Progressive Transmission of Digital Recurrent Video

by

Mr. Wai-Wa Wilson Chan

A thesis submitted in partial fulfilment
of the requirements for
the degree of MASTER OF SCIENCE

Department of Information Engineering
The Chinese University of Hong Kong

June 1992

Abstract

The problem considered in this thesis is the development of an algorithm for the transmission of grey scale digital recurrent video over a low bandwidth channel. The proposed solution is a loss-less progressive transmission algorithm that enable early viewing.

In the proposed algorithm, the video to be sent is encoded into a depth-spatial-temporal pyramid. Progressive transmission is obtained by first transmitting the lowest level with reduced-depth resolution, reduced-spatial resolution, and reduced-temporal resolution that is refined by further transmission until the original video is reconstructed. Two approaches to increasing the rate of video quality improvement of the intermediate levels are investigated. One is a flexible approach which enable the viewer to interactively bias progressive transmission towards depth-resolution, spatial-resolution and/or temporal-resolution. Another is to encode the depth-spatial-temporal pyramid according to a pre-defined normalized video quality.

The normalized video quality is defined as the weighted mean of a normalized mean which represent depth quality, a normalized variance which represent spatial quality and a normalized temporal quality measure. With the adjustment of the weighting factors of the normalized video quality, it is possible to bias progressive transmission towards depth-resolution or spatial-resolution. Further extension of these two approaches is suggested to be a formal subjective test so as to find the appropriate bias in resolution.

A computer program was written to simulate progressive transmission of video at 9600 bps and 64 kbps using the flexible or the pre-defined approach. Representative experimental results are given, demonstrating the high performance that can be achieved.

Preface

Though progressive transmission research has been dominated by spatial pyramid [1,2] for a number of years, a unified approach to the change of resolution [3] has been proposed more recently. The unified approach has demonstrated that, by defining a numerical measure for the comparison of still images, it is possible to encode an image into a depth-spatial pyramid which offer more efficient trade-off in quality and transmission time.

This thesis extends the unified approach to encode a special class of video, digital recurrent video, into a depth-spatial-temporal pyramid which is suitable for progressive transmission over low bandwidth channel. A simpler measure for the comparison of video is defined.

Related discussions are rare. It is thus intended to present a one-volume treatment of the essentials of progressive transmission of digital recurrent video with computer simulation results. The materials of this thesis are divided into 5 chapters.

Chapter 1 is introduction. It discusses the problem of transmitting digital recurrent video over low bandwidth channel and defines the scope of this thesis. Some of the possible solutions in relevant research are briefly outlined. The objective is the development of a loss-less progressive algorithm for the transmission of grey scale digital recurrent video over a low bandwidth channel.

Chapter 2 is theory. Firstly, it introduces the basic concepts of depth resolution, spatial resolution and temporal resolution in digital video to make subsequent analysis meaningful. Secondly, it defines numerical scales for normalized depth quality, normalized spatial quality, normalized temporal quality and normalized video quality so that comparison of video quality can be made.

Depth pyramid, spatial pyramid and temporal pyramid will form the basis of a depth-spatial-temporal pyramid which is suitable for progressive transmission over low bandwidth channel. Two approaches to encode the pyramid are investigated. One is a flexible approach which allow the to viewer interactively bias progressive transmission towards depth resolution, spatial resolution or temporal resolution. Another is a pre-defined approach which encode the pyramid according to the normalized video quality.

Chapter 3 reports some experimental results. A computer program was written to simulate progressive transmission of video at 9600 bps and 64 kbps using the flexible or pre-defined approach. Subjective results are illustrated with photographs whereas objective measurement on normalized video quality is analyzed.

Chapter 4 is conclusions and discussions.

In chapter 5, relevant references are included which may be useful for later year thesis students. Listing of the computer program for simulation is also attached.

The writer is grateful to Dr. Ng Wai-Yin, Mr. Vong Wai-kin and Ms. Maxi Hui for their kind assistance throughout this year. The writer is also grateful to Mr. Li Lung-Ming for his help in deriving the subjective video quality. Special thanks are due to my parents for their supports with all educational years.

Content

Chapter 1

Introduction

A picture is worth a thousand words and so the browsing and transmission of images are important and worth investigating. For example, much of the medical information in hospitals is in the form of images, 3-D object or video. If the images can be browsed and transmitted easily, more time can be saved.

Video browsing is a system for retrieving computer based image information. The viewer uses a simple software interface to select images of interest and obtain additional information about them. The software interface offers the viewer flexible control. The viewer can browse a video segment, searching backward and forward to switch to a particular image at the flick of a button. In this way, the viewer can quickly and easily locate their desired images in the sea of information. The viewer is better off because only the cost for transmitting relevant images is to be paid and time is not wasted to transmit irrelevant images. Therefore, flexible transmission is the primarily concern in this kind of application.

The concept of 3-D fax is different from video browsing. It is a system for transmitting 3-D objects from one place to another. Catalog shopping is application of the 3-D fax concept. In 1988, Bellcore had demonstrated an experimental prototype in which an electronic clothes catalog was displayed at 1 frame per second [4]. The viewer can stop the display at any point and select his interested cloth to take a closer look. When the choice is made, an electronic order form is ready to send.

Although the concept of video browsing and 3-D fax are attractive, they cost viewer a lot to transmit. If the viewer wish to receive continuous ideal images which has much details, the cost may be prohibitively expensive even though it may be technologically possible. In the context of present day technology, the higher the resolution would require the longer the transmission time. Very expensive equipment and very large bandwidth transmission media are required to reduce the unacceptable long transmission time.

If, however, the viewer is willing to receive less ideal images, significant cost savings is possible. This is because the viewer is very sensitive to long transmission time while small transmission error can usually be tolerable. For this reason, many research on images transmission has aimed at finding a suitable compromise between quality and bandwidth. Therefore, more efficient coding is also the concern for the above application.

Transmission of images to some viewers is now possible via many small scale private networks which have been installed world wide. These viewers can afford the cost of the private network and are not the general public. Their private networks exist in isolation and the sharing of information among them is not easy.

It would be more desirable if the transmission of images is also for the general public. It will then be easy for everyone to have immediate access to images databases and economic of scale will apply.

The well established Public Service Telephone Network (PSTN) is a candidate for economical world-wide networking. The PSTN's bandwidth is, however, very limited. Even with a good 9600 bps modem, it usually takes a very long time to transmit an image. In narrow-band Integrated Service Digital Network (ISDN) being implemented with PSTN, the transmission of 50 frames small image size of 256 x 256 with 8 bit grey scale at 64 kbps still take more than 400 seconds.

It is within the context of using low bandwidth channel with range of throughput from 9600 bps to 64 kbps that this thesis aim to explore an algorithm for the transmission of images to the public. The algorithm should be flexible enough to suit individual viewer's requirement. The algorithm also should be efficient so as to be transmitted over the low bandwidth channel. The public should be able to share, browse, and select images as easily as using their existing telephone.

1.1 Problem under study and scope

The problem considered in this thesis is the transmission of grey scale digital recurrent video over a low bandwidth channel.

This simple statement needs some explanations.

A video is a sequence of still images which contains information about the visible energy incident on a two-dimensional plane and the changes in visible energy as a function of time. A scene with moving objects is an example of video. Video can has different classes such as broadcast video and interactive video according to its transmission characteristic. Recurrent video is a special class of video such that the video repeated itself at the end of the sequence of images. The 3-D fax of 3-D objects such as a 3-D human head is an example of recurrent video. The recurrent video would look like a rotating human head in the example.

In digital recurrent video, the sequence of images is sampled in space and time and then quantized. The temporal sample in number of frames indicates the temporal resolution of the video. Too many temporal samples are unnecessary because the eye acts as a low-pass filter for the higher temporal samples. The spatial sample in number of pixel per row by number of pixel per column indicates the spatial resolution of the video. Too many spatial samples are also not necessary since the eye also act as a two-dimensional low-pass filter for the higher spatial samples. The number of bits per pixel indicates the depth resolution of the video. For example, the number of bits per pixel for grey scale image is usually 8 and that for colour image is usually 24.

Transmission can be classified into loss-less and lossy. Loss-less transmission means that the viewer receives an exact replica of the original grey scale digital recurrent video. Lossy transmission means the received images are distorted. Low bandwidth means data rate of 9600 bps or 64 kbps which is not prohibitively expensive to the public via PSTN or narrow-band ISDN.

The scope of this thesis is restricted to the transmission of grey scale digital recurrent video. Areas outside this scope such as the transmission of colour digital recurrent video is subject to further research.

1.2 Review of relevant research

If we take the bits in 50 frames of 256 x 256 8 bit/pixel video then we get 26214400 bits and takes 2731 seconds to transmit at data rate of 9600 bps. This tells us something about the problem of video transmission in low bandwidth channel.

In a networking environment, the problem is more severe. There will be transmission overhead such as error-correction code added to the image data file. There also may be transmission contention and processor overload problem. That means throughput will be much lower than the available bandwidth. For example, the throughput of Ethernet with 10 Mbps available bandwidth is usually lower than 1 Mbps.

One of the two approaches is usually taken to tackle the above problem. One approach is data compression and another approach is progressive transmission.

1.2.1 Data Compression as a solution

An impressive number of data compression schemes and very sophisticated compression encoder [5] has already been developed so far. The concept of data compression is to reduce the redundance in images and to take advantage of the human vision property.

Discrete Cosine Transform (DCT) is a well-established data compression algorithm to reach lossy compression ratio as high as 10:1 and still retain excellent image quality. In the Joint Photographic Experts Group (JPEG) standard, an image is first divided into 8 x 8 pixel non-overlapping blocks. Each block is then transformed from

the time domain into the frequency domain resulting in some frequency coefficients. Since a few frequency coefficients has already contained the major part of image information so that some frequency coefficients can be quantized with fewer bits than others. Huffman coding which uses pre-determined variable-length code words is then applied to encode the coefficients for transmission.

Full frame Cosine Transform coder is demonstrated recently [6]. By combining DCT with some suggested threshold scheme and quantization procedure, video bandwidth requirement can be as low as 2.5 Mbps.

To reduce the bit-rate further, motion estimation is usually combined with DCT. A motion vector is the offset between the current image block and the prior image block that form a best match. The motion vector is transmitted to indicate the location of the predicted block so the transmission of the block is not required. In the MPEG standard, PAL quality at 1.5 Mbps has been achieved.

The main disadvantages of DCT to solve our problem are that much higher compression ratio is required and that DCT requires numerous multiplications and additions operations. Implementation of DCT usually needs quite a few printed circuit boards and special design digital signal processor and associated hardware circuitry.

Another data compression algorithm is to sub-sample the images and interpolating the missing samples at the receiver. MUSE (Multiple Sub-Nyquist Sampling Encoding) is the most well known uniform sub-sampling system currently used in Japan for High Definition Television (HDTV) broadcasting [7]. It operates at around 100-160 Mbps. The disadvantage of MUSE to solve our problem is its low compression ratio.

There are many other data compression algorithms. Generally, high compression ratios are lossy and the received images are not an exact replica of the original. The compression ratios of loss-less data compression algorithm compression ratios are usually no more than 3:1 which is too low to solve our problem.

As a matter of fact, data compression is necessary to reduce the required transmission bandwidth. There is a limit, however, to how far it can go.

Can we tackle the problem without data compression? The answer is yes. Progressive transmission is an alternative to data compression for the delivery of images over low bandwidth channel.

In progressive transmission, low quality images are sent first to give immediate recognition to the viewer. The images are then refined progressively by further transmission until an exact replica of the original images is obtained. The viewer is free to abort the transmission as soon as the image quality is sufficient good to the viewer. Even if the transmitter or the transmission media is destroyed during the transmission, the best possible images still remain.

1.2.2 Progressive Transmission as a solution

Lots of algorithm had been proposed for progressive transmission. In 1980, Knowlton had described a simple algorithm for loss-less transmission of grey scale images over low bandwidth channel [8]. In his algorithm, every 2 pixels in an image are combined to form a composite value and a difference value. The composite value is the average value with look-up table adjustment to eliminate round-off error. A hierarchical structure is recursively defined on the resulting composite values until there is only one composite value representing the entire image. Progressive transmission is obtained by sending the composite value followed by the successive difference values which enable progressive finer reconstruction of the original image. The main advantage of Knowlton's algorithm is simple and no extra bits are required to represent the redundant pyramid data structure of the original images.

In 1983, Burt had proposed another progressive transmission algorithm [9]. The algorithm first subtracts a low-pass filtered copy of the image from the original image. The difference image will have low entropy and the low-pass filtered image can be coarsely sampled. The same step then repeated to the low-pass filtered image until a desired pyramid data structure is obtained. Progressive transmission is obtained by sending from top to the bottom of the pyramid. This algorithm is lossy and some degree of data compression is achieved.

Although there are many research on progressive transmission for many years, the current status in progressive transmission research has concentrated only on exploring spatial resolution of an image. Its application is successful in the transmission of still images over low bandwidth channel.

Progressive transmission of digital recurrent video is also possible. We will describe a very versatile multi-resolution data structure to represent the digital recurrent video. Depth resolution and temporal resolution are explored in addition to spatial resolution. We also will explore what makes a progressive transmission algorithm good. A numerical quality measure of video will be defined.

1.3 Objectives

The objective of this thesis is the development of a loss-less progressive transmission algorithm for the transmission of digital recurrent video over low bandwidth channel.

Specific objectives are:

1.3.1 To devise a flexible mutil-resolution data structure to represent digital recurrent video so as to enable efficient progressive transmission.

1.3.2 To devise numerical scales of quality for depth resolution, spatial resolution, temporal resolution and the overall video resolution so that comparison of algorithm can be based.

1.3.3 To develop flexible and efficient progressive algorithms so as to closely match the requirement of the video and the viewer.

1.3.4 To write a computer program to simulate progressive transmission of digital recurrent video at 9600 bps and 64 kbps.

Chapter 2

Theory

The first step in the theory is to represent the resolution of video in a convenient way. Although there might be many representations, depth resolution, spatial resolution, and temporal resolution are assumed to define sufficiently the overall resolution of a video. In this way, the resolution of a video is represented by a co-ordinate (d,s,t) in 3-dimensional space. For simplicity, we also will assume a square image in which the number of rows equals the number of columns.

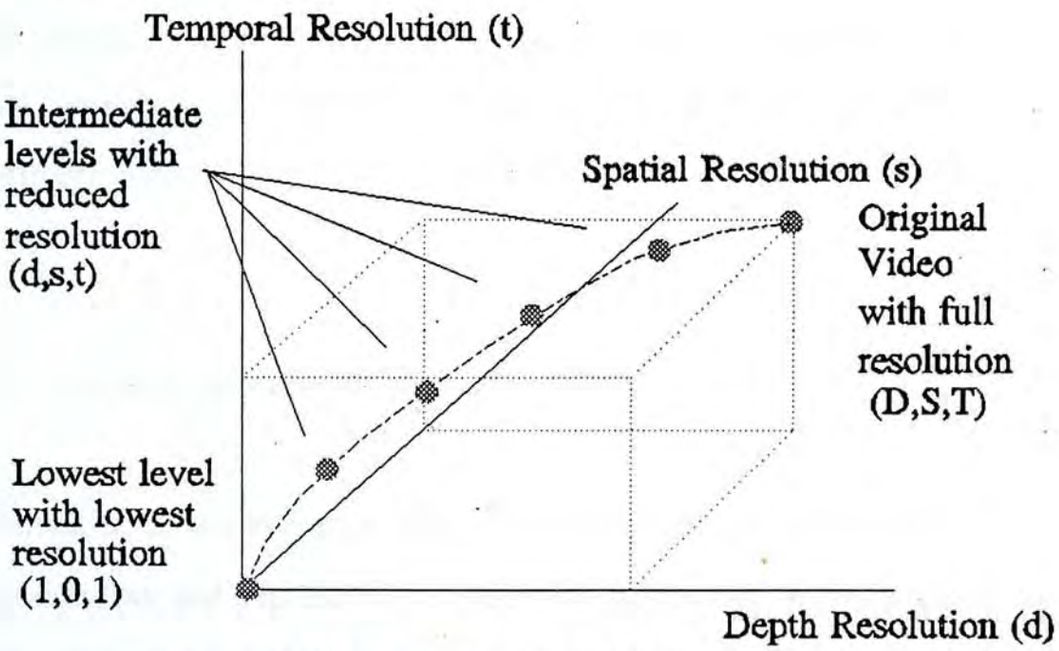


Figure [1] Representation of Multi-resolution Video

2.1 Multi-resolution representation of digital video

2.1.1 Depth Resolution (d bit/pixel)

Each image frame is consisted of pixel. Number of bits that used to represent a pixel is the depth resolution. It is the response of the eye to radiant energy. The full depth resolution in grey scale digital video in our experiment is 8 bit/pixel so that d ranges from 1 to 8.

2.1.2 Spatial Resolution (2^s pixel/column for square image)

The number of pixel in a row multiplies the number of pixel in a column is defined as spatial resolution. Since we assume a square image and the size of the image is 2 to the power of s for some integer s for simplicity, we simplify the definition of spatial resolution to s which is dimension-less. The size of image used in our experiment is 256 x 256. The full spatial resolution is therefore 8 and s ranges from 0 to 8.

2.1.3 Temporal Resolution (t frame)

The number of frames in a video is defined as the temporal resolution. It is dimension-less and represents the response of the eye to temporal patterns. The number of frames of video used in our experiment is 50. The full temporal resolution is therefore 50 and t ranges from 1 to 50.

It is tempting to define the temporal resolution in number of frames per second. This definition is not suitable for progressive transmission because the frame rate for progressive display is an independent quantity and is usually maintained to be constant for video with reduced number of frames. For example, if the original 50 frames video is displayed at 10 frames per second for 5 seconds, a reduced 20 frames video will usually be displayed at the same 10 frames per second for 2 seconds.

To sum up, resolution of a video is represented by a co-ordinate (d,s,t) in a 3-dimensional domain and progressive transmission is obtained by traversing from (1,0,1) through a sequence of coordinates (d,s,t) to the final coordinate (D,S,T) of the original video. The final co-ordinate used in our experiment is (8,8,50).

2.2 Performance measure for progressive algorithm

The term "progressive transmission" is sometimes restricted. To many people, "progressive transmission" is restricted to the refinement of spatial resolution over time. To some people, "progressive transmission" means refinement of colour. To generalize the meaning, progressive transmission of digital recurrent video is the combination of refinement in depth resolution, spatial resolution and temporal resolution which associate with the viewer's inherent visual property.

Based on the multi-resolution representation of digital video in (d,s,t) coordinates, there are sequences traversing from (1,0,1) through some intermediate (d,s,t) to the final co-ordinate (D,S,T) representing the resolution of the original video.

The design of a progressive transmission algorithm is equivalent to the design of a flexible and efficient trajectory of (d,s,t) co-ordinates so as to meet viewer requirement and bandwidth constraints. Any particular progressive traversal is a trajectory from (1,0,1) to (D,S,T) with non-decreasing values of the 3 co-ordinates. By flexible, we mean the viewer can interactivel^Xxadjust to sequence. By efficient, we mean early recognition by the viewer.

To evaluate the quality of early recognition, it is desirable if there is a numerical scale for the comparison of video quality for difference sequences. We can then adjust the sequences of intermediate (d,s,t) to maximize the quality scale in earliest time.

2.2.1 A performance model for progressive transmission

To recap that one of the goal of progressive transmission is early recognition of the images. This is true only if the early (d,s,t) co-ordinates allow quick recognition of the images. We will develop a class of numerical scales of quality and adjust our algorithm so as to maximize the numerical quality. By selecting a particular numerical quality, progressive transmission can be made to bias towards depth resolution, spatial resolution and temporal resolution. This will form the basis of a pre-defined approach in progressive transmission.

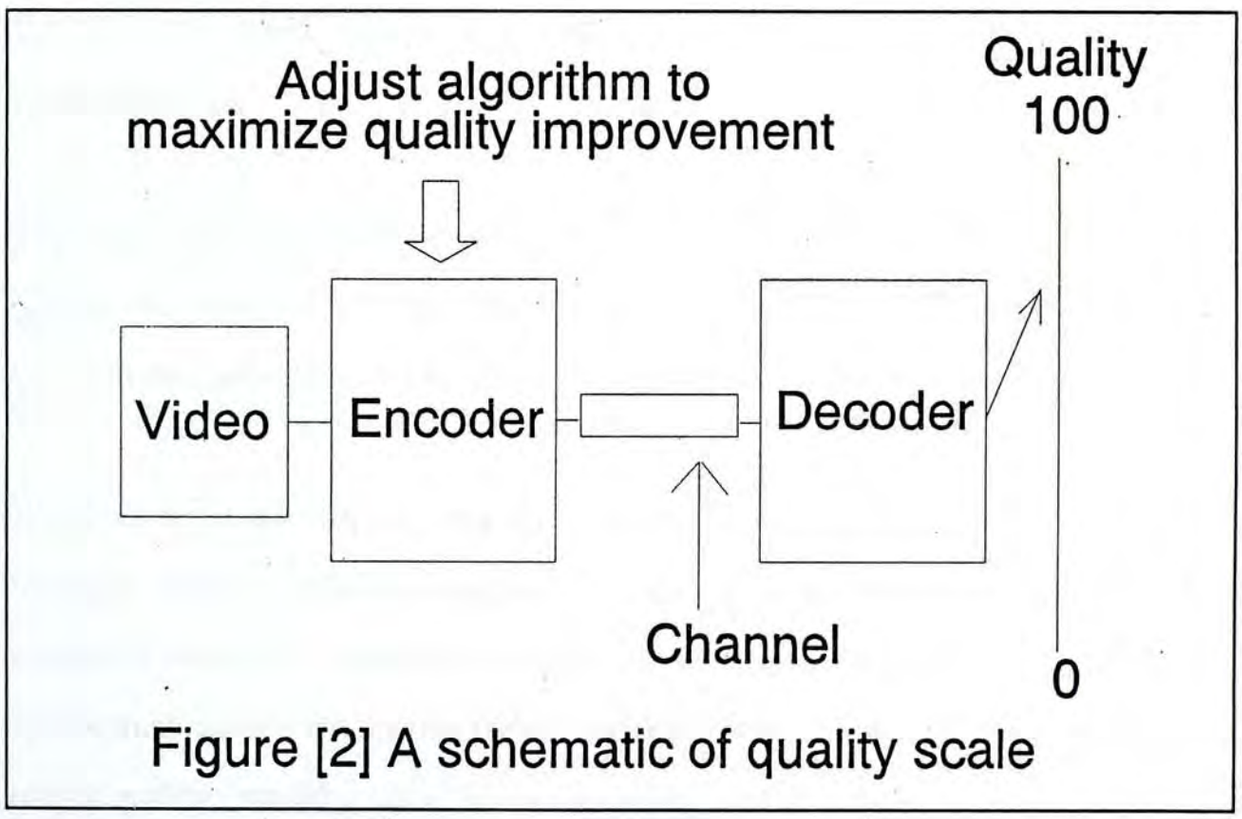


Figure [2] A schematic of quality scale

At the starting co-ordinate (1,0,1), the recognition of the images to the viewer is 'annoying'. As the co-ordinate (d,s,t) traverse towards the final co-ordinate (D,S,T), the overall resolution to the viewer increases. A co-ordinate is finally encountered where the viewer sees 'perceptible' recognition. This co-ordinate is the lower threshold of the overall resolution of the particular images.

When the (d,s,t) co-ordinate traverses through the lower threshold, visual quality to the viewer continues to improve. In fact, there is often an upper threshold co-ordinate where the viewer sees 'imperceptible' improvement on recognition even though the overall resolution is progressively increased. This upper threshold represents the upper boundary in the capability of the viewer's eye for the particular images. Usually, the upper threshold is seldom reached in progressive transmission applications.

