A LOSSY IMAGE COMPRESSION ALGORITHM BASED ON NONUNIFORM SAMPLING AND INTERPOLATION OF THE IMAGE INTENSITY SURFACE

BY

CHARLES JOSEPH ROSENBERG

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT FOR THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE IN COMPUTER SCIENCE

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER 1990

Signature of Author _____

Department of Electrical Engineering and Computer Science
September 1990

Certified by _____

Dr. Walter Bender
Principal Research Scientist, Media Laboratory

Certified by _____

Ms. Jeanne Wiseman
Hewlett-Packard Laboratories

Accepted by _____

Chairman
Departmental Graduate Committee
Department of Electrical Engineering and Computer Science

Abstract

This thesis deals with lossy digital image compression. The goal of this work is to develop an algorithm which allows for high quality image reconstructions with low computational complexity. Because of complexity requirements a spatial domain technique was focused on. This technique is an extension of fan based redundancy reduction style compression algorithms developed in the late 1960's. These algorithms nonuniformly sample the image data. More samples are taken where the data is more complex and less where it is simpler. Upon reconstruction samples which were not retained during the compression process are interpolated from retained samples. Accordingly, the algorithm developed here has been dubbed nonuniform sampling and interpolation, NSI.

Throughout the course of this thesis a number of extensions were are to algorithms developed in the 1960's to increase decompressed image quality without increasing decompression complexity. The three major contributions are: an improved sample selection criterion through a sum of squared error metric, improved edge fidelity through "sample point jittering" and the incorporation of two dimensional correlation by alternate scan path processing of the image data. This work consists of a description of a series of experiments which explore a variety of algorithm extensions. At each point along the way image quality is evaluated through a statistical measure, peak signal to noise ratio (PSNR) measured in decibels, described in the body of the thesis. A compression algorithm based on the discrete cosine transform (DCT) was used as a basis for comparison, because of it's acknowledged high image quality and acceptance as an international image compression standard by JPEG.

As a result of this investigation an algorithm was developed which performs very close in terms of quality to that of a DCT, being only 2.5 dB worse. Yet decompressing an image 24 times faster and compression an image 48 times slower. This algorithm is highly suitable for situations where an image is more often decompressed than compressed and where the decompression is to take place on a low power compute platform.

Acknowledgements

I would like to thank Hewlett-Packard, the Image Peripherals Department, and the VI-A program for supporting me and giving me the opportunity to perform this work. At Hewlett-Packard there is a whole bunch of people that I want to thank. I would especially like to thank Vasudev Bhaskaran, our local image compression guru for the enormous amount of help and extremely insightful suggestions he had during the course of my work. I would also like to thank Thomas Malzbender and Helena Trontelj, both at HP, for advice and listening to hair-brain schemes. I'd like to thank my supervisor Jeanne Wiseman for taking me into her group after my first unsuccessful summer at HP. I'd also like to thank Joyce Farrell for helping design the subjective test and carrying out the ensuing data analysis. And thanks to Ricardo Motta for lending us his lab space for the experiments.

I'd also like to mention my roommates from the 6A program in the summer of '89: Brad, Stone, Thuan, and Yonald. They provided moral support and companionship and endless patience in looking at various printouts of "Lena."

I want to thank Walter Bender at the Media Lab for saying *"yes"* to advising my thesis and ending the dreaded 6A advisor quest.

I want to thank my parents for getting me to where I am and helping me through MIT.

Table of Contents

List of Tables

List of Figures

List of Graphs

List of Photographs

1. Introduction

1.1 Overview

This thesis deals with digital image compression. Throughout this thesis a lossy image compression algorithm is developed by building on work done in redundancy reduction algorithms in the late 1960's. The various parameters of the algorithm are tested and evaluated. The result is an image compression algorithm, dubbed NSI, for nonuniform sampling and interpolation, which can decode images extremely quickly and has image quality which rivals that of a more popular algorithm, the discrete cosine transform. These attributes make the algorithm extremely well suited for use on a personal computer.

1.2 Description of Image Compression

More and more frequently images are stored in digital form. The most standard way to store an image is on a pixel-by-pixel basis. In this format, one number is used to represent the intensity of a particular pixel. In the case of a color image a pixel is most often represented as three numbers, one for red, one for green and one for blue. Typical image sizes are 512 pixels on a side or 1024 pixels on a side. This means that a typical black and white image can require 256 kilobytes or 1 megabyte of storage space and three times that amount for a color image. In view of these figures it is obvious that storage and transmission of these images is expensive.

Compression provides a means by which the amount of information needed to represent an image can be reduced. Since the data in these images do not come from completely random sources, there is a certain amount of predictability in the data which can be exploited to reduce the amount of data needed to represent an image. The amount of information which is contained in a signal is called its entropy and is defined by the following equation: [Shannon and Weaver 63]

$$\text{Entropy} = H(f) = -\sum_{i=1}^{n} P(f_i)\log_2 P(f_i) \qquad [1.2\text{-}1]$$

In this equation $P(f_i)$ is the probability that a particular symbol f_i will be encountered in the signal. The higher the probability that symbol is encountered, the lower the contribution in bits per symbol to the overall entropy of the signal. The entropy of a signal, H(f), is the average number of bits per symbol needed to encode the signal, in the case of images it is bits per pixel. A totally random black and white image with 256 brightness levels would require eight bits per pixel. Whereas an image which was totally blank would require zero bits per pixel. Entropy encoding algorithms like Huffman and Lempel-Ziv coding attempt to extract the symbol probability distribution from the signal. The problem is that these algorithms can never be one hundred

percent efficient and can only encode an image close that of its entropy. Table [1.2-1] summarizes the theoretical entropy per pixel and the entropy encoding performance for three 8.0 bit per pixel images which are referred to later in this document:

Image name	First Order Entropy	Huffman compression	Lempel-Ziv compression	Higher Order Entropy
Lena	7.48	7.49	7.04	5.24
Tiff	6.63	6.64	6.23	5.03
Smag	7.43	7.44	7.54	6.49

Table [1.2-1]: The entropy and entropy encoding results of various images 8.0 bit per pixel images.

Lempel-Ziv actually does a little better than Huffman coding or the actual entropy of the signal because it exploits higher order entropy. Higher order entropy uses conditional probability to calculate the probability of the next symbol. Therefore the probability of the occurrence of a particular symbol changes depending on what pixels have occurred before it in a stream of data. The higher order entropy column in Table [1.2-1] uses the pixel above to help predict the pixel value. The problem is that even exploiting higher order entropy results in a relatively low compression rate, about 35% savings for typical images.

This lower bound imposed by entropy is only related to a lossless reconstruction of the image data. A lossless reconstruction means that the original image data is reconstructed exactly, bit for bit. This is an overly strict requirement in certain situations and it turns out by loosening this requirement it is possible to greatly improve the compressibility of image data. This class of algorithms are called a lossy compression algorithm because they modifies the original image data so as to make it more compressible.

What types of data modifications are "acceptable?" The first class are modifications which are visually imperceptible. In most cases the primary use of image data is for viewing. If this is the case, then various aspects of the human visual system can be taken into account. This allows certain parts of image information to be changed without a visually perceived loss in information. Images coded in this manner are considered visually lossless. Another part of image data which can be removed is noise. Most digital images originate from some digitization process. This process usually introduces noise into the image data. If the noisy part of the signal can be identified and removed then a useless part of the signal with a high information content can be eliminated. Noise can have a large apparent information content because of its unpredictability which increases it entropy. It is also possible to distort the image in some perceptible way that is

not particularly objectionable for a specific application. In a situation where the compressed image is only used for browsing and identification purposes, the image may contain many distortions and still be acceptable.

The usefulness of a lossy compression algorithm in many situations is obvious. The high compression rate achievable by the use of a lossy algorithm can reduce storage requirements and data communications time significantly. The two desired characteristics for such an algorithm are that it achieve highest possible quality reconstructions and the fastest possible coding and encoding. Many lossy compression algorithms exist and each has its own advantages and disadvantages.

The limited storage capacity available to microcomputers makes lossy image compression extremely attractive in this domain. Even a 100 megabyte hard drive can only store thirty-three 1024 x 1024 color images. The need has recently become more apparent because of the decrease in the cost of high quality frame buffers. The major limiting factor in the application of compression methods in this domain is limited system processing power. This makes the computational complexity of the compression algorithm a major factor in its usability in the microcomputer domain. The alternative is to use a computationally demanding algorithm and supply expensive hardware specific support for the algorithm.

2. Background

Many algorithms have been developed to perform lossy image compression. All of these algorithms attempt to exploit some regularities in the input data in order to achieve coding efficiency. There are two broad classes of algorithms for lossy image compression. The first class are transform domain techniques. These techniques first perform a linear transformation of the image data before attempting to code it. The other class of algorithms are spatial domain techniques. These techniques exploit local inter-pixel correlation directly to achieve compression.

2.1 Transform based techniques

Transform based techniques linearly transform the image so as to concentrate as much energy as possible into as few transform coefficients as possible. This in and of itself does not provide any compression but allows the many small coefficients to be approximated by zero. These techniques usually perform extremely well. This is because many sections of natural scenes are relatively uniform and will not have very many high frequency components. These techniques can also be easily adapted to take advantage of characteristics of the human visual system's frequency response by quantizing the different coefficients corresponding to various frequency ranges to different accuracies.

The major disadvantage of these techniques is their computational complexity. The two dimensional transformation of an image whose dimensions are n x n pixels takes on the order of $n\log_2 n$ real multiplications and additions per pixel.[Pratt 78] This would make it prohibitively expensive to calculate the two dimensional transform of large pictures. A number of techniques have been used to make this number more reasonable. The first is the use of separable transforms. This reduces the calculations to taking all the horizontal and then all of the vertical transformations of the image. The complexity of this operation is order $\log_2 n$. The second technique exploits the local nature of image correlation. This implies that the parts of the image which are not spatially adjacent have very little to do with one another. This is taken advantage of by breaking the image up into blocks and transforming each of these blocks separately. If block size is kept constant over the entire image then complexity becomes linear in the number of pixels in the image.

One aspect of transform domain techniques are their encoder-decoder symmetry. Since performing a forward and inverse transform are equally complex, compression time is roughly equal to decompression time.

The preferred transform for this type for image coding has been the discrete cosine transform (DCT) because of its good performance. One reason for its good performance is that its

windowed transform implies even symmetry. Even symmetry makes the signal continuous at the boundaries of the image tiles so that sharp discontinuities not created which can introduce spurious high frequency components and reduce compression rate. The DCT is usually computed for 8 x 8 blocks of the image and can be computed as a separable transform. This places its complexity at 3 multiplications and 3 additions per pixel.[Makhoul 80, Feig 90]

Recently a committee of the CCITT, the joint photographic experts group (JPEG), has chosen the DCT as the world standard for continuous image compression. This decision was based on the high quality reconstructions, form both subjective and objective evaluations, possible with the DCT and the "reasonable" computationally complexity.[Hudson and Yasuda and Sebestyen 88, Laeger and Joan and Yamazaki 88, Wallace and Vivian and Poulsen 88] The existence of this standard is triggering the development hardware and software to perform a discrete cosine transform quickly. In the hardware arena a company named C-cubed Microsystems located in San Jose California is producing a integrated circuit which can compress or decompress an image at 10 million color pixels per second. This is more than fast enough to encode and decode NTSC video in real time. In terms of software and algorithms, IBM has recently developed an algorithm which can perform the DCT in 0.84 multiplies and 7.2 additions per pixel.[Feig 90] The fact that IBM is patenting this algorithm may limit its widespread use. The complexity of the DCT still puts it beyond software implementations with reasonable response time on low end compute platforms.

The DCT does have certain failings. One is the fact that the image is processed in a block-wise manner can create artifacts where these blocks are visible in the image. The other effect is that often an image compressed by the DCT looks low passed filtered. This is because, most often, the high frequency coefficients are truncated to zero. This tends to have the effect of blurring sharp edges in the image. A more thorough analysis of DCT image quality is given in a later section of this thesis.

2.2 Spatial techniques

These are techniques which operate on the image in its original form. They attempt to compress the image by taking advantage of spatial correlation between the pixels in the image.

One of the simplest of these techniques is subsampling. This technique simply throws away a certain percentage of the image pixels. One version of this algorithm first low pass filters the image and then resamples it. The problem with this technique is that the high frequency portions of the image are removed. A variant on this technique just resamples the image without first filtering it. This reduces the computational complexity of the algorithm but introduces aliasing

artifacts in high frequency regions of the image and at sharp edges. Subsampling actually performs amazingly well for simple images with large uniform areas.

Another spatial technique is block truncation coding. This technique divides an image up into blocks and process each block of the image individually. The algorithm chooses two representative pixel values for each block such that when it assigns one of the two values to each pixel in the block, the mean and variance of the coded block is approximately the same as the original image block. The simplicity of this algorithm allows for very fast encoding and decoding. The are two major artifacts generated by the algorithm. The first is aliasing. This occurs because only two intensity values can be used near high contrast edges, so the anti-aliasing effect of a continuous grayscale are eliminated. The other artifact is blockiness.

Yet another technique is vector quantization. This technique divides the image up into blocks and treats each block as if it is a vector. The pixel values are the components of that vector. Compression of the image consists of calculating a small set of all possible vectors, called a code book, which can adequately represent all of the vectors which do occur in the image. Each block in the image is then assigned an index to the entry in the code book which best approximates it. Decoding of the image consists of placing the correct codebook entries in the correct places in the image. Code book generation and searching and can make this algorithm extremely expensive in the compression phase. (Code book generation overhead can be reduced by creating a "universal" code book, but usually such a code book does not allow as great a compression rate as a specific code book). The table look up nature of decompression makes this phase extremely fast. This algorithm has been found to perform extremely well. One of its major disadvantages is the use of the codebook. Code book searching and storage can pose problems because code books can get to be extremely large. This results from the large number of possible image vectors. Another problem is that certain details in the image may not be well represented by a closest vector. "Universal" code book usage can make this problem worse.

2.3 Redundancy reduction

A variant on subsampling is a set of algorithms which perform redundancy reduction. This class of algorithms adaptively samples the image. They were mainly developed and investigated in the late 1960's. These algorithms process the image on a scan line by scan line basis placing sample points such that the reconstructed scan line is with in a particular error bound of the original data. Many of these algorithms used simple linear interpolation to reconstruct missing samples during decompression.

The are a number of variants of redundancy reduction, but the version which will be used in this thesis operates as follows: The process of choosing a sample point consists of the

following steps. First, the starting point of the segment is defined. This point can be the first point on the line or the end point of the previous segment. An approximating segment is then extended from that point, pixel by pixel. At each step the error is calculated between the original image data and an approximation of that data based on the current position of the segment end point. When a error metric threshold is exceeded a sample point is placed immediately before the point which exceeded the threshold. In this way, the approximation is guaranteed to be within a certain error bound (the error metric threshold) of the original image data. The new end point forms the start point of the next approximating segment. This process is continued until the end of a scan line is reached. (The last sample on every line segment must always be taken.) This scan line process is repeated until all of the lines of the image have been processed. This version of redundancy reduction is often referred to as the fan based algorithm. Figure [2.3-1] is illustrative of this process. In this Figure an intensity waveform, where the vertical axis is intensity and the horizontal axis is across the scanline, is to be approximated. Point P0 is chosen as the first point on the curve. Points P1 and P2 are later chosen such that the dotted segments which will approximated the curve will not result in an approximation error above threshold.

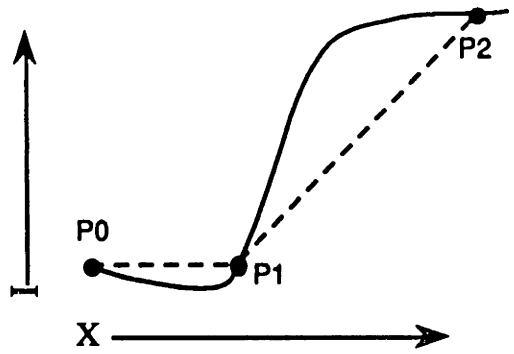


Figure [2.3-1]: An approximation is made to a curve by choosing sample points P0, P1, and P2.

A large part of the work performed in the sixties dealt with evaluating the performance of different redundancy reduction algorithms given models of the source data. In [Davisson 68] an analysis is performed comparing fan based redundancy reduction to run length encoding for a lossless encoding of a source signal. Based on a statistical model of the source, the author found that in certain instances, when the data is generated by a first order Markov process, that run length encoding out performs the fan algorithm. The stipulation that an exact reconstruction of the original signal be made did not allow the author to analyze all of the benefits of the fan-based algorithm.

In [Ehrman 67] a comparison is made of three algorithms, a floating aperture predictor, a zero-order interpolator, and a fan based interpolator. A floating aperture predictor is basically a run

length encoder. A zero order interpolator is different in that the pixel value which is replicated is optimally chosen and is not bound to be on the signal itself. In all three cases a maximum deviation error measure was used. The expected time between sample points for a given error is calculated for a source modelled as a first order Markov process and compression efficiency is compared to a PCM algorithm. The analysis showed the fan algorithm to perform better then the floating aperture predictor or the zero order interpolator. The fan algorithm was found to perform comparably to PCM based uniform sampling. However, this is only the case where the bandwidth of the signal to be compressed is known. It could be calculated on the fly for a PCM encoder but this would seem to greatly increase its complexity. The author suggests in this situation a fan based algorithm is a better choice.

In [Kortman 67] a fan-based algorithm is compared to run length encoder style algorithms in a satellite telemetry compression task. The fan-based algorithm is found to perform the best.

In [Gardenhire '64] three methods were evaluated: the step method, the two point projection method (the first two sample points define the slope of the interpolated line), and the fan method. The results were that the fan method worked best. The two point projection method was found to be extremely sensitive to noise. The only case where the step method seem to function better than the fan method was when the data being coded was of a binary nature, like that generated by switch closures.

One of the nice aspects of redundancy reduction techniques is that decompression is extremely simple. The sample point just needs to be placed in the image and the pixels between are then reconstructed by linear interpolation.

The major disadvantage of these algorithms is that they process the data in a one dimensional fashion. Accordingly, compression rate is low because the two dimensional structure of the image is not extracted. Also, because each scan line is processed independently, the image tends to stripey with smearing occuring in the direction of the scan line processing.

Some more current work is extremely similar to this early redundancy reduction work. [Wallach 86] describes an algorithm which is "inspired" by fractal geometry. The algorithm uses a line segment of a fixed length. This segment is pivoted until it intersects the image intensity waveform. (In practice this operation can be performed via a table look up.) The compressed image is then made up of the horizontal distance traversed by the line and its sign. With this information a piecewise linear approximation of an image can be formed. The author gives no precise image quality measures , image quality is just described as being "excellent."

[Yang and Wu and Mills 88] extend this method by incorporating the Peano scan to take advantage of two dimensional correlation and by keeping a fixed set of yardsticks, whose choice is modulated based on local image statistics. [Goel and Kwatra 88] integrate two dimensional correlation through the use of DPCM in the vertical direction. Color images were also tested and

non-edge regions of the image were subsampled by 2:1. The claim is that good subjective quality color images can still be achieved at 1.2 bits per pixel even though the measured objective quality is low.

Related recent work is on the polygonal decomposition of curves. The goal of this work is to reduce the amount of data needed to represent a two dimensional sampled curve. [Sklansky 80] presents an algorithm which utilizes a scan along method which reduces the complexity of finding those points which fall within a specified error tolerance. This same method is applied by [Fowell and McNeil 89] for the compression of plot data. [Wall and Danielsson 83] have devised an algorithm which uses the error area generated for determining when a new sample point is necessary.

There are two parts to the data created by a redundancy reduction algorithm. The first part is the sample values themselves. The second part is the distances between these samples. As has been found in run length encoding investigations, a reduction of overall compressed image data size can be achieved by entropy encoding the distances between the sample points. Also, as might be expected, shorter segments are more probable in images with a fair amount of detail and hence can be encoded with very few bits.

2.4 Nonuniform sampling

A more general approach is to choose sample points on a two dimensional basis instead of on a scan line basis. Since in the general case this is an extremely computationally expensive process, work in this area has focused on image segmentation. These algorithms segment the image into regions which contain near equiluminance. The boundaries of these regions are then coded and the pixels within are interpolated. Two recent papers have used this approach. One paper by [Hovig 88] segments the image into polygonal regions which are of near equiluminance. The image is stored as the outlines of these regions and the intensity within. Another paper, [Lee 89], uses a slightly different approach. Regions are created by grouping together pixels within a certain range. The boundaries of these regions then form something akin to a topographic map of the intensity surface. These contours are then coded as splines. Part of this coding process is to reduce the number of points needed to code these boundaries. The image is reconstructed using techniques similar to reconstructing a terrain surface from a topographic map.

Other work which is related is the compression and smoothing of a digitized depth map.[Schmitt and Barsky and Du 86] In this work a bust of Victor Hugo was digitized. The algorithm used a quadtree subdivision approach to successively divide up the surface. Each surface patch was approximated by a spline based reconstruction of that patch. The patch was further subdivided if a complexity measure of detail within the surface patch was exceeded.

3. Research and Results

3.1 Proposed work

3.1.1 Extensions to existing algorithms

3.1.1.1 Goals

The goal of this work is to develop an algorithm which will decode an image quickly with very little compute power and which will generate high quality images. To achieve these desired goals, an algorithm which operated in the spatial domain seemed like the best choice. As discussed in the previous section, a search of the literature showed redundancy reduction algorithms to have the most promise. The major drawback of these algorithms is low decompressed image quality. Since it is desired that decompression be quick, the body of this work investigates ways of improving decompressed image quality without significantly increasing decompression complexity. This requires increasing the complexity of the compression half of the algorithm and therefore making the algorithm highly asymmetrical in terms of compression and decompression complexity. Two means of improving image quality were investigated: optimization of the sample point selection process and the extension of redundancy style algorithms to utilize the two dimensional correlation present in image data. Since the resultant algorithm seems to fall somewhere between redundancy reduction algorithms and nonuniform sampling algorithms, it is referred to as the nonuniform sampling and interpolation algorithm, or NSI.

The process used to investigate improved image quality consisted of experimenting with a number of different variations at each step and evaluating the resultant image quality and artifacts introduced. Three images were used for evaluation purposes. The first two are the "classic" Lena and Tiffany images from the USC data base. The original images were color and were mapped into intensity space by a weighted sum of the red. These images can be seen in Photographs [8.1-2] and [8.1-2]. (Note that all Photographs in this thesis are included in the Appendix section 1.) The third image is a section of a magazine scanned in at 150 dpi on a Hewlett-Packard ScanJet+. It was then filtered and resampled down to 75 dpi. Since the scanner does not optically filter the image before scanning, some aliasing is noticeable due to the halftoning of the image. This image can be seen in Photograph [8.1-3]. (Unless otherwise noted, all graphs of data and photographs will be of the Lena image.)

Two components need to be specified to carry on this investigation. The first is an image quality metric. Peak signal to noise ratio was chosen and is detailed in a following section. The second is a lower and an upper bound for comparative image quality. Since the assumption is that nonuniform sampling should perform better than uniform sampling, uniform sampling is

considered to be a lower bound on image quality. An analysis of uniform sampling is contained in a following section. Because of the acknowledged [Wallace and Vivian and Poulsen 88] good performance of the DCT in terms of image quality, it was considered to be a reasonable upper bound on image quality.

3.1.2 Description of the Image Quality Metric

Throughout this investigation the major goal of this work is to improve the reconstructed image quality. This implies that a means of measuring image quality is needed. Part of the problem is that "true" image quality is dependent on the human viewer and therefore complex models of the human visual system, which is not fully understood, are necessary to measure image quality. A statistical measure commonly used by researchers and which is used throughout this work is peak signal to noise ratio (PSNR). The formula is as follows: [Pratt 78]

$$PSNR = 10 \log \frac{255^2}{\frac{1}{N}\sum_y \sum_x (\hat{I}_{xy} - I_{xy})^2}$$

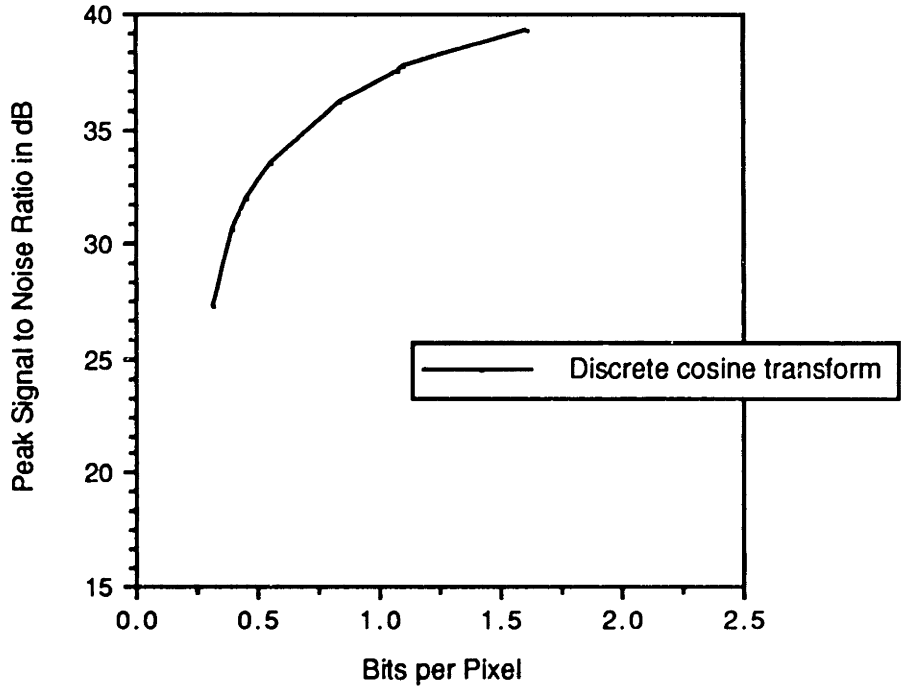
[3.1-1]

This metric is a normalized sum of squared error measure on a logarithmic scale, the peak value of the signal is assumed to be 255 for an 8 bit per pixel image, N is the total number of pixels in the image, $\hat{I}_{xy}$ is the reconstructed pixel value, and I_{xy} is the original pixel value.

PSNR can be utilized to generate rate-distortion graphs which are used to compare the performance of different compression algorithms. A rate-distortion graph is generated by calculating PSNR at a variety of compression rates and then connecting the points to form a curve on a graph. One such graph is Graph [3.1-1]. The vertical axis is PSNR, the higher the number the better the image quality. The horizontal axis is compression rate in terms of bits per pixel. Bits per pixel is calculated by taking the total number of bits needed to store the image in compressed form and dividing it by the total number of pixels in the image. When comparing algorithms, an upward shift in the rate-distortion curve implies that the algorithm's image quality has improved. The rate of droop of the curve, how fast image quality drops off with decreased bitrate, says something about the efficiency of algorithm in utilizing its bits. A sudden drop in image quality implies that algorithm has some reached some lower bound where it can no longer perform in a reasonable manner with the given amount of data. Another interesting thing to note is that rate-distortion curves may intersect. This means that at a particularly low bit rate a worse performing algorithm may gain an advantage over an algorithm which typically performs better. This advantage may not be significant, however, in that at these low bit rates image quality is below a minimally acceptable level anyway.

3.1.3 The Discrete Cosine Transform

As discussed previously, the discrete cosine transform (DCT) has been acknowledged for its high quality. For this reason, it is being used as a practical upper bound in terms of image quality. Graph [3.1-1] plots the DCT's (with an 8x8 block size) rate-distortion function. Photograph [8.1-4] shows Lena compressed to 1.0 bpp and a PSNR of 35.56 dB. (This PSNR is slightly lower than what might be expected from the rate-distortion graph because the image has been post-filtered to remove some DCT blocking artifacts.)



Graph [3.1-1]: Rate distortion curve for the discrete cosine transform.

The major disadvantage of the DCT is complexity. The DCT is symmetric in terms of compression and decompression complexity. This is because both processes require that a transformation be performed. In general $2n^2\log_2(n)$ multiplications and additions are required to transform a two dimensional string of data consisting of n x n data points.[Pratt 78] This translates into $2\log_2(n)$ multiplications and additions per pixel, where n is the square root of the total number of pixels in the image. The problem is that this makes makes the algorithm $O(n\log_2(n))$. To overcome this problem, images processed by the DCT are usually broken up into 8 x 8 blocks and then each block is transformed separately. This fixes the total number of operations per pixel at six multiplies and six adds and makes the algorithm O(n), where in this case n is the total number of

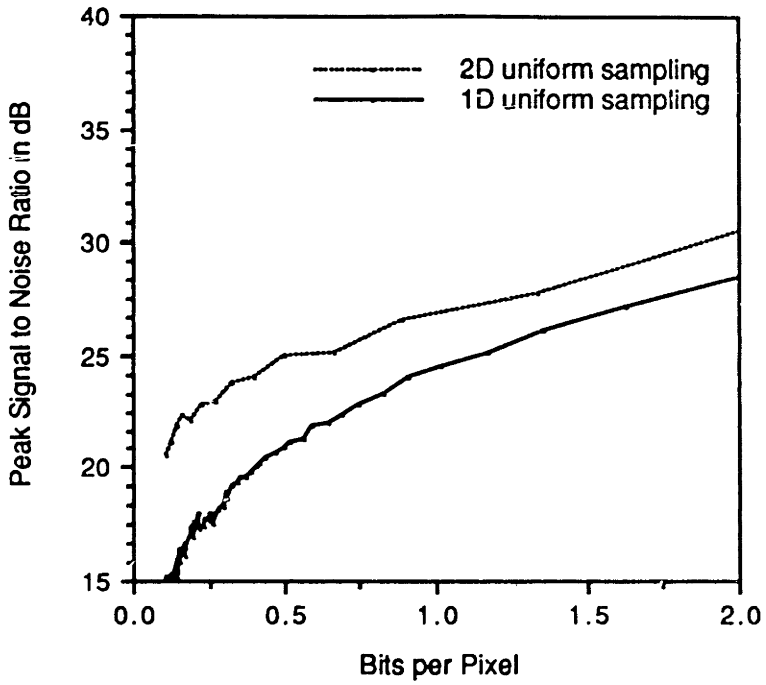
pixels in the image. The only problem with processing each block of the image separately is that the borders between these blocks become visible at very low bit rates. A variety of techniques have been introduced which filter these boundaries, but these tend to blur image details which intersect these boundaries.

Recently the DCT has received a lot of attention due to the fact that it is being accepted as an international image compression standard. Accordingly, much work is being done to decrease the complexity of computing the DCT and the inverse DCT. [Feig 90] describes an algorithm which can compute the DCT in 0.84 multiplies per pixel and 7.2 additions per pixel. However, IBM is patenting this algorithm and therefore its widespread use is questionable. A survey of techniques [Chen and Smith 77, Makhoul 80] seems to put most DCT algorithms at around 3 multiplies and 3 additions per pixel.

3.1.4 Uniform sampling

The most obvious way to reduce the size of an image is uniform sub-sampling. This process discards every nth pixel. If the image is not filtered before subsampling then aliasing will occur because the image is probably being sampled below its Nyquist frequency. Some subsampling algorithms pre-filter the image and others do not. The decision to filter is usually based on a computational complexity requirements at compression time. Filtering the image will cause textures to disappear and sharp edges will be blurred. Sampling the image without pre-filtering causes aliasing in the textured regions of the image and along sharp edges.

Because of the nature of the redundancy reduction algorithms which will be investigated, a version of subsampling which does not prefilter each line of the image will be evaluated. Graph [3.1-2] plots the rate distortion curves for one dimensional and two dimensional versions of uniform sampling. As this graph portrays, the two dimensional version of the algorithm gains about 3 dB in quality over the 1D version, by the fact that it takes advantage of the two dimensional correlation between image lines. Photograph [8.1-5] shows Lena subsampled by 8 in the horizontal direction to 1.0 bit per pixel and a PSNR of 21.10 dB, aliasing is readily apparent. Photograph [8.1-6] subsamples Lena two dimensionally, by four in the horizontal direction and by two in the vertical direction. The resultant image is 1.0 bit per pixel and has a PSNR of 24.89 dB. Aliasing is still apparent, but is reduced from Photograph [8.1-5].



Graph [3.1-2]: Rate-distortion curve for uniformly sampled Lena images.

3.2 Analysis of various error metrics

3.2.1 Introduction

A variety of methods for sample point selection are evaluated in the sections which follow and then compared. The methods fall into two categories. The first are methods which calculate some measure of the error between the original image data and the reconstructed data when choosing sample points. When a specified error threshold is exceeded, the current segment is terminated and a new one is started. One of the advantages of these methods are that they are quality controlled. Specifying a particular error threshold guarantees that no part of the image will exceed that threshold. It is therefore advantageous that the error measured is somehow related to a measure of the image quality as a whole. One of the problems with some of these methods is that as the approximating segment is extended to a new sample point, the slope of the approximating line changes, requiring that the error measure be recalculated for each point being approximated by the segment. The second category is made up of methods which try to identify some feature of the data and place sample points on or near the identified features.

Given the goals set for NSI, the main feature that a sample point selection method should exhibit is resultant high decompressed image quality. A secondary, although not completely

insignificant, aspect is the computational complexity of the method. Following, a variety of methods are evaluated based on these criteria and their individual characteristics are described.

3.2.2 Cone intersection method

This error metric was originally developed for generating approximations to two dimensional line drawings.[Sklansky and Gonzalez 80] However, this approximation technique can apply to approximating a two dimensional intensity waveform as well. This method chooses sample point locations by segmenting the plane occupied by the image intensity waveform into bounded regions, called cones. Points which fall within these cones can be approximated to the specified error. Each time the approximating segment is extended the active "cone" was readjusted.

Figure [3.2-1] shows the cones defined by P1 and P2. In this Figure, point P0 is the starting point of the data to be approximated. Point P1 is the next data point along the curve. P1 defines a cone (drawn as a pair of dotted lines in Figure [3.2-1].) which is tangent to a circle of radius equal to the error threshold, ε, centered on P2. Any line though P0 which lies within that cone will satisfy the error metric threshold. The next point on the image intensity waveform after P1, point P2, defines a different acceptable error bound cone. (Shown as the pair of solid lines in Figure [3.2-1].) The intersection of these two cones (the "active" cone) define the set of approximating line segments that are satisfy the constraints of both points. A cone intersection algorithm maintains the active cone on a point by point basis as the line segment is extended. When the intersection of the active cone and the current cone is the empty set, then a sample point must be placed. The nature of this algorithm simplifies its computation because the active cone can be computed in a scan along manner and does not need to be recomputed as the slope of the approximating line changes.

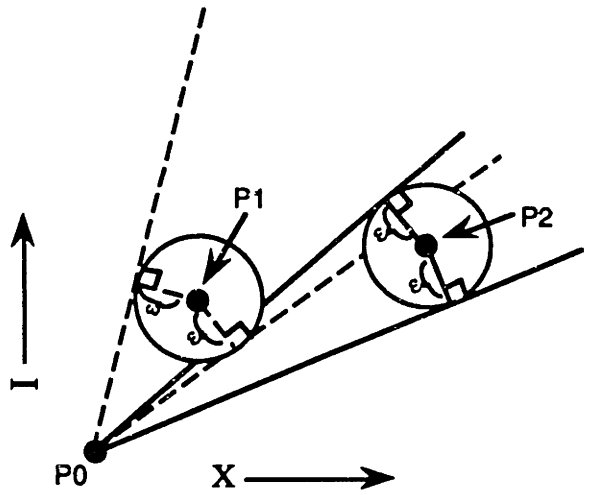
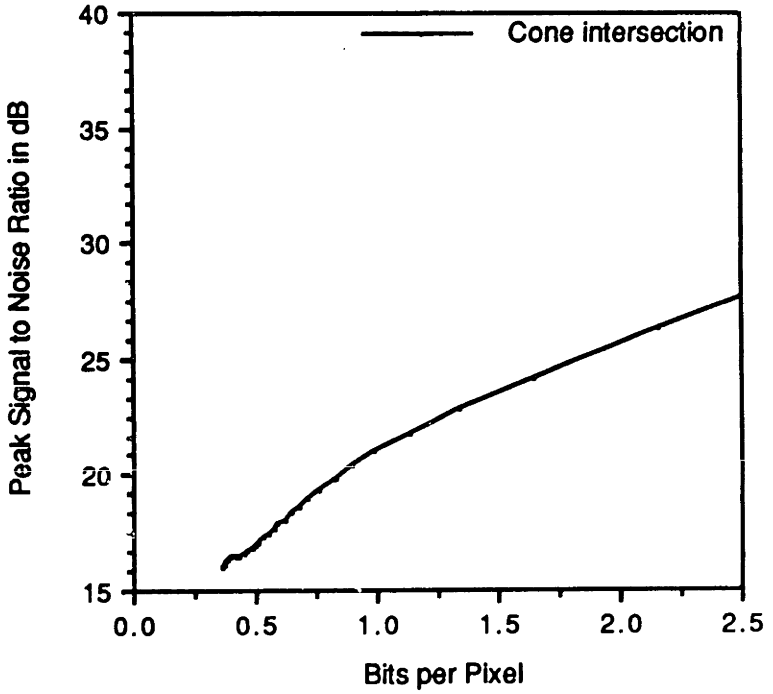


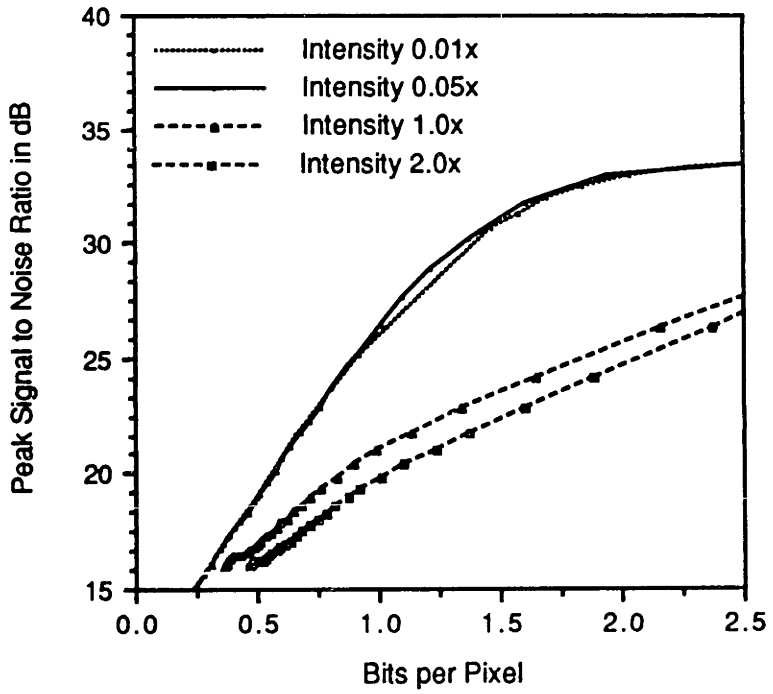
Figure [3.2-1]: P1 and P2 define two cones whose intersection bounds possible locations of future sample points.

