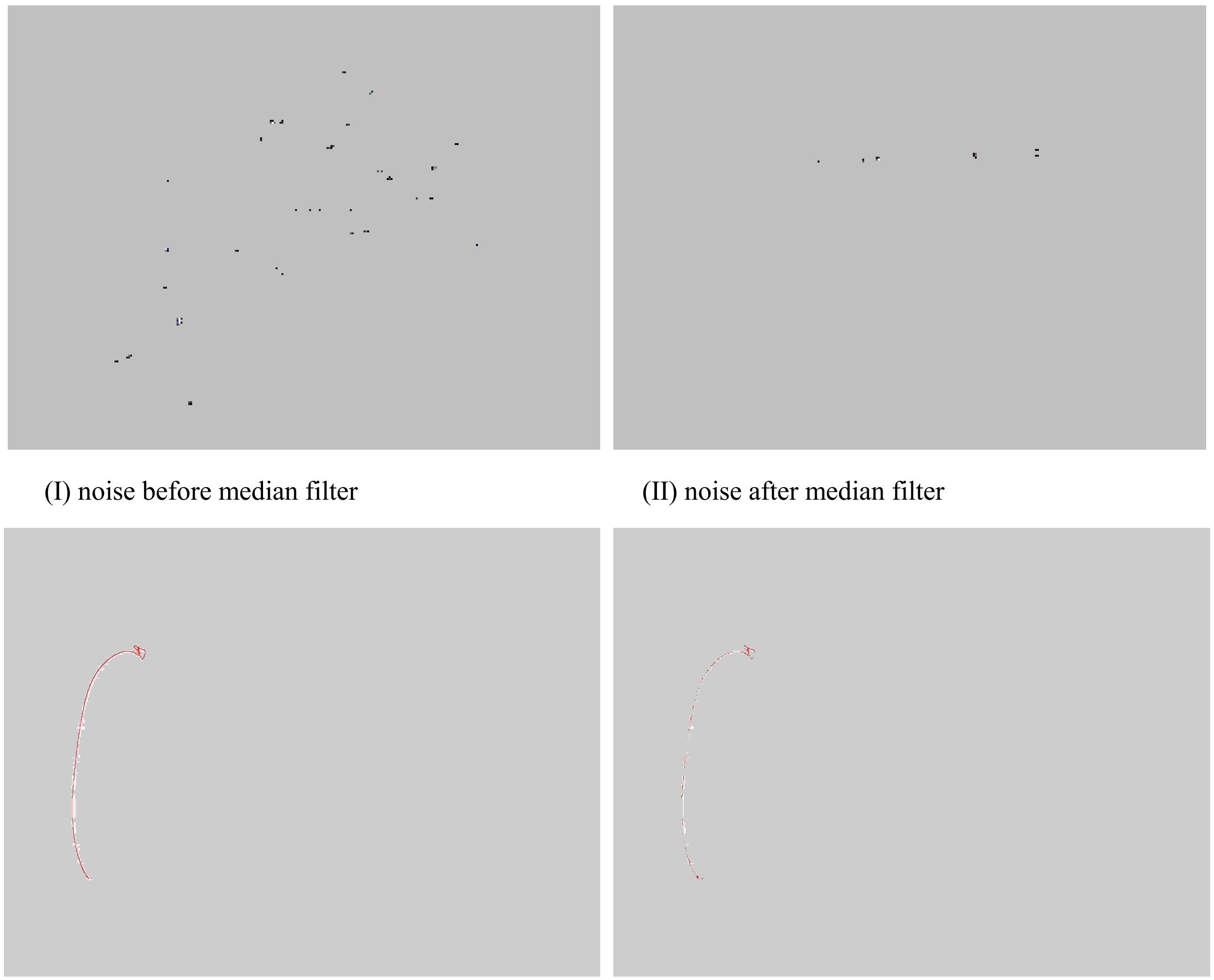
In addition, part of time is exhausted in the I/O file writing. For example, a 30-minutes slide video can result in thousands of output files, so in the release version it is better to use one single output file.

5.3 Searching Time

The player's searching time is satisfactory with BIN file as P-frame, but not satisfactory with PNG file as P-frame. The time is exhausted in reading and decoding dozens to hundreds of P-frames saved as PNG files, then mixing them with the previous frame. Some related algorithms in player should be optimized.

5.4 Filter

For the lossy source, filter can be used to reduce the noise, as seen in (I) and (II) in Figure 18. We can find that with median filter we get the most compressed files. But problem also occurs, since the lines of annotation are so slender, which in most case only occupy one or two pixels in width, that some non-noise information is also be filtered. This is shown in (III) and (IV) in Figure 18. So it is not suggested to use median filter in slide-videos.



(I) noise before median filter (II) noise after median filter

(III) annotation before median filter (IV) annotation after median filter

Figure 18: problem with median filter

The mean filter can distinguish the noise and the annotations quite well, which is recommended for the lossy source. There are many other filters not tested, and finding alternative filters with more accuracy is a future task.

5.5 XML

The demanded changes of slides are not performed explicitly because they cannot result in optimal results. We can find the placement of each slide just by setting maximal I-frame threshold larger, to say, 10% is enough for most slide videos. But the biggest problem exists in the ignorance of movements, as shown in Figure 17. A few movements result in large duplication.

To standardize the output file, one of the future works is to combine Inf file, I-frame files and P-frames into one XML file.

6 Conclusion

Easy Extractor and Easy Player provide a potential alterative to the MPEG4 codec for slide-videos. The compression rate, encoding time and searching time are better than XviD codec for the lossless source. Furthermore since extractor uses lossless encoding method, player can reconstruct the original video without any loss.

But since the extractor doesn't consider the other motions acting on the I-frame, like movement, which result in a mass of similar P-frames, its compression rate cannot match TSCC codec. For the problems and suggestions mentioned in chapter 5, there is a long way for this project to go.

With development of remote education, more and more slide-videos will be used. I hope that this project can be a one of the choice for the solutions of slide-videos.

7 Reference

[1] "Reference of CxImage"---http://www.xdp.it/cximage.htm
Davide Pizzolato, 08 Feb 2004

[2] "Working with AVI Files"---http://www.gamedev.net/reference/articles/article840.asp
Jonathan Nix, 11 Aug 1999

[3] "Digital Filters"---http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm
R. Fisher, S. Perkins, A. Walker and E. Wolfart 2003

[4] "ImageMagick"--- http://www.imagemagick.org
John Cristy

[5] "VirtualDub"--- http://www.virtualdub.org/
Avery Lee

[6] "Motion Detection"---http://motiontrack.sourceforge.net/readme.txt
Corvus V Corax

[7] “Gd Graphics Library”---http://stein.cshl.org/WWW/software/GD/
Lincoln D. Stein

[8] “png.lib” --- http://www.libpng.org/pub/png/
Greg Roelofs.