Order Number 9510628

# Scaled differential image compression for videoconferencing applications

Avcilar, Mustafa Tamer, Ph.D.

City University of New York, 1994



U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

# SCALED DIFFERENTIAL IMAGE COMPRESSION FOR VIDEOCONFERENCING APPLICATIONS

by

MUSTAFA TAMER AVCILAR

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1994

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

May 20, 1994
Date

Chair of Examining Committee

May 20, 1994
Date

Executive Officer

Prof. Michael Anshel
Prof. Ron Ashany
Prof. Dorothy Dologite
Prof. Stanley Habib

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

# ABSTRACT

## SCALED DIFFERENTIAL IMAGE COMPRESSION FOR VIDEOCONFERENCING APPLICATIONS

by

MUSTAFA TAMER AVCILAR

Advisor : Professor Ron Ashany

There many number of problems facing the compression of motion video. Even the most state of the art compression methods are not capable of handling motion video without sacrificing either one of the "compression speed" vs. "image quality" options[1]. In a process where it takes up to 30 frames of images to make 1 second of motion video, the process of coding images without significant delay is critical and often very difficult to overcome. Systems that use a fixed bandwidth for transmission of compressed video must also use methods that can dynamically adjust the compression ratio to maintain a constant rate of data transmission costing only some degradation of image quality.

We present a method called "Scaled Differential Image Compression" (SDIC) for Videoconferencing Applications and propose an effective means of gaining high compression ratios and real-time compression for the transmission of motion video. SDIC takes advantage of the similarities between consecutive image frames and provides a means to only store the pixels that have changed their value since the previous frame.

A new frame of an image gets compared to the previous one, creating a new image vector containing only those pixels that have changed. SDIC achieves compression by taking advantage of both spacial and temporal redundancies in images. Our algorithm also allows the definition of a "scale of change" which can dynamically adjust to achieve varying compression ratios to keep the information flow at a constant rate making it an ideal algorithm for use in videoconferencing applications that uses fixed capacity ISDN lines as the transmission medium.

I would like to express my sincere gratitude and appreciation to Prof. Ron Ashany who made it possible for me to complete this dissertation, for his support and encouragement. I like to thank Prof. Stanley Habib, Prof. Michael Anshel, and Prof. Dorothy Dologite for always being there for me and offer their help whenever I needed. I thank Mr. Joseph Driscoll for his endless support, kindness and help in answering my endless questions throughout my studies. I also like to thank Prof. Jacob Rootenberg for arousing my interest in this subject and his valuable guidance. And finally thank to all my friends and family for their everlasting support and trust in me.

# TABLE OF CONTENTS


1. Introduction . . . . . 1
2. Videoconferencing . . . . . 4
   2.1. Benefits . . . . . 4
   2.2. Cost of Videoconference . . . . . 6
3. Fundamentals of Video Compression . . . . . 9
   3.1. Considerations for a Compression Algorithm . . . . . 9
   3.2. Storage Media for Compressed Video . . . . . 11
   3.3. Asymmetric Applications . . . . . 12
   3.4. Symmetric Applications . . . . . 13
   3.5. Features of a Video Compression Algorithm . . . . . 13
      3.5.1. Random Access . . . . . 13
      3.5.2. Fast Forward/Reverse Searches . . . . . 14
      3.5.3. Reverse Playback . . . . . 14
      3.5.4. Audio-Visual Synchronization . . . . . 14
      3.5.5. Recovery from Errors . . . . . 14
      3.5.6. Compression/Decompression Delay . . . . . 15
      3.5.7. Support of Different Formats . . . . . 15
4. Scaled Differential Image Compression . . . . . 16
   4.1. Pseudo-code that Shows the Steps of the SDIC . . . . . 21
   4.2. The Advantages of this Method . . . . . 22
   4.3. Methods to reduce the size of the bit-map . . . . . 22
      4.3.1. Sparse Matrix Concepts . . . . . 27
      4.3.2. Bit-Map Scheme (BMS) . . . . . 27
      4.3.3. Single Indexing Scheme (SIS) . . . . . 30
      4.3.4. Double Indexing Scheme (DIS) . . . . . 33
      4.3.5. Huffman Coding . . . . . 36
         4.3.5.1. Huffman Codes . . . . . 37
         4.3.5.2. Construction of Binary Huffman Codes . . . . . 37
      4.3.6. Arithmetic Coding . . . . . 39
      4.3.7. Lempel-Ziv Coding . . . . . 41
5. Video Compression . . . . . 44
   5.1. How does it work? . . . . . 44
   5.2. Lossy Video Compression Algorithms . . . . . 45
   5.3. Discussion of Existing Video Compression Algorithms and their comparison to SDIC . . . . . 46
   5.4. Spacial Redundancy Reduction and the Use of Transform Coding . . 60
      5.4.1. Principle Component Transform . . . . . 60
      5.4.2. The Fourier Cosine Transform . . . . . 64
      5.4.3. DCT via FFT . . . . . 65
      5.4.4. Two-dimensional DCT by reduction to one-dimensional DCT 67
      5.4.5. Why is DCT used for the reduction of entropy

rather than DFT? . . . . . 68
5.4.6 The algorithm of DCT . . . . . 72
5.5. Other image coding methods . . . . . 73
5.6. Lossless Video Compression . . . . . 77
5.7. Why is Scaled Differential Image Compression Better? . . . . . 80
6.. ISDN Services and Video Conferencing . . . . . 82
6.1. Time Division Multiplexing . . . . . 84
6.2. ISDN Configuration . . . . . 85
6.3. The ISDN Communications Architecture . . . . . 88
7. Programs for Compression/Decompression . . . . . 90
7.1. Display.pas . . . . . 90
7.2. Compress.pas . . . . . 92
7.3. Decompress.pas . . . . . 93
8. Results . . . . . 96
8.1. Criterias for the Evaluation of Performance of a Video Compression codec . . . . . 104
8.2. Quantative Criteria . . . . . 105
8.2.1. Mean Square Error (MSE) . . . . . 105
8.2.2. Normalized Mean Square Error (NMSE) . . . . . 108
8.2.3. Averages . . . . . 112
8.2.3.1. Arithmetic Mean . . . . . 112
8.2.4. Scatter or Dispersion . . . . . 120
8.2.4.1. The Mean Deviation . . . . . 120
8.2.4.2. The Standard Deviation . . . . . 120
8.2.4.3. Significance of the difference between means . . 122
9. Conclusion . . . . . 125
10. Appendix . . . . . 126
11. References . . . . . 141

## TABLE OF FIGURES AND TABLES


Figure 1. Decrease in the Cost of Videoconferencing due to reduced Bandwidth Usage . . . . . 6
Figure 2. Sample Values for SDIC . . . . . 19
Figure 3. Basic MPEG Algorithm . . . . . 47
Figure 4. MPEG-1 Frame Coding . . . . . 50
Figure 5. Geometry for manipulation in MxN reference image U with (M+2p) x (N+2p) image $U_r$(search area) in the previous frame 55
Figure 6. ISDN Decomposition . . . . . 87
Figure 7. Bitmaps for each frame after compression . . . . . 98
Figure 8. Pixel files with intraframe . . . . . 100
Figure 9. Pixel files without the intraframe . . . . . 101
Figure 10. Total size of the frames after compression . . . . . 103
Figure 11. Mean Square Error . . . . . 107
Figure 12. Normalized Mean Square Error . . . . . 111
Figure 13. Energy Distribution, Picture 1 . . . . . 113
Figure 14. Energy Distribution, Picture 2 . . . . . 114
Figure 15. Energy Distribution, Picture 5 . . . . . 115
Figure 16. Energy Distribution, Picture 10 . . . . . 116
Figure 17. Energy Distribution, Picture 30 . . . . . 117
Figure 18. Arithmetic Mean . . . . . 119
Figure 19. Significance . . . . . 124

Table 1. Videoconference System Related Recommendations . . . . . 8
Table 2. List of Storage Media for Digital Compressed Video . . . . . 12
Table 3. Asymmetric Applications of Digital Video . . . . . 12
Table 4. Symmetric Applications of Digital Video . . . . . 13
Table 5. Comparison of H.261 to MPEG-1 . . . . . 49
Table 6. Comparison of the performance of Block Matching Algorithms . 56
Table 7. Compression ratios on different test pictures using lossless JPEG 78
Table 8. Total size of bitmaps after compression . . . . . 97
Table 9. Total size of pixelfiles after compression . . . . . 99
Table 10. Total size of all files after compression . . . . . 102
Table 11. Mean Square Error values . . . . . 106
Table 12. Comparison of MSE values . . . . . 108
Table 13. Normalized Mean Square Error values . . . . . 110
Table 14. Arithmetic Mean values . . . . . 118
Table 15. Significance . . . . . 123


## TABLE OF APPENDICES


Appendix 1. Source code of programs used in our experiments . . . . . 140

# 1. INTRODUCTION

During the past few years various methods were developed for compressing still video images with relatively high compression ratios and acceptable image quality making it feasible to store, view and transmit them using computers. Products using such algorithms are already available in the market[2,3]. JPEG being the most popular standard among all, is currently integrated into many computer applications[4].

A motion-video segment with acceptable image quality requires the storage/transmission of 30 frames per second, which adds up to huge amounts of data to be stored or transmitted even within short periods of time. Interframe compression methods in addition to intraframe compression must be implemented in order to achieve compression ratios that may make this a practical technology. In other words, use of JPEG or similar algorithms to compress individual frames that makes a motion-video sequence is not sufficient. Algorithms for the compression of motion-video must also take advantage of the redundancies between the consecutive frames.

The second important requirement for applications that require real-time transmission of motion-video between two or more locations is the use of fast and efficient compression/-decompression algorithms.

Scaled Differential Image Compression with its minimum complexity, ease of implementation, and acceptable compression ratios offers an acceptable compromise between compression speed and compression ratio. That is why we believe Scaled

Differential Image Compression is a suitable method for videoconferencing.

In our second chapter, we introduce videoconferencing, its benefits and the reasons why it is so important for businesses, education and public institutions. Videoconferencing may soon become a preferred medium for business meetings and a cost effective way to bring education to a wide range of students/learners. One factor that will make videoconferencing more accepted and widely used is the reduction of the cost of videoconference sessions through the use of smaller bandwidth channels. This can only be achieved through the use of more efficient compression algorithms. Figure-1 in chapter two clearly shows the drastic drop in the cost of videoconferencing due to the use of compression. Third chapter describes the issues related to the compression of motion-video. In the design of our algorithm, we have tried to implement most of the listed requirements, especially the ones that are more relevant to videoconferencing. Some of these requirements are random access, fast recovery from errors, and the ability to do fast forward searches by only displaying the intraframes. "Constant bit rate" which is an important requirement for a compression algorithm for videoconferencing applications is also implemented in SDIC. Fourth chapter, explains SDIC in detail, considers its advantages and shows the methodology we have used to solve certain problems that we encountered for the compression of bit-map tables. Efficient compression of bit-map tables which have to be saved in addition to pixel vectors for each frame is essential to achieve high compression ratios. We found LZW coding most suitable for our application. In chapter five, we discuss some of the existing video compression algorithms and compare their advantages and weaknesses in detail. We also compare other methods with SDIC and emphasize the advantages of SDIC for videoconferencing.

In chapter six, we discuss another important component of video compression for videoconferencing. Integrated Services Digital Network(ISDN) is the preferred and possibly the most advantageous transmission medium for videoconferencing sessions. Therefore, any compression algorithm for videoconferencing must carefully consider the properties of ISDN lines and seamlessly integrate with the channel hierarchy of ISDN. In the next chapter, we present and discuss our computer programs written to implement SDIC compression/decompression and explain their operation in detail. In chapter eight, we present the results that we obtained after running our tests on sample images.

# 2. VIDEOCONFERENCING

The business meeting- crucial to effectively conducting business- is often costly, time-consuming and difficult to coordinate, particularly when participants involved from multiple, long-distance locations. Yet the communication imparted at meetings is essential to a successful business. Most business meetings today, are conducted at a central location where one or many participants of the meeting has to travel, sometimes great distances to attend. The cost of business travels is always a major source of expense for companies.

Videoconferencing is ready to change this. Eliminating these traditional roadblocks is videoconferencing, a rapidly growing strategic management tool that reduces travel costs, improves productivity and communications, and facilitates decision-making.

Videoconferencing is quickly becoming recognized as a cost effective alternative to many face-to-face meetings. In addition to cost savings, competitive and strategic advantages are also being realized by increased productivity and time efficiency.

## 2.1. Benefits

- Reduces costs incurred by face-to-face meeting.

  Participants of a project or parties to a business operation may reach each other right from their office during any convenient time. The videoconferencing does not have to be between the participants from two sites but may also include three or more by using point-to-multipoint bridges. This brings the meetings to each participants' office eliminating the need for any one to travel. The benefits of

videoconferencing is even more emphasized for businesses where constant travel is required for supervision of projects or consulting. The establishment of global ISDN lines for the transmission of videoconferencing signals even eliminates international travel. All of this adds up to considerable savings in the travel expenses for any business.

- Saves time

  The elimination of travel for meetings not only saves the cost of transportation-/lodging/food etc. but also saves the time spent on travel.

- Increased productivity

  Participants can be more focused and relaxed during the meetings as they do not have to go through long tiring trips.

- Expands training potential

  Offers a whole new array of training potential for employees/managers. The use of videoconferencing for the training of employees in various remote locations offers a cost effective solution to reaching people whom would otherwise be very costly to fly to a central training facility. A professional trainer who is located at one location may be available to many employees from different sites, which offers the possibility to use the most qualified trainer for the job rather than hiring many trainers and sending them to remote sites/factories/offices.

## 2.2. Cost of videoconference

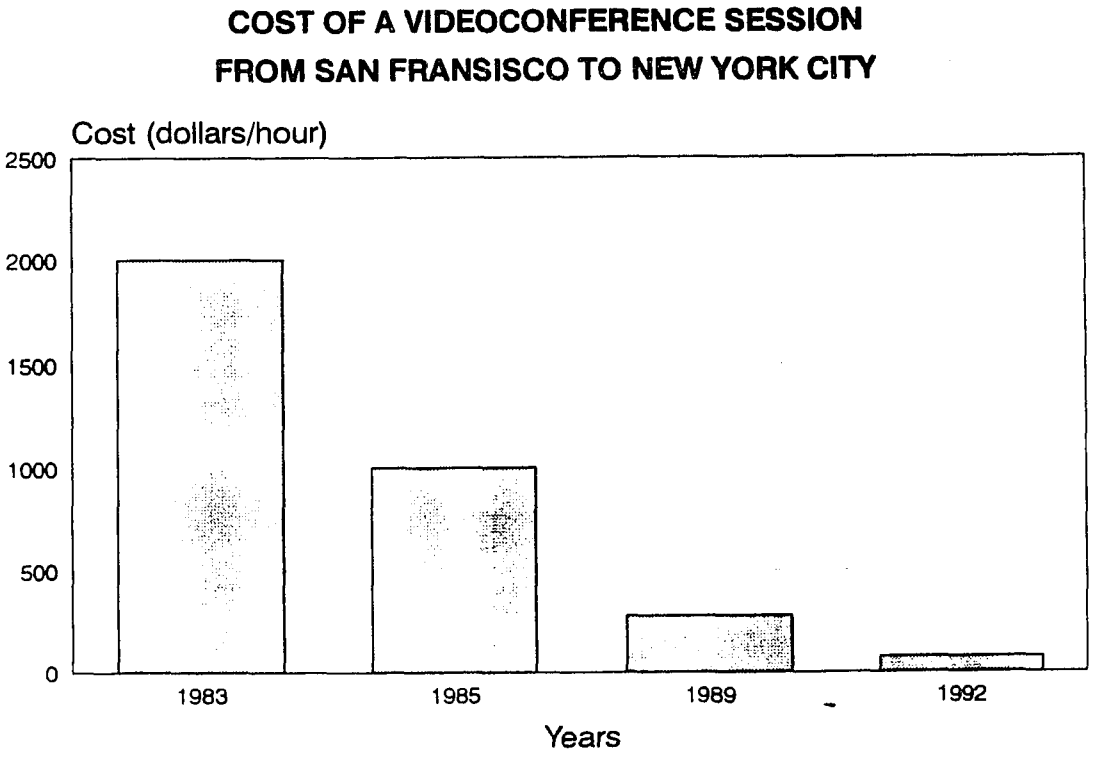



Figure 1. Decrease in the cost of videoconferencing due to the reduced bandwidth usage

This sharp decrease in the cost of videoconferencing is mostly due to development of ever more efficient compression methods used in the transmission of audio/video signals. A two-way interactive videoconference during early 1980's required a minimum bandwidth of 3 Megabits/sec whereas today, a similar conference can be conducted using a bandwidth of a mere 128 Kilobits/sec. This also demonstrates the importance of video compression for these types of applications where higher bandwidth means more expensive

communications. Compression enabled the use of regular switched-digital phone lines for a videoconference rather than special network connections or even satellite communications thus making the cost of communications equal to the cost of a regular phone call.

The technical difficulties for the transmission of video and audio signals still exist. Video signals can be transmitted either directly as analog signals or the analog signals can be converted to digital signals and transmitted that way. The digital transmission offers a higher picture quality especially if it is transmitted over long distances.

High bandwidth signals can take up to 100 Mb/s, but by compressing the digital signals, the bit rate can be reduced to just a few Mb/s or even less.

Standardization of the codec systems and the transmission method used is another important factor in the wide acceptance of the videoconferencing systems. There are three color TV standards: NTSC used in Japan and North America, PAL used in Western Europe, and SECAM used in Eastern Europe. Obviously additional processing is required to convert between these different systems to make them interwork. To accommodate these differences, a Common Intermediate Format (CIF) has been defined in CCITT Recommendation. With this CIF, different color TV systems can intercommunicate without any additional processing. The video signal of any of the three color TV standard systems is coded into Common Intermediate Format on the transmission side and the inverse operation is performed on the receiving side.

| Rec. No. | Title |
|---|---|
| H.261 | Video codec for audiovisual services at px64 kb/s |
| H.242 | System for establishing communication between audiovisual terminals using digital channels up to 2 Mb/s |
| H.221 | Frame structure for a 64 to 1,920 kb/s channel in audiovisual systems |
| H.230 | Frame synchronous control and indication signals for audiovisual systems |
| H.320 | Narrowband visual telephone systems and terminal equipment |

Table 1. Videoconference system related recommendations

# 3. FUNDAMENTALS OF VIDEO COMPRESSION

## 3.1. Considerations for a compression algorithm

Great many number of compression algorithms are developed for a wide variety of applications such as the compression of medical images, of images transmitted from space crafts or compression of video. The nature of each application defines several factors that has to be considered in the design of the most suitable compression algorithm. We would like to review some of these factors that requires consideration.

* Image quality vs. compression ratio : Either one these factors must be compromised for almost all lossy compression methods, as higher compression ratios can mostly be achieved by reducing the quality of the reconstructed image. Reducing the entropy of the images, in other words, reducing the image quality allows higher compression ratios. Certain applications have a higher tolerance for this, such as a videophone. It is usually acceptable to have 10 frames per second video speed and somewhat blurred images. Certain applications such as the transmission of images taken from satellites have very little if any tolerance to image degradation due to compression. These applications either use no compression at all or use lossless methods which offer considerably less compression but no loss in image quality.

* Complexity of the compression algorithm vs. compression speed : Certain highly complex compression algorithms seem to give extremely high compression ratios, such as fractal compression. This method achieves compression by finding

algorithms necessary to reduce the scene to a set of fractals. Compression ratios greater than 1000 to 1 is achieved with this method but it takes about 100 hours to compress each image[5]. It is clear that this is completely unacceptable for most applications. JPEG applications can compress an image within a second or so, MPEG, which uses a complex algorithm to reduce the redundancies between consecutive frames may take longer to do the job. The major factor for a videconferencing system is the compression speed, whereas an archive system will be more concerned with the compression ratio.

* Similar compression/decompression algorithms : For applications that transmit and receive compressed video, it is desirable to have similar complexity for both the compression and decompression algorithms.

* Channel error tolerance : Compression of information before transmission makes it especially vulnerable to interference and data loss. Efficient error correction codes must be used for applications that can not tolerate data loss during transmission.

* Constant bit rate vs. constant quality : For most applications a constant image quality must be maintained. This maybe achieved by variable bit rate transmission to compensate for changing information rate depending on the content of the pictures transmitted. This requires the allocation of a larger bandwidth that is large enough to transmit that may contain the highest entropy.

Sometimes the reverse operation may be necessary if the allocated bandwidth is fixed. The compression ratio should be adjusted to maintain a constant rate of data transmission costing some degradation of image quality. This is an extremely useful feature for videoconferencing as the preferred transmission medium is the ISDN lines in which all channels have a fixed bandwidth. During moments of fast motion or extreme changes in the view, it will not be possible to increase the bandwidth to compensate for increased data transmission but this feature may automaticly increase the compression factor to maintain a constant data rate.

## 3.2. Storage Media for Compressed Video

Various data storage media available in the market can be used for the storage of digital compressed video information. The major distinction between these media is their access method and access speed. Devices such as Digital Audio Tape, although perfectly suitable for the storage of video, can not offer random access which is a must for multimedia applications. CD-ROM is an important candidate for an ideal storage medium as a low cost and random access device. But its read-only nature is a major drawback for many applications. The wide availability of Writable Optical Disks will eliminate this drawback. Magnetic Hard Disks will probably continue to be an alternative regardless of their relatively high prices as they offer the fastest access speeds.

Local Area Networks, ISDN lines and even regular phone lines will be the most likely candidates for the transmission of compressed video images.

| |
|---|
| CD-ROM |
| DAT |
| Video Disks |
| Writable Optical Disks |
| Hard Disks |

Table 2. List of Storage Media for digital compressed video.

### 3.3. Asymmetric Applications

Another important distinction between digital video compression algorithms is the way compression and decompression performed. Certain applications do not need on-line compression, such as a video movie recorded on a CD-ROM. This video is compressed once at the manufacturing facilities and recorded on the CD-ROM, viewing device only runs decompression algorithm to display the movie. These types of applications may use more complex and better compression algorithms as the compression time is not a critical factor. Table 3 lists such applications

| |
|---|
| Home Movies |
| Archiving |
| Multimedia Presentations |
| Video Databases |
| Video Games |

Table 3. Asymmetric applications of digital video

| |
|---|
| Videoconference |
| Videotelephone |
| Desktop Videoconferencing |
| Videoconference over WAN's |

Table 4. Symmetric applications of digital video

### 3.4. Symmetric Applications

These are the types of applications that require on-line compression and decompression of digital video. A videoconferencing system simultaneously runs both the compression algorithm before transmitting and decompression algorithm to display the images received. The speed and efficiency of compression is clearly as important as the quality of the images transmitted. The quality and efficiency has to be optimized against the compression speed. Table 4 shows the symmetric applications of digital video.

### 3.5. Features of a Video Compression Algorithm

Compression of video is applied to a wide variety of applications each of which has different requirements in terms of transmission, presentation and manipulation of compressed video. A general video compression algorithm needs to address most of these issues. Below, we discuss the features that has to be implemented in a video compression algorithm.

#### 3.5.1. Random Access

This is possibly the most essential feature of a compression algorithm. The viewer should be able randomly access the video starting from any location. Algorithms that store/transmit only the changes in consecutive images must make sure "intraframes", frames that contains the complete set of pixels, are inserted into every so many frames so that they can be used as random access points. It is expected that there should be a random access point at least every second throughout the video segment.

### 3.5.2. Fast Forward/Reverse Searches

It should be possible to scan the compressed video in fast forward or fast reverse. Any video compression algorithm to be used in home-movie applications must offer this feature.

### 3.5.3. Reverse Playback

This option should be considered for home-movie and multimedia applications but may not be necessary for most other applications.

### 3.5.4. Audio-Visual Synchronization

Special attention must be paid to the synchronization of audio with video during playback. It is impossible to keep a constant decompression speed for display as compression is achieved in varying ratios depending on the image content. This causes the audio to move ahead of the video or vice versa. In order to prevent this situation, certain synchronization signals must be inserted into the audio/video tracks that ensures synchronized output.

### 3.5.5. Recovery form Errors

Error correction codes which reduces the error rate are implemented in almost all forms

of communications and storage media. Error correction methods have certain limitations and can not guarantee a 100% error free communication. Certain errors may still be transmitted without detection or correction. Should such a condition arise, the algorithm must be able to recover from its error condition within the minimum amount of time and only cause a temporary reduction in image/audio quality.

### 3.5.6. Compression/Decompression Delay

Symmetric applications of video requires that this delay be minimum so that a real-time communication can be achieved. This may not be true for certain applications such a video archives or video-disk players. Manufacturers of these products may use compression algorithms that may take extended periods of time to do the compression. It is only the speed of the decompression algorithm that is critical.

### 3.5.7. Support of Different Formats

Variety of video formats should be supported to ensure global use of the method. In addition, different frame rates, raster sizes should also be supported and preferably selectable by the user.

# 4. SCALED DIFFERENTIAL IMAGE COMPRESSION

Differential Image Compression takes advantage of the similarities between consecutive image frames and provides a means to only store the pixels that have changed their value since the previous frame[6]. A new frame of an image gets compared to the previous one, creating a new image vector containing only those pixels that have changed. After saving the complete Intraframe pixels, we only save the difference of the coming frames from the previous one for a sequence of 30 frames (Equivalent to 1 second of motion video). Then another complete Intraframe is saved and the next 30 frames are compared to the previous one starting from the Intraframe. Using a new Intraframe after every 30 frames enables random access to any part of the video with an average error of $\pm$ 0.5 seconds. Each compressed frame must also have a Bit Table associated with it which contains a bit for each byte of the image. Each consecutive bit of the Bit Map corresponds to the consecutive bytes that represent each pixel. When a byte that represent a pixel that have changed is saved, the bit that corresponds to that byte in the bit map is set to one, so that during decompression that byte can be written back to its correct position. Scaled Differential Image Compression allows the selection of a "scale of change" which defines the scale of tolerance which will accept a pixel as changed or unchanged in comparison to the previous one. In other words, if the scale of change is $\pm$ 5, the previous pixel has a value of 25 and the following one 29, the algorithm stores this pixel as 25, as its new value is within the defined scale. This process shows the lossy nature of the SDIC algorithm. Scale of change factor improves compression ratio as it discards slight variations between pixel values due to lighting conditions and accepts them as the same pixel as the previous. Another advantage is that, scale of change factor

allows the system to be dynamic in terms of its response to the bandwidth requirement for the transmission. The system automatically senses the increase in the information content of the image due to fast motion or complex image content and it can adjust this scale to increase the compression ratio to compensate for the increased bandwidth requirement. This allows the system to maintain a constant transmission bandwidth regardless of the changing image content.

This nature of SDIC makes it an ideal algorithm to be used for videoconferencing applications that uses ISDN lines as the transmission medium. Circuit-switched ISDN network only allows the use of fixed bandwidth communication channels which presents a problem for the transmission of video. The amount of information transmitted varies as the scenery and the content of the images change. This may add up to information rates beyond the capacity of the transmission channel. The scale of change factor in SDIC , which can dynamically be adjusted by the system, offers a unique and efficient solution to this problem.

We have also investigated the hardware implementation of our SDIC algorithm and realized that using a variation of Set Associative Memory we may perform this comparison almost simultaneously. Set Associative Memory system which in this case compares a set of values with a set of previous values instantly sets flag bits to indicate the bytes that are different then the permitted range. This allows a even faster implementation than software.

We can use the following example to explain our method;

Consider the values in Figure-2 each of which represents the pixel values of a 3 x 3 pixel image. "Scale of change" is $\pm$ 2, which means pixel values that are not more than $\pm$ 2 will be considered unchanged. Each time a comparison is made between the first frame and the next, the pixel values that are different, set the flag associated with that specific byte. Only those bytes with a flag of one is saved in storage. The problem of keeping track of which byte belongs to which address can be solved by associating a "bit-map table" to each frame which contains a "1" for each byte stored in memory and a "0" for those that are not saved.