Graph [3.2-1] plots a rate-distortion curve for the Lena image compressed using this error metric.



Graph [3.2-1]: Rate-distortion curve for Lena image processed using the cone intersection error metric.

The major problem with this error metric is the distance measure which it employs. As can be seen in Figure [3.2-1], this distance is the sum of both the spatial distance between points along a line and the difference in intensity between points. It is clear that distances in the different dimensions should be scaled before being combined, but it is not clear how to scale between the two axes. Graph [3.2-2] shows the effect on image quality, of scaling the intensity axis by four different scaling constants, 0.01, 0.05, 1.0 and 2.0. This graph shows that this scaling factor significantly effects image quality. For this image 0.05 seems to be a close to optimal scaling factor. It seems reasonable that decreasing the contribution of intensity is the correct thing to do to improve image quality. This is because most approximating line segments will be short, less than eight pixels long, therefore the intensity differences would dominate the distance measure. This makes it easy to see that the the local image height effects what this scaling constant is and how it would effect image quality. For this reason it seems unlikely that an optimal scaling constant could be determined experimentally which would generalize to all images. Photograph [8.1-7] shows Lena compressed at 1.0 bits per pixel with an intensity scaling factor of 0.05x and at resultant PSNR of 26.43 dB



Graph [3.2-2]: Rate-distortion curves illustrating the effects of different intensity scaling factors.

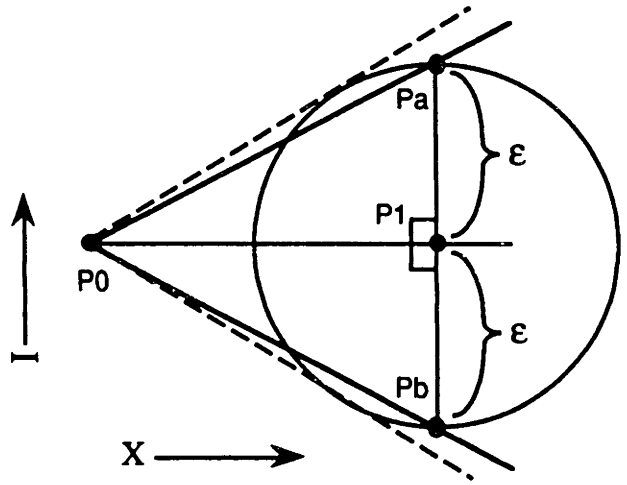


Figure [3.2-2]: Diagram of the cone intersection metric used for complexity calculation.

Calculating the complexity of this error metric is not a simple task, because the calculation of the active cone is not a simple task. The most straightforward way to calculate the active cone is to use trigonometric functions and to keep track of the current cone in terms of their angles relative to the X-axis. A simpler calculation is described in [Fowell and McNeil 89]. This computation involves a slightly different error metric which results in a tighter error bound and hence worse compression. The error calculated by this algorithm is labeled ε in Figure [3.2-2]. This error defines the solid cone in the Figure. The dotted cone is the one which would be defined by the "standard" cone intersection method. The distance measure employed in this algorithm always implies a tighter bound than the tangent method. Two computations are necessary in this algorithm for sample point selection. The first is to check to see if the new point being evaluated falls within the active cone. If it does, then a new active cone can be calculated. This check requires instantiating the point into the equations of the lines which define the edges of the cone. This requires two multiplications and four additions. Calculating the cone defined by the new point P1 requires the following steps, as detailed in the list below: Steps 1, 2, and 3 calculate the length of $\overline{P0P1}$ and require two subtractions, one addition, two multiplications and one square root. Steps 4 and 5 calculate intermediate values, this requires one division, two multiplications, and four additions. Steps 6 and 7 show how to calculate the location of Pa and Pb, but do not actually have to be performed. Steps 8 and 9 calculate the slopes of $\overline{P0Pa}$ and $\overline{P0Pb}$, this requires two additional divisions. This results in an algorithm total of: 2 subtractions, 5 additions, 4 multiplications, 3, divisions, and 1 square root per pixel.

1) $dx = (P1_y - P0_y)$

2) $dy = (P1_x - P0_x)$

3) $L = |\overline{P0P1}| = \sqrt{dx^2 + dy^2}$

4) $D_x = \frac{dx}{L} * \varepsilon$

5) $D_y = \frac{dy}{L} * \varepsilon$

6) $Pa = ((P1_x + D_y), (P1_y - Dx))$

7) $Pb = ((P1_x - D_y), (P1y + Dx))$

8) $Ma = \frac{Pa_y - P0_y}{Pa_x - P0_x} = \frac{(P1_y - D_x) - P0_y}{(P1_x + D_y) - P0_x} = \frac{dy - Dx}{dx + D_y}$

9) $Mb = \frac{Pb_y - P0_y}{Pb_x - P0_x} = \frac{(P1_y + D_x) - P0_y}{(P1_x - D_y) - P0_x} = \frac{dy + Dx}{dx - D_y}$

3.2.3 Maximum deviation

Another error measure that was examined was the absolute value of the maximum deviation between original and interpolated pixel values along an interpolating line. This measure was the one most frequently used by earlier researchers working on redundancy reduction algorithms. This measure is diagrammed in Figure [3.2-3], denoted as ε and described by Equation [3.2-1]. If this value exceeds some preset threshold then a sample is placed. The rate-distortion graph in Graph [3.2-3] shows the image quality achievable using this error measure. Photograph [8.1-8] shows Lena compressed at 1.0 bit per pixel with a PSNR of 26.31 dB.

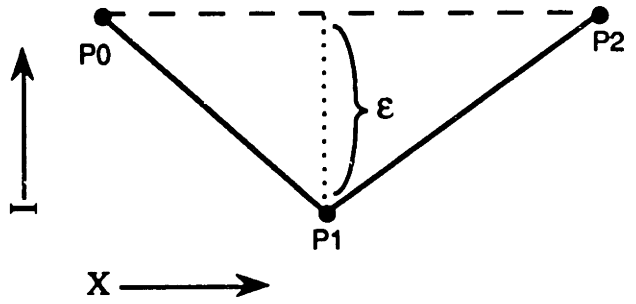
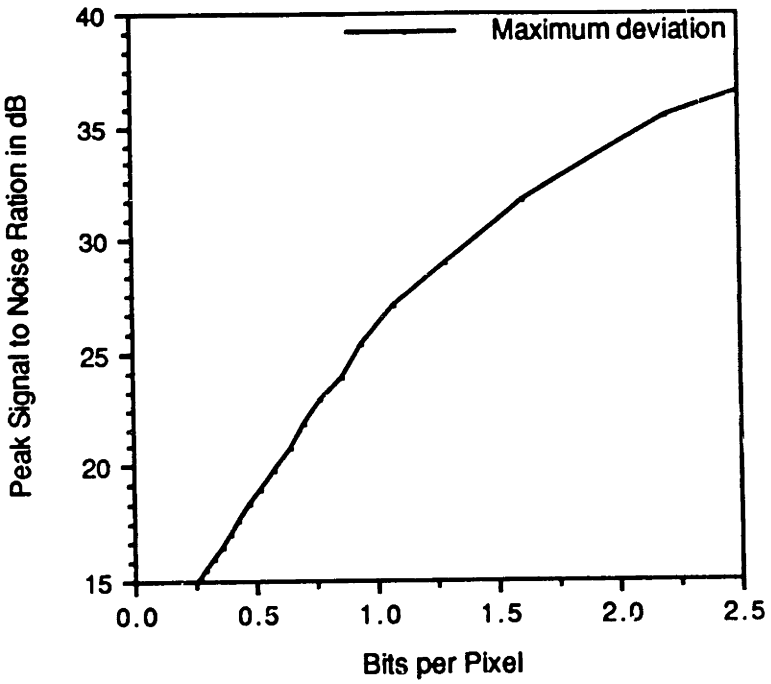


Figure [3.2-3]: The maximum deviation error measure.

$$\text{Error} = \max_{i=1}^{n} |\hat{I}_i - I_i| \qquad [3.2\text{-}1]$$



Graph [3.2-3]: Rate-distortion curve for the maximum deviation error metric.

One of the major drawbacks of this error measure is that each time the approximating line is extended and the slope of that line changes, the error measure must be recalculated for all of the points on the line. Because of this, this algorithm takes n (n + 1) / 2 additions per line, where n is the final length of the approximating line segment. This makes the algorithm $O(n^2)$ in terms of the line lengths used in the image. Therefore, as compression rate goes up and line lengths increase, compression time will rise dramatically.

3.2.4 Sum of error

Another error metric is the sum of the differences between the original data and the approximating line segment. This is diagrammed in Figure [3.2-4], where Ɛ1, Ɛ2, and Ɛ3 are the errors calculated from the individual data points. The problem with this metric is illustrated in Figure [3.2-4]. When combined, errors of opposite signs cancel one another out. The rate-distortion graph in Graph [3.2-4] shows the image quality achievable with this error metric. Photograph [8.1-9] shows Lena compressed at 1.0 bits per pixel and with a PSNR of 27.03 dB.

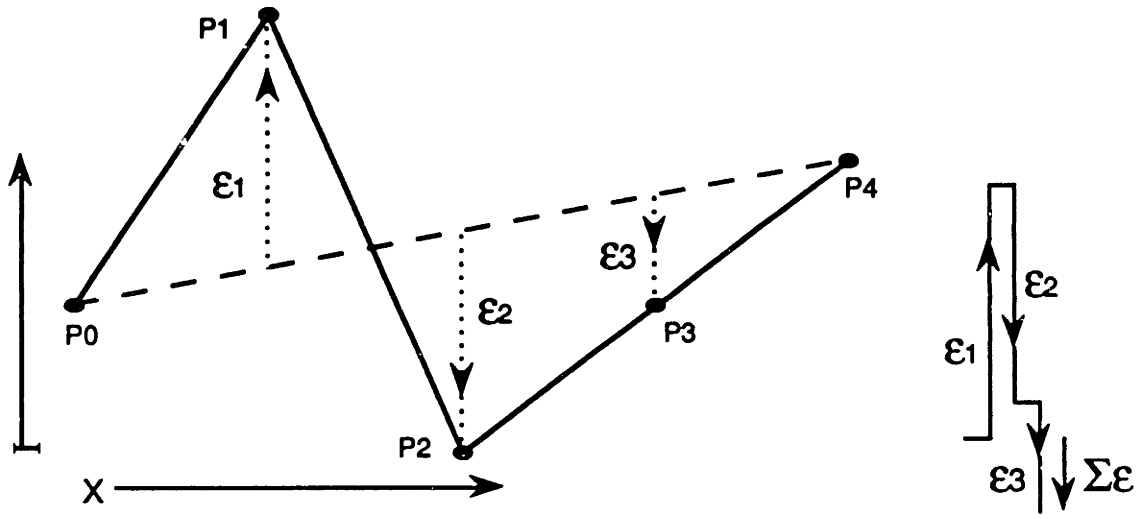
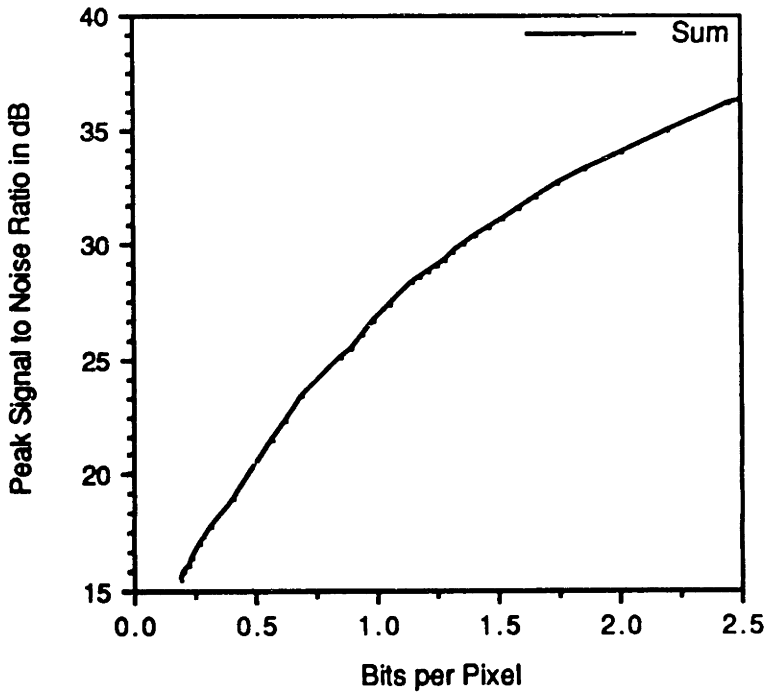


Figure [3.2-4]: The sum of errors metric.



Graph [3.2-4]: Rate-distortion curve for the sum of error metric.

This metric is desirable in that it is computationally simple. Given Equation [3.2-2], it would seem that the form of this metric, like the maximum distance metric, would require that it would have to be recalculated over all of the pixels of the approximating line each time the slope of the approximation changed. However, simple algebra shows that only a few recalculations and the maintenance of a few running sums are necessary to recompute the metric each time the slope of the approximating line changes, as is illustrated in Equations [3.2-3] and [3.2-4]. Equation [3.2-3]

replaces $\hat{I}_i$ with its linear approximation, where m is the slope of the line and b is the intercept (the first point of the line). Equation [3.2-4] expands and separates the terms of the summation into individual components. This expansion shows that the metric is O(n) to compute and consumes three additions and one multiply per pixel and the maintenance of three running sums.

$$\text{Error} = \sum_{i=1}^{n} (\hat{I}_i - I_i) \qquad [3.2\text{-}2]$$

$$\text{Error} = \sum_{i=1}^{n} (mi + b - I_i) \qquad [3.2\text{-}3]$$

$$\text{Error} = m \sum_{i=1}^{n} i + nb - \sum_{i=1}^{n} I_i \qquad [3.2\text{-}4]$$

3.2.5 Sum of squared error

This error measure is the obvious solution to some of the problems of the previous one. Here error is measured as the sum of squared (SOS) errors between the original data and the approximating line, as diagrammed in Figure [3.2-5] which shows how the squares of the errors from the previous section would add together. The calculation of this metric is detailed in Equation [3.2-5]. The image quality achieved is presented in Graph [3.2-5]. Photograph [8.1-10] shows Lena compressed at 1.0 bits per pixel at 27.58 dB. It is interesting to compare this with Photograph [8.1-9] and notice that the sharp edges in the image (at the crest of the hat and at the shoulder area) are better represented. One possible reason for this is the large penalty that an edge shift incurs because the magnitude of the error is squared.

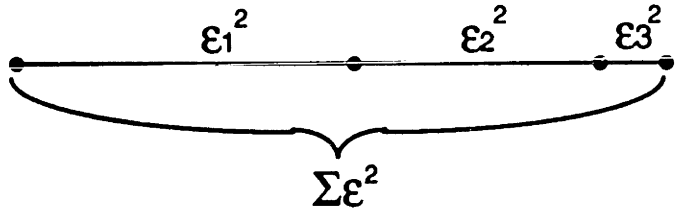
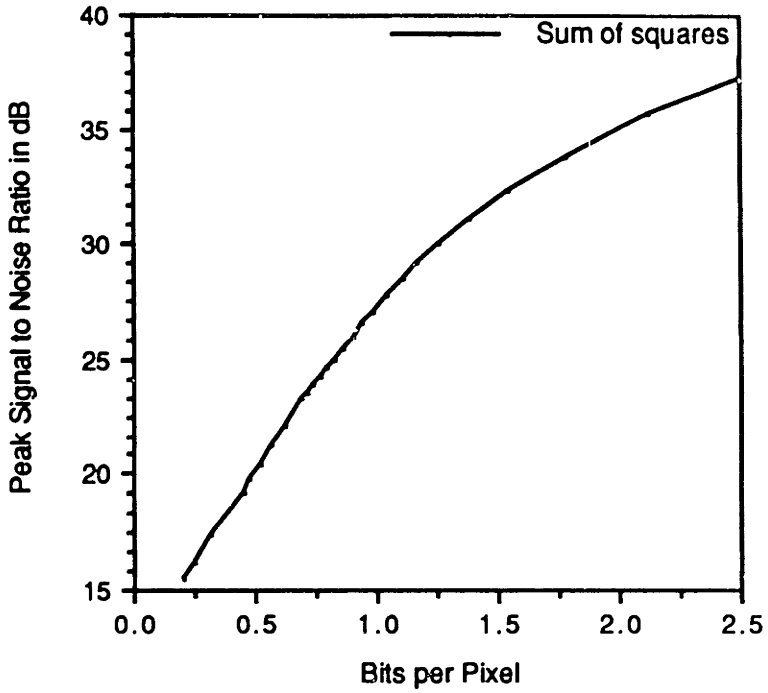


Figure [3.2-5]: The sum of squared error metric.

$$\text{Error} = \sum_{i=1}^{n} (\hat{I}_i - I_i)^2 \qquad [3.2\text{-}5]$$



Graph [3.2-5]: Rate-distortion curve for the SOS error metric.

This metric has the advantage in that positive and negative errors do not cancel one another out. This algorithm can also be implemented such that the errors can be accumulated in a scan-along manner just as in the straight sum of error described previously. This is illustrated in Equations [3.2-6], [3.2-7] and [3.2-8]. Equation [3.2-6] replaces $\hat{I}_i$ with its linear approximation, where m is the slope of the line and b is the intercept. Equation [3.2-7] is an algebraic expansion of Equation [3.2-6]. Equation [3.2-8] groups all of the terms multiplied by the slope together. Only six running sums need to be kept and combined together each time the approximating segment is extended and its slope changes. As is shown in Equation [3.2-8], computation of the metric is O(n), where n is the total number of pixels in the image, and requires $\bar{7}$ multiplies, one shift, and 10 additions per pixel. In addition it requires one multiply and one shift per line segment in order to calculate b^2 and 2b.

$$\text{Error} = \sum_{i=1}^{n} (mi + b - I_i)^2 \qquad [3.2\text{-}6]$$

$$\text{Error} = m^2 \sum_{i=1}^{n} i^2 + 2mb \sum_{i=1}^{n} i + nb^2 - 2m \sum_{i=1}^{n} (I_i i) - 2b \sum_{i=1}^{n} I_i + \sum_{i=1}^{n} I_i^2 \qquad [3.2\text{-}7]$$

$$\text{Error} = m\left(m\sum_{i=1}^{n} i^2 + 2b\sum_{i=1}^{n} i - 2\sum_{i=1}^{n}(I_i i)\right) + nb^2 - 2b\sum_{i=1}^{n} I_i + \sum_{i=1}^{n} I_i^2$$

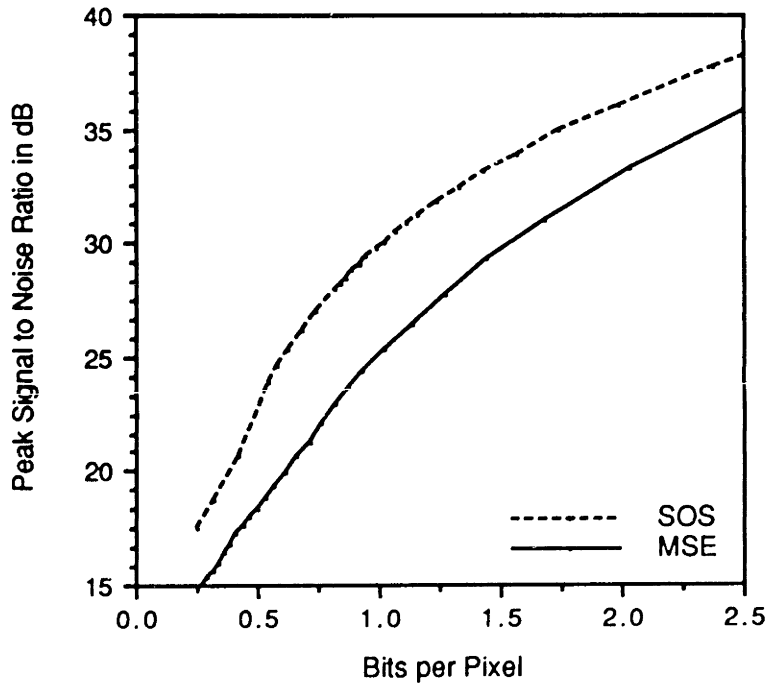
[3.2-8]

3.2.6 Mean squared error

A possible modification to the sum of squared error metric is a scaling factor which normalizes the measured error based on line length. The simplest thing to do is to divide the error by the line length. This only requires adding a single division for each point processed, so it is not objectionable in terms of computational complexity. Equation [3.2-9] illustrates the modified error metric. Graph [3.2-6] is a rate distortion graph comparing image quality of images processed by scaled (mean squared error, MSE) and non-scaled sum of squared (SOS) metrics. The quality of images compressed with a mean squared error metric are, on average, 4 dB worse than those processed by sum of squares.

$$\text{Error} = \frac{\sum_{i=1}^{n} (\hat{I}_i - I_i)^2}{n}$$

[3.2-9]



Graph [3.2-6]: Rate-distortion curves for scaled and unscaled metrics.

A problem arises with this metric because of the scan-along method employed to process the pixels. This is illustrated in Figure [3.2-6]. Since the scaling factor is always small at the beginning of a line and large at the end of a line, a similar feature which is encountered at the beginning of the line and at the end of a line will have a larger effect on sample point selection if it is encountered at the beginning of a line. The effect noticeable in an image processed in this manner is that edges immediately following large flat areas tend to be smeared out in the direction of the scan line processing.

However, not scaling the error also causes certain probiems. The first is that relatively flat areas of the image which could be completely approximated by a line segment will, over time, accumulate small bits of error which will finally become big enough to cause a sample point to be placed even though it is not necessary. Also, as a line gets longer and longer the accumulated error will get larger so that a very small deviation can cause a sample point to be placed.

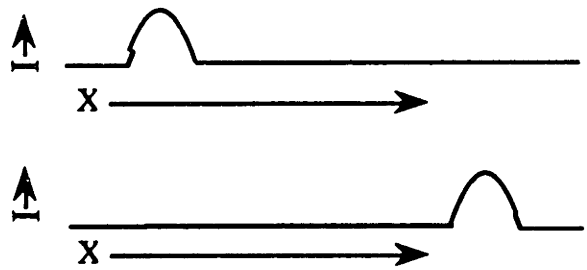


Figure [3.2-6]: Image features encountered at the beginning of a line are not treated the same as those encountered at the end.

3.2.7 Local slope

Another possible way to choose sample positions is to ignore reconstructed image error and perform a type of edge detection by examining local slope. The calculation of which is illustrated in Equation [3.2-10] involves calculating inter-pixel differences. If inter-pixel difference exceeds some threshold then a sample point is placed before and after this "edge." This is diagrammed in Figure [3.2-7]. A sample point needs to be placed before and after the edge because the slope of the intensity waveform need not be continuous. If it is discontinuous then two sample point are needed to capture the discontinuity. Graph [3.2-7] shows the image quality achievable with this sample point selection process.

$$\text{Slope} = I_i - I_{i-1} \qquad [3.2\text{-}10]$$

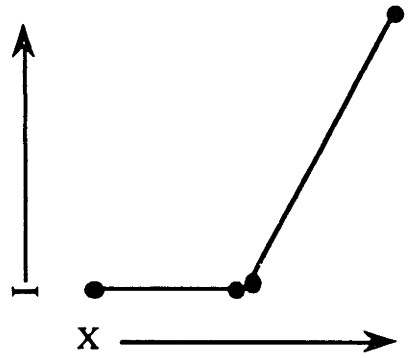
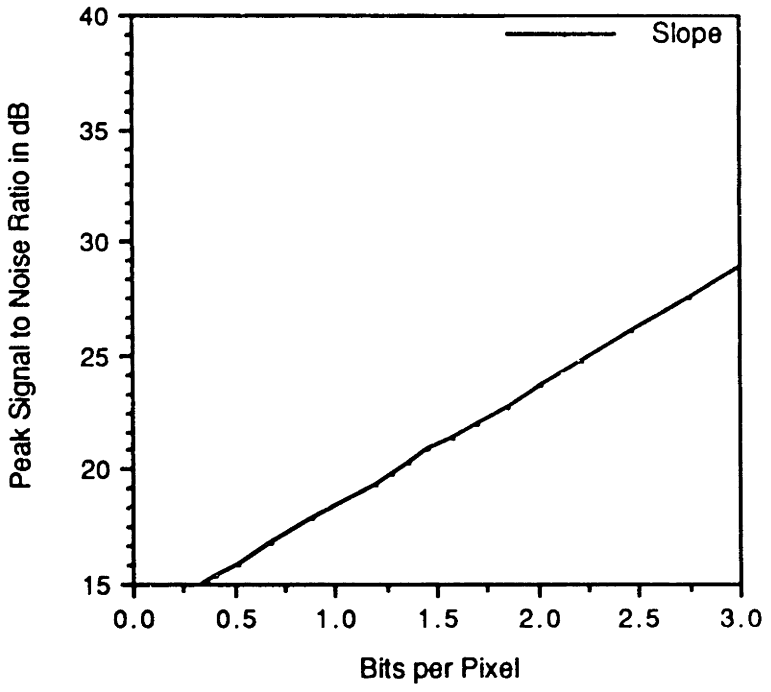


Figure [3.2-7]: Sample point selection based on slope.



Graph [3.2-7]: Rate-distortion curve for slope metric.

This process is advantageous in that computational complexity is low. The algorithm is O(n) and only requires one subtraction per pixel.

It does not perform well when faced with certain image features. One such feature is diagrammed in Figure [3.2-8]. Areas where the slope changes slowly are not assigned sample points and are therefore flattened out undesirably.

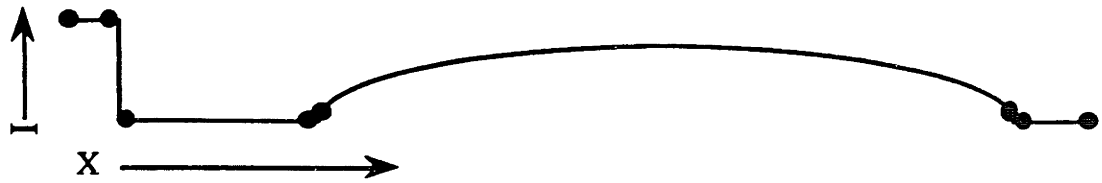


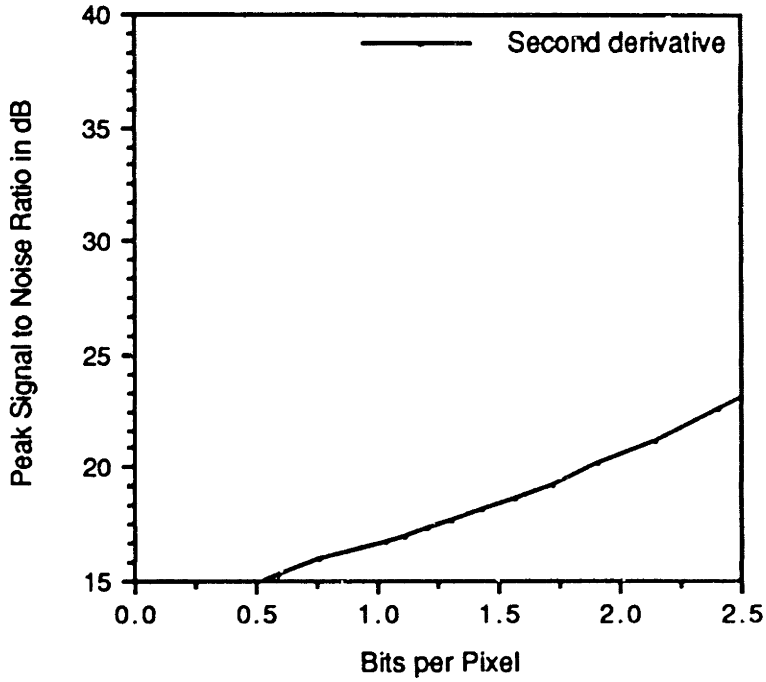
Figure [3.2-8]: The slope metric does not sample the curve where the slope changes slowly.

3.2.8 Second derivative method

An improvement to the previous sample point selection scheme is to place samples based on the second derivative of the image intensity waveform, as calculated in Equation [3.2-11]. Since a piecewise linear approximation to the image intensity waveform is being made, the assumption is that along each linear section the second derivative of the original data is zero. Therefore, places where the second derivative of the original data deviates from zero by some threshold should be good places to put sample points. Second derivative sample point selection for the curve in the previous section is diagrammed in Figure [3.2-9]. Graph [3.2-8] shows the image quality achieved with this method.

$$\text{Second derivative} = (I_i - I_{i-1}) - (I_{i-1} - I_{i-2}) \qquad [3.2\text{-}11]$$

Figure [3.2-9]: Second derivative based sample point selection.



Graph [3.2-8]: Rate-distortion curve for the second derivative metric.

Since this method is local, its computational complexity is low, the metric is O(n) and requires only two subtractions per pixel if the previous pixel difference is buffered. The problem with this method is that it is very sensitive to noise and often places sample points in unnecessary places. Some low pass filtering of the image before sample point selection might fix this problem, but this would increase the computational complexity and would tend to smooth out the sharp edges in the image that are desirable to maintain.

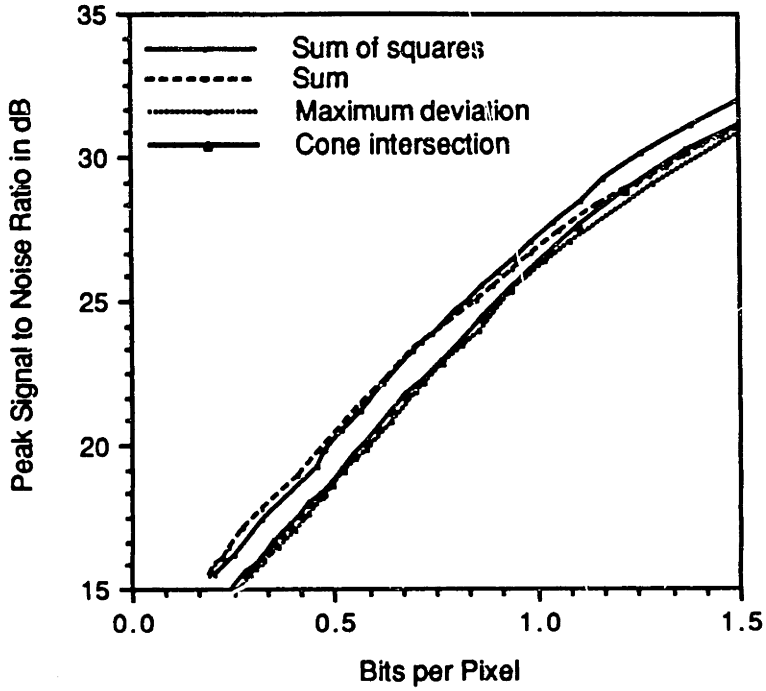
3.2.9 Comparison of Sample Selection Methods

Each of the previously described error metrics has its advantages and disadvantages in terms of computationally complexity and resultant image quality. Table [3.2-1] compares the computational complexities of the various metrics. Also included, for comparison, is the complexity of the DCT and uniform sampling. In this table **n** is the number of points which make up a line, not including the initial starting point. The registers column refers to how many values need to maintained while processing one segment. All of the metrics are O(n) in terms of line length, and hence the number of pixels in the image, except the maximum distance metric. In terms of computational complexity, the derivative methods are least expensive.

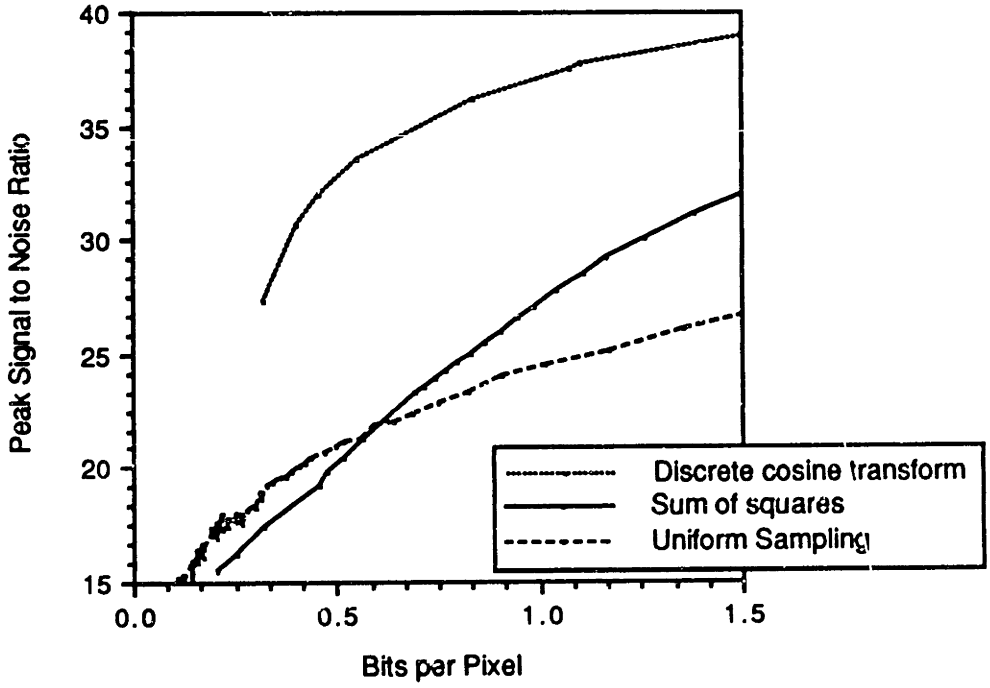
Error Metric / Algorithm	Multiplies & Divisions	Adds & Subtracts	Square Roots	Shifts	Registers
DCT	3n	3n			
Uniform Samp		< n			
cone intersection	9n	9n	n		
max. distance		$(n^2+n) / 2$			
sum of error	n	3n			3
sum of sq. error	7n + 1	10n		n + 1	6
mean sq. error	8n + 1	10n		n + 1	6
slope		n			
second deriv.		2n			1

Table [3.2-1]: The computational complexities of various error metrics.

Since one of the stated goals of this work was the best decoded image quality with a lesser concern towards encoding complexity, it is also necessary to take decoded image quality into consideration when evaluating these metrics. Graph [3.2-9] is a rate-distortion curve that compares four error metrics: sum of squares, straight sum, maximum deviation, and the cone intersection algorithm. (The slope method and the second derivative perform so poorly that they are not included in this graph.) Maximum deviation and cone intersection perform to almost the same quality. Sum of squares metric clearly performs better, but only by a small amount, 1.5 dB. In this case the subjective quality increase gained by moving to the sum of squared error metric is not completely reflected in PSNR. An interesting point is that the SOS metric performs better than the maximum deviation measure used by early researchers in this field, yet the SOS metric is O(n) where the maximum deviation measure is $O(n^2)$. Graph [3.2-10] shows how the SOS error metric fairs when compared to the DCT and one dimensional uniform sampling. The SOS obviously performs much better than straight uniform sampling. It still has a long way to go to reach the quality of the DCT. Part of the reason for this is that the DCT is taking advantage of two dimensional correlation whereas NSI is not. Extending NSI to two dimensions is discussed in a following section.



Graph [3.2-9]: Rate-distortion curves to compare the performance of various error metrics.



Graph [3.2-10]: Rate-distortion curves for the DCT, one dimensional uniform sampling, and NSI using the SOS error metric.

3.3 Heuristics to improve sample point placement

3.3.1 Sample point jittering

The algorithm, as described so far, has one particular flaw. Because a sample is not actually placed until after the error metric threshold has been exceeded, a sample is often placed after an edge as illustrated in Figure [3.3-1], where the solid line is the original intensity waveform, the dotted line is the approximation and the arrow points to sample point P1 placed after the edge.