The range of (d,s,t) co-ordinates between the lower threshold and the upper threshold defines our range of interest. If there is a lack of quality in the intermediate (d,s,t) co-ordinates, application of progressive transmission will be limited.

We need a performance model for progressive transmission. The importance of having a model in relation to algorithm design is the possibility of carrying out the algorithm design in "perceptual space". To transform the (d,s,t) co-ordinates into "perceptual space", we assume the existence of 3 variables (M,V,F) which represent (depth quality, spatial quality, temporal quality) to the viewer.

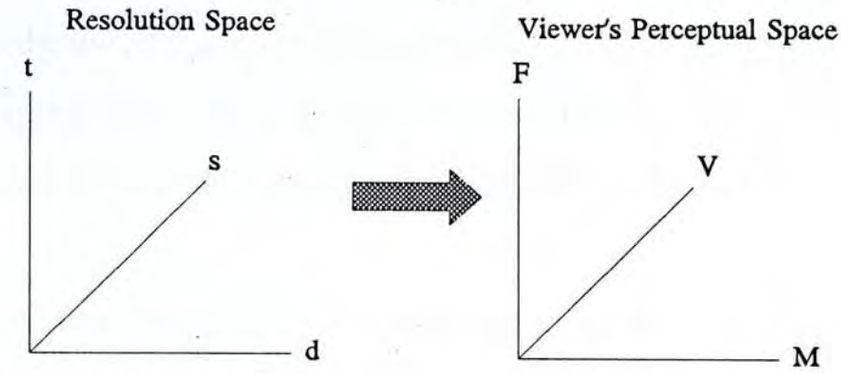


Figure [3] Transformation to perceptual space

An ideal progressive algorithm looks like a step function in the (M,V,F) space and gives 100 percent quality improvement in the first reduced resolution video. The worst progressive algorithm is the same as a non-progressive algorithm which gives a step 100 percent quality improvement only at the end of transmission. An efficient progressive algorithm is the one which approximates the ideal progressive algorithm.

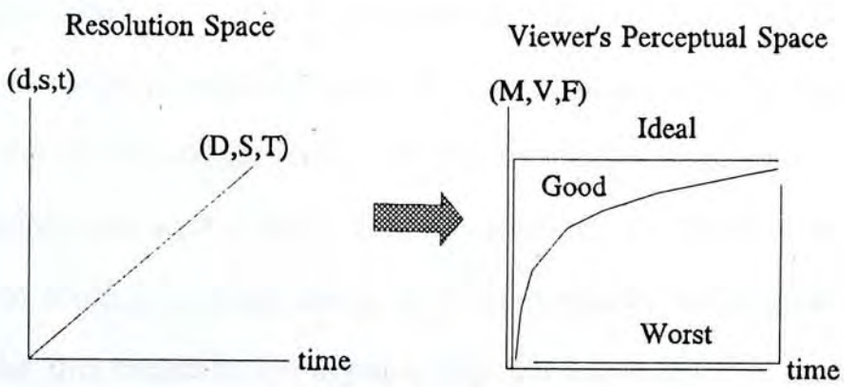


Figure [4] Performance in perceptual space

Defining a numerical video quality is by itself an extremely difficult problem remains to be solved. We do not claim that we had solved this problem in this thesis. We only indicate that the use of some justified numerical quality is an approach for obtaining an efficient (d,s,t) sequences in progressive transmission. We wish this approach is useful in laying the foundation for comparing different algorithms.

Learning from the proliferation of progressive transmission algorithm in many research papers, we see that the style of using heuristics to cause the best visual quality of the original images within the available bandwidth. Although heuristics do not guarantee an optimum algorithm and may have constraints on its application, it reduces the effort needed to obtain a nearly optimum algorithm and is sufficient for some restricted application. In that way, our experience gets encoded and can be translated into implementation.

If we have two different cartoon-like images with similar background and only differ greatly near edges, a very small normalized root mean square error would be claimed even though the two images are very different in "perceptual space".

Despite of the above fact, it is still generally accepted that normalized root mean square error between a reduced resolution image and the original image is good for the comparison of still images. And the goal of many researches is to minimize the normalized root mean square error. This heuristic assumes that the value of the pixel in the reduced resolution image should be close to that of the original image. One of the reason for this emphasis on depth-spatial correspondence is that it is a source measurable parameter in digital image.

In our study, the key to recognition is a function of depth resolution, spatial resolution, temporal resolution and the video context. If only a low resolution image is displayed to the viewer, it is not necessary that the root mean square error will be subjectively recognized. It is also not necessary that the reduced depth resolution and the reduced spatial resolution contribute equally to the normal root mean square error. As normalized root mean square error is a single quality measure on depth-spatial correspondence, it cannot be made to bias towards depth resolution or spatial resolution. Therefore it is not suitable in our study.

In our study, we need a class of quality measure on the combination of depth-quality, spatial-quality, and temporal-quality so that we can bias progressive transmission towards depth resolution, spatial resolution or temporal resolution as required. Therefore, we use 3 variables depth quality (M), spatial quality (V) and Temporal quality (F) instead of a single normalized root mean square error. The viewer's "perceptual space" is then represented by 3 variables co-ordinate (M,V,F) rather than a fixed depth-spatial correspondence as in the normalized root mean square error.

By representing the viewer's "perceptual space", we imply a correspondence found between the images and a prior representation of images in the mind of the viewer. The matching will depend on many factors. Obvious factors are depth resolution, spatial resolution and temporal resolution. However, the quality of images depends on other factors as well. Lighting conditions, graphic display adjustments, the kinds of images, the amount of motion in the images, the experience and expectation of the viewer are some of the other important factors.

Figure [5] Matching in the perceptual space

The importance of the experience and expectation of the viewer can be illustrated in an old children story about three men. These three men are holding in their hand a small piece of wood and let go of each piece of wood.

In the case of the first man the wood falls downwards.
In the case of the second man the wood goes upwards.
In the case of the third man the wood remains where just where it is.

The behaviour of the wood in the first case is normal and expected. The behaviour of the wood in the other two case is unbelievable by our limited experience and expectation.

The mystery is instantly solved when we explain that the three situations are different. In the first case the man is standing on the surface of the earth so the wood falls downwards in the expected manner. In the second case the second man is standing under water and in this situation the wood naturally floats upwards. In the third case the third man is in an orbiting space-craft so the weightless wood remains just where it is.

The reason we have not understood the mystery is that we are also limited by our experience and expectation. It is important for us to design the algorithm in the viewer's perceptual space which is related to expectation and experience.

2.2.2 Limitations of the performance model

Performance model is amongst the virtual perceptual space. In creating the model, we create an artificial formal domain thereby create a blindness to everything that is not expressible by depth resolution, spatial resolution and temporal resolution. For example, a traffic system would models the types of cars and how long it takes to across the intersection. It will not model the mood of the driver. Modelling ignores things so we can cope.

An algorithm can be claimed to work in the model. It does not say that the model is the viewer's perceptual space. Inconsistency of the viewer's perceptual space poses a severe problem for modelling. A program with a single bug that makes it inconsistent is as meaningless and incoherent as gibberish. However, there is no graded notion of coherence in the concept of perceptual space. If the perceptual space of the viewer is inconsistent, then no model is suitable.

Because what is known is so small compared with what is not known that good model is not the only ones which fill the gaps in progressive transmission research. Model is situational. In other words the effectiveness is dependent on viewer and application. If the model is built with the help of the viewer, then it follows that the model will be able to avail itself of the subject matter in carrying out its symbol manipulation or reasoning. We need the flexibility to change the model parameters through interactive communication with the viewer.

Modelling has also been shown to be fallible. For example in the past, earth was said to be square and sun was revolving around the earth. So we need empirical study to test the generalisation of the model. One of the way is again to let viewer to have interactive communication with the computer.

Modelling has a cost. It would be hard to argue that modelling is not powerful, but it carries a price of potentially unwarranted definiteness, premature categorization, and resistance to empirical approach.

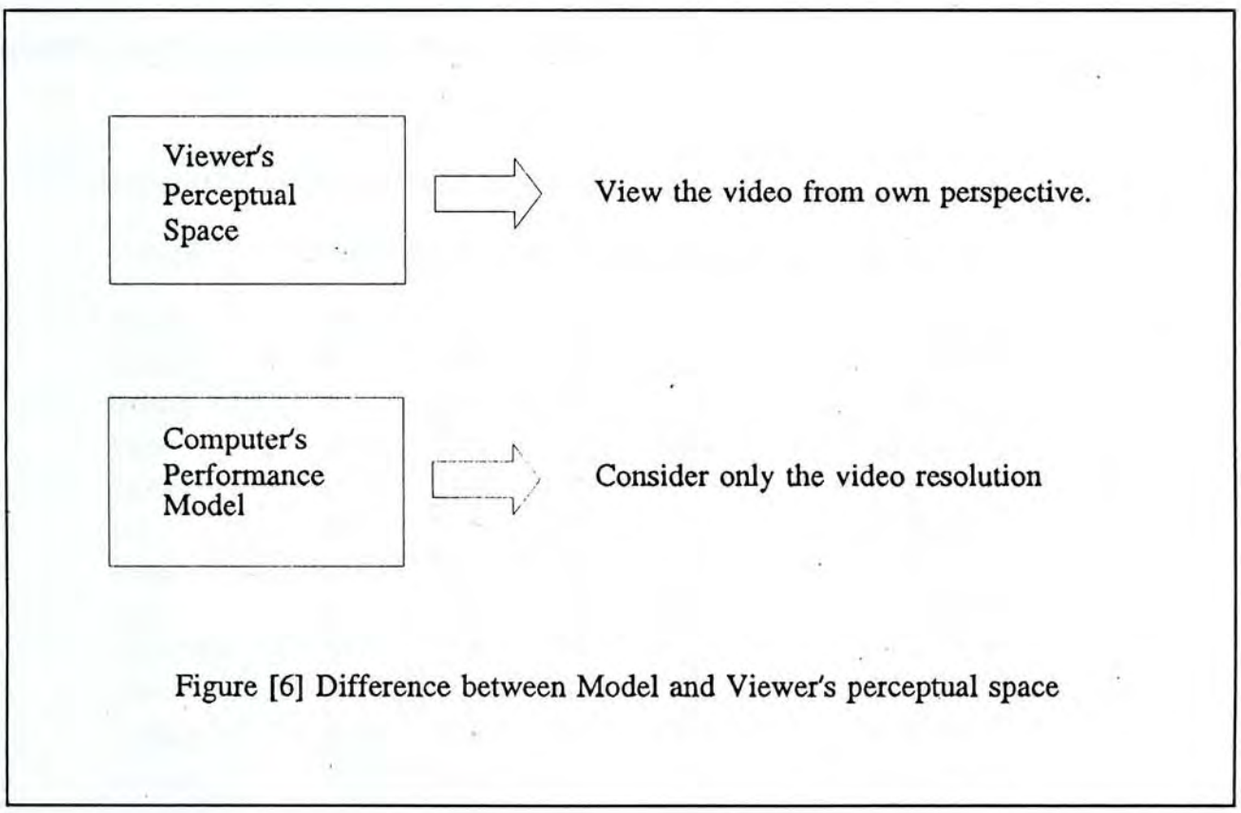


Figure [6] Difference between Model and Viewer's perceptual space

Because the assessment is so variable and we know that the basic of visual quality is subjective, our approach is to make initial judgement from the statistics of about 17 images, generate a nearly right numerical quality scale and perform a formal subjective test later. It will be the formal subjective test to fine tune the initial judgement.

By varying the combination of depth resolution and spatial resolution of the above 17 images, initial judgement finds no relevance of entropy to the quality of image. However, two postulates are proposed for the relations between mean and depth quality and that between variance and spatial quality.

The images used in the initial judgement are listed below:

Image	Depth	Size	Mean	Variance	Entropy
ram1	8	256	170	4457	7.250187
tiffany	8	256	209	961	6.569950
urban	8	256	152	3766	6.936275
face	8	256	113	4924	6.550231
earth	8	256	119	5876	5.545727
jet	8	256	175	465	5.366972
lena	8	256	99	2767	7.598966
tank	8	256	178	4454	5.320782
building	8	256	186	4334	4.412987
girl	8	256	73	1816	6.414508
baboon	8	256	128	1909	7.367599
bridge	8	256	113	2820	7.668557
camera	8	256	118	3886	7.009717
couple	8	256	40	1222	6.220064
diane	8	256	152	5260	7.367846
moffet	8	256	155	1640	7.198854
pepper	8	256	115	5517	7.559621

2.2.2 Normalized mean (M) to represent depth quality

A reduced depth resolution image is obtained according to a bit-plane algorithm described in chapter 2.3. Based on that algorithm, the mean is monotonic increasing with depth-resolution irrespective of the spatial resolution. The magnitude of mean also seems consistent with initial subjective judgement. Therefore, we postulate a value M to represent the depth quality as illustrated in the statistics of lena.hips.

Let m_o be the mean of the original image and m_i be the mean of the reduced-depth resolution image

$$\text{Normalized mean } M = 100 \times (m_o - \text{abs}(m_o - m_i))/m_o$$

Please note that the value M is monotonic increasing with depth-resolution irrespective of the spatial resolution in the reduced depth resolution image. Figure 7 is the subjective judgement and the author and the guest viewer.

Image	Depth	Size	mean	variance	entropy	M
lena.hips	1	256	39	3472	0.887236	39
lena.hips	2	256	69	3098	1.781664	70
lena.hips	3	256	83	2923	2.698812	84
lena.hips	4	256	91	2781	3.627758	92
lena.hips	5	256	95	2772	4.610365	96
lena.hips	6	256	97	2768	5.604208	98
lena.hips	7	256	98	2768	6.600924	99
lena.hips	8	256	99	2767	7.598966	100

Depth Quality versus Depth Resolution

Test Image: lena.hips

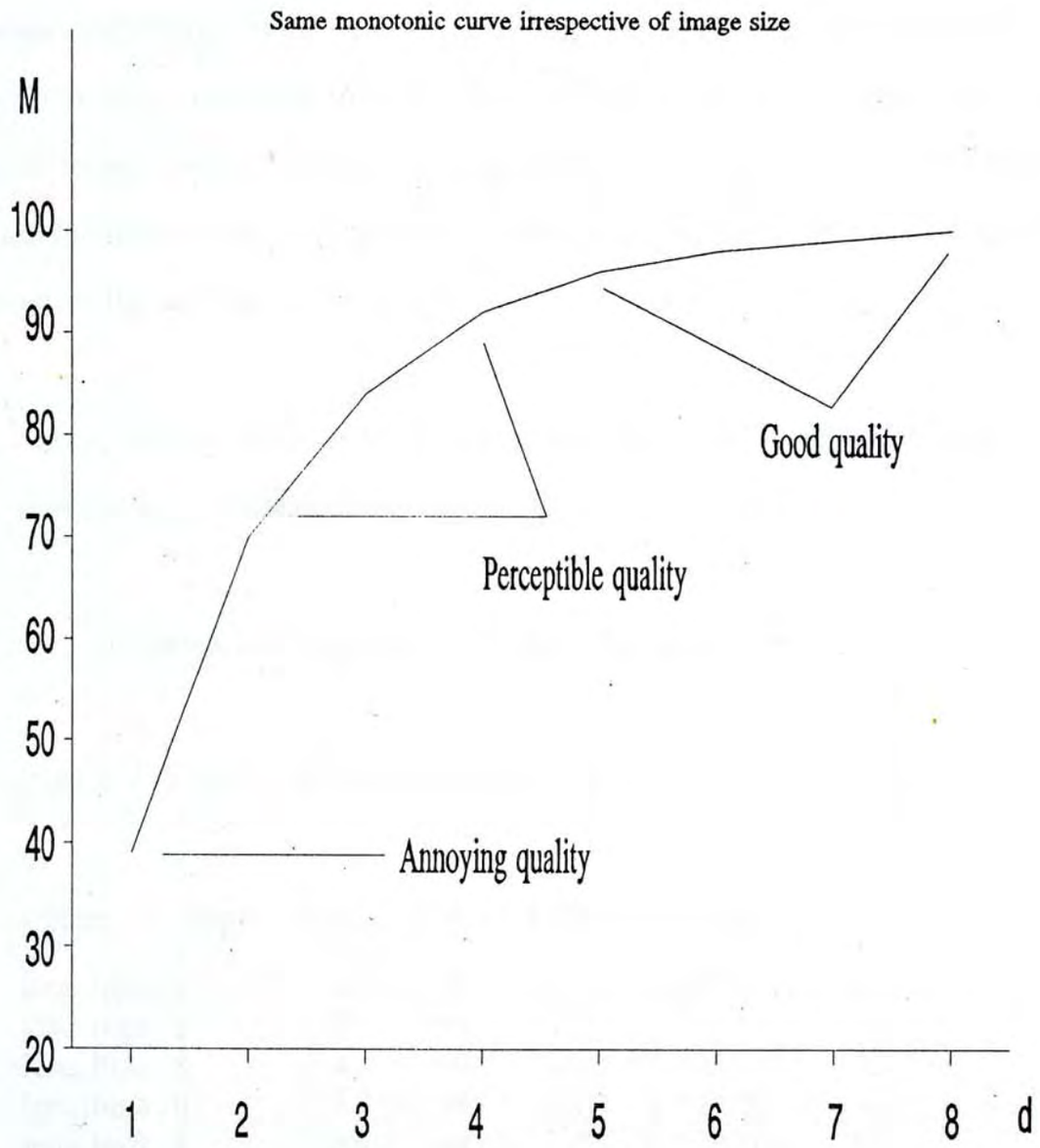


Figure [7]

2.2.3 Normalized variance (V) to represent spatial quality

A reduced spatial resolution image is obtained according to the reduced-difference algorithm described in chapter 2.4. Based on that algorithm, the variance is monotonic decreasing with spatial-resolution for a fixed depth resolution. The magnitude of variance also seems consistent with initial subjective judgement. A sharp change in image content results in a sharp change in variance. This is the underlying syntactic relation for us to postulate a value V to represent the spatial quality as illustrated in the statistics of lena.hips.

Let v_o be the variance of the original image and v_i be the variance of the reduced-spatial resolution image

$$\text{Normalized variance } V = 100 \times (v_o - \text{abs}(v_o - v_i))/v_o$$

Figure 7 is the subjective judgement and the author and the guest viewer.

Image	Depth	Size	mean	variance	entropy	V
lena.hips	8	1	99	0	0.000000	0
lena.hips	8	2	99	312	1.500000	11
lena.hips	8	4	99	570	3.875000	21
lena.hips	8	8	99	1169	5.593750	42
lena.hips	8	16	99	1753	6.758099	63
lena.hips	8	32	99	2136	7.237924	77
lena.hips	8	64	99	2442	7.448914	88
lena.hips	8	128	99	2637	7.535016	95
lena.hips	8	256	99	2767	7.598966	100

Spatial Quality versus Spatial Resolution

Test Image: lena.hips

Montonic curve for fixed depth resolution

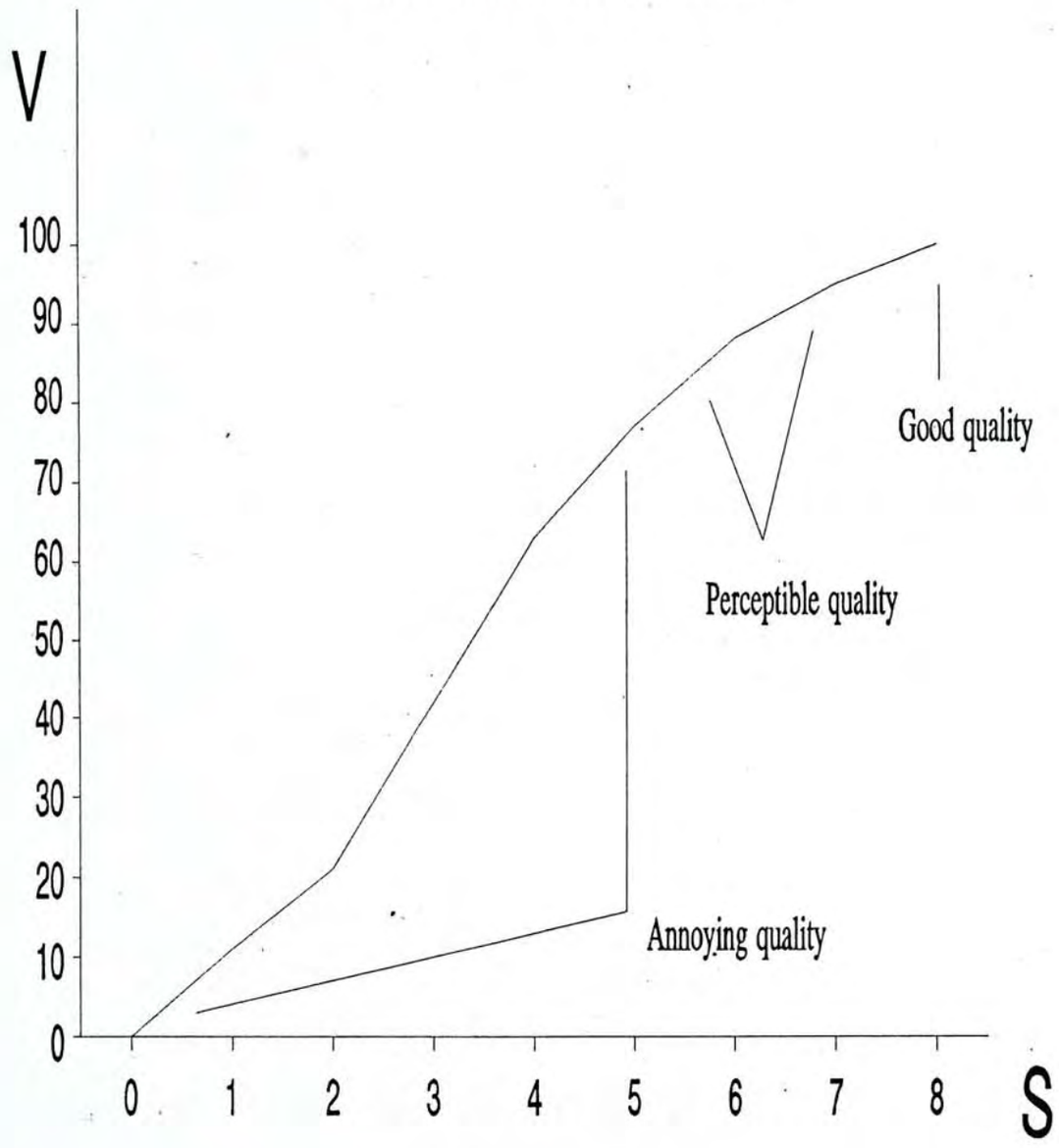


Figure [8]

Please note that the value V is monotonic decreasing with spatial-resolution for a fixed depth resolution in the reduced spatial resolution image. For the comparison of spatial quality of images with different depth resolutions, the monotonic decreasing property cannot be expected.

2.2.4 Temporal Resolution to represent temporal quality (F)

A reduced temporal resolution video is obtained by skipping frames which is described in chapter 2.4.

In digital recurrent video, image frames close together are usually similar. If the viewer's requirement is to see dis-similar images as soon as possible, then the temporal quality of reduced temporal resolution video formed by nearby frames contains less temporal quality than that formed by distant frames. Suppose an original video has temporal resolution of 50. This means the video has 50 frames in our assumption. On one hand, we select frame 1 and frame 2 to form a video A. The reduced temporal resolution is 2. One the other hand, we select frame 1 and frame 25 to form another video B. The reduced temporal resolution is also 2. The temporal quality of video B is higher than that of video A if the viewer's requirement is to browse the video.