This method achieves compression by storing only those bytes that are changed in reference to the previous frame. But the fact that each frame must have a bit map saved with it creates an overhead which might even take away the whole savings if there are a great number of bytes changed between frames.

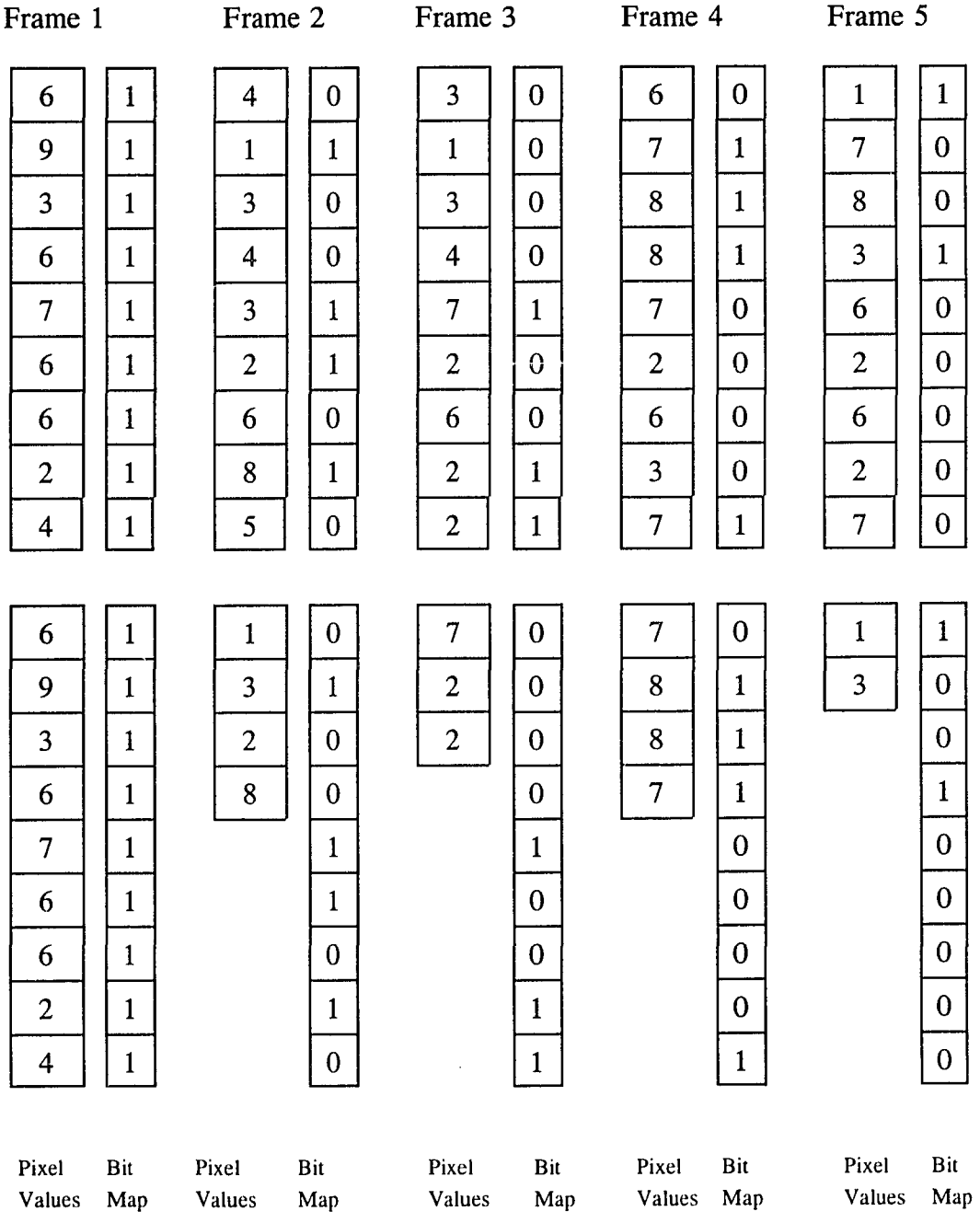



Figure 2. Sample values for SDIC

The following shows how to find the size of the bit-map table for each frame;

A 640x320 frame will contain

640 x 320 = 204,800 pixels

If we associate a bit for each pixel

204,800 pixels x 1 bits/pixel = 204,800 bits

Or 204,800 / 8 = 25,600 bytes ~ 26Kbytes.

Having to save this information along side each frame is unacceptable for most applications. Therefore, we have to find ways to eliminate or reduce significantly the overhead storage of the bit-map creates.

Four methods can be used to overcome this. We will explain each one of these methods in detail in 4.3. After careful evaluation of these methods we have selected to use Lempel-Ziv-Welch coding for the compression of bitmap tables as we have seen that this type of lossless compression gives the best results in reducing the size of the bitmap table. If we consider the fact that the bit map will only contain two characters, namely 1s and 0s with considerable number of repetitions, the bit map table can be compressed into a few kilobytes eliminating the overhead of storing the bit table with each frame.

## 4.1. Pseudo-code that shows the steps of the Scaled Differential Image Compression

Next30: Read a new frame. Save all the pixel values (Intraframe)

Bit Map Table is 1

Next: Read the next frame

Compare each pixel value of the new frame with the same pixels in the previous frame

Is the change in the value of any compared pixel within the given scale?

Set the flags of the pixels that are changed

Get the bit map for the whole frame

Compress the bit map by applying run-length coding

Save the compressed bit map for the frame

Save the values of the changed pixels

Are 30 frames completed?

If NO jump "next"

If YES jump "next30"

## 4.2. The advantages of this method

Compressing only each individual frame to be displayed in a Motion Video sequence requires unacceptably large storage space and long transmission time. Therefore further compression by eliminating the redundancies between consecutive frames is essential to achieve effective compression and transmission of motion video. But this additional requirement should not slow down the compression speed and still allow for real-time compression of motion video.

Reaching compression ratios beyond the compression of single frames can be done by removing the redundancies between each frame using Interframe Compression methods.

Our method offers a simple, straight forward and a fast means of compressing motion video by comparing each pixel of two consecutive frames and storing or transmitting only the pixels that have changed its value beyond a given range. This method does not require any complex mathematical computations and also can be performed in real-time through software. Only the bytes that change its value sets the flag bit associated with each byte and only those bytes are saved along with the bit map for each frame which contains the address information of the bytes that are saved for decompression.

## 4.3. Methods to reduce the size of the bit-map:

1) Limiting the size of the bit vector, assuming that the number of changed pixels between two frames can not exceed a certain number.

   This might result in high distortion on image quality when and if the images

contain fast motion or frequent changes. A general algorithm must be fit to handle even the worst cases with some acceptable image degradation. Limiting the size of the bit vector clearly will take away the generality of our algorithm.

2) We may define a least-fit rectangle that defines the area in which changes take place. But in cases where motion is spread over a wide area within the picture, the bit-map becomes just as large as a bit-map for the complete frame, eliminating all the compression gains and introducing a new set of complexity for decompression of the compressed frames.

> "For instance, take a sequence of images 1024 x 768 bytes in size which have four bouncing balls in each image. The balls are "all over the place" but fairly small in size (maybe 16x16). The sequence is 32 frames long and then it repeats. Because the balls are often widely spaced (at opposite corners of the image), the least fit rectangle is, on average, 90 percent of the full image size. This would mean the bitmap for each image would be about 88K, or a total of 2.8 Mbytes for all 32 frames ! While the actual image, would still be 2K per frame, or 64K for all 32 frames. Also, those 88K bitmaps have to be scanned for positive bits each time a frame is displayed. For most computers this is an overwhelming task to accomplish for real-time animation." [6]

3) Considering the fact that the bitmap obtained is nothing but a Sparse Binary Matrix we may use some of the compression methods developed for the compression of Sparse Matrices.

Statement of the problem:

When two consecutive frames are compared, only the bytes that are different are saved. In addition to this, we need to save a bit-map table that is to be used during

decompression of that specific frame as a guide to display the saved pixels in the correct locations. This bit-map is a binary sparse matrix, therefore how can we make use of the sparse matrix compression methods to be able to reduce the size of the bit-maps to a minimum?

Example : Scale of Change = $\pm$ 1

First Frame :

| | | | |
|---|---|---|---|
| 3 | 5 | 4 | 3 |
| 4 | 3 | 9 | 4 |
| 1 | 2 | 6 | 4 |
| 4 | 5 | 2 | 8 |

Next Frame :

| | | | |
|---|---|---|---|
| 3 | 5 | 6 | 4 |
| 2 | 2 | 9 | 5 |
| 1 | 2 | 6 | 4 |
| 7 | 2 | 2 | 8 |

Bit-Map :

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |

Pixel Value Vector (V) that holds pixels that are different then the previous:

| | | | | |
|---|---|---|---|---|
| 6 | 2 | 5 | 7 | 2 |

STATEMENT 1 :

For any two frames of size N x M, it is sufficient to store only the bit-map matrix and the pixel value vector to represent the complete frame in reference to the previous one.

PROOF 1 :

We can print the complete Intraframe on the screen by using the N x M values saved as a whole. Next, we scan the Bit-Map from left-to-right and top-to-bottom; each time we read a "1" at column N-k and row M-j of the Bit-Map table we read the first entry from the Pixel Value Vector and print a pixel that corresponds to that value on the identical coordinate N-k,M-j on the screen. Next time we encounter a "1" in the Bit-Map table we read the second value from the Pixel Value Vector and print it on the identical coordinate of the "1" in the Bit-Map Table. If we repeat this process for all N x M bits in the Bit-Map Table we can display the complete consecutive images.

STATEMENT 2 :

Total storage space required to save any two consecutive frames of size N x M, is equal to

$$N \times M + Size(V()) + Size(BitMap())$$

where;

M x N is the size of the Intraframe(first frame);

the V() is the length of the Pixel Value Vector(for the second frame);

Size(V()) is the number of elements in the vector;

BitMap() is equal to N x M bits (one bit for each row and column);

Size(BitMap()) is equal to N x M divided by 8 (result is represented in bytes).

PROOF 2 :

Statement 1 shows that an Intraframe, a Bit-Map Table and a Pixel Value Vector is sufficient to save two consecutive images in full. Therefore, the storage space required will be equal to the size of the Intraframe and the sizes of the Bit-Map Table and Pixel Value Vector consecutively.

STATEMENT 3 :

The ratio of the uncompressed frames to the compressed ones can only be 1 to 1 if and only if all the pixel values are different in the consecutive frame from the first one. Otherwise, the ratio will always be less then one and certain compression will be achieved.

PROOF 3 :

If we save these two frames without compression the total storage space will be equal to;

$$N \times M \times 2$$

After compression the total space becomes;

$$N \times M + Size(V()) + Size(BitMap())$$

$Size(V()) + Size(BitMap()) = N \times M$ only if all pixels are different

$Size(V()) + Size(BitMap()) < N \times M$ for all others

Therefore total saving will be:

$$\frac{N \times M + Size(V()) + Size(BitMap())}{N \times M + N \times M}$$

### 4.3.1. Sparse Matrix Concepts

Matrices having only a small percentage of nonzero elements are said to be sparse. Sparse matrix techniques have been developed and successfully applied to structural analyses, network theory, power distribution systems, linear programming, nuclear reactors, genetic theory etc. The techniques have been in a state of development and refinement for almost three decades; hence, they have reached a state of high performance matched only by the most sophisticated computer algorithms.

Many algorithms dealing with sparse matrices have one common denominator - only the nonzero elements of the matrix are stored. The goal is to operate on these matrices as though the entire matrix were present, but to save storage space, and in particular to reduce the access and execution time, because the zero entries need not be represented and manipulated. One of the major concerns in the investigation was the selection of an appropriate indexing method. Minimization of the storage space and the access time were the main considerations for the selection. Many different indexing techniques have been investigated, but three schemes were actually considered; 1) Bit-Map Scheme (BMS), 2) Single Indexing Scheme (SIS), and 3) Double Indexing Scheme (DIS).

### 4.3.2. Bit-Map Scheme (BMS)

Any matrix E, that has M rows and N columns can be represented by a binary matrix BE with corresponding rows and columns but with ones replacing the nonzero elements. In addition, a V vector, containing the values of the nonzero elements is needed. The elements of V can be ordered as they appear when the rows are scanned from 1 to M, or when the columns are scanned from 1 to N. The binary matrix BE can be stored in

a bit-addressing scheme therefore only one bit is required to store each element of the matrix (or in other words one byte stores 8 elements of the matrix BE).

The savings by processing variables in this manner are substantial. To store, for example, a 2048-D vector would normally require 4096 bytes (two bytes per word) as opposed to 256 bytes if they are stored as bit strings.

Concatenating the rows of the BE matrix, one can store the information in a bit string BS. Denoting the number of bytes required, to store the BS string, $S_B$ , its value can be calculated by

$$S_B = \Gamma [ (MxN) / B ]$$

where B is the number of bits per byte and the symbol $\Gamma$ represents the ceiling; i.e., rounding up to the nearest integer. If a row index vector IE is used to indicate the address of the first nonzero element in each row (counting only the nonzero elements), a more rapid access to any element of the row is achieved.

Denoting the IE(i) the value of element i, in vector IE, where i=1,2,3 ... ,M; by T(i) the number of nonzero elements in row i, and by NZ the total number of nonzero elements of the matrix, one can write

$$T(i) = IE(i+1) - IE(i) \ , \quad NZ = \sum_{i=1}^{M} T(i) \qquad (1)$$

The density (DY) of the sparse matrix is defined as the ratio between the total number of nonzero elements and the total number of elements contained in the matrix.

$$DY = NZ / (M \times N) \qquad 100 \times DY = \text{Density in percents}$$

Denoting by G the number of bits required for addressing any element in the IE vector; p the number of bytes required for indexing IE is $S_I$, where $S_I = \Gamma (M \times G) / B$. The total number of bytes for indexing the BE matrix is given by $S_T$, where

$$S_T = S_B + S_I = \Gamma [(M \times N) / B] + \Gamma [(M \times G) / B]$$

One should recall, however, that for nonbinary matrices there is a V vector that contains the nonzero elements, each of which requires one word (we assume each word to be a byte size for our purposes as each pixel value can be an integer between 0 to 256). In this particular case the V vector requires $S_V$ bytes of storage, thus $S_V = (NZ \times 8) / B$.

Assume that 512x512 sparse nonbinary matrix with 5% density is to be stored. The storage required to store the full matrix is 262,144 bytes. Since NZ=13,107; G=16 bits will be sufficient for indexing any element of the IE vector. Actually, 16 bits are sufficient for indexing any matrix that contains below 65,536 nonzero elements ($2^{16}$). The storage required to store BE is:

$S_T = S_B + S_I = 32768 + 1024 = 33,792$ bytes

$S_V = 13,107$ bytes

$S_T + S_V = 46,899$ bytes

Therefore, the storage space required to store the bit-map BE, the row vector IE and the value vector V is only 18 % of the full matrix.

### 4.3.3. Single Indexing Scheme (SIS)

In this scheme, as opposed to the BMS, only the non-zero elements are manipulated. For the sake of clarity, the following 4 x 5 matrices named A and B are given:

$$A = \begin{bmatrix} 0 & 3 & 0 & 0 & 2 \\ 5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 6 & 0 & 4 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad (2)$$

A linear mapping function f(i,j) = k can be defined. This function will enable one to refer to a specific location of an element (i,j) by specifying a value k. For an MxN matrix

$$f(i,j) = k = j + (i-1) \times N$$

For element (4,3) of A

$$k = 3 + (4-1) \times 5 = 18$$

Any matrix in SIS can be completely defined by specifying the number of rows M; number of columns N; number of nonzero elements NZ; and two vectors; L and V. The

L vector indicates the k values (the location) of the nonzero elements, and the V vector indicates the values of the corresponding elements. Obviously, for representing a binary matrix, the V vector is not needed. The A and B matrices can be represented by:

| | |
|---|---|
| LA = [2,5,6,9,13,16,18] | LB = [3,5,7,9,11,15,17,19] |
| VA = [3,2,5,1,7,6,4] | VB = Not needed |
| $M_A = 4$ | $M_B = 4$ |
| $N_A = 5$ | $N_B = 5$ |
| $N_Z = 7$ | $N_Z = 8$ |

To store the A matrix in full form, by using a byte per element requires 20 bytes, while in this form it requires 17 bytes. Not much of a saving, but the M matrix is not very sparse, DY = 35%. If a 10 x 10 matrix has the same number of nonzero elements, the density DY = 7%. To store such a matrix in full form requires 100 bytes, but is SIS form, 17 bytes. It is quite clear that to store the B matrix in full form requires 20 bytes, and in SIS form, it requires 11 bytes. The density DY = 40% is very large. One should realize, however, that as long as M x N < 65,536, any element of LB can be indexed by G = 16 bits. Actually, NZ is implied by the number of elements in LB and does not need to be stated explicitly. Under these conditions, to store the B matrix in SIS form requires 24 bytes and in BMS form requires 11 bytes.

$$
\begin{aligned}
ST = SB + SI \quad &= \Gamma\,[(M \times N) / 8] + \Gamma\,[(M \times G) / 8] \\
&= \Gamma\,(20 / 8) + (64 / 8) \\
&= 11 \text{ bytes}
\end{aligned}
$$

If the same number of nonzero elements NZ=8 are contained in a 40 x 10 matrix; i.e., DY=2%, the storage requirements for SIS and BMS are 24 bytes and 130 bytes, respectively. A clear indication that for sparser matrices the SIS form requires less storage.

Recovery: Given the value k of an element in the L vector, what are the corresponding (i,j) indexes? In other words, what is the inverse of the f(i,j)=k function?

$$f(i,j) = k = j + (i-1) \times N$$

Solving the above equation for i, one obtains

$$i = 1 + k/N - j/N$$

There are a number of constraints to be considered: i and j must be integers, $0<j/N\leq 1$ and $0<k/N\leq M$. From the equation and the constraints above, it is clear that $j = N$ yields $i = k/N$ and it must be an integer. Thus, k must be in this case an integer equal to N or a multiple of N, which leads to the conclusion that for $m<k/N\leq m+1$ where $m = 0,1,2 \ldots ,M-1$

$$i = \Gamma\, k/N = m+1$$

Solving for j, one gets

$$j = k - (i-1) \times N = k - m \times N = k \text{ MOD } N$$

Thus, the row value i is determined by taking the ceiling of k/N, and the column value j is determined by the module division k MOD N. For k = 18 and N = 5

$$i = \Gamma\ 18/5 = 4 \qquad j = 18 < MOD\ 5 = 3 \qquad (i,j) = (4,3)$$

During the investigation, it was determined that the SIS is quite efficient for storing the binary property matrix in a compact form as an L vector. The recovery of the information from the compact vector L is also quite efficient when performing the division and modulo division.

### 4.3.4. Double Indexing Scheme (DIS)

The scheme relies on two vectors for indexing the nonzero elements of binary matrices, and a third vector is necessary for representing the values of the elements contained in nonbinary matrices. This scheme is sometimes referred to as the Row-column indexing scheme or Block Indexing Scheme.

To represent any MxN binary matrix named B in the DIS form, two vectors, IB and JB are needed. For a nonbinary matrix, a third vector, VB, that contains the values of all nonzero elements has to be added. The 4 x 3 B matrix and its associated IB and JB vectors, represented below, are used to illustrate the concept.

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \begin{matrix} 1\ 2\ 3\ 4\ 5 \\ IB = [\ 1,3,4,5,7\ ] \end{matrix} \qquad \begin{matrix} 1\ 2\ 3\ 4\ 5\ 6\ 7 \\ JB = [1,3,3,1,2,3,\Delta\ ] \end{matrix} \tag{3}$$

The IB vector contains M+1 elements that reflect the count number until the first nonzero element in each row (counting rowwise the nonzero elements). Actually, as shown above, they are pointers that indicate the position of the corresponding elements in the JB vector. Only IB(M+1), the last element of IB, points to a symbol that indicates the END of JB. The small indexes represent the position of the elements in the vector -- for the IB vector they indicate the corresponding row number, except for the last one that indicates the position of the pointer to END of JB. The JB vector contains a number of elements equal to the number of nonzero elements in the B matrix. Each element represents the column position of the corresponding nonzero element. Therefore,

$$IB(i+1) - IB(i) = T(i)$$

T(i) represents the number of nonzero elements in row i, and also the number of elements in vector JB that are associated with the $i^{th}$ pointer. One counts T(i) elements from the $i^{th}$ element, inclusively, to its right, to determine the column indexes of all nonzero elements of the $i^{th}$ row. If IB(i+1) - IB(i) = 0, it indicates that the $i^{th}$ row

contains only zero elements.

Ordering the elements by rows as in the example above makes it unnecessary to maintain the row index for every nonzero element; only the row need be identified for the first nonzero element of each row, all the following elements up to the next pointer belong to the same row.

Storage Requirements: Denoting by G, as before the number of bits that are required to address an index, and by $S_I$ the total number of bytes needed to store the IB vector, $S_I$ can be calculated by

$$SI = \Gamma [ G \times (M + 1) / B ]$$

where B is the number of bits per byte. If the number of words is needed, B can be thought of as the number of bits per word. In a similar manner, the number of bytes that is required to store the JB vector, denoted by $S_J$, can be calculated by

$$S_J = \Gamma [ (M \times N \times G \times DY) / B ]$$

where DY represents the density of the matrix. The total number f bytes that is required to store the two vectors is denoted by $S_T = S_I + S_J$. If the matrix is not binary, the V vector requires $S_V$ bytes

$$S_V = \Gamma [ (M \times N \times W \times DY) / B ]$$

where W represents the number of bits per word.

4) If we study the structure of the bit-map obtained after the compression of each frame, we can see that in addition to being a 512x512 binary sparse matrix it is also is a 512x512 string of multiple consecutive 0's and 1's which is very suitable for size reduction by using any one of the compression methods for text.

Statement of the Problem:

The bit-map created after the comparison of each frame during compression consists of consecutive 0's and 1's. Therefore, a lossless compression method can be most suitable to reduce the size of this bit-map without losing its contents. This compressed file which will be attached to the pixel file for each frame, can later be decompressed and used to obtain the original image.

We have investigated various compression algorithms to achieve this job with the highest performance. Currently, there are three major compression methods: Huffman, Arithmetic and Lempel-Ziv [7],[8],[9],[10]. Among these three, Lempel-Ziv-Welsh which is a variation of Lempel-Ziv is the most popular one because of its speed, space requirement and performance.

### 4.3.5. Huffman Coding

Definition: An information source is a source alphabet together with a probability distribution; i.e., a set $\{a_1, \ldots\ldots, a_n\}$ together with numbers $P(a_1), \ldots\ldots, P(a_n)$ satisfying

$$\sum_{i=1}^{n} P(A_i) = 1 \quad , \quad 0 \leq P(A_i) \leq 1 \tag{4}$$

Our information source is discrete and zero memory as we have discrete source symbols and the source symbols appear independently. The probabilities $P(a_i)$ fully describe the statistics of the source message. Therefore,

$$P(a_{i1}, a_{i2}, \ldots, a_{ir}) = P(a_{i1})P(a_{i2}) \ldots\ldots P(a_{ir})$$

#### 4.3.5.1. Huffman Codes

Let K be a coding of an information source. That is, for each source symbol $a_i$, we have a code word $K(a_i)$ and we know the probability $P(a_i)$ of $a_i$. Denoting by $d_i$ the length of the word $K(a_i)$, we can compute the average length L of code words:

$$L = \sum_{i=1}^{n} d_i P(a_i) \tag{5}$$

#### 4.3.5.2. Construction of Binary Huffman Codes

A source of two symbols has an obvious Huffman code with the code words 0 and 1 [and $L_{min}(S) = 1$]. A source with three symbols, $a_1$, $a_2$, $a_3$, of which $a_1$ is the most probable, can be reduced to the case of two symbols, $a_1$ and $a_{2,3}$, where $P(a_{2,3}) = P(a_2) + P(a_3)$. We find a Huffman code for the reduced source

| $a_1$ | $a_{2,3}$ |
|---|---|
| 0 | 1 |

and then we "split" the code word 1 into two words, 10 and 11, thus obtaining a Huffman code for the original source:

| $a_1$ | $a_2$ | $a_3$ |
|---|---|---|
| 0 | 10 | 11 |

In general, let S be an information source with symbols $a_1, a_2, \ldots, a_n$ ordered by probability, i.e.,

$$P(a_1) \geq P(a_2) \geq \ldots\ldots \geq P(a_n).$$

The reduced source $S^*$ has symbols $a_1, a_2, \ldots, a_{n-2}$ and a new symbol $a_{n-1,n}$ ,and its probabilities are $P(a_1), \ldots, P(a_{n-2})$, whereas $P(a_{n-1,n}) = P(a_{n-1}) + P(a_n)$. Suppose that we are able to find a Huffman code $K^*$ for the reduced source $S^*$. Then the following code

| $a_1$ | $a_2$ | $\ldots\ldots$ | $a_{n-2}$ | $a_{n-1}$ | $a_n$ |
|---|---|---|---|---|---|
| $K^*(a_1)$ | $K^*(a_2)$ | $\ldots\ldots$ | $K^*(a_{n-2})$ | $K^*(a_{n-1,n})0$ | $K^*(a_{n-1,n})1$ |

[obtained by "splitting" the last code word $K^*(a_{n-1,n})$ into two words] is a Huffman code for S.

If we can not find a Huffman code for $S^*$, we continue in the same way, finding a

reduction of S* and a reduction of this reduction, etc. Eventually, we end up with two source symbols.

We observe that the average length L(K) of the code in (5) is related to the average length $L(K^*)$ of the original code by

$$L(K) = L(K^*) + P(a_{n-1}) + P(a_n)$$

In fact, if the lengths of code words of $K^*$ are $d_1, d_2, \ldots, d_{n-2}, d^*$, then the lengths of code words of K are $d_1, d_2, \ldots, d_{n-2}, d^*+1$. Thus,

$$\begin{aligned} L(K) &= \sum_{i=1}^{n-2} d_i P(a_i) + (d^*+1)P(a_{n-1}) + (d^*+1)P(a_n) \\ &= \sum_{i=1}^{n-2} d_i P(a_i) + d^*[P(a_{n-1}) + P(a_n)] + P(a_{n-1}) + P(a_n) \\ &= L(K^*) + P(a_{n-1}) + P(a_n) \end{aligned} \qquad \textbf{(6)}$$

### 4.3.6. Arithmetic Coding

The original concept of Arithmetic coding is first proposed by P. Elias, which is reported in Abramson[11] and improved by Langton[12] and Jones[13]. This method also assumes a prior knowledge of the probabilities for each input symbol and assigns a point in the unit interval for each based on their probabilities. The following explains how the

algorithm works.

The encoder at each time n will look at an input symbol $x_n$ and then, given past actions, determine a subinterval $I_n = [a_n + b_n)$. It may or may not then output code symbols, depending on what $I_n$ is. Beginning at time n = 0, if the first input symbol $x_0$ =0, set $I_0 = [0,q)$. If the first input symbol is a 1, set $I_0 = [q,1)$. Thus the time 0 subinterval has length equal to the probability of the symbol seen. Note that if we know the first subinterval, we also know the first input symbol $x_0$. We then divide the selected subinterval $I_0$ into two further subintervals proportional to the input probabilities: $[a_0,a_0 + q(b_0 - a_0))$ (having length $q(b_0-a_0)$) and $[a_0 + q(b_0 - a_0)), b_0))$ (having length $(1-q)(b_0-a_0)$). If $x_1 = 0$, then $I_1 =$ is the first subinterval. Otherwise it is the second.