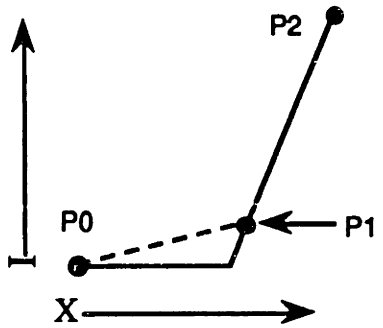


Figure [3.3-1]: A sample being placed after an edge, when the error threshold is exceeded.

This is not the optimal position for the sample point. If the sample point was placed closer to the edge, the position of the edge would be better maintained and the SOS error over the line segment would be reduced. This is illustrated in the Figure [3.3-2].

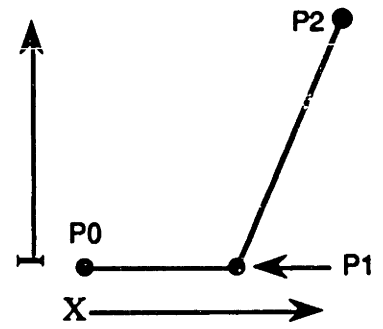


Figure [3.3-2]: A sample being placed, correctly, on an edge.

Obviously it is desirable to design an automatic procedure which can readjust the original sample point location so that it is located on the edge. Simply backing up the sample point a fixed amount after it is chosen so that the SOS error is reduced will not work because the amount that it needs to be backed up is dependent on the local characteristics of the image intensity waveform.

An algorithm was devised which could perform this adjustment automatically. This algorithm uses the approximate location of the next sample point as the constraining factor. It

attempts to reposition P1 so as to minimize the total of the SOS error of the two approximating segments, $\overline{P0P1}$ and $\overline{P1P2}$. The four steps involved in this process are:

1) Choose a sample point based on an error metric. (P_1)
2) Approximate the location of the next sample point. (P_2)
3) Backup the position of P_1 such that the total SOS error over the line segments $\overline{P_0P_1}$ and $\overline{P_1P_2}$ is minimized.
4) Discard the approximation P_2, set P_0 to the current position of P_1 and return to step 1.

In this way, the length of $\overline{P_0P_1}$ is constrained by the fact that as $\overline{P_0P_1}$ gets shorter, the SOS error on $\overline{P_1P_2}$ gets larger. The problem then collapses to deciding how to choose the approximating point P_2. The problem is that the final location of P_2 cannot actually be determined until the final position of P_1 is determined. Ideally P_2 needs to be placed somewhere after P_1, on the next stable slope so that the slope of $\overline{P_1P_2}$ after P_1 is optimized is close to what the slope of the segment $\overline{P_1P_2}$ will be after the final position of P_2 is chosen. Hopefully, this slope will be close to that of the edge which $\overline{P_1P_2}$ is trying to approximate.

The first approximation that comes to mind is to scan ahead using the sample point selection algorithm and choose the location of P_2 to be the next point which exceeds the error metric threshold. Figure [3.3-3] represents the sample point before optimization (a) and after optimization (b).

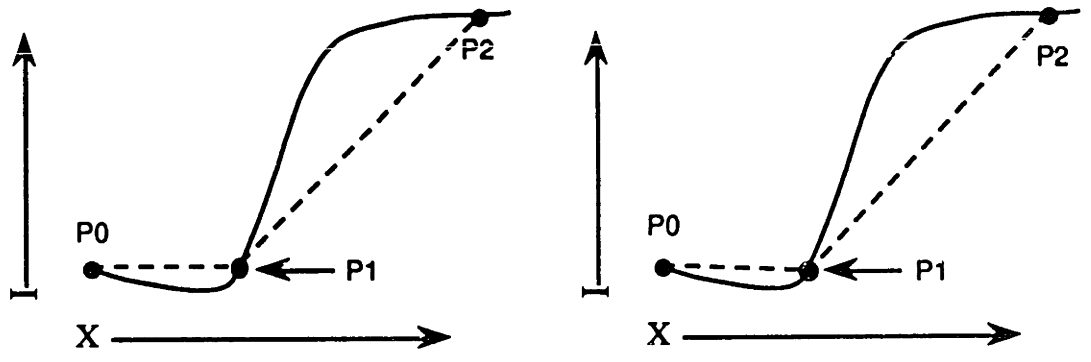


Figure [3.3-3]: (a) P0,P1,P2 before optimization. (b) P0,P1,P2 after optimization. P2 is chosen based on the full SOS error measure.

Since P_2 is also placed after the edge, the slope of segment $\overline{P_1P_2}$ is not close to that of the edge. Therefore the optimization will not be correctly calculated, as can be seen in Figure [3.3-3]. The sample point P_1 is not backed up enough. On the other extreme, an alternate approximation for P_2 is the next possible sample point, as is illustrated in Figure [3.3-4].

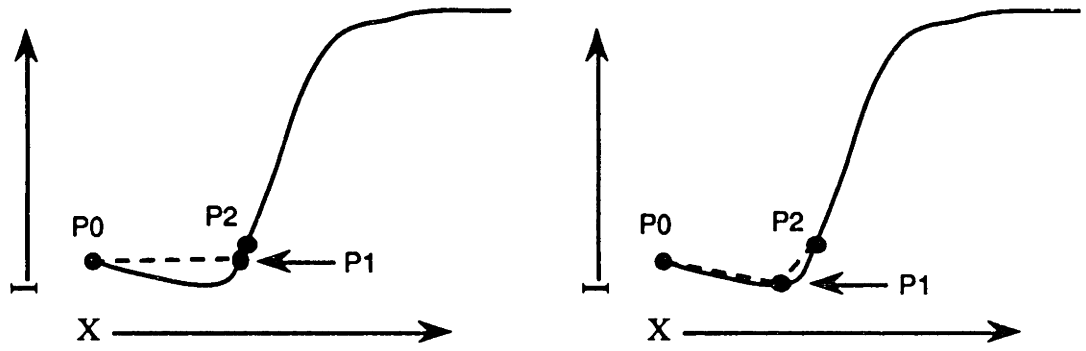


Figure [3.3-4]: (a) P0,P1,P2 before optimization. (b) P0,P1,P2 after optimization. P2 is the next point after P1.

The problem with this approximation is that since P_2 is so close to P_1, the point will be backed up too much. The final approximation used was a comprise between the two just mentioned, P_2 was chosen by scanning forward after P_1 until error metric exceeded one quarter of the error threshold. As illustrated in Figure [3.3-5]. This Figure illustrates that P_2 is placed in such a position so that the slope of $\overline{P_1P_2}$ is close to that of the edge itself.

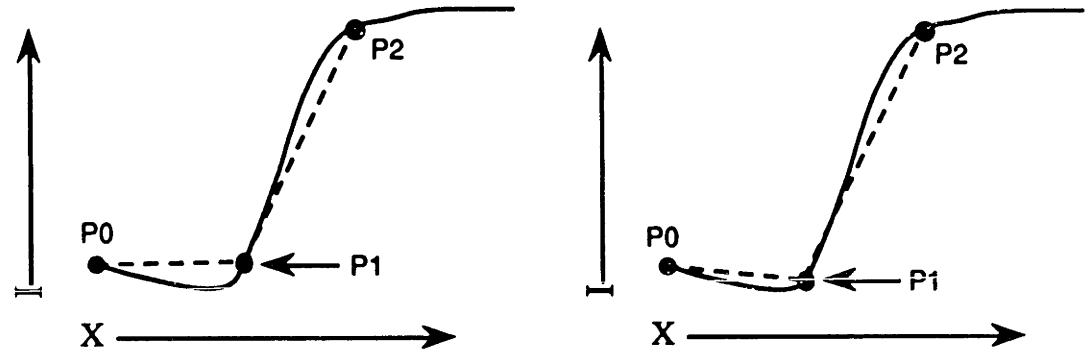
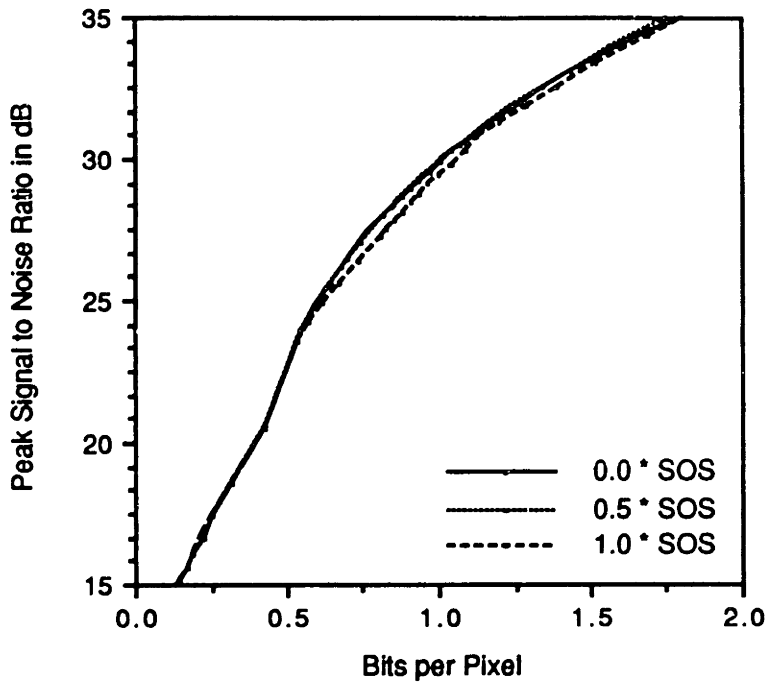


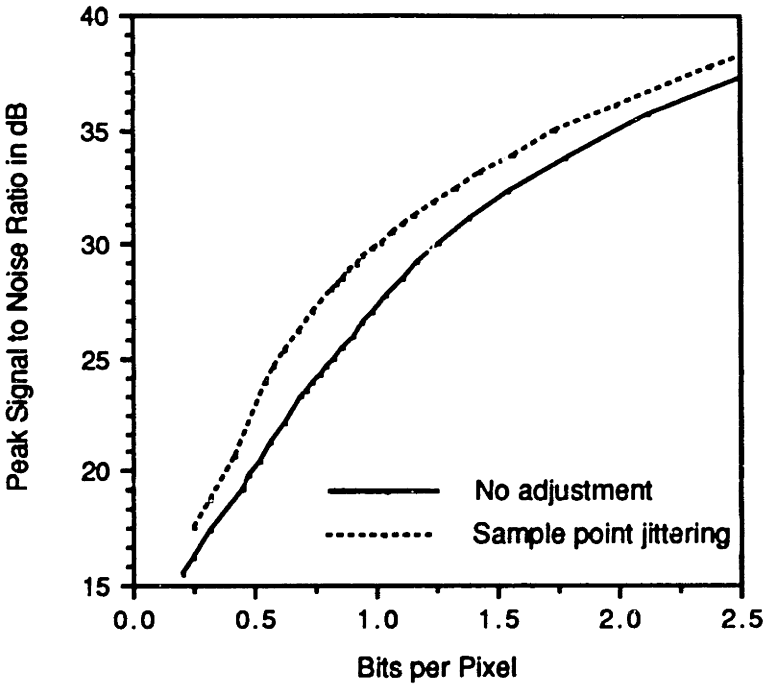
Figure [3.3-5]: (a) P0,P1,P2 before optimization. (b) P0,P1,P2 after optimization. P2 is chosen based on 1/2 threshold.

Graph [3.3-1] is a graph which shows the effect of these different ways of choosing P2 on image quality. As can be seen, the difference between these methods is minimal. This is a case where this particular parameter adjustment is significant to subjective image quality, but does affect objective quality greatly.



Graph [3.3-1]: Rate-distortion curves for different second point backup methods.

The effect of sample point jittering on image quality (on average a 3 dB improvement in PSNR) is illustrated in Graph [3.3-2]. Photograph [8.1-11] shows Lena compressed at 1.0 bits per pixel with a PSNR of 30.11 dB. It is interesting to compare this Photograph with Photograph [8.1-10]. This comparison shows how jittering greatly reduces "edge bleed". This is especially noticeable on the strand of hair which makes an arc on the shoulder.

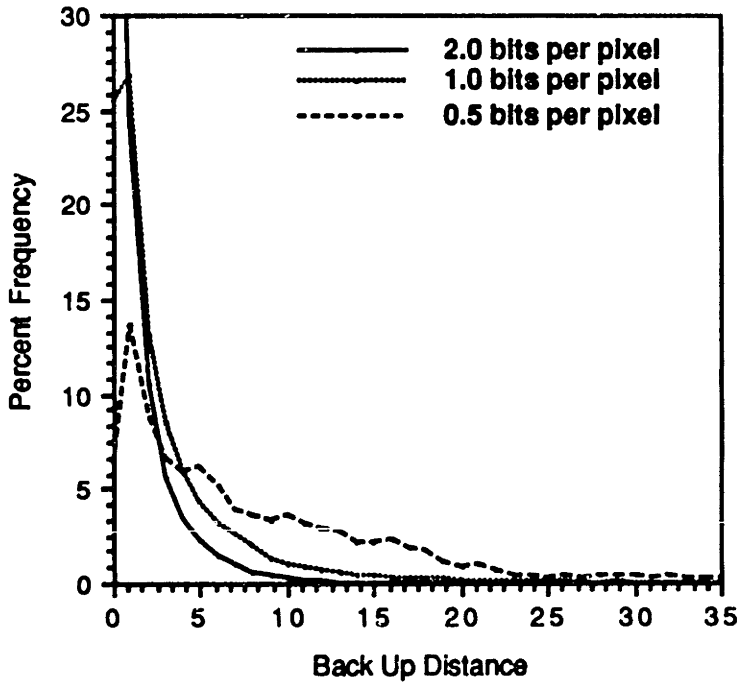


Graph [3.3-2]: Rate-distortion curves comparing NSI performance before and after sample point jittering.

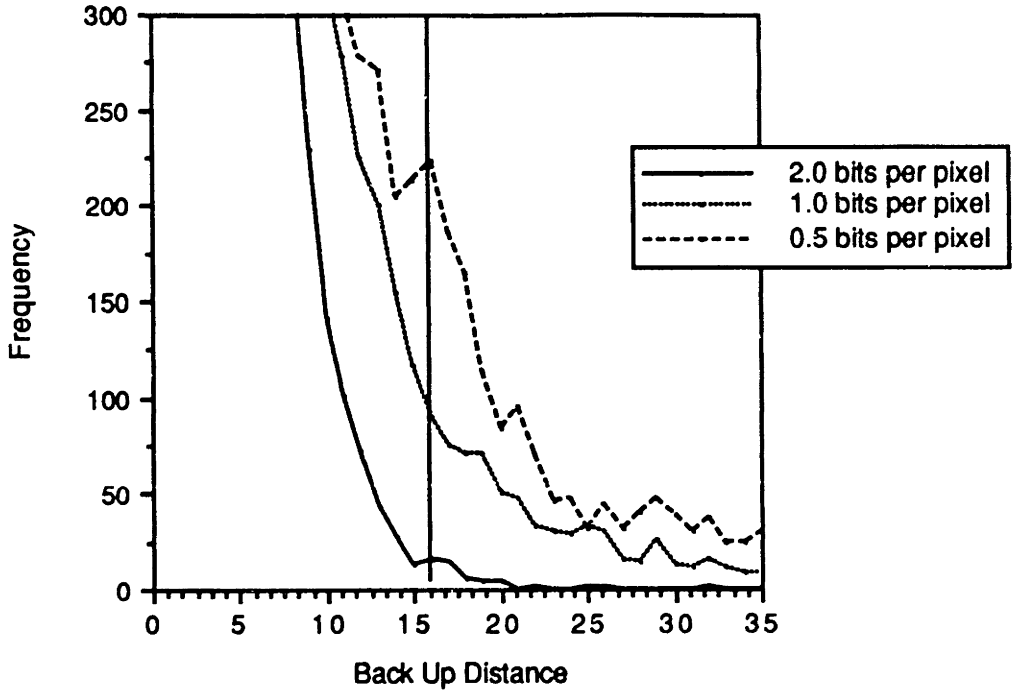
Another interesting side effect of sample point jittering is the interpretation of the error threshold. Previously, every pixel in the image was guaranteed to be approximated with an error less than or equal to the error threshold. Given sample point jittering, this is still the case. However, the pixels which are near an edge will be approximated with a higher degree of accuracy depending on how much backup actually takes place, the more backup, the greater the accuracy.

In a practical implementation it is extremely computationally expensive to have an unlimited amount of backup. Practically, it is necessary to limit the maximum allowable backup. One of the advantages of limiting the amount of backup is that some of the calculations for the error of the first segment can be cached and do not need to be recalculated. In the worse case, all points in the image would have to be processed twice, but limiting the allowable backup can reduce this. Graph [3.3-3] shows the percentage of the frequency of particular backup distances for the "Lena" image compressed at 2.0, 1.0 and 0.5 bits per pixel. As would be expected, the more compressed an image is the longer the backups will be because the error threshold is higher so that, on average, an edge tends to be overshot by a greater distance. Graph [3.3-3] shows that for backup distances more than 16 pixels, each individual backup distance makes up less than 3% of overall back up distances. Graph [3.3-4] shows the individual back up frequencies at large

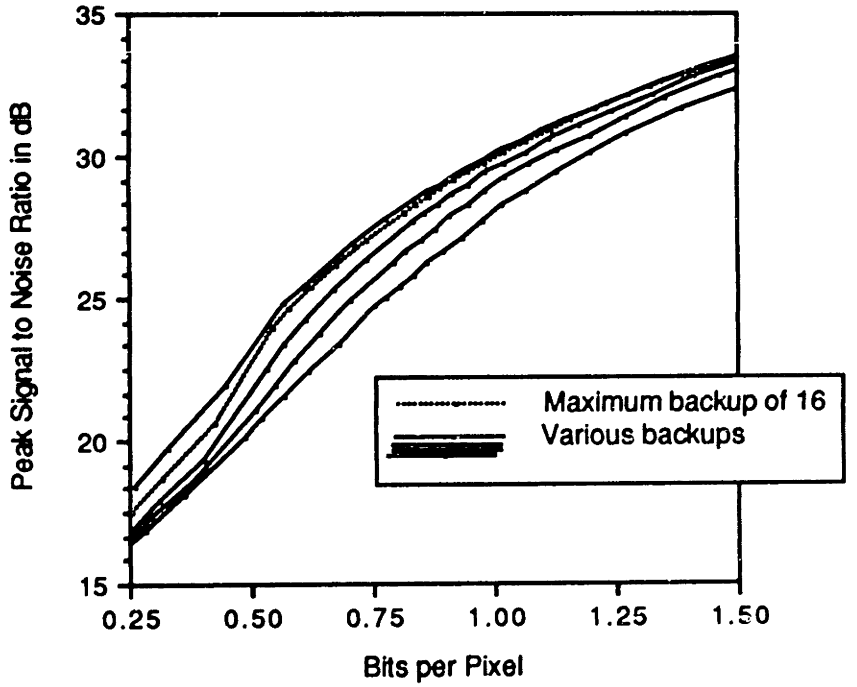
backup values. It is clear that the number of backups drops off quickly until 16, and then levels off at a very low level. As a further confirmation, a number of different backup bounds were tried, and their effects on image quality was invested. The results of this experiment are displayed in Graph [3.3-5]. This Figure contains rate-distortion curves with backup bounds of 2, 4, 8, 16, and 32 pixels, respectively, from the lowest to highest curve. The dotted curve plots the data for a maximum backup distance of 16. As the graph shows, no appreciable improvement in image quality is gained between a backup of 16 and 32 pixels, accept at very low bit rates where the image quality is so low that the relevance of this data is questionable. In light of this data, a maximum backup of 16 was chosen to be a best all around bound.



Graph [3.3-3]: Percentage distribution of backup distances Lena compressed at three bitrates.



Graph [3.3-4]: Frequency distribution of backup distances for the Lena image compressed at three bitrates. The vertical line, positioned at a backup of 16, represents what was considered an optimal backup cutoff.



Graph [3.3-5]: Rate-distortion curves for bounded backups of 2, 4, 8, 16, and 32 pixels.

3.3.2 Look ahead when threshold is exceeded

Another drawback of NSI, as currently described, occurs because of the SOS error metric and the scan-along nature of the algorithm. This situation is diagrammed in Figure [3.3-6].

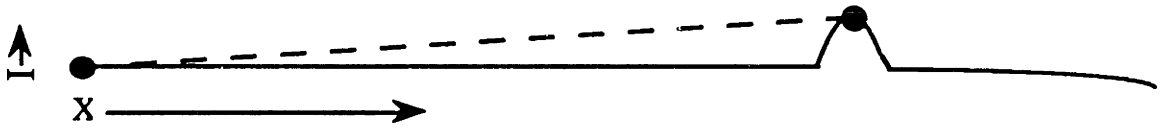
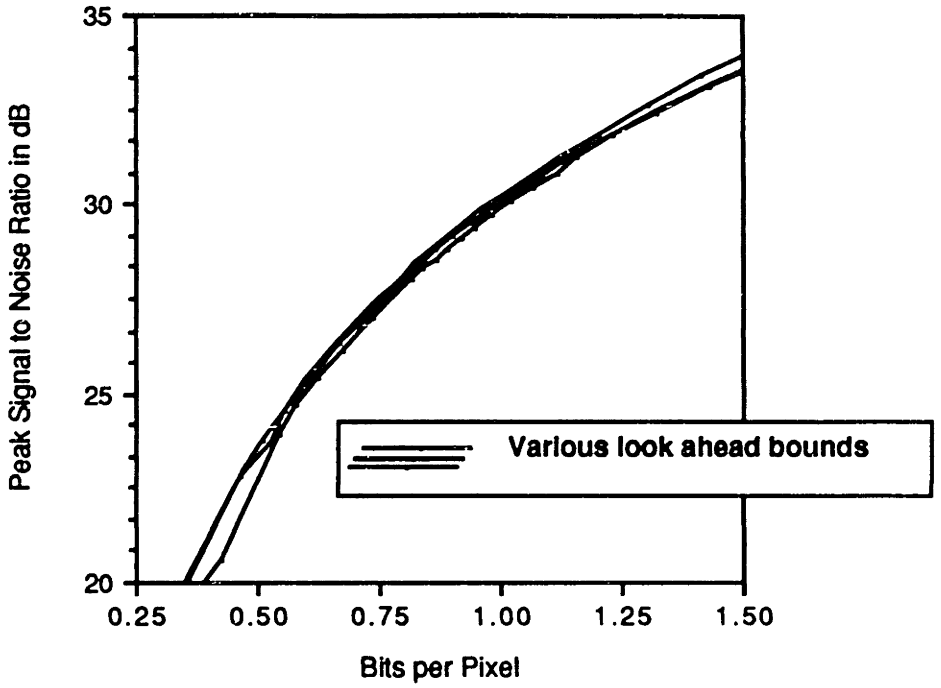


Figure [3.3-6]: A noise bump causing an unnecessary sample to be taken.

In this situation, a small bump in the intensity waveform can cause the slope of the interpolating line to be pitched up which may cause the error metric threshold to be exceeded and a sample point to be placed. It is obvious that if the approximating line is extended further, then the intensity waveform can be approximated to within the error metric threshold without the additional sample. This is illustrated in Figure [3.3-7].

Figure [3.3-7]: Extending the segment past the noise bump via look ahead.

To solve this problem, NSI was modified so that if the error metric threshold is exceeded, the line is extended further to see if the approximation can be made to move back within the error threshold. The obvious problem is that if the approximating line is extended indefinitely, each time the error metric threshold is exceeded then the algorithm will be untolerably slow. This is because the error metric may have to be calculated many times per point. Therefore, a practical implementation must limit its look ahead to a fixed number of pixels. Graph [3.3-6] shows a graph which contains rate-distortion curves for various amounts of bounded look ahead. The three data sets plotted are: 8, 16, and 32 pixels of look ahead, respectively for the bottom to top curves. The performance gained by this optimization is small and and is not greatly enhanced by extending the bound of the look beyond 16 pixels. A bound of 16 pixels was considered reasonable. Even though this does not seem like a significant optimization, it is highly dependent on the characteristics of the the image. Since many images which are processed contain noise, it was considered significant enough to include in the algorithm.



Graph [3.3-6]: Rate-distortion curves for bounded look aheads of 8, 16, and 32 pixels.

3.4 Two dimensional extensions

3.4.1 Introduction

NSI as described thus far is only one dimensional in nature. This approach has two major drawbacks. The first is that since each line of the image is processed independently, the image looses vertical coherence. This is because the slopes of the approximating waveforms of vertically adjacent rows of pixels may not be continuous vertically as was the case in the original image. This makes itself visible as stripes in the image which adversely affect image quality. This can be seen in Photograph [8.1-11]. The second disadvantage is that vertically adjacent lines are highly correlated. Ignoring this correlation degrades the algorithm's performance. This correlation can be utilized to achieve better compression ratios.

A truly two dimensional version of the algorithm would have to operate over regions of the image. One possible method is to start with minimal size regions and then grow them until an error threshold is exceeded. There are a number of problems with this method. One is that managing these regions is a difficult problem. As the regions are grown, empty spaces arise between the regions which must be kept track of and filled in later, as illustrated in Figure [3.4-1]. Another issue is that of how to choose the size of the regions. If the regions are constrained to be rectangles, then they can grow arbitrarily in the horizontal or vertical directions and can be positioned

anywhere. This is a very complex problem. Another disadvantage in this algorithm, is that, depending on how the blocks of the image are transmitted, a sequential flow of decoding may not possible. Large sections of the image may need to be buffered. In a display application where a full screen buffer is available this may not be a problem, but in a situation where the image is being decoded in a printer with a limited buffer this may be impossible. A common method used for subdividing blocks of data, which solves some of these problems, is a quadtree decomposition which is described in the following section.

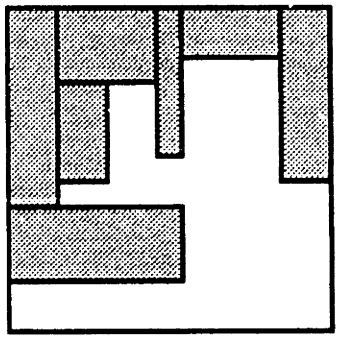


Figure [3.4-1]: Growing regions can cause empty spaces to form between blocks.

3.4.2 Quadtree based bilinear interpolation

A quadtree decomposition of an image solves certain problems introduced by a general blocking algorithm. This algorithm operates based on a test and then subdivide regime. To start with, the entire image is made the current block and is tested. This test can be any test performed on the pixels of the block. The specific test used in this experiment is described in the following paragraphs. If the block passes the given test then the algorithm accepts the block. Otherwise the block will subdivided into four equal size squares and each of those will be tested, in turn. If those blocks do not pass the test, then those blocks will be subdivided. This process continues until the all of the generated blocks of the image pass the specified test or until they reach 1 pixel on a side. A characteristic subdivision of an image by this process is illustrated in Figure [3.4-2]

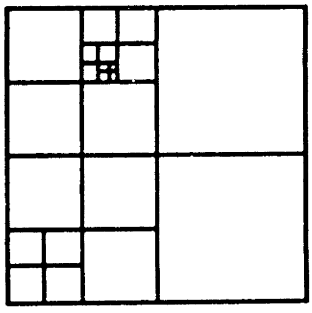
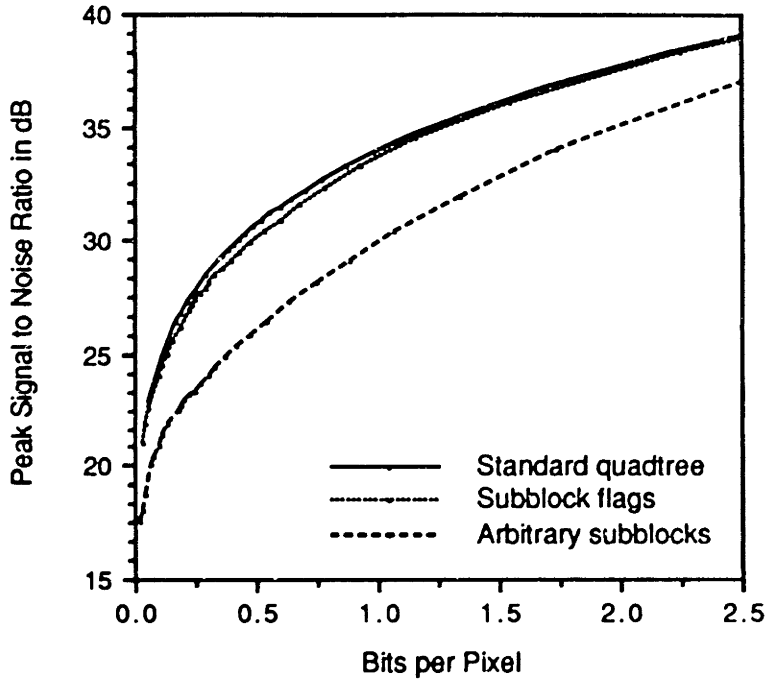


Figure [3.4-2]: A typical subblock decomposition by a quadtree algorithm.

Three variations of this algorithm were implemented and evaluated. In all of these implementations the test used was thresholded sum of squared error for the block. The error was calculated by comparing the block of original data to a version bilinearly interpolated from the four corner points. If the block could be interpolated to within the specified error then it was not subdivided. The first implementation was a relatively straight forward quadtree implementation. The major problem with a straight forward implementation is that redundant samples will be taken on block boundaries unless a map is maintained of the samples which have already been taken. This problem arises from the fact the two blocks from different subdivisions may border each other and share sample points on the boundaries. Maintaining a sample map helps eliminate duplicate samples and keep compression rate down, but increases the memory overhead associated with decompression. The rate-distortion curve for this algorithm is plotted in Graph [3.4-1]. Photograph [8.1-12] shows Lena compressed at 1.0 bits per pixel and 34.12 dB. In this Photograph, some artifacts of the block based processing of the quadtree are noticeable at edges. A problem with quadtree decompositions, in general, is that if only a small section of an image is complex and the rest is simple, then many unnecessary samples will be taken in order to get the block size small enough block to satisfy the error threshold. This is because blocks must be recursively subdivided and each time a block is subdivided, samples need to be taken at its corners. A second implementation tried to eliminate this problem by maintaining a flag for each of the four quadrants of each subdivided block. Samples were only taken for the quadrant which could not interpolated to satisfy the error metric threshold. The rate-distortion curve for this algorithm is also plotted in Graph [3.4-1], notice that no significant improvement to image quality is gained. This method reduces some of unnecessary samples, but samples still need to be taken for every level of the pyramid even if only a small detail in the block needs to be isolated. The third version solves this problem by subdividing a block down to the level of the smallest block which causes the block to approximated within the error metric threshold. In this way, samples are only taken at the bottom of the branches of the quadtree instead of each level down. This is accomplished by categorizing each block into one of three types during subdivision: (1) block passes approximation test OK to stop subdivision, (2) none of the blocks are OK - subdivide all blocks, or (3) some of the blocks are OK - four bits of flag will identify which quadrants need to be made into subblocks. The rate distortion curve for this algorithm is also in Graph [3.4-1].



Graph [3.4-1]: Rate-distortion curves three quadtree implementations.

The two latter versions of the quadtree algorithm attempt to solve the problem of arbitrary placement of sample points. The third version of this algorithm seems to solve this problem nicely, however, the added information that is carried with each block reduces the compression rate. In and of itself the bits required for each block are minimal, but they accumulate each time the algorithm moves down a level in the quadtree. Going down to a 1 x 1 block to pick up one error can be an expensive operation. This can be observed in Graph [3.4-1]. The image quality is significantly reduced with the increased overhead of flagging which samples should not to be stored. It would seem that decreasing the number of "useless" samples by increasing the algorithms' control of sample point placement would improve image quality, this turns out not to be the case. An explanation for this is that the "useless" samples are not useless, but indeed contribute to improving the image quality. Another explanation for this result is the fact that all three implementations, unlike straight forward quadtree implementations, maintain a sample map, so that an extra subdivision of a block may cost very few additional samples if any.

A separate problem with a quadtree subdivision is that because the image is processed in blocks there tend to be blocking artifacts. This occurs when the edge of a block just skims an edge in the image or if the corner of a box intersects an edge of the image. The effect of the small part of the edge may be washed out by the other pixels in the block. This can cause notches to

appear in edges in the decompressed image. Another problem which cropped up with this algorithm is that it tended to make the image look low pass filtered. Some of these effects are noticeable in Photograph [8.1-12].

The complexity and relatively low image quality of these quadtree based algorithms make them undesirable candidates for a two dimensional extension to NSI. The following section describes an alternate way of extending the method to 2D.

3.4.3 Alternate scan path techniques

A common way of extending algorithms which are inherently one dimensional to handle multi-dimensional data is to map the pixels into a lower dimensional array. One such mapping method is an "alternate scan path." These algorithms process the pixels sequentially and usually attempt to maintain the spatial locality of the higher dimensional arrangement in the lower dimensional array. In this way, if spatially local pixels are correlated in the higher dimensional representation then that correlation can be utilized by an algorithm processing the lower dimensional representation.

The three major considerations utilized to choose an alternate scanning algorithm for NSI were: (1) that edge fidelity should not be sacrificed, (2) that large smooth areas should be able to be coded efficiently, and (3) that the scan path should not generate visually noticeable artifacts in the final decompressed image.

Perhaps the simplest way to aid NSI's use of two dimensional correlation is to extend the lengths of the lines that it operates over. One way to do this is to combine the even and odd scan lines in an image into one scan line, twice as long. Figure [3.4-3] shows two ways of doing this. The first way is to scan the pixels out in a sawtooth pattern, in effect interleaving the pixels from the two lines. The other method involves scanning out the pixels in a square wave pattern.

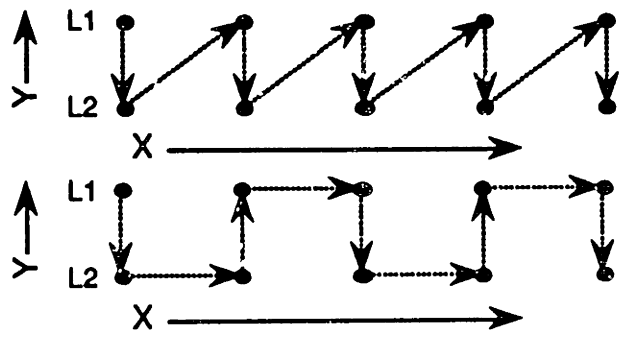
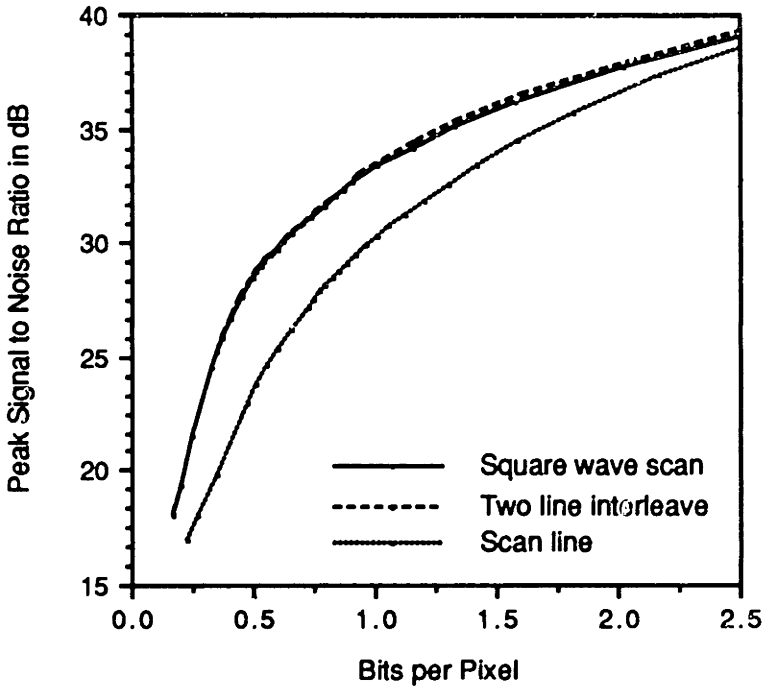


Figure [3.4-3]: Two methods for two line interleave, a sawtooth and a square wave pattern.

The reasoning here is that if a two dimensional region of the image is relatively flat, then extending the length of the line will increase the size of that region in its one dimensional

representation. Of course, an edge may cause extra samples to be taken because it may be traversed additional times by the scan, in effect, creating additional edges in the one dimensional version.



Graph [3.4-2]: Rate-distortion curves for a pair of two line interleave methods and straight scan line processing

The image quality results of two line interleave are summarized in Graph [3.4-2] and compared to a straight scan line processing of the image. Both of these algorithms improve image quality significantly, about 3 dB, over a scan line order processing. The two line interleave seems to perform slightly better than the square wave scan. Presumably this is because the interpolation algorithm benefits in reasonably smooth areas by the fact that adjacent lines are always traversed in the same direction. However, some problems still exist. One is that the image still appears to have streaks because each pair of lines is processed independently. Also, all of the two dimensional correlation in the image is not being exploited. Many smoothly shaded regions of the image span more than two lines and could be better encoded. These problems arise from the fact that only a limited area of the image is covered by the scanning pattern. If more pixels could be brought into the pattern, or possibly the entire image, then decompressed image quality could be improved.