Since the temporal quality depends a lot on the viewer's requirement, it would be reasonable to continue to use temporal resolution to represent temporal quality. That is, we postulate a value F to represent the temporal quality of video.

> Let the original video has frame n and the reduced-temporal resolution video has frame k.
>
> Normalized temporal quality (F) = k / n.

2.2.5 Normalized quality (Q) to represent the overall quality of digital recurrent video

Depth quality and spatial quality are only defined for a particular still image. The extension to digital recurrent video needs modification. Considering the fact that digital recurrent video usually contains similar images, the depth quality and spatial quality of the first frame are chosen to represent the depth quality and spatial quality of the digital recurrent video.

We assume that the overall quality is a function of depth quality, spatial quality, and temporal quality. For the sake of simplicity, we postulate a value Q to represent the overall quality of video.

Let the depth quality be M, the spatial quality be V, the temporal quality be F, and w_m, w_v, w_f be the weighting factor of M, V, F respectively.

Normalized quality $Q = (w_m M + w_v V)\, w_f F / 300$

where $(w_m + w_v)\, w_f = 1$.

For simplicity, we will assume $w_m = w_v = w_f = 1$.

Depth-bias quality can be obtained by increasing w_m.
Spatial-bias quality can be obtained by increasing w_v.
Temporal-bias quality can be obtained by increasing w_f.
Figure 9 shows the variations of Q as a function of the weighting factors. Figure 10 shows the variations of Q as a function of depth resolution and spatial resolution.

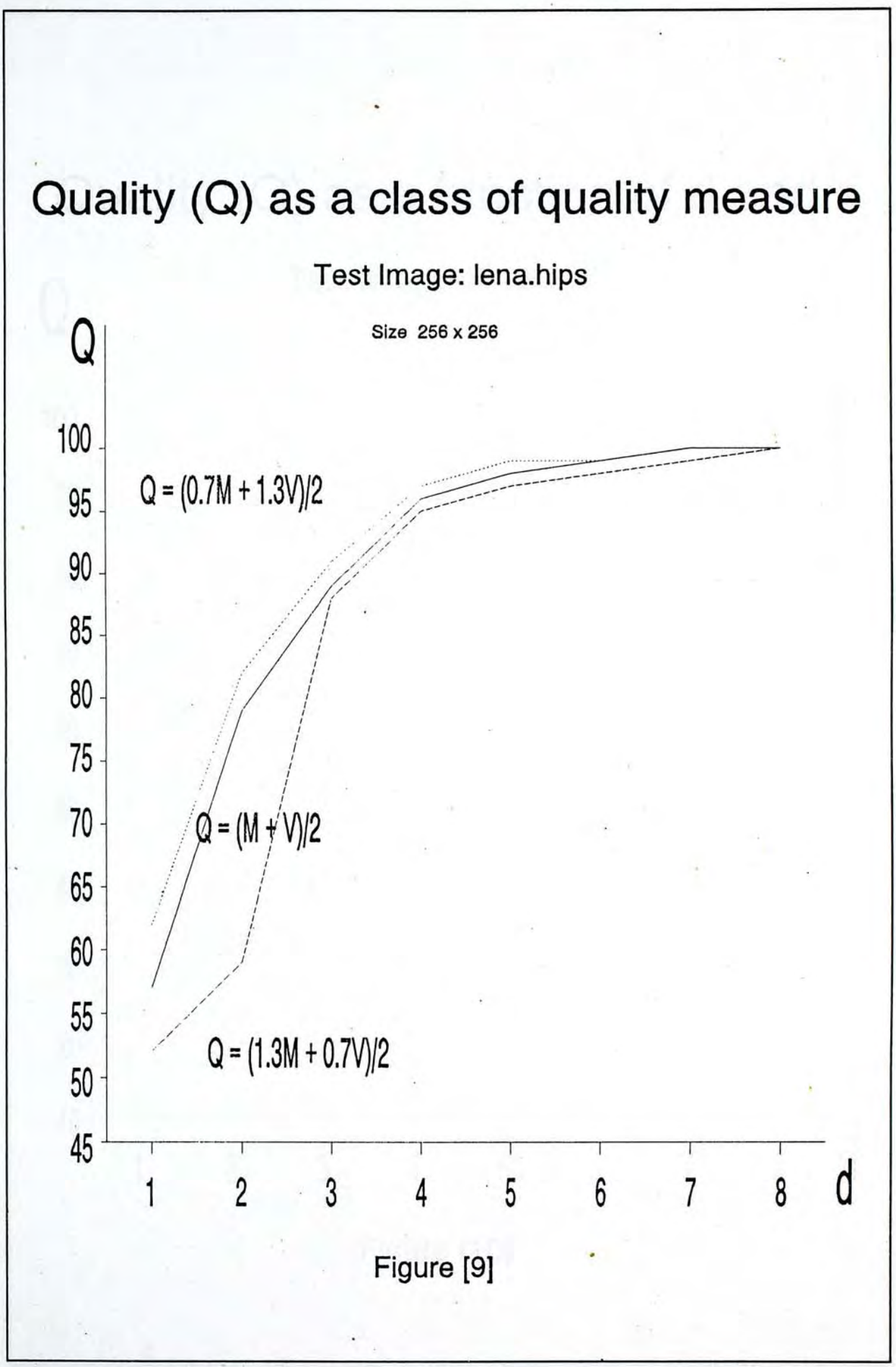

Quality (Q) as a class of quality measure
Test Image: lena.hips
Size 256 x 256
Q
100
95
90
85
80
75
70
65
60
55
50
45
Q = (0.7M + 1.3V)/2
Q = (M + V)/2
Q = (1.3M + 0.7V)/2
1
2
3
4
5
6
7
8
d

Figure [9]

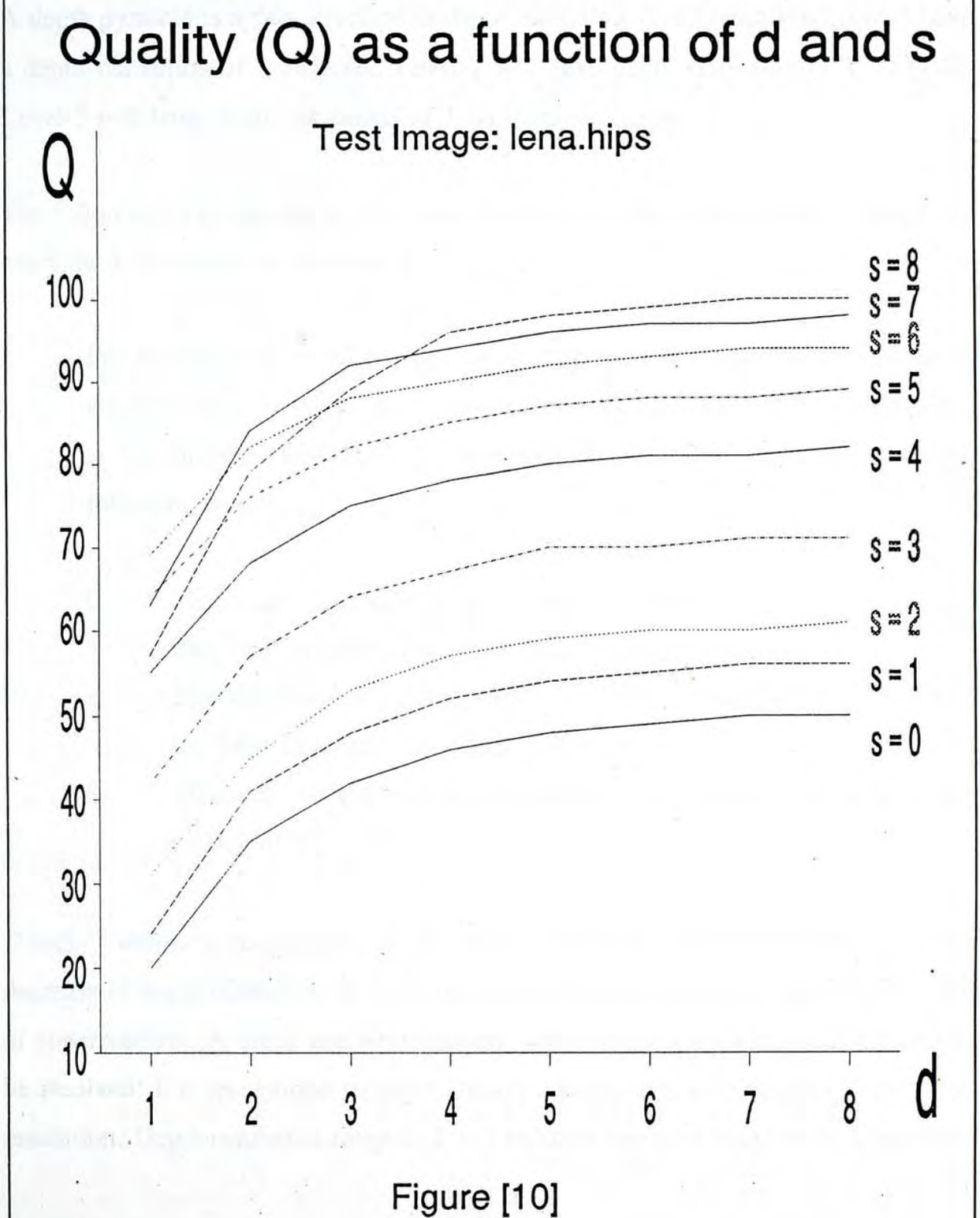

Quality (Q) as a function of d and s
Test Image: lena.hips
Q
100
90
80
70
60
50
40
30
20
10
1
2
3
4
5
6
7
8
d
s = 8
s = 7
s = 6
s = 5
s = 4
s = 3
s = 2
s = 1
s = 0

Figure [10]

2.3 Introduction to depth pyramid

A depth pyramid is a data structure in depth resolution. The lowest level 0 with have a depth resolution of 1 bit/pixel. Level 2 will have depth resolution of 2 bit/pixel. Level 3 will have depth resolution of 3 bit/pixel and so on.

The following is an algorithm of progressive transmission using a depth pyramid. An example is illustrated in figure [11].

> For an image X_n of n bit/pixel, a depth pyramid is defined as a sequence of matrices $\{X_k\}$, such that X_{k-1} is a reduced-depth resolution of X_k. That is, X_{k-1} is k-1 bit/pixel and X_k is k bit/pixel. In algorithm form, we have the following steps:
>
> 0. The image X_n is divided into a set of n bit planes.
> 1. The most significant bit plane in the set is transmitted.
> 2. The scheme next transmits the second significant bit plane, and then the third significant bit plane, and so on.
> 3. When all bit planes are transmitted, the receiver obtains an exact replica of X_n.

Usually, viewer's recognition of an image does not decrease linearly with the decrease of depth resolution. The eye functions like a low-pass filter and obey the law of superposition. A black and white grating will become grey when it is too fine to be resolved. For recognition purpose, viewer can manage with relatively low depth resolution. Depth resolution range in 2 to 3 bit/pixel has been found to be acceptable.

Depth pyramid for display

Level 0

0	0	0	0
8	8	8	0
8	0	8	0
8	8	8	0

Level 1

0	0	0	4
12	12	12	4
8	0	12	4
8	8	8	4

Level 2

0	2	2	4
12	12	14	4
10	0	14	6
10	8	8	6

Level 3

1	2	3	4
12	13	14	5
11	0	15	6
10	9	8	7

(Original Image)

Bit plane for transmission

Most significant bit plane

0	0	0	0
1	1	1	0
1	0	1	0
1	1	1	0

2nd significant bit plane

0	0	0	1
1	1	1	1
0	0	1	1
0	0	0	1

3rd significant bit plane

0	1	1	0
0	0	1	0
1	0	1	1
1	0	0	1

least significant bit plane

1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

Figure [11] An example of depth pyramid with 4 bit plane.

2.4 Introduction to spatial pyramid

Extensive research in the past has considered the encoding of an image spatial pyramid data structure. Encoding is bottom up and progressive transmission is from top (lowest spatial resolution) to the bottom level (highest spatial-resolution). Various spatial pyramid structures have been proposed including the mean pyramid [10], the difference pyramid [10], the Laplacian pyramid [9], the S-transform pyramid [11], the reduced-sum pyramid [1], and the reduced-difference pyramid [12]. Goldberg and Wang [13] have reviewed the above pyramid data structures. The performance criteria used are:

1. The equivalent entropy

$$H^{eq} = \frac{N_n}{N_p} \times H$$

where H is the first-order entropy of the pyramid, N_n is the number of pyramid nodes and N_p is the number of pixel in the image.

2. The normalized mean square error

$$NSME(k) = \frac{\sum_{i,j} (X_{ij} - X^*_{k,ij})^2}{\sum_{i,j} X^2_{ij}} \times 100\%$$

where $X_{i,j}$ and $X^*_{i,j}$ are the (i,j)th pixel of the original image and its kth approximation.

3. The total loss-less transmission bit rate

$$R_L = R_p(n) + H_e$$

where $R_p(n)$ is the progressive bit rate up to level n and H_e is the entropy of the residual error image.

They conclude that the reduced-difference pyramid is the best spatial pyramid in terms of minimum equivalent entropy, minimum normalized mean square error, and minimum total loss-less transmission bit rate. We will use the reduced-difference pyramid to illustrate the spatial pyramid. Their algorithms are as follows:

2.4.1 Formation of truncated mean pyramid for progressive display

For an image X_n of size 2^n x 2^n, a truncated mean pyramid used for display is defined as a sequence of matrices $\{X_k\}$, such that X_{k-1} is a reduced-spatial resolution version of X_k. In algorithm form, we have the following steps:

0. Initialization: Let level k = n and $X_k = X_n$.
1. Formation of Level k-1: For each spatially contiguous, non-overlapping block of 2 x 2 nodes at level k, calculate the truncated mean,

$$X_{k-1,[\frac{i+1}{2}],[\frac{j+1}{2}]} = \frac{X_{k,i,j}+X_{k,i,j+1}+X_{k,i+1,j}+X_{k,i+1,j+1}}{4}.$$

where i,j = 1,3, ... 2k-1

and [X] is the truncation of [X + 0.5].

2. Termination: Let k = k-1 and if k ≠ 0, return to step 1; otherwise, stop.

The intermediate levels of the truncated mean pyramid are a set of reduced-spatial resolution approximations of the image suitable for display. Since there is no preceding filtering of the image, we have a violation of the sampling theorem. Usually viewer can tolerate the aliasing in most natural scenes. We need some filling policy to complete the missing pixel so as to keep a fixed size on the display. Repeated pixel causes a strong brick effect due lo large pixel. Interpolation is superior but require computation [14].

2.4.2 Formation of the reduced-difference pyramid for transmission

The reduced-difference pyramid is formed from the truncated mean pyramid as follows:

$$D_{k,i,j} = X_{k,i,j} - X_{k,i,j+1}$$
$$D_{k,i,j+1} = X_{k,i,j+1} - X_{k,i+1,j+1}$$
$$D_{k,i+1,j+1} = X_{k,i+1,j+1} - X_{k,i+1,j}$$
$$D_{k,i+1,j} = X_{k,i+1,j} - X_{k,i,j}$$

where $k = 1,2,\ldots n$ and $i,j = 1,3,\ldots 2^k-1$.

Only three out of the above four difference are transmitted as they are sufficient to reconstruct the truncated mean pyramid $\{X_k\}$.

2.4.3 Reconstruction of the truncated mean pyramid from the reduced-difference pyramid

0. Initialization: Let the top level of the truncated mean pyramid X_o be level 0 and k=1

1. Reconstruction of level k: the node values of level k are obtained by

$$X_{k,i,j} = \frac{4 \times X_{k-1,\lfloor \frac{i+1}{2} \rfloor, \lfloor \frac{j+1}{2} \rfloor} + 3 \times D_{k,i,j} + 2 \times D_{k,i,j+1} + D_{k,i+1,j+1}}{4}$$

$$X_{k,i,j+1} = X_{k,i,j} - D_{k,i,j}$$

$$X_{k,i+1,j+1} = X_{k,i,j+1} - D_{k,i,j+1}$$

$$X_{k,i+1,j} = X_{k,i+1,j+1} - D_{k,i+1,j+1}$$

A detail example of the truncated mean pyramid and reduced-difference pyramid formation for the n = 2 is shown in Figure [12].

The truncated mean pyramid for display

level 0

9

level 1

7	7
12	9

level 2

(original image)

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

The corresponding reduced-difference pyramid for transmission

level 0

9

level 1

0	-2
	-3

level 2

-1	-11	-1	-1
	1		-11
-5	7	9	-1
	-1		-1

Figure [12] An example of truncated mean followed by the corresponding reduced-difference pyramid of image size of 4 x 4.

2.5 Introduction to temporal pyramid

In progressive transmission of digital recurrent video using temporal pyramid, frames of image is sent to viewer sequentially so that the temporal resolution of the display is refined progressively. The viewer's receiving terminal must be able to store the received images so that it can be examined by viewer. With the received frames on local storage, viewer can interactively browse through the reduced-temporal resolution video in a random-access manner. The stored video can be displayed several times at a suitable frame rate to satisfy the flicker requirement and this is suitable for digital recurrent video.

A temporal pyramid of the original video is formed by re-sequencing the image frames before transmission. Difference method of re-sequencing the image frames will result in different subjective quality with the minimum transmission time. The overhead of the transmission is the necessity of transmitting re-sequence information so that the receiver can insert the received frame in the right place in the storage device.

Interpolating the skipped frames at the receiver appears as a method for improving the intermediate temporal quality. However, simple frame repetition technique usually generate jerkily moving objects and linear interpolation usually exhibits blurring in moving areas. Blurring occurs because moving object and stationary background are mixed in the interpolation. These techniques are still in early stage and requires very complex computations. In our study, there will be no post-processing at the receiving end. The reduction of temporal resolution will be by frame skipping.

For a video X_n of n frames numbered from 0 to n-1, a temporal pyramid is defined as a sequence of matrices $\{X_k\}$, such that X_{k-1} is a reduced-temporal resolution version of X_k. That is, X_{k-1} has k-1 frames and X_k has k frames. In algorithm form, we have the following steps:

0. Frame 0 is selected as the first frame.
1. The remaining frames are then re-sequenced. If we resequence the images in binary scan order, we call it naive algorithm. If we maximize the differences of image variances, we call it delta-variance algorithm.
2. The frames are then sequentially transmitted until full temporal resolution is obtained.

A naive algorithm re-sequences the image frames for a video with 8 frames is illustrated below:

Original video: F0 F1 F2 F3 F4 F5 F6 F7	
Re-sequenced video: F0 F4 F2 F6 F1 F3 F5 F7	
Temporal pyramid for display	Transmitted frame
Level 0: F0	F0
Level 1: F0 F4	F4
Level 2: F0 F2 F4	F2
Level 3: F0 F2 F4 F6	F6
Level 4: F0 F1 F2 F4 F6	F1
Level 5: F0 F1 F2 F3 F4 F6	F3
Level 6: F0 F1 F2 F3 F4 F5 F6	F5
Level 7: F0 F1 F2 F3 F4 F5 F6 F7	F7
(Original video)	

A delta-variance algorithm re-sequences the image frames so as to maximize the difference of variance between successive images. The assumption is that we should send the image frame which result in greatest improvement in information content. The information content of two similar image is assumed to be less than the information content of two different images. An example of temporal pyramid for a video with 8 frames is illustrated below:

Original
Video: F0 F1 F2 F3 F4 F5 F6 F7
Variance: 3925 4308 3512 3951 4510 4731 4235 4135
Re-sequenced video: F0 F2 F4 F1 F5 F7 F6 F3

Temporal pyramid for display	Transmitted frame
Level 0: F0	F0
Level 1: F0 F2	F2
Level 2: F0 F2 F4	F4
Level 3: F0 F1 F2 F4	F1
Level 4: F0 F1 F2 F4 F5	F5
Level 5: F0 F1 F2 F4 F5 F7	F7
Level 6: F0 F1 F2 F4 F5 F6 F7	F6
Level 7: F0 F1 F2 F3 F4 F5 F6 F7	F3

(Original video)

In the MPEG standard, Group-Of-Pictures (GOP) is composed of Intrapictures (I), Predicted pictures (P) and Interpolated pictures (B - bi-directional prediction). Both P frame and B frame require a I frame for reference. The application of temporal pyramid to MPEG standard will be restricted to I frames or the constraints imposed on the sequence of forming I frames, P frames and B frames.

```
* A FoxPro 2.0 program for re-sequencing 8 image frames
* using delta-variance algorithm
* frame(i) is the re-sequenced frames, f(i) is the original frames
declare f(8),done(8),frame(8),variance(8)
f(1)=3925
f(2)=4308
f(3)=3512
f(4)=3951
f(5)=4510
f(6)=4731
f(7)=4235
f(8)=4135
frame(1)=0,variance(1)=f(1),done(1)=.T.,olddiff=0
for i=2 to 8
        newdiff=abs(f(i)-f(1))
        if newdiff > olddiff
                olddiff=newdiff
        endif
endfor
olddiff=olddiff/2
oldf=variance(1)
k=2
for l = 2 to 8
        if k <> 9
                for i = 2 to 8
                        add=.t.
                        for j=1 to 8
                                if done(j)
                                  if abs(f(i) - variance(j)) < olddiff
                                                add = .f.
                                                exit
                                  endif
                                endif
                        endfor
                        if add
                                variance(k)=f(i)
                                done(k)= .T.
                                frame(k)=i-1
                                k=k+1
                        endif
                endfor
                olddiff=olddiff/2
        endif
endfor
```

2.6 Proposed algorithm for progressive transmission using depth-spatial-temporal pyramid

For the progressive transmission of digital recurrent video to enable early viewing, there will be waste of time in excessive depth resolution and temporal resolution if only spatial pyramid is used. There will also be waste of time in excessive spatial resolution and temporal resolution if only depth pyramid is used. Similarly, there will be waste of time in excessive depth resolution and spatial resolution if only temporal pyramid is used. More efficient transmission for early viewing would result if we can explore depth resolution, spatial resolution and temporal resolution to form a depth-spatial-temporal pyramid.

A depth-spatial-temporal pyramid of digital recurrent video is displayed in 3 steps:

0. At first step, the lowest level with (d,s,t) equals (1,0,1) is first displayed.

1. Progressive resolution refinement is obtained in a step-wise fashion. At each subsequent step, the depth resolution is increased by one, the spatial resolution is increased by one, and temporal resolution increased by one. Since the quality improvement between levels is non-uniform, the step size is not necessary uniform and is not necessary one. If (d,s,t) are advanced, it is not necessary to advance depth resolution, spatial resolution and temporal resolution with same step since they are not equally sensitively to viewer.