Knowing the second subinterval tells us the second symbol $x_1$. In addition, it specifies the first subinterval since it is a subset of the first subinterval. Thus it specifies also the first input symbol. This procedure is then continued, each time dividing the previous subinterval into two subintervals proportional to the input probabilities. In general the new end points are given in terms of the previous endpoints by

$$a_n = \begin{cases} a_{N-1} & \text{if } x_n = 0 \\ a_{N-1} + q(b_{N-1} - a_{N-1}) & \text{if } x_n = 1 \end{cases}$$

$$b_n = \begin{cases} a_{N-1} + q(b_{N-1} - a_{N-1}) & \text{if } x_n = 0 \\ b_{N-1} & \text{if } x_n = 1 \end{cases} \qquad (7)$$

The algorithm produces at time n a subinterval of length equal to the probability of the input sequence produced up to that time. Knowing the subinterval $I_n$ is sufficient to completely determine the original input sequence up to the $n^{th}$ symbol, i.e. $x_0, \ldots, x_{n-1}$.

The sequence of subintervals $I_n$ is itself used to produce the code symbol sequence as follows. For each n, the subinterval endpoints $a_n$ and $b_n$ of $I_n$ both have binary expansions. At time 0 check to see if the first term in the binary expansions of $a_0$ and $b_0$ agree. This will be the case if either $I_0$ ( [0,1/2) or $I_0$ ( [1/2,1). If this is the case, the encoder produces the common symbol in the binary expansion, $u_0 = 0$ if $I_0$ ( [0,1/2) and $u_0 = 1$ if $I_0$ ( [1/2,1). If further binary symbols agree, then these too are released.

If the first symbols in the binary expansions of the interval endpoints do not agree, then no encoder symbol is output and the same test is conducted on $I_1$. The encoder repeats the test for $I_n$ for increasing n until an n is found for which the first symbol of the binary expansions of $a_n$ and $b_n$ agree, at which point the common binary symbol is released to the channel. The encoder then tests to see if the second symbols in the binary expansion agree. If yes, the second symbol is output to the channel. If no, the next subinterval is tested and so on until an n is achieved for which the first two binary symbols in the binary expansion of the endpoints of $a_n$ and $b_n$ of $I_n$ agree.

### 4.3.7. Ziv-Lempel Coding

This method is named after its inventors [9],[10]. The codes shares the property of the arithmetic codes that variable numbers of input symbols are required to produce each code symbol. Unlike both Huffman and arithmetic codes, however, the codes does not

use any knowledge of the probability distribution of the input.

We use a variation of the Ziv-Lempel algorithm suggested by Welch [14], who corrected an error in the original algorithm.

The basic idea is that an input sequence is recursively parsed into nonoverlapping blocks of variable size while constructing a dictionary of blocks seen thus far. The dictionary is initialized with the available single symbols, here 0 and 1. Each successive block in the parsing is chosen to be the largest (longest) word $w$ that has appeared in the dictionary and hence the word wa formed by concatenating $w$ and the following symbol is not in the dictionary. Before continuing the parsing $wa$ is added to the dictionary and a becomes the first symbol in the next block. Before detailing the parsing and dictionary construction, we show how the parsing alone works. Suppose that we see the sequence

01100110010110000100110

Parsing this sequence using the above rule yields the following segmentation, where the parsed blocks are enclosed in parentheses and the new dictionary word (the parsed block followed by the first symbol of the next block) written as a subscript:

$(0)_{01}(1)_{11}(1)_{10}(0)_{00}(01)_{011}(10)_{100}(01)_{010}(011)_{0110}(00)_{000}(00)_{001}(100)_{1001}(11)_{110}(0)$

The parsed data implies a code by indexing the dictionary words in the order in which they were added and sending the indices to the decoder.

The code is noiseless (lossless) and can be shown to be optimum in the limit of unbounded table size. The disadvantage of the algorithm is that unmanageably large tables may be required in some applications in order to achieve the desired performance. Any real application must necessarily have a bound on the table size. Once reached, the encoder can no longer add code words and must simply use the existing dictionary. This dictionary can be used in a static fashion to encode the remaining inputs, or it can dynamically adapt to track varying input behavior.

# 5. VIDEO COMPRESSION

Digitally encoded video is now a reality and slowly making its way into many computer and telecommunications applications. Just as audio compact discs have revolutionized the recording industry, so this new technology promises to unleash a whole slew of video applications- among them, the digital laser disk, electronic camera, video phone, and videoconferencing systems, interactive video tools on personal computers and workstations and high definition TV (HDTV).

Unlike the digital audio technology of the '80s, however, many of the applications of digital video hinge on the use of data compression. The audio bandwidth, after all, is about 20kHz, which translates into a digital data rate of about 1.4 Megabits per second for high-quality stereo sound. Sampled video source signals, on the other hand, require much higher bit rates, ranging from 10Mbits/sec for broadcast quality video to more then 100Mbits/sec for HDTV signals.

Even when still pictures are involved, as in image archival systems, a mountain of data is needed to represent them. For example, a color image with resolution of 1000 by 1000 picture elements (pixels) at 24 bits each will occupy 3 megabytes of storage in an uncompressed form. This will not fit onto a high-density floppy diskette, which can just hold 1.44Megs.

## 5.1. How does it work?

Compression methods are build on both redundancies and nonlinearities in the data and

the nonlinearities of human vision. They exploit correlation in space for still images and in both space and time for video signals. Compression in space is known as intra-frame compression, while compression in time is called inter-frame compression. Generally, methods that achieve high compression ratios( 10:1 to 100:1) are lossy in that the reconstructed data are not identical to the original.

The lossy algorithms also generally exploit aspects of the human visual system, For instance, the eye is much more receptive to fine detail in the luminance (or brightness) signal than in the chrominance(or color) signals. That's also why, the luminance signal is usually sampled at a higher spatial resolution. For example, in broadcast quality television, the digital resolution of the sampled luminance signal is 720 by 480 pixels, while for color signals it may be only 360 by 240 pixels. Second, the encoded (compressed) representation of the luminance signal is assigned more bits, a higher dynamic range, than are the chrominance signals.

Also, the eye is less sensitive to energy with high spatial frequency than with low spatial frequency. If an image on a 14-inch personal computer monitor were formed by an alternating spacial signal of black and white, the human viewer would see a uniform gray instead of the alternating checkerboard pattern. This deficiency is exploited by coding the high-frequency coefficients with fewer bits and the low ones with more bits.

## 5.2. Lossy Video Compression Algorithms

Compression of complex video material at bit-rates around 1 Mbps is a challenging task. Efforts are now underway by the motion picture experts group (MPEG) to standardize

a video-coding algorithm for interactive applications with digital storage media. The bit-rate for the compressed video, which is limited by the medium speed, is currently set by MPEG at 1.15Mbps.

The MPEG algorithm is based heavily on the px64 reference model developed for the videoconferencing application by a CCITT subgroup this model is referred to as the CCITT-RM [15]. The CCITT-RM is essentially a motion-compensated interframe predictive coding technique.

Efficient interframe coding requires motion estimation and compensation. This topic is discussed in depth in [16]; here we will discuss various algorithms used by MPEG and compare these algorithms with SDIC.

## 5.3. Discussion of existing video compression algorithms and their comparison to SDIC

Figure 3 shows the main components of the basic MPEG algorithm. This design is shown to give the optimum results for the requirements of the proposed MPEG-Video standard. The quality requirements of this standard demands a high compression ratio which can not be achieved by the compression of individual frames alone, on the other hand, random, access, reverse playback, fast forward/reverse searches are best satisfied by the use of intraframe compression only. The algorithm satisfies both of these requirements with a delicate balance between the inter- and intra frame compression.

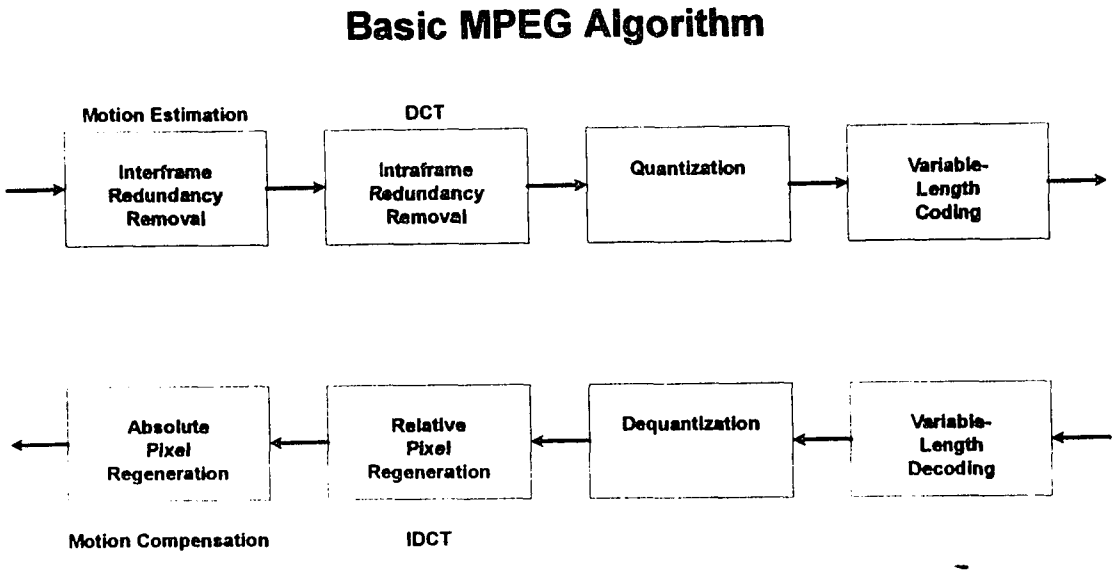



Figure 3. Basic MPEG Algorithm

The MPEG algorithm breaks a video sequence into a series of short segments, which are processed independently. Each segment, which typically contains fifteen frames, is known as a group of frames (GOF). A GOF contains three types of frames; intraframe, predicted and bidirectionally predicted (also known as interpolated frames).

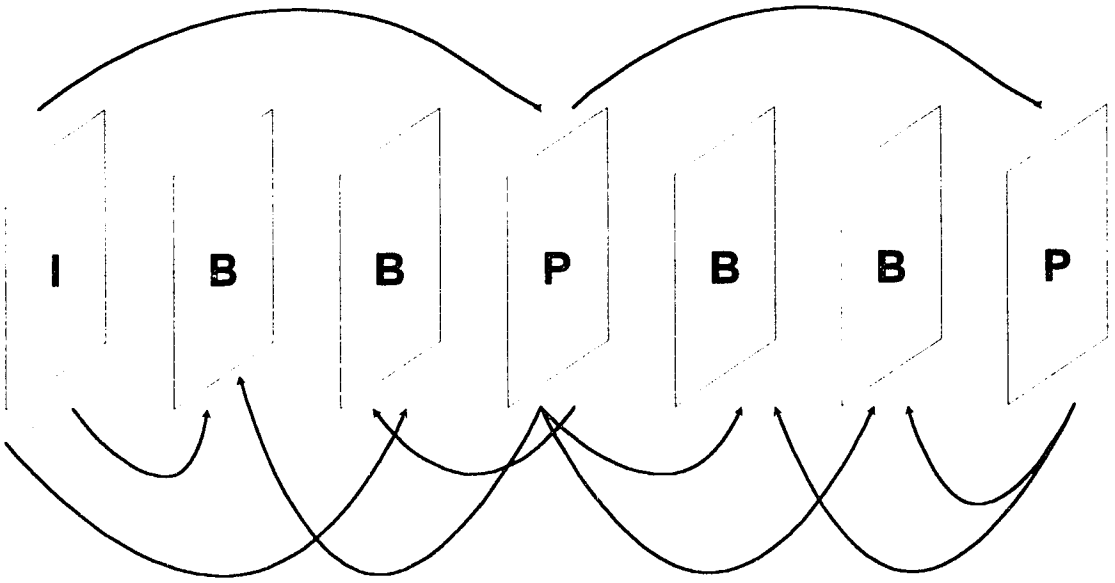
The first frame in a GOF is coded as an Intraframe (I), which is independent of other frames. This allows random access to any part of the video sequence as the display of the video can start by displaying the Intraframe followed by the predicted frames based on that Intraframe.

# MPEG - 1

- Coding at 1.5 Mbits/s for Digital Storage Media
- Committee Draft ISO 11172: Nov. 1991
- Draft International Standard: March 1992
- Hardware Demonstrated
- "VCR" Quality
- Noninterlaced Resolution
- Temporal Structure: I-P-B

| | H.261 | MPEG-1 |
|---|---|---|
| **Application** | **Communication** | **Storage/Retrieval** |
| **Bit Rate** | 64 Kbps - 1.92 Mbps | 1 - 1.5 Mbps |
| **Coding Algorithm** | Interframe Prediction<br>DCT,<br>Motion Compensation | H.261 + Interpolation |
| **Resolution** | QCIF Mandatory<br>CIF Optional | Up to 4,000 pixels<br>352 x 240, 352 x 288<br>Nominal |
| **Frame Rate** | 30,15,10, or min 7.5 | 25,30 Nominal |

Table 5. Comparison of H.261 to MPEG-1




**I - INTRAFRAME**
**P - PREDICTED**
**B - BIDIRECTIONAL INTERPOLATION**

Figure 4. MPEG-1 Frame Coding

This scheme solves the problem of random access, but there are still a number of problems that has to be addressed for Fast Forward and Fast Reverse. This problem is addressed in a paper by Kamikura and Watanabe[17] by the introduction of a hierarchical scheme in which the I-frames are coded into two layers. In the encoder, original DCT coefficient C is first roughly quantized by quantizer $Q_1$. Then, difference D between original coefficient C and quantized coefficient C' is finely quantized by quantizer $Q_2$. D' denotes the quantized difference. In the decoder, a roughly reconstructed image is decoded from only C' with a shorter display period per frame, thus the movement is natural.

The luminance component of the I frame is divided into 16x16 blocks and chrominance component into 8x8 blocks. The chrominance and luminance blocks are known as Macroblocks (MB). These macroblocks are then transformed using a two dimensional DCT.

Even though the coding of the Intraframes is conceptually and computationaly the simplest type of coding operation performed in the system, it turns out that the quality of the intraframes critically effects the overall quality of the reconstructed video.

MPEG specifies the use of an intraframe every 15 frames. This 15 frame group is short enough for spacial distortion(reduction in image quality) in the intraframe to persist through the group. As intraframes are coded independently the distortion created in even neighboring intraframes can be quite different. When the entire video is played back, a flicker appears every 15 frames. This so called "2-Hz flicker" is annoying. One way to

avoid this is to code intraframes as close to the original frame as possible, which of course leads to reduction in the compression ratio of the video sequence.

In a 1990 paper published in SPIE, Visual Communications and Image Processing by Puri and Arovind[18] this flicker problem is addressed and a possible solution is proposed. Realizing the fact that using different quantization parameters for the compression of an intraframe is one of the reasons for the distortion in image quality, quantization parameter in an intraframe is kept constant. Selected intraframes are encoded throughout the sequence to obtain quantization values suitable to use in all intraframes, and intraframes are quantized based on this optimum value. This of course results in a two pass approach which has a negative effect on the compression speed.

In our compression algorithm, Scaled Differential Image Compression, this problem does not exist due to the nature of the compression algorithm itself. SDIC uses lossless compression algorithm for the compression of the intraframes as they play a crucial role in the reconstruction of the following group of frames. Each interframe is displayed in reference to the previous one by redisplaying the pixel values that are different than the previous frame. In order to achieve a high video quality it is important that intraframes are kept exactly as the original. This approach naturally eliminates the 2-Hz flicker problem as the uncompressed intraframes are not subject to distortion.

Following the intraframe, every $M^{th}$ frame is coded in predictive (P) mode (typically M=3). The first P-frame(predictive frame) is predicted using the I-frame(intraframe), and subsequent P-frames predicted using the previous P-frame. Each MB in a P-frame

can be coded in one of several modes: predictively, predictively with motion compensation or, if prediction is not accurate enough, in intraframe mode.

Between P-frames, M-1 frames are coded using bidirectional predictive mode (B). Coding modes allowed for B-frames include motion compensation using prediction from either the previous P-frame (forward prediction) or the next P-frame (backward prediction), motion compensated interpolation using a weighted average of the previous and next P-frames (interpolative), and intraframe mode.

Efficient interframe coding requires motion estimation and compensation. Both pel recursive algorithms (PRA) and block matching algorithms have been developed. The former relates to motion estimation of individual pels, whereas the latter deals with the motion of a block of pels. PRA is computationally intensive and seldom used in practice.

In BMA, the block of pels of size (MxN) is compared with a corresponding block within the same search area of size (M+2p x N+2p) in the previous frame (Figure 5) and the best match is found based on cost function such as minimum MSE, minimum MAE, or maximum cross correlation,[19,20] defined as follows:

Cross-correlation function(CCF)

$$M_1(i,j)=\frac{\sum_{m=1}^{M}\sum_{n=1}^{N}U(m,n)U_R(m+i,n+j)}{\left[\sum_{m=1}^{M}\sum_{n=1}^{N}U^2(m,n)\right]^{1/2}\left[\sum_{m=1}^{M}\sum_{n=1}^{N}U_R^2(m+i,n+j)\right]^{1/2}} \quad \textbf{(8)}$$

$-p <= i, j <= p$

Normalized MSE(NMSE)

$$M_2(i,j)=\frac{\sum_{m=1}^{M}\sum_{n=1}^{N}[U(m,n)-U_R(m+i,n+j)]^2}{\sum_{m=1}^{M}\sum_{n=1}^{N}U^2(m,n)} \quad \textbf{(9)}$$

$-p <= i,j <= p$

Mean absolute error (MAE)

$$M_3(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}|U(m,n)-U_R(m+i,n+j)| \quad \textbf{(10)}$$

$-p <= i,j <= p$

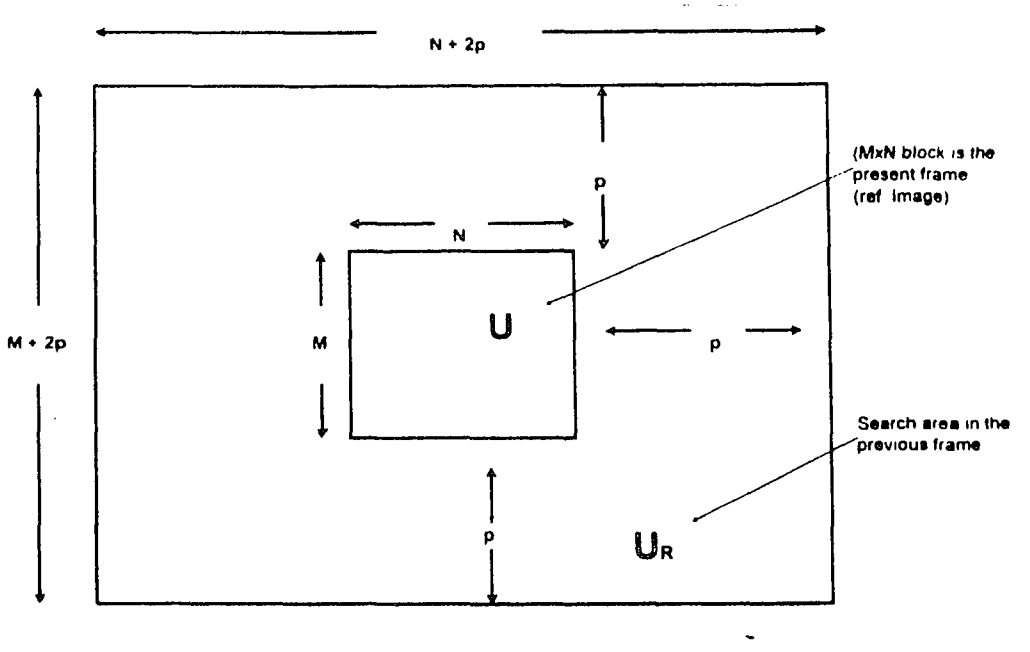



Figure 5. Geometry for manipulation in MxN reference image U with (M+2p) x (N+2p) image $U_r$(search area) in the previous frame.

There are a number of fast algorithms developed for Block Matching for the estimation of motion on a block by block basis. In his paper titled "The cross-search algorithm for motion estimation", Ghanbari offers a fast block matching algorithm.

To locate the best match by full search, $(2w + 1)^2$ evaluations of the matching criterion are required. To reduce the computational complexity, Jain and Jain [21] use a two-dimensional logarithmic search method (TDL) to track the direction of a minimum mean squared error distortion measure.

Koga *et al* [22] use a three-step motion vector direction search (TSS) to compute displacements up to 6 pels/frame. In Kappagantula and Rao's [19] modified motion

estimation algorithm (MMEA), prior to halving the step sizes, two more positions are also searched. Puri *et al* [23] have introduced the orthogonal search algorithm (OSA) where with a logarithmic step size, at each iteration 4 new locations are searched.

| Algorithm | Maximum number of search points | w | | |
|---|---|---|---|---|
| | | 4 | 8 | 16 |
| TDL | $2+7\log_2 w$ | 16 | 23 | 30 |
| TSS | $1+8\log_2 w$ | 17 | 25 | 33 |
| MMEA | $1+6\log_2 w$ | 13 | 19 | 25 |
| OSA | $1+4\log_2 w$ | 9 | 13 | 17 |
| CSA | $5+4\log_2 w$ | 13 | 17 | 21 |

Table 6. Comparison of the performance of Block Matching Algorithms[24]

There are also other methods in addition to the above mentioned methods for the estimation of motion between P-frames. Tu and Cleymans in their paper titled "Low bit-rate image coding based on evaluation of motion sequence"[25] presents a new strategy for low-bit rate image sequence transmission. On the contrary to the conventional motion estimation methods in which only the immediate former frame is referenced for obtaining the motion patterns, more than one previous frames are used in the motion evaluation procedure to estimate the temporary motion behavior.

In addition to various interframe coding methods mentioned above there is another problem that has to be resolved for a more efficient prediction. New coding methods are required for the efficient coding of the uncovered background areas which appear behind

moving objects. In a paper titled "Uncovered background prediction in interframe coding"[26], Mukawa and Kurada proposes an uncovered background predictive coding algorithm which increases prediction efficiency. According to Mukawa and Kurada a video scene can be divided into three kinds of areas. The first area represents the still object area or background area, the second indicates the moving object or foreground area, and the last represents the background area which appears behind the moving object, namely the uncovered background area.

A different coding method should be used for each one of these areas to achieve highest efficiency. For the still object area, an interframe coding method should be employed, for moving object area motion-compensated interframe coding is efficient. For the coding of the uncovered background the following background generation algorithm is proposed.

$$\hat{x}_b = \hat{x}_b^{-f} + (1/m)\, sign\,(\vec{x}-\hat{x}_b^{-f})\, u(\vec{x}-\hat{x}^{-f}) \tag{11}$$

where

$$\left.\begin{aligned} u(\vec{x}-\vec{x}^{-f}) &= 1 \quad for\ |\vec{x}-\vec{x}^{-f}| \le L \\ u(\vec{x}-\vec{x}^{-f}) &= 0 \quad for\ |\vec{x}-\vec{x}^{-f}| > L \end{aligned}\right\} \tag{12}$$

and

$\hat{x}_b$ generated background prediction signal at the present frame

$\hat{x}_b^{-f}$ generated background prediction signal at the previous frame

$x$ local decoded signal at the present frame (input signal available)

$x^{-f}$ local decoded signal at the previous frame (input signal at the previous frame available)

u still area detection function

m a parameter which controls background signal alteration

L a threshold

These equations denote the conditions:

1) when frame difference ($x - x^{-f}$) on a pel is large, the background prediction signal keeps its previous value by assuming the pel is in the moving areas, and

2) when the frame difference is small, the background prediction signal gradually approaches the original signal, namely, the local decoded signal, by assuming the pel is in the still areas.

The third step in MPEG coding is the quantization of the DCT coefficients. Quantization combined with the run-length coding contributes to the overall compression efficiency of MPEG.

Intra- and interframes are quantized differently. Intraframe blocks contain high energy in all frequencies and are likely to produce "blocking effects" if too coarsely quantized;

on the other hand predictive frame blocks mostly contain high frequency blocks can be subject to much coarser quantization.

Three forms of quantization can be used for this purpose.

1) Zero-memory quantization

2) Block quantization

3) Sequential quantization

Following quantization, quantized values are coded using a lossless redundancy reduction method. Most MPEG applications use Huffman coding with probability distributions embedded in the Huffman code itself, with more probable codes associated with shorter codewords. Dynamic Huffman coding is avoided to reduce the complexity of the overall algorithm even though certain improvement could have achieved by its use.

In a paper titled "Encoding of motion video sequences for the MPEG environment using arithmetic coding" [27], Viscito and Gonzales offers the use of arithmetic coding to achieve better results. In their solution they take advantage of the fact that arithmetic coding can be made adaptive and perform the probability estimation in parallel with the coding. This eliminates the need to estimate the probability distributions ahead of time by collecting data on the statistics of either the sequence to be encoded or a representative source and adaptive approach offers more precise probability distribution values thus resulting in shorter codewords.

## 5.4. Spacial redundancy reduction and the use of transform coding

Both still image and prediction signals have a very high spacial redundancy. The redundancy reduction techniques usable to this effect are many, but because of the block based nature of the motion compensation process, block based techniques are preferred. Transform coding involves linear transformations in which the signal space is mapped into a transform space where the transformed samples are then compressed for transmission or storage. The reconstruction operation involves an inverse transformation of the transformed samples.

The sequence involves two operations on the source side of the channel: transformation and quantization; and two operations on the receiver side of the channel: decoding and inverse transformation. Each of these operations requires some amount of memory, and this need for memory-particularly at the source end of the system-is involved in one of the important system complexity considerations for compression by transform coding.

The choice of the transformation (and the inverse transformation as well) is often dictated by practical implementation consideration, even though it is possible to design an optimum transformation, the principle component or Karhunen-Loeve, transformation.

### 5.4.1. Principle Component Transform

The principle component transform has been considered as an optimum transformation, and for this reason many other transformations have been compared to it for performance. Hotelling(1993) was the first to derive and publish this transformation under the name "principle component". This is a discrete transform.

This is how principle component transforms an image of picture elements(pixels):

For an image with L lines and L pixels per line, let $f(x,y_j)$ represent all the L pixels in the $j^{th}$ line, where j = 1,2,....L.

Then

$$[f(z)] \equiv [f(x,y_1),f(x,y_2), \ldots, f(x,y_L)] \tag{13}$$

is the $L^2$ vector composed of all the pixels taken in the normal raster pattern sequence, in other words, the whole image.

Then we define as $L^2$ x $L^2$ matrix [A] such that the transformed pixels are defined as:

$$[F(w)] = [A][f(z)] \tag{14}$$

Each component of [F(w)] can be expressed as a linear combination of all the original pixels:

$$F(w_k) = \sum_{i=1}^{L^2} f(z_i)A_{ij} \qquad k=1,2,\ldots,L^2 \tag{15}$$

The original pixels can be reconstructed from the transformed pixels by means of the inverse transformation

$$[f(z)] = [A]^t[F(w)] \tag{16}$$

and similarly, each original pixel can be expressed as a linear combination of all the coefficients, $F(w_k)$;

$$f(z_i) = \sum_{k=1}^{L^2} F(w_k)A_{ki}, \qquad i=1,2,.....,L^2 \tag{17}$$

The ideal would be for transformation, [A], to produce independent coefficients, $F(w_k)$. However, the closest to this is with a transformation that produces uncorrelated coefficients. Such a transformation is a matrix whose columns are the normalized eigenvectors of the covariance matrix of the original pixels.