A scan path which would appear to solve most of these problems is a space-filling curve. These curves are guaranteed to scan out all of the points in a higher dimensional space into a one

dimensional space such that the closest pixels in the higher dimensional space are maximally close to one another in the lower dimensional space. A very standard curve used for scanning a two dimensional array of pixels into a one dimension array is the two dimensional Peano scan.[Bially 69] The Peano scan can be generated by a recursive scanning scheme. The basic pattern for a two dimensional scan is illustrated in Figure [3.4-4a]. This is the path followed by an algorithm scanning a 3 x 3 pixel array. Scanning a 9 x 9 pixel array consists of dividing it into 3 x 3 subblocks and scanning each of these with the basic pattern. To make the path contiguous, the order in which the 3 x 3 blocks are processed is the same as the path for scanning a 3 x 3 array. Also, the paths in the individual 3 x 3 blocks are rotated so that the path taken through the pixels is contiguous. A 9 x 9 scan is illustrated in Figure [3.4-4b]. Larger arrays of pixels can be scanned by following the same algorithm.

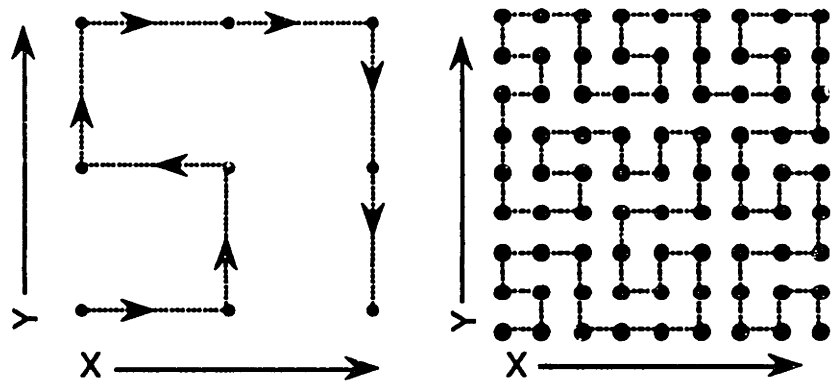


Figure [3.4-4]: (a) The basic 3 x 3 Peano scan pattern. (b) A 9x9 Peano scan pattern.

The properties of the Peano scan would seem to make it ideal for use with NSI. However this turns out not to be the case. One of the major characteristics desired of a scanning pattern for use with NSI is that a smoothly shaded patch of the image can be well coded. The problem is illustrated in Figure [3.4-5]. In this diagram a fixed slope is used to interpolate the pixels on the scan in order. As can be seen, spatially adjacent pixels have wildly varying inter-pixel intensity differences. These differences range from 0.5 to 3.5. The problem arises from an interaction between the Peano scan and the one dimensional linear interpolant used by the algorithm. This is also made apparent looking at this situation from the other direction. Figure [3.4-6] shows a bilinearly interpolable section of the image and the one dimensional Peano scan version. The one dimensional Peano scan version creates an extra edge that did not originally exist in the data. This shows that the Peano scan would not allow this portion of the image to be compressed efficiently.

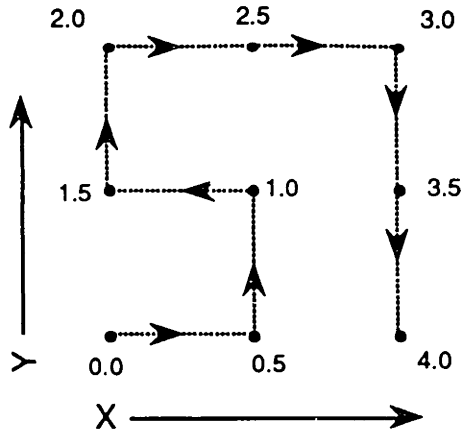


Figure [3.4-5]: A of 3 x 3 block of pixel values linearly interpolated in the Peano scan pattern with slope = 0.05.

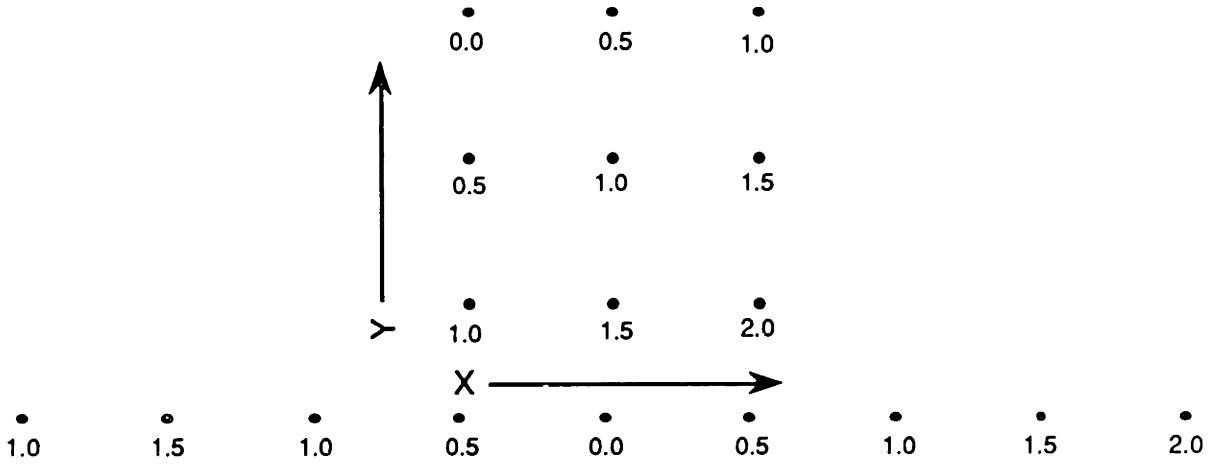


Figure [3.4-6]: A 3 x 3 bilinearly interpolably block of pixels and their sequence when scanned by the Peano scan.

This does not exclude the use of the Peano scan. It seems as though it would be possible to maintain an X and a Y slope and keep track of what direction (X, Y) the scan stepped between each pixel in the one dimensional version of the data. This possibility was not examined because the added computational overhead introduced did not seem to promise the necessary pay off in terms of image quality.

Another thought was to extend the two line interleave methods introduced before. One extension is to just increase the number of lines involved in the scan, specifically to increase the height of the scan. The problem with this straight forward approach is that as the height of the patch increases, the same problems, like banding, that occur with processing long one dimensional lines start to crop up in the vertical direction. Another method is two apply to line interleave recursively. An example of this is illustrated in Figure [3.4-7]. A 4 x 3 patch of pixels is

first scanned in two line pairs and then the one dimensional versions of these pairs are recombined into a one dimensional pattern. The resultant order of the pixel scanning reveals that the same problem will occur as with the Peano scan, the intensity of adjacent pixels in smooth patches cannot be interpolated in a reasonable fashion using a simple one dimensional linear interpolant.

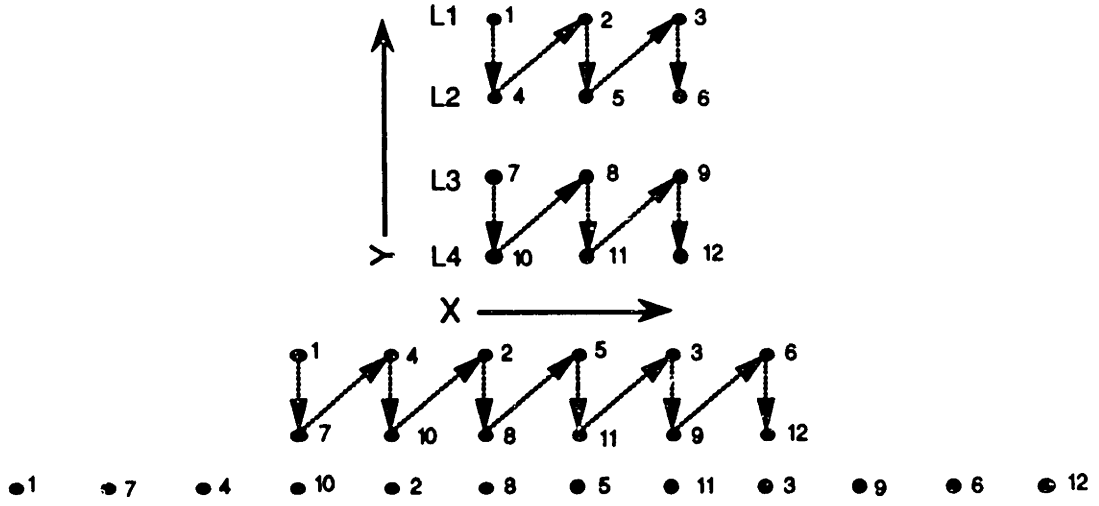


Figure [3.4-7]: A pair of lines repeatedly rescanned in a sawtooth pattern and the resultant pixel order

The scan path that was eventually settled on is the one diagrammed in Figure [3.4-8]. In this method every nth line is processed horizontally using the single line version of NSI. (Where a reasonable value of n for use with most pictures was determined experimentally.) The pixels between the lines are then processed as independent columns. The start and end point of each of these columns are the reconstructed image values from the horizontally processed lines from below and above.

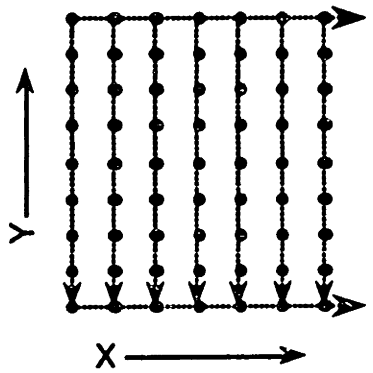
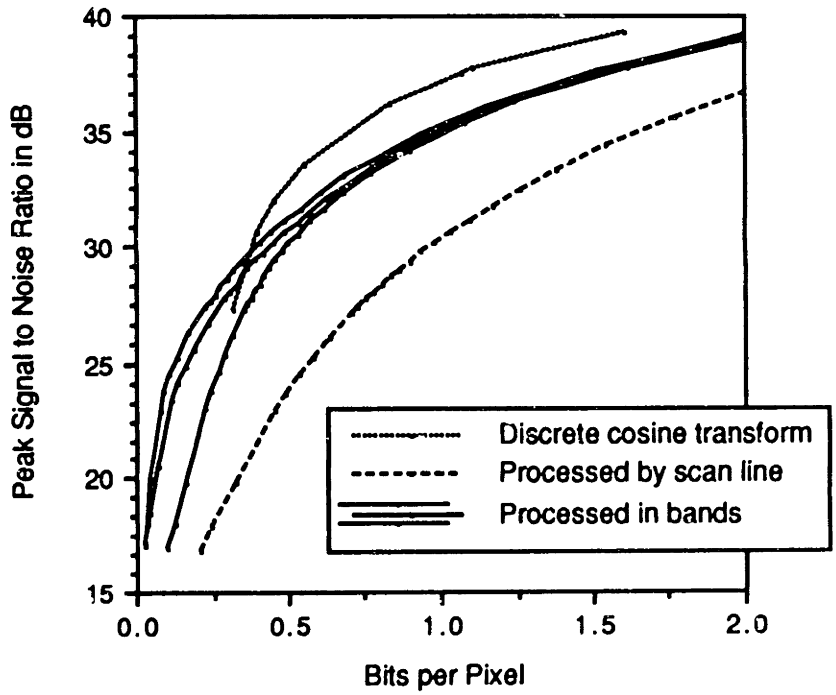


Figure [3.4-8]: Scan pattern followed by the multi-line scanning algorithm.

This scanning method statisfies the aforementioned criteria. If an image patch can be bilinearly interpolated, no samples need to be taken from the space in between the lines. Also, because each column of pixels is processed with the full algorithm, every pixel is guaranteed to be approximated to within the error metric threshold. Graph [3.4-3] shows rate-distortion curves for various band heights of 2, 8, 16, and an image processed in scan line order. Most of the improvement is gained from just increasing the number of lines involved in the processing to two lines. Not much of an improvement in quality is gained from larger bands. It is therefore not clear, what an optimal band height should be. Visually, a band height of 8 makes the boundaries between the bands least objectionable. Also band height of eight also seems justifiable in light of previous work which seems to indicate the inter-pixel correlation tends to fall off after 8 pixels, this is one reason that algorithms like the DCT divide an image up into 8 x 8 blocks. For these reasons, a band height of eight is considered optimal. Photograph [8.1-13] shows Lena compressed with a band height of eight and all previous algorithm enhancements described. This image is compressed to 1.0 bit per pixel and the PSNR is 35.27 dB.



Graph [3.4-3]: Rate distortion curves comparing the DCT, one dimensional NSI and band processing NSI algorithms.

Certain visual artifacts still occur in images processed by this scan method. One is that directional aliasing is noticeable. Since the algorithm is only looking for edges in one direction, it is

possible that a nearly horizontal or vertical which just skims a scan path will cause samples to be taken by a vertical or a horizontal pass but not both. An example of this situation is diagrammed in Figure [3.4-9]. In this Figure, the solid line represents an edge in the image, the circles represent pixels, and the boxed in pixels represent places where samples were taken. In this situation the curved edge is completely flattened by the lack of horizontal processing in the middle of the band. This can cause notches in certain parts of the image, which is illustrated in the lips in Photograph [8.1-13].

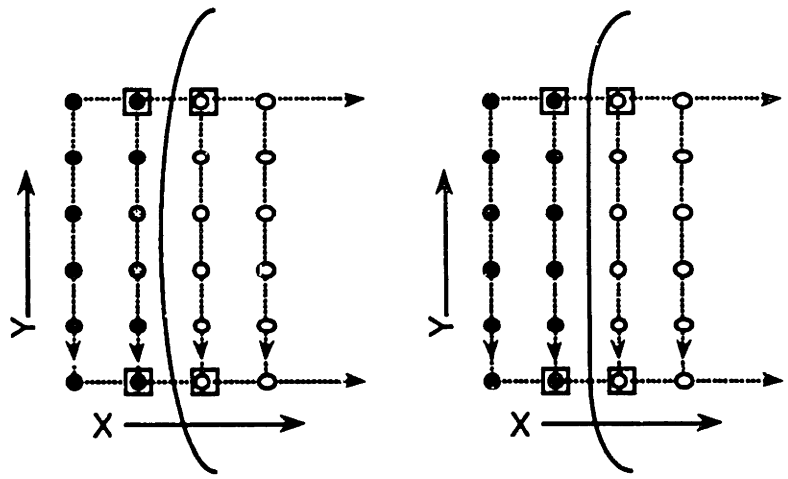
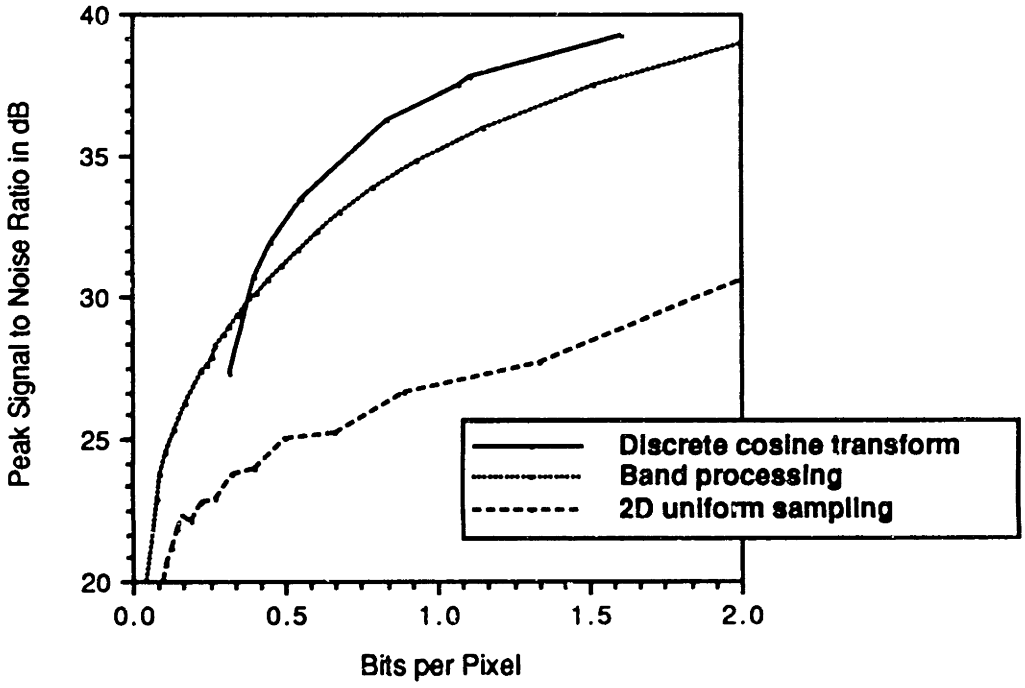


Figure [3.4-9]: The curved edge in (a) is completely eradicated by the lack of horizontal scanning in the middle of a band (b).

Another problem is that some banding is visible in the image, because every eighth line is processed independently. Another artifact is some blotchiness in the image. Both of these can be seen on the light vertical column on the left hand of Photograph [8.1-13]. This occurs because the influence of noise in the image becomes variable with this scanning technique. If a noisy pixel happens to be chosen as a sample in a column then the noise contained in a pixel will be smeared out over that column in the vertical direction. However, if a pixel which is taken as a sample has noise during the horizontal processing of pixels, its influence is different. One difference is that the noise will be smeared out in the horizontal direction. Another difference is that since the horizontal segments are not limited in length, the noise can influence a greater portion of the horizontal line. The scanning pattern itself extends its influence even more. This is because the starting and endpoint of each column are the interpolated pixels from the horizontally processed lines above and below.

3.4.4 Final evaluation

Now that the algorithm is extended to two dimensions and is in its final form, it can be reasonably compared to the discrete cosine transform and two dimensional uniform sampling. Graph [3.4-4] contains rate-distortion curves for these two algorithms and the two-dimensional version of NSI. This shows that two dimensional uniform sampling performs abysmally poorly compared to the DCT or NSI. NSI still performs worse than the DCT, but only by 2.5 dB on average. An interesting thing to note is that the rate-distortion curve for NSI falls off more slowly than that of DCT, and that the two curves actually cross at a particular bitrate. Unfortunately at the quality level where this cross over occurs, the images are not of a useful quality.



Graph [3.4-4]: Rate-distortion curves comparing 2D uniform sampling, the DCT and NSI.

Now is also the time to compare the original image in Photograph [8.1-1], the DCT compressed image at 1.0 bits per pixel in Photograph [8.1-4] and the NSI compressed image at 1.0 bits per pixel in Photograph [8.1-13]. The major things to notice are that the DCT image has an overall low pass filtered appearance, yet maintains textures in the hat. On the other hand, sharp edges in the NSI image are well maintained but, textures exhibit splotchy aliasing noise. Two other Photographs are also of interest. Photograph [8.1-14] shows how the algorithm performs at low bitrates. This image shows Lena compressed to 0.5 bits per pixel and a PSNR of 31.28 dB. Notice how textures in the image have completely disappeared and that low contrast edges are smeared. However, the sharp edges in the image do not degrade significantly.

Photograph [8.1-15] shows the image compressed at 2.0 bits per pixel with a PSNR of 39.28 dB. In this case the image is practically a lossless reconstruction of the original. The performance of the sample point selection algorithm can also be observed. Photograph [8.1-16] is a plot of samples taken by NSI to generate the image in Photograph [8.1-13]. In this image white dots represent sample locations. The first thing to notice is that samples to track edges fairly well. At the locations of discontinuities in the image intensity, one sample is taken before and one is taken after the discontinuity. Many samples are taken in the regions of the image like the feathers where there are many edges. However, in flat regions like the shoulder, relatively few samples are taken. The effect of the banded processing of the image is also noticeable. In many parts of the image, only one sample is taken every eighth line

Examining magnifications of the DCT, NSI, and original images, helps to clarify the strong and weak points of the two compression algorithms. Photograph [8.1-17] shows the left eye region of the original Lena magnified eight times. Photograph [8.1-18] shows the corresponding region in the NSI 1.0 bit per pixel image and Photograph [8.1-19] shows the corresponding region in the DCT 1.0 bit per pixel image. It is clear from these images that the DCT has a low pass effect on the edges, whereas NSI reconstructs them more faithfully. A situation where NSI does not perform as well can be seen in the next set of photographs. Photograph [8.1-20] shows the hat region of the original Lena image magnified four times. Photographs [8.1-21] and [8.1-22] show the corresponding regions in the NSI and DCT images respectively. The DCT does a reasonable job at reconstructing the texture. NSI, however, performs abysmally, in two respects. The first respect is that the texture is blotchy and aliased and smeared out. The other way in which NSI fails is that the low contrast boundary of the hat and some of the internal regions of the hat are aliased because of the eight line scanning pattern.

Further data comparing NSI and the DCT can be found in the Appendix section 3.

3.5 Algorithm Failures

NSI performs extremely well, but still has problems compressing certain image features and introduces certain idiosyncratic artifacts into the decompressed image. One quick way to examine NSI's failings is to look at a difference image. This is an image which is generated by subtracting the original image from the reconstructed image and then adding 128 so the the differences can be viewed as an image themselves. This new image is, in effect, what is missing from the reconstruction. Gray areas in the image correspond to places of little or no error, white places correspond to a positive error and black places correspond to a negative error. Photograph [8.1-23] is just such an image for Lena compressed to 1.0 bit per pixel. To make the errors more apparent, the differences where scaled by 8. (Differences which fell out of the 0-255 range were clipped.) The most noticeable artifact in the difference image is that textures show up strongly. This makes sense since that is one of the types of image features which NSI does not code well, as is described in the next paragraph.

The most obvious deficiency of this algorithm is that low contrast, high frequency image regions will not be reproduced with high quality. The reason for this can be seen in Figure [3.5-1]. A high frequency, low intensity region of the image will be tiled by line segments which will create an unevenness in intensity, but will not reproduce the signal. The only way to reproduce such a texture using linear interpolation is to place a sample on each peak and trough, which would greatly reduce the compression rate. This type of failure shows up in textured regions of images which are inherently composed of low-amplitude, high-frequency signals. This type of distortion is apparent in Photograph [8.1-21] as smearing and uneven brightness in the regions containing texture.

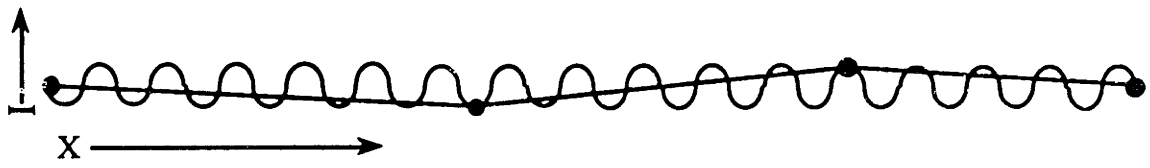


Figure [3.5-1]: A high frequency low amplitude signal is aliased.

Another distortion generated by this algorithm is that low contrast regions of the image may disappear when compressed. The reason for this can be seen in Figure [3.5-2]. A low contrast detail in the image may only have one sample taken at its maximum, which will, in effect, reduce its brightness difference from the background over a region, making it less visible. This is noticeable by comparing Photographs [8.1-24] and [8.1-25] which are magnifications of a section of the original and NSI compressed Lena images. In the NSI image it is readily apparent that the radial arms of the semicircle at the left of the image seem to fade from view.

Figure [3.5-2]: Low contrast details may fade from the reconstructed image

Another set of artifacts are caused by the scan path used to extend the algorithm to two dimensions. These are described in the section on two dimensional algorithm extensions and this section should be referenced for a full description. There are three major effects are listed there. The first is that some banding may be visible in the image, because every eighth line of the image is scanned independently of the others. The second artifact is splotchiness caused by the asymmetrical way in which noise can spread in image. The third artifact is "notching" of certain image features because the scan path only "detects" edges in the image of particular orientations, depending on how these edges intersect the scan path. All of these effect are apparent in Photograph [8.1-21] which is a magnification of the hat region of the NSI compressed Lena image.

Another drawback of NSI, as it stands, is that the contrast of the image effects how the error threshold is related to image quality. This is diagrammed in Figure [3.5-3]. A high contrast image feature will generate a larger error than a low contrast error. If contrast varies from image to image, then the contrast of the image can be calculated by computing the variance of the pixels in the image and adjusting the error threshold accordingly. The real problem comes when an image is composed of high and low contrast regions, such as text and a low contrast image. To correct this case the error threshold would have to be varied locally in the image. This could be done by computing a local contrast measure. Another possible way to achieve this is to some how factor the slope of the approximating line segment into the error threshold. In this way steeper approximating line slopes, corresponding to higher contrast region artifacts, could cause the error threshold to be increased.

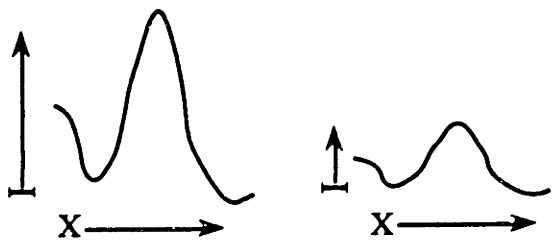


Figure [3.5-3]: The same image feature at different contrasts will be compressed differently.

Many of the drawbacks of NSI could possibly be solved by using NSI in combination with another compression algorithm so that they could compliment one another's weak points. This possibility is discussed in the "Subband Coding" and "Future Work" sections.

3.6 Frequency Domain Analysis

An informative way to analyze the characteristics of an algorithm is to look at the magnitude plot of a two dimensional fourier transform of the original image compared to the decompressed image. Photograph [8.1-26] is a fourier magnitude plot for the original Lena image and Photograph [8.1-27] is a fourier magnitude plot for the NSI compressed 1.0 bit per pixel image. It is interesting to note that most of the high frequency information still seems to be present in the compressed image. In contrast, an algorithm which tended to low pass the image would generate a fourier magnitude plot in which the center "blob" of the plot would be shrunken.

Perhaps a more informative way of using the fourier transform for analysis is taking the fourier transform of the difference image described in the previous section. Photograph [8.1-28] is a fourier transform magnitude plot of the scaled difference image in Photograph [8.1-23]. (Again, this means that the magnitude of the errors in the fourier transform plot are scaled by 8.) Two things immediately become apparent. The first thing is that the the errors from the algorithm seem to relatively evenly distributed over the frequency plane. This shows that, in general, no group of frequencies is being badly represented. The second thing is that there is a strong perfectly vertical artifact in the fourier image. This is caused by the scan path used to process the image. Because the majority of the pixels in the image are processed as small independant columns there is a discontinuity between these columns which explains the artifacts visible in the plot.

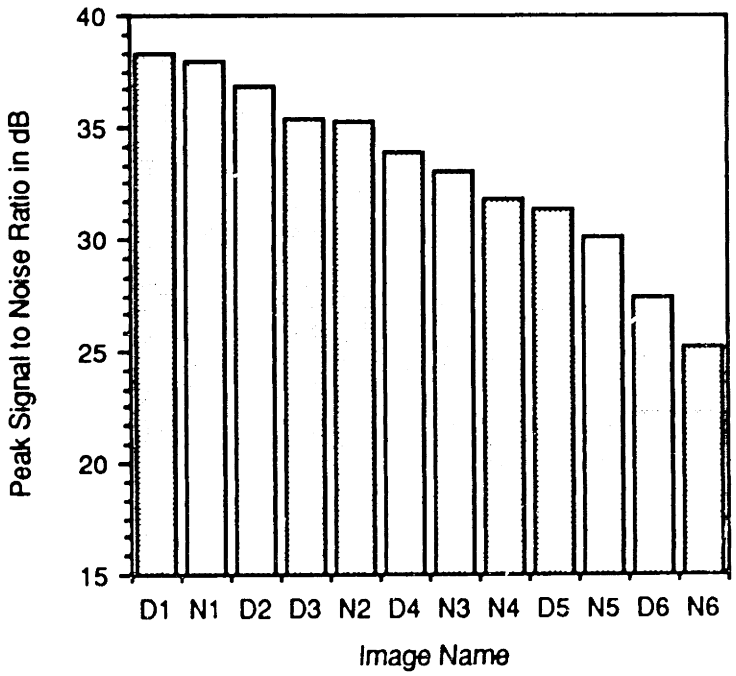
3.7 Subjective Evaluation Experiment

Throughout these experiments image quality has been compared across compression algorithms using PSNR. It is possible, however, that a particular distortion introduced by a specific compression algorithm might be exceptionally more objectionable to a human viewer than another and that PSNR would not capture this. For example, the DCT tends to create blocking artifacts and blur sharp edges whereas NSI tends to smear edges at low bit rates and generate aliasing in textured regions. To justify the use of PSNR, an experiment was performed to evaluate image quality subjectively. Twelve compressed and decompressed versions of the Lena image were used for this experiment. Six were compressed using the DCT and six using NSI. (The boundaries between the blocks in the DCT images were filtered so that the blocking artifacts of the DCT would not unduly bias the subjects.) The statistics of the images used is summarized in

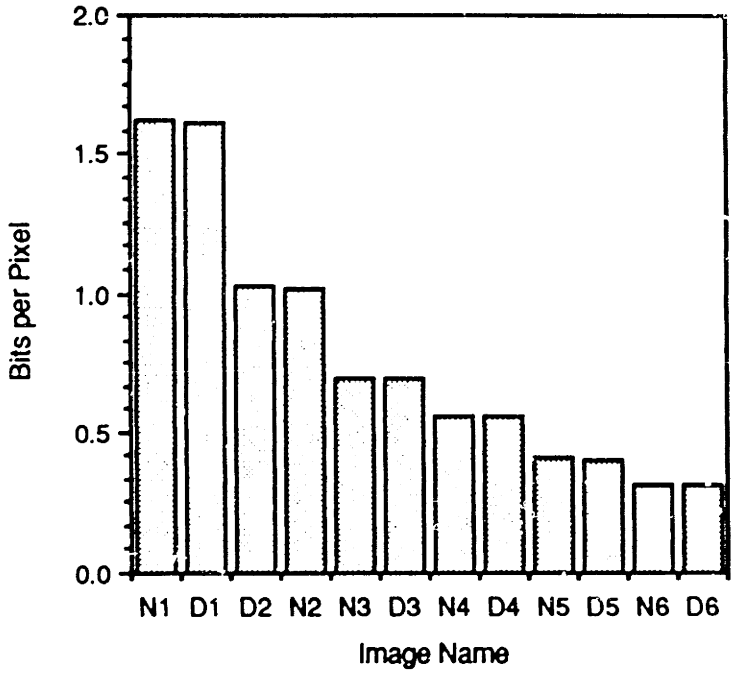
Table [3.7-1]. Graph [3.7-1] plots the experimental images' PSNR in descending order. Graph [3.7-2] plots bitrate in descending order. Images were generated such that the bitrates were equivalent across the two algorithms.

Image Name	Compression Type	Bits Per Pixel	PSNR In dB
D1	DCT	1.610	38.257
D2	DCT	1.024	36.805
D3	DCT	0.696	35.318
D4	DCT	0.553	33.874
D5	DCT	0.400	31.312
D6	DCT	0.311	27.419
N1	NSI	1.624	37.931
N2	NSI	1.013	35.269
N3	NSI	0.697	33.077
N4	NSI	0.559	31.774
N5	NSI	0.406	30.114
N6	NSI	0.317	25.260

Table [3.7-1]: A list of the DCT and NSI images used in the subjective evaluation experiment.



Graph [3.7-1]: Sorted graph of the PSNR of the images used in the experiment.



Graph [3.7-2]: Subjective bit per pixel ranking

The experiment charged subjects with the task of choosing which of two displayed images was a more faithful rendition of the original image, also displayed. Images were displayed in the arrangement shown in Figure [3.7-1]. The top image is the original image and the bottom two images are reconstructed images. Image display was performed on a Barco calibrated monitor in a dark room, to control the viewing situation. The images were displayed on a background of neutral gray (intensity value = 128) to eliminate background contrast effects. A single presentation of the images consisted of blanking the screen blanking entirely to gray, loading the three images, displaying the three images simultaneously, and then a subject pressing the number "1" or "2" on the keyboard followed by "RETURN" to specify which image was the better rendition of the original. The screen was blanked between presentations to eliminate any bias introduced by image loading order, or by having an image being replaced line by line with another image which would allow a direct comparison between images. Subjects were given as much time as they required to make their decision. Screen refresh between each presentation was only 5 seconds. A trial consisted of showing a subject all pair-wise comparisons of all twelve images, for a total of 66 triplets. (Images were not compared to themselves.) Images were presented in a random order and left-right position was randomized. Each subject performed 3 trials.

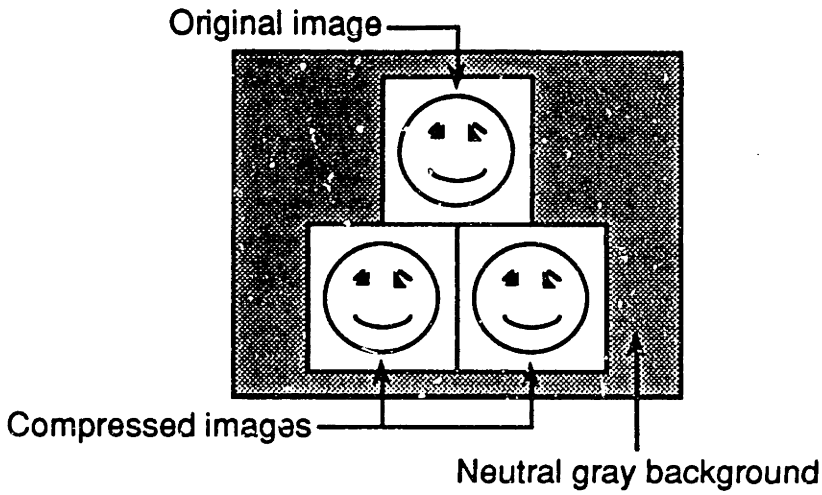


Figure [3.7-1]: The displayed arrangement of images for the experiment.

To further control the viewing parameters, subjects' head positions were fixed using a chin rest which placed them approximately 71 centimeters from the screen. The images themselves were made up of 512 x 512 pixels and when displayed were approximately 14 centimeters square. Given these dimensions, each image subtended 11 degrees of visual angle.

A total of 18 subjects were used for the experiment. Six manipulated images in their work every day, the other twelve were unskilled observers.

As the subjects progressed through the three trials their response time became quicker. It seemed as though they had developed particular criteria for judging the images. To try to capture this aspect of the experiment, subjects were asked what parts of the image and criteria they felt that they were using to judge the images. Table [3.7-2] lists what each subject stated was the image region they looked to first when evaluating images. An interesting observation that can be made from this table is that 75% of the subjects who were considered "naive" used portions of the face for discrimination, where as only 33% of the "experienced" observers used the face. The more experienced observers seemed to use the regions of the image which contained finer detail. This seems to hint at the different requirements for different users of images.

Image region evaluated	Naive	Experienced
Eyes	1	1
Mouth and chin	2	1
Shoulder smoothness	1	
Lines in hat and brim	2	3
Details in the feathers	1	1
Face in general	5	

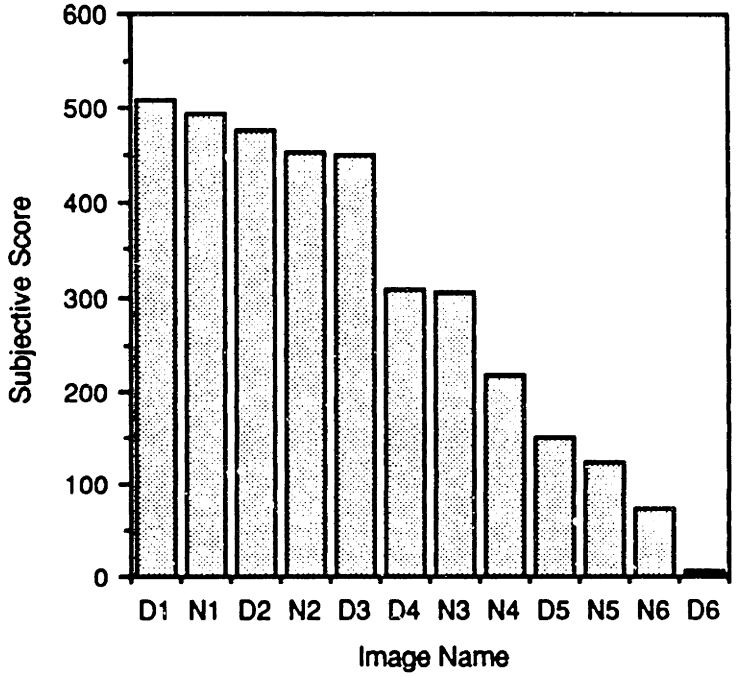
Table [3.7-2]: A tabulation of remarks from subjects describing image judging criteria.

The data from the experiment was analyzed to compare the ranking of images based on the subjects' opinions to that of PSNR. To generate this ranking, each image was assigned a

score as to how many times it was preferred over another image by a subject. Table [3.7-3] summarizes that data and Graph [3.7-3] plots it. It is obvious that the subjects' ordering of the images is close but not the same as that of PSNR, plotted in Graph [3.7-1]. Figure [3.7-2] compares that ranking, taking statistical significance into account. Images in the same column have rankings which are not statistically significantly different. Statistical significance was determined to the $p < 0.05$ level using the Wilcoxon signed ranks test.[Siegel and Castellan 88] This analysis brings PSNR into agreement with the subjective ranking. The only exception is a reversal at the two lowest bitrate images. One possible explanation for this is that the DCT's performance tends to fall off dramatically at very low bitrates because of certain fixed overhead associated with its implementation. Also, at these low bitrates the blocking artifact of the DCT becomes very apparent and may be considered objectionable.