2. Step 1 is repeated until the full resolution (D,S,T) is obtained.

Figure [13] An example of a depth-spatial-temporal pyramid of an video with 4 image frames F0, F1, F2, F3 size 8 x 8 and 4 bit/pixel

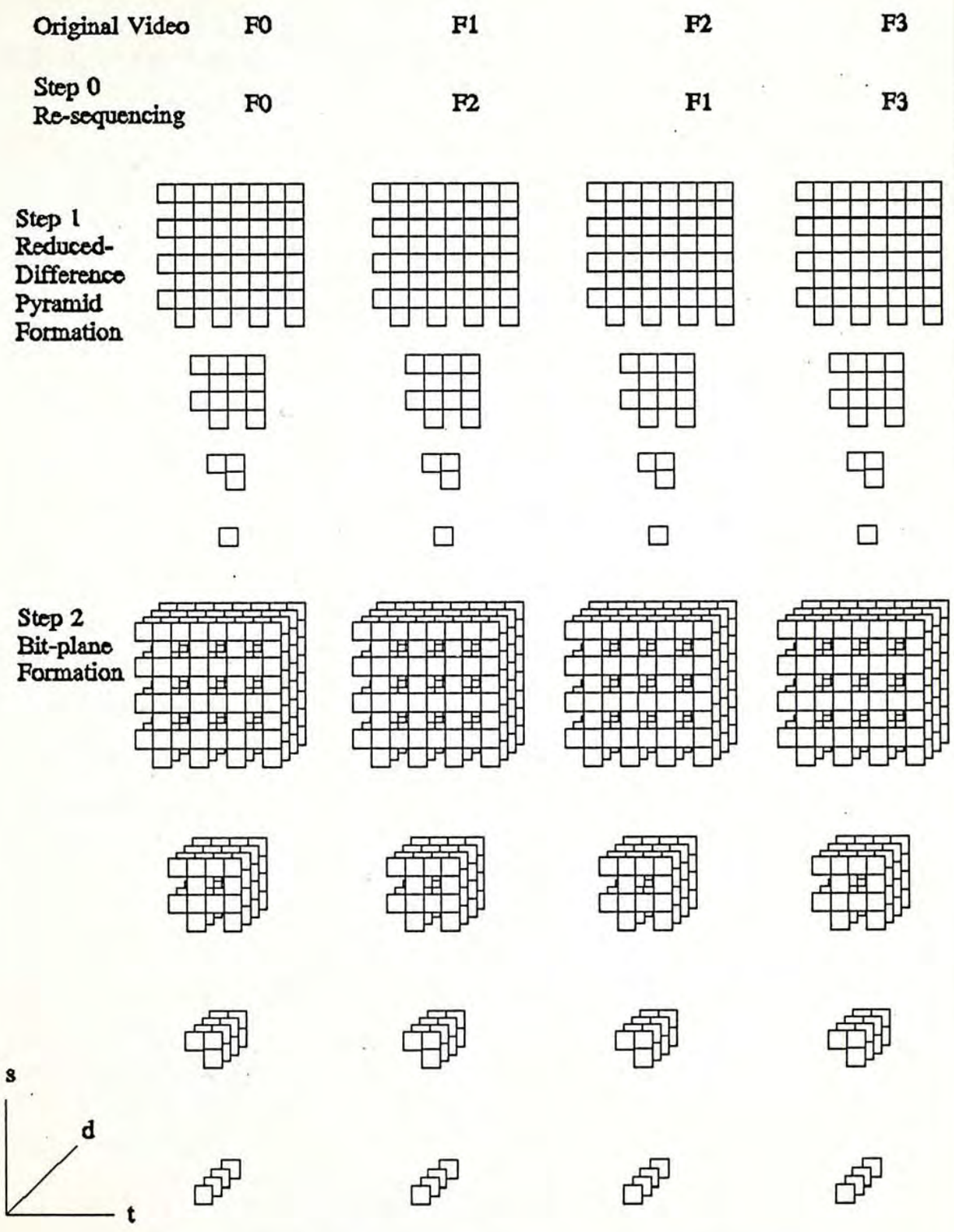


Figure [14] An example of encoding for the transmission of an video with 4 image frames F0, F1, F2, F3 size 8 x 8 and 4 bit/pixel

The corresponding encoding for transmission is also done in 3 steps:

0. A temporal pyramid is first encoded by re-sequencing the image frames.

1. A spatial-temporal pyramid is then encoded by applying the reduced-difference pyramid algorithm on each image frame in the encoded temporal pyramid.

2. Finally, each image in the encoded spatial-temporal pyramid is split into with n bit planes. The resultant data structure is an encoded depth-spatial-temporal pyramid which is suitable for progressive transmission.

Decoding of the encoded depth-spatial-temporal pyramid is done in the reverse order of the encoding steps. The refinement in depth resolution, spatial resolution, and temporal resolution are obtained by transmitting more bit-plane, pixel and image frame respectively.

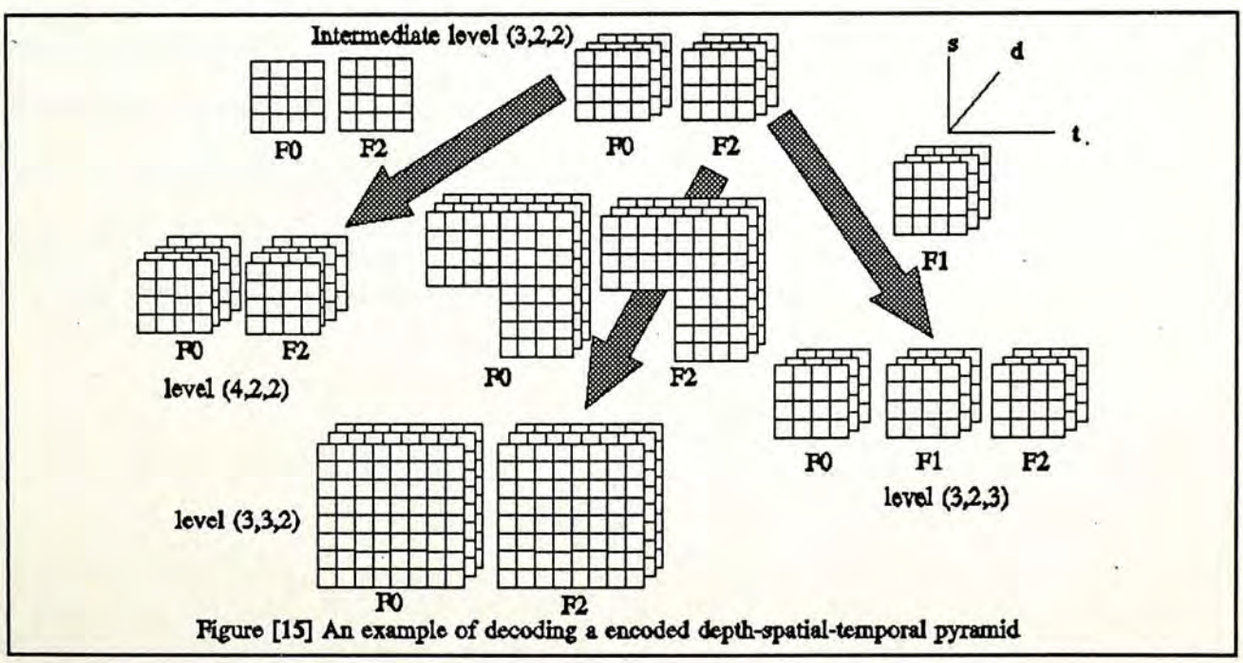


Figure [15] An example of decoding a encoded depth-spatial-temporal pyramid

In the depth-spatial-temporal pyramid, it is not necessary to refine depth resolution, spatial resolution, and temporal resolution by the same step size. The choice of any constant value step size to suit viewer's requirement is difficult. For example, the viewer may like to have a small step size in depth resolution when watching medical recurrent video. He may also like to have a larger step size in depth resolution when watching real-estate catalog. The viewer may also wish to have larger step size in depth resolution initially and a smaller one later. We know that it is very difficult to determine a simple value step size for all possible viewer and all possible images so that an interactive approach is adopted.

An interactive approach can be implemented in two ways. One is a flexible approach which allows the viewer to interactively bias progressive transmission towards depth resolution, spatial resolution or temporal resolution. A user friendly interface will be required. Another is a pre-defined approach which pre-define the step size for depth resolution and spatial resolution by heuristic. Viewer can only bias progressive transmission towards or away from temporal resolution.

Since an interactive approach would involve viewer's request, there is transmission overhead to communicate the desired step size. In our study, a default step size is set initially by experimental trial and error. Then viewer is free to adjust the step size to suit his own requirement. In this way, the next step in the determination of optimum step size is a formal subjective test which determine the optimum step size for an average viewer and a particular type of digital recurrent video.

2.6.1 The Flexible Approach

In the flexible approach, the primary concern is viewer's flexibility. Progressive transmission is obtained in a step-wise fashion. At each step, the depth resolution is increased by Δd, the spatial resolution is increased by Δs, and temporal resolution increased by Δt.

The numerical values of Δd, Δs, Δt can be set by the viewer from 0 to 4. Default values are 1 which means equal bias for depth resolution, spatial resolution, and temporal resolution. If Δd is 0, there is no refinement for depth resolution. If Δd is 4, there is very strong bias towards depth resolution. Overall, the ratio Δd : Δs : Δt would determine the bias of the resolution. During transmission, viewer can stop transmission, changes the Δd, Δs, Δt and restart transmission so as to best fit his requirement. If Δd, Δs, Δt are set to 0 in the following table, then we can have display of all the special case of depth-spatial-temporal pyramid. Please note that the 'don't care' inside the table is any non-zero positive integer value.

Δd	Δs	Δt	Display
0	Don't care	Don't care	Spatial-temporal pyramid
Don't care	0	Don't care	Depth-temporal pyramid
Don't care	Don't care	0	Depth-spatial pyramid
0	0	Don't care	Temporal pyramid
Don't care	0	0	Depth pyramid
0	Don't care	0	Spatial pyramid

2.6.2 The Pre-defined Approach

In the pre-defined approach, the primary concern is transmission efficiency instead of viewer's flexibility. Progressive transmission is obtained in a step-wise fashion. At each step, depth resolution and spatial resolution is increased by a pre-defined sequence, and temporal resolution increased by Δt.

The numerical values of Δd, Δs, Δt can be set by the viewer from 0 to 4. Default values are 1 which means equal bias for pre-defined depth resolution and spatial resolution, and temporal resolution. If Δd or Δs is 0, there is no refinement for depth resolution and spatial resolution. If Δd or Δs is 4, there is very strong bias towards depth and spatial resolution. Overall, the ratio $\Delta d : \Delta t$ or $\Delta s : \Delta t$ whichever is larger would determines the bias towards or away temporal resolution. During transmission, viewer can also stop transmission, changes the Δd, Δs, Δt and restart transmission so as to best fit his requirement.

The pre-defined sequence of depth resolution and spatial resolution is calculated according to the normalized video quality Q. The use of normalized video quality Q is to transform the resolution domain into the perceptual space. The sequence can be calculated by the following steps:

0. Start at the (d,s,t) = (1,0,1) co-ordinate.

1. Among the neighbourhood not yet visited, choose as the next (d,s,t) co-ordinate the one that has the highest normalized video quality Q. Choose bias to d if two neighbours have the same Q.

2. Repeat step 1 until the co-ordinate (D,S,T) corresponding to the original video has been visited.

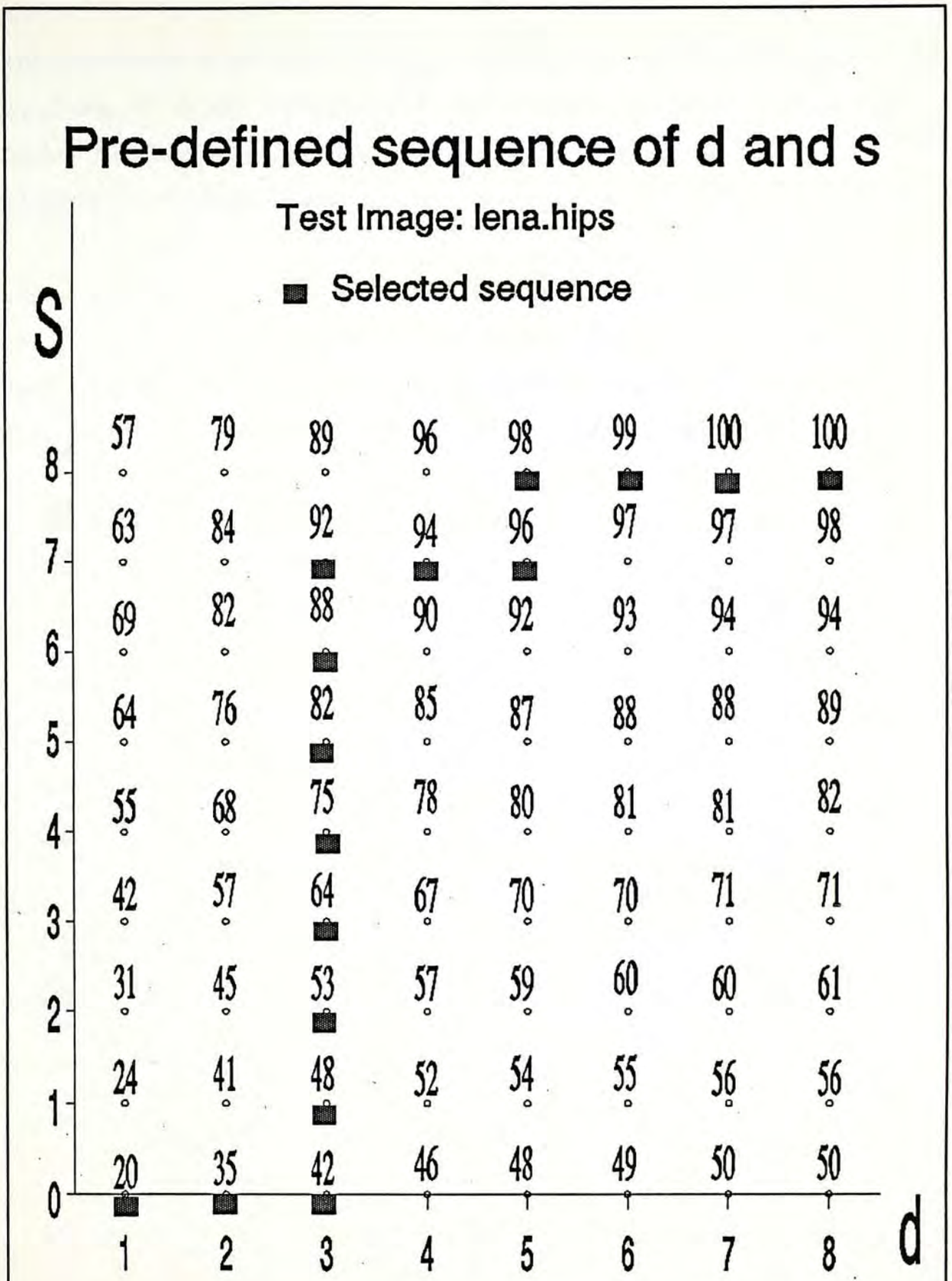


Figure [16] An example of pre-defined d, s

The normalized video quality Q is a class of quality measure depending on the weighting of depth resolution and spatial resolution. Therefore, progressive transmission bias towards depth resolution or spatial resolution can be obtained by adjusting the weighting factors.

The value Q also enables us to compare the different combination of depth resolution and spatial resolution. For the test image lena.hips, the quality Q is 92 for both (d,s) equal (3,7) and (d,s) equals (5,6) even though their numbers of bits are different. This means we should choose the one with fewest bits to enable early recognition.

Chapter 3

Experiment

In the search of a good progressive transmission algorithm, the viewer's judgement is important because he is also part of a communication system. It is imperative to demonstrate the algorithm in real time before any conclusion is made. Therefore, computer simulation can allow subjective test on the algorithm.

Subjective test can evaluate three things in our study. The first is a threshold test. This determines the threshold of resolution for meaningful recognition. This information is useful to assess the relative importance of depth resolution, spatial resolution and temporal resolution. The second is a pair-comparison test. The same video is processed by both the flexible approach and the pre-defined approach. This procedure compares the two approaches. The third is comment scale testing. Each choice of step size Δd, Δt and Δs are given a comment number from 1 to 7. This procedure helps us to determine the optimum step size.

A computer program was written in C to simulate the progressive transmission of digital recurrent video in a SUN work-station. The calculations of normalized video quality are done externally. The simulation program can simulate a throughput from 300 bps up to 64 kbps using the flexible approach and the pre-defined approach. Throughput is simulated by introducing appropriate delay in the display. Depth pyramid, spatial pyramid, temporal pyramid, depth-spatial pyramid, depth-temporal pyramid, spatial-temporal pyramid, and depth-spatial-temporal pyramid can all be simulated.

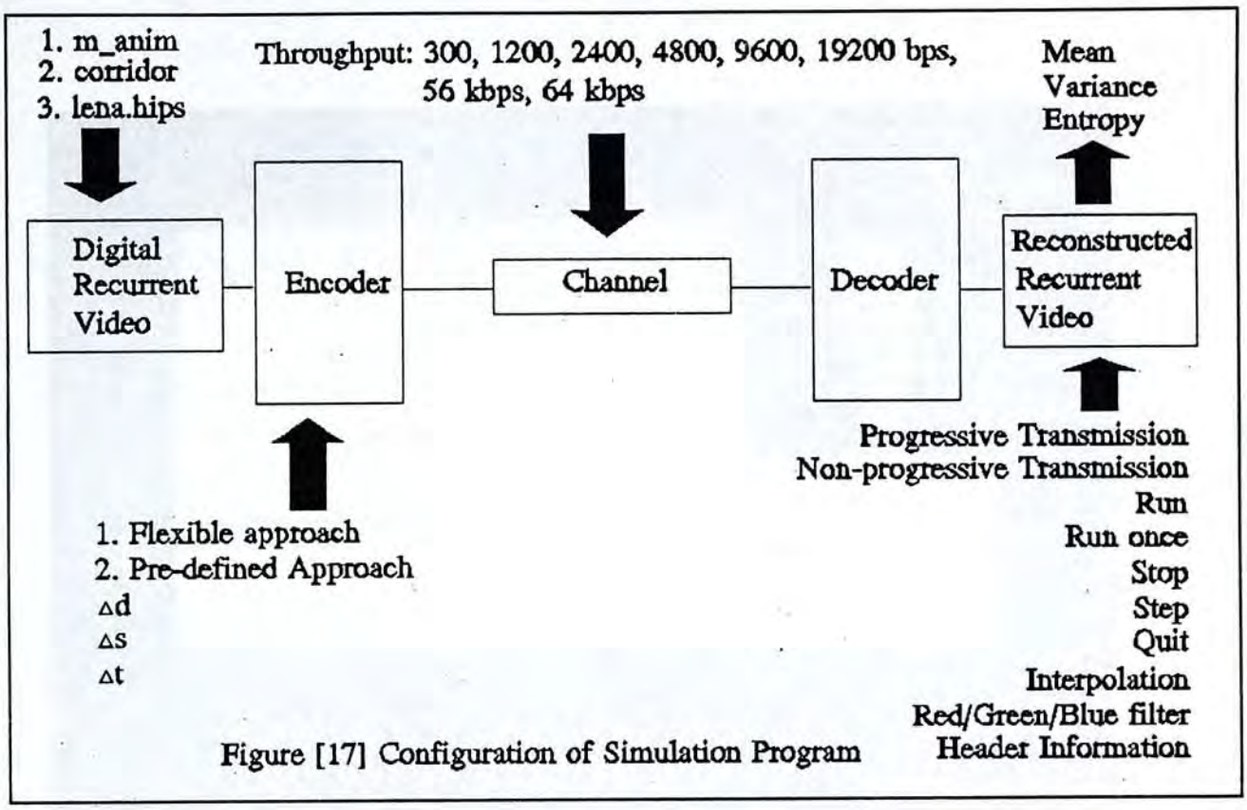
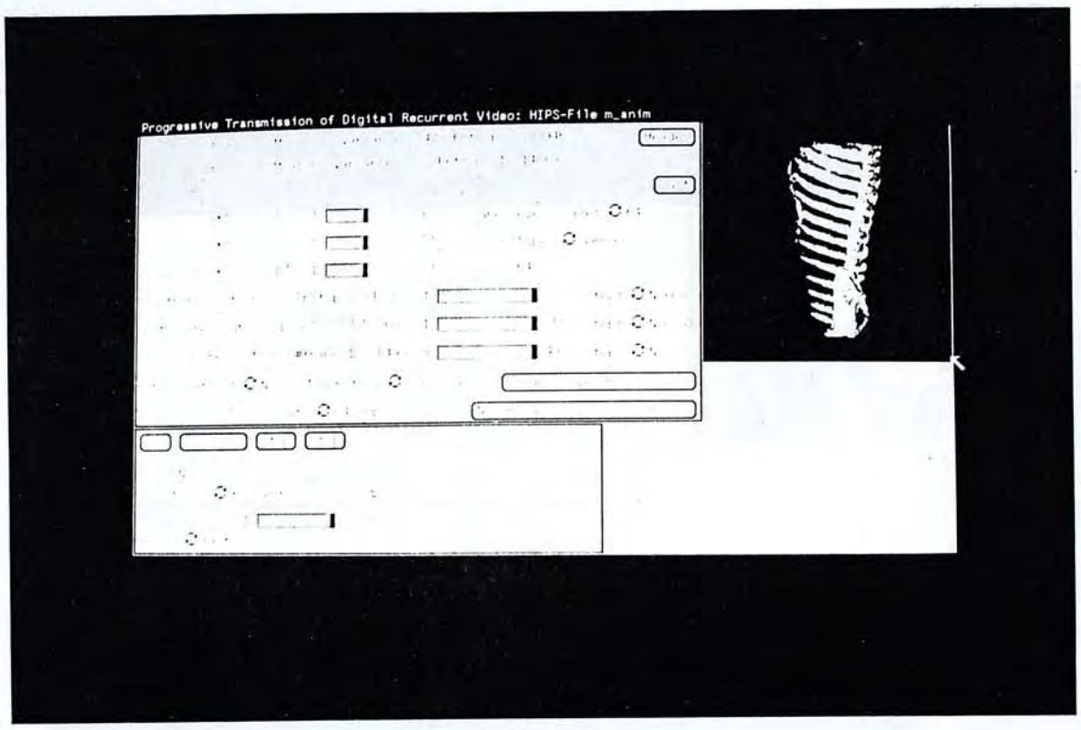


Figure [17] Configuration of Simulation Program

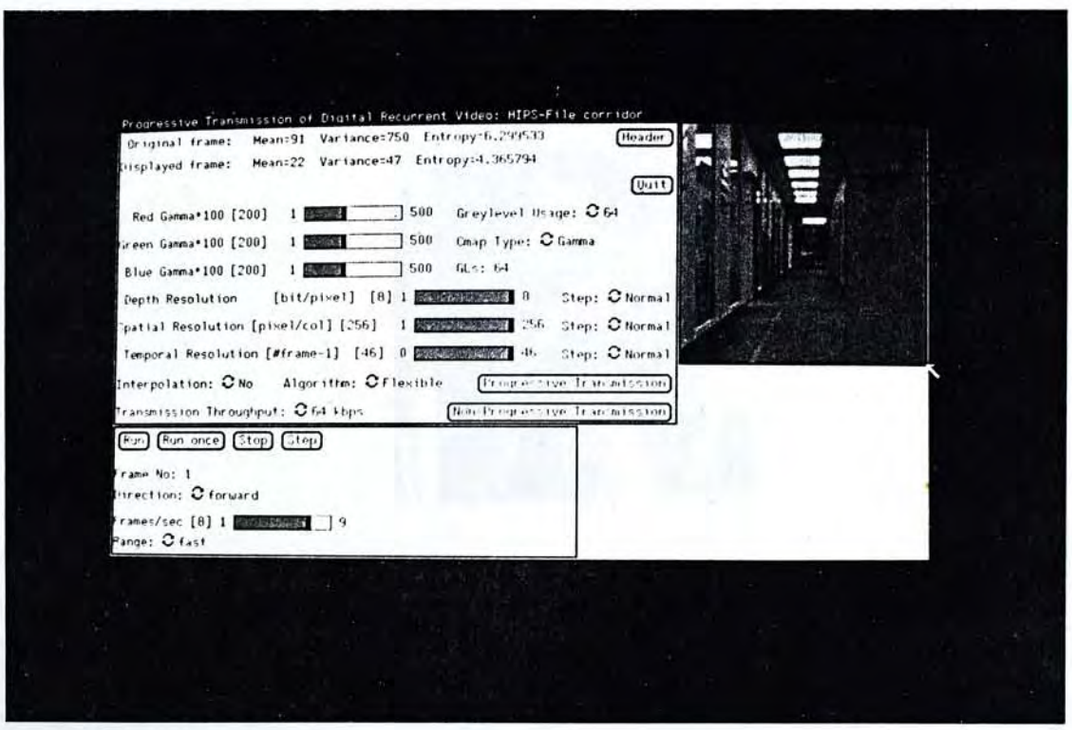
The digital recurrent video is first decomposed into an encoded depth-spatial pyramid. The reconstructed recurrent video is according to the viewer's selected step size Δd, Δt and Δs. Other viewer's selected features are throughput, display frame/second, red/green/blue filter, header information, recurrent run, run once, step, forward/backward, stop and quit. Quantitative measurements including mean, variance and entropy are recorded.