The covariance matrix is defined as

$$C_f = E\{([f(z)] - E\{[f(z)]\})([f(z)] - E\{[f(z)]\})'\} \tag{18}$$

Writing this expression out for $C_f$ with the simplifying assumption that $E\{[f(z)]\}=0$ and the simplifying notation $f(z_i)=f_i$.

$$C_f = \begin{bmatrix} E(f_1^2) & E(f_1f_2) & E(f_1f_3) & ..... & E(f_1f_{L^2}) \\ E(f_2f_1) & E(f_2^2) & E(f_2f_3) & ..... & E(f_2f_{L^2}) \\ E(f_3f_1) & E(f_3f_2) & E(f_3^2) & ..... & E(f_3f_{L^2}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ E(f_{L^2}f_1) & E(f_{L^2}f_2) & E(f_{L^2}f_3) & ..... & E(f_{L^2}^2) \end{bmatrix} \tag{19}$$

Now the expected values of the various squares and cross-products in $C_f$ are typically obtained by averaging products of pixels with the same spatial separation and relative location from a larger image of which the $L^2$ dimensional vector [f(z)] is an L x L subimage.

The eigenvalues of $C_f$ are the solutions $\phi$ to the matrix equation:

$$C_f\phi = \lambda\phi \tag{20}$$

where $\lambda$ represents the eigenvalues. If we solve for eigenvalues from the characteristic equation:

$$\det[C_f - \lambda I] = 0 \tag{21}$$

where I is the unit matrix. We then arrange the $\lambda$'s in decreasing order such that $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_L^2$ and substitute into $(Cf - \lambda I)\phi = 0$ to solve for the eigenvectors. When the matrix [A] (whose columns are the $\phi$ solutions) is applied to [f(z)], the covariance of the resulting coefficients $F(w_k)$ is a diagonal matrix with diagonal elements $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_L^2$ . Thus, the coefficients, $F(w_k)$, are uncorelated by definition.

The principal component transform is said to be an optimal transform because it has the following properties:

1) It completely decorrelates the signal in the transform domain.
2) It minimizes the MSE in bandwidth reduction or data compression.
3) It contains the most variance(energy) in the fewest number of transform coefficients.
4) It minimizes the total representation entropy of the sequence.

Practical implementations of principal component transform involves the estimation of the auto-covariance matrix of the data sequence, its diagonalization, and the construction of the basis vectors. The inability to determine the basis vectors in the transform domain

has made principal component transform an ideal but impractical tool and also the fact that there is no "fast" principal component transformation makes it almost impossible to implement in practical applications. This transform therefore is mostly used as a benchmark against which other discrete transforms may be judged.

### 5.4.2. The Fourier Cosine Transform

Given a function x(t) for $-\infty < t < \infty$, its Fourier Transform is given by

$$X(w) \equiv F[x((t)] = \left(\frac{1}{2\Pi}\right)^{\frac{1}{2}} \int_{-\infty}^{\infty} x(t)e^{-jwt}\,dt \qquad (22)$$

subject to the existence conditions of the integral. Here, $j=\sqrt{-1}$, $w=2\pi f$ is the radian frequency and f is the frequency in Hertz. The function x(t) can be recovered by the inverse Fourier transform, i.e.

$$x(t) = F^{-1}[X(w)] = \left(\frac{1}{2\Pi}\right)^{\frac{1}{2}} \int_{-\infty}^{\infty} X(w)\,e^{jwt}\,dw \qquad (23)$$

F[.] and $F^{-1}[.]$ denote forward and inverse Fourier transforms of the functions enclosed. If x(t) is defined only for $t \geq 0$, we can construct a function y(t) given by:

$$y(t) = \begin{array}{ll} x(t) & t \geq 0, \\ x(-t) & t \leq 0 \end{array}$$

Then

$$F[y(t)] = \left(\frac{1}{2\Pi}\right)^{\frac{1}{2}} \left\{ \int_0^{\infty} x(t)\, e^{-jwt} dt + \int_{-\infty}^{0} x(-t)\, e^{-jwt} dt \right\}$$

$$= \left(\frac{1}{2\Pi}\right)^{\frac{1}{2}} \int_0^{\infty} x(t)[\, e^{-jwt} + e^{jwt}\,] dt \qquad \textbf{(25)}$$

$$= \left(\frac{2}{\Pi}\right)^{\frac{1}{2}} \int_0^{\infty} x(t)\cos(wt) dt$$

We can now define this as the Fourier cosine transform (FCT) of x(t) given by

$$X_c(w) \equiv F_c\,[x(t)] = \left(\frac{2}{\pi}\right)^{\frac{1}{2}} \int_0^{\infty} X_c(w)\cos(wt)\, dw \qquad (t \geq 0) \qquad \textbf{(26)}$$

Noting that $X_c(w)$ is an even function of w, we can apply the Fourier inversion to obtain

$$y(t) = x(t) \equiv F_c^{-1}\,[X_c(w)\,] = \left(\frac{2}{\pi}\right)^{\frac{1}{2}} \int_0^{\infty} x_c(w)\cos(wt)\, dw \qquad (t \geq 0) \qquad \textbf{(27)}$$

### 5.4.3. DCT via FFT

It is shown above that the Fourier transform of an even function results in the definition for the Fourier cosine transform. This simple property can be exploited for the computation of the DCT.

Let {x(n)}, n=0,1,2, .... ,N-1 be a given sequence. Then an extended sequence {y(n)} symmetric about the (2N-1)/2 point can be constructed so that

$$\begin{aligned} y(n) &= x(n) && n = 0,1,2, \ldots , N-1 \\ &= x(2N-n-1) && n = N, N+1, \ldots , 2N-1 \end{aligned}$$

If we use $W_{2N}$ to denote $\exp(-j2\pi/2N)$, it can be seen that the DFT of $\{y(n)\}$ given by

$$Y(m) = \sum_{n=0}^{2N-1} y(n)\, W_{2N}^{nm} \tag{28}$$

is easily reduced to

$$\begin{aligned} Y(m) &= \sum_{n=0}^{N-1} x(n)\, W_{2N}^{nm} + \sum_{n=N}^{2N-1} y(n)\, W_{2N}^{nm} \\ &= \sum_{n=0}^{N-1} x(n)\, W_{2N}^{nm} + \sum_{n=N}^{2N-1} x(2N-n-1)\, W_{2N}^{nm} \\ &= \sum_{n=0}^{N-1} x(n)\, W_{2N}^{nm} + \sum_{n=0}^{N-1} x(n)\, W_{2N}^{(2N-n-1)m} \\ &= \sum_{n=0}^{N-1} x(n)[\, W_{2N}^{nm} + W_{2N}^{-(n+1)m}], \qquad m = 0,1,2,....,2N-1 \end{aligned} \tag{29}$$

If we now multiply both sides of 29 by a factor of $1/2W_{2N}^{m/2}$, we directly obtain

$$\frac{1}{2} W_{2N}^{m/2}\, Y(m) = \sum_{n=0}^{N-1} x(n) \cos\left[(2n+1)\frac{m\pi}{2N}\right], \qquad m = 0,1,2,....,N-1 \tag{30}$$

### 5.4.4. Two dimensional DCT by reduction to one-dimensional DCT

Let an (M x N) matrix [g] represent a black-and-white (for simplicity) digital picture, where the matrix element $g_{mn}$ may be interpreted as the grey level or the intensity of the pixel (picture element) at the (m,n) location. [G], the two-dimensional orthogonal transform of the matrix [g], and its inverse are respectively, defined as

$$[G] = [T_M][g][T_N]^T$$

and

$$[g] = [T_M]^T[G][T_N],$$

where $[T_M]$ and $[T_N]$ are the (M x N) and (N x N) real transformation matrices. The above formula assumes that the two-dimensional transform can be implemented by a series of one-dimensional transforms.

Let [g] be an (M x N) matrix representing the two-dimensional data and [G] be the two-dimensional DCT of [g]. Then the uv-element of [G] is given by

$$G_{uv} = \frac{2c(u)c(v)}{\sqrt{(MN)}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{mn} \cos\left[\frac{(2m+1)u\pi}{2M}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right], \tag{31}$$

$$\textit{where} \quad u = 0,1,....,M-1\,,\ v = 0,1,....,N-1\,,\ \textit{and}$$

$$c(k) = \frac{1}{\sqrt{2}} \quad \textit{if } k = 0\,,$$

$$= 1 \quad \textit{otherwise}.$$

Similarly, the mn-element of [g] is given by the two-dimensional IDCT of [G], defined as

$$g_{mn} = \frac{2}{\sqrt{(MN)}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} c(u)\, c(v)\, G_{uv} \cos\left[\frac{(2m + 1)u\pi}{2M}\right] \cos\left[\frac{(2n + 1)v\pi}{2N}\right], \tag{32}$$

*where* $m = 0,....,M-1 \wedge n = 0,....,N-1$

### 5.4.5. Why is DCT used for the reduction of entropy rather than DFT?

Most data compression techniques that are used for the compression of digitized frames and video sequences use a technique called transform coding, which is a linear transformation in which signal space is mapped into a transform space. The reconstruction operation involves an inverse transformation of the compressed images. The speed and efficiency of the computation of the transform algorithms are directly related to the speed and efficiency of the proposed compression method. Therefore, choosing the right transform method is extremely important to increase the efficiency of the compression algorithm.

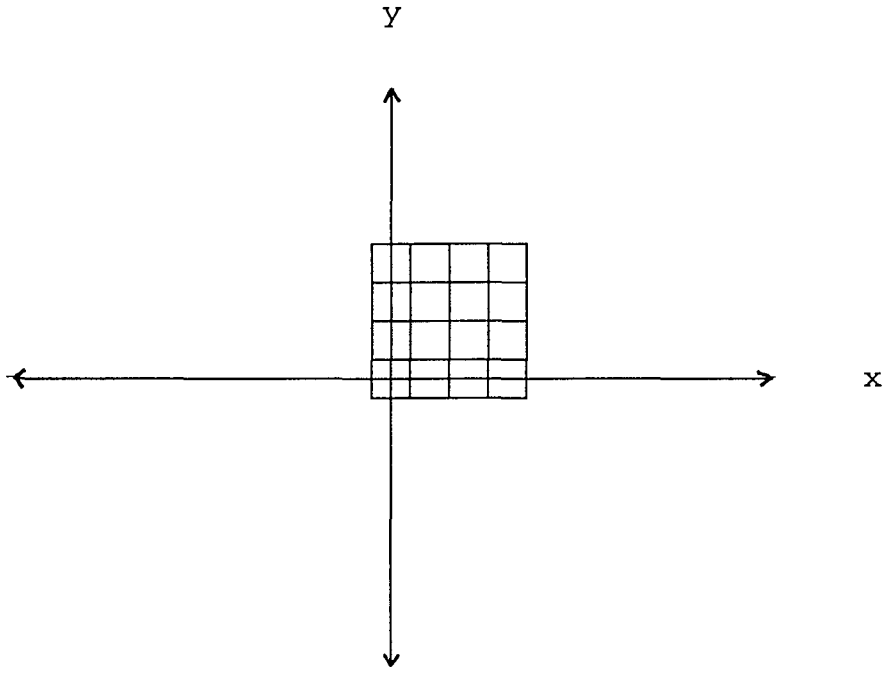
The fourier transform is probably the most popular transformation method but still the question remains to whether it is the most efficient transform for image compression or not?

Consider the following formula for the discrete fourier transform(DFT) where X(m) is a sequence of N finite-valued real or complex sample values, with m=0,1,2...,N-1.

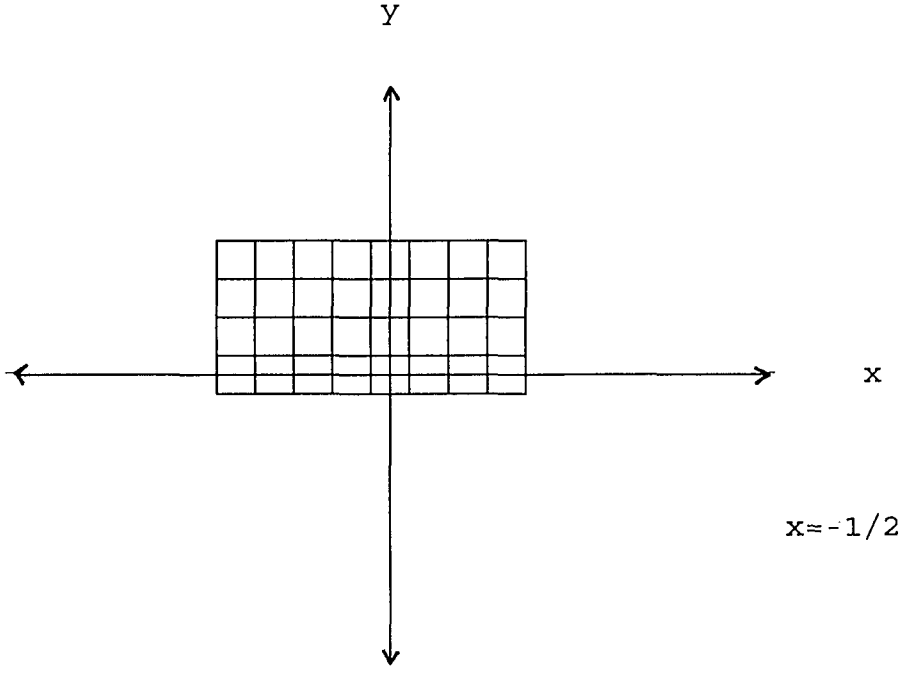
$$C_x(k) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)\, e^{-\frac{j2\Pi km}{N}} \qquad k=0,1,2,..,N-1 \tag{33}$$

If we apply this transform to a sample image segment we obtain the following formula:

Given an image,




We may flip this image along the x-axis as shown, find the mirror image.




x=-1/2 y=0

The DCT representation of the above image can be found as follows;

$$F(n)=\sum_{-\frac{N}{2}}^{\frac{N}{2}-1} f(k)*e^{-2\pi j n(k+\frac{1}{2})} \tag{34}$$

f(-k) = f(k-1)

f(k) = f(k)

We may rewrite F(n) as

$$F(n)=\sum_{-\frac{N}{2}}^{-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})}+\sum_{0}^{\frac{N}{2}-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})} \tag{35}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})}+\sum_{k=-\frac{N}{2}}^{-1} \hat{f}(k)*e^{-2\pi jn(k+\frac{1}{2})} \tag{36}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})}+\sum_{k=-\frac{N}{2}}^{-1} f(-k-1)*e^{-2\pi jn(k+\frac{1}{2})} \tag{37}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})}+\sum_{k=1}^{\frac{N}{2}} f(k-1)*e^{-2\pi jn(-k+\frac{1}{2})} \quad \textbf{(38)}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} f(k)*e^{-2\pi jn(k+\frac{1}{2})}+\sum_{k=0}^{\frac{n}{2}-1} f(k)*e^{-2\pi jn(-k-\frac{1}{2})} \quad \textbf{(39)}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} (f(k)*e^{-2\pi jn(k+\frac{1}{2})}+f(k)*e^{2\pi jn(k+\frac{1}{2})}) \quad \textbf{(40)}$$

$$NOTE\ :\ \cos\theta\ =\ \frac{e^{j\theta}+e^{-j\theta}}{2} \quad \textbf{(41)}$$

$$F(n)=\sum_{k=0}^{\frac{N}{2}-1} f(k)*\cos 2\pi n(k+\frac{1}{2}) \quad \textbf{(42)}$$

If we expand this function to the two-dimensional n x n DCT of the image with values for f(n,m);

$$F(n,m) = \sum_{k=0}^{N-1} \sum_{p=0}^{N-1} f(k,p) * \cos 2\pi n(k+\frac{1}{2}) * \cos 2\pi m(p+\frac{1}{2}) \qquad (43)$$

From this result we can see that the fourier transform of an even function results in the definition of the fourier cosine transform which leads to the computation of discrete cosine transform. Therefore, DCT with less complexity which transforms even functions is preferred to DFT which transforms functions with both the real and imaginary parts.

### 5.4.6. The algorithm of DCT

Source image samples are grouped into 8x8 blocks; each sample has a value in the range of $[-2^{p-1}, 2^{p-1}-1]$, where p can be 8 or 12. Each block is input to the Forward DCT(FDCT). The output of FDCT can be input to Inverse DCT(IDCT) to obtain the original blocks to reconstruct the original source image samples. The equations of 8x8 FDCT and 8x8 are as follows:

$$F(u,v) = \frac{1}{4} C(u)C(v) \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) * \cos \frac{(2x+1)u\Pi}{16} \cos \frac{(2y+1)v\Pi}{16} \right] \qquad (44)$$

$$f(x,y) = \frac{1}{4} C(u)C(v) \left[ \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v) * \cos \frac{(2x+1)u\Pi}{16} \cos \frac{(2y+1)v\Pi}{16} \right] \qquad (45)$$

where C(u), C(v) = $1/\sqrt{2}$ for u,v = 0; C(u), C(v) = 1 otherwise.

The original source samples are the functions of two spatial dimensions x and y. Each block has 8x8 = 64 samples. FDCT takes each block of 64 discrete signals in space domain, transforms into 64 discrete signals in frequency domain. By definition of FDCT and IDCT, the FDCT output of each block of samples will be 64 signals in the orthonormal basis of 64 vectors in the frequency domain. The output is also called "DCT coefficients" which are uniquely determined by the original 64 samples in the block. The coefficient with zero frequencies in both x and y directions is called the "DC coefficient", and the other 63 coefficients are called the "AC coefficients". Because image samples vary slowly from point to point most of the time, most of the high frequency coefficients in the FDCT output would be zero, great compression can be achieved as we only need to store and transmit nonzero coefficients. DCT introduces no loss in the encoding and decoding process, because it only transforms signals in spatial domain to that in frequency domain. But in actual calculation of transcendental functions, it is impossible to get perfect accuracy. Therefore, some losses will occur.

## 5.5. Other image coding methods

Willemin and Reed describes a method for reducing the information contained in an image sequence, while retaining the information necessary for the interpolation of the sequence by a human observer in their paper "Image Sequence Coding by Split Merge"[28]. The method consists of first locating the redundant information, reducing the degree of redundancy, and coding the result. In their experiments they have used four frames from the original Trevor sequence and in the coding of the sequence they used 5000 regions, composed of 17772 parallelpipeds and 10000 regions composed of 25454 parallelpipeds. This required 80 kb and 160 kb, respectively, for the coding of the

polynomial coefficients and for boundary information, 115 kb and 152 kb for a total of 195 kb and 312 kb for these two cases. The resulting transmission rates were 152 kb/s and 244 kb/s.

Test of their proposed algorithm have been performed using standard images, 256 x 256 pixels in size, transmitted at 25 images/sec and achieved a compression ratio of nearly 200. The quality of the restored images is suggested to be reasonably good. However, this high compression ratio is achieved with various side effects of the split and merge and reconstruction process. Another defect is imprecision of the borders of some objects, "some important borders sometimes disappear after merging neighboring regions" says the authors.

The proposed method have number of limitations, for example, because of memory limitations, the method has not yet been tested on image sequences longer that 32 frames and another major drawback of the algorithm is the fact that the entire image sequence has to be coded before it is transmitted. Real time operation is therefore not possible using such a technique. Use of this method for videoconferencing applications will require implementation using parallel architectures or specialized processors.

The compression ratio of the proposed method seems impressive but the fact that it can not operate in real time makes it impossible to apply this technique to real-life applications. Our method, Scaled Differential Image Compression do not seem to achieve such high compression ratios but with its fast compression/decompression speed it seems to be the best choice for videoconferencing and other applications that require real time

operation. The image quality and the NMSE of the compressed/decompressed images of the SDIC is also far superior. Due to the nature of its compression algorithm SDIC does not experience imprecision of borders of some objects and does not require filtering of the images to eliminate fuzzy edges.

In another set of papers published by Kodak, their new product called "Kodak SV9600 Still Video Transreceiver"[29] is explained. This is a transreceiver designed by Kodak engineers to operate over the standard telephone lines. The transreceiver captures a full frame of video, digitizes and stores it in memory for manipulation and display of the image data. The data is compressed by using a JPEG-like algorithm developed by Kodak.

For RGB color input, the image is first transformed into Y, R-Y, and B-Y using the following equation:

$$Y = .299R + .587G + .114B$$

This transformation packs most of the signal energy into the luminance component Y and is easily implementable. For a NTSC input image, the data is already in the similar YIQ format and, hence no transformation is required. Next, the image is partitioned into n x n blocks and each block is independently transformed using DCT. A block size of $n=16$ is used for this implementation. The DCT coefficients are reformatted into a one dimensional vector according to a zig-zag format which is equivalent to rearranging the coefficients in decreasing order of their expected energy content. The components that do not contain significant energy values are eliminated. The transform coefficients are normalized according to their visual importance and quantized uniformly to the nearest integer and the quantized coefficients are encoded using Huffman code tables.

The compression ratio vs. transmission time measurements of the implemented method are given as follows by Kodak.

If the image was stored as red, green, and blue (RGB) with 512 samples per line in all three colors, and the vertical resolution was 512 lines of video, the digital image would require 786,432 bytes for memory space. This assumes each sample has 8 bits per pixel which is equivalent to a bit rate of 24 bits per pixel. The stored image in the transreceiver only requires 393,216 bytes, but only 294,912 bytes are used since the color difference samples are subsampled and averaged vertically. The compression algorithm typically provides a compression ratio of 6:1, which gives a total compression ratio relative to the RGB image of 16:1.

From the above specifications it can be seen that Kodak's still video transreceiver achieves, on the average, almost similar compression ratios to Scaled Differential Image Compression. One point that has to be emphasized is the fact that Kodak's transreceiver does not take advantage of the temporal redundancies between the consecutive frames, as it is designed to transmit still images. One possible reason for this is the long transmission time it takes to send an image to the receiver site which is given as 41 seconds. This makes it impossible to use this system for transmission of motion video as the delay between consecutive images will be unacceptably long.

A solution to this may be to use faster transmission methods over the standard phone lines (Kodak's system uses 9600 bps modem) and to increase the compression ratio by using inter/intra frame compression methods.

## 5.6. Lossless Video Compression

Lossless methods do exist, but with very limited success. The compression ratios achieved by the current lossless methods [30,31,32,33] on gray scale images seem not to exceed 3:1 [34].

The JPEG *lossless* arithmetic coding algorithm and a predecessor algorithm called Sunset both employ adaptive arithmetic coding with the context model and parameter reduction approach [35]. A context model(model structure) [36,37] operates recursively on the selected data items of the data sequence, and employs conditioning states x based on the past history of the data seen. In classical information theory, conditioning states x are in 1:1 correspondence with probability distributions. In [36], context function G(x) selects the probability distribution for the next event by partitioning conditioning states x. Now each context(partition) z is in 1:1 correspondence with its probability distribution z.

In lossless predictive coding algorithms using a multiple-context model, parameter reduction[38] is one purpose for quantizing the error to form error buckets. In contrast, lossy algorithms quantize the error to reduce the amount of information, such that only the error bucket index is encoded.

The JPEG group is developing a still color image compression standard [39,40]. In [41] a proposal called EXACT which is made for an independent lossless gray-scale image compression algorithm. The independent JPEG lossless algorithm uses predictive coding. One version has Huffman coding as in [42]. The JPEG lossless version considered here

uses prediction with error bucketing for parameter reduction, does context selection based on bucketed prediction errors, encodes with an adaptive binary arithmetic coder called QM-Coder, and uses a binarization of the prediction error [43].

Table 7 below shows the compression ratios on different test pictures obtained by JPEG lossless compression using a QM-Decoder.

| Image | QM-Coder |
|---|---|
| Lenna(512x512x8) | 1.62:1 |
| Liver(512x512x8) | 1.76:1 |
| Sjband(512x512x8) | 2.77:1 |
| Jbaloon(576x720x8) | 2.24:1 |

Table 7. Compression ratios on different test pictures using lossless JPEG [35]

As seen from the Table above, the compression ratios for lossless compression methods are rather small and can not be used for video applications where the bandwidth easily approaches to multi-megabits and high compression ratios are a must for transmission of this large sums of information.

One other lossless compression method that is worthwhile mentioning here is the Lossless Interframe Compression of Medical Images [34]. Due to the nature of CT and MRI imaging technologies medical images are often organized as a sequence of two-dimensional cross-sections of a 3-dimensional object. This sequence is commonly represented as a 3-dimensional array of voxels

$g(x,y,z), 0 <= x < N_x, 0 <= y < N_y$, and

$0 <= z < N_z$ , where Nx,Ny,Nz define the image resolution, and

$0 <= g(x,y,z) < 2^m$ is an integer encoded my m bits.

The key to compression of the above data source will be the removal of correlations among $g(x,y,z)$ values. The best performed compression technique for intraframe medical image coding is a two step process: decorrelation by two-dimensional polynomial interpolation of pixel values and entropy coding of prediction errors. This scheme can be generalized from two-dimensions to 3-dimensions for compressing a sequence of medical images. That is, take every other voxel in each of x,y, and z directions, and then predict the voxel values in between by 3-dimensional polynomial interpolation of the known voxels. But due to uneven sampling rates in different directions in medical imaging practice (the interframe distance is usually much larger than the distance between adjacent samples in x-y plane, typically around 5mm versus around 0.7 mm), 3-dimensional interpolation of voxel values does not work well.

First, as in [32], gray level values of voxels are encoded by gray codes. The reason for this is that gray code guarantees that the binary representation of the codes of adjacent integer gray levels differ by only on bit. Let

$$g_{m-1} \cdots g_1 g_0$$

be the intensity value encoded by gray code. Then for a fixed j, $0 <= j < m$, the bits

$g_j$ of all voxels of image sequence constitute a 3-dimensional $N_x$ x $N_y$ x $N_z$ array of 0's and 1's. This array is called the $j^{th}$ significant bit volume $V_j$ of the image sequence. Consequently the whole image sequence consists of a 4-dimensional volume of bits in which $V_j$, $0 <= j < m$, are the layers of subvolumes. In the sequel, b(x,y,z,j) is denoted by the $j^{th}$ significant bit of the integer voxel value g(x,y,z). Due to the adjacency property of gray code, for larger j's (higher significant bits of gray code), $V_j$ will have large connected components of uniform bits, or large chunks of 0's and 1's, if the 3-dimensional intensity function g(x,y,z) is reasonably smooth. This is the case for typical medical images because organs usually consist of approximately uniform substance and their spatial occupancy is contiguous[34].

Such techniques are used only in sensitive applications such as medical images or images transmitted from space taken by spacecrafts. For example, artifacts introduced by a lossy algorithm into an X-ray radiograph may suggest an incorrect interpretation and alter the diagnosis of a medical condition. Conversely, for commercial, industrial and consumer applications, lossy algorithms are preferred because they save on storage and communication bandwidth.

## 5.7. Why is Scaled Differential Image Compression better?

Scaled Differential Image Compression (SDIC) is an ideal compression method for video images transmitted during a videoconference. A videoconference which is basically the interaction of a number of persons in a closed area with limited motion allows SDIC to achieve very high compression ratios while maintaining a real-time compression. We will show that using SDIC compression we can reduce the bandwidth of the video

transmission between sites to well under 500 Kbytes using full 30 frames per second.

This is an impressive performance. SDIC offers high compression ratios using an extremely fast algorithm with minimum complexity and ease of implementation.

Compression methods based on motion compensation and estimation seems to give acceptable results if they are run on powerful multi-processor machines[44] or expensive dedicated pieces of hardware. Methods offered as a possible means of compression for motion video are still highly compute intensive, mostly based on imprecise prediction and poor in quality vs. performance considerations. Even though, decompression in real-time can be achieved considerably easier, compression in real-time and transmission of those images still requires much improvement[45].

The method described in this paper does not use prediction but a straight-forward comparison of consecutive images and introduces a way to store only the pixels that change their values within a predefined scale. Software implementation is easy and fast, hardware implementation would even give faster compression / decompression times and better overall performance.

# 6. ISDN SERVICES AND VIDEOCONFERENCING

In recent years the world's telecommunications network have gone through a number of dramatic evolutionary changes that have been largely prompted by two basic factors.

The first is the expanding nature of the user requirements. Initial goals for the design and installation of the telecommunications networks were for the transmission of voice. Today, residential and business users have an increasing demand for the effective transmission of fax, data, graphics, audio and video.