Image Name	Subjective Score
D1	508
D2	478
D3	451
D4	306
D5	149
D6	6
N1	495
N2	453
N3	304
N4	218
N5	124
N6	72

Table [3.7-3]: Subjects' scores for experiment images.



Graph [3.7-3]: Sorted graph of subject's score data.

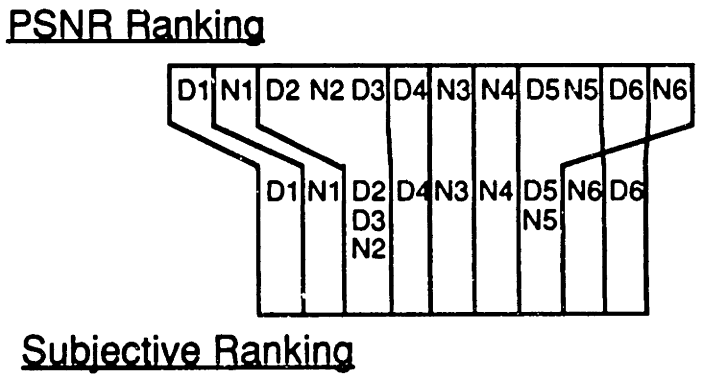


Figure [3.7-2]: A comparison of image ordering generated from PSNR and subjective data.

The result of this experiment is that PSNR seems to be a reasonable measure of image quality and agrees favorably with the subjects ranking of the images. A possible explanation for the good performance of PSNR in this situation is that neither the DCT or NSI implementations were optimized to take advantage of human visual system characteristics.

3.8 Extensions to Color

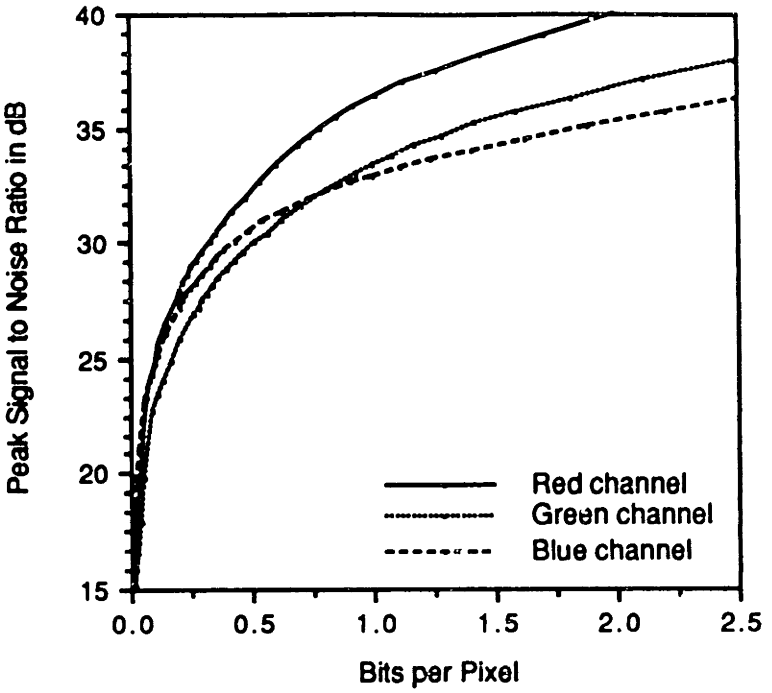
All of the work presented thus far has concerned itself with monochrome images. In most images, the majority of the information in the image is contained in the intensity channel. However

the compression of color images is often desired. There are a number of different options available for compressing an color image.

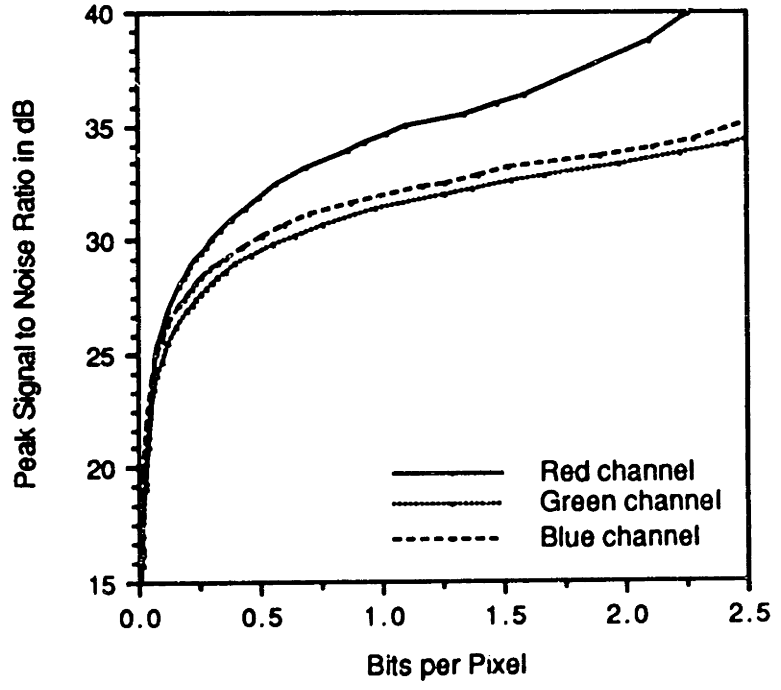
One method is to compress the red, green and blue channels separately. The problem with this method is that the red, green, and blue channels are highly correlated. This can be seen in Photographs [8.1-30], [8.1-31] and [8.1-32] which are the red, green, and blue channels, respectively, of the color Lena image in Photograph [8.1-29]. Therefore, in certain situations, like at the location of a black to white transition, three samples will be taken at the same location in all three color channels. One way to fix this is to transform the image into an alternate representation which decorrelates the different channels. One such color space is the YIQ color space. This space is a linear transformation of the RGB colorspace into a luminance channel (Y) and two color channels (IQ). Photographs [8.1-33], [8.1-34] and [8.1-35] are the Y, I and Q channels, respectively, of the color Lena Image. It is immediately noticeable that the I and Q channels contain much less detailed information compared to the Y channel. Another trick that is frequently used with YIQ data is to take advantage of the fact that the human eye's color sensors resolve things at a lower resolution than its intensity sensors. Some algorithms exploit this fact by subsampling the color channels to gain extra compression.

To evaluate the performance of NSI on color images, two 24 bit per pixel color images were used. The first is the color version of Lena shown in Photograph [8.1-29] and the Frog image shown in Photograph [8.1-36].

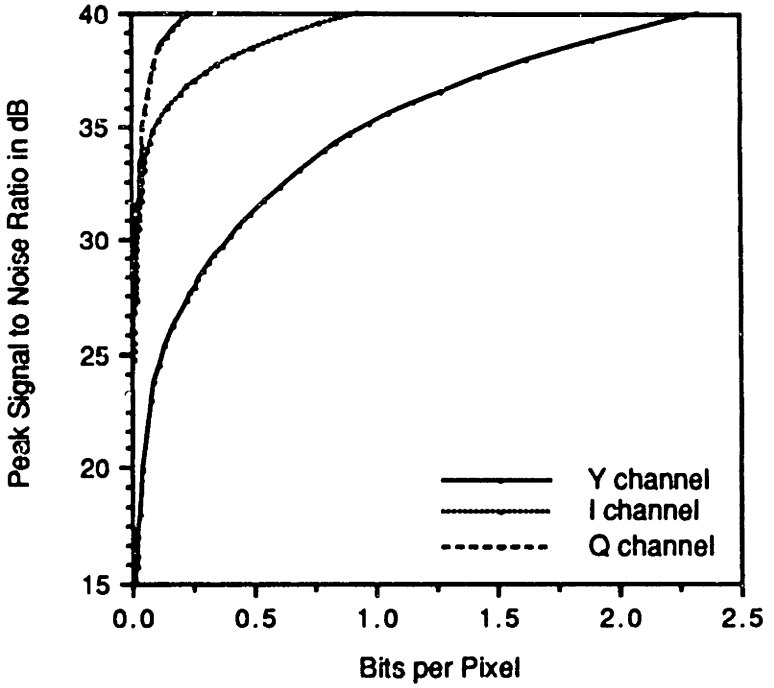
The Graphs [3.8-1], [3.8-2], [3.8-3], [3.8-4] show the image quality achieved for the two different images and different color spaces. It is obvious from Graphs [3.8-3] and [3.8-4] that the I and Q channels compress very efficiently and can boost overall image quality.



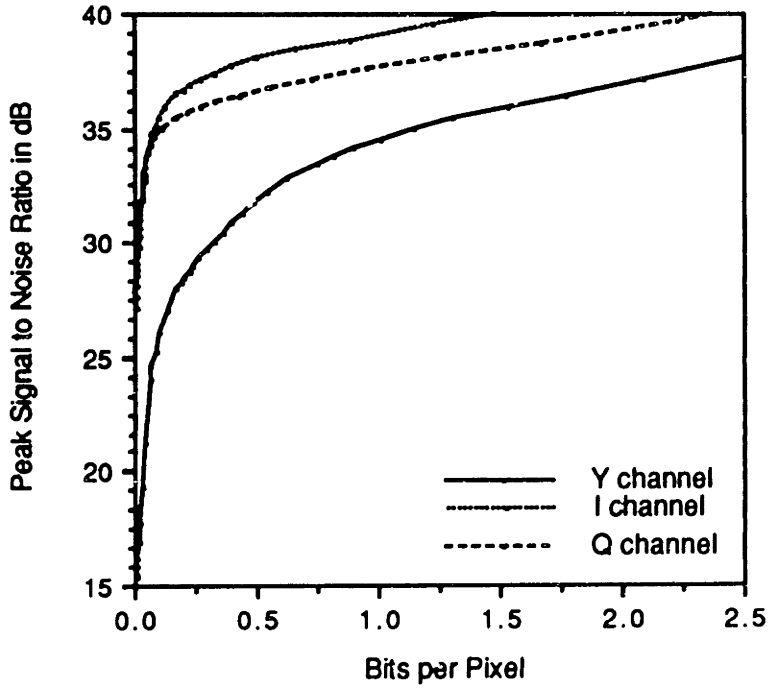
Graph [3.8-1]: RGB rate-distortion curves for RGB processing of the "Lena" image.



Graph [3.8-2]: RGB rate-distortion curves for RGB processing of the "frog" image.



Graph [3.8-3]: YIQ rate-distortion curves for YIQ processing of the "Lena" image.



Graph [3.8-4]: YIQ rate-distortion curves for YIQ processing of the "frog" image.

For a better comparison between RGB and YIQ performance some examples of color images are included. Photograph [8.1-37] shows color Lena compressed with an RGB representation. The overall bitrate is 2.0 bits per pixel and the specific channel contributions are listed in Table [3.8-1]. Photograph [8.1-38] shows the Frog image compressed with an RGB representation. The overall bitrate is 2.0 bits per pixel and the specific channel contributions are listed in Table [3.8-2].

Channel	Bitrate	PSNR
red	0.67	34.04 dB
green	0.67	31.47 dB
blue	0.66	31.60 dB

Table [3.8-1]: Channel contributions and quality for 2.0 bit per pixel RGB Lena image.

Channel	Bitrate	PSNR
red	0.69	33.16 dB
green	0.67	30.28 dB
blue	0.68	30.96 dB

Table [3.8-2]: Channel contributions and quality for 2.0 bit per pixel RGB Frog image.

This next set of images shows the efficiency of color image compression using a YIQ representation. (Because the range of I and Q values fell outside the 0 to 255 range they were biased and scaled to fit into this range.) Photograph [8.1-39] shows color Lena compressed with a YIQ representation. The overall bitrate is 2.0 bits per pixel and the specific channel contributions are listed in Table [3.8-3]. Photograph [8.1-40] shows the Frog image compressed with a YIQ representation. The overall bitrate is 1.9 bits per pixel and the specific channel contributions are listed in Table [3.8-4]. Comparing the RGB and YIQ images shows the definite quality advantage of the YIQ representation. In the YIQ images the finer detail in the image is visible because more bits can be allocated to intensity image, the Y channel. Also color spill between regions does not seem apparent as one might expect from the very low bitrates used on the I and Q channels.

Channel	Bitrate	PSNR
Y	1.51	37.53 dB
I	0.25	36.85 dB
Q	0.24	40.09 dB

Table [3.8-3]: Channel contributions and quality for 2.0 bit per pixel YIQ Lena image.

Channel	Bitrate	PSNR
Y	1.42	35.65 dB
I	0.25	37.03 dB
Q	0.27	36.74 dB

Table [3.8-4]: Channel contributions and quality for 1.9 bit per pixel YIQ Frog image.

The added compression efficiency gained by moving an image into an alternate color space makes such a transformation highly desirable. But transforming from RGB into YIQ or back to RGB from YIQ requires multiplying each pixel by a 3 x 3 matrix. This means that it would take 9 multiplies and 6 additions for each direction. Because this is too computationally expensive for certain applications, other less efficient color spaces are often used. One frequently used space is Y, Ry, By. In this space the three channels are intensity (Y) and the red channel minus Y and the blue channel minus Y. This space requires 3 multiplies per pixel and 4 additions per pixel to translate RGB to YRyBy. The inverse transformation has the same complexity. This is still a relatively complex operation. However, because NSI performs linear interpolation and the fact that these transformations are linear transformations, the computational burden can be reduced. This is because during decompression only the samples need to be transformed back into RGB and then can be interpolated in the RGB color space. For an image which has been compressed 8:1 this translates into a factor of 8 complexity reduction at decompression time. This means that a YIQ transformation would cost 1.1 multiplies and 1.1 additions per pixel. An YRyBy transformation would cost 0.38 multiplies per pixel and 0.5 additions per pixel. Also there is an added overhead of 9 additions per sample, which translates to 1.1 additions per pixel, needed to manage the conversions between slopes in RGB space and the alternate color space. Of course during image compression, the entire image would still have to be transformed into the alternate color space.

3.9 Using NSI to Scale Images

An interesting aspect of NSI is that during the compression process it performs an analysis of and imparts a structure to the image. This process can be useful for certain other types of processing performed on the image data. One very common type of processing performed on images is scaling. Images are often scaled so that they can fit into a specific geometry in a document or on a screen. However, scaling an image up can blur sharp edges in the image. This is because when images are scaled up "holes" (pixels with unassigned brightness values) are created between the original pixels which need to be assigned values. This can be seen in Figure

[3.9-1], the solid matrix represents the original pixel grid and the dotted lines represent the new pixel grid introduced by scaling the image by a factor of two. The simplest way of filling up these "holes" is pixel replication, that is, just assigning each new pixel the value of the pixel adjacent to it. This is illustrated in Figure [3.9-2] for the pixels in Figure [3.9-1]. This method does not perform well on smooth regions of the image. The problem is that in the original image, the pixel intensity values vary smoothly from pixel to pixel creating the appearance of a continuously shaded region. But pixel replication, in effect, increases the size of these pixels and can make their boundaries visible. In the scaled image these regions will appear blocky.

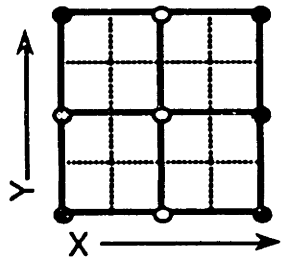


Figure [3.9-1]: Scaling an image by two, requires filling in the pixels at the intersections of the dotted gridlines.

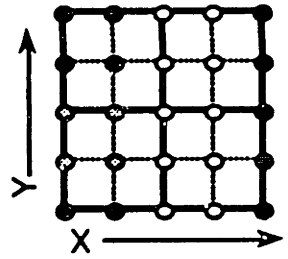


Figure [3.9-2]: Missing pixels generated by pixel replication.

Another method for filling in the "holes" is interpolation. That is, based on the surrounding pixels make a reasonable guess as to what the pixel value is. One common method used is bilinear interpolation. Here each pixel value is the weighted sum of the values of the four pixels surrounding it based on its distance from those individual pixels. NSI provides a means for reducing the computational complexity of bilinear scaling. This is because NSI already uses linear interpolation to fill in unsampled pixel values. This means that the slopes needed to fill in scaled pixels values which fall on the rows and columns of the original image processed by NSI are already calculated by NSI during the decompression process. Therefore, if image decompression is combined with scaling, certain slope calculations can be avoided.

Bilinear interpolation handles smoothly shaded regions well because each pixel in the scaled image is shaded continuously. This algorithm, however, does not do a very good job

scaling high contrast regions of the image with sharp edges. This is illustrated in Figure [3.9-3] which is a scaled version of the pixels in Figure [3.9-1]. If a sharp edge occurs between two pixels in the original image, bilinear interpolation will smoothly shade the region between those two pixels. But, if in fact, it was known that an edge existed there, then if pixel replication was used to scale that portion of the image, then the sharp edge would have been correctly scaled.

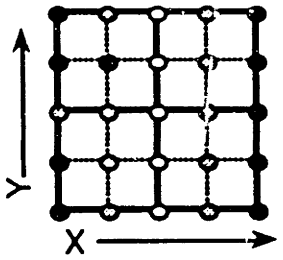


Figure [3.9-3]: Missing pixels generated by bilinear scaling.

Obviously if the locations of these sharp edges in the image were known at the time of the scaling then the different scaling algorithms could be correctly applied in the different cases. This analysis can be performed before scaling an image, but is time consuming. However, during the compression process, NSI performs just such an analysis by its choice of sample points. Because a piecewise linear interpolation of the image intensity waveform is being made, each sample point location corresponds to a change in the slope of the image intensity waveform. If two of these samples are placed very close together, then, most likely, a discontinuity or a sharp edge occurred somewhere between those two sample points. Just such a situation is illustrated in Figure [3.9-4].

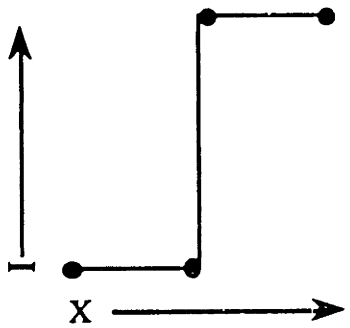


Figure [3.9-4]: Samples taken by NSI before and after an edge.

Therefore scaling an image which has been compressed using NSI seems extremely simple: Use linear interpolation to scale between sample points, unless two sample points are very close together, then assume that a sharp edge occurs directly in the middle of the two points and use pixel replication to fill the pixels in between such that a sharp edge appears in the center

of the two sample points. The only problem with this is that edges occur in both directions (horizontally and vertically), but NSI only processes the image one dimensionally, in one direction at any one time. In the simple case where each scan line is processed independently, if any horizontal edge occurs in the image then it will be missed and cannot be enhanced during scaling. This is seen in Figure [3.9-5] where the solid line represents an edge in the image, the dotted lines the scan path, the circles pixels and the triangles regions of blur.

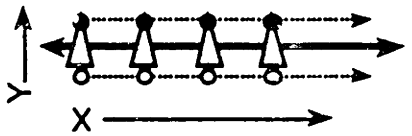


Figure [3.9-5]: An undetected horizontal edge causing blurring in a scan line version of NSI.

When the two dimensional version of NSI is used to process the image in bands, this problem is not as bad. This is because the image is processed both in the vertical and the horizontal directions. Because the vertical columns processed between the horizontal rows processed use the pixels in these rows as the end points of the columns, no horizontal edge will be missed. Figure [3.9-6] shows how samples, pixels surrounded by squares, will be chosen on either side of the edge.

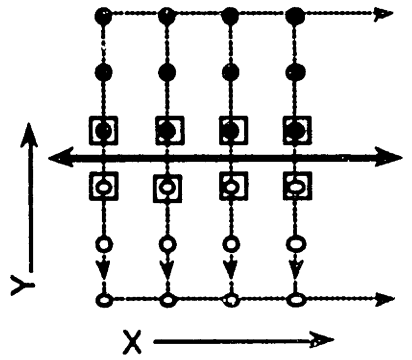


Figure [3.9-6]: Every horizontal edge is detected by the scan path. (Boxes represent sample point locations.)

The problem now is with vertical edges. If a vertical edge intersects a row, then it will be detected and can be enhanced on that row only. However, as an edge travels between the columns, it will not be detected. Figure [3.9-7a] shows how samples are not taken on both sides of the vertical edge, causing it to be blurred. The way to fix this problem is to propagate the edge information from rows bordering each band to the columns in it. If an edged is sensed at the top of a row, it can be propagated down the columns until a sample is taken in the column, which

indicates that the edge nas crossed out of that column and should no longer be enhanced. Similarly, information can also be propagated up a column. Figure [3.9-7b] shows how this can improve the enhancement, shaded regions indicate groups of pixels effected by edge information propagation. Unfortunately, this does not fix every situation. The situation in Figure [3.9-8a] is still not corrected. This is the case where a vertical edge exists, but does not traverse any of the pixels processed as rows. This is the situation created by small text in the image. The situation in Figure [3.9-8b] also is not fixed. Here a diagonal line is not properly enhanced. Another rule for scaling can be added which helps fix some of these problems. That is whenever two samples are adjacent vertically in a column, that since there is a horizontal edge, it must accompanied by a vertical edge nearby, therefore treat the pixels on the left and right of the pair as sharp edges. Because the band height is limited this will correct many small edges and diagonal lines. Certain diagonal lines which are nearly vertical will still not be corrected because they will not cause samples to be taken in an adjacent manner in columns. Of course this "fix" will also generate blockiness in a class of horizontal edges which have various continuous pixel values on both sides of the edge, which are rare.

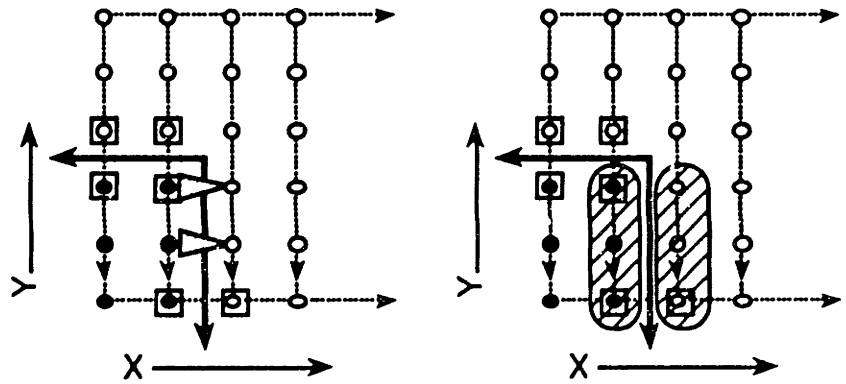


Figure [3.9-7]: (a) A vertical edge going undetected. (b) Edge propagation improves enhancement.

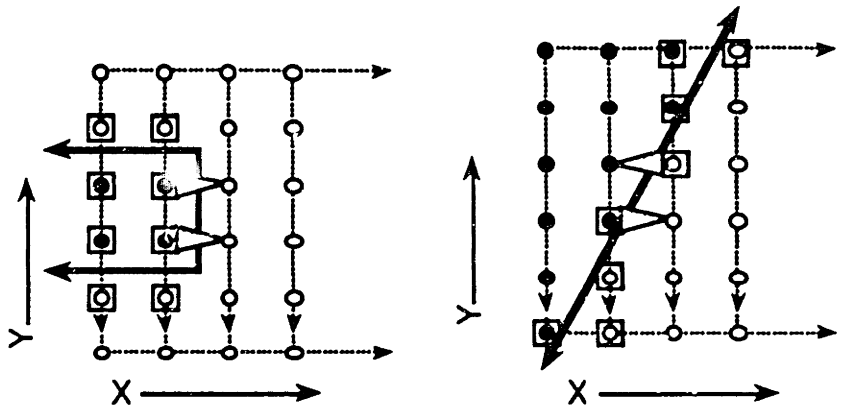


Figure [3.9-8]: (a) An edge too small to be enhanced. (b) A diagonal line is not correctly enhanced.

Pixel replication is not the complete answer to scaling edges. If pixel replication is used to scale curved or diagonal edges in the original image, then these edges will appear blocky in the scaled image, as can be seen in Figure [3.9-9a]. This type of artifact is called aliasing. The appearance of these scaled edges could be improved if the edges could be detected and then anti-aliased. Fortunately, NSI also provides a means of doing this. If bilinear interpolation is first used to scale the image and then the pixel values between enhanced samples are thresholded, based on the average of the enhanced sample values, then in effect, the edges will be anti-aliased. An anti-aliased edge can be seen in Figure [3.9-9b].

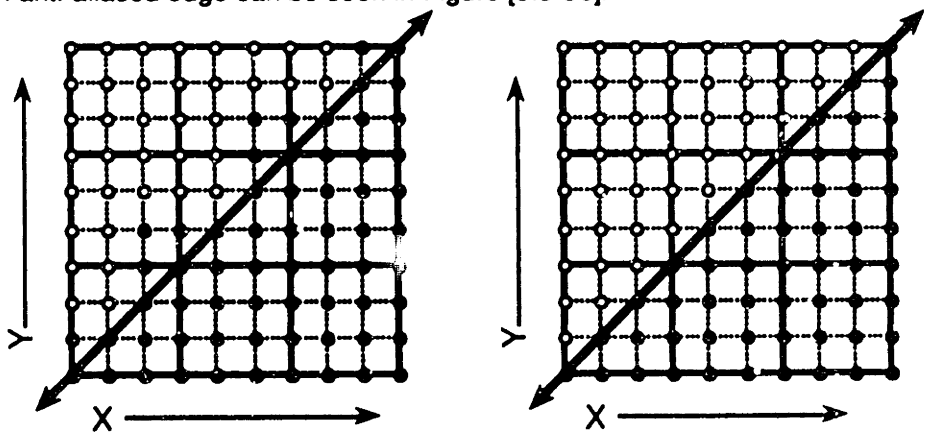


Figure [3.9-9]: (a) An edge aliased by enhanced scaling. (b) The same edge after being anti-aliased.

To evaluate this scaling algorithm's image quality on high contrast edges, two binary images were compressed and processed. One is a thresholded version of some text from the Smag image, pictured in Photograph [8.1-41], this image will be referred to as "Text." The original image here is shown magnified so greater detail can be seen. The second is a thresholded version of the eye region of the Lena image, pictured in Photograph [8.1-42], this image will be

referred to as "Eye." The original image here is magnified so that finer detail is visible. Photographs [8.1-43] and [8.1-44] show the samples chosen by NSI for the Text and Eye images, respectively. These are included for reference because the sampling pattern effects which pixels will be scaled in what manner. (Note that when the Text and Eye images were compressed by NSI, they were compressed losslessly because they consisted entirely of regions which could be encoded as piece-wise linear approximations.) Photographs [8.1-45] and [8.1-46] show the effect of using NSI to bilinearly scale the Text image by four and the Eye image eight, respectively. As should be expected, this type of scaling blurs the edges and makes the images appear to be out of focus. Photographs [8.1-47] and [8.1-48] show the effect of using NSI to scale and enhance, through pixel replication, the Text image by four and the Eye image by eight, respectively. The scaled images appear blocky, but the algorithm did effectively find the edges in the image to enhance, only missing a small percentage which does not appear to be an objectionable artifact, subjectively. Photographs [8.1-49] and [8.1-50] show the effect of using NSI to scale and anti-alias the Text image by four and the Eye image by eight, respectively. In these images edges have a more pleasant rendition, being sharp but not as jagged as the straight enhancement scaling. The only artifact noticeable in the images scaled by anti-aliased edges is some "notching" caused by round-off errors in the thresholding process.

Both of these images show the effect of these NSI scaling algorithms on scaling high contrast images. The interaction of the NSI scaling algorithm with images consisting of high contrast regions and continuous tone regions is demonstrated in the next section.

One apparent problem with this method of enhancing scaled edges is its dependence on the distance between sample points, which varies according to the compression rate. This will have an effect on which regions of the image will be enhanced during scaling, but not a major one. This is especially true if enhancement is limited to occur between adjacent pixels. Since, images will typically be compressed by more than 2 to 1, most samples will have to be at least one pixel apart. For this reason most of the pixels in an image will never be enhanced.

3.10 Combination with Subband and Pyramidal Coding

Inspired by the good performance of the NSI anti-aliased scaler, other applications were sought for this scaler. One application which came to mind was using the scaler in conjunction with subband or pyramidal coding. In this class of algorithms an image is split up into low frequency and high frequency components. The low frequency component is then subsampled and the process can be repeated recursively, generating successively "lower" sublevels of the image. The hope here is that the NSI anti-aliased scaling algorithm will allow the high frequency components in the higher sublevels to be synthesized in a reasonable way from the high

frequency components of the lower sublevels. The are a number of possible applications of such a capability. One is a better rendition of an image in progressive transmission systems. In systems which employ progressive transmission the lower sublevels are sent first to generate a preview of the image. The hope is that if the higher sublevels can be synthesized then an image with a higher apparent resolution would be generated and the progressive transmission of the image could be terminated at a lower sublevel. Another application is better compression rates. If the some of the higher sublevels can be discarded and synthesized from lower sublevels, with little degradation of image quality, then overall bitrate can be reduced. A third application is better scaled image quality. It is possible that scaling the high and low frequencies of an image separately may result in better scaled image quality then just using NSI to scale the image directly.

There are many avenues of inquiry possible in this area of sublevel scaling, but, because of time constraints, only a cursory investigation was performed. There are many areas which were left unexplored and could be the focus of future inquiry.

The scaling of the high frequency sublevel information is not a unique idea. In [Holtzman 90], a pattern classification technique is used. During sublevel generation a pattern table is generated which relates image patterns in the high frequency bands between sublevels. This table is used to scale the lower level bands with enhancement to generate the higher level bands. The author states that subjective image quality was found to improve significantly over straight Gausian scaling of the images. The following paragraphs explore an alternate way of performing this scaling and sublevel synthesis through NSI.

Subband coding is a means by which images are separated into four frequency components or subbands.[Woods and O'Neil 86] The four subbands correspond to the following aspects of the image: the DC information, the horizontal high frequency information, the vertical high frequency information and the "diagonal" high frequency information. Each of these subbands images is one quarter of the size of the original image. This process can be repeated recursively to the DC subband to generate successively "lower" subbands. Subband coding most often utilizes quadrature mirror filters and has been found to out perform many other techniques in the image compression domain.[Woods and O'Neil 86; Simoncelli 84]

To test NSI's performance in this domain the image in Photograph [8.1-51] was used, it will be referred to as "Moonlight." Three levels of subbands were then generated, where level 0 is the original image and level 3 is the lowest group of subbands. Table [3.10-1] shows the bitrates and qualities achieved for compressing level 3 subbands with NSI. The level 3 subbands were scaled by 2 using NSI to form the level two subbands and by four to form the level 1 subbands. Finally, the full sized image (level 0) was completely reconstructed from the synthesized level 1 and level 2 subbands and with the unmodified level 3 subbands, with the results shown in Photograph [8.1-52]. These results show that these efforts were not completely

successful. The image does have a good appearance yet is plagued with ringing at edges. A possible explanation for the ringing is a phase distortion generated in the higher frequency subbands by the NSI scaling algorithm. This distortion does occur with the version of NSI scaling used here because sample values are placed at the edges of the scaled regions they influence instead of at the centers. Theoretically this could be corrected but there was not enough time, in this investigation, to do so. From a statistical standpoint, the image in Photograph [8.1-52] is 0.55 bits per pixel and has a PSNR of 28.56 dB. The quality of this image is significantly lower (3.5 dB) than what one would expect by just compressing an image at this bitrate. The phase distortions contribute to this low PSNR in two ways: first, they cause the ringing at edges and second PSNR penalizes phase shifts immensely. PSNR generates this penalty because it measures squared differences so that a small shift in an edge will create a large statistical error but not much of a subjective one.

Subband	Bits per pixel	PSNR
horizontal	1.01	38.04 dB
vertical	1.10	34.53 dB
diagonal	1.13	41.99 dB

Table [3.10-1]: Compression rates and quality for "Moonlight" third level subbands.

The windowed Gausian filters used in pyramidal coding are not as phase sensitive as the quadrature mirror filters used in subband coding. In pyramidal coding, an image is low pass filtered by a windowed Gausian to generate a low pass band. This low pass band is then subtracted from the image to generate a high pass band. The low pass band is then subsampled to one half size. The smaller low pass band can go through the process repeatedly, resulting in lower and lower levels of the pyramid in a manner analogous to subband coding. This processing of an image has found to be extremely useful in image compression as well as other types of image processing.[Adelson and Anderson and Bergen and Burt and Ogden 84]

To test NSI scaling performance in conjunction with pyramidal coding, a three level Gausian pyramid was generated and the lowest level (one quarter size) image was used to generate the higher levels. Photographs [8.1-53] and [8.1-54] show the low and high frequency portions of the third level pyramid of the Moonlight image, each has been magnified, to make detail more visible. So that scaling can take place, the high frequency subband was compressed by NSI. It was compressed to 2 6 bits per pixel and a PSNR of 36.84 dB. For comparison, the third level subband was also scaled using the NSI scaling algorithm directly. To facilitate this, the image in Photograph [8.1-55], which is the composite of the information in Photographs [8.1-53]

and [8.1-54], was compressed and scaled. It was compressed to 4.08 bits per pixel and a PSNR of 39.53 dB.

The images summarize the results. Photographs [8.1-56] and [8.1-57] show the result of scaling by two using NSI combined with pyramidal coding and using NSI directly, respectively. Both of these images have been magnified using pixel replication so that the detail can be better examined. The first thing that is obvious is that these images are a better rendition than either of Photographs [8.1-53] or [8.1-55]. Edges are much sharper in these scaled images. Comparing these two images is difficult, both do a respectable job of high-frequency synthesis. Each has its own specific artifacts and it seems doubtful than one could be classified as "better" than the other. Photographs [8.1-56] and [8.1-57] are the result of scaling the lower pyramid level by four back to full size, using NSI combined with pyramidal coding and using NSI directly, respectively. At this level it seems as though NSI does a better scaling job than the synthesized pyramid which contains more serious smears and blotches. But again, both techniques do a reasonable job of generating apparent resolution from nothing.

The statistical evaluation of this experiment is summarized in Table [3.10-2]. An examination of the scale by two figures is rather disappointing. At at the 1.0 bit per pixel bit rate, most algorithms can give better quality than this. (NSI gives a PSNR of over 35 dB.) One possible explanation for the poor performance is the phase shift induced by NSI scaling. PSNR penalizes a phase shift of an edge severely because it squares the difference. At this level of scaling, both algorithms seem to perform similarly in terms of quality, the hybrid pyramidal scaling performing slightly better. The really interesting result is from scaling by four. Here the quality remains stable when scaling up. For the direct NSI scaling, the quality even improves slightly. This is interesting because it implies that over a reasonable range of scale factors, NSI based scaling (both the pyramidal hybrid and the direct version) is a reliable way of enhancing the resolution while scaling images. Compared to NSI compression of this image at a similar bitrate, one would expect the quality to be 2 dB worse. This means that there may be something to gain by using these scaling based algorithms in conjunction with compressing images to obtain better compressed image quality. In fact it hints at the possibility of generating a "resolution independent" representation of an image of constant quality.

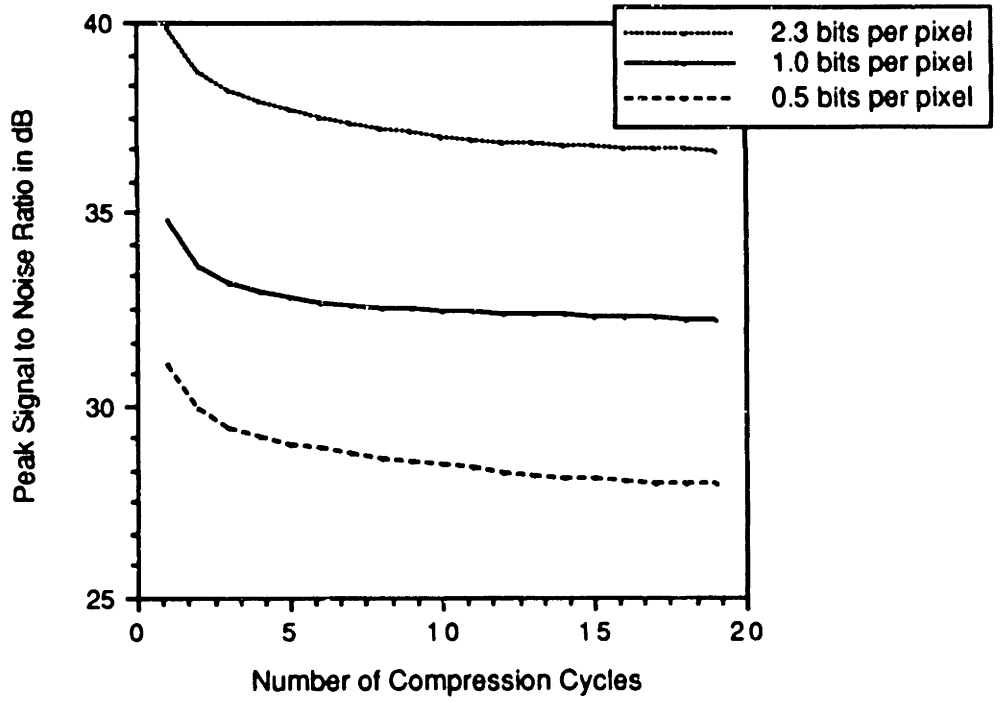
Scale	Algorithm	Bits per Pixel	PSNR
x2	Pyramid and NSI	0.9	30.44 dB
x2	NSI direct	1.02	29.88 dB
x4	Pyramid and NSI	0.23	29.53 dB
x4	NSI direct	0.26	30.02 dB

Table [3.10-2]: NSI pyramidal scaling performance.