The image format of digital recurrent video 'm_anim' consists of 50 frames 256 x 256 pixel with 8 bit/pixel. Its content is about the rotation of a skeleton. The image format of digital recurrent video 'corridor' consists of 50 frames 256 x 256 pixel with 8 bit/pixel. Its content is the zoom-in and zoom-out of a corridor. The image format of digital recurrent video 'lena.hips' consists of 1 frame 256 x 256 pixel with 8 bit.pixel. These images are used in the experiment.

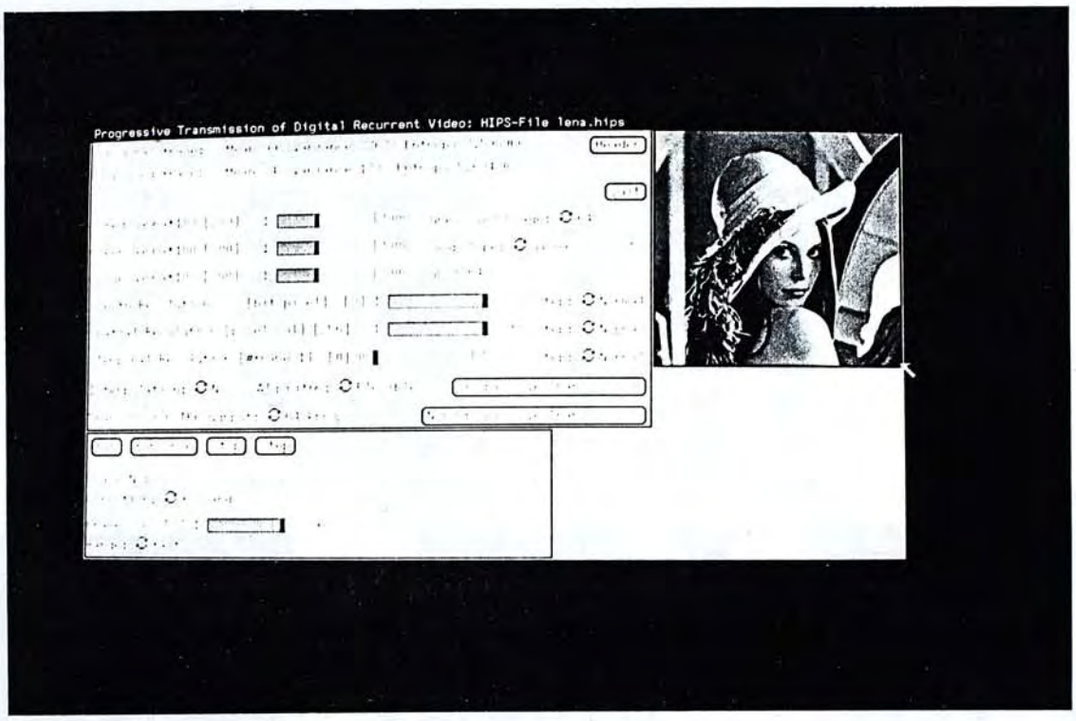
Video 1: m_anim size 256 x 256 depth 8 bit 50 frames



Video 2: corridor size 256 x 256 depth 8 bit 50 frames



Still image: lena.hips size 256 x 256 depth 8 bit



3.1 Simulation on depth pyramid

We observe that sending more than 3 bit is diminishing return from both subjectively and objectively by M.

Depth	Size	mean	variance	entropy	M	V	Q
1	256	39	3472	0.887236	39	75	57
2	256	69	3098	1.781664	70	88	79
3	256	83	2923	2.698812	84	94	89
4	256	91	2781	3.627758	92	99	96
5	256	95	2772	4.610365	96	100	98
6	256	97	2768	5.604208	98	100	99
7	256	98	2768	6.600924	99	100	100
8	256	99	2767	7.598966	100	100	100

d=7

d=6

d=5

d=4

d=3

d=2

d=1

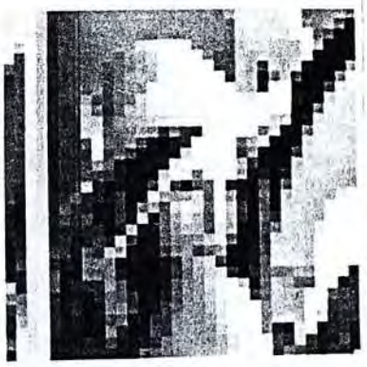
3.2 Simulation on spatial pyramid

We observe that size more than 64 x64 or 128 x 128 is diminishing return from both subjectively and objectively by V.

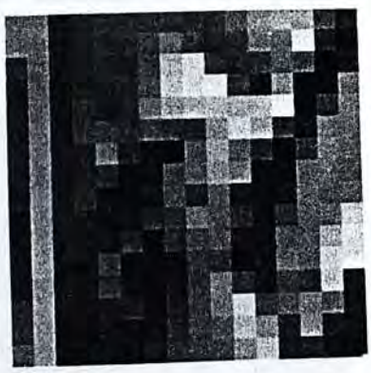
Depth	Size	mean	variance	entropy	M	V	Q
8	1	99	0	0.000000	100	0	50
8	2	99	312	1.500000	100	11	56
8	4	99	570	3.875000	100	21	61
8	8	99	1169	5.593750	100	42	71
8	16	99	1753	6.758099	100	63	82
8	32	99	2136	7.237924	100	77	89
8	64	99	2442	7.448914	100	88	94
8	128	99	2637	7.535016	100	95	98
8	256	99	2767	7.598966	100	100	100

s=7

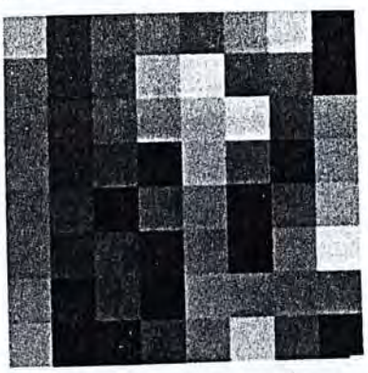
s=6



s=5



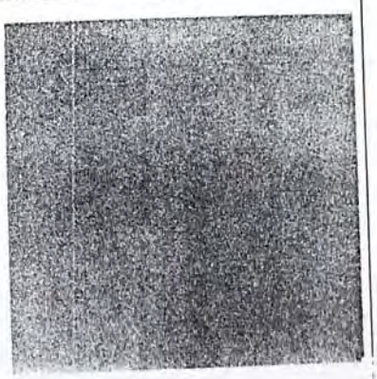
s=4



s=3

s=2

s=1



s=0

3.2.1 Simulation of interpolation to 'brick effect' on spatial pyramid

We observe that the 'brick effect' has been improved significantly.

s=7

s=6

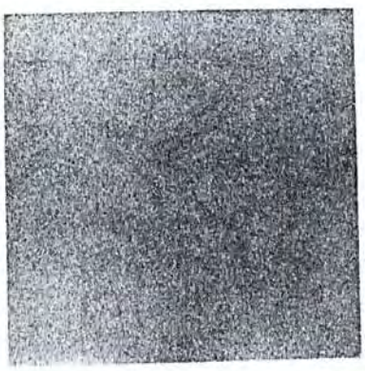
s=5

s=4

s=3

s=2

s=1



s=0

3.3 Simulation on temporal pyramid

Original m_anim

F0

F7

F14

F21

F28

F35

F42

F49

We observe from simulation that the result of the naive algorithm is similar to the delta-variance algorithm. The sending orders of the two algorithms are listed below:

m_anim	Delta-Variance algorithm		Naive algorithm
Sending order	Frame	Variance	Frame
1	0	3925	0
2	5	4456	25
3	3	3524	38
4	31	4843	7
5	1	4292	14
6	2	3748	45
7	22	4089	18
8	27	4590	29
9	8	4223	33

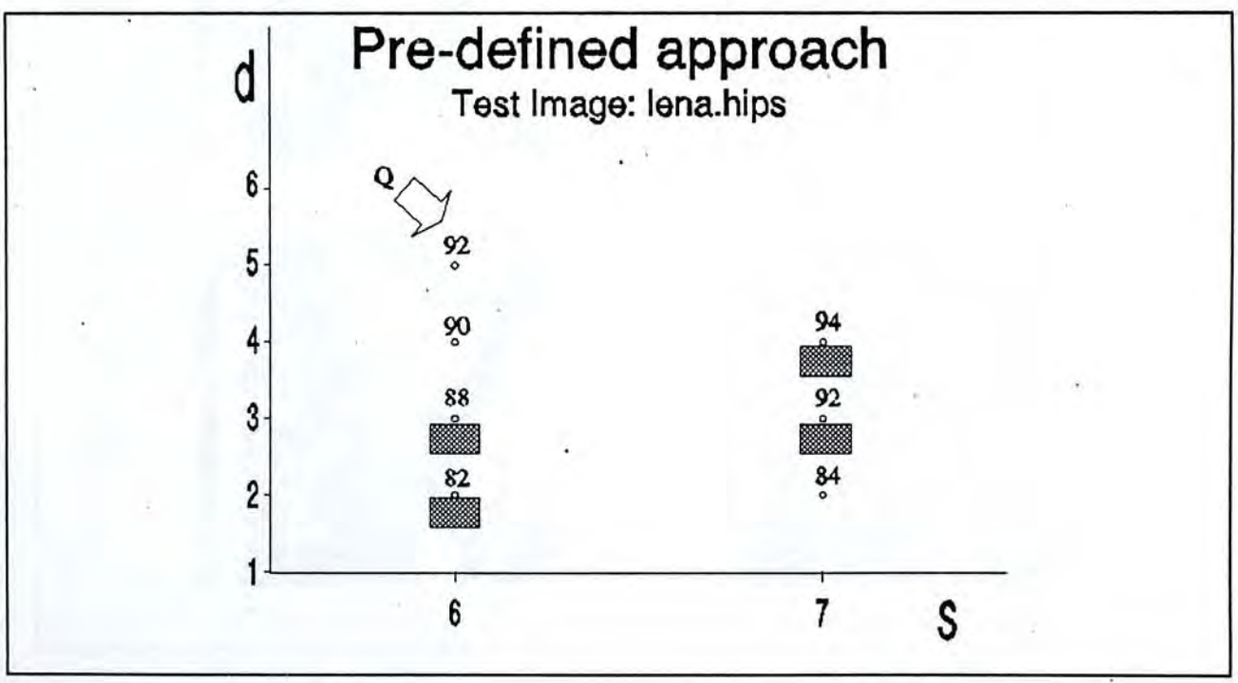
	Delta-Variance algorithm		Naive algorithm
m_anim Sending order	Frame	Variance	Frame
10	9	4016	49
11	11	3830	2
12	13	3631	4
13	32	4946	9
14	33	4755	11
15	16	3583	16
16	17	3663	20
17	18	3698	22
18	24	4344	27
19	26	4501	31
20	29	4415	35
21	43	4161	40
22	46	4048	42
23	48	3970	47
24	4	4243	1
25	7	4308	3
26	12	3720	5
27	15	3496	6
28	21	3951	8
29	25	4328	10
30	35	4731	12
31	36	4618	13
32	37	4565	15
33	38	4532	17
34	40	3990	19
35	45	4070	21
36	49	4137	23
37	6	4430	24
38	14	3512	26
39	19	3820	28
40	28	4510	30
41	42	4235	32
42	47	4027	34
43	23	4230	36
44	44	4095	37
45	34	4458	39
46	41	4220	41
47	39	4429	43
48	1	4292	44
49	2	3748	46
50	3	3524	48

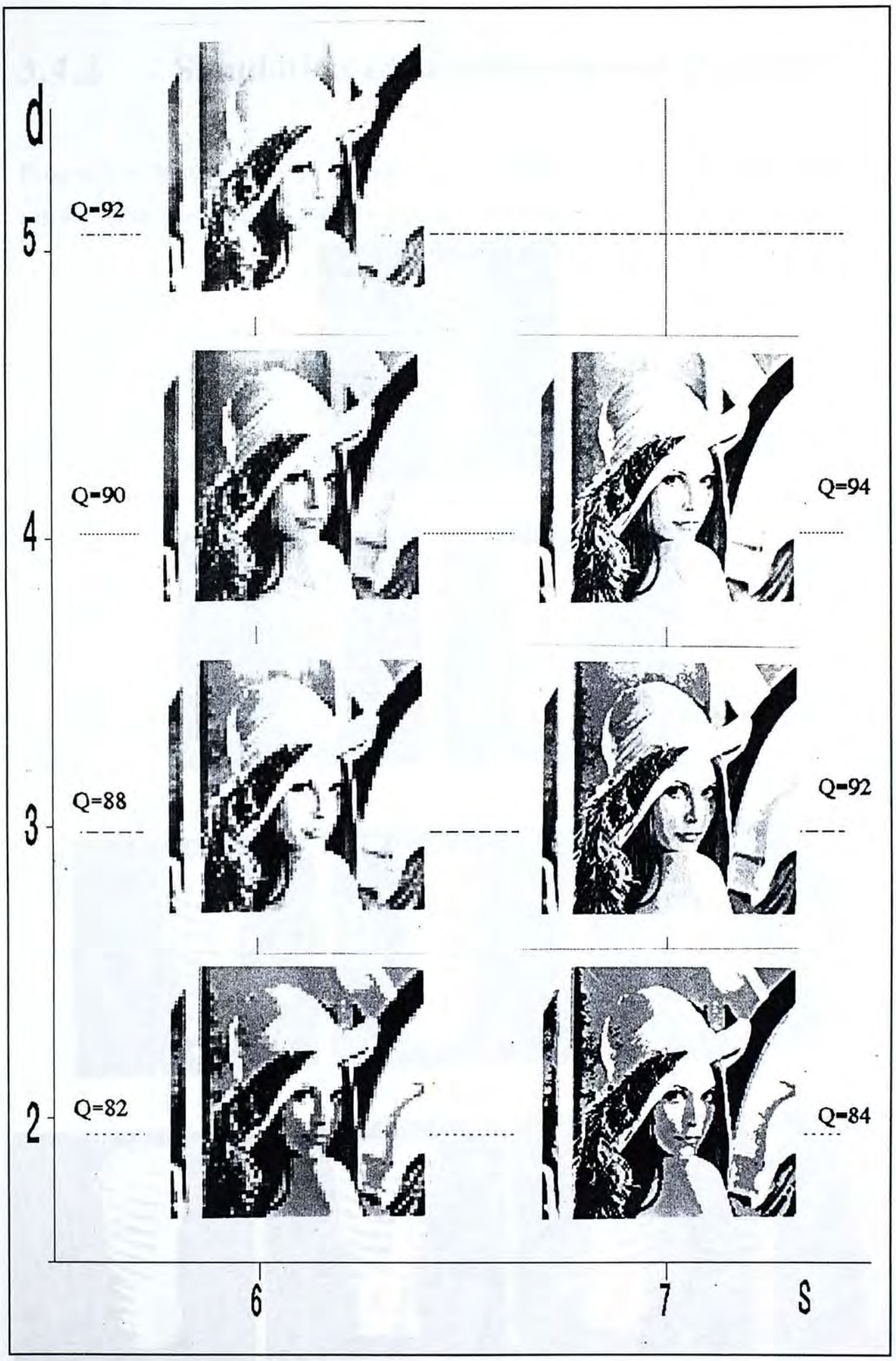
3.4 Simulation on algorithm for progressive transmission using depth-spatial-temporal pyramid

3.4.1 Simulation on depth-spatial pyramid

We observe that depth resolution and spatial resolution can be exchanged according to the class of Q.

Depth	Size	mean	variance	entropy	M	V	Q
2	64	69	2599	4.433237	70	94	82
2	128	69	2844	2.861250	70	97	84
3	64	83	2532	6.102042	84	92	88
4	64	91	2439	7.330325	92	88	90
5	64	95	2442	7.445222	96	88	92
3	128	83	2746	4.192983	84	99	92
4	128	91	2637	5.393473	92	95	94




d
5
4
3
2
Q=92
Q=90
Q=88
Q=82
Q=94
Q=92
Q=84
6
7
S

3.4.2 Simulation on depth-temporal pyramid

Progressive transmission is obtained in a step-wise fashion. In each step, depth resolution is increased by 1 and temporal resolution is also increased by 1.

Level 0

F0 d=1

Level 1

F0 d=2 F25 d=2

Level 2

F0 d=3 F25 d=3 F38 d=3

Level 3

F0 d=4 F7 d=4 F25 d=4 F38 d=4

3.4.3 Simulation on spatial-temporal pyramid

Progressive transmission is obtained in a step-wise fashion. In each step, spatial resolution is increased by a factor of 1 and temporal resolution is also increased by 1.

Level 0

F0 s=0

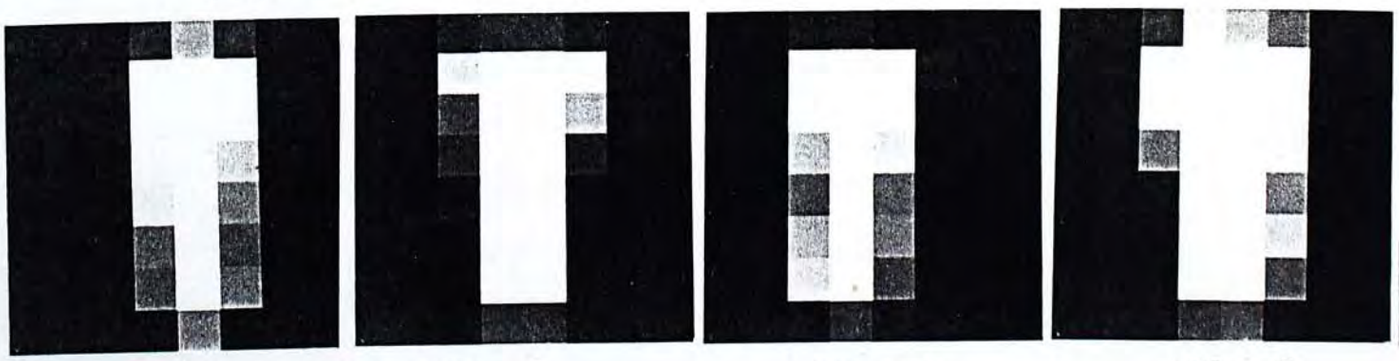
Level 1

F0 s=1 F25 s=1

Level 2

F0 s=2 F25 s=2 F38 s=2

Level 3



F0 s=3 F7 s=3 F25 s=3 F38 s=3

Level 5

F0 s=6

F7 s=6

F14 s=6

F18 s=6

F25 s=6

F38 s=6

F45 s=6

3.4.4 Simulation on depth-spatial-temporal pyramid

For both test digital recurrent video 'm_anim' and 'corridor', non-progressive transmission over 64 kbps requires a total of 409 seconds. The corresponding display rate is 8 seconds per frame. The transmission over 9600 bps requires a total of 2731 seconds and the corresponding display frame rate is 54 seconds per frame. Early viewing can be achieved by progressive transmission.

Progressive transmission is in a step-wise fashion. In each step, depth resolution, spatial resolution, and temporal resolution is increased by Δd, Δs and Δt respectively.

For flexible approach on video m_anim, we find by trial-and-error that $\Delta d=1$, $\Delta s=3$, $\Delta t=2$ would give early perceptible recognition at (d,s,t) co-ordinate of (3,64,5). Percentage of bits sent is about 0.23% of the total number of bits. The corresponding transmission time is 0.96 seconds at 64 kbps and 6.4 seconds at 9600 bps. Further improvement in (d,s,t) co-ordinate is observed at (4,128,6). The corresponding transmission time is 6.14 seconds at 64 kbps and 40.96 seconds at 9600 bps.

The relatively short transmission time for early perceptible recognition is consistent with the theory. The use of interpolation also improves the subjective quality significantly for video m_anim. This is probably due to the removal of the 'brick effect'. However, interpolation may not be good for viewing if the video is square in nature such as a building with many square windows. Transmission time is observed to be different from display time. Transmission time will be longer if the throughput of the channel is lower. The lower bound of the display time is the transmission time. If the display frame rate is low, then the display time will be longer than the transmission time.

For pre-defined approach on video m_anim, we find by trial-and-error that $\Delta d=4$, $\Delta s=4$, $\Delta t=1$ would give early perceptible recognition at (d,s,t) co-ordinate of (4,64,4). Percentage of bits sent is about 0.25% of the total number of bits. The corresponding transmission time is 1.02 seconds at 64 kbps and 6.83 seconds at 9600 bps. Further improvement of (d,s,t) co-ordinate is observed at (4,128,4). The corresponding transmission time is 4.10 seconds at 64 kbps and 27.31 seconds at 9600 bps.

The transmission time for early perceptible recognition between flexible approach and pre-defined approach is not significant. Since the pre-defined approach requires more computation so that the flexible approach is preferred.

For flexible approach on another video corridor, we find by trial-and-error that $\Delta d=1$, $\Delta s=4$, $\Delta t=2$ would give early perceptible recognition at (d,s,t) co-ordinate of (3,32,4). Percentage of bits sent is about 0.23% of the total number of bits. The corresponding transmission time is 0.19 seconds at 64 kbps and 1.28 seconds at 9600 bps. Further improvement of (d,s,t) co-ordinate is observed at (3,64,5). The corresponding transmission time is 0.96 seconds at 64 kbps and 6.40 seconds at 9600 bps.

The transmission time for early perceptible recognition between the two different recurrent video is also not significant.

With the help of guest viewer Mr Li Lung Ming, we observe that the transmission time for early perceptible recognition between two viewers is different. Without telling the guest viewer what he is looking at, the guest viewer is only able to recognize the video m_anim later than the author. Once the guest viewer is told what the video corridor is about, the guest viewer is able to recognize the video as early as the author. This means early viewing can be enhanced by enriching viewer's experience and expectation. .

Other factors such as display monitor setting, the use of colour filter, lighting condition do not affect early viewing significantly.