The demand for these services is especially pronounced in the business community. Ever expanding local and international networks require effective and clear communication not only in audio form but also as motion-video so that meetings can take place without having to physically be in these locations. The same applies to the academic community which has a growing need for reaching out communities and students through distance learning which otherwise be impossible to offer a wide selection of courses and lectures. In addition to the transmission of audio and video, there is a great need for the transmission of digital data such as database records, CAD files, electronic mail, etc.

The second factor in the evolution of telecommunications is the technological changes and development in the microelectronics industry. The increasing digitization of telecommunications switching and transmission hardware allowed for the deployment of inexpensive, reliable, software driven, intelligent wide-band networks. It is in this environment of changing user needs, intelligent digital communication technologies, and

declining costs that many of the world's telecommunications networks began to deploy the Integrated Services Digital Network or ISDN.

Implementation of fully digital and intelligent ISDN networks allowed for the development of a wide array of services. ISDN is designed to offer worldwide end-to-end connections over digital circuits at bit rates and qualities far better than the currently available analog voice network. End-users have the ability to establish either point-to-point or point-to-multipoint links which can be configured at almost any bit rate, starting from 64 kbps up to giga bps rates. There are also new developing technologies that assign "bandwidth on demand" allowing a much more effective use of the available bandwidth. This approach enables users to take up only as much bandwidth as necessary for the current session, rather than always using a prefixed bandwidth which in most cases may be more than necessary.

ISDN is the medium for the transmission of digital motion video. The availability of dial-up ISDN lines offers a convenient way to establish videoconferencing over circuit-switched lines. Since combination of audio and video, which contains rapidly changing images, can only be supported by a high rate link, the cost of running a videoconferencing session for long periods over long distances has traditionally been quite expensive. Compression, of course, is the solution to reduce the number of bits that have to be transmitted thus requiring a smaller bandwith for the transmission of video images. This in turn translates into greater savings in the cost of transmission which will ultimately be the driving force for the acceptance and the wide use of this technology.

## 6.1. Time division multiplexing

In digital communication systems, multiple systems are transmitted over a channel by time division multiplexing, or TDM. Here bits or small clumps of bits from different sources are interleaved to form a faster bit stream. Some means of framing is needed to identify which bits belong to which original source so that signals can be separated at the receiving end.

The first level in the multiplex hierarchy is DS-1, 1544 kpbs, which is the signal carried by the very widely deployed T1 carrier system. In most common application, the signal carries 24 digitized voice channels. A voice channel is digitized by sampling 8000 times per second and representing each sample by a binary number of up to 8 bits, for an aggregate bit rate of 64 kbps.

The DS-1 signal consists of frames 193 bits in length, generated 8000 times per second, to produce the 1544 kbps total bit rate. Each frame contains a single framing bit, followed by 8 bits in sequence for each of the 24 constituent channels.

The basic 64 kbps channel is often referred to as the DS-0 level. It can readily be used for the transmission of digital information in place of digitized voice. Various transmission constraints and the possible need for signaling and administrative information frequently do not permit the free use of the entire 64 kbps in the DS-0 channel, so user information is restricted to a slightly lower rate, such as 56 kbps.

Four DS-1 bits stream are interleaved and framing and other overhead bits are added to

produce the 6.312 Mbps DS-2 level. Similarly, seven DS-2 signals plus overhead bits form the DS-3 level of 44.736 Mbps, and the 274.176 Mbps DS-4 level is formed from 6 DS-3 level bit streams.

## 6.2. ISDN Configuration

Central to any ISDN is the interexchange network(IEN), which consists of the physical and logical components of the backbone transmission network, including a number of network transit exchanges and the transmission trunks connecting these exchanges. As far as the end-user is concerned, the IEN's main purpose is to provide physical and logical transmission and switching facilities across which the user information flows may be conveyed. There are two types of IEN, those providing circuit switched connections (CSIEN) and those based on the store-and-forward principle, of which packet switching (PSIEN) is an example.

Superimposed on the IEN and interacting with it is the common channel signalling network (CCSN), which combines the functions required for the control, management, and maintenance of the ISDN. It provides the physical and logical transmission capacity for the transfer of connection control signals between the components of the IEN, for the management and allocation of network resources, and for the performance of maintenance functions.

The last major part of the ISDN is the subscriber-access network (SAN) which consists of the part of the ISDN between the end-user or subscriber and the IEN and CCSN. It can be divided in turn into three components, namely the customer-premises installation

(CPI), the digital section (DS), and the logical exchange termination (LET). The CPI combines those aspects of the SAN that are directly under the control of the subscriber. The DS consists of the local loop or subscriber-access line and the physical line termination equipment. It provides the transmission capacity for carrying information between the local exchange and the customer premises. The purpose of the LET is to terminate these transmissions in a logical sense.

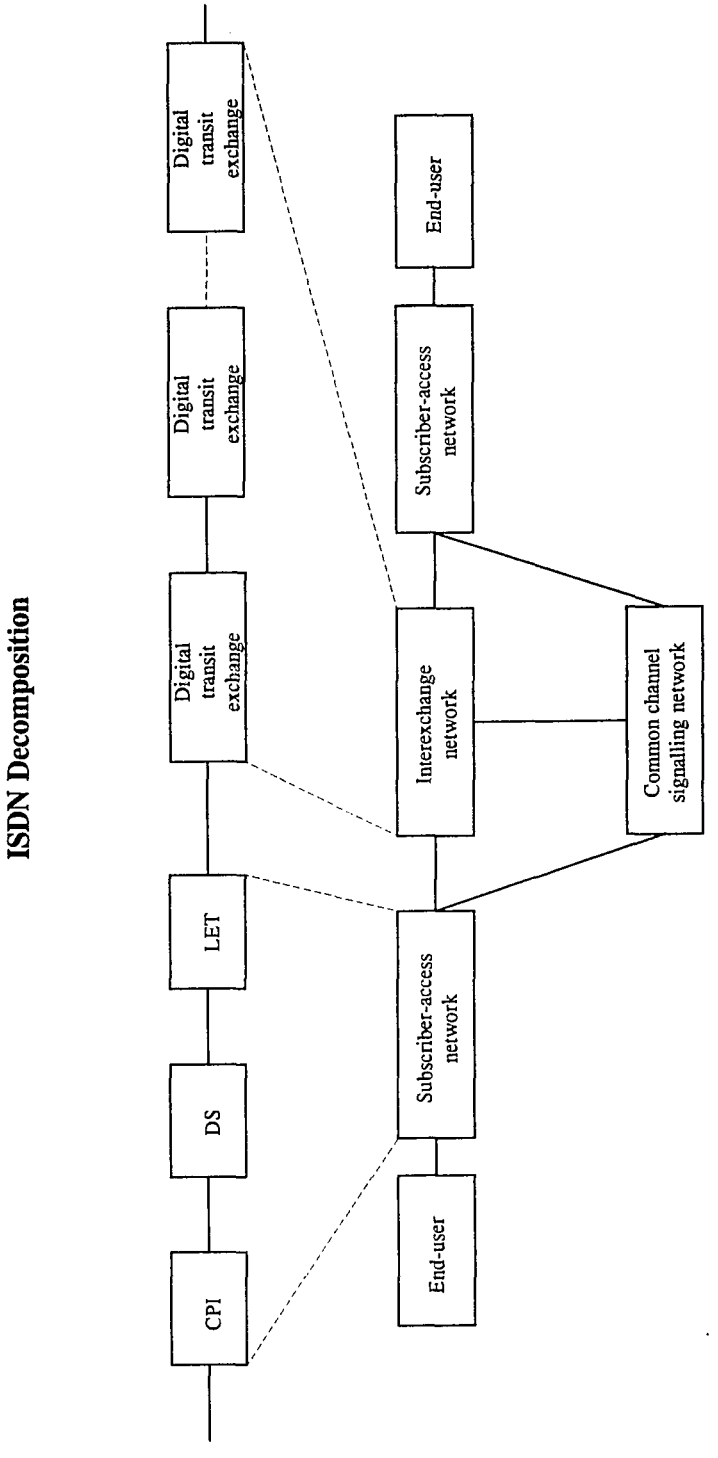



Figure 6. ISDN Decomposition

## 6.3. The ISDN Communications Architecture

We need to develop an abstract model that describes the overall structural aspects of the ISDN's communications capabilities. Such a communications architecture allows us to describe the characteristics of the services that an ISDN makes available to the end-users. One of the most important objectives of the ISDN development is to standardize these characteristics of the user-to-network interface in order to obtain several major benefits. First, as each component is well-defined and standardized, many suppliers can manufacture one or many parts of the ISDN components. Second, end-user devices can be developed independent of the network equipment, third end-user devices can be used from any access point without any compatibility problems.

Another important consideration in the development of ISDN is to offer a cost effective solution to each user's varying bandwidth requirements which sometimes can be as little as 64 kbit/s or as large as many megabits per second. It is clear that network resources should be allocated to users depending on their needs and requirements without greatly increasing the complexity of the ISDN architecture.

Therefore, in order to balance the requirements of efficiency, universality, flexibility, low complexity and low cost to wide variety of existing and evolving devices and applications, several distinct standard user-to-network interfaces should be defined. Three of these are known as basic-access, primary access and broadband access. The basic-access consists of two B channels and one 16 kbit/s D channel, for an aggregate rate of 144 kbit/s. The B channels may be used simultaneously or independently of each other.

Primary rate channel is designed for applications that require data rates in access of 144 kbit/s. Primary rate channel carries information upto 1544 kbit/s in any combination of B and/or H0 channels (H0 channel = 384 kbit/s). This channel is designed for fast facsimile machines, slow motion-video terminals and local area networks.

The broadband channel carries information upto 129.024 Mbit/s which is equivalent to 2016 B channels. This channel also may be used in various combinations of B, H0 and D channels. Originally envisioned to transport videoconferencing applications and compressed High Definition Television.

This clearly defined hierarchical channel structure offers savings in the cost of transmission by supporting multiple channels in increasing capacity and therefore offering the users the smallest data rate greater than their maximum need.

One factor that is drastically changing the ISDN services is the development of compression methods. Compression reduces the bandwidth required by applications by reducing the redundancies and/or entropy of the information transmitted. This results in two fold savings for the user. First, the use of compression allows the use of smaller channels which means that the cost and the complexity of the network equipment will be less and second the smaller the bandwidth required the less the cost of communication will become. Videoconferencing applications which were initially envisioned to use broadband ISDN services can use primary access or even basic access channels after incorporating compression methods such as our proposed Scaled Differential Image Compression.

# 7. PROGRAMS FOR COMPRESSION/DECOMPRESSION

We have considered a sequence of 30 consecutive frames. Each frame is digitized by a resolution of 512 by 512 pixels containing 64 shades of gray. The images are taken from the video database of the Speech and Hearing Department at the Graduate Center which shows a single individual talking in front of a camera from a close-up view. We considered this as an acceptable example of a usual video that may simulate a video conference session in which mostly the images of individuals sitting or standing in front of a plain background are transmitted. The details of how a videoconference is conducted and the environment in which the conference takes place is explained in the third chapter. Most videoconferences are conducted with very little motion which means that people that take part in it are always in a sitting position or standing up and mostly moving their arms to point to the presentation. The background is always plain and single color. Our images reflect a short segment of such a conference where a close up image of a participant is taken.

## 7.1. Display.pas

Next, we wrote a program using Turbo Pascal 6.0 to display these images on a PC. This program later evolved into a complete image viewer program (See appendix A). We have tested each image before any compression or manipulation to see if the image is digitized accurately and the pixel values obtained after digitization are acceptable. We decided that the best way to verify the accuracy of the digitized images was to actually display them on our monitor and see that the digitized frames are the same as the original video image and can be displayed back without any distortion.

"Display.pas" uses a VGA driver for 256 colors and can display up to 64 shades of grey values. It also uses the block read function of Turbo Pascal for fast access to the image file to be displayed. Images are read in chunks of 32Kbytes which greatly improves the speed of display for each image.

Procedure "initialize" initializes the VGA driver for 256 colors and sets the color palette for 64 greyscale color values. This is done by setting the RGB values that represents each pixel to a value that is equal for all three. R=G=B = 64 for Black, R=G=B = 0 for white and all other shades are spread between values 1 to 63.

After initialization and the setting of the image palette, image values are read in chunks of 32 kilobyte blocks and directly written into display memory again to improve the time it takes to display the image. One problem we had to overcome was the fact that the digitized images were digitized with a resolution of 512x512 pixels, which did not exactly match the resolution of our screen which was 640x480. In other words, we could display all the columns in the image but not all the rows. Our program takes care of this by skipping the remaining rows once it reaches the row 480. This operation is done only for display purposes, we do not skip the rows 480 to 512 for the compression and make sure to use all the rows that make the image.

We have also experimented with number of programs which displays images consecutively without any compression to verify that the digitized images represent the correct sequence of motion as expected.

## 7.2. Compress.pas

After verification of each frame, we wrote our main compression program that implements the Scaled Differential Image Compression algorithm. "Compress.pas" is the compression engine of our algorithm.

It performs the following algorithm:

* Save the first frame as a whole to constitute the Intraframe for the following sequence of frames.
* "Bit Map Table" is set to 1 for all bits, and only a single 1 is stored to represent the whole table.
* Read the next frame
* Compare the current frame with the previous frame, save the pixel values of only those that change their values.
* Build a "Bit Map Table" assigning a single bit for each pixel on the frame and set the bits to 0 if two pixel values are different beyond the "range of change".
* Apply "Run Length Coding" to the Bit Map Table for compression.
* Save the compressed Bit Map Table.
* Save the values of pixels that changed.
* Return to the beginning of the loop and repeat this for the next 29 frames.
* After compressing a total of 29 frames (30 with the Intraframe), go top and read the next frame as an Intraframe and repeat the above steps.

Compress.pas uses two input and two output files. Infile1 is the first frame and infile2 is the second. The program reads these two files consecutively and compares each pixel value with the corresponding one in the second frame. This process leads to two output

files. Outfile2 is the file that contains the values of the pixels that are different in the infile2 from infile1 and outfile1 contains the complete bitmap for the comparison of two frames.

The program starts by procedure "initialize' which opens the necessary input and output files, creates the filenames to be assigned to the compressed output files after computation and begins compression by reading the first two frames. It creates the difference and the bitmap files. This program also accesses disk one block at a time rather then a byte at a time, greatly improving the disk access time. The pixels that change within 10 % of the previous pixel value are accepted to be unchanged and recorded as the same value as the previous frame, thus causing a 10 % loss of precision versus the original image. This could have been reduced to 5 % or even to a lesser value but 10 % loss is seen to be acceptable as far as the quality of the images concerned. For further information about the variables used in this program, please see Appendix B. The program is well commented and self-explanatory.

### 7.3. Decompress.pas

This is the program that performs the decompression of the video sequence compressed by the compress.pas. It reads two files as input, one is the bitmap file which contains bits that represent which pixels are saved and which are not and their X-Y locations on the screen.

Decompress.pas performs the following algorithm:

* Initialize the screen and set the video mode
* Display the first frame, which is the intraframe saved without any compression.
* Read the bitmap for the next frame
* Check each bit in the bitmap

  If the bit is 0 then read a byte from the pixel file and display it, to the same position with the bit just read from the bitmap

  If the bit is 1 then it means leave the pixel as it is on the screen and skip to the next bit
* Do this until finishing all the bits in the bitmap
* Read the bitmap for the next frame and repeat the same process for 29 frames(first one is the intraframe)
* After 29 frames, read the next intraframe and repeat the process.

The decompress.pas starts by initializing the video memory and the video mode to 640x480x256. It then sets the color palette to 64 shades of grey by initializing the R,G,B values to values ranging from 0 to 63. next, it displays the first frame which is the intraframe saved without compression. This frame is displayed just as any other image file in a column by column fashion.

Decompression starts in the second procedure "decompress". It reads the bitmap for the next image to be displayed on to the intraframe that is already on the screen. The bitmap is read on block at a time and then divided into bytes and each byte is divided into bits.

Each bit represents a pixel, or in other words it represents whether the pixel is changed or not. If the bit is equal to 0 then a byte is read from the image file and a pixel representing that value is displayed on the screen. This operation is repeated until all the bits in the bitmap is read which signals the end of the frame.

This operation is repeated for the next 29 frames. At the end of frame 29, a new intraframe is read and displayed on the screen, followed by the display of the 29 compressed frames until the next intraframe.

# 8. RESULTS

The following shows the results we have obtained from the compression of our thirty digitized frames.

The frames used in our experiments were monochrome 512 x 512 pixel images. Each frame required

$$512 \times 512 \times 8 = 262,144 \text{ bytes}$$

of total storage space which totaled to 7,864,320 bytes for all 30 frames used in our experiment.

We have compressed these images using SDIC compression algorithm and the following results are obtained. It is important to notice that there are two sets of values for each frame compressed, one is the size of the bit-map and the other is the size of the pixel values. Final compression values are the sum of these two.

**AFTER COMPRESSION :**

The list of bitmaps obtained for each frame pair and their values after compression:

| Frame Numbers | Frame Size | Frame Numbers | Frame Size |
|---|---|---|---|
| **Bitmap 1** | 960 bytes | **Bitmap 16** | 7312 bytes |
| **Bitmap 2** | 6115 bytes | **Bitmap 17** | 1609 bytes |
| **Bitmap 3** | 1113 bytes | **Bitmap 18** | 7046 bytes |
| **Bitmap 4** | 1670 bytes | **Bitmap 19** | 849 bytes |
| **Bitmap 5** | 7149 bytes | **Bitmap 20** | 6428 bytes |
| **Bitmap 6** | 1727 bytes | **Bitmap 21** | 1049 bytes |
| **Bitmap 7** | 1056 bytes | **Bitmap 22** | 7308 bytes |
| **Bitmap 8** | 6162 bytes | **Bitmap 23** | 1696 bytes |
| **Bitmap 9** | 1737 bytes | **Bitmap 24** | 690 bytes |
| **Bitmap 10** | 6683 bytes | **Bitmap 25** | 880 bytes |
| **Bitmap 11** | 6586 bytes | **Bitmap 26** | 1760 bytes |
| **Bitmap 12** | 1849 bytes | **Bitmap 27** | 1475 bytes |
| **Bitmap 13** | 1889 bytes | **Bitmap 28** | 2297 bytes |
| **Bitmap 14** | 6477 bytes | **Bitmap 29** | 1088 bytes |
| **Bitmap 15** | 1952 bytes | | |
| **TOTAL SIZE OF BITMAPS AFTER COMPRESSION :** | | | 94,612 bytes |

Table 8. Total size of bitmaps after compression

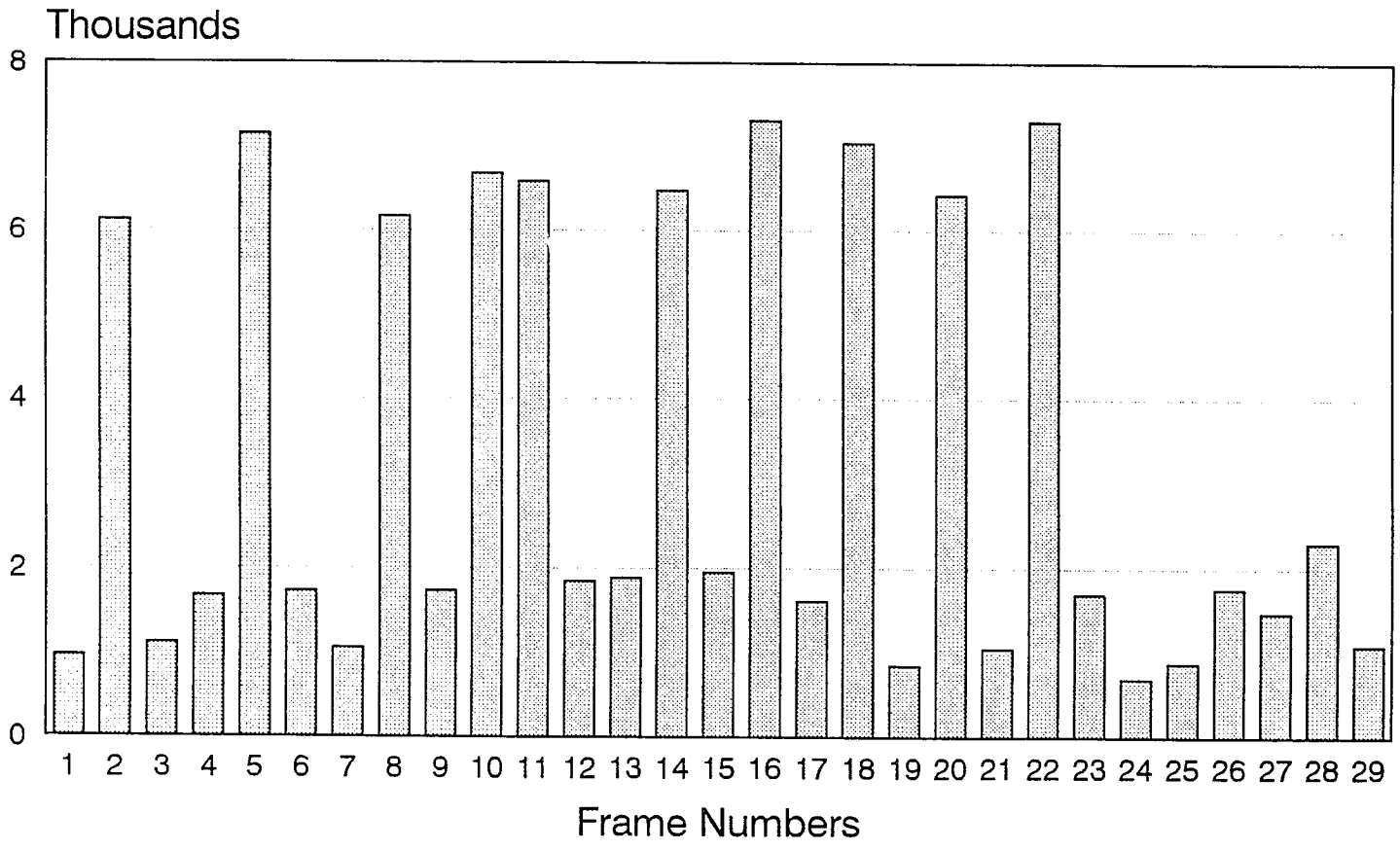



Figure 7. Bitmaps for each frame after compression

The list of pixel files which contains the value of those pixels that changed their value in the following frame beyond the defined scale :

| Frame Numbers | Frame Size | Frame Numbers | Frame Size |
|---|---|---|---|
| **Pixelfile 1** | 175143 bytes | **Pixelfile 16** | 8855 bytes |
| **Pixelfile 2** | 955 bytes | **Pixelfile 17** | 18602 bytes |
| **Pixelfile 3** | 10871 bytes | **Pixelfile 18** | 8207 bytes |
| **Pixelfile 4** | 1091 bytes | **Pixelfile 19** | 18050 bytes |
| **Pixelfile 5** | 8049 bytes | **Pixelfile 20** | 870 bytes |
| **Pixelfile 6** | 18658 bytes | **Pixelfile 21** | 11581 bytes |
| **Pixelfile 7** | 8357 bytes | **Pixelfile 22** | 1086 bytes |
| **Pixelfile 8** | 1096 bytes | **Pixelfile 23** | 18937 bytes |
| **Pixelfile 9** | 10980 bytes | **Pixelfile 24** | 8309 bytes |
| **Pixelfile 10** | 8330 bytes | **Pixelfile 25** | 559 bytes |
| **Pixelfile 11** | 11950 bytes | **Pixelfile 26** | 761 bytes |
| **Pixelfile 12** | 11817 bytes | **Pixelfile 27** | 8603 bytes |
| **Pixelfile 13** | 8523 bytes | **Pixelfile 28** | 2228 bytes |
| **Pixelfile 14** | 8913 bytes | **Pixelfile 29** | 9632 bytes |
| **Pixelfile 15** | 11649 bytes | **Pixelfile 30** | 1201 bytes |
| **TOTAL SIZE OF PIXELFILES AFTER COMPRESSION :** | | | 413,863 bytes |