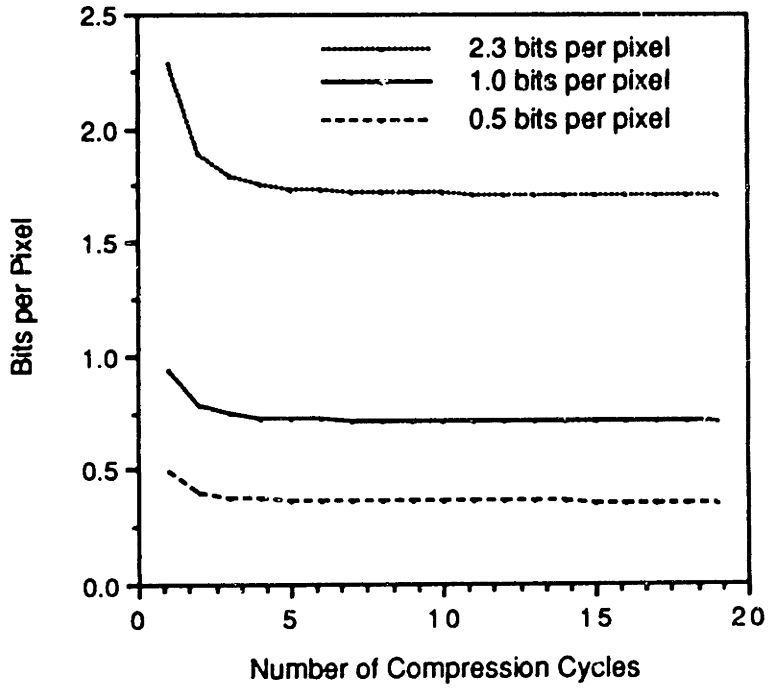
This experiment shows that the NSI scaling algorithm can be useful for scaling higher frequency subband information in situations where the filters in use are not phase sensitive. It also shows is that the NSI scaling algorithm does a fair job of scaling images which contain both sharp edges and smoothly shaded areas.

3.11 Multiple Compression Passes

In certain applications a compressed image is decompressed and then modified and then recompressed. It is therefore advantageous that an image compression algorithm be able to compress and decompress an image through multiple cycles without a significant degradation in quality through each cycle. Graph [3.11-1] is a graph of the "Lena" image passed through nineteen compression-decompression cycles at 0.5, 1.0, and 2.3 bits per pixel. The vertical axis is image quality and the horizontal axis is compression cycles. What is apparent from this graph is that image quality degrades asymptotically, the biggest drops happening during the first 6 compression cycles and then levelling off to a near negligible decline. This trends seems to be similar at all three bit rates. Graph [3.11-2] is a related graph which plots bit rate on the vertical axis and the number of compression cycles on the horizontal axis. Here bit rate takes a small drop during the first three cycles and then seems to level off. What is apparent from both of these graphs, is that image quality takes a significant drop of 2-3 dB during the first few compression cycles. The reason that this occurs is that the error metric threshold is kept constant during all of these cycles. Once the image is compressed and decompressed once, it should be able to be compressed again losslessly by NSI. This is because it should find all the line segments in the reconstructed waveforms and be able to find all of the sample points exactly. In practice this is not the case, as the reconstructed intensity values are calculated to a limited precision during decoding. Because of this round-off, error will accumulate during the compression process.

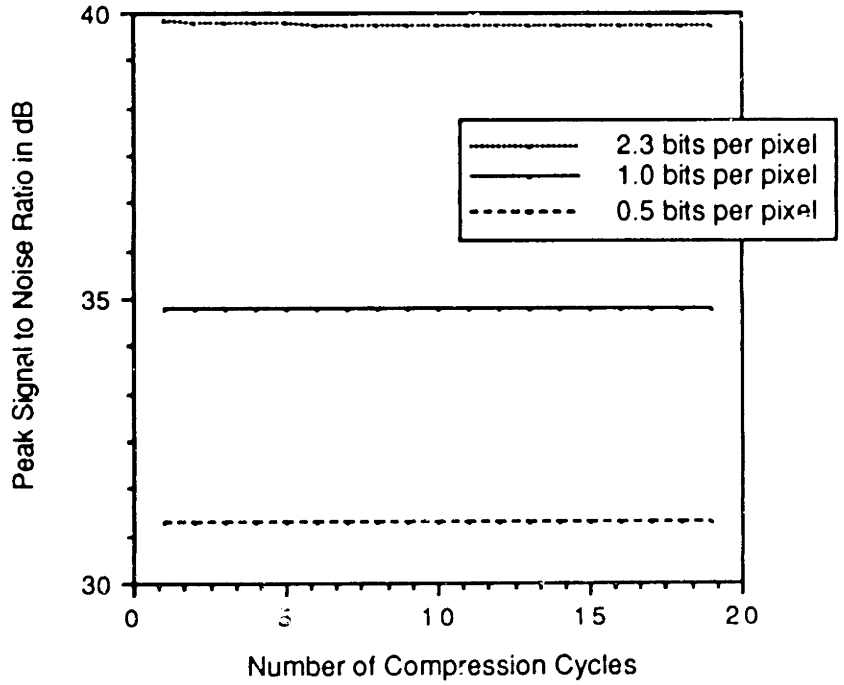


Graph [3.11-1]: Plot of image quality over 19 successive compression-decompression cycles with a constant error threshold for three bitrates.

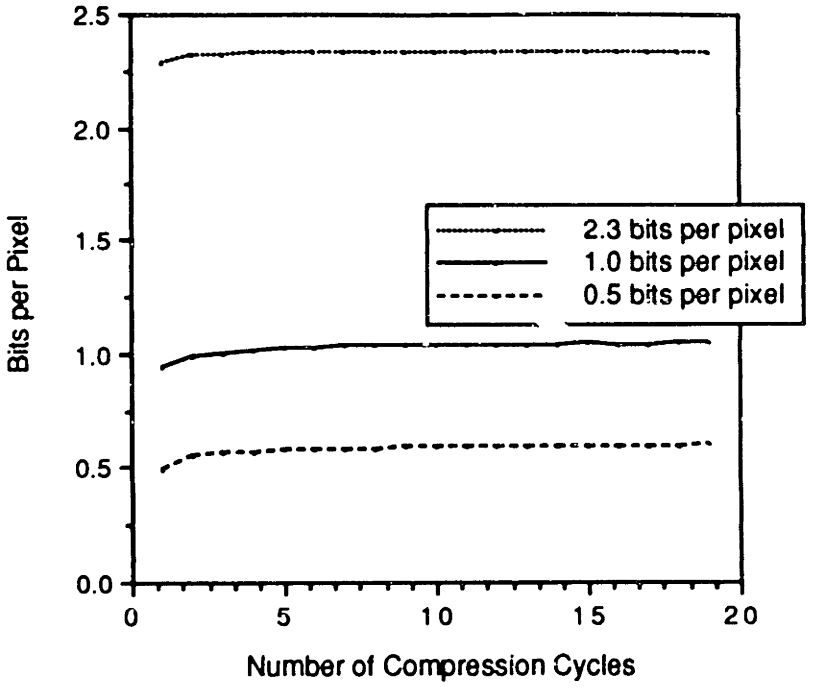


Graph [3.11-2]: Plot of compression rate over 19 successive compression-decompression cycles with a constant error threshold for three bitrates.

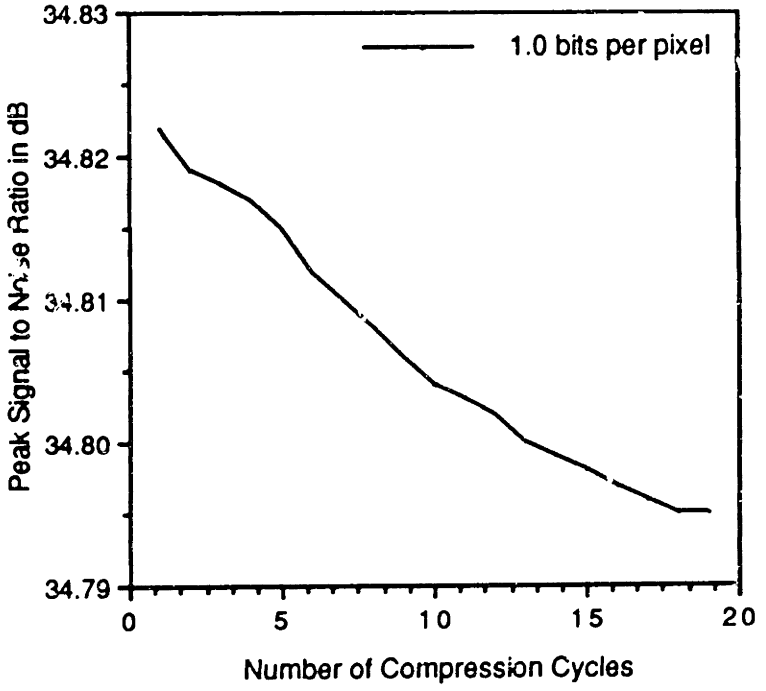
Graph [3.11-3] plots PSNR on the vertical axis and compression cycles on the horizontal axis. After the first compression cycle, the error metric threshold was set to four. Four was chosen because it would allow 16 round off errors (0.5) to accumulate in a row. (0.5 is squared, consequently 16 times 0.25 is 4.) As the Graph shows, there is no visible reduction in image quality on this scale. Graph [3.11-4] plots compression cycles versus compression rate. This Graph shows that there is approximately a 0.1 bit per pixel increase in the first compression cycle and then no appreciable increase occurs. Graph [3.11-5] is a plot of quality versus compression cycles centering on the 1.0 bit per pixel data. This seems to show a relatively linear decrease in quality, of 1.5 thousandth's of a decibel, an unobservable amount. Graph [3.11-6] zooms in on the same data, plotting compression rate versus compression cycles. This Graph shows that bitrate seems to increase asymptotically, averaging about 0.2 bits per pixel (3500 bytes) during the first 10 compression cycles and then decreases, to less than 0.01 bits per pixel (10 bytes) in subsequent cycles.



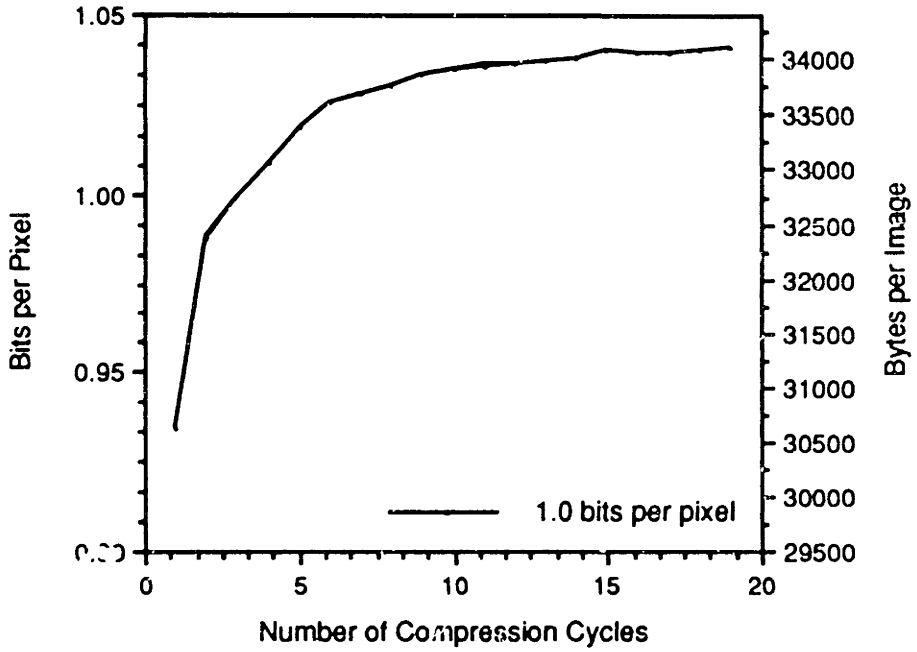
Graph [3.11-3]: Plot of image quality over 19 successive compression-decompression cycles with a small error threshold for three bitrates.



Graph [3.11-4]: Plot of image quality over 19 successive compression-decompression cycles with a small error threshold for three bitrates.



Graph [3.11-5]: Magnified plot of image quality over 19 successive compression-decompression cycles with a small error threshold for 1.0 bits per pixel.



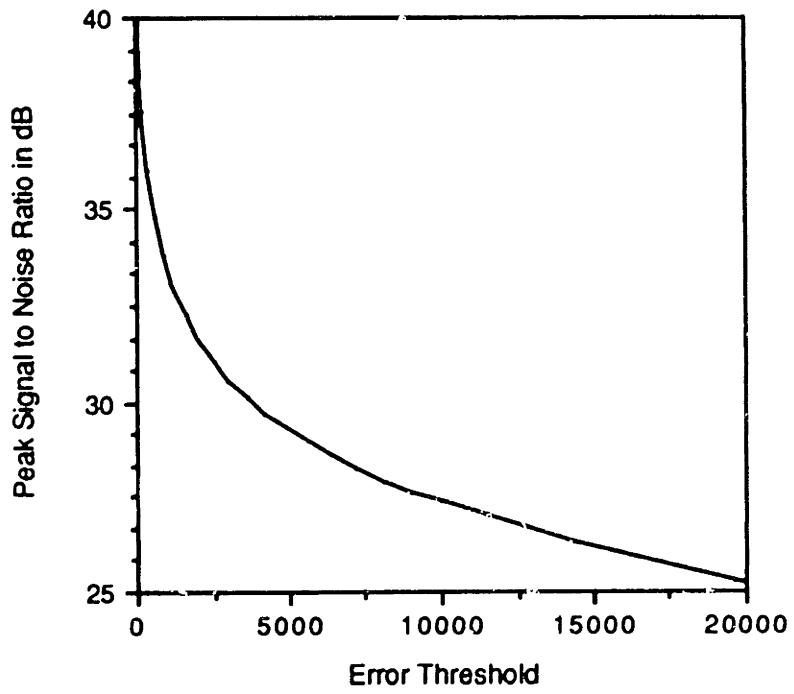
Graph [3.11-6]: Magnified plot of image compression rate over 19 successive compression-decompression cycles with a small error threshold for 1.0 bit per pixel.

This data argues, extremely strongly, that NSI performs very well in a situation where images go through multiple compression-decompression cycles. In order for this level of performance to be achieved, an image should be flagged as previously compressed, so that the error threshold can be appropriately set. Selecting a threshold is not as simple if the data is modified when decompressed. If the modification is limited to a small region of the image then a small threshold could be used for the unmodified part of the image and a larger threshold for the modified part of the image. If the entire image is modified, one would still assume that the modifications made would not add a significant amount of noise to the image so that the chosen threshold could be smaller than normal.

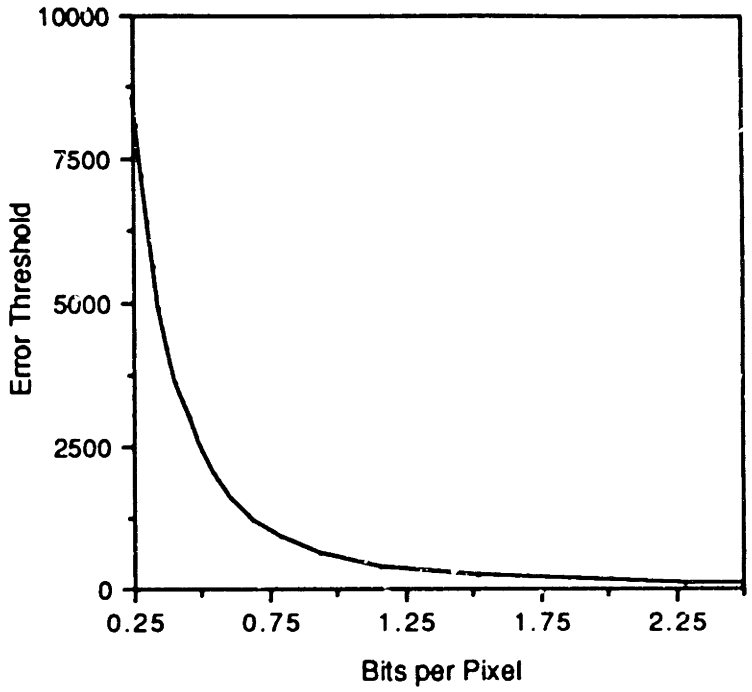
NSI's good performance in this area due to the fact sample point jittering ensures that sample points will be placed on the edges when the decompressed image is compressed again. The slight quality degradation can be accounted for by samples being removed from areas which were originally noisy and the merging of line segments with relatively close slopes.

3.12 Threshold vs. Bitrate

One characteristic of NSI is that it is supposed to be a quality controlled algorithm. That is, the error threshold of the image is supposed to be related to image quality. Graph [3.12-1] plots image quality on the vertical axis and the error threshold on the horizontal axis for the "Lena" image. Image quality, improves in an exponential manner in an inverse relation to the error threshold. Graph [3.12-2] plots compression rate on the vertical axis and error metric threshold on the horizontal axis for the same three images. As would be expected, compression rate falls off very quickly, in an exponential manner, as error threshold is decreased.



Graph [3.12-1]: Plot of error threshold vs. image quality.

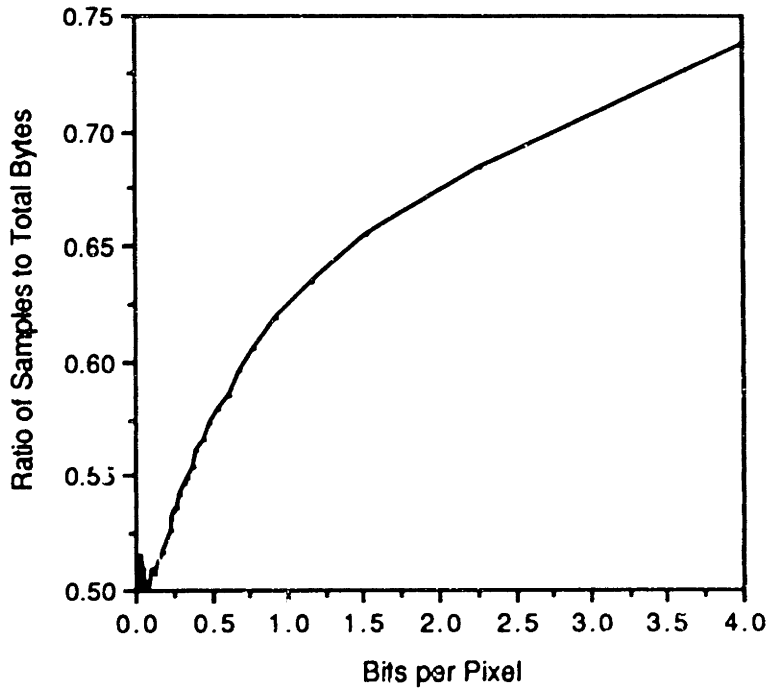


Graph [3.12-2]: Plot of error threshold vs compression rate.

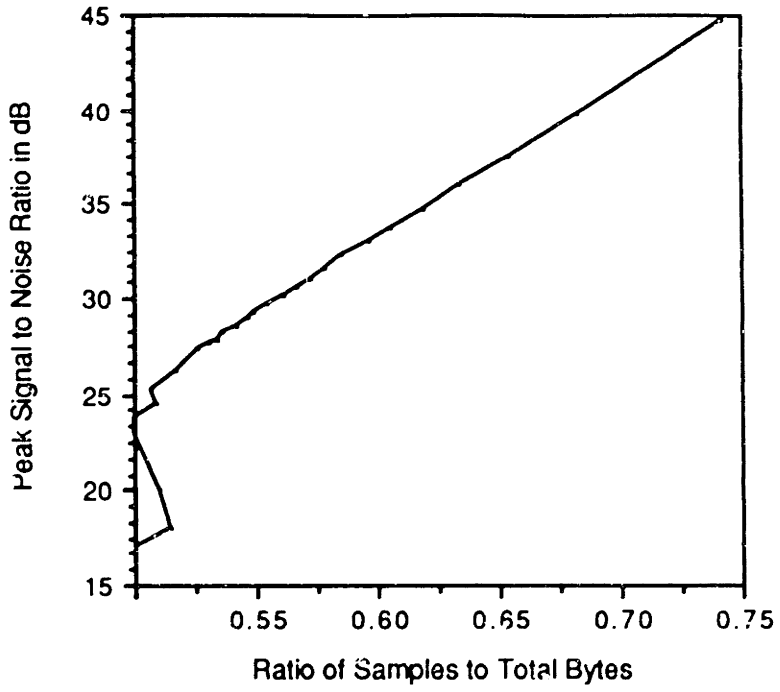
3.13 Sample vs. Positional Information

One interesting thing about an image compressed by NSI is that part of the compressed data is the actual pixel values and some of it is positional information. It is informative to look at that composition, because it predicts the effect of better compression on the positional information on reducing overall bitrate. Also, decompression time and overall decoding computational complexity per image depend on the number of samples, not the total number of bytes which make up the compressed image. Graph [3.13-1] plots the ratio of sample bytes to the total number of compressed bytes on the horizontal axis and compression rate in bits per pixel on the vertical axis. There are a number of interesting things about this graph. The first is that the lower bound of the ratio seems to be a 50-50 composition of sample bytes to positional bytes. Another is that compression rate is is directly related to this ratio. This is expected because at higher compression rates, the line lengths are longer per sample and there are fewer of them. Whereas at lower compression rates, the line lengths are shorter per sample and there are more of them. The extremely interesting thing, though, is that the shape of the curve on this graph looks extremely similar to a rate-distortion curve. Prompted by this Graph [3.13-2] plots PSNR on the vertical axis and the sample ratio on the horizontal axis. Plainly, there is a linear relationship between PSNR and the ratio down to when the ratio first hits 50%. The question is, how does this

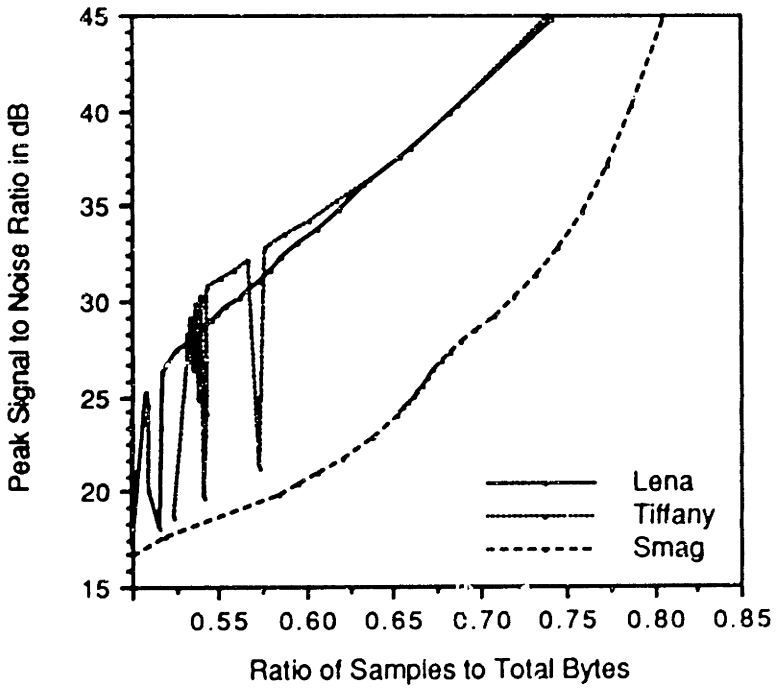
relationship generalize to other images? Graph [3.13-3] answers this question. This is a plot of sample data ratio versus PSNR for three images: Lena, Tiffany, and Smag. What is surprising is that except for a small number of outliers, Lena and Tiffany follow the same line. Smag also seems to follow a practically linear course once it gets above a particular ratio. Although is an extremely interesting relationship, the reason why it exists is not clear.



Graph [3.13-1]: Plot of the of ratio of sample bytes to total data versus compression rate.



Graph [3.13-2]: Plot of the ratio of sample bytes to total data versus image quality.



Graph [3.13-3]: Plot of the ratio of sample bytes to total data versus image quality for three images.

4. Conclusions

An algorithm has been developed which brings the quality of a redundancy reduction style algorithms close to that of transformed based techniques. The investigated algorithm NSI is within 2.5 dB of quality of a DCT and is able decode an image approximately 24 times faster. It also exhibits the useful property of speeding up standard scaling and color transformations. It also has the feature that it can enhance during scaling to improve scaled image quality. NSI also able to compress and decompress an image repeatedly without significantly distorting the image. These characteristics make NSI extremely attractive for an environment where decoder complexity and decoded image quality are of primary concern.

5. Summary

An initial survey of existing image compression algorithms showed that spatial domain techniques were most amenable for use on low power computer because of the low computational complexity of decoding the image. The problem, however, is that the majority of these techniques result in image quality significantly less than that of transform based techniques. Of the spatial techniques reviewed it seemed as though redundancy reduction techniques had the most promise of improved image quality. Accordingly, the failings of the current state of these algorithms were investigated and attempts were made to remedy them. There were three significant extensions. The first was sample point jittering which improved image quality and especially edge fidelity by repositioning sample points so as to minimize local error. The second was sample point look ahead which helps limit the number of unnecessary sample points taken by the algorithm due to noise. This was accomplished by extending the approximating line to see if the approximation could once again be brought into bounds. The third extension was an alternate scan path for processing the pixels to allow the algorithm to exploit two dimensional image correlation, which processed the pixels in horizontal bands. The tops and bottoms of the bands bands are processed as rows and the pixels in between as columns. The performance of each of these extensions was evaluated by generating rate-distortion curves based on peak-signal to noise ratio as a measure of image quality. Finally the quality of the algorithm (NSI) was compared in terms of speed and quality to that of the DCT. NSI was found to be approximately 2.5 dB in terms of quality worse, 24 times faster in decoding and 48 times slower in encoding.

6. Future work

During the course of this investigation many possibilities for extending the current work developed. This section summarizes some of the more significant of these.

6.1 Algorithm Hybridization

One of the major weaknesses of NSI as it stands is its performance in textured regions of the image. In many cases this level of performance is unacceptable. One possible way to improve performance is to combine NSI with another algorithm which can efficiently code textures. Since it is still desirable to keep decompression complexity low, a spatial domain technique seems like the most reasonable choice.

The best candidate would seem to be block truncation coding. Block truncation coding is extremely fast in encoding and decoding. It is also extremely good at representing textures. This is because a textured region will most often be composed of two different brightness levels which can be encoding as one of the two values used by block truncation coding.

To make block truncation coding more amenable for use in the context of NSI it could be modified to operate on a one dimensional basis. In this case a particular line segment would be coded using this one dimensional version of block truncation coding instead of the standard interpolation technique. A special flag value could be inserted in the sequence of samples which would tell the algorithm when the block truncation coding method should be used.

The difficult thing to do is to decide when to use block truncation coding. One way to do this would be to examine the difference signal between the approximation and the original image. Presumably the error in textured regions is relatively uncorrelated with the approximation and the error in texture. Whereas the error in flatter regions will be more correlated with the approximation.

6.2 Human Visual System Extensions

Throughout the field of image compression it is widely accepted that better subjective image quality are possible at any given compression ratio if the characteristics of the human visual system is taken into account in the compression algorithm. These characteristics are the spatio-temporal characteristics and the contrast sensitivity function of the human visual system. The use of this information allows an algorithm to spend more of its bits in the regions of the image which are more significant for the human observer.

NSI as it now stands does not take human visual system characteristics into account. One relatively straight forward way of incorporating these characteristics into the algorithm are in the error metric. The sum of squared error metric seems to be an acceptable statistical measure of fit and a coarse measure of subjective image quality as demonstrated in the subjective experiment

described earlier. It seems reasonable, though, that an error measure which included HVS characteristics could generate an image which has better apparent quality, possibly at the cost of signal to noise ratio.

Another way to include HVS characteristics is via adaptive quantization. Currently, if the original image data was eight bits per sample, then the samples will be stored at eight bits per sample. However, it is known that characteristics of the human visual system are such that the human eye is much more sensitive to quantization error in flat regions of the image, versus high contrast regions of the image. Given this observation, it should be possible to modulate the quantization of the sample data based on previous distances between the samples. Short distances imply sharp edges whereas long distances imply flat regions.

6.3 Better sample point choices

Even though many of the optimizations developed in this thesis refine the sample selection process, the sample points chosen should in no way be considered optimal. There are many methods which could further improve the sample point values chosen, as will be described in the following paragraphs.

6.3.1 Sample points off of the image surface

The first possible improvement is to choose sample points off of the image surface. It seems most likely that the image regions where this will be most useful will contain high frequency noise. Usually, in these regions the slope pitches pack and forth over the intensity surface. The image would look better if just one smooth slope was maintained. In these regions the sample values could be changed to flatten out the approximation. The problem is to automatically decide which regions of the image should get this treatment. The process could be similar to that of deciding where block truncation coding should be used. The error between the approximation and the original image can be examined. If the error is decorrelated then that is one sign of texture. Another sign is the ratio of the positive to negative error. In this case of a curved region, most of the error will be of the same sign. However, in the case of a textured region the error will frequently alternate between positive and negative values.

6.3.2 Better band coherence

One problem with NSI is the fact that there is little coherence between bands. There does not seem to an obvious way to eliminate this problem

6.3.3 Better error metric

The sum of squared error metric is most probably not the optimal error metric. One choice of a better class of error metrics are those which take human visual system characteristics into account, as described in a previous section. Another possibility is that the slope of the approximating line or its length modulates the scale of the error metric in some manner. As has been stated previously, one of the problems with the current algorithm is that as a flat line grows longer and longer it will tend to accumulate error and cause a samples to be taken which are not necessary. Modulating the error metric could help reduce that problem.

6.3.4 Sample point removal

It is possible that during the compression process sample points were chosen such that adjacent segments ended up with approximately the same slope. In this case, the sample point between the segments could be removed and the resulting segment would have approximately the same slope and would not greatly increase overall image error. This process of "line segment" joining would require some decision criterion. One simple criterion could be a simple error bound on the difference of the slopes of the two segments. If the slopes are within the error bound then the segments could be joined. Another possible criterion is comparing the total of the two sum of squared errors generated by the two separate segments to that of approximating that data with one segment. If the errors are within a certain bound of each other then the middle sample point can be removed.

6.4 Coding of motion sequences

NSI would seem to be well suited to the coding of motion sequences because its decoding complexity is so low. There are certain aspects of NSI which might hamper this application. The major aspect is edge buzziness. Since the position of sample points in the image can be dependent on image noise and local image characteristics, it seems likely that the positions of these samples might wander between frames, even if the objects are reasonably stationary in the frame. This would result in an artifact called edge buzziness where edges in the image seem to waver. A related problem would occur in textured regions. In these regions the pattern of the aliasing noise is highly dependent on small intensity fluctuations in the image. It is theorized that textured regions would exhibit a changing pattern over time would be highly objectionable.

Capturing temporal correlation using NSI should be possible using a three dimensional extension of the banding scheme. In this three dimensional extension, individual frames could be stacked to form a "thick" frame, the thickness in the time axis. Every nth frame could then be processed by standard NSI. Then the frames between these "key" frames could be processed in a manner similar to the columns processed between bands of a single image. The major

disadvantage in this scenario is that each individual frame cannot be decoded without context from another frame, which doubles memory requirements in the decoder.

The problems described in this section make it unlikely that NSI will perform in an acceptable manner in the context of coding motion video.

6.5 Bitrate control

In many applications it is advantageous to be able to specify a compression rate at which an image need be compressed. As NSI is currently described, it is a quality controlled algorithm. That is, the only specifiable parameter, which is the error metric threshold is directly related to image quality. It should, however, be possible do adapt NSI so that a particular bitrate can be specified and the algorithm can deliver that bitrate.

The way to introduce bitrate control is via feedback. The feedback process would make a guess at an initial error metric threshold and then attempt to compress a portion of an image. If the bitrate was too low then the threshold could be raised. If the bitrate was too high then the threshold could be lowered. The algorithm could then continue to compress the image, continually adapting the threshold. Or, alternately, it could throw away the work it had done compressing the block and attempt to recompress the block and adjust the threshold until the desired bitrate was achieved. It is assumed that the former method would be preferred because it would result in less wasted computation.

6.6 Contrast adjustment

One of the stated problems with this algorithm is that the contrast of the image being compressed effects the relationship between error metric threshold and compressed image bitrate and decompressed image quality. This problem becomes more serious when various parts of the image have greatly different contrasts. In this situation, one error threshold is not a good choice over the entire image.

The solution is either to adjust the error metric threshold according to image contrast or to adjust the image contrast itself. On a global level, a histogram of an image can be generated. From this histogram, statistics, the the mean and the variance, which characterize the distribution of intensity values in the image can be calculated. Given this information the intensity values can be shifted and scaled so that their mean and variance is some "standard" value. On a local level the image can be separated into blocks. Each block can then be processed separately and have its contrast adjusted separately. The problem with this approach is that blocking artifacts may arise. One possible way to prevent this is to continuously vary the contrast adjustment parameters between adjacent blocks so that is no large jump between blocks. One problem with a local block

contrast adjustment is that if an entire block is made up of a relatively flat region then any noise in that region will be greatly amplified. This can partially be avoided by making sure that the contrast adjustment is continuous between adjacent blocks, but can also be prevented by putting a bound on the contrast adjustment. This bound may be derivable from model of the contrast sensitivity thresholds of the human visual system so that invisible features in the image are not amplified and unnecessarily coded.

7. References

Adelson, E. H.; Anderson, C. H.; Bergen, J. R.; Burt, P. J.; Odgen, J. M. "Pyramid methods in image processing," RCA Engineer, vol. 29, no. 6, Nov./Dec. 1984, pp. 33-41.

Andrews, C. A.; Davies, J. M.; Schwarz, G. R. "Adaptive Data Compression," Proceedings of the IEEE, vol. 55, no. 3, March 1967, pp. 267-277.

Balakrishnan, A. V. "An Adaptive Nonlinear Data Predictor," Proceedings 1962 National Telemetering Conference, vol. 2, pap. 6-5, 1962, pp. 1-15.

Besl, Paul J.; Jain, Ramesh C. "Segmentation Through Variable-Order Surface Fitting," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 2, March 1988, pp. 167-192.

Bially, Theodore. "Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction," IEEE Transactions on Information Theory, vol. IT-15, no. 6, November 1969, pp. 658-664.

Chan, Kelby K. "Implementation of Fast Cosine Transforms with Digital Signal Processors for Image Compression," SPIE Medical Imaging II, vol. 914, pt. 8, 1988, pp. 782-785.

Chen, Wen-Hsiung; Smith, Harrison C.; Fralick S. C. "A Fast Computational Algorithm for the Discrete Cosine Transform," IEEE Transactions on Communications, vol. COM-25, no. 9, September 1977, pp. 1004-1009.

Davisson, Lee D. "Data Compression Using Straight Line Interpolation," IEEE Transactions on Information Theory, vol. IT-14, no. 3, May 1968, pp. 390-394.

Davisson, Lee D. "Theory of Adaptive Data Compression" in the book Advances in Communications Systems, New York: Academic Press, 1966, pp. 173-192.

Dierckx, Paul; Suentens, Paul; Vandermeulen, Dirk. "An Algorithm for Surface Reconstruction from Planar Contours Using Smoothing Splines," Journal of Computational and Applied Mathematics, vol. 23, no. 3, Sept 1988, pp. 367-388.

Ehrman, L. "Analysis of Some Redundancy Removal Bandwidth Compression Techniques," Proceedings of the IEEE, vol. 55, no. 3, March 1967, pp. 278-287.

Feig, Ephraim. " A fast scaled-DCT algorithm," paper presented at the SPIE/SPSE Symposium on Electronic Imaging Science and Technology, February 1990.

Fowell, Richard A.; McNeil, David D. "Faster Plots by Fan Data-Compression," IEEE Computer Graphics and Applications, vol. 9, no. 2, March 1989, pp. 58-66.

Fukinuki, Takahiko. "Data Reduction of Picture Signals - Review on the Studies in Japan," Future Generation Computer Systems, vol. 1, no. 5, Sept 1985, pp. 279-308.

Gardenhire, Lawrence W., "Redundancy Reduction the Key to Adaptive Telemetry," Proc. of the 1964 National Telemetering Conference, 1964, pp. 1-16.

Gharavi, H.; Tabatabai, A. "Application of Quadrature Mirror Filtering to the Coding of Monochrome and Color Images," Proceedings ICASSP 87, vol. 4, pp. 2384-2387.

Goel, B. D.; Kwatra, S. C. "A Data Compression Algorithm for Color Images Based on Run-Length Coding and Fractal Geometry," IEEE International Conference on Communications, June 1988, pp. 1253-1256.

Gottesman, Jon; Rubin Gary S.; Legge, Gordon E. "A Power Law for Perceived Contrast in Human Vision," Vision Research, vol. 21, 1981, pp. 791-799.

Hochman, D.; Katzman, H.; Weber, D. R. "Application of Redundancy Reduction to Television Bandwidth Compression," Proceedings of the IEEE, vol. 55, no. 3, March 1967, pp. 263-266.