The result of the simulation at 9600 bps and 64 kbps are as follows:

Early perceptible recognition of m_anim using flexible approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	3	2	(3,64,5)	61440	0.23	6.40	0.96
1	4	2	(3,64,5)	61440	0.23	6.40	0.96
1	2	1	(4,64,4)	65536	0.25	6.83	1.02
1	4	3	(3,64,6)	73728	0.28	7.68	1.15
1	3	3	(3,64,7)	86016	0.33	8.96	1.34
1	4	4	(3,64,7)	86016	0.33	8.96	1.34
1	3	4	(3,64,8)	98304	0.38	10.24	1.54
1	2	2	(4,64,7)	114688	0.44	11.95	1.79
1	0	1	(1,256,2)	131072	0.50	13.65	2.05
1	2	3	(4,64,9)	147456	0.56	15.36	2.30
1	2	4	(4,64,11)	180224	0.69	18.77	2.82
1	1	1	(7,64,7)	200704	0.77	20.91	3.14
0	1	1	(8,64,7)	229376	0.88	23.89	3.58
2	1	1	(8,64,7)	229376	0.88	23.89	3.58
1	1	2	(7,64,12)	344064	1.31	35.84	5.38
0	0	1	(8,256,2)	1048576	4.00	109.23	16.38
1	1	0	(7,64,50)	1433600	5.47	149.33	22.40
0	1	0	(8,64,50)	1638400	6.25	170.67	25.60
1	0	0	(1,256,50)	3276800	12.50	341.33	51.20

Early perceptible recognition of m_anim using pre-defined approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	1	1	(4,64,10)	163840	0.63	17.07	2.56
2	2	1	(4,64,9)	147456	0.56	15.36	2.30
3	3	1	(4,64,4)	65536	0.25	6.83	1.02
4	4	1	(4,64,4)	65536	0.25	6.83	1.02

Further improvement of (d,s,t) of m_anim using flexible approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	3	4	(3,64,9)	110592	0.42	11.52	1.73
1	2	3	(4,64,10)	163840	0.63	17.07	2.56
1	2	4	(4,64,12)	196608	0.75	20.48	3.07
1	4	2	(3,128,5)	245760	0.94	25.60	3.84
1	4	3	(3,128,7)	344064	1.31	35.84	5.38
1	1	2	(7,64,13)	372736	1.42	38.83	5.82
1	0	1	(2,256,3)	393216	1.50	40.96	6.14
1	3	2	(4,128,6)	393216	1.50	40.96	6.14
1	4	4	(3,128,8)	393216	1.50	40.96	6.14
1	2	1	(5,128,5)	409600	1.56	42.67	6.40
1	3	3	(4,128,8)	524288	2.00	54.61	8.19
1	2	2	(5,128,8)	655360	2.50	68.27	10.24
1	1	1	(8,128,8)	1048576	4.00	109.23	16.38
0	1	1	(8,128,8)	1048576	4.00	109.23	16.38
2	1	1	(8,128,8)	1048576	4.00	109.23	16.38
0	0	1	(8,256,3)	1572864	6.00	163.84	24.58
1	0	0	(2,256,50)	6553600	25.00	682.67	102.40
0	1	0	(8,128,50)	6553600	25.00	682.67	102.40
1	1	0	(8,128,50)	6553600	25.00	682.67	102.40

Further improvement of (d,s,t) of m_anim using pre-defined approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	1	1	(4,128,11)	720896	2.75	75.09	11.26
2	2	1	(4,128,9)	589824	2.25	61.44	9.22
3	3	1	(4,128,5)	327680	1.25	34.13	5.12
4	4	1	(4,128,4)	262144	1.00	27.31	4.10

Early perceptible recognition of corridor using flexible approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	4	2	(3,32,4)	12288	0.05	1.28	0.19
1	3	2	(3,32,5)	15360	0.06	1.60	0.24
1	4	3	(3,32,5)	15360	0.06	1.60	0.24
1	2	1	(4,32,4)	16384	0.06	1.71	0.26
1	3	3	(3,32,6)	18432	0.07	1.92	0.29
1	4	4	(3,32,6)	18432	0.07	1.92	0.29
1	3	4	(3,32,7)	21504	0.08	2.24	0.34
1	2	2	(4,32,6)	24576	0.09	2.56	0.38
1	2	3	(4,32,8)	32768	0.13	3.41	0.51
1	1	1	(6,32,6)	36864	0.14	3.84	0.58
1	2	4	(4,32,10)	40960	0.16	4.27	0.64
0	1	1	(8,32,6)	49152	0.19	5.12	0.77
2	1	1	(8,32,6)	49152	0.19	5.12	0.77
1	1	2	(6,32,10)	61440	0.23	6.40	0.96
1	1	0	(6,32,50)	307200	1.17	32.00	4.80
0	1	0	(8,32,50)	409600	1.56	42.67	6.40
1	0	1	(3,256,3)	589824	2.25	61.44	9.22
0	0	1	(8,256,3)	1572864	6.00	163.84	24.58
1	0	0	(2,256,50)	6553600	25.00	682.67	102.40

Further improvement of (d,s,t) of corridor using flexible approach

Δd	Δs	Δt	(d,s,t)	bit-sent	percent-sent (%)	time at 9600 bps	time at 64 kbps
1	3	2	(3,64,5)	61440	0.23	6.40	0.96
1	4	2	(3,64,5)	61440	0.23	6.40	0.96
1	2	1	(4,64,4)	65536	0.25	6.83	1.02
1	1	2	(6,32,11)	67584	0.26	7.04	1.06
1	4	3	(3,64,6)	73728	0.28	7.68	1.15
1	3	3	(3,64,7)	86016	0.33	8.96	1.34
1	4	4	(3,64,7)	86016	0.33	8.96	1.34
1	3	4	(3,64,8)	98304	0.38	10.24	1.54
1	2	2	(4,64,7)	114688	0.44	11.95	1.79
1	2	3	(4,64,9)	147456	0.56	15.36	2.30
1	2	4	(4,64,11)	180224	0.69	18.77	2.82
1	1	1	(7,64,7)	200704	0.77	20.91	3.14
0	1	1	(8,64,7)	229376	0.88	23.89	3.58
2	1	1	(8,64,7)	229376	0.88	23.89	3.58
1	0	1	(4,256,4)	1048576	4.00	109.23	16.38
1	1	0	(7,64,50)	1433600	5.47	149.33	22.40
0	1	0	(8,64,50)	1638400	6.25	170.67	25.60
0	0	1	(8,256,4)	2097152	8.00	218.45	32.77
1	0	0	(3,256,50)	9830400	37.50	1024.00	153.60

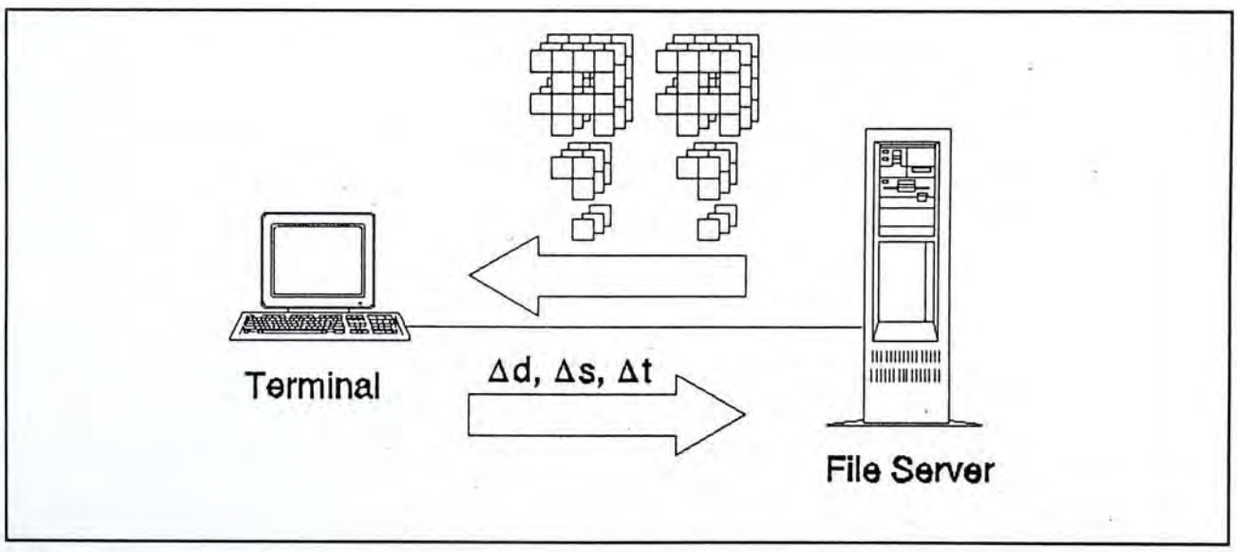
Chapter 4

Conclusions and Discussions

4.1 Discussions

The depth-spatial-temporal pyramid is observed to be a very versatile data structure for representing digital recurrent video with multi-resolution feature. The encoding process involves only simple calculations in the time domain.

The minimal overhead for random-access is to have a label to associate with each (d,s,t) co-ordinate in the multi-resolution space. This overhead is small compared with the file size of the recurrent video. Since the (d,s,t) co-ordinates are labelled, operation on part of the file such as reading and locking is easy. Progressive transmission is achieved by a trajectory from (1,0,1) to (D,S,T). The choice of a particular progressive traversal can be flexibly altered by specifying Δd, Δs, and Δt to suit the requirement of particular viewer and particular applications.



Progressive transmission of digital recurrent video using the depth-spatial-temporal pyramid is a good algorithm because it is flexible to suit the viewer's requirement and also efficient in early recognition. The utmost skill is to make the video look understandable. We reproduce their resolution so as to match the viewer's perceptual space.

For the successful application of progressive transmission, we need low-cost computer, low-cost storage for video and the simple user interface software. We need to develop more sophisticated quantitative model on human vision. Such a development could be the key issue in the future development of progressive video transmission. We also need to adhere to a common testing procedure and use common images to enable the comparison with other algorithms.

4.2 Conclusions

The problem considered in this study is the transmission of digital recurrent video over low bandwidth channel.

We propose a data structure of depth-spatial-temporal pyramid and its use for flexible and efficient progressive transmission to solve the problem. It relies on the ability of the terminal device to store the video and to do some decoding computation. It provides a trade-off between transmission time and user comprehension. It does not apply to video in general but on those are recurrent in nature. In theory, early viewing should be achieved in relatively short transmission time compared with non-progressive transmission. Interactive detailing in resolution also should be achieved by specifying parameters Δd, Δs, and Δt. By the appropriate choice of Δd, Δs, and Δt, it is expected transmission over low bandwidth channel with throughput at 9600 bps or 64 kbps should be satisfactory to the viewer.

Under the simulation experiment conditions, impairments produced by reduced depth resolution, reduced spatial resolution and reduced temporal resolution are found to be very different.

For reduced depth resolution, impairment is coarse contrast which seems tolerable by the viewer. About 2 to 3 bit/pixel is found to be sufficient for early viewing.

For reduced spatial resolution, impairment is a very strong brick effect. Early viewing for size less than 64 x 64 pixel is difficult. The brick effect is annoying to the viewer for the majority of the test images such as lena.hips. However, brick effect is not annoying in the test image building.hips which is a building with many square windows. Although the use of spatial interpolation is observed to reduce the brick effect significantly, it does not always yield pleasing image to the viewer.

For reduced temporal resolution, impairment is jerky motion which also seems tolerable by viewer. 2 to 3 frames are found to be sufficient for early viewing.

The difference of early viewing in using the two different approaches and two different recurrent video is not significant. Sufficient spatial resolution is observed to be the most important for early viewing. Sufficient temporal resolution is important and sufficient depth resolution is relatively less important.

The author is able to recognize the recurrent video earlier than a guest viewer. It is observed that early viewing of the guest viewer can be enhanced by telling him what he is expected to look at.

It is also observed that the display time is longer than the transmission time under the conditions of the experiment.

The overall result shown by computer simulations have demonstrated that progressive algorithm is a viable technique for the transmission of digital recurrent video over low bandwidth channel. Early viewing for the test recurrent video is achieved in about 1 second transmission time at 64 kbps and in about 7 seconds transmission time at 9600 bps. The transmission time means we only need less than 1 percent of the total number of bits for early viewing. Interactive detailing of resolution is also flexible.

Progressive algorithm can be applied to transmit a 3-D object over low bandwidth channel. The progressive algorithm can also be used for the rapid repaint of computer display. It is suitable for computer display with different resolution capability.

However, the problem of recurrent video transmission in low bandwidth channel is not completely solved. The relevance of this thesis is only to show the feasibility of progressive transmission of digital recurrent video over low bandwidth channel. The simulation program only demonstrates the feasibility under the experimental conditions with the author and a guest viewer to give subjective judgement. We need to understand and to quantify the subjective assessment of the general viewers. We need to know the perceptual trade-off between depth resolution, spatial resolution, and temporal resolution. We need to have pilot trial instead of simulation. More experiments and more subjective tests are needed.

Since related paper on this topic is hardly found, further study of the relative merits remains to be assessed.

Chapter 5

5.1 Reference

[1] Kenneth R. Sloan, S. L. Tanimoto, "Progressive Refinement of Raster images", IEEE Transaction on Computers", Vol. 28, No. 11, pp. 871-874, November 1979.

[2] A. J. Frank, J. D. Daniels, D. R. Unangst, "Progressive Image transmission using a growth-geometry coding", Proceedings of the IEEE, Vol. 68, No. 7, pp. 897-909, July 1980.

[3] S. Peleg, M. Werman, H. Rom, "A unified approach to the change of resolution: Space and Gray-level", IEEE Transactions on pattern analysis and machine intelligence, Vol. 11, No. 7, pp. 739-742, July 1989.

[4] Judith H. Irven, "Research in multimedia services", Bellcore Exchange, pp. 6-11, September/October 1988.

[5] Alain Arteri, Oswald Colavin, "A chip set core for image compression", IEEE Transcations on Consumer Electronics, Vol. 36, No. 3, pp.395-402, August 1990.

[6] Bruce K. T. Ho, "Full-Frame Cosine Transform Image Compression for Medial and Industrial Applications", Machine Vision and Applications (1991), pp. 89-96, Vol. 3.

[7] Yuichi Ninomiya, "HDTV Broadcast Systems", IEEE Communication Magazine, pp. 15-22, August 1991.

[8] Ken Knowlton, "Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and lossless Encoding Schemes", Proceedings of the IEEE, Vol. 68, No. 7, pp. 885-896, July 1980.

[9] P. J. Burt, E. H. Adelson, "The Laplacian pyramid as a compact image code", IEEE Transcations on Communications, Vol. 31, No. 4, pp. 532-540, April 1983.

[10] L. Wang, M. Goldberg, " Progressive transmission using vector quantization", IEEE Transaction on Communication, Vol. 37, pp. 1339-1349, Dec. 1989.

[11] T.H. Wendler, D. M. Ebrecht, "Proposed standard for variable format picture processing and a codec approach to match diverse imaging devices", SPIE Vol. 318, pp. 298-305, 1982.

[12] L. Wang, M. Goldberg, "Reduced-difference pyramid: a data structure for progressive image transmission", Journal of Optical Engineering, pp. 708-716, July 1989.

[13] M. Goldberg, L. Wang, "Comparative performance of pyramid data structures for progressive image transmission", IEEE Transaction on Communications, Vol. 39, No. 4, pp. 540-548, April 1991.

[14] A. Sanz, C. Munoz, N. Garcia, "Approximation quality improvement techniques in progressive image transmission", IEEE Journal on slected areas in communications, Vol. 2, No. 2, pp. 359-373, March 1984.