Table 9. Total size of pixelfiles after compression

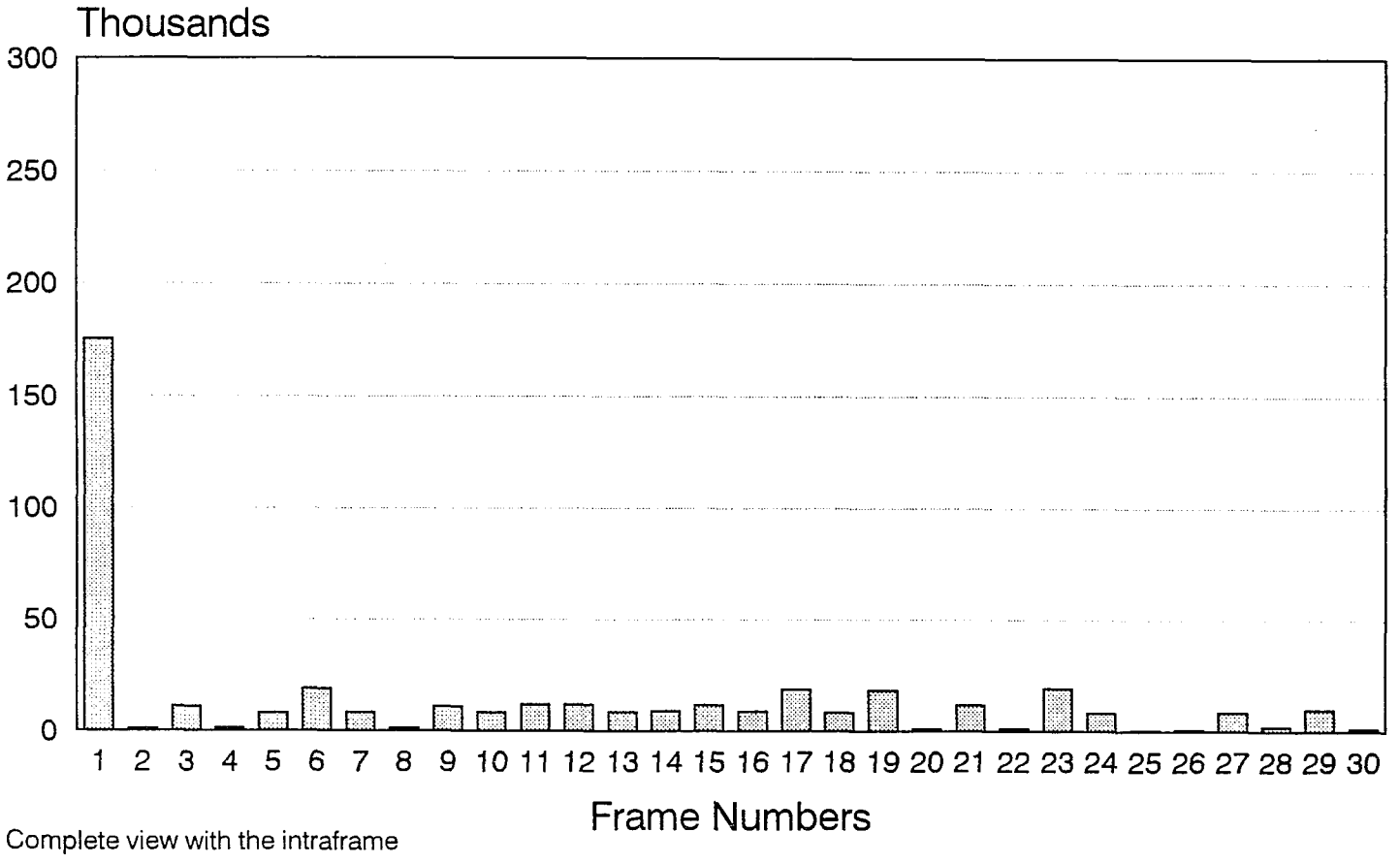



Figure 8. Pixel files with intraframe

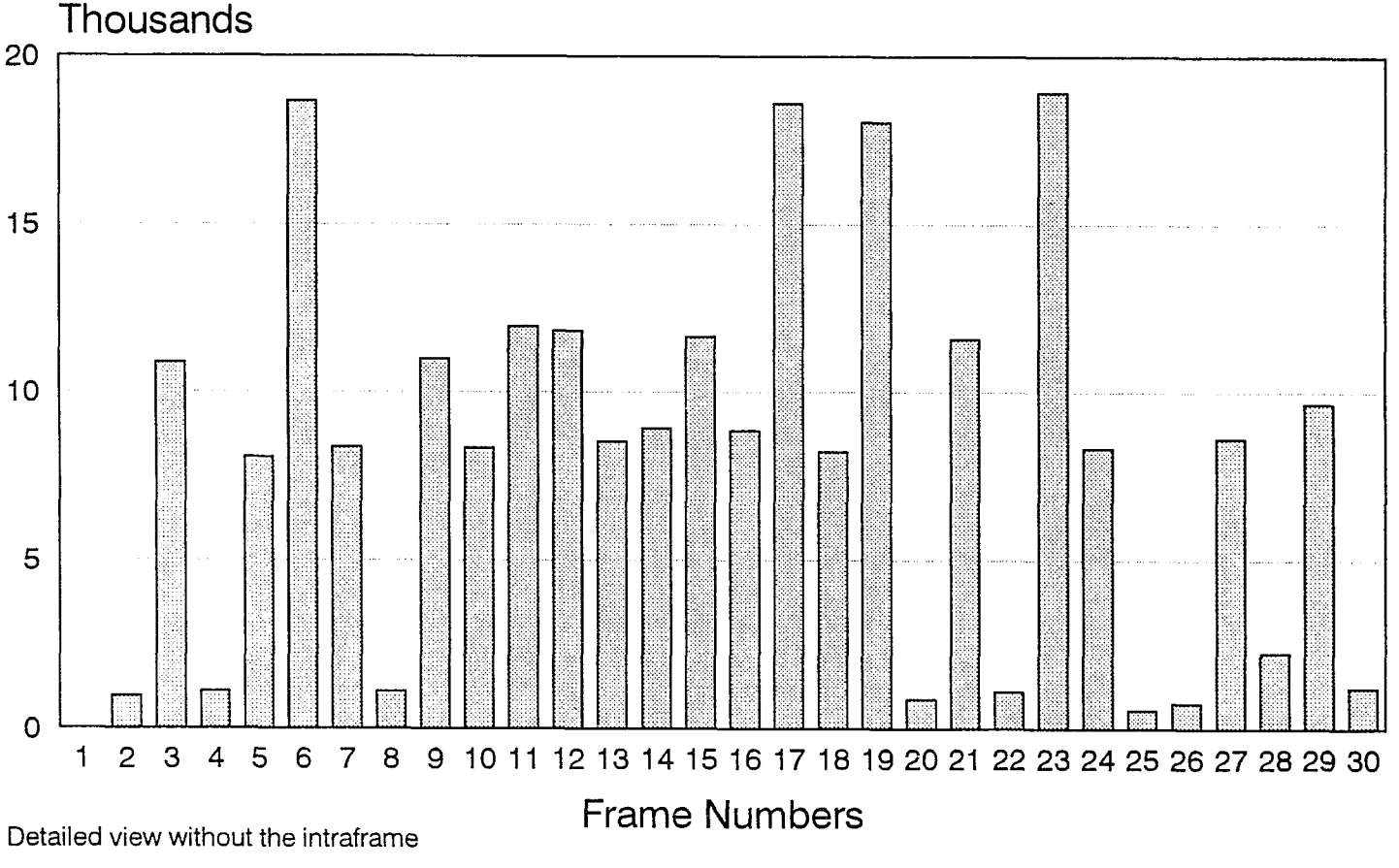



Figure 9. Pixel files without the intraframe

Sum of bitmaps and pixel files for each frame, which represents the total amount of data to be transmitted/stored:

| Frame Numbers | Frame Size | Frame Numbers | Frame Size |
|---|---|---|---|
| **Pixelfile 1** | 175143 bytes | **Pixelfile 16** | 10807 bytes |
| **Pixelfile 2** | 1915 bytes | **Pixelfile 17** | 25914 bytes |
| **Pixelfile 3** | 16986 bytes | **Pixelfile 18** | 9816 bytes |
| **Pixelfile 4** | 2204 bytes | **Pixelfile 19** | 25096 bytes |
| **Pixelfile 5** | 9719 bytes | **Pixelfile 20** | 1719 bytes |
| **Pixelfile 6** | 25807 bytes | **Pixelfile 21** | 18009 bytes |
| **Pixelfile 7** | 10084 bytes | **Pixelfile 22** | 2135 bytes |
| **Pixelfile 8** | 2152 bytes | **Pixelfile 23** | 26245 bytes |
| **Pixelfile 9** | 17142 bytes | **Pixelfile 24** | 10005 bytes |
| **Pixelfile 10** | 10067 bytes | **Pixelfile 25** | 1249 bytes |
| **Pixelfile 11** | 18633 bytes | **Pixelfile 26** | 1641 bytes |
| **Pixelfile 12** | 18403 bytes | **Pixelfile 27** | 10363 bytes |
| **Pixelfile 13** | 10372 bytes | **Pixelfile 28** | 3703 bytes |
| **Pixelfile 14** | 10802 bytes | **Pixelfile 29** | 11929 bytes |
| **Pixelfile 15** | 18162 bytes | **Pixelfile 30** | 2289 bytes |
| **TOTAL SIZE OF ALL FILES AFTER COMPRESSION :** | | | 508,475 bytes |

Table 10. Total size of all files after compression

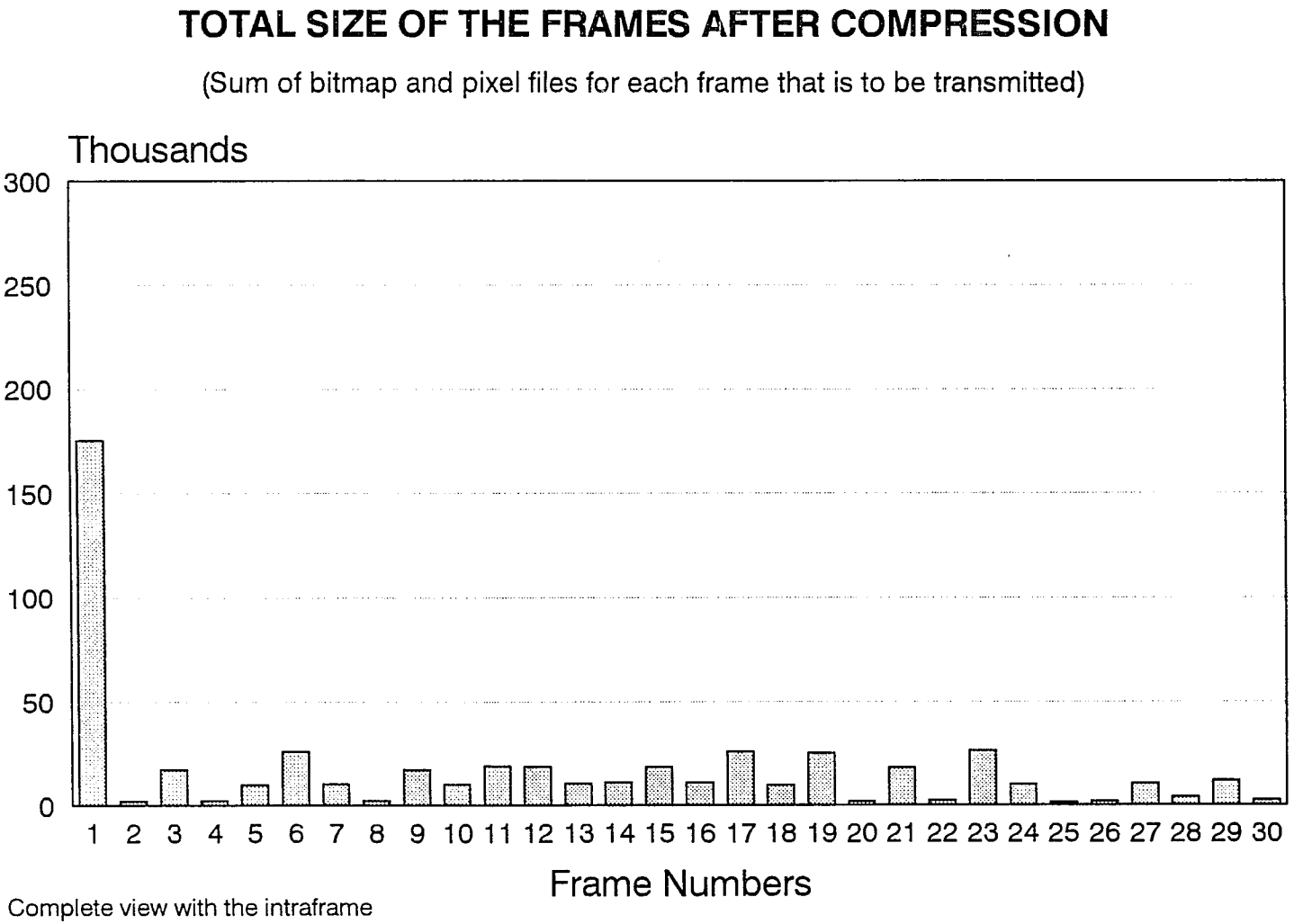



Figure 10. Total size of the frames after compression

## 8.1. Criterias for the evaluation performance of a video compression codec

The final evaluation of the performance of our compression algorithm will consider its effectiveness in terms of image quality before and after compression. The objective of our method, as it is in any other compression algorithm is to achieve the highest possible compression ratios within an acceptable compression/decompression time frame. The ultimate compression method is to offer highest compression ratio and compression /decompression efficiency that can handle up to 30 frames per second.

Scaled Differential Image Compression is a lossy compression method therefore it introduces a certain amount of distortion to the compressed images. In other words, the images transmitted to the receiver location after compression is received with a certain amount of distortion from the original values of the frames. Minimizing this distortion is meaningful in a subjective sense, even though many quantative measures have been defined. The human viewer, however, is the ultimate judge regarding the quality of the processed images. We have also run tests on human observers to test the quality of the images transmitted and made adjustments on the compression efficiency of our algorithm.

The following are the results that we have obtained from various quantative and subjective analysis we have performed on our test frames.

## 8.2. Quantative Criterias

### 8.2.1. Mean Square Error (MSE)

$$E\{[x(m,n) - \hat{x}(m,n)]^2\} = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[x(m,n) - \hat{x}(m,n)]^2 \quad \textbf{(46)}$$

where x(m,n) and $\hat{x}$(m,n) and the intensities of the original and reconstructed images in row m and column n, respectively. The image size is M x N. The difference between the original and the reconstructed images is the reconstruction error.

We performed a Mean Square Error analysis of our method and the following results are obtained. The results indicates the MSE between each original frame and the values of the same frame after decompression. The error rates are basicly limited to 2 % to 5 %. We could even reduce these error rates if had chosen to limit the "scale of change" to a value less then 10 %, but both the quantative and subjective evaluation of the video sequence, has given results that indicates the sufficiency of these values. Based on these results we can conclude that the error introduced by the lossy component of our compression algorithm does not create a significant distortion to the quality of the original image.

Mean Square Error values of each frame after decompression:

| Frame Numbers | MSE | Frame Numbers | MSE |
|---|---|---|---|
| **Frame 1** | 0.00 | **Frame 16** | 2.19 |
| **Frame 2** | 2.24 | **Frame 17** | 5.73 |
| **Frame 3** | 5.67 | **Frame 18** | 2.12 |
| **Frame 4** | 2.29 | **Frame 19** | 5.70 |
| **Frame 5** | 2.14 | **Frame 20** | 2.16 |
| **Frame 6** | 5.68 | **Frame 21** | 5.71 |
| **Frame 7** | 2.15 | **Frame 22** | 2.22 |
| **Frame 8** | 2.19 | **Frame 23** | 5.72 |
| **Frame 9** | 5.69 | **Frame 24** | 2.19 |
| **Frame 10** | 2.21 | **Frame 25** | 2.12 |
| **Frame 11** | 5.69 | **Frame 26** | 2.14 |
| **Frame 12** | 5.67 | **Frame 27** | 2.16 |
| **Frame 13** | 2.24 | **Frame 28** | 2.43 |
| **Frame 14** | 2.24 | **Frame 29** | 2.55 |
| **Frame 15** | 5.70 | **Frame 30** | 2.26 |

Table 11. Mean Square Error values

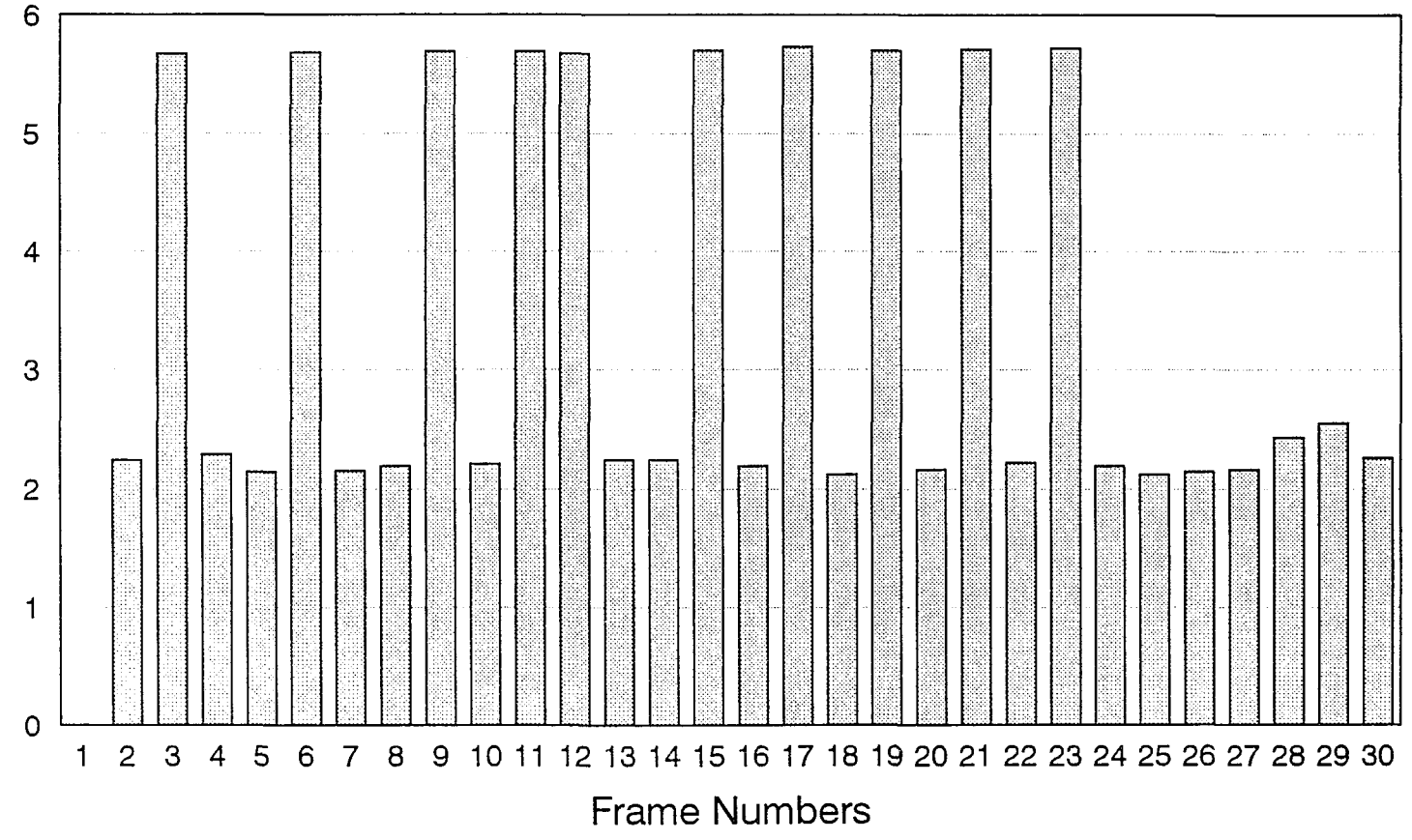



Figure 11. Mean Square Error

### 8.2.2. Normalized Mean Square Error (NMSE)

Mean Square Error gives the difference between the values of the original and the reconstructed images which is the reconstruction error. This measure does not take into consideration the image energy and may not lead to precise evaluation of the reconstruction error without the image energy levels taken into consideration. We may give the following example:

If we have 5 pairs of values to be compared:

| X1 | X2 | $(X1-X2)^2$ | | Y1 | Y2 | $(Y1-Y2)^2$ |
|---|---|---|---|---|---|---|
| 3 | 5 | 4 | | 223 | 225 | 4 |
| 6 | 3 | 9 | | 134 | 131 | 9 |
| 4 | 4 | 0 | | 367 | 367 | 0 |
| 4 | 6 | 4 | | 147 | 149 | 4 |
| 3 | 4 | 1 | | 157 | 158 | 1 |
| | | 18 | | | | 18 |
| MSE | | 4.24 | | MSE | | 4.24 |

Table 12. Comparison of MSE values

This table indicates that MSE calculated on different set of values with significantly different range of values still produce the same MSE values. This is due to the fact the Mean Square Error considers only the change in energy levels and adds the difference of those values without taking into account the intensities of the pixel values. For our

purposes this is not significant. Inorder to have a precise comparison of the reconstruction error we need to normalize the MSE values. Normalizing those MSE values will allow us to take into consideration the intensities of each pixel.

$$NMSE_{\hat{x}} = \frac{\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[x(m,n)-\hat{x}(m,n)]^2}{\frac{1}{MN}\sum_{m=1}^{M-1}\sum_{n=1}^{N-1}[x(m,n)]^2} \quad \textbf{(47)}$$

After normalizing the MSE with the image intensities, we have obtained the following results:

Normalized Mean Square Error values of each frame after decompression:

| Frame Numbers | NMSE | Frame Numbers | NMSE |
|---|---|---|---|
| **Frame 1** | 0.000000 | **Frame 16** | 0.000279 |
| **Frame 2** | 0.000291 | **Frame 17** | 0.001905 |
| **Frame 3** | 0.001856 | **Frame 18** | 0.000260 |
| **Frame 4** | 0.000305 | **Frame 19** | 0.001882 |
| **Frame 5** | 0.000265 | **Frame 20** | 0.000270 |
| **Frame 6** | 0.001872 | **Frame 21** | 0.001890 |
| **Frame 7** | 0.000270 | **Frame 22** | 0.000286 |
| **Frame 8** | 0.000279 | **Frame 23** | 0.001899 |
| **Frame 9** | 0.001882 | **Frame 24** | 0.000276 |
| **Frame 10** | 0.000283 | **Frame 25** | 0.000259 |
| **Frame 11** | 0.001871 | **Frame 26** | 0.000265 |
| **Frame 12** | 0.001864 | **Frame 27** | 0.000270 |
| **Frame 13** | 0.000291 | **Frame 28** | 0.000339 |
| **Frame 14** | 0.000291 | **Frame 29** | 0.000377 |
| **Frame 15** | 0.001884 | **Frame 30** | 0.000295 |

Table 13. Normalized Mean Square Error values

These results once again is a clear indication that the original and compressed image values are closely related and there is a minimum amount of distortion introduced after the compression.

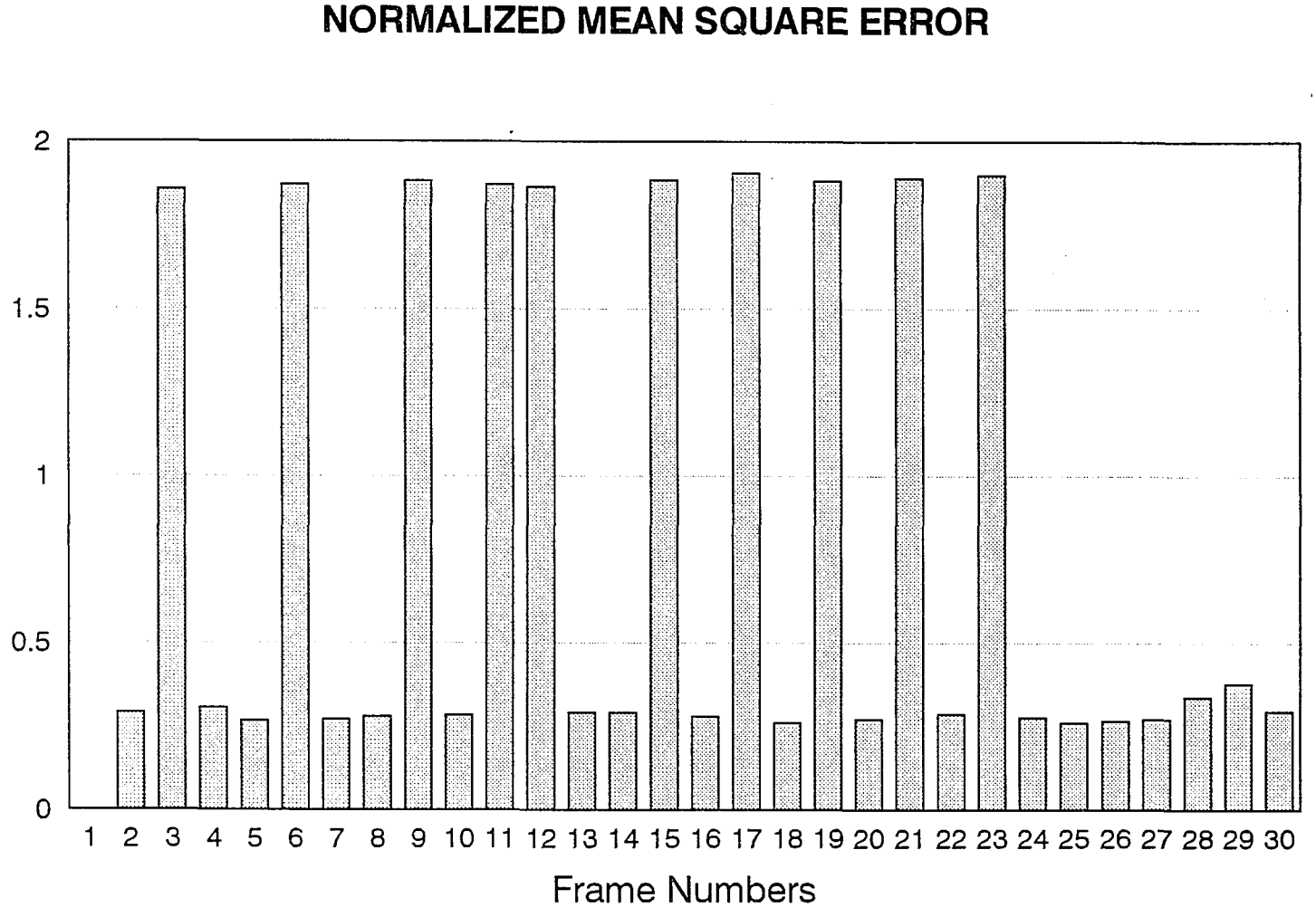



Figure 12. Normalized Mean Square Error

### 8.2.3. Averages

#### 8.2.3.1. Arithmetic Mean

The best known and most useful form of average is the arithmetic mean, usually referred to as the mean or the average. It is easily calculated by adding together all the pixel values to be averaged and dividing the sum or total by the number of pixel values.

If X is a variable which has different values $X_1, X_2, X_3$ ...., then the arithmetic mean of a number N of such values is the sum of the various values of X, which we denote by S(X), divided by N. In general, therefore,

$$m_x = \bar{X} = \frac{S(X)}{N} \tag{48}$$

If N is large and no adding machine is available, the process of addition can be made easier by the construction of a frequency distribution table. This is a table showing how often each value of the variable occurs in the image under consideration. If the frequency distribution of each pixel value is f and each pixel value is $X_k$ then the arithmetic mean is obtained by

$$m_x = \bar{X} = \frac{\sum(fX)}{N} \tag{49}$$

The following are sample frequency distribution charts for some of the frames used in our video segment.