Holtzman, Henry. "Increasing resolution using Pyramid coding and pattern matching," unpublished technical memorandum. Massachusetts Institute of Technology Media Laboratory: Cambridge Massachusetts, May 1990.

Hovig, Ingvil. "Image Compression and Coding, with Emphasize on Polygon Based Techniques and Representation," ICAS '88, pp. 467-470.

Hudson, Graham P.; Yasuda, Hiroshi; Sebestyen, Istvan. "The International Standardisation of a Still Picture Compression Technique," Globecom'88, pp. 1016-1021.

Imai, Hrioshi; Iri, Masao. "Computational-Geometric Methods for Polygonal Approximations of a Curve," Computer Vision, Graphics, and Image Processing, vol. 36, 1981, pp. 31-41.

Kortman, C. M. "Data Compression by Redundancy Reduction," IEEE Spectrum, March 1967, pp. 133-139.

Kortman, C. M. "Redundancy Reduction - A Practical Method of Data Compression," Proceedings of the IEEE, vol. 55, no. 3, March 1967.

Kunt, Murat. "Redundancy Reduction in Digital Images," International Journal of Optoelectronics, vol. 3, no. 1, 1988, pp. 3-44.

Kutz, R. L. Sciuli, J. A. "An Adaptive Image Compression System and its Performance in a Noisy Channel," IEEE Transactions on Information Theory, vol. IT-14, no. 6, Nov 1968, pp. 838-839.

Laeger, Alain; Mitchell, Joan L.; Yamazaki, Yasuhiro. "Still Picture Compression Algorithms Evaluated for International Standardisation," Golbecom'88, pp. 1028-1032.

Lancaster, Peter; Salkauskas, Kestutis. Curve and Surface Fitting - an Introduction, Academic Press, San Diego: 1986.

Lee, Chin-hwa. "Image Surface Approximation with Irregular Samples," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 2, Feb 1989, pp. 206-212.

Makhoul, John. "A Fast Cosine Transform in One and Two Dimensions," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSIP-28, no. 1, Feb 1980, pp. 27-34.

Mazumder, Pinaki. "Planar Decomposition for Quadtree Data Structure," Computer Vision, Graphics, and Image Processing, vol. 38, no. 3, 1987, pp. 258-274.

McCaughey, Dennis G. "An Image Coding Algorithm Using Spline Functions," SPIE Applications of Digital Image Processing, vol. 149, 1978, pp. 51-61.

Naik, Sanjeev M.; Jain, Ramesh C. "Spline-Based Surface Fitting on Range Images for CAD Applications," 88 Computer Vision and Pattern Recognition Conference, June 1988, pp. 249-253.

Narasimha, Madihally J.; Peterson, Allen M. "On the Computation of the Discrete Cosine Transform," IEEE Transactions on Communications, vol. COM-26, no. 6, June 1978, pp. 934-936.

Pratt, William K. Digital Image Processing, John Wiley and Sons, New York: 1978.

Sanz, Alberto; Munoz, Carlos; Garcia, Narciso. "On the Use of Splines in Hierarchical Image Transmission," Proc. IEEE International Conference on Acoustics Speech and Signal Processing, vol. 1, 1982, pp. 456-459.

Shannon, Claude E.; Weaver, Warren. The Mathematical Theory of Communication, University of Illinois Press, Urbana: 1963.

Schmitt, Francis J. M.; Barsky, Brian A.; Du, Wen-Hui. "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data," Proc. Siggraph, vol. 20, no. 4, 1986, pp. 179-188.

Schreiber, W. F. "Picture Coding," Proceedings of the IEEE, vol. 55, no. 3, March 1967, pp. 320-330.

Siegel, Sidney; Castellan Jr., N. John. Nonparametric Statistics, McGraw-Hill Inc., New York: 1988, 1956.

Simoncelli, Eero Peter. "Orthogonal Sub-band Image Transforms," Master's thesis, Massachusetts Institute of Technology: Cambridge Massachusetts, June 1988.

Sklansky, Jack; Gonzalez, Victor. "Fast Polygonal Approximation of Digitized Curves," Pattern Recognition, vol. 12, 1980, pp. 327-331.

Stark, Henry; Webb Heywood. "Bounds on Errors in Reconstructing from Undersampled Images," Journal of Optical Society of America, vol. 69, no. 7, July 1979, pp. 1042-1043.

Tokuta, Alade O. "A Fast Method for Surface Fitting from Sampled Data," Proc. IEEE Southeastcon, vol. 2, 1987, pp. 598-600.

Wallach, E.; Kamin E. "A Fractal Based Approach to Image Compression " Proc. ICASSP, vol. 1, 1986, pp. 529-532.

Wall, Karin; Danielsson, Per-Erik. "A Fast Sequential Method for Polygonal Approximation of Digitized Curves," Computer Vision, Graphics and Image Processing, vol. 28, 1984, pp. 220-227.

Wall, Karin; Danielsson, Per-Erik. "A New Method for Polygonal Approximation of Digitized Curves," Proc. of the 3rd Scandinavian Conference on Image Analysis, Jul 12-14 1983, pp. 60-66.

Wallace, Gregory; Vivian, Roy; Poulsen, Henning. "Subjective Testing Results for Still Picture Compression Algorithms for International Standardization," Golbecom'88, pp.1022-1027.

Woods, John W.; O'Neil, Sean D. "Subband Coding of Images," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-34, no. 5, October 1986, pp. 1278-1288.

Yang, Kun-Min; Wu, Lance; Mills, Machael. "Fractal Based Image Coding Scheme Using the Peano Scan," ISCAS'88, 1988, pp. 2301-2304.

8. Appendix

8.1 Photographs

This section contains all of the photographs referenced in this thesis. Each photograph in this section is labeled with a short description of its contents.

All of the photographs were taken on a matrix film recorder. The black and white images were generated by a Datacube 8 bit per pixel frame buffer driven by a Sun 3 and were photographed with Kodak ASA 100 Tri-X film and printed on fiber-based paper. Color images were generated by a Taac 24 bit per pixel frame buffer driven by a Sun 3 and were photographed with Kodak ASA 100 Kodacolor film and printed on resin-coated paper.

The six color Photographs [8.1-29], [8.1-36], [8.1-37], [8.1-38], [8.1-39], and [8.1-40] are reprinted in black and white in photographs [8.1-1], [8.1-60], [8.1-61], [8.1-62], [8.1-63], and [8.1-64], respectively.

Certain photographs are magnified versions of the image. This is not to be confused with the scaling algorithms which involve the NSI algorithm. When an image is said to be magnified it means that the size of image was increased through the use of pixel replication to make detail in the image more visible. Therefore if an image is magnified four times then each pixel of the original image was displayed as blocks four pixels on a side when it was displayed to be photographed.

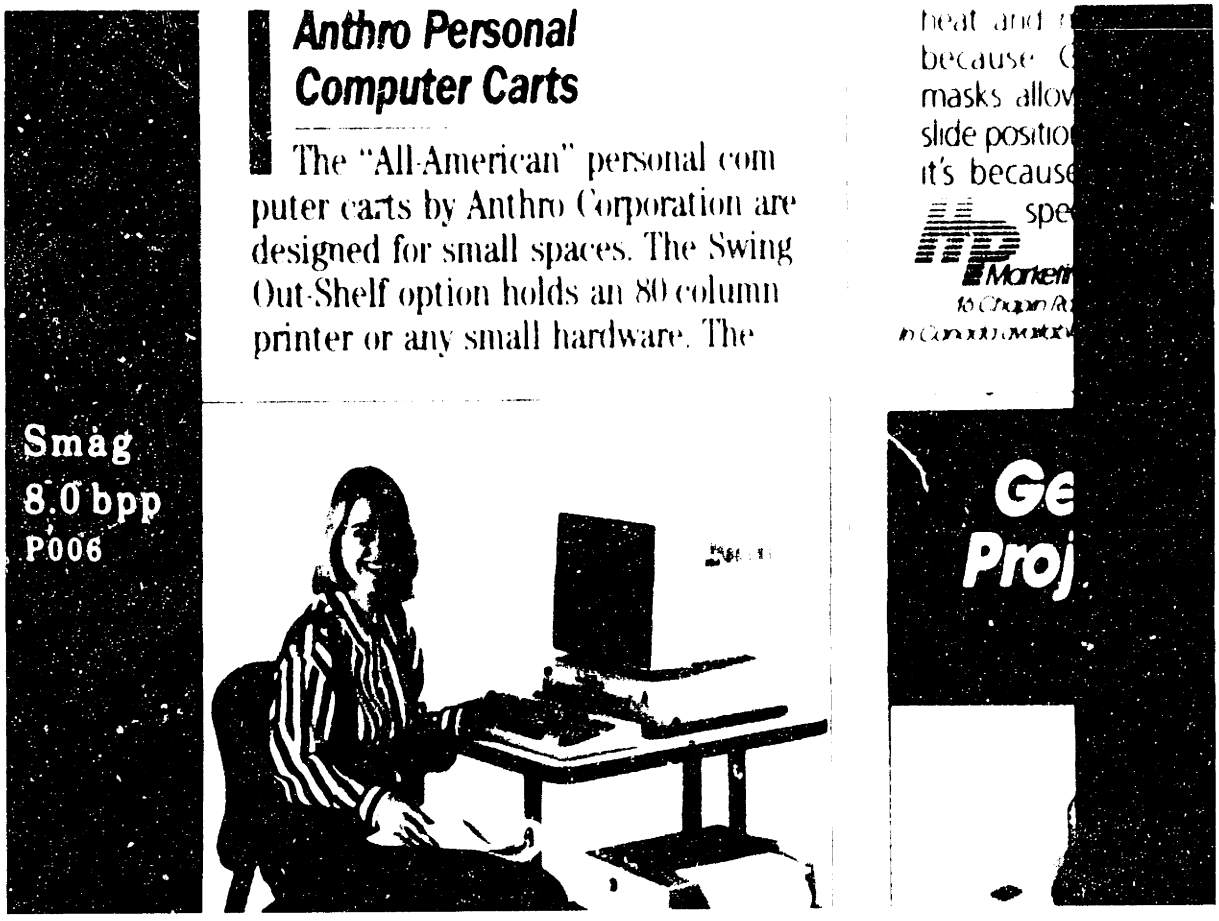
List of Photographs

Photograph [8.1-1]: Original "Lena" image at 8.0 bits per pixel.

ID #[P004]

Photograph [8.1-2]: Original "Tiff" image at 8.0 bits per pixel.

ID #[P005]



Photograph [8.1-3]: Original "Smag" image at 8.0 bits per pixel.

ID #[P006]

Photograph [8.1-4]: Lena image compressed to 1.0 bit per pixel by a discrete cosine transform.

ID #[P014]

Photograph [8.1-5]. Lena image compressed to 1.0 bit per pixel by subsampling the image by eight in the horizontal direction.

ID #[P015]

Photograph [8.1-6]: Lena image compressed to 1.0 bit per pixel by subsampling the image by four in the horizontal direction and two in the vertical direction.

ID #[P016]

Photograph [8.1-7]: Lena image compressed to 1.0 bit per pixel by one dimensional sample point selection and a scaled cone intersection error metric.

ID #[P061]

Photograph [8.1-8]: Lena image compressed to 1.0 bit per pixel by one dimensional sample point selection and a maximum deviation error metric.

ID #[P017]

Photograph [8.1-9]: Lena image compressed to 1.0 bit per pixel by one dimensional sample point selection and a sum of error error metric.

ID #[P018]

Photograph [8.1-10]: Lena image compressed to 1.0 bit per pixel by one dimensional sample point selection and a sum of squared error metric.

ID #[P019]

Photograph [8.1-11]: Lena image compressed to 1.0 bit per pixel by one dimensional sample point selection and a sum of squared error metric with sample point jittering.

ID #[P020]

Photograph [8.1-12]: Lena image compressed to 1.0 bit per pixel by quadtree image decomposition and a sum of squared error error metric.

ID #[P021]

Photograph [8.1-13]: Lena image compressed to 1.0 bit per pixel by the final version of NSI with two dimensional sample point selection, a sum of squared error error metric, sample point jittering, and sample point look ahead.

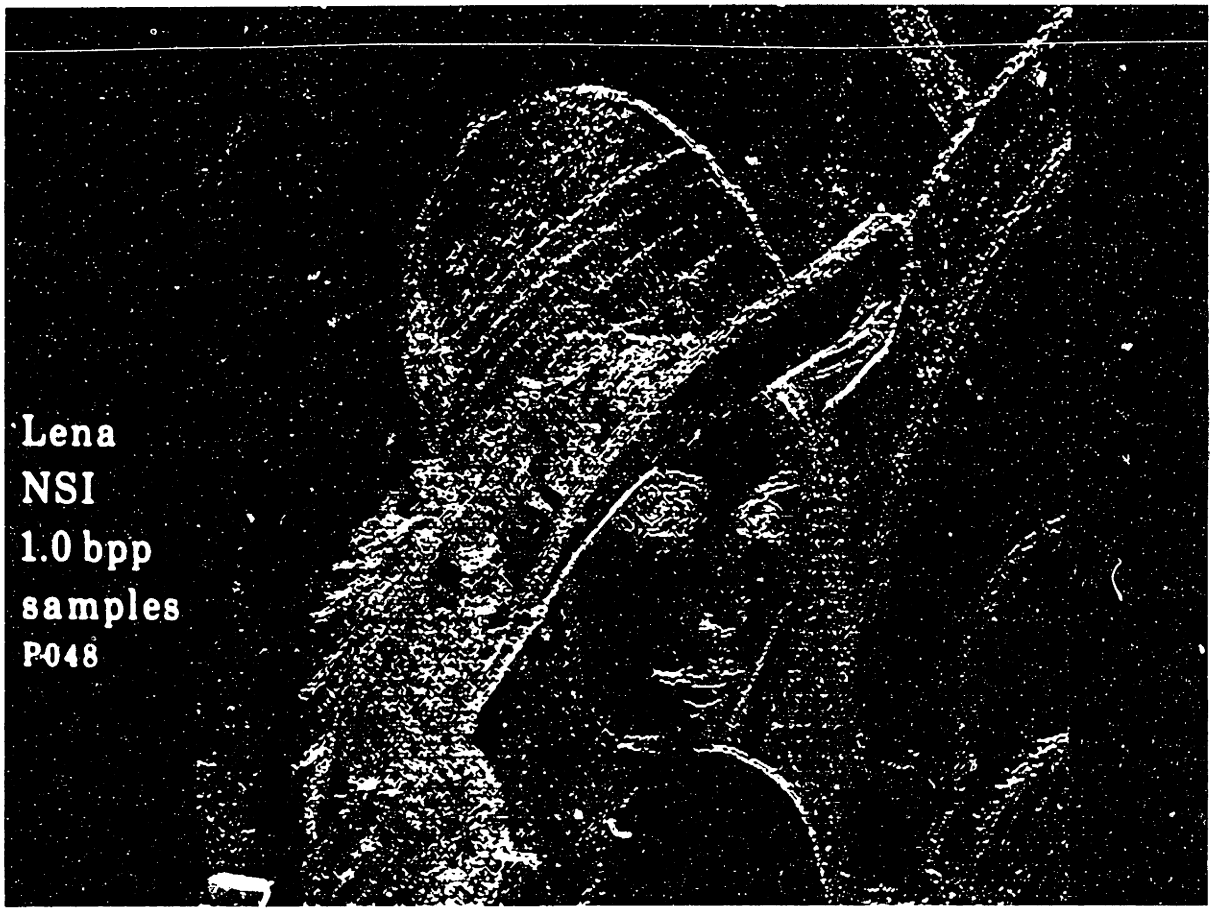
ID #[P025]

Photograph [8.1-14]: Lena image compressed to 0.5 bit per pixel by the final version of NSI with two dimensional sample point selection, a sum of squared error error metric, sample point jittering, and sample point look ahead.

ID #[P049]

Photograph [8.1-15]: Lena image compressed to 2.0 bits per pixel by the final version of NSI with two dimensional sample point selection, a sum of squared error error metric, sample point jittering, and sample point look ahead.

ID #[P050]

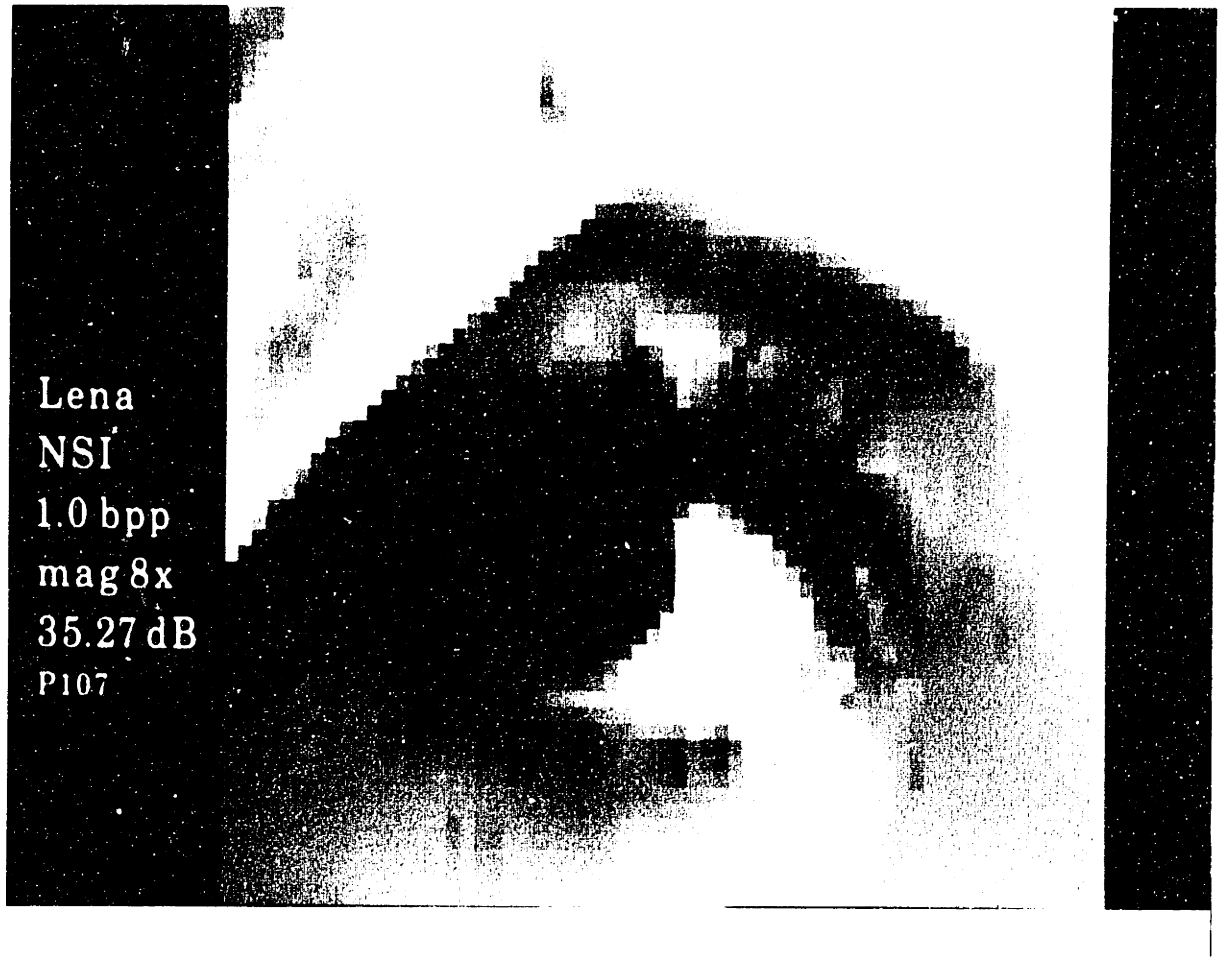


Photograph [8.1-16]: Samples (represented as white dots) chosen by NSI when compressing the Lena image to 1.0 bits per pixel. These samples correspond to Photograph [8.1-13].

ID #[P048]

Photograph [8.1-17]: Eye region of the original 8.0 bit per pixel Lena image. Magnified eight times. The complete image is in Photograph [8.1-1].

ID #[P109]

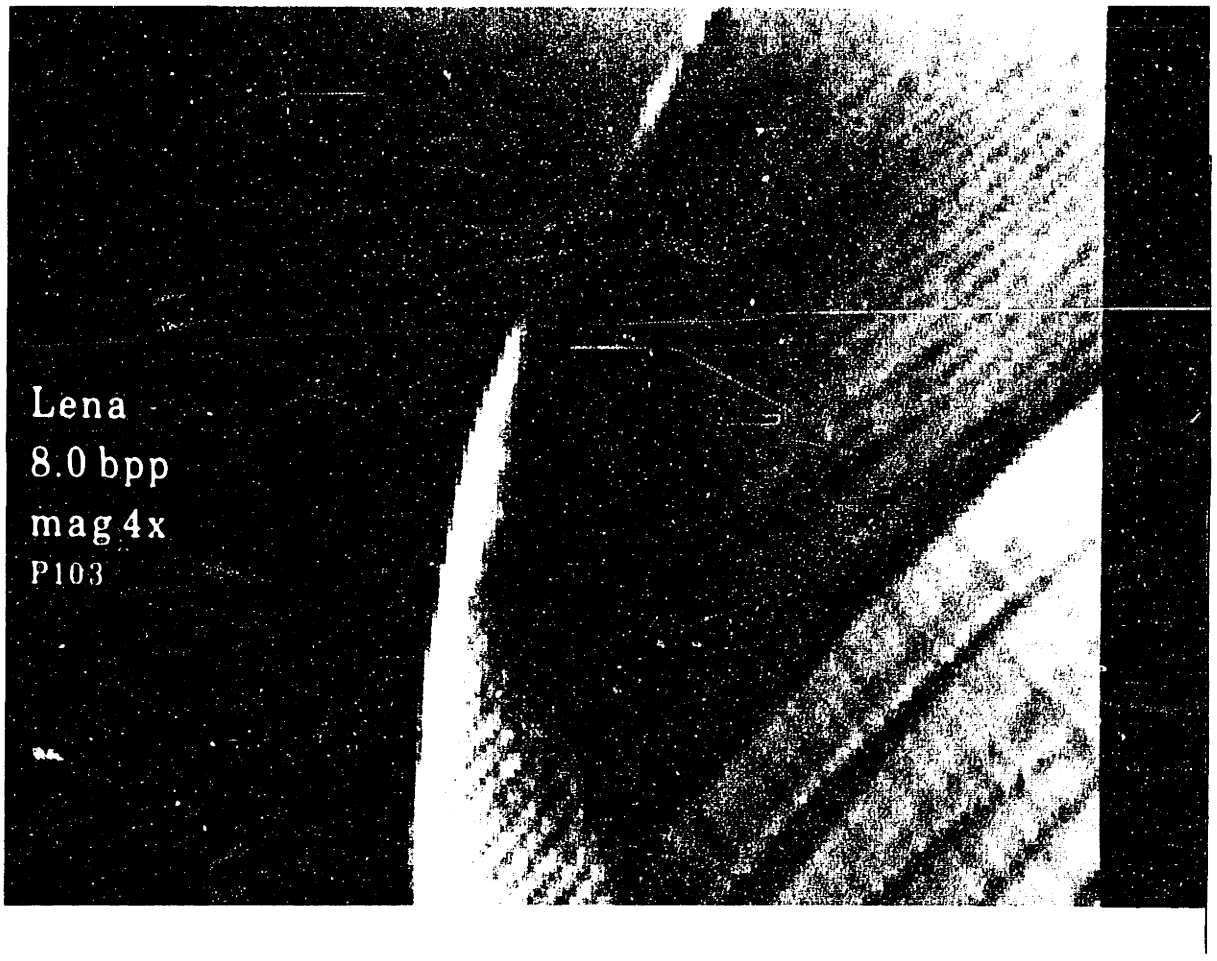


Photograph [8.1-18]: Eye region of the Lena image compressed to 1.0 bit per pixel by NSI. Magnified eight times. The complete image is in Photograph [8.1-13].

ID #[P107]

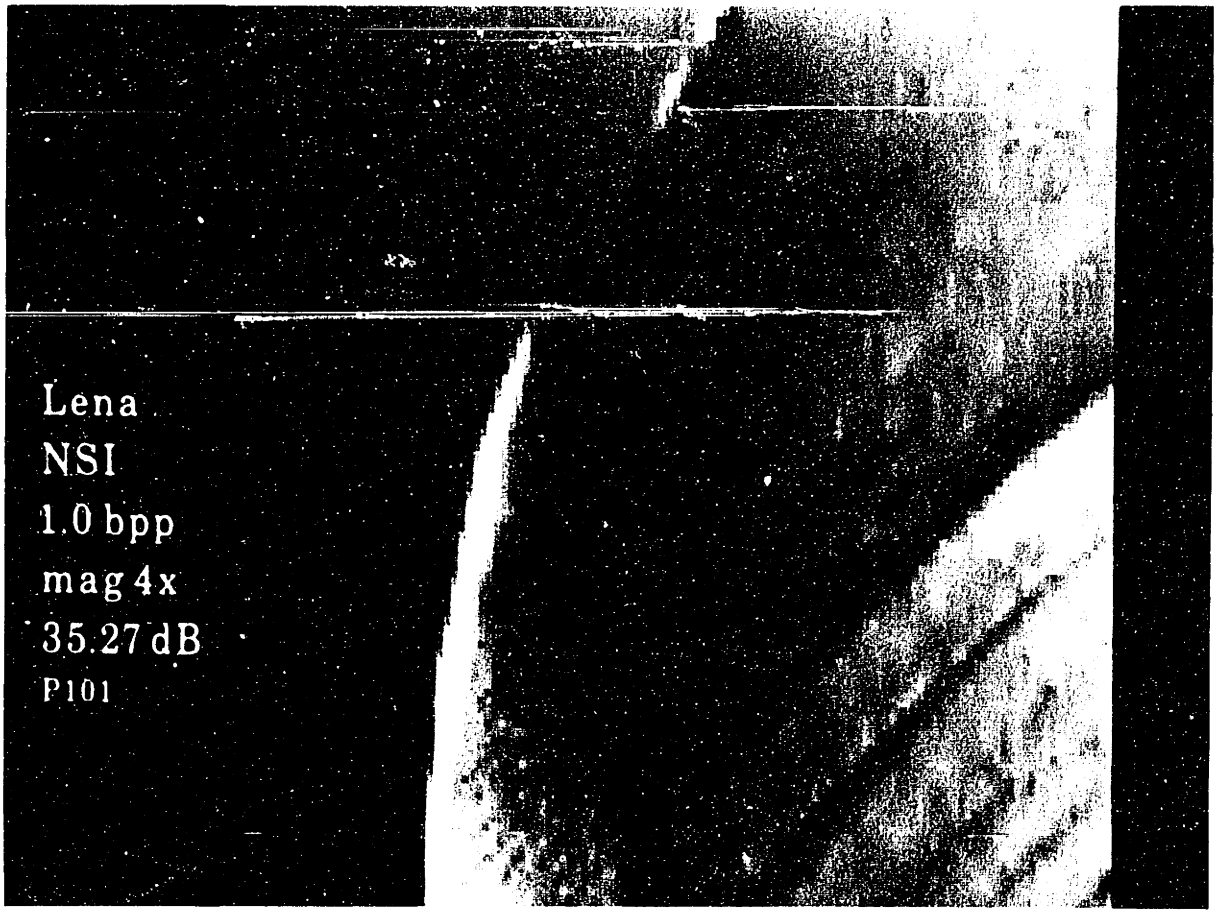
Photograph [8.1-19]: Eye region of the Lena image compressed to 1.0 bit per pixel by the DCT. Magnified eight times. The complete image is in Photograph [8.1-4].

ID #[P108]



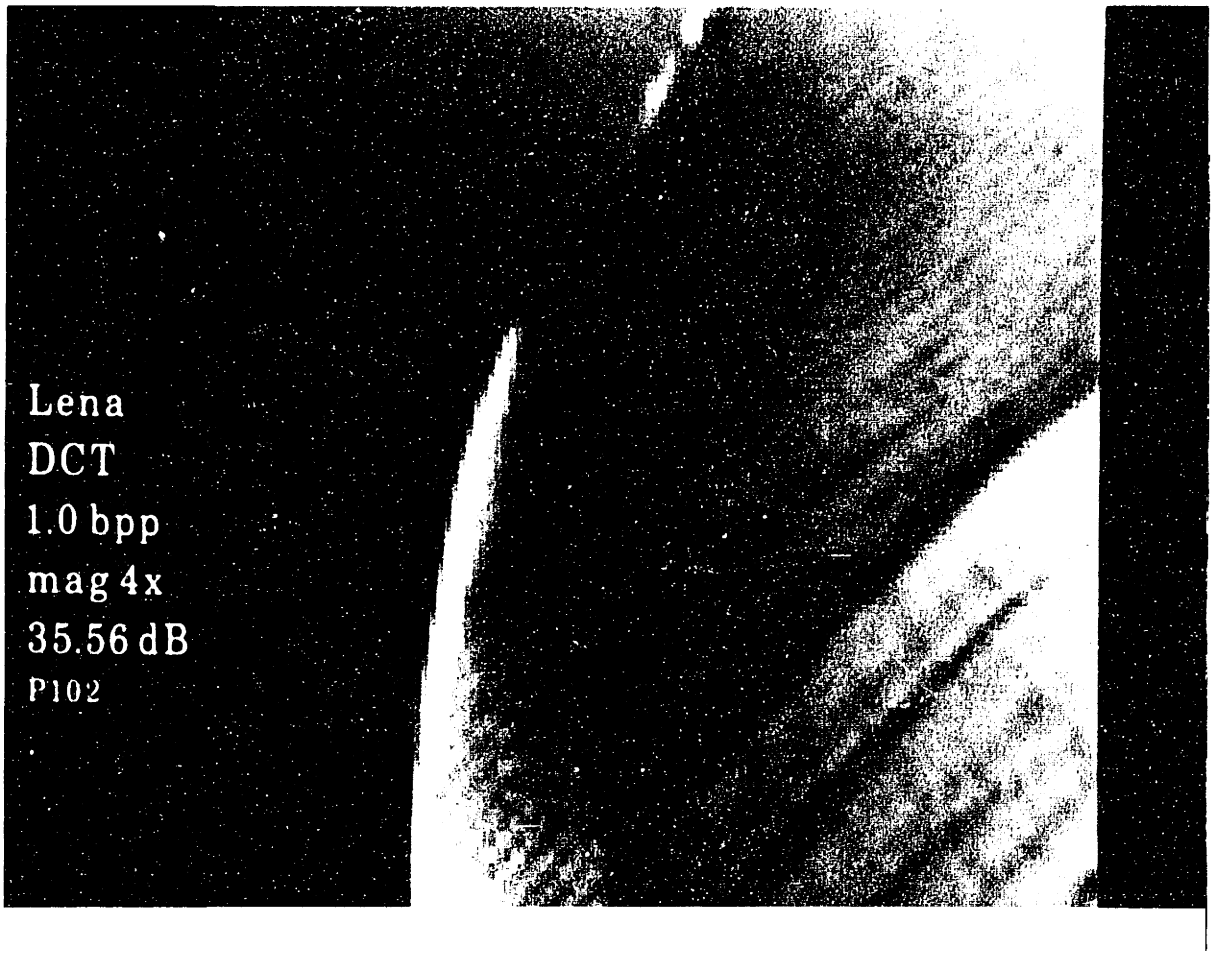
Photograph [8.1-20]: Textured hat region of the original 8.0 bit per pixel Lena image. Magnified four times. The complete image is in Photograph [8.1-1].

ID #[P103]



Photograph [8.1-21]: Textured hat region of the Lena image compressed to 1.0 bit per pixel by NSI. Magnified four times. The complete image is in Photograph [8.1-13].

ID #[P101]



Photograph [8.1-22]: Textured hat region of the Lena image compressed to 1.0 bit per pixel by the DCT. Magnified four times. The complete image is in Photograph [8.1-4].

ID #[P102]

Photograph [8.1-23]: The difference image between the original 8.0 bit per pixel Lena image and the 1.0 bit per pixel NSI version. The differences are scaled by eight and values outside of the 0-255 range were clipped. A gray dot represents no error, a white dot represents a positive error and a black dot represents a negative error. A plot of the magnitude of the fourier transform of this image is pictured in Photograph [8.1-28].

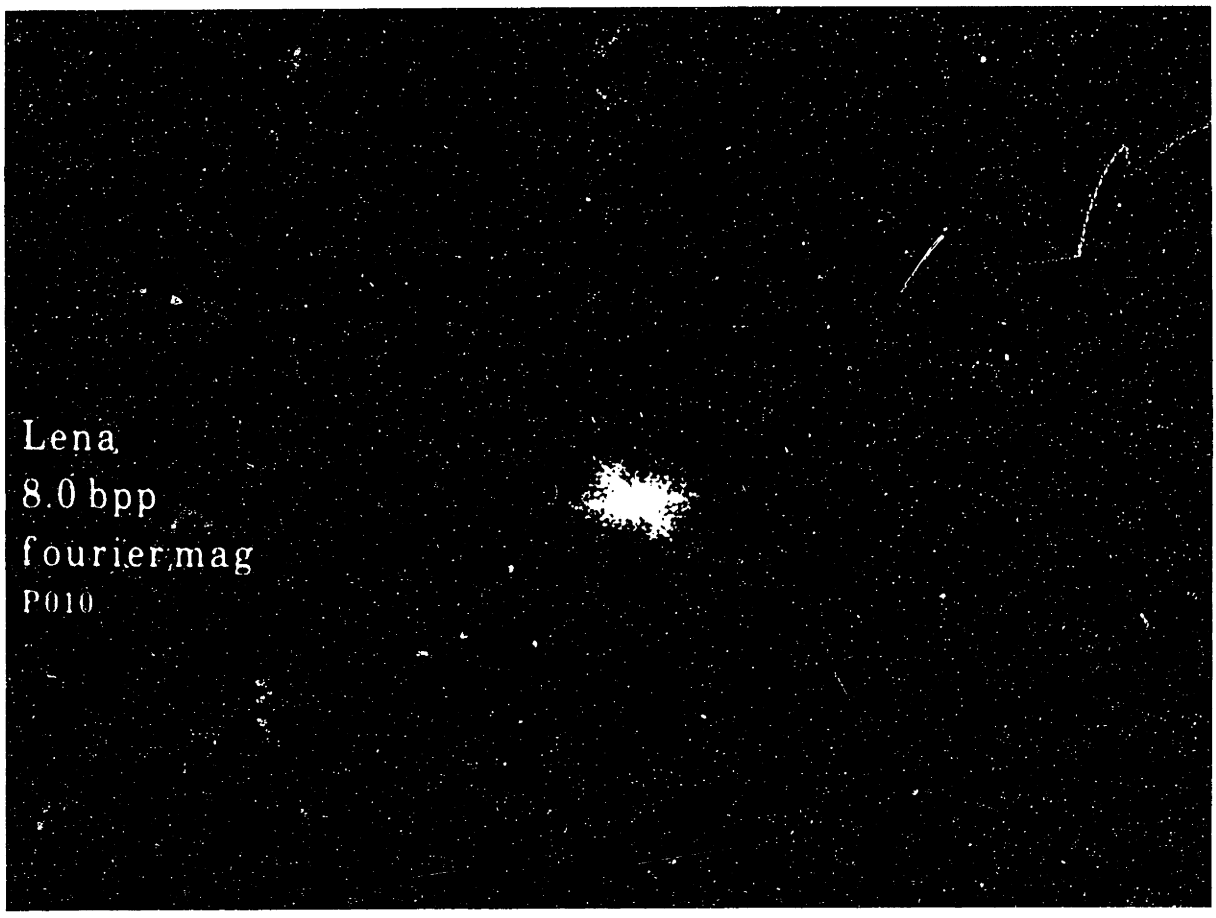
ID #[P100]

Photograph [8.1-24]: Radial arm region of the original 8.0 bit per pixel Lena image. Magnified two times. The complete image is in Photograph [8.1-1].