5.2 Appendix

Listing of the simulation program is attached.

```
/*
 *  display: Suntools-based HIPS-format progressive image displayer
 *
 *  Compile this with
 *    cc -O -o display display.c -lsuntool -lsunwindow -lpixrect -lhipsh
 *       -lhips -lm -I/usr/local/include/hips2 -L/usr/local/lib/hips2
 *
 *  Calling format:
 *    display HIPSfile
 *
 */

#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
#include <suntool/seln.h>
#include <suntool/textsw.h>
#include <pixrect/pixrect_hs.h>
#include <pixrect/memvar.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <hipl_format.h>
#include <math.h>

static short HIPSicon_image[]={
/* Format_version=1, Width=64, Height=64, Depth=1, Valid_bits_per_item=16
 */
        0xFFFF,0xFFFF,0xFFFF,0xFFFF,0x8000,0x0000,0x0000,0x0001,
        0x8000,0x0000,0x0000,0x0001,0x83FF,0xFFFF,0xFF00,0x0001,
        0x8200,0x0000,0x0100,0x0001,0x8200,0x0000,0x0100,0x0001,
        0x8200,0x0000,0x0100,0x0001,0x823F,0xFFFF,0xFFF0,0x0001,
        0x8220,0x0000,0x0010,0x0001,0x8220,0x0000,0x0010,0x0001,
        0x8220,0x0000,0x0010,0x0001,0x8223,0xFFFF,0xFFFF,0x0001,
        0x8222,0x0000,0x0001,0x0001,0x8222,0x0000,0x0001,0x0001,
        0x8222,0x0000,0x0001,0x0001,0x8222,0x3FFF,0xFFFF,0xF001,
        0x8222,0x2000,0x0000,0x1001,0x8222,0x2000,0x0000,0x1001,
        0x8222,0x2000,0x0000,0x1001,0x8222,0x2000,0x0000,0x1001,
        0x8222,0x23FF,0xFFFF,0xFF01,0x8222,0x2200,0x0000,0x0101,
        0x8222,0x2200,0x0000,0x0101,0x8222,0x2200,0x0000,0x0101,
        0x8222,0x223F,0xFFFF,0xFFF9,0x8222,0x2220,0x0000,0x0009,
        0x8222,0x2220,0x0000,0x0009,0x8222,0x2220,0x0000,0x0009,
        0x8222,0x2220,0x0000,0x0009,0x8222,0x2220,0x0000,0x0009,
        0x8222,0x2220,0x0000,0x0009,0x8222,0x2220,0x0000,0x0009,
        0x8222,0x2220,0x1C44,0x8809,0x8222,0x2220,0x2244,0xC809,
        0x8222,0x2220,0x2044,0xC809,0x8222,0x2220,0x1044,0xA809,
        0x8222,0x2220,0x0C44,0xA809,0x8222,0x2220,0x0244,0x9809,
        0x8222,0x2220,0x2244,0x9809,0x8222,0x2220,0x2244,0x8809,
        0x83E2,0x2220,0x1C38,0x8809,0x8022,0x2220,0x0000,0x0009,
        0x8022,0x2220,0x0000,0x0009,0x8022,0x2220,0x0000,0x0009,
        0x803E,0x2220,0x0000,0x0009,0x8002,0x2220,0x844F,0xA109,
        0x8002,0x2220,0x8642,0x2109,0x8002,0x2221,0x4642,0x3309,
        0x8003,0xE221,0x4542,0x3309,0x8000,0x2221,0x4542,0x2D09,
        0x8000,0x2222,0x24C2,0x2D09,0x8000,0x2223,0xE4C2,0x2109,
        0x8000,0x3E22,0x2442,0x2109,0x8000,0x0222,0x244F,0xA109,
        0x8000,0x0220,0x0000,0x0009,0x8000,0x0220,0x0000,0x0009,
        0x8000,0x03E0,0x0000,0x0009,0x8000,0x0020,0x0000,0x0009,
        0x8000,0x0020,0x0000,0x0009,0x8000,0x0020,0x0000,0x0009,
        0x8000,0x003F,0xFFFF,0xFFF9,0x8000,0x0000,0x0000,0x0001,
        0x8000,0x0000,0x0000,0x0001,0xFFFF,0xFFFF,0xFFFF,0xFFFF
};

DEFINE_ICON_FROM_IMAGE(icon,HIPSicon_image);
#define ITIMER_NULL ((struct itimerval *) 0)

Panel_item dslider,sslider,tslider,ditem,sitem,titem,optionitem,optionbw,qitem;
int dfactor=8,sfactor=256,tfactor=0,dstep=1,sstep=1,tstep=1,interpolated=1;
int next=0,frameno,bandwidth=64000,state=0,delay,dcount=0,scount=0,tcount=0;
int newd=1,news=1,newt=0,level=0,qstep=1;
static void dfrom_proc(),sfrom_proc(),tfrom_proc(),dset(),sset(),tset();
static void setoption(),setbw(),progressive(),nonprogressive(),qset();
static void speed_proc(),frame_proc(),setrange(),newfile_proc();
static void nextframe(),displayit(),run_proc(),stop_proc(),step_proc();
static void rev_step_proc(),setdir(),runonce_proc(),done_proc();
static void header_proc(),quit_proc(),gammar_proc(),gammag_proc(),gammab_proc();
```

```
static void setpixrange(),setcmapchoice(),loadcmap_proc();
static Panel_setting text_proc();
static Notify_value resett(),catchclose();
struct itimerval frame_timer;
Frame base_frame,header_frame;
Canvas canvas;
Panel filepanel,controlpanel,header_panel1;
Panel_item speedslider,frameslider,rangeitem;
Panel_item currfile,message,fileitem,maptypeitem,percent;
Panel_item hdrfname,redslider,greenslider,blueslider,numgls;
Textsw header_panel2;
Pixwin *canpixwin;

int fps = 8;
int spf = 1;
int mode = 0;           /* 0=fast, 1=slow */
int pixelsize,space;
Boolean greydisplay,obinary;
Boolean runonce = FALSE,transmit = FALSE;
Boolean image_shown = FALSE;
Boolean cmap_loaded = FALSE;
Boolean fullsw;         /* TRUE = 256 pixel values, FALSE = 64 pixel values
                           and shift (applies to greyscale only) */
int frcount = 0;
int dir = 0;            /* 0=forward, 1=backward*/
int cmaptype = 0;       /* 0=hdr/ramp, 1=ramp, 2=gamma, 3=file */
Boolean running = FALSE;
Boolean iconic = FALSE;
struct header hd;
Boolean fflag,rflag,cflag,gflag,sgflag;
double gammar,gammag,gammab;
Boolean cmapvalid = FALSE;      /* TRUE if file colormap has been read */
Boolean hmapvalid;              /* TRUE if header cmap has been found */
byte cred[256],cgreen[256],cblue[256],gred[256],ggreen[256],gblue[256];
byte sgred[64],sggreen[64],sgblue[64],rred[256],rgreen[256],rblue[256];
byte srred[64],srgreen[64],srblue[64],*hred,*hgreen,*hblue;
byte bred[2]={0,255},bgreen[2]={0,255},bblue[2]={0,255};
char hmapname[50],gmapname[100],sgmapname[100],*cmapname;
int cnumcol,hnumcol;

int ourpackedtype =
#ifdef MSBFVERSION
        PFMSBF;
#else
        PFLSBF;
#endif

static Flag_Format flagfmt[] = {
    {"f",{"r","c","g","sg",LASTFLAG},0,{{PTBOOLEAN,"FALSE"},LASTPARAMETER}},
    {"r",{"f","c","g","sg",LASTFLAG},0,{{PTBOOLEAN,"FALSE"},LASTPARAMETER}},
    {"c",{"f","r","g","sg",LASTFLAG},1,{{PTBOOLEAN,"FALSE"},
        {PTFILENAME,"","colormapfile"},LASTPARAMETER}},
    {"g",{"f","r","c","sg",LASTFLAG},0,{{PTBOOLEAN,"FALSE"},
        {PTDOUBLE,"2","gammar"},
        {PTDOUBLE,"-1","gammag"},{PTDOUBLE,"-1","gammab"},LASTPARAMETER}},
    {"sg",{"f","r","c","g",LASTFLAG},0,{{PTBOOLEAN,"FALSE"},
        {PTDOUBLE,"2","gammar"},
        {PTDOUBLE,"-1","gammag"},{PTDOUBLE,"-1","gammab"},LASTPARAMETER}},
    {"s",{LASTFLAG},0,{{PTINT,"1","sizepixel"},LASTPARAMETER}},
    {"Wi",{LASTFLAG},0,{LASTPARAMETER}},
    {"WI",{LASTFLAG},1,{{PTSTRING,"",""},LASTPARAMETER}},
    {"Wl",{LASTFLAG},1,{{PTSTRING,"",""},LASTPARAMETER}},
    {"WL",{LASTFLAG},1,{{PTSTRING,"",""},LASTPARAMETER}},
    {"Wt",{LASTFLAG},1,{{PTSTRING,"",""},LASTPARAMETER}},
    {"WT",{LASTFLAG},1,{{PTSTRING,"",""},LASTPARAMETER}},
    {"Wp",{LASTFLAG},2,{{PTSTRING,"",""},{PTSTRING,"",""},LASTPARAMETER}},
    {"WP",{LASTFLAG},2,{{PTSTRING,"",""},{PTSTRING,"",""},LASTPARAMETER}},
    LASTFLAG};

static qsequence[]={0,5,3,31,1,2,22,27,8,9,
                    11,13,32,33,16,17,18,24,26,29,
                    43,46,48,4,7,12,15,21,25,35,
                    36,37,38,40,45,49,6,14,19,28,
                    42,47,23,44,34,41,39,1,2,3};
```

```
static nsequence[]={0,25,38,7,14,45,18,29,33,49,
                    2,4,9,11,16,20,22,27,31,35,
                    40,42,47,1,3,5,6,8,10,12,
                    13,15,17,19,21,23,24,26,28,30,
                    32,34,36,37,39,41,43,44,46,48};
static   qdepth[]={1,2,3,4,4,4,4, 4, 4, 4,  4,  5,  5,  6,  7,  8};
static qspatial[]={1,1,1,1,2,4,8,16,32,64,128,128,256,256,256,256};
int storage[100];
int types[] = {PFBYTE,LASTTYPE};
char str[100];
int nrows,ncols,nframes,fpanelhgt,fpanelwdth,cpanelhgt;
Boolean validheader;
struct pixrect **mpr;
Filename filename;

main(argc,argv)

int argc;
char *argv[];
{
        int i;
        char c;
        char *dummy;
        Filename mapfile;
        FILE *fp;

        Progname = strsave(*argv);
        parseargs(argc,argv,flagfmt,&fflag,&rflag,&cflag,&mapfile,&gflag,
                &gammar,&gammag,&gammab,&sgflag,&gammar,&gammag,&gammab,
                &pixelsize,&dummy,&dummy,&dummy,&dummy,&dummy,&dummy,
                &dummy,&dummy,FFONE,&filename);
        if (gammag < 0)
                gammag = gammar;
        if (gammab < 0)
                gammab = gammag;
        fullsw = fflag || gflag || cflag || (findparam(&hd,"cmap") != NULLPAR);
        fullsw = TRUE;
        if (rflag || fflag)
                cmaptype = 1;
        else if (gflag || sgflag)
                cmaptype = 2;
        else if (cflag) {
                readcmap(mapfile,256,&cnumcol,cred,cgreen,cblue);
                cmapvalid = TRUE;
                cmaptype = 3;
                cmapname = strsave(mapfile);
        }
        else
                cmaptype = 0;
        sprintf(hmapname,"sunv%d",getpid());
        gammalut();
        for (i=0;i<256;i++)
                rred[i] = rgreen[i] = rblue[i] = i;
        for (i=0;i<64;i++)
                srred[i] = srgreen[i] = srblue[i] = i*4;
        fp = hfopenr(filename);
        init_windows(argc,argv);
        if ((fflag || cflag || gflag || sgflag) && !greydisplay)
                perr(HE_IMSG,
                    "full color map for binary monitors is not meaningful");
        fileinit(fp,filename,TRUE);
        panel_set(maptypeitem,PANEL_VALUE,2,0);
        cmaptype=2;
        setcmap();
        notify_interpose_event_func(base_frame,catchclose,NOTIFY_SAFE);
        notify_interpose_destroy_func(base_frame,resett);
        settimer();
        hipserrprt = hipserrlev = HEL_SEVERE;
        window_main_loop(base_frame);
}
```

```
init_windows(argc,argv)

int argc;
char **argv;

{
        sprintf(str,"Progressive Transmission of Digital Recurrent Video: HIPS-File %s",filename);
        base_frame = window_create(NULL,FRAME,FRAME_LABEL,str,
                FRAME_ARGS,argc,argv,WIN_ERROR_MSG,
                "can't create window, program must be run under suntools",0);
        window_set(base_frame,FRAME_ICON,&icon,0);
        filepanel = window_create(base_frame,PANEL,
                WIN_COLUMNS,75,
                0);
        canvas = window_create(base_frame,CANVAS,
                WIN_RIGHT_OF,filepanel,
                WIN_Y,0,
                CANVAS_REPAINT_PROC,displayit,0);
        canpixwin=canvas_pixwin(canvas);
        greydisplay = (canpixwin -> pw_pixrect -> pr_depth) >= 8 ? TRUE : FALSE;
        window_set(filepanel,WIN_ROWS,greydisplay ? 15 : 4,0);
        currfile = panel_create_item(filepanel,PANEL_MESSAGE,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,6,
                PANEL_LABEL_STRING,"",
                0);
        panel_create_item(filepanel,PANEL_BUTTON,
                PANEL_ITEM_X,ATTR_COL(67),
                PANEL_ITEM_Y,6,
                PANEL_LABEL_IMAGE,panel_button_image(filepanel,"Header",0,0),
                PANEL_NOTIFY_PROC,header_proc,
                0);
        message = panel_create_item(filepanel,PANEL_MESSAGE,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,58,
                PANEL_LABEL_STRING,"",
                0);
        percent = panel_create_item(filepanel,PANEL_MESSAGE,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,32,
                PANEL_LABEL_STRING,"",
                0);
        panel_create_item(filepanel,PANEL_BUTTON,
                PANEL_ITEM_X,ATTR_COL(69),
                PANEL_ITEM_Y,58,
                PANEL_LABEL_IMAGE,panel_button_image(filepanel,"Quit",0,0),
                PANEL_NOTIFY_PROC,quit_proc,
                0);
        dslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,178,
                PANEL_LABEL_STRING," Depth Resolution     [bit/pixel] ",
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,dfrom_proc,
                0);
        sslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,208,
                PANEL_LABEL_STRING,"Spatial Resolution [pixel/col]",
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,sfrom_proc,    0);
        tslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,238,
                PANEL_LABEL_STRING," Temporal Resolution [#frame-1] ",
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,tfrom_proc,
                0);
        ditem = panel_create_item(filepanel,PANEL_CYCLE,
                PANEL_ITEM_X,ATTR_COL(60),
                PANEL_ITEM_Y,178,
                PANEL_LABEL_STRING,"Step:",
                PANEL_CHOICE_STRINGS,"None","Normal","Fast","Faster","Fastest",0,
                PANEL_VALUE,1,
                PANEL_NOTIFY_PROC,dset,
                0);
```

```
sitem = panel_create_item(filepanel,PANEL_CYCLE,
        PANEL_ITEM_X,ATTR_COL(60),
        PANEL_ITEM_Y,208,
        PANEL_LABEL_STRING,"Step:",
        PANEL_CHOICE_STRINGS,"None","Normal","Fast","Faster","Fastest",0,
        PANEL_VALUE,1,
        PANEL_NOTIFY_PROC,sset,
        0);
titem = panel_create_item(filepanel,PANEL_CYCLE,
        PANEL_ITEM_X,ATTR_COL(60),
        PANEL_ITEM_Y,238,
        PANEL_LABEL_STRING,"Step:",
        PANEL_CHOICE_STRINGS,"None","Normal","Fast","Faster","Fastest",0,
        PANEL_VALUE,1,
        PANEL_NOTIFY_PROC,tset,
        0);
optionitem = panel_create_item(filepanel,PANEL_CYCLE,
        PANEL_ITEM_X,0,
        PANEL_ITEM_Y,268,
        PANEL_LABEL_STRING,"Interpolation:",
        PANEL_CHOICE_STRINGS,"Yes","No",0,
        PANEL_VALUE,interpolated,
        PANEL_NOTIFY_PROC,setoption,
        0);
optionbw = panel_create_item(filepanel,PANEL_CYCLE,
        PANEL_ITEM_X,0,
        PANEL_ITEM_Y,298,
        PANEL_LABEL_STRING,"Transmission Throughput:",
        PANEL_CHOICE_STRINGS,"64 kbps","56 kbps","19200 bps",
                "9600 bps","4800 bps","2400 bps","1200 bps","300 bps",0,
        PANEL_VALUE,0,
        PANEL_NOTIFY_PROC,setbw,
        0);
panel_create_item(filepanel,PANEL_BUTTON,
        PANEL_ITEM_X,ATTR_COL(49),
        PANEL_ITEM_Y,268,
        PANEL_LABEL_IMAGE,panel_button_image(filepanel,"Progressive Transmission",0,0),
        PANEL_NOTIFY_PROC,progressive,
        0);
panel_create_item(filepanel,PANEL_BUTTON,
        PANEL_ITEM_X,ATTR_COL(45),
        PANEL_ITEM_Y,298,
        PANEL_LABEL_IMAGE,panel_button_image(filepanel,"Non-Progressive  Transmission",0,0),
        PANEL_NOTIFY_PROC,nonprogressive,
        0);
qitem = panel_create_item(filepanel,PANEL_CYCLE,
        PANEL_ITEM_X,ATTR_COL(23),
        PANEL_ITEM_Y,268,
        PANEL_LABEL_STRING,"Algorithm:",
        PANEL_CHOICE_STRINGS,"Pre-defined","Flexible",0,
        PANEL_VALUE,1,
        PANEL_NOTIFY_PROC,qset,
        0);
if (greydisplay) {
        redslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,88,
                PANEL_LABEL_STRING,"  Red Gamma*100",
                PANEL_VALUE,200,
                PANEL_MIN_VALUE,1,
                PANEL_MAX_VALUE,500,
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,gammar_proc,
                0);
        greenslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,118,
                PANEL_LABEL_STRING,"Green Gamma*100",
                PANEL_VALUE,200,
                PANEL_MIN_VALUE,1,
                PANEL_MAX_VALUE,500,
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,gammag_proc,
                0);
```

```
        blueslider = panel_create_item(filepanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,148,
                PANEL_LABEL_STRING," Blue Gamma*100",
                PANEL_VALUE,200,
                PANEL_MIN_VALUE,1,
                PANEL_MAX_VALUE,500,
                PANEL_NOTIFY_LEVEL,PANEL_ALL,
                PANEL_NOTIFY_PROC,gammab_proc,
                0);
        panel_create_item(filepanel,PANEL_CYCLE,
                PANEL_ITEM_X,ATTR_COL(46),
                PANEL_ITEM_Y,88,
                PANEL_LABEL_STRING,"Greylevel Usage:",
                PANEL_CHOICE_STRINGS,"64","256",0,
                PANEL_VALUE,fullsw ? 1 : 0,
                PANEL_NOTIFY_PROC,setpixrange,
                0);
        if (cmapvalid) {
            maptypeitem = panel_create_item(filepanel,PANEL_CYCLE,
                PANEL_ITEM_X,ATTR_COL(46),
                PANEL_ITEM_Y,118,
                PANEL_LABEL_STRING,"Cmap Type:",
                PANEL_CHOICE_STRINGS,"Hdr/Ramp","Ramp","Gamma","File",0,
                PANEL_VALUE,3,
                PANEL_NOTIFY_PROC,setcmapchoice,
                0);
        }
        else {
            maptypeitem = panel_create_item(filepanel,PANEL_CYCLE,
                PANEL_ITEM_X,ATTR_COL(46),
                PANEL_ITEM_Y,118,
                PANEL_LABEL_STRING,"Cmap Type:",
                PANEL_CHOICE_STRINGS,"Hdr/Ramp","Ramp","Gamma",0,
                PANEL_VALUE,(rflag || fflag) ? 1 :
                        ((gflag || sgflag) ? 2 : 0),
                PANEL_NOTIFY_PROC,setcmapchoice,
                0);
        }
        numgls = panel_create_item(filepanel,PANEL_MESSAGE,
                PANEL_ITEM_X,ATTR_COL(46),
                PANEL_ITEM_Y,148,
                PANEL_LABEL_STRING,"",
                0);
}
controlpanel = window_create(base_frame,PANEL,
        WIN_BELOW,filepanel,
        WIN_X,0,
        WIN_COLUMNS,62,
        0);
panel_create_item(controlpanel,PANEL_BUTTON,
        PANEL_LABEL_IMAGE,panel_button_image(controlpanel,"Run",0,0),
        PANEL_NOTIFY_PROC,run_proc,
        0);
panel_create_item(controlpanel,PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
                panel_button_image(controlpanel,"Run once",0,0),
        PANEL_NOTIFY_PROC,runonce_proc,
        0);
panel_create_item(controlpanel,PANEL_BUTTON,
        PANEL_LABEL_IMAGE,panel_button_image(controlpanel,"Stop",0,0),
        PANEL_NOTIFY_PROC,stop_proc,
        0);
panel_create_item(controlpanel,PANEL_BUTTON,
        PANEL_LABEL_IMAGE,panel_button_image(controlpanel,"Step",0,0),
        PANEL_NOTIFY_PROC,step_proc,0);
frameslider = panel_create_item(controlpanel,PANEL_MESSAGE,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,42,
                PANEL_LABEL_STRING,"",   0);
panel_create_item(controlpanel,PANEL_CYCLE,
        PANEL_ITEM_X,0,
        PANEL_ITEM_Y,62,
        PANEL_LABEL_STRING,"Direction:",
        PANEL_CHOICE_STRINGS,"forward","reverse",0,
        PANEL_NOTIFY_PROC,setdir,0);
```

```
        speedslider = panel_create_item(controlpanel,PANEL_SLIDER,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,92,
                PANEL_LABEL_STRING,"Frames/sec",
                PANEL_VALUE,fps,
                PANEL_MIN_VALUE,1,
                PANEL_MAX_VALUE,fps+1,
                PANEL_NOTIFY_LEVEL,PANEL_DONE,
                PANEL_NOTIFY_PROC,speed_proc,
                0);
        rangeitem = panel_create_item(controlpanel,PANEL_CYCLE,
                PANEL_ITEM_X,0,
                PANEL_ITEM_Y,112,
                PANEL_LABEL_STRING,"Range:",
                PANEL_CHOICE_STRINGS,"fast","slow",0,
                PANEL_NOTIFY_PROC,setrange,
                0);
        window_fit_height(controlpanel);
        fpanelhgt = (int) window_get(filepanel,WIN_HEIGHT);
        fpanelwdth = (int) window_get(filepanel,WIN_WIDTH);
        cpanelhgt = (int) window_get(controlpanel,WIN_HEIGHT);
        header_frame = window_create(base_frame,FRAME,
                WIN_SHOW,FALSE,0);
        header_panel1 = window_create(header_frame,PANEL,0);
        panel_create_item(header_panel1,PANEL_BUTTON,
                PANEL_LABEL_IMAGE,panel_button_image(header_panel1,"Done",0,0),
                PANEL_NOTIFY_PROC,done_proc,0);
        hdrfname = panel_create_item(header_panel1,PANEL_MESSAGE,
                PANEL_LABEL_STRING,"",   0);
        window_fit_height(header_panel1);
        header_panel2 = window_create(header_frame,TEXTSW,
                WIN_BELOW,header_panel1,
                WIN_X,0,
                WIN_ROWS,20,
                WIN_COLUMNS,60,
                TEXTSW_BROWSING,TRUE,
                TEXTSW_LINE_BREAK_ACTION,TEXTSW_WRAP_AT_CHAR,
                TEXTSW_DISABLE_CD,TRUE,
                TEXTSW_DISABLE_LOAD,TRUE,
                0);
        window_fit(header_frame);
}

fileinit(fp,fname,firsttime)

FILE *fp;
Boolean firsttime;
Filename fname;

{
        char *glmsg;
        Boolean ibinary,piped=FALSE;
        int method,doshift,herror,k,l,m,pos,count,*table,mcount,*mtable;
        register int i,j,fr;
        struct header hd2,hd3,hd4,hd5,*chd,hdi,hdo;
        struct hips_roi saveroi;
        struct stat buf;
        struct hips_stats stats;
        struct hips_stats mstats;
        Pixelval zero,scale;
        int gnelem,mean,var;
        int mgnelem,mmean,mvar;
        double gsum,gssq,gmean,gvar;
        double mgsum,mgssq,mgmean,mgvar;
        double entropy,h_entropy(),mentropy;

        validheader = FALSE;
        if (fread_header(fp,&hd,fname) == HIPS_ERROR) {
                fclose(fp);
                free_image(&hd);
                msg(hipserr);
                return(0);
        }
```

```
        if (image_shown) {
                mpr_d(mpr[0])->md_primary = 1;
                for (i=0;i<nframes;i++)
                        pr_close(mpr[i]);
                free(mpr);
                image_shown = FALSE;
        }
        ibinary = (hd.pixel_format == PFMSBF || hd.pixel_format == PFLSBF)
                ? TRUE : FALSE;
        frameno = hd.num_frame;
        nrows=hd.orows; ncols=hd.ocols; nframes = hd.num_frame;
        nframes=1;
        space = sfactor;
        for (space=1;space<=ncols;space=space*2)
                if (sfactor*space>=ncols)
                        break;
        pixelsize = space;
        if (firsttime) {
                for (i=0;i<frameno;i++)
                        storage[i]=-1;
                for (i=0;i<=tfactor;i++) {
                        if (qstep==0)
                                pos=qsequence[i];
                        else
                                pos=nsequence[i];
                        storage[pos]=1;
                }
        }
        obinary = (ibinary || (! greydisplay)) ? TRUE : FALSE;
        if ((fflag || cflag || gflag || sgflag) && ibinary && firsttime)
                perr(HE_IMSG,
                        "full color map for binary images is not meaningful");
        doshift = (!(fullsw || obinary));
        hd2.imdealloc = hd3.imdealloc = hd4.imdealloc = hd5.imdealloc = FALSE;
        hdi.imdealloc = hdo.imdealloc = FALSE;
        if ((mpr = (struct pixrect **)
                malloc(nframes*sizeof(struct pixrect *))) == 0) {
                        if (firsttime)
                                perr(HE_MSG,"can't allocate mpr");
                        else {
                                fclose(fp);
                                free_image(&hd);
                                msg("can't allocate mpr");
                                return(0);
                        }
        }
        getroi(&hd,&saveroi);
        clearroi(&hd);
        herror = 0;
        if (fseek(fp,0L,1) < 0)
                piped = TRUE;
#ifdef S_IFIFO
        if (!piped) {
                fstat(fileno(fp),&buf);
                if ((buf.st_mode & S_IFMT) == S_IFIFO)
                        piped = TRUE;
        }
#endif
        for (fr=0;fr<nframes;fr++) {
                if (hd.pixel_format == ourpackedtype && pixelsize == 1) {
                        if (!hd.imdealloc)
                                if (alloc_image(&hd) == HIPS_ERROR)
                                        {herror++;break;}
                        if (fread_image(fp,&hd,fr,fname) == HIPS_ERROR)
                                {herror++;break;}
                        chd = &hd;
                }
                else {
                        if (fr == 0)
                                if ((method = fset_conversion(&hd,&hd2,types,
                                    fname)) == HIPS_ERROR)
                                        {herror++;break;}
                        if (!hd2.imdealloc)
                                if (alloc_image(&hd2) == HIPS_ERROR)
                                        {herror++;break;}
                        chd = &hd2;
```

```
        if (piped || fseek(fp,next*hd.sizeimage,1) < 0) {
                for (m=0;m<next;m++)
                        if (fread_imagec(fp,&hd,&hd2,method,m,fname)
                            == HIPS_ERROR)
                                {herror++;break;}
        }
        if (fread_imagec(fp,&hd,&hd2,method,fr,fname)
            == HIPS_ERROR)
                {herror++;break;}
mtable = (int *) halloc(256,sizeof(int));
mgnelem = mgsum = mgssq = 0;
h_stats(&hd2,&mstats,FALSE);
if (mstats.nelem) {
        mgsum += mstats.sum;
        mgssq += mstats.ssq;
        mgnelem += mstats.nelem;
}
h_entropycnt(&hd2,mtable,FALSE);
mmean=mstats.mean;
mvar=mstats.var;
mcount = hd.rows*hd.cols;
mentropy = h_entropy(mtable,mcount,FALSE);
        if (doshift)
                if (h_shift_b(chd,chd,-2) == HIPS_ERROR)
                        {herror++;break;}
        if (h_shift_b(chd,chd,dfactor-8) == HIPS_ERROR)
                {herror++;break;}
        if (h_shift_b(chd,chd,8-dfactor) == HIPS_ERROR)
                {herror++;break;}
        if (space > 1) {
            if (fr == 0) {
                dup_headern(chd,&hdi);
                setformat(&hdi,PFINT);
                if ((!obinary) && ((ncols / space) & 01)) {
                        setsize(&hdi,(nrows / space),(ncols / space) + 1);
                        setroi(&hdi,0,0,hdi.orows,hdi.ocols - 1);
                }
                else
                        setsize(&hdi,(nrows / space),(ncols / space));
                dup_headern(&hdi,&hdo);
                setformat(&hdo,PFBYTE);
            }
            if (!hdi.imdealloc)
                        if (alloc_image(&hdi) == HIPS_ERROR)
                            {herror++;break;}
            if (!hdo.imdealloc)
                        if (alloc_image(&hdo) == HIPS_ERROR)
                            {herror++;break;}
            zero.v_int = 0;
            h_setimage(&hdi,&zero);
            h_reduce_bi(chd,&hdi,space,space);
            scale.v_int = space * space;
            h_divscale(&hdi,&hdo,&scale);
            chd = &hdo;
        }
        if (pixelsize > 1) {
            if (fr == 0) {
                dup_headern(chd,&hd3);
                if ((!obinary) && ((ncols * pixelsize/space) & 01)) {
                    if (setsize(&hd3,(nrows * pixelsize/space),
                        (ncols * pixelsize/space) + 1)
                        == HIPS_ERROR)
                            {herror++;break;}
                    if (setroi(&hd3,0,0,hd3.orows,hd3.ocols - 1)
                        == HIPS_ERROR)
                            {herror++;break;}
                }
                else
                    if (setsize(&hd3,(nrows * pixelsize/space),
                        (ncols * pixelsize/space)) == HIPS_ERROR)
                            {herror++;break;}
            }
            if (!hd3.imdealloc)
                if (alloc_image(&hd3) == HIPS_ERROR)
                        {herror++;break;}
```

```
                        if (interpolated == 1) {
                            if (h_enlarge_b(chd,&hd3,pixelsize,pixelsize)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                            }
                        else {
                            if (h_ienlarge_b(chd,&hd3,pixelsize,pixelsize)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                            }
                        chd = &hd3;
                    }
                    if (obinary) {
                            if (!ibinary)
                                    if (h_halftone(chd,chd) == HIPS_ERROR)
                                        {herror++;break;}
                            if (fr == 0) {
                                    dup_headern(chd,&hd4);
                                    if (setformat(&hd4,ourpackedtype) ==
                                        HIPS_ERROR)
                                        {herror++;break;}
                            }
                            if (!hd4.imdealloc)
                                    if (alloc_image(&hd4) == HIPS_ERROR)
                                        {herror++;break;}
#ifdef MSBFVERSION
                            if (h_btomp(chd,&hd4) == HIPS_ERROR)
                                    {herror++;break;}
#else
                            if (h_btolp(chd,&hd4) == HIPS_ERROR)
                                    {herror++;break;}
#endif
                            chd = &hd4;
                    }
                }
                if (obinary) {
                        if ((((chd->ocols)+7)/8) & 01) {
                            if (fr == 0) {
                                dup_headern(chd,&hd5);
                                if (setsize(&hd5,hd5.orows,hd5.ocols+8)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                                if (setroi(&hd5,0,0,hd5.orows,hd5.ocols-8)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                            }
                            if (!hd5.imdealloc)
                                    if (alloc_image(&hd5) == HIPS_ERROR)
                                        {herror++;break;}
#ifdef MSBFVERSION
                            if (h_copy_mp(chd,&hd5) == HIPS_ERROR)
                                    {herror++;break;}
#else
                            if (h_copy_lp(chd,&hd5) == HIPS_ERROR)