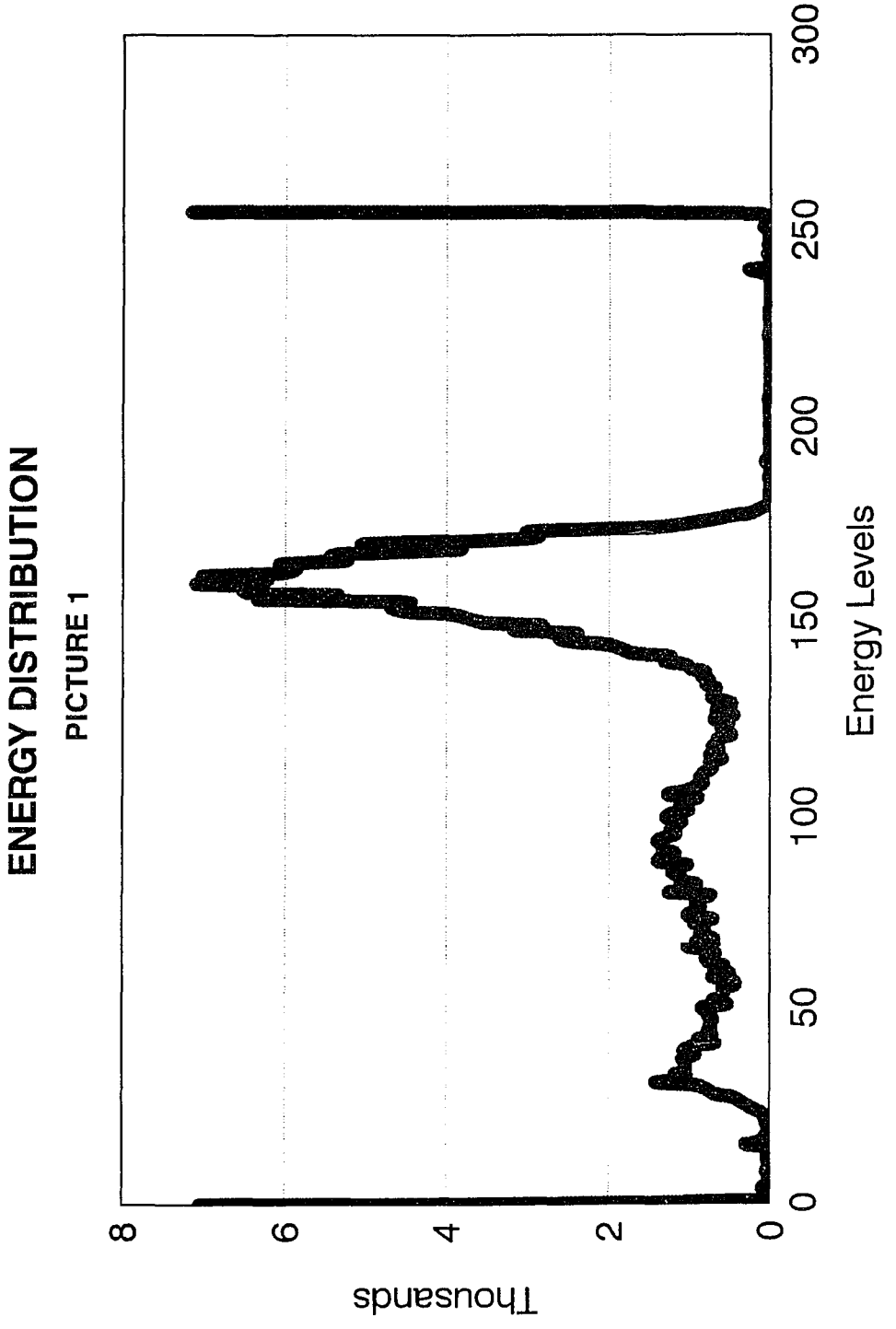



Figure 13. Energy Distribution, Picture 1

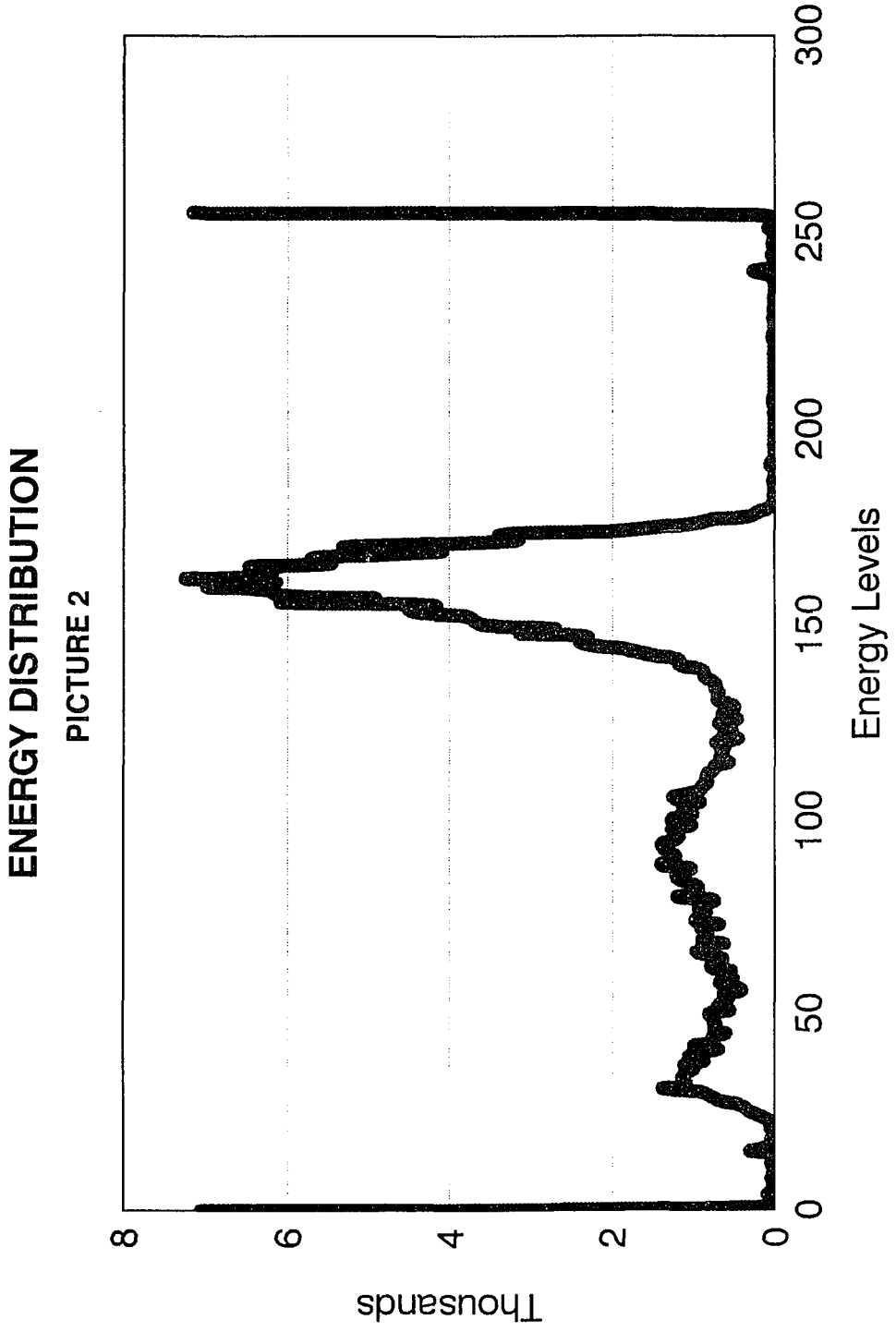



Figure 14. Energy Distribution, Picture 2

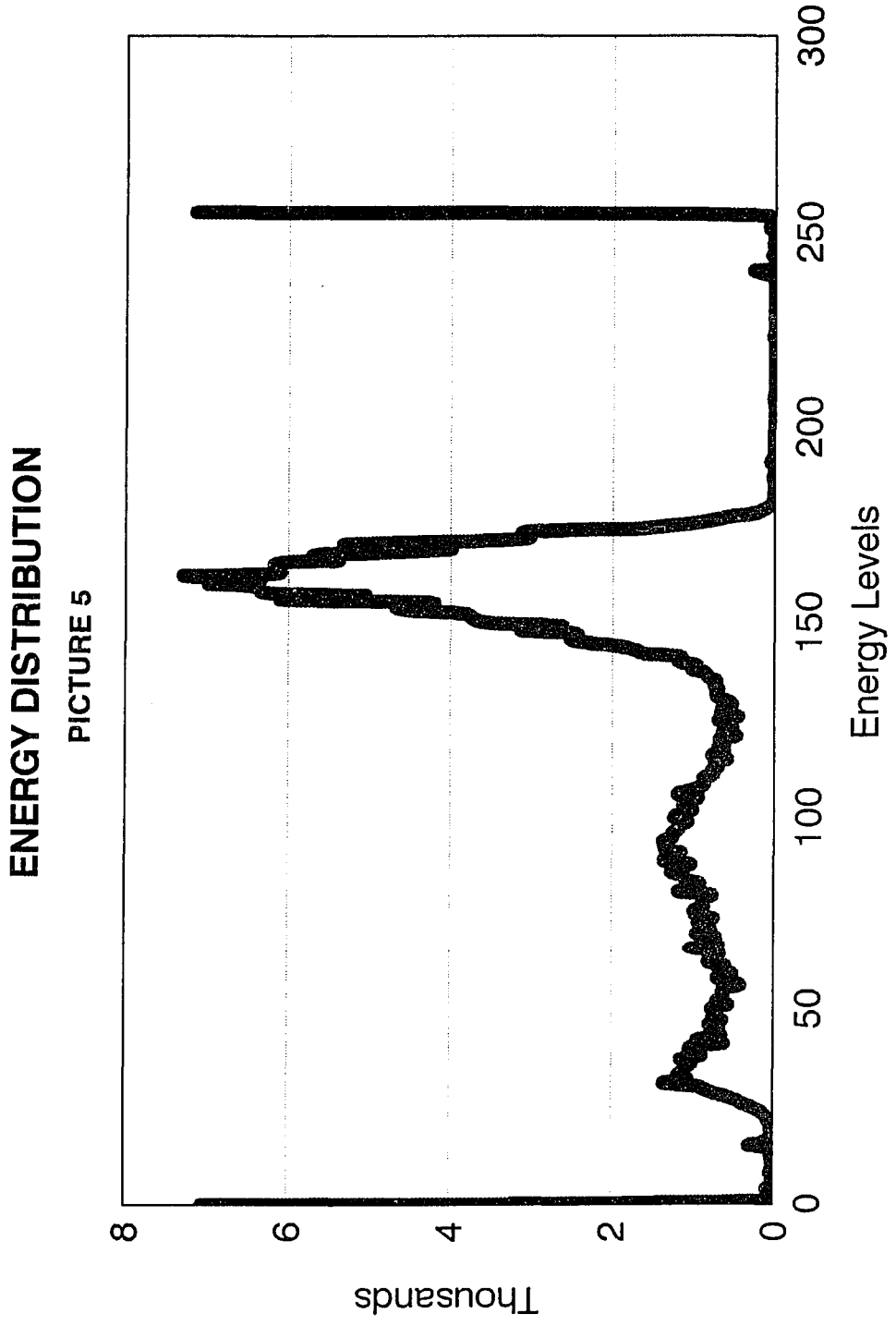



Figure 15. Energy Distribution, Picture 5

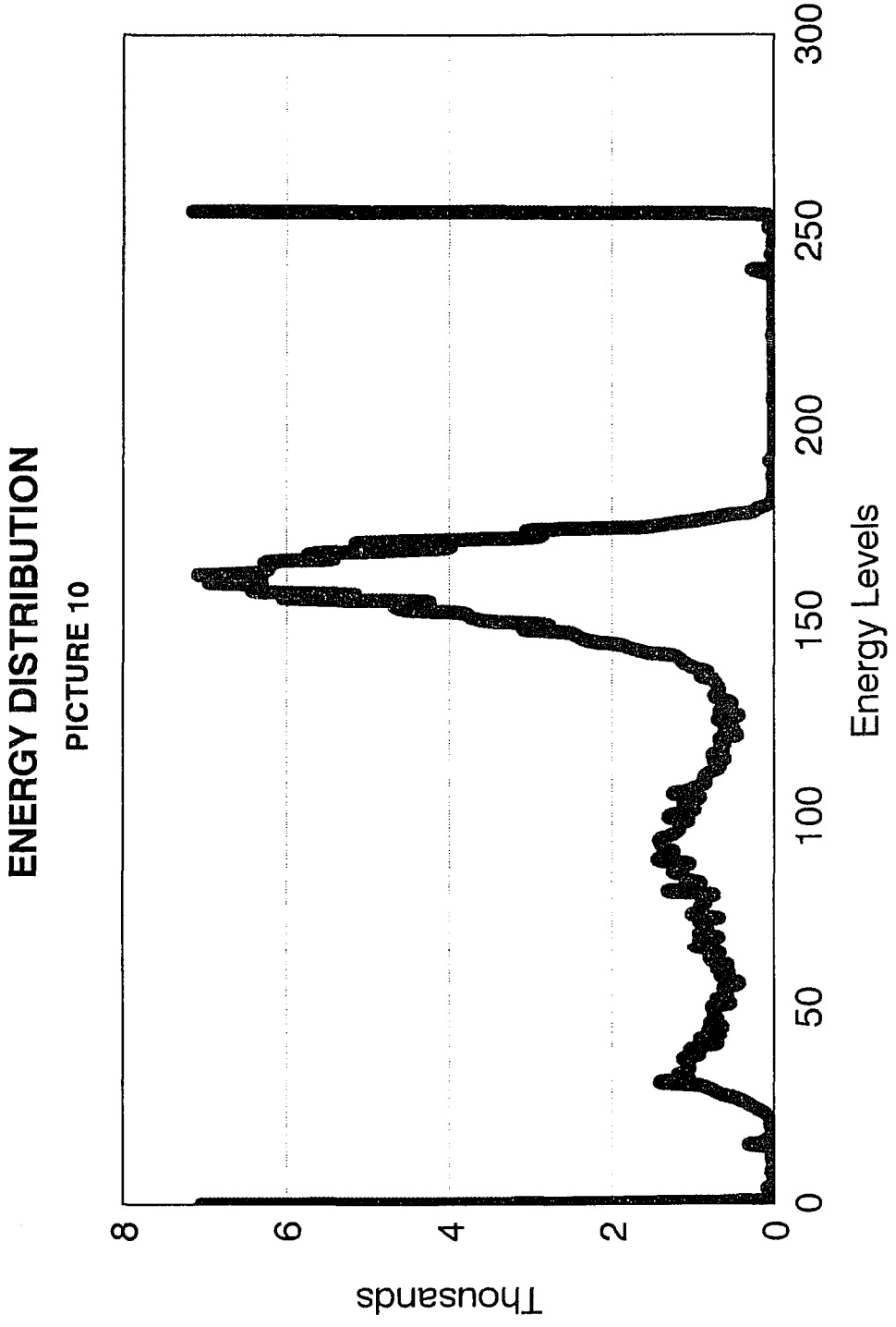



Figure 16. Energy Distribution, Picture 10

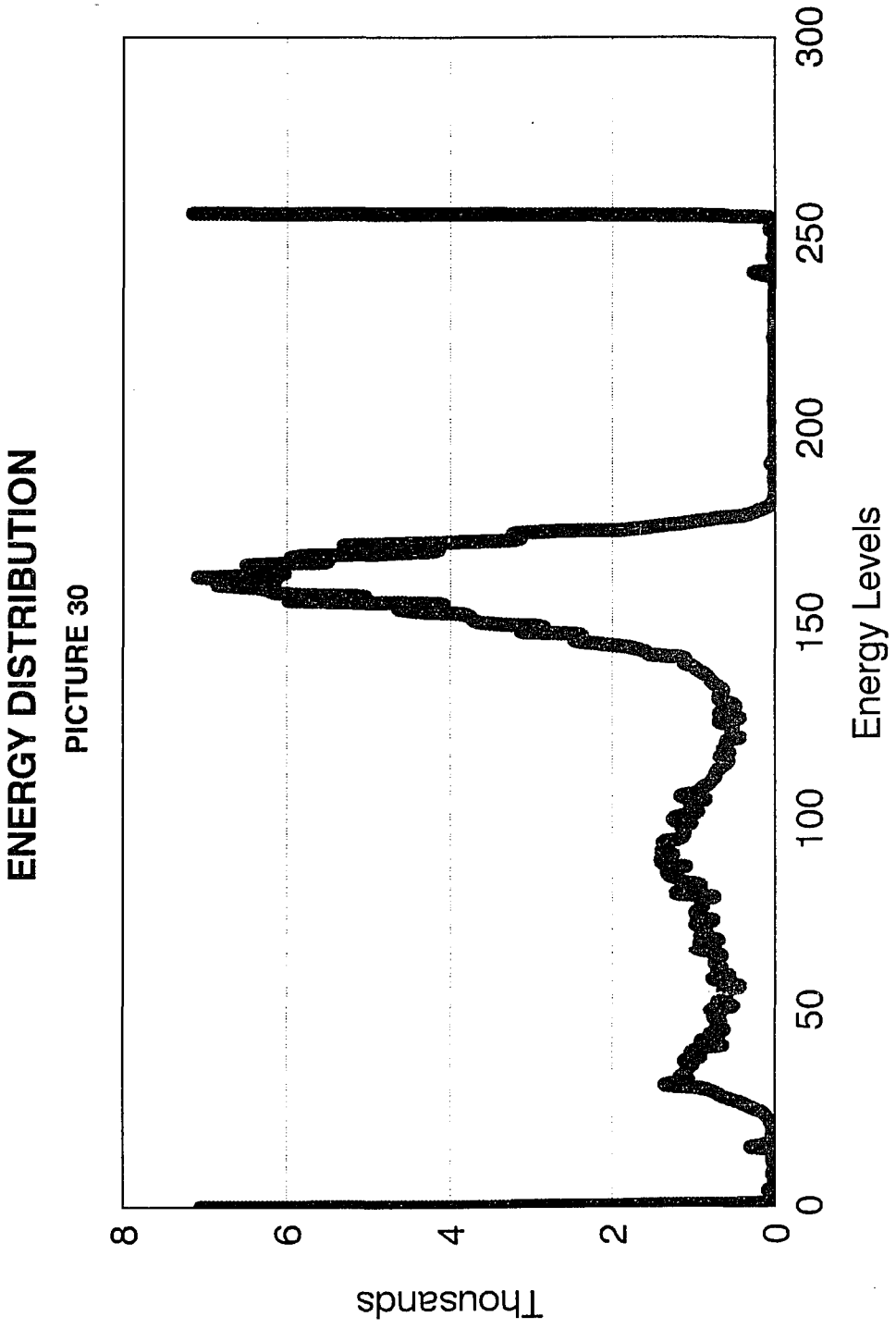



Figure 17. Energy Distribution, Picture 30

The following table shows the Arithmetic Mean values of each frame:

| Frame Numbers | Arithmetic Mean | Frame Numbers | Arithmetic Mean |
|---|---|---|---|
| **Frame 1** | 128.29 | **Frame 16** | 128.54 |
| **Frame 2** | 128.69 | **Frame 17** | 128.43 |
| **Frame 3** | 128.81 | **Frame 18** | 128.47 |
| **Frame 4** | 128.35 | **Frame 19** | 128.62 |
| **Frame 5** | 128.38 | **Frame 20** | 128.47 |
| **Frame 6** | 128.40 | **Frame 21** | 128.50 |
| **Frame 7** | 128.45 | **Frame 22** | 128.43 |
| **Frame 8** | 128.43 | **Frame 23** | 128.51 |
| **Frame 9** | 128.52 | **Frame 24** | 128.72 |
| **Frame 10** | 128.43 | **Frame 25** | 128.47 |
| **Frame 11** | 128.84 | **Frame 26** | 128.54 |
| **Frame 12** | 128.54 | **Frame 27** | 128.58 |
| **Frame 13** | 128.61 | **Frame 28** | 129.59 |
| **Frame 14** | 128.51 | **Frame 29** | 128.51 |
| **Frame 15** | 128.66 | **Frame 30** | 128.53 |

Table 14. Arithmetic Mean values

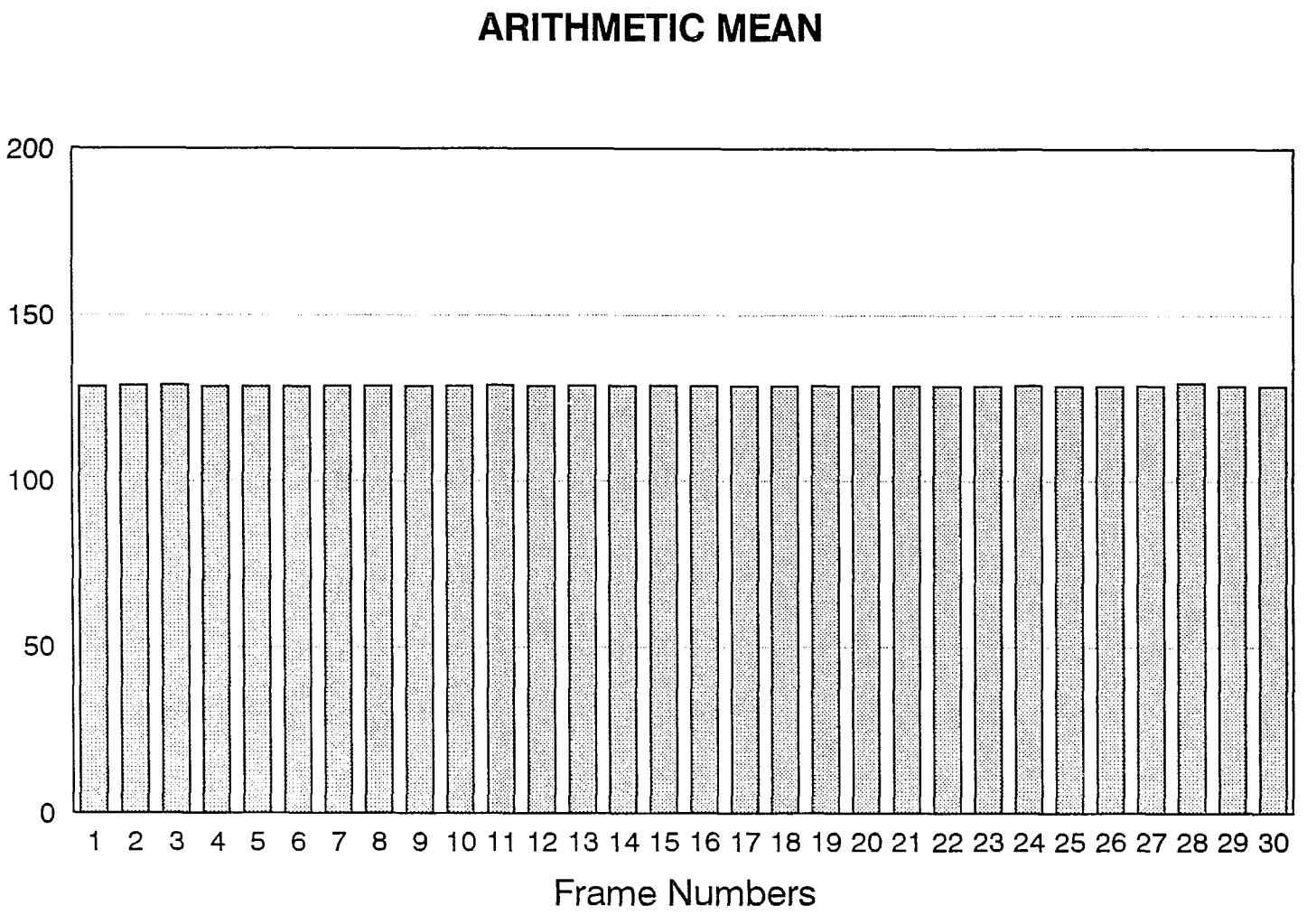



Figure 18. Arithmetic Mean

### 8.2.4. Scatter or dispersion

While an average to some extent represents the whole series of observations of which it is the mean, yet it does not by itself convey sufficient information about those observations. As a general rule it is also necessary to know how the observations are scattered around their average. Obviously the average of a given number of pixel values which lie closely about the mean is more reliable as a representative statistic than one which is in the middle of a widely dispersed series of readings. It is important, therefore, to give an indication of the amount of scatter of the observations averaged.

#### 8.2.4.1. The mean deviation

A measure of scatter which makes use of all the pixel values is obtained by writing down the difference between each separate pixel value and the average, adding together all these differences without regard to their signs, and dividing the total by the number of pixel values. This is called the mean deviation or mean variation.

#### 8.2.4.2. The standard variation

By far the best and most useful measure of scatter is the standard deviation. This is the square root of the mean of the squares of the deviations of pixel values from their arithmetic mean. If $(X - \bar{X})$ represents the deviation of an individual reading from the mean and $S(X - \bar{X})^2$ the sum of the squares of all such deviations, then the standard deviation, $\sigma$, is given by the formula

$$\sigma = \sqrt{\frac{S(X-\bar{X})^2}{N}} \qquad \textbf{(50)}$$

The square of the standard deviation is called the variance or second moment, the latter

usually being denoted by $\mu_2$.

Hence the variance,

$$\sigma^2 = \mu_2 = \frac{S(X-\bar{X})^2}{N} \tag{51}$$

The method of calculating the standard deviation depends on the data and the number of observations.

If the standard deviation of the whole population is $\sigma_p$ and we take a large number of random samples of n pixel values, then the means of the sample will be distributed with a standard deviation $\sigma_p / \sqrt{n}$ . If the population is normally distributed, the means also will be normally distributed. Even if the distribution of the population is not normal, the distribution of the means of samples still tends to be normal, if the size of the samples is sufficiently large, but in the case of small samples the distribution of the means is not normal. In our case, of course, the population (the total number of pixels in a single frame) is sufficiently large. Therefore, our calculations will be based on formulas designed for a large number of samples.

Usually we do not know the standard deviation of the whole population but have to take the standard deviation of an observed sample as an estimate of it. In this case we estimate the standard deviation of the sampling distribution from the number and standard deviation of a single sample. This estimated value is called the standard error of the mean.

$$Standard\quad Error\quad of\quad mean = \frac{\sigma}{\sqrt{N}} \tag{52}$$

where $\sigma$ is the standard deviation of the sample and N the number of observations (pixel values) in it.

### 8.2.4.3. Significance of the difference between means

An important and often occurring problem is to determine whether there is a real significant difference between the pixel values of the frames before and after compression. If we can somehow calculate this significance of the difference between compressed and uncompressed images we can have a solid measure of the quality of the images transmitted and uncompressed at the receiver site.

If the means are $X_1$ and $X_2$, their standard deviations $\sigma_1$ and $\sigma_2$ are the numbers in the samples $N_1$ and $N_2$ respectively, then the standard error of the difference between the means is given by the formula

$$Standard\quad error\quad of\quad difference = \sqrt{\left(\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}\right)} \tag{53}$$

If the difference between the two means is greater than twice its standard error then the means are significantly different.

Hence,

$$\frac{Difference\quad between\quad means}{Standard\quad error\quad of\quad difference} \leq 2 \tag{54}$$

The following results are obtained for our 30 frames:

| Frame Numbers | Significance | Frame Numbers | Significance |
|---|---|---|---|
| **Frame 1** | 0.0000 | **Frame 16** | 0.5910 |
| **Frame 2** | 2.9564 | **Frame 17** | 0.4955 |
| **Frame 3** | 0.6152 | **Frame 18** | 0.3088 |
| **Frame 4** | 3.1307 | **Frame 19** | 0.7730 |
| **Frame 5** | 0.0227 | **Frame 20** | 0.7331 |
| **Frame 6** | 0.2500 | **Frame 21** | 0.0701 |
| **Frame 7** | 0.3128 | **Frame 22** | 0.2164 |
| **Frame 8** | 0.0779 | **Frame 23** | 0.4570 |
| **Frame 9** | 0.3307 | **Frame 24** | 1.4616 |
| **Frame 10** | 0.2945 | **Frame 25** | 1.8419 |
| **Frame 11** | 1.6175 | **Frame 26** | 0.6424 |
| **Frame 12** | 1.6505 | **Frame 27** | 0.2449 |
| **Frame 13** | 0.3635 | **Frame 28** | 4.5527 |
| **Frame 14** | 0.4733 | **Frame 29** | 5.8463 |
| **Frame 15** | 0.7912 | **Frame 30** | 0.3376 |

Table 15. Significance

These results show that only 4 frames out of 30 have a significance value of greater than 2, which means that only those frames had a significantly different values in comparison to the original ones.

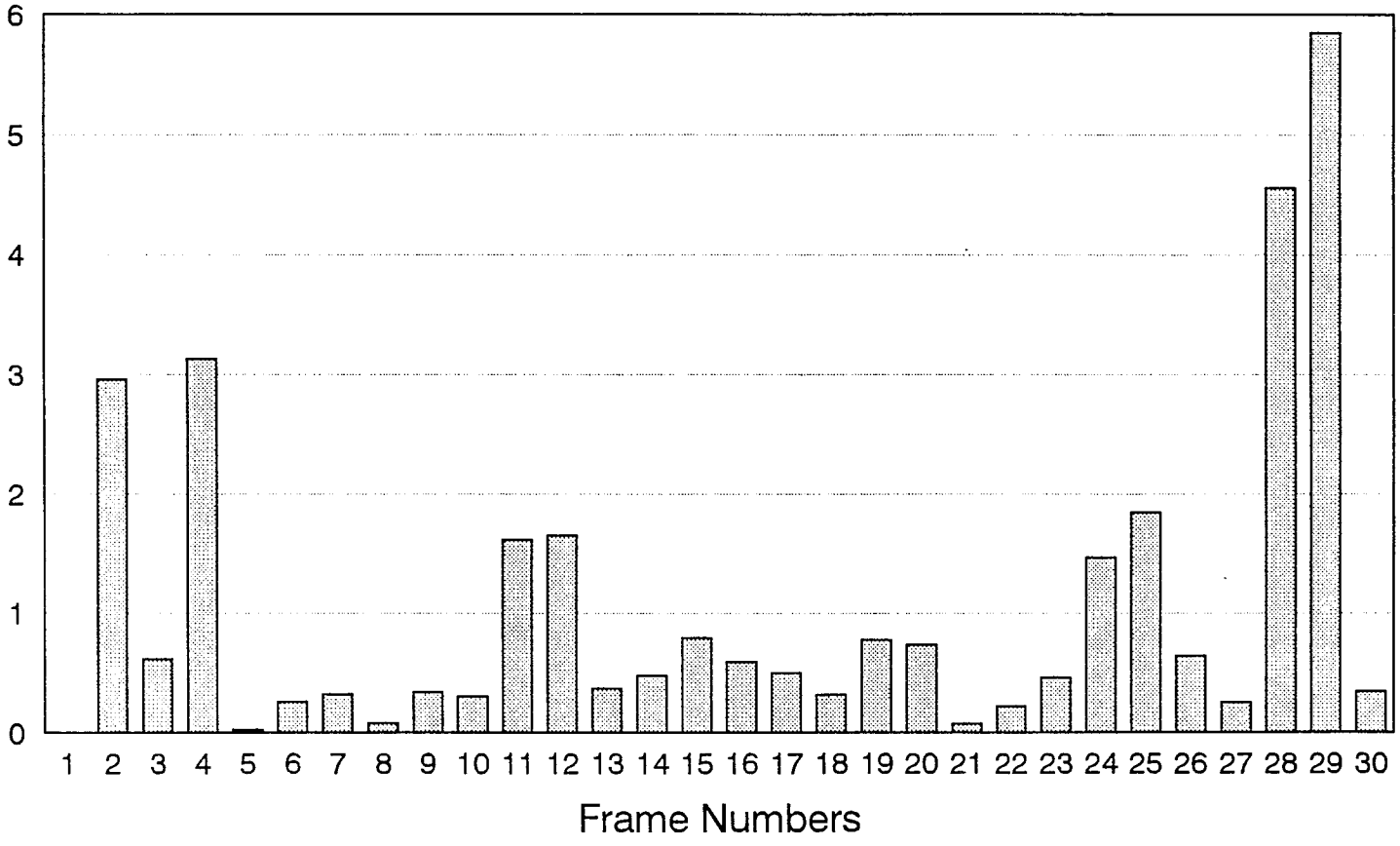



Figure 19. Significance

## 9. CONCLUSION

We have presented a compression/decompression algorithm suitable for videoconferencing applications. SDIC offers a number of promising solutions to problems that exist in symmetric applications that require real-time compression/decompression. It achieves high compression ratios without sacrificing image quality as demonstrated in our sample compressed video sequence. It offers features that are essential to a widely accepted compression algorithm such as random access to any point within the video segment, fast-forward searches, and quick recovery from errors.

"Scale of change" parameter which can be dynamically adjusted by the system depending on the bit rate at the output of the compressor, enables the system to maintain a constant bit rate. This allows SDIC to seamlessly integrate with ISDN lines without additional considerations to stay within the allocated bandwidth. This property alone, puts SDIC ahead of its competitors in terms of ease of implementation and applicability to videoconferencing.

Finally, SDIC offers all of this while keeping a high image quality with minimum distortion, making the compression process almost invisible to the user. After all, the ultimate factor in deciding for the best compression method will be the users of the technology and their satisfaction with the quality of the video that they see.

**APPENDIX 1.**

```
program readfast;
{ This is a program that reads an image file with upto
  256 shades of gray and displays it. }
{ One point to notice is the fact that our sample
  images has a resolution of 512 x 512. This is
  resolution is not supported by VGA, the closest
  resolution was 640 x 480. Therefore, we used this
  resolution to display our images but used the full
  512 x 512 image for processing and compression }

uses
 graph, crt, dos;

type    { ColorValue fields hold the value of each
          Red, Green and Blue color components of
          the picture }
  ColorValue     = record
                     Rvalue, Gvalue, Bvalue : byte;
                   end;
  VGAPaletteType = array[0..255] of ColorValue;

var   { This is the list of variables }
 row, column : longint;
 infile      : file ;
 i,offset    : word;
 value       : byte;
 buffer      : array[1..32768] of byte;
               { The disk buffer used to read a
                 complete 32K within one block read }
 total       : integer;

{ This procedure sets the VGA Palette colors. Assigns
  New Palette as the current palette with all new color
  assignments affected immediately. The color values
  for New Palatte must be assigned using SetAllPalette}

procedure VGASetAllPalette(var P : VGAPaletteType);
var
  Regs : Registers;
begin
  with Regs do
  begin
    AX := $1012;
    BX := 0;
    CX := 255;
    ES := Seg(P);
    DX := Ofs(P);
  end;
    Intr($10, Regs);
```

```
end; { VGASetAllPalette }

{ This function detects the Video Graphics card and
  initializes Graphics Driver. If a problem occurs,
  GraphDriver returns an error code. }

{$F+}
Function DetectVGA256 : integer;
var
  DetectedDriver : integer;
  SuggestedMode  : integer;
begin
  DetectGraph(DetectedDriver, SuggestedMode);
  if (DetectedDriver = VGA) or (DetectedDriver = MCGA)
 then
   DetectVGA256 := 2          { Default video mode = 0 }
 else
   DetectVGA256 := grError; { Couldn't detect hardware}
end; { DetectVGA256 }
{$F-}

{ This procedure sets the initial graphics parameter
  values, loads the proper graphics driver, and set the
  system to the desired graphics mode. }

procedure initialize;
var
  Z                  : integer;
  C                  : ColorValue;
  P                  : VGAPaletteType;
  AutoDetectPointer : pointer;
  Driver, Mode       : integer;
  ErrorCode          : integer;
begin
  ClrScr;
  DirectVideo := false;
  AutoDetectPointer := @DetectVGA256;
                       { Point to detection routine }
  Driver := InstallUserDriver('sVGA256',
                        AutoDetectPointer);
  Driver := Detect;
  InitGraph(Driver, mode, 'c:\tp\bgi');

  ErrorCode := GraphResult;
  if ErrorCode <> grOK then { If there is anykind of
                              error }
     begin          { display the error code and stop. }

      Writeln('Error: ', GraphErrorMsg(ErrorCode));
      Halt
     end;
```

```
{ otherwise continue .. }
{ create grey scale, these are the R,G, and B values
  for a gray scale image }
for Z := 0 to 255 do
    with P[Z] do
        begin
           Rvalue := Z div 4;
           Gvalue := Z div 4;
           Bvalue := Z div 4;
        end;
  VGASetAllPalette(P);
end;  { Initialize }

{ This is the main program }

begin
 initialize;    { Call the initialize subroutine }
 clrscr;
 assign(infile,'c:\tp\pict\pict01');
 reset(infile,1);
 offset:=0;
 i:=1;
 for total:=1 to 8 do
   begin
     blockread(infile,buffer,32768);
          { Read the first block, and so on }
     i:=1;                { Total of 8 blocks }
       for column:=1 to 64 do
        begin
          for row:=1 to 480 do
            begin
             value:=buffer[i];
             putpixel((column+offset),row,value);
                     { Display the pixel values }
             i:=i+1;
            end;
            i:=i+32;
     { This is offset is to skip any lines beyond 480 }
         end;
        offset:=offset+64;
    end;
 readln;    { Stop until a key is pressed }
 closegraph;
 close(infile);
end.  { Main program }
```

```
{ This is the program that does the Scaled Differential
  Image Compression. It runs the compression until all
  the image files are compressed, in our case, for 30
  frames. It starts by reading the first frame and
  saves it as an Intraframe, consecutive frames are
  compared to this frame and pixels that are
  P(i+1)(m,n) > P(i)(m,n) + 10% or P(i+1)(m,n)
        < P(i)(m,n) - 10%
  are saved in a new pixel file along with a bitmap
  table which contains the address information for
  decompression }


program compreser;
uses
 graph,crt,dos;
var
 column, eightpack, count, notsame, same,j : longint;
 total, byteno, digit1, digit2, nextfile : integer;
 pict : string;
 picnotsame : byte;
 infile1 : file ; { First frame }
 infile2 : file ; { Next frame }
 outfile1 : file; { Bitmap file }
 outfile2 : file of byte; { Pixel value file }
 pic1, pic2 : array [1..8192] of byte;
 bitmap : array [1..32768] of byte;

Procedure Initialize;  { Clear the previos bit-map }
begin
  for j:=1 to 32768 do
    bitmap[j]:=0;
End;

Procedure Compress;
begin
 clrscr;
 digit1 := $30;
 digit2 := $31;
 pict   := 'pict';
 While true do
    begin
       writeln('Compressing file no:
              ',pict+chr(digit1)+chr(digit2));
assign(infile1,'c:\tp\pict\'+pict+chr(digit1)+chr(digit2));
writeln('First file  : ',pict+chr(digit1)+chr(digit2));
       digit2:=digit2+1;
       if digit2 = $3A then
          begin
```

```
                digit1:=digit1+1;
                digit2:=$30;
               end;

assign(infile2,'c:\tp\pict\'+pict+chr(digit1)+chr(digit2));
assign(outfile1,'c:\tp\pict\bitmp'+chr(digit1)+chr(digit2));
assign(outfile2,'c:\tp\pict\cmprs'+chr(digit1)+chr(digit2));

       reset(infile1,8192);
       reset(infile2,8192);
       If IOResult <> 0 then
         begin
           exit
         End;
       rewrite(outfile1,32768);
       rewrite(outfile2);
       writeln('Second file  :
              ',pict+chr(digit1)+chr(digit2));
       same:=0;
       notsame:=0;
       eightpack:=1;

      for total:=1 to 32 do
        begin
          column:=1;
          blockread(infile1,pic1,1);
          blockread(infile2,pic2,1);
            for count:=1 to 1024 do
              begin
                for byteno:=1 to 8 do
                   begin
                      if ((pic1[column]+13 > pic2[column])
                            and (pic2[column] >
                             pic1[column]-13)) then
                         begin

                bitmap[eightpack]:=bitmap[eightpack] shl 1;
                bitmap[eightpack]:=bitmap[eightpack] or 1;
                            column:=column+1;
                            same:=same+1;
                          end
                       else
                          begin

                bitmap[eightpack]:=bitmap[eightpack] shl 1;
                            picnotsame:=pic2[column];
                            write(outfile2,picnotsame);
                            column:=column+1;
                            notsame:=notsame+1;
                          end;
```

```
                end;
              eightpack:=eightpack+1;
          end;
    end;

    blockwrite(outfile1,bitmap,1);

  close(infile1);
  close(infile2);
  close(outfile1);
  close(outfile2);
  writeln('notsame ',notsame);
  writeln('same ',same);
end;
end;

begin   { Main program }
   Initialize;
   Compress;
end.
```

```
 {This is the uncompression program. It
 uncompresses all the images compressed by the
 Scaled Differential Image Compression algorithm.
 It uses two input files, one is the bitmap and
 the second is the pixel file. Initially displays
 the complete Intraframe, following that it reads
 the pixel file associated with the next frame,
 traverses each bit and whenever it encounters a
 "1" it displays a pixel from the pixel value
 file. It performs this operation for every N x M
 pixel and then for the next 29 frames. }

program uncompresser;

uses
 graph,crt,dos;

type
  ColorValue     = record
                     Rvalue, Gvalue, Bvalue : byte;
                   end;
  VGAPaletteType = array[0..255] of ColorValue;

var
 i, row, column : longint;
 infile  : file ;
 infile2 : file;
 infile3 : file of byte;
 eightpack,offset, notequal : word;
 value, pictbyte,mask : byte;
 buffer : array[1..16384] of byte;
 bitmap : array[1..32768] of byte;
 total, byteno, bitvalue, rowact : integer;
 digit1, digit2, newfile : integer;

procedure VGASetAllPalette(var P:VGAPaletteType);
var
  Regs : Registers;
begin
  with Regs do
  begin
    AX := $1012;
    BX := 0;
    CX := 255;
    ES := Seg(P);
    DX := Ofs(P);
  end;
  Intr($10, Regs);
end; { VGASetAllPalette }
```

```
{$F+}
function DetectVGA256 : integer;
var
  DetectedDriver : integer;
  SuggestedMode  : integer;
begin
  DetectGraph(DetectedDriver, SuggestedMode);
  if (DetectedDriver = VGA) or (DetectedDriver =
      MCGA) then  DetectVGA256 := 2
            { Default video mode = 0 }
  else
    DetectVGA256 := grError;
      { Couldn't detect hardware }
end; { DetectVGA256 }
{$F-}

procedure initialize;
var
  Z : integer;
  C       : ColorValue;
  P       : VGAPaletteType;
  AutoDetectPointer : pointer;
  Driver, Mode : integer;
  ErrorCode : integer;
begin
  ClrScr;
  DirectVideo := false;
  AutoDetectPointer := @DetectVGA256;
       {Point to detection routine }
  Driver := InstallUserDriver('sVGA256',
                              AutoDetectPointer);
  Driver := Detect;
  InitGraph(Driver, mode, 'c:\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln('Error: ', GraphErrorMsg(ErrorCode));
    Halt;
  end;
{ create grey scale }
  for Z := 0 to 255 do
    with P[Z] do
    begin
      Rvalue := Z div 4;
      Gvalue := Z div 4;
      Bvalue := Z div 4;
    end;
  VGASetAllPalette(P);
end;
```

```
begin {main}
 initialize;
 clrscr;
 assign(infile,'c:\tp\pict\pict01');
 reset(infile,16384);

 offset:=0;
 i:=1;
 for total:=1 to 16 do
   begin
     blockread(infile,buffer,1);
       i:=1;
       for column:=1 to 32 do
        begin
          for row:=1 to 480 do
            begin
             value:=buffer[i];
             putpixel((column+offset),row,value);
             i:=i+1;
            end;
            i:=i+32;
         end;
        offset:=offset+32;
   end;
 readln;
 close(infile);
 digit1:=$30;
 digit2:=$32;

 for newfile:=1 to 29 do begin

 offset:=0;
 i:=1;
 mask:=128;
 eightpack:=1;
 row:=1;
 value:=0;


assign(infile2,'c:\tp\pict\bitmp'+chr(digit1)+chr(digit2));
assign(infile3,'c:\tp\pict\cmprs'+chr(digit1)+chr(digit2));
            reset(infile2,32768);
            reset(infile3);
            digit2:=digit2+1;
            if digit2=$3A then
               begin
                  digit1:=digit1+1;
                  digit2:=$30;
               end;
```

```
blockread(infile2,bitmap,1);
for total:=1 to 16 do
   begin
     i:=1;
       for column:=1 to 32 do
        begin
          for rowact:=1 to 64 do
            begin
              for byteno:=1 to 8 do
                 begin
                      bitvalue:=bitmap[eightpack]
                                  and mask;
       bitmap[eightpack]:=bitmap[eightpack] shl 1;
                       if (bitvalue = 0) then
                          begin
                            read(infile3,value);
                            if (row<480) then
            putpixel((column+offset),row,value);
                            row:=row+1;
                          end
                        else
                          begin
                            row:=row+1;
                           end;
                    end;
                 eightpack:=eightpack+1;
            end;
            row:=1;
         end;
      offset:=offset+32;
   end;
  readln;

  close(infile2);
  close(infile3);
 end;
 closegraph;
end. {main}
```

```
{ This program calculates the Standard Error of
  Difference and the significance of the change of
  values between the original images and the compressed
  ones}

program standarderr;
uses
 graph,crt,dos;
var
 column, eightpack, count,notsame, same,j : longint;
 total, byteno, digit1, digit2,code, nextfile :integer;
 pict,line1,line2,m1,m2 : string;
 sig,sumx1, sumx2, bolum, seofdif : real;
 first,second, diff1, diff2, mean1, mean2 : real;
 ro1, ro2, average1, average2 : real;
 picnotsame, picsame : byte;
 i : integer;
 infile1 : file ;  { First frame }
 infile2 : file ;  { Following frame}
 infile3 : text ;  { The frame numbers are read from
                     this file}
 infile4 : text ;  { The arithmetic mean values are
                     read from this file}
 outfile : text ;  { This is the output file where the
                     results are written}
 number1,number2 : string[2];
 pic1, pic2 : array [1..16384] of byte;
begin
  clrscr;
  assign(outfile,'d:\tp\stat\stnddev.dat');
  rewrite(outfile);
  assign(infile3,'d:\tp\stat\numbers.lis');
  reset(infile3);
  assign(infile4,'d:\tp\stat\mean1_2.dat');
  reset(infile4);
  for i := 1 to 29 do
    begin
      clrscr;
      Readln(infile3,number1);
      Readln(infile3,number2);
      write(number1,'  ',number2);
      Readln(infile4);
      Readln(infile4);
      Readln(infile4,line1);
      Readln(infile4,line2);
      Readln(infile4);
      val(copy(line1,11,10),mean1,code);
      val(copy(line2,11,10),mean2,code);
      assign(infile1,'c:\tp\pict\pict'+number1);
      assign(infile2,'c:\tp\pict\pict'+number2);
```

```
        reset(infile1,16384);
        reset(infile2,16384);
        same:=0;
        notsame:=0;
        sumx1 :=0;
        sumx2 := 0;
        diff1:=0;
        diff2:=0;
        average1 :=0;
        average2:=0;
        ro1:=0;
        ro2:=0;
        for total:=1 to 16 do
          begin
           write(total);
             blockread(infile1,pic1,1);
             blockread(infile2,pic2,1);
              for column:=1 to 16384 do
               begin
                if ((pic1[column]+13 > pic2[column]) and
                    (pic2[column] > pic1[column]-13))
                   then
                    begin
                      picsame:=pic1[column];
                      diff1 := sqr(pic2[column] - mean1);
                      diff2 := sqr(pic1[column] - mean2);
                      sumx1 := sumx1 + diff1;
                      sumx2 := sumx2 + diff2;
                      same:=same+1;
                    end
                   else
                     begin
                      picnotsame:=pic2[column];
                      diff1 := sqr(pic2[column] - mean1);
                      diff2 := sqr(pic2[column] - mean2);
                      sumx1 := sumx1 + diff1;
                      sumx2 := sumx2 + diff2;
                      notsame:=notsame+1;
                   end;
               end;
  end;
close(infile1);
close(infile2);
average1 := sumx1 / 262144;
average2 := sumx2 / 262144;
ro1 := sqrt(average1);
ro2 := sqrt(average2);
```

```
 bolum := (sqr(ro1)/262144)+(sqr(ro2)/262144);
 seofdif := sqrt(bolum);
 sig := (mean1 - mean2) / seofdif;
 writeln(outfile,'Notsame = ',notsame,'     Same =
        ',same);
 writeln(outfile,' Ro1 = ',ro1:8:6);
 writeln(outfile,' Ro2 = ',ro2:8:6);
 writeln(outfile,' S.e. of difference = ',seofdif:8:6);
 writeln(outfile,' Significance       = ',sig:8:4);
 writeln(outfile);
end;
close(outfile);
close(infile3);
close(infile4);
End.
```

```
{ This program calculates the Mean Square Error and
  Normalized Mean Square Error values}

program meanerrorvals;
uses
 graph,crt,dos;
var
 column, eightpack, count,notsame, same,j : longint;
 total, byteno, digit1, digit2, nextfile  : integer;
 pict : string;
 first,second,gtotal,mse,pic2total,nmse,bottom : real;
 picnotsame, picsame : byte;
 i : integer;
 number1,number2 : string[2];
 infile1 : file ;  { First frame }
 infile2 : file ;  { Next frame }
 infile3 : text ;  { Frame numbers }
 outfile : text ;  { Output file where the results are
                     written }
 pic1, pic2 : array [1..16384] of byte;
begin
assign(outfile,'d:\tp\stat\nmse.dat');
rewrite(outfile);
assign(infile3,'d:\tp\stat\numbers.lis');
reset(infile3);
for i := 1 to 29 do
begin
  clrscr;
  Readln(infile3,number1);
  Readln(infile3,number2);
  assign(infile1,'c:\tp\pict\pict'+number1);
  assign(infile2,'c:\tp\pict\pict'+number2);
  reset(infile1,16384);
  reset(infile2,16384);
  same:=0;
  notsame:=0;
  gtotal:=0;
  pic2total := 0;
 first:=0;
 second:=0;
 for total:=1 to 16 do
  begin
     write(total);
     blockread(infile1,pic1,1);
     blockread(infile2,pic2,1);
     for column:=1 to 16384 do
      begin
       if ((pic1[column]+13 > pic2[column]) and
           (pic2[column] > pic1[column]-13)) then
```

```
          begin
             picsame:=pic1[column];
             first:=picsame-pic2[column];
             second:=sqr(first);
             gtotal := gtotal+second;
             pic2total := pic2total + sqr(pic2[column]);
             same:=same+1;
          end
         else
          begin
             picnotsame:=pic2[column];
             first:=picnotsame-pic2[column];
             second:=sqr(first);
             gtotal := gtotal+second;
             pic2total := pic2total + sqr(pic2[column]);
             notsame:=notsame+1;
           end;
       end;
    end;

 close(infile1);
 close(infile2);
 writeln(outfile,'Picture :',number1,'  to ',number2);
 writeln(outfile,'Notsame = ',notsame,'      Same =
         ',same);
 writeln(outfile,'Total    = ',gtotal:12:0);
 mse := gtotal/262144;
 writeln(outfile,'MSE      = ', mse:4:2);
 writeln(outfile,'Picture 2 total : ',pic2total:14:2);
 bottom:=pic2total/262144;
 writeln(outfile,'Bottom   = ',bottom:10:2);
 nmse:=mse/bottom;
 writeln(outfile,'NMSE     = ',nmse:8:6);
 writeln(outfile);
end;
 close(outfile);
 close(infile3);
End.
```

## 10. REFERENCES

1. P. Ang, P. Ruetz, D. Auld. *Video Compression Makes Big Gains*. IEEE Spectrum. October, 1991. pp. 16-19.

2. M. Mann. *Image Compression Resolves Storage Hassles*. PC Week Reviews. June 10,1991. pp. 94/98.

3. G. Frenkel. *Three Mac Compression Solutions Excel*. PC Week Reviews. June 10,1991. pp. 94/96.

4. G.K. Wallace. *The JPEG Still Picture Compression Standard*. Comm. of the ACM. April, 1991. pp. 31-44.

5. G. Zorpette. *Fractals: not just another pretty picture*. IEEE Spectrum. October 1988. pp. 29-31.

6. J. Bridges. *Differential Image Compression: an ideal technique for compressing animated sequences*. Dr. Dobb's Journal. Feb, 1991. pp. 38-46.

7. D. Knuth. *Dynamic Huffman Coding*. Journal of Algorithms. June, 1985. pp. 163-180.

8. V. Miller, M. Wegman. *Variations on a theme by Ziv and Lempel*. IBM Res. Rep. RC 10630 (#47798). July, 1984.

9. J. Ziv, A. Lempel. *A Universal Algorithm for Sequential Data Compression*. Computer, Vol. 17. No. 6. 1984. pp. 8-19.

10. J. Ziv, A. Lempel. *Compression of Individual Sequences Via Variable-Rate Coding*. IEEE Trans. Info. Theory. IT-24. Sept. 1978. pp. 530-536.

11. N.M. Abramson. *Information Theory and Coding*. McGraw-Hill, New York. 1963.

12. J. Rissanen, G.G. Langdon. *Universal modeling and coding*. IEEE Trans. Inform. Theory. IT-27. 1981

13. C.B. Jones. *An efficient coding system for long sequences*. IEEE Trans. Inform. Theory. IT-27. 1981. pp. 280-291.

14. T.A. Welch. *A technique for high-performance data compression*. Computer. 1984. pp. 8-18.

15. CCITT SG XV, Specialists Group on Coding for Visual Telephony. *Description of Reference Model 8*. Document 525, June 1989.

16. A. Puri, R. Aravind. *On Comparing Motion-Interpolation Structures for Video*

*Coding*. SPIE, Visual Communications and Image Processing '90. Vol 1360. pp. 1560-1571.

17. K. Kamikura, H. Watanabe. *Video coding for digital storage media using hierarchical intraframe scheme*. SPIE Visual Communications and Image Processing. 1990. Vol 1360. pp. 1540-1549.

18. A. Puri, R. Aravind. *On comparing motion-interpolation structures for video coding*. SPIE Visual Communications and Image Processing. 1990. Vol. 1360. pp. 1560-1567.

19. S. Kappagantula, K.R. Rao. *Motion compensated predictive coding*. Proc. Int. Tech. Symp. SPIE. Aug. 1983.

20. H.C. Bergmann. *Displacement estimation based on the correlation of image segments*. IRE Conf. Electron. Image Processing. July 1982.

21. J.R. Jain, A.K. Jain. *Displacement measurement and its application in interframe image coding*. IEEE Trans. Commun. COM-29. Dec. 1981. pp. 1799-1808.

22. T. Koga, K. Iinuma, A. Hirano, T. Ishiguro. *Motion compensated interframe coding for videoconferencing*. Proc. Nat. Telecommun. Conf. Dec. 1981. pp. G5.3.1-5.3.5

23. A. Puri, H.M. Hang, D.L. Schilling. *An efficient block-matching algorithm for motion compensated coding*. Proc. IEEE ICASSP. 1987. pp. 25.4.1-25.4.4.

24. M. Ghanbari. *The Cross-Search Algorithm for motion estimation*. IEEE Trans. Commun. July 1990. Vol. 38. No.7. pp. 950-953.

25. G. Tu, L. Cleymans, L. Van Eycken, A. Oosterlinck. *Low bit-rate coding based on evaluation of motion sequence*. SPIE Applications of Digital Image Processing XII. 1989. pp.40-47.

26. N. Mukawa, H. Kuroda. *Uncovered Background Prediction in Interframe Coding*. IEEE Trans. on Comm. COM-33. Nov. 1985. pp. 1227-1231.

27. E. Viscito, C.A. Gonzales. *Encoding of motion video sequences for the MPEG environment using arithmetic coding*. SPIE Visual Comm. and Image Proc. 1990. pp. 1572-1576.

28. P. Willemin, T. Reed, M. Kunt. *Image Coding by Split and Merge*. IEEE Trans. on Commun. Vol. 39. No 12. December 1991. pp. 1852-1854.

29. K. Hadley. *Kodak SV9600 Still Video Transreceiver*. SPIE. Vol. 1071. 1989. pp. 238-245.

30. G.G. Langdon, J.J. Rissanen. *Compression of black-white images with arithmetic coding*. IEEE Trans. Comm. June 1981. Vol. COM-29, no. 6, pp. 858-867.

31. E. Kawaguchi, T. Endo. *On a method of binary-picture representation and its application to data compression.* IEEE Trans. Machine Intel. Patt. Anal., January 1980. Vol. PAMI-2, No.1. pp. 27-35

32. E. Kawaguchi, T. Endo, J. Matsunaga. *Depth-first expression viewed from digital picture processing.* IEEE Trans. Machine Intel. Patt. Anal., July 1983. Vol. PAMI-5, No.4. pp. 373-384.

33. J. Ziv, A. Lempel. *A universal algorithm for sequential data compression.* IEEE Trans. Information Theory. May 1989. Vol. IT-23. No.3. pp. 337-343.

34. X. Wu, Y. Fang. *Lossless Interframe Compression of Medical Images.* Proceedings of the Data Compression Conference. March, 1992. pp. 249-253.

35. G. Langdon, A. Gulati, E. Seiler. *On the JPEG Model for Lossless Image Compression.* Proceedings of the Data Compression Conference. March, 1992. pp. 172-180

36. J. Rissanen, G. Langdon. *Universal modeling and coding.* IEEE Trans. Info. Theory, January, 1981. IT-27(1):pp. 12-23

37. G. Langdon, J. Rissanen. *Compression of black&white images with arithmetic coding.* IEEE Trans. Commun., June 1981 COM-29(6):pp. 858-867

38. S. Todd, G. Langdon, J. Rissanen. *Parameter reduction and context selection for compression of the gray-scale images.* IBM Jl. Res. & Develop., March 1985, 29(2):pp. 188-193

39. W.B. Pennebaker, J.L. Mitchell. *Standardization of color image compression.* I. Sequential coding. Electronic Imaging '89. October 1989

40. G. Wallece. *Overview of the JPEG still image compression standard.* Proc. SPIE. 1990. pp. 1244:220-233

41. W.B. Pennebaker, J.L. Mitchell. *EXACT - an algorithm for lossless coding of continuous tone images.* Working group document JPEG-186, ISO/IEC/JTC1/SC2/WG8. September, 1988.

42. O. Pahlm, P.O. Borjesson, O. Werner. *Compact digital storage ECGs. Computer Programs in Biomedicine.* 1979. pp. 9:293-300.

43. G. Langton. *Probabilistic and Q-Coder algorithms for binary source adaptation.* Data Compression Conference '91. April 1991. pp. 12-22.

44. Sijstermans, F., Van der Meer, J. *CD-I Full-Motion Video Encoding on a Parallel Computer.* Comm. of the ACM. April, 1991. pp. 82-91.

45. *C-Cube demonstrates MPEG Decoder Chip; chip supports base-level MPEG and JVC extensions*. Microprocessor Report. April, 1991. pp. 8.