                                    {herror++;break;}
#endif
                            chd = &hd5;
                        }
                }
                else {
                        if ((chd->ocols) & 01) {
                            if (fr == 0) {
                                dup_headern(chd,&hd5);
                                if (setsize(&hd5,hd5.orows,hd5.ocols+1)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                                if (setroi(&hd5,0,0,hd5.orows,hd5.ocols-1)
                                    == HIPS_ERROR)
                                        {herror++;break;}
                            }
                            if (!hd5.imdealloc)
                                    if (alloc_image(&hd5) == HIPS_ERROR)
                                        {herror++;break;}
                            if (h_copy_b(chd,&hd5) == HIPS_ERROR)
                                    {herror++;break;}
                            chd = &hd5;
```

```
                }
        }
        table = (int *) halloc(256,sizeof(int));
        gnelem = gsum = gssq = 0;
        h_stats(chd,&stats,FALSE);
        if (stats.nelem) {
                gsum += stats.sum;
                gssq += stats.ssq;
                gnelem += stats.nelem;
        }
        h_entropycnt(chd,table,FALSE);
        mean=stats.mean;
        var=stats.var;
        count = hd.rows*hd.cols;
        entropy = h_entropy(table,count,FALSE);
        mpr[fr] = mem_point(ncols*pixelsize/space,nrows*pixelsize/space,obinary?1:8,
                (short *) chd->image);
        chd->imdealloc = FALSE;
}


fclose(fp);
free_image(&hd);
free_image(&hd2);
free_image(&hd3);
free_image(&hd4);
free_image(&hd5);
free_image(&hdi);
free_image(&hdo);
if (herror) {
        msg(hipserr);
        for (i=0;i<fr;i++)
                pr_close(mpr[i]);
        free(mpr);
        return(0);
}
setroi2(&hd,&saveroi);
validheader = image_shown = TRUE;
sprintf(str," Original frame:   Mean=%d  Variance=%d  Entropy=%f",
        mmean,mvar,(float) mentropy);
panel_set(currfile,PANEL_LABEL_STRING,str,0);
sprintf(str,"Displayed frame:   Mean=%d  Variance=%d  Entropy=%f",
        mean,var,(float) entropy);
panel_set(percent,PANEL_LABEL_STRING,str,0);
hmapvalid = FALSE;
if (findparam(&hd,"cmap") != NULLPAR) {
        hnumcol = 768;
        getparam(&hd,"cmap",PFBYTE,&hnumcol,&hred);
        if (hnumcol % 3) {
                if (firsttime)
                    perr(HE_MSG,"colormap length not a multiple of 3");
                else
                    msg("colormap length not a multiple of 3");
        }
        else {
                hnumcol /= 3;
                hgreen = hred + hnumcol;
                hblue = hred + 2*hnumcol;
                hmapvalid = TRUE;
        }
}
setcmap();
setfrm();
panel_set(dslider,
        PANEL_VALUE,dfactor,
        PANEL_MIN_VALUE,1,
        PANEL_MAX_VALUE,8,
        0);
panel_set(sslider,
        PANEL_VALUE,sfactor,
        PANEL_MIN_VALUE,1,
        PANEL_MAX_VALUE,ncols,
        0);
```

```
        panel_set(tslider,
                PANEL_VALUE,tfactor,
                PANEL_MIN_VALUE,0,
                PANEL_MAX_VALUE,frameno-1,
                0);
        if (greydisplay) {
                if (obinary)
                        glmsg = "GLs: 2";
                else if (fullsw)
                        glmsg = "GLs: 256";
                else
                        glmsg = "GLs: 64";
        }
        panel_set(numgls,PANEL_LABEL_STRING,glmsg,0);
        if (firsttime) {
                window_set(canvas,
                        CANVAS_WIDTH,ncols*pixelsize/space,
                        CANVAS_HEIGHT,nrows*pixelsize/space,
                        WIN_WIDTH,ncols*pixelsize/space,
                        WIN_HEIGHT,nrows*pixelsize/space,
                        CANVAS_DEPTH,obinary ? 1 : 8,
                        0);
                i = fpanelhgt + cpanelhgt + 28;
                j = (int) window_get(canvas,WIN_HEIGHT) + 21;
                k = fpanelwdth;
                l = (int) window_get(canvas,WIN_WIDTH);
                window_set(filepanel,
                        WIN_COLUMNS,75,
                        WIN_ROWS,greydisplay ? 15 : 4,
                        0);
                window_set(base_frame,
                        WIN_HEIGHT,((i>j) ? i : j),
                        WIN_WIDTH,k+l+15,
                        0);
        }
}

static void dfrom_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        dfactor = value;
        msg("change in depth resolution takes effect at next file load");
}

static void sfrom_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        int i;

        for (i=1;i<= 2*ncols;i=i*2) {
                if (i>value) {
                        sfactor = i/2;
                        break;
                }
        }
        msg("change in spatial resolution takes effect at next file load");
}

static void tfrom_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        int i,pos;

        tfactor = value;
```

```
        for (i=0;i<frameno;i++)
                storage[i]=-1;
        for (i=0;i<=tfactor;i++) {
                if (qstep==0)
                        pos=qsequence[i];
                else
                        pos=nsequence[i];
                storage[pos]=1;
        }
        msg("change in temporal resolution takes effect at next file load");
}

static void dset(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("change in depth bias takes effect at next file load");
        switch (value) {
        case 0:                         /* None */
                dstep = 0;
                break;
        case 1:                         /* Normal */
                dstep = 1;
                break;
        case 2:                         /* Fast */
                dstep = 2;
                break;
        case 3:                         /* Faster */
                dstep = 3;
                break;
        case 4:                         /* Fastest*/
                dstep = 4;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"Invalid range value, help!!?\n");
        }
}

static void sset(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("change in spatial bias takes effect at next file load");
        switch (value) {
        case 0:                         /* None */
                sstep = 0;
                break;
        case 1:                         /* Normal */
                sstep = 1;
                break;
        case 2:                         /* Fast */
                sstep = 2;
                break;
        case 3:                         /* Faster */
                sstep = 3;
                break;
        case 4:                         /* Fastest*/
                sstep = 4;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"Invalid range value, help!!?\n");
        }
}
```

```
static void tset(item,value,event)
Panel_item item;
unsigned int value;
Event *event;
{
        msg("change in temporal bias takes effect at next file load");
        switch (value) {
        case 0:                 /* None */
                tstep = 0;
                break;
        case 1:                 /* Normal */
                tstep = 1;
                break;
        case 2:                 /* Fast */
                tstep = 2;
                break;
        case 3:                 /* Faster */
                tstep = 3;
                break;
        case 4:                 /* Fastest*/
                tstep = 4;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"Invalid range value, help!!?\n");
        }
}

static void qset(item,value,event)
Panel_item item;
unsigned int value;
Event *event;
{
        msg("change in temporal bias takes effect at next file load");
        switch (value) {
        case 0:                 /* Quality */
                qstep = 0;
                break;
        case 1:                 /* Naive */
                qstep = 1;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"Invalid range value, help!!?\n");
        }
}

static void setoption(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        interpolated = value;
        msg("change in display option takes effect at next file load");
}

static void setbw(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        switch(value) {
        case 0:
                bandwidth = 64000;
                break;
        case 1:
                bandwidth = 56000;
                break;
        case 2:
                bandwidth = 19200;
                break;
```

```
        case 3:
                bandwidth = 9600;
                break;
        case 4:
                bandwidth = 4800;
                break;
        case 5:
                bandwidth = 2400;
                break;
        case 6:
                bandwidth = 1200;
                break;
        case 7:
                bandwidth = 300;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"invalid bandwidth value, help!!?!?");
        }
        msg("change in bandwidth takes effect at next file load");
}

static void progressive(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        int i;

        msg("");
        if (image_shown) {
                state = 0;
                for (i=0;i<16;i++) {
                        if (dfactor==qdepth[i])
                                level=i;
                }
                if (dir == 0)
                        next = -1;
                if (dir == 1)
                        next = frameno;
                mode = 0;
                fps = 8;
                panel_set(speedslider,
                        PANEL_LABEL_STRING,"Frames/sec",
                        PANEL_MIN_VALUE,1,
                        PANEL_MAX_VALUE,fps+1,
                        PANEL_VALUE,fps,
                        0);
                panel_set(rangeitem,
                        PANEL_VALUE,0);
                running = TRUE;
                runonce = FALSE;
                transmit = TRUE;
                settimer();
        }
}

static void nonprogressive(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        int i;

        i = dfactor*sfactor*sfactor*(tfactor+1)/bandwidth;
        sprintf(str,"Non-progressive transmission will take about %d seconds.",i);
        msg(str);
        if (image_shown) {
                if (dir == 0)
                        next = -1;
                if (dir == 1)
                        next = frameno-1;
```

```
                i = dfactor*sfactor*sfactor/bandwidth;
                if (i >= 1) {
                        mode = 1;
                        spf = i;
                        panel_set(speedslider,
                                PANEL_LABEL_STRING,"Seconds/frame",
                                PANEL_MIN_VALUE,1,
                                PANEL_MAX_VALUE,spf+1,
                                PANEL_VALUE,spf,
                                0);
                        panel_set(rangeitem,
                                PANEL_VALUE,1);
                }
                if (i < 1) {
                        mode = 0;
                        fps = bandwidth/(dfactor*sfactor*sfactor);
                        panel_set(speedslider,
                                PANEL_LABEL_STRING,"Frames/sec",
                                PANEL_MIN_VALUE,1,
                                PANEL_MAX_VALUE,fps+1,
                                PANEL_VALUE,fps,
                                0);
                        panel_set(rangeitem,
                                PANEL_VALUE,0);
                }
                running = TRUE;
                runonce = FALSE;
                transmit = FALSE;
                settimer();
        }
}

static void displayit(canv,pw,repaint_area)

Canvas canv;
Pixwin *pw;
Rectlist *repaint_area;

{
        if (image_shown)
                pw_rop(pw,0,0,ncols*pixelsize/space,nrows*pixelsize/space,PIX_SRC,
                        mpr[0],0,0);
}

static void speed_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        if (mode)
                spf = value;
        else
                fps = value;
        settimer();
}

static void run_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        if (image_shown) {
                running = TRUE;
                runonce = FALSE;
                transmit = FALSE;
                settimer();
        }
}
```

```
static void runonce_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;
{
        FILE *fp;
        int i;
        msg("");
        if (image_shown) {
                running = TRUE;
                runonce = TRUE;
                transmit = FALSE;
                frcount=0;
                if (dir == 0)
                        next = 0;
                else {
                        for (i=frameno-1;0<=i;i--)
                                if (storage[i] != -1)
                                        break;
                                next = i;
                }
                if (storage[next] != -1) {
                        if ((fp = fopen(filename,"r")) == NULL) {
                                msg("can't open file");
                                return;
                        }
                        fileinit(fp,filename,FALSE);
                        displayit(canvas,canpixwin,(Rectlist *) 0);
                }
                settimer();
        }
}

static void stop_proc(item,value,event)
Panel_item item;
unsigned int value;
Event *event;
{
        msg("");
        if (image_shown) {
                running = FALSE;
                settimer();
        }
}

static void step_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        FILE *fp;
        int i;

        msg("");
        if (image_shown) {
                running = FALSE;
                runonce = FALSE;
                transmit = FALSE;
                if (dir == 0) {
                        next = (next + 1) % frameno;
                        for (i=0;i<frameno;i++)
                                if (storage[next] == -1)
                                        next = (next + 1) % frameno;
                }
                else if (dir == 1) {
                        for (i=0;i<frameno;i++) {
                                next--;
                                if (next < 0)
                                        next =  frameno - 1;
                                if (storage[next] != -1)
                                        break;
                        }
                }
```

```
                if (storage[next] != -1) {
                        if ((fp = fopen(filename,"r")) == NULL) {
                                msg("can't open file");
                                return;
                        }
                        fileinit(fp,filename,FALSE);
                        displayit(canvas,canpixwin,(Rectlist *) 0);
                        settimer();
                }
        }

}

static void setdir(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        switch(value) {
        case 0:                 /* forward */
                dir = 0;
                break;
        case 1:                 /* backward */
                dir = 1;
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"invalid dirchoice value, help!!?!?");
        }
}

static void nextframe(client,itimer_type)

Notify_client client;
int itimer_type;

{
        FILE *fp;
        int i,j,pos;

        if (image_shown) {
                if (runonce && (frcount == tfactor)) {
                        frcount = 0;
                        running = FALSE;
                        runonce = FALSE;
                        settimer();
                        return;
                }
                if (dir == 0) {
                        next = (next + 1)%frameno;
                        for (i=0;i<frameno;i++)
                                if (storage[next] == -1)
                                        next = (next + 1) % frameno;
                }
                else if (dir == 1) {
                        for (i=0;i<frameno;i++) {
                                next--;
                                if (next < 0)
                                        next =  frameno - 1;
                                if (storage[next] != -1)
                                        break;
                        }
                }
                if (transmit && (next==0) ) {
                        if (state == 0) {
                                if (qstep == 0) {
                                        newd=qdepth[level];
                                        news=qspatial[level];
                                        newt=tfactor;
                                        if ((dstep > dcount) || (sstep > scount)) {
                                                dcount++;
                                                scount++;
                                                level++;
```

```
                        newd=qdepth[level];
                        news=qspatial[level];
                }
                if (tstep > tcount) {
                        tcount++;
                        newt = tfactor + 1;
                }
                if ((dstep <= dcount) && (sstep <= scount)
                        && (tstep <= tcount)) {
                        dcount=0;
                        scount=0;
                        tcount=0;
                }
                if (level > 15)
                  level=15;
        }
        else {
                newd = dfactor;
                news = sfactor;
                newt = tfactor;
                if (dstep > dcount) {
                        dcount++;
                        newd = dfactor + 1;
                }
                if (sstep > scount) {
                        scount++;
                        news = sfactor*2;
                }
                if (tstep > tcount) {
                        tcount++;
                        newt = tfactor + 1;
                }
                if ((dstep <= dcount) && (sstep <= scount)
                        && (tstep <= tcount)) {
                        dcount=0;
                        scount=0;
                        tcount=0;
                }
        }
        if (newd > 8)
                newd = 8;
        if (news > ncols)
                news = ncols;
        if (newt > frameno-1)
                newt = frameno-1;
        delay = (newd*news*news*(newt+1) -
                dfactor*sfactor*sfactor*(tfactor+1))/bandwidth;
}
state ++;
if (mode == 0) {
        if (state >= delay*fps/(tfactor+1)) {
                dfactor = newd;
                sfactor = news;
                tfactor = newt;
                for (i=0;i<frameno;i++)
                        storage[i]=-1;
                for (i=0;i<=tfactor;i++) {
                        if (qstep==0)
                                pos=qsequence[i];
                        else
                                pos=nsequence[i];
                        storage[pos]=1;
                }
                state = 0;
        }
}
if (mode == 1) {
        if (state >= delay/(spf*(tfactor+1))) {
                dfactor = newd;
                sfactor = news;
                tfactor = newt;
                for (i=0;i<frameno;i++)
                        storage[i]=-1;
                for (i=0;i<=tfactor;i++) {
                        if (qstep==0)
                                pos=qsequence[i];
```

```
                                        else
                                                pos=nsequence[i];
                                        storage[pos]=1;
                                }
                                state = 0;
                        }
                }
                if ((dfactor == 8) && (sfactor == ncols) && (tfactor == frameno-1))
                        transmit = FALSE;
                i = (100*dfactor*sfactor*sfactor)/(8*ncols*ncols)
                    *(tfactor+1)/frameno;
                j = dfactor*sfactor*sfactor*(tfactor+1)/bandwidth;
                sprintf(str,"%d percent of bits has been displayed in %d seconds.",i,j);
                msg(str);
        }
        if (storage[next] != -1) {
                if ((fp = fopen(filename,"r")) == NULL) {
                        msg("can't open file");
                        return;
                }
                fileinit(fp,filename,FALSE);
                displayit(canvas,canpixwin,(Rectlist *) 0);
        }
        frcount++;
        if (frcount > tfactor) {
                frcount=0;
        }
    }
    else
        settimer();     /* to clear the timer, just in case */
}

setfrm()

{
    sprintf(str,"Frame No: %d",next);
    panel_set(frameslider,PANEL_LABEL_STRING,str,0);
}

settimer()

{
    if (!image_shown)
        running = FALSE;
    if ((!running) || iconic) {
        frame_timer.it_interval.tv_usec = 0;
        frame_timer.it_interval.tv_sec = 0;
        frame_timer.it_value.tv_usec = 0;
        frame_timer.it_value.tv_sec = 0;
    }
    else if (mode) {
        frame_timer.it_interval.tv_usec = 0;
        frame_timer.it_interval.tv_sec = spf;
        frame_timer.it_value.tv_usec = 0;
        frame_timer.it_value.tv_sec = spf;
    }
    else if (fps == 1) {
        frame_timer.it_interval.tv_usec = 0;
        frame_timer.it_interval.tv_sec = 1;
        frame_timer.it_value.tv_usec = 0;
        frame_timer.it_value.tv_sec = 1;
    }
    else {
        frame_timer.it_interval.tv_usec = 1000000 / fps;
        frame_timer.it_interval.tv_sec = 0;
        frame_timer.it_value.tv_usec = 1000000 / fps;
        frame_timer.it_value.tv_sec = 0;
    }
    notify_set_itimer_func(base_frame,nextframe,ITIMER_REAL,
        &frame_timer,ITIMER_NULL);
}
```

```
static void setrange(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        switch (value) {
        case 0:                 /* fast */
                mode = 0;
                panel_set(speedslider,
                        PANEL_LABEL_STRING,"Frames/sec",
                        PANEL_VALUE,fps,
                        0);
                break;
        case 1:                 /* slow */
                mode = 1;
                panel_set(speedslider,
                        PANEL_LABEL_STRING,"Seconds/frame",
                        PANEL_VALUE,spf,
                        0);
                break;
        default:
                hipserrprt = hipserrlev = HEL_ERROR;
                perr(HE_MSG,"Invalid range value, help!!?\n");
        }
        settimer();
}

msg(s)

char *s;

{
        panel_set(message,PANEL_LABEL_STRING,s,0);
}

static void done_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        window_set(header_frame,WIN_SHOW,FALSE,0);
        textsw_reset(header_panel2,0,0);
}

static void header_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        static char invhdtext[] = "Current header contents invalid.\n\
Reload this HIPS file and try again.";
        char *hstring,*formatheader();

        if (image_shown) {
                textsw_reset(header_panel2,0,0);
                if (validheader)
                        hstring = formatheader(&hd);
                else
                        hstring = invhdtext;
                textsw_replace_bytes(header_panel2,0,TEXTSW_INFINITY,
                        hstring,strlen(hstring));
                textsw_normalize_view(header_panel2,0);
                panel_set(hdrfname,PANEL_LABEL_STRING,
                        panel_get(currfile,PANEL_LABEL_STRING),0);
                window_set(header_frame,WIN_SHOW,TRUE,0);
        }
}
```

```
static Notify_value resett(frame,status)

Frame frame;
Destroy_status status;

{
        window_set(frame,FRAME_NO_CONFIRM,FALSE,0);
        window_set(header_frame,WIN_SHOW,FALSE,0);
        textsw_reset(header_panel2,0,0);
        return(notify_next_destroy_func(frame,status));
}

static Notify_value catchclose(frame,event,arg,type)

Frame frame;
Event *event;
Notify_arg arg;
Notify_event_type type;

{
        Notify_value value;

        value = notify_next_event_func(frame,event,arg,type);
        iconic = (int) window_get(frame,FRAME_CLOSED);
        settimer();
        return(value);
}

static void quit_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        window_destroy(base_frame);
}

gammalut()

{
        int i;
        double gr,gg,gb;

        gr = 1./gammar;
        gg = 1./gammag;
        gb = 1./gammab;
        gred[0] = ggreen[0] = gblue[0] = 0;
        for (i=1;i<256;i++) {
                gred[i] = 255.*pow((double) i/255.,gr) + .5;
                ggreen[i] = 255.*pow((double) i/255.,gg) + .5;
                gblue[i] = 255.*pow((double) i/255.,gb) + .5;
        }
        sgred[0] = sggreen[0] = sgblue[0] = 0;
        for (i=1;i<64;i++) {
                sgred[i] = 255.*pow((double) i/63.,gr) + .5;
                sggreen[i] = 255.*pow((double) i/63.,gg) + .5;
                sgblue[i] = 255.*pow((double) i/63.,gb) + .5;
        }
        sprintf(gmapname,"gammar%fg%fb%f",gammar,gammag,gammab);
        sprintf(sgmapname,"sgammar%fg%fb%f",gammar,gammag,gammab);
}

setcmap()

{
        if (obinary) {
                pw_setcmsname(canpixwin,"sunvb");
                pw_putcolormap(canpixwin,0,2,bred,bgreen,bblue);
        }
        else if (cmaptype == 1 || (cmaptype == 0 && !hmapvalid)) { /* ramp */
                if (fullsw) {
                        pw_setcmsname(canpixwin,"sunvf");
                        pw_putcolormap(canpixwin,0,256,rred,rgreen,rblue);
                }
                else {
```

```
                        pw_setcmsname(canpixwin,"sunv");
                        pw_putcolormap(canpixwin,0,64,srred,srgreen,srblue);
                }
        }
        else if (cmaptype == 0) {       /* header map */
                pw_setcmsname(canpixwin,hmapname);
                pw_putcolormap(canpixwin,0,hnumcol,hred,hgreen,hblue);
        }
        else if (cmaptype == 2) {       /* gamma map */
                if (fullsw) {
                        pw_setcmsname(canpixwin,gmapname);
                        pw_putcolormap(canpixwin,0,256,gred,ggreen,gblue);
                }
                else {
                        pw_setcmsname(canpixwin,sgmapname);
                        pw_putcolormap(canpixwin,0,64,sgred,sggreen,sgblue);
                }
        }
        else {  /* file map */
                pw_setcmsname(canpixwin,cmapname);
                pw_putcolormap(canpixwin,0,cnumcol,cred,cgreen,cblue);
        }
        if (image_shown)
                displayit(canvas,canpixwin,(Rectlist *) 0);
}

static void gammar_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;
{
        msg("");
        gammar = ((double) value)/100.;
        gammalut();
        if (cmaptype == 2)
                setcmap();
}

static void gammag_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        gammag = ((double) value)/100.;
        gammalut();
        if (cmaptype == 2)
                setcmap();
}

static void gammab_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        gammab = ((double) value)/100.;
        gammalut();
        if (cmaptype == 2)
                setcmap();
}

static void setpixrange(item,value,event)
Panel_item item;
unsigned int value;
Event *event;

{
        fullsw = (value == 1);
        msg("change in greylevel usage takes effect at next file load");
}
```

```
static void setcmapchoice(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        msg("");
        cmaptype = value;
        if (obinary)
                msg("color map ignored for binary image");
        else
                setcmap();
}

static void loadcmap_proc(item,value,event)

Panel_item item;
unsigned int value;
Event *event;

{
        Filename fname;

        msg("");
        fname = (Filename) panel_get_value(fileitem);
        if (strlen(fname) == 0) {
                msg("null file name");
                return;
        }
        if (readcmap(fname,256,&cnumcol,cred,cgreen,cblue) == HIPS_ERROR) {
                msg(hipserr);
                cmapvalid = FALSE;
                panel_set(maptypeitem,
                        PANEL_CHOICE_STRINGS,"Hdr/Ramp","Ramp","Gamma",0,0);
                if (cmaptype == 3) {
                        cmaptype = 0;
                        panel_set(maptypeitem,PANEL_VALUE,0,0);
                }
        }
        else {
                cmapvalid = TRUE;
                panel_set(maptypeitem,
                        PANEL_CHOICE_STRINGS,"Hdr/Ramp","Ramp","Gamma","File",0,
                        PANEL_VALUE,3,
                        0);
                cmaptype = 3;
        }
        cmapname = strsave(fname);
        if (obinary)
                msg("color map ignored for binary image");
        else
                setcmap();
}
```