ID #[P115]

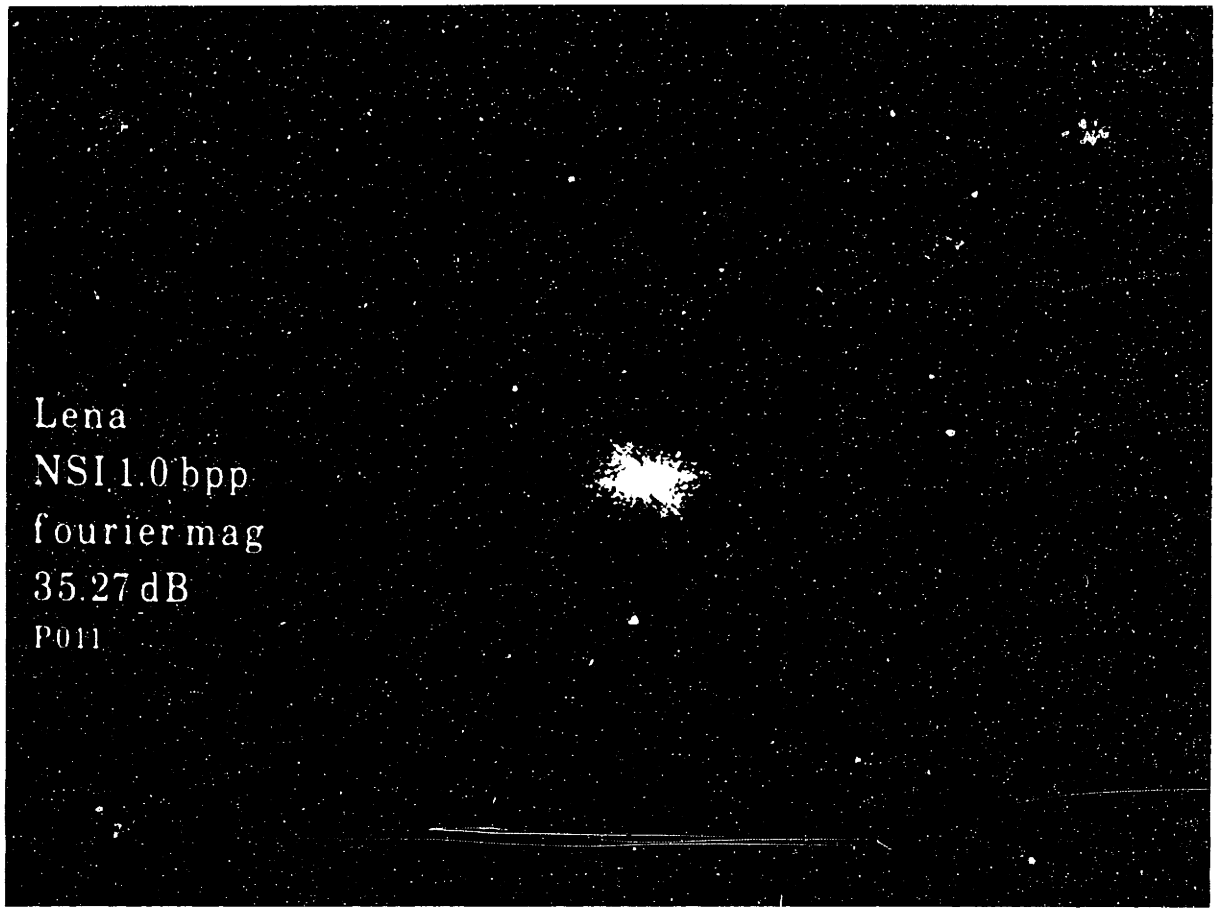
Photograph [8.1-25]: Radial arm region of the Lena image compressed to 1.0 bit per pixel by NSI. Magnified two times. The complete image is in Photograph [8.1-13].

ID #[P113]



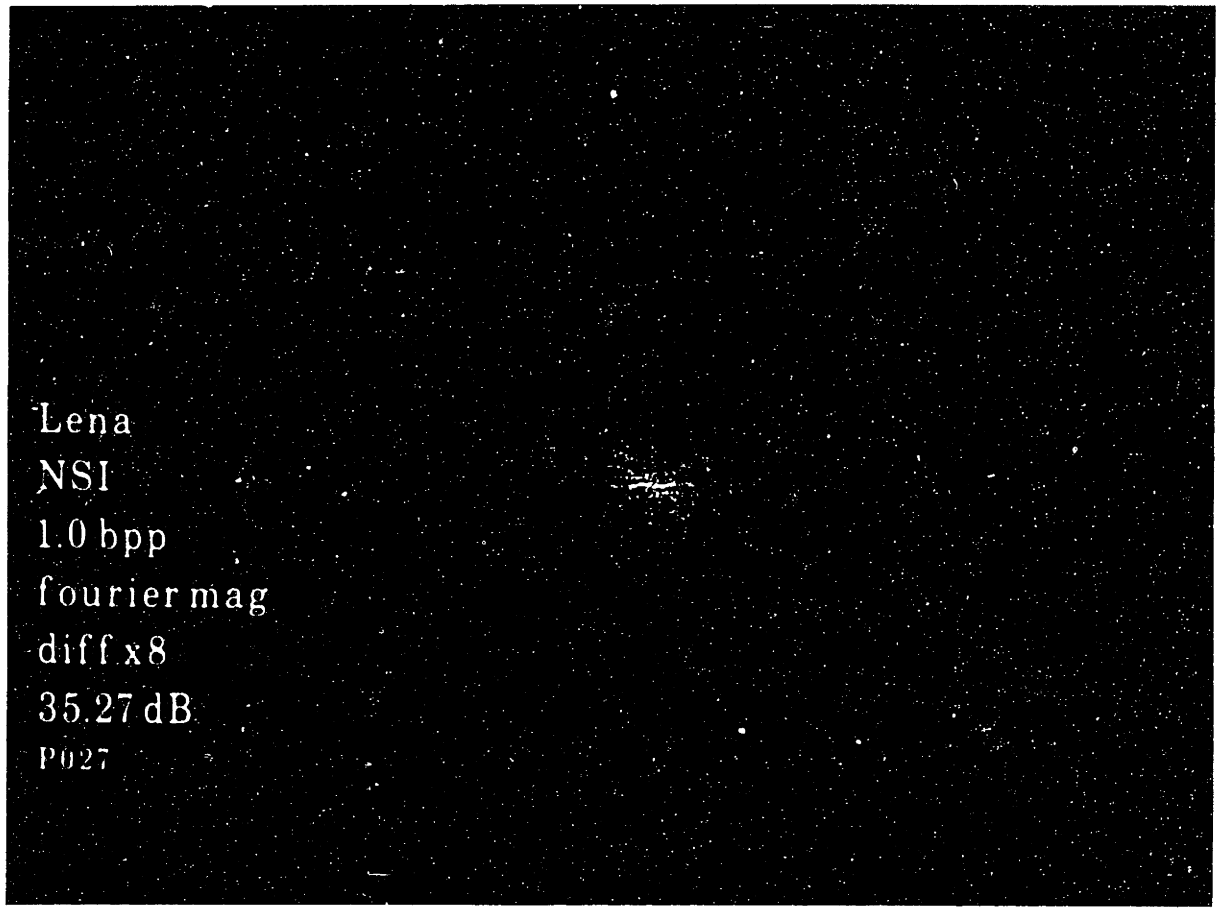
Photograph [8.1-26]: Plot of the magnitude of the fourier transform coefficients of the original 8.0 bit per pixel Lena image. Values outside of the 0-255 range were clipped.

ID #[P010]



Photograph [8.1-27]: Plot of the magnitude of the fourier transform coefficients of the Lena image compressed to 1.0 bit per pixel, shown in Photograph [8.1-13]. Values outside of the 0-255 range were clipped.

ID #[P011]



Photograph [8.1-28]: Plot of the magnitude of the fourier transform coefficients of the scaled difference image between the original 8.0 bit per pixel Lena image and the 1.0 bit per pixel NSI version. This image is pictured in Photograph [8.1-23]. Values outside of the 0-255 range were clipped.

ID #[P027]

Photograph [8.1-29]: Original 24.0 bit per pixel color Lena image. A black and white version of this image can be seen in Photograph [8.1-1].

ID #[P028]

Photograph [8.1-30]: The red channel of the original 24.0 bit per pixel color Lena image.

ID #[P030]

Photograph [8.1-31]: The green channel of the original 24.0 bit per pixel color Lena image.

ID #(P031)

Photograph [8.1-32]: The blue channel of the original 24.0 bit per pixel color Lena image.

ID #[P032]

Photograph [8.1-33]: The Y channel of the original 24.0 bit per pixel color Lena image.

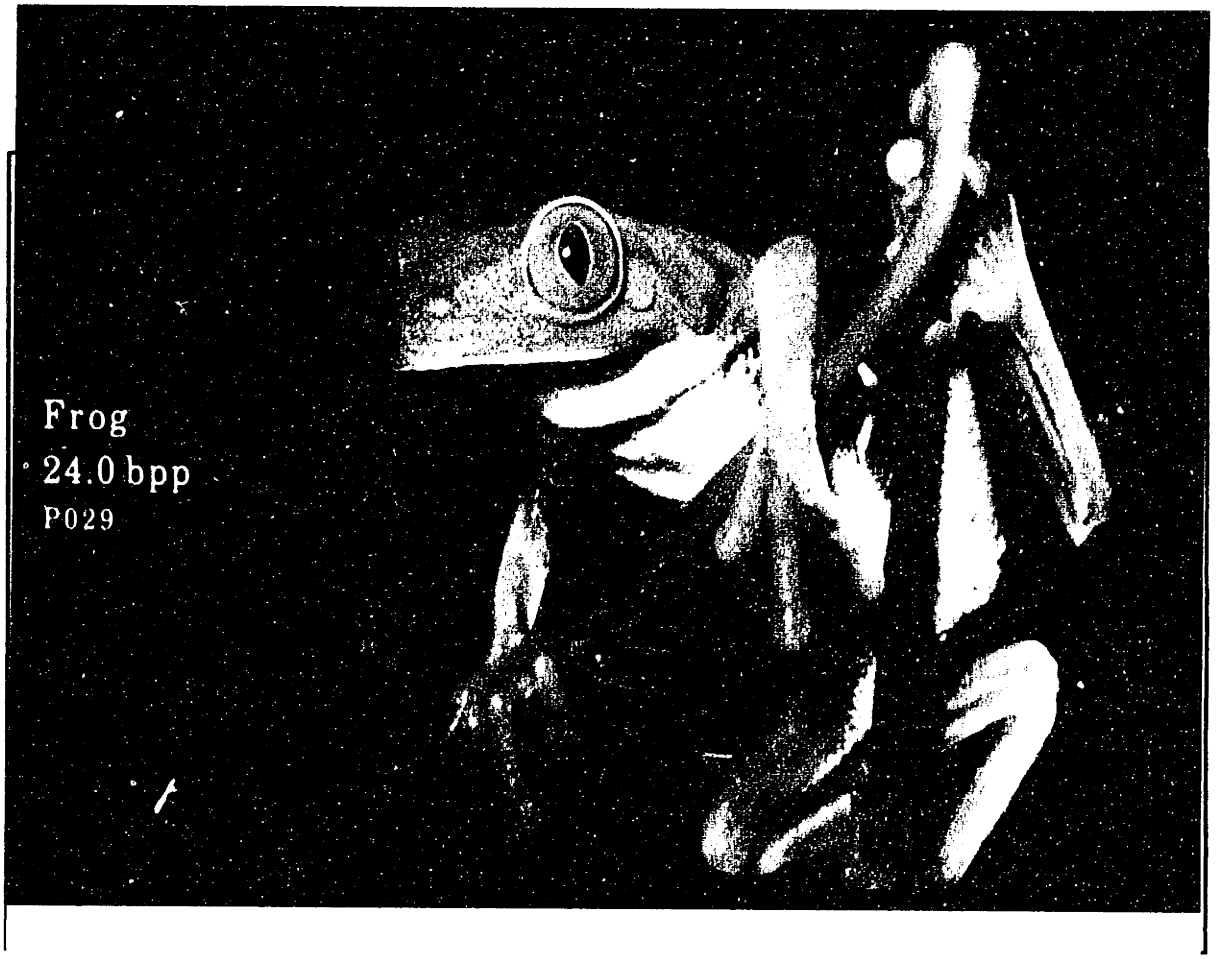
ID #[P033]

Photograph [8.1-34]: The I channel of the original 24.0 bit per pixel color Lena image.

ID #[P034]

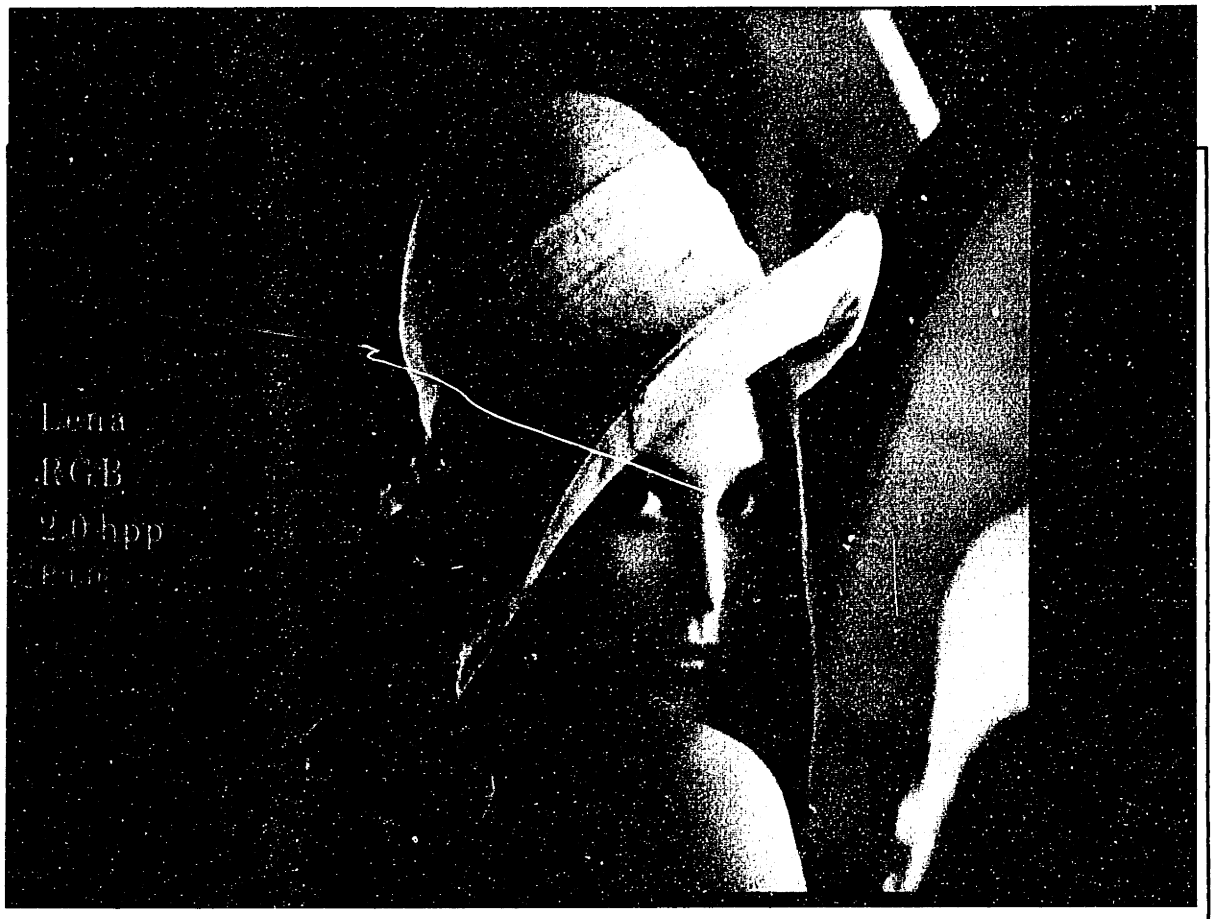
Photograph [8.1-35]: The Q channel of the original 24.0 bit per pixel color Lena image.

ID #[P035]



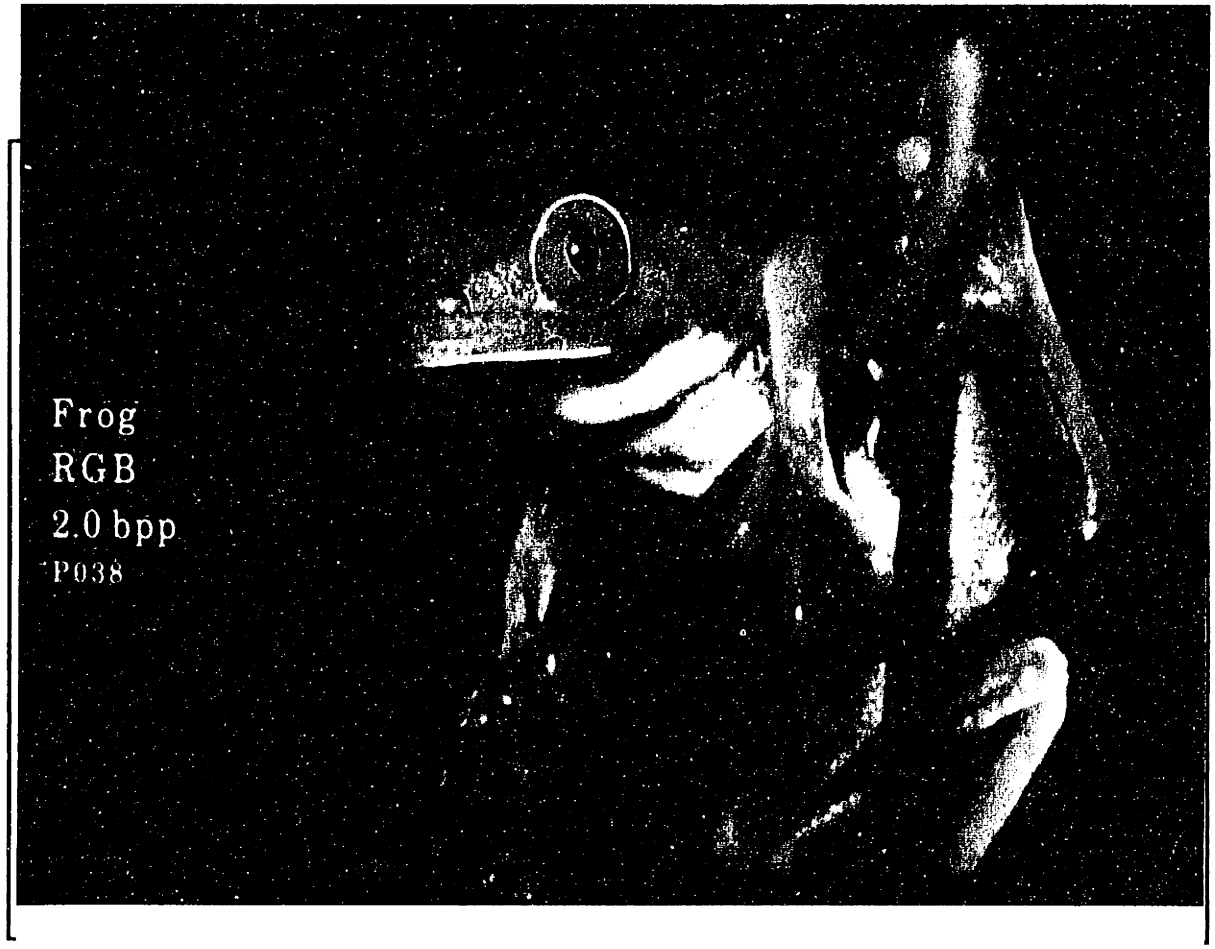
Photograph [8.1-36]: Original 24.0 bit per pixel color Frog image. A black and white version of this photograph can be seen in Photograph [8.1-60].

ID #[P029]



Photograph [8.1-37]: Color Lena image compressed by NSI to 2.0 bits per pixel by compressing the R, G, and B channels individually. A black and white version of this image can be seen in Photograph [8.1-61].

ID #[P036]



Photograph [8.1-38]: Color Frog image compressed by NSI to 2.0 bits per pixel by compressing the R, G, and B channels individually. A black and white version of this image can be seen in Photograph [8.1-62].

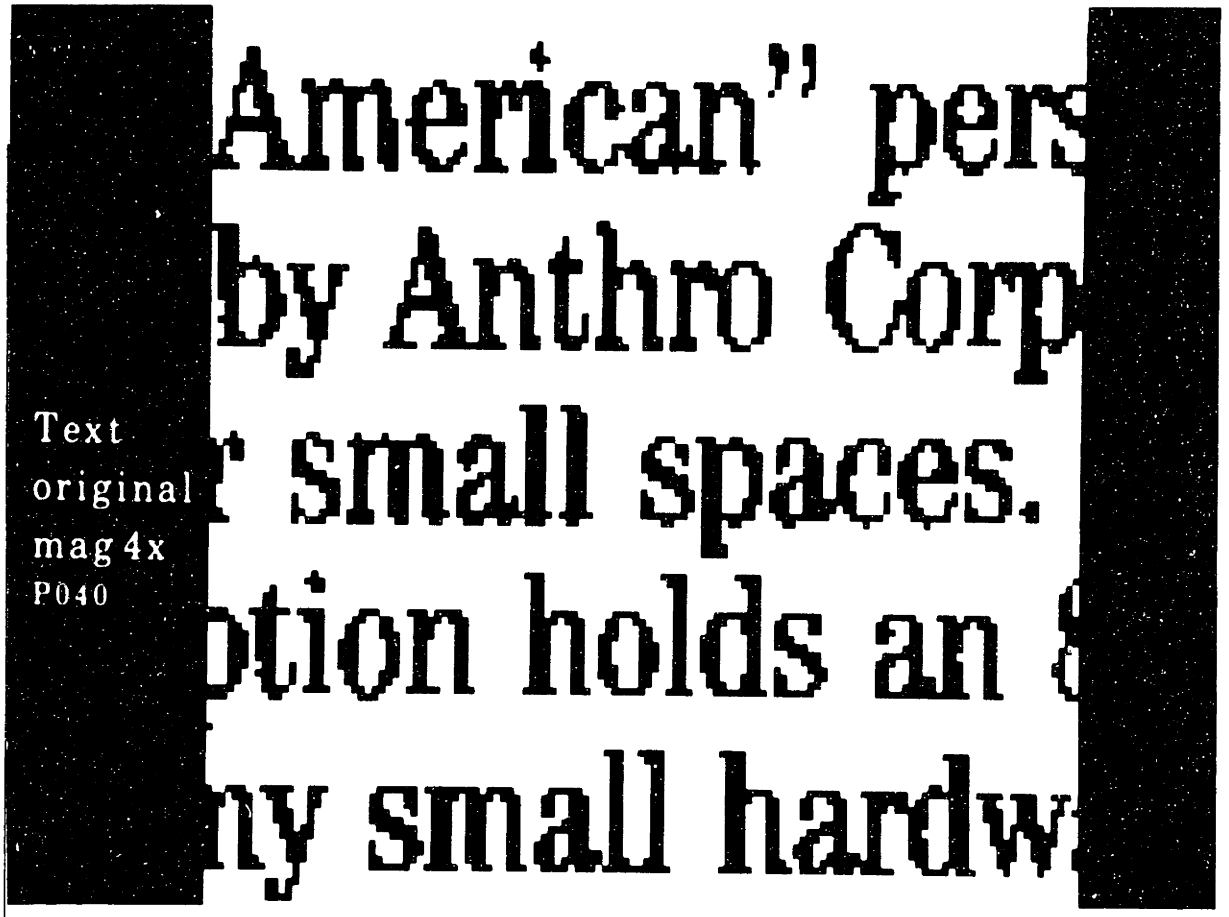
ID #[P038]

Photograph [8.1-39]: Color Lena image compressed by NSI to 2.0 bits per pixel by compressing the Y, I, and Q channels individually. A black and white version of this image can be seen in Photograph [8.1-63].

ID #[P037]

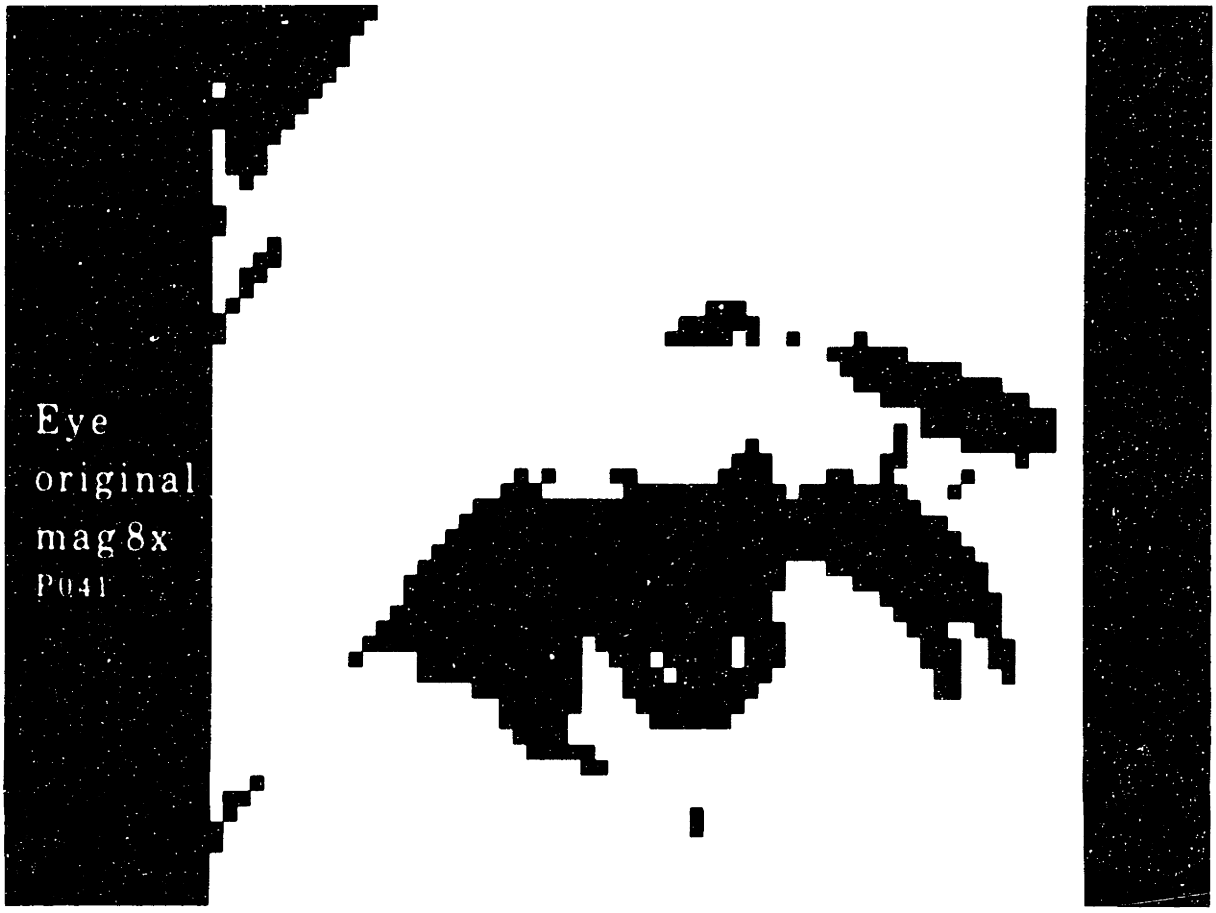
Photograph [8.1-40]: Color Frog image compressed by NSI to 1.9 bits per pixel by compressing the Y, I, and Q channels individually. A black and white version of this image can be seen in Photograph [8.1-64].

ID #[P039]



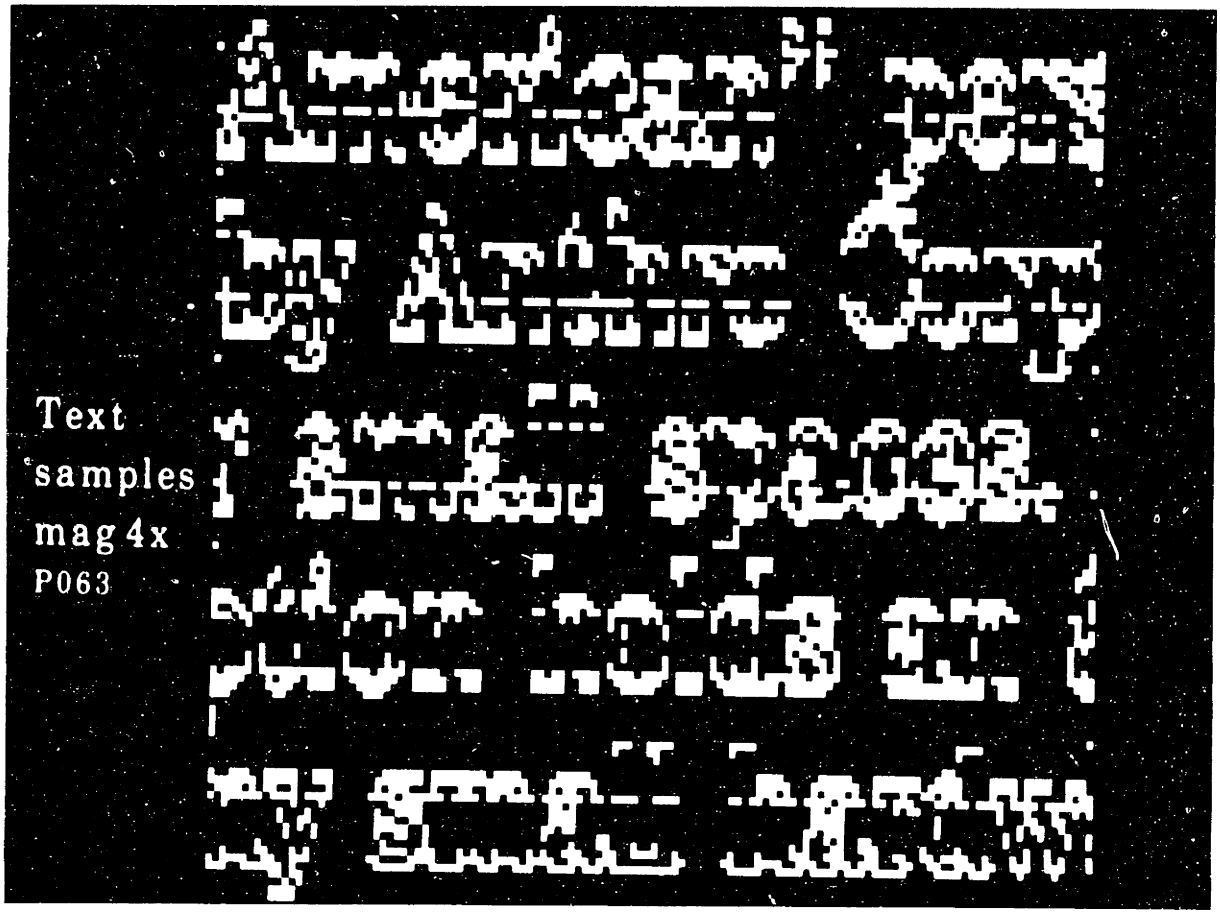
Photograph [8.1-41]: Original 8.0 bits per pixel Text image. Magnified four times to show detail.

ID #[P040]



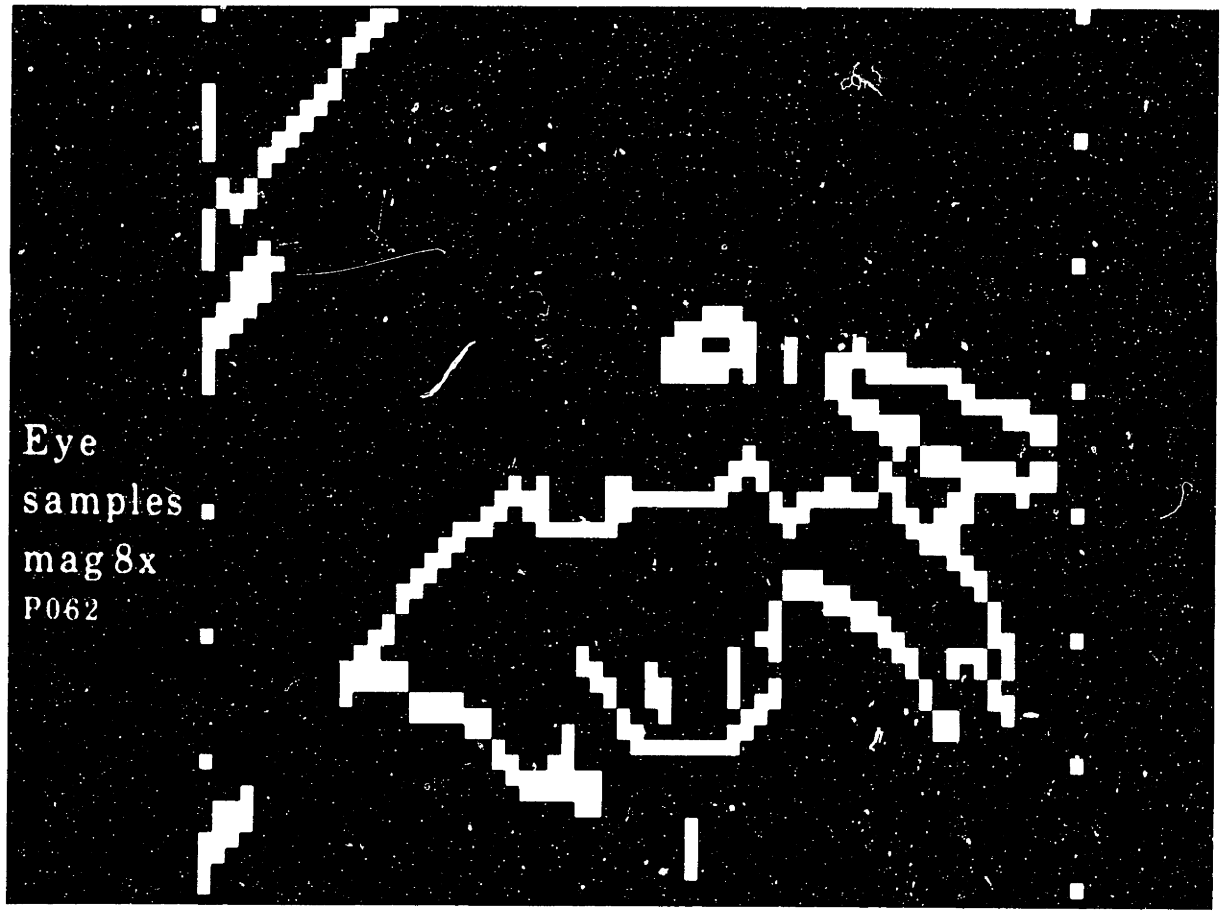
Photograph [8.1-42]: Original 8.0 bits per pixel Eye image. Magnified eight times to show detail.

ID #[P041]



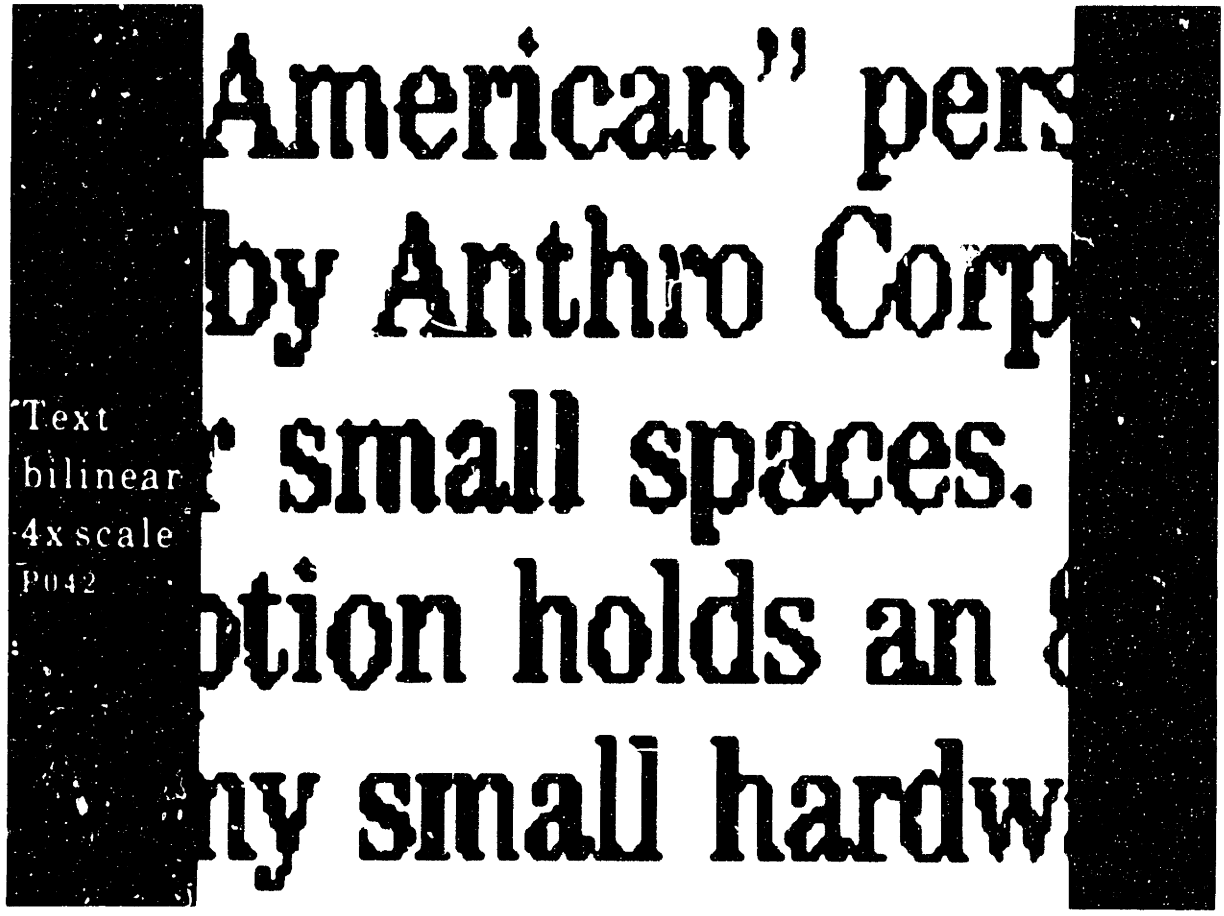
Photograph [8.1-43]: Samples (represented as white dots) chosen by NSI when compressing the Text image of Photograph [8.1-41]. Magnified four times.

ID #[P063]



Photograph [8.1-44]: Samples (represented as white dots) chosen by NSI when compressing the Eye image of Photograph [8.1-42]. Magnified eight times.

ID #[P062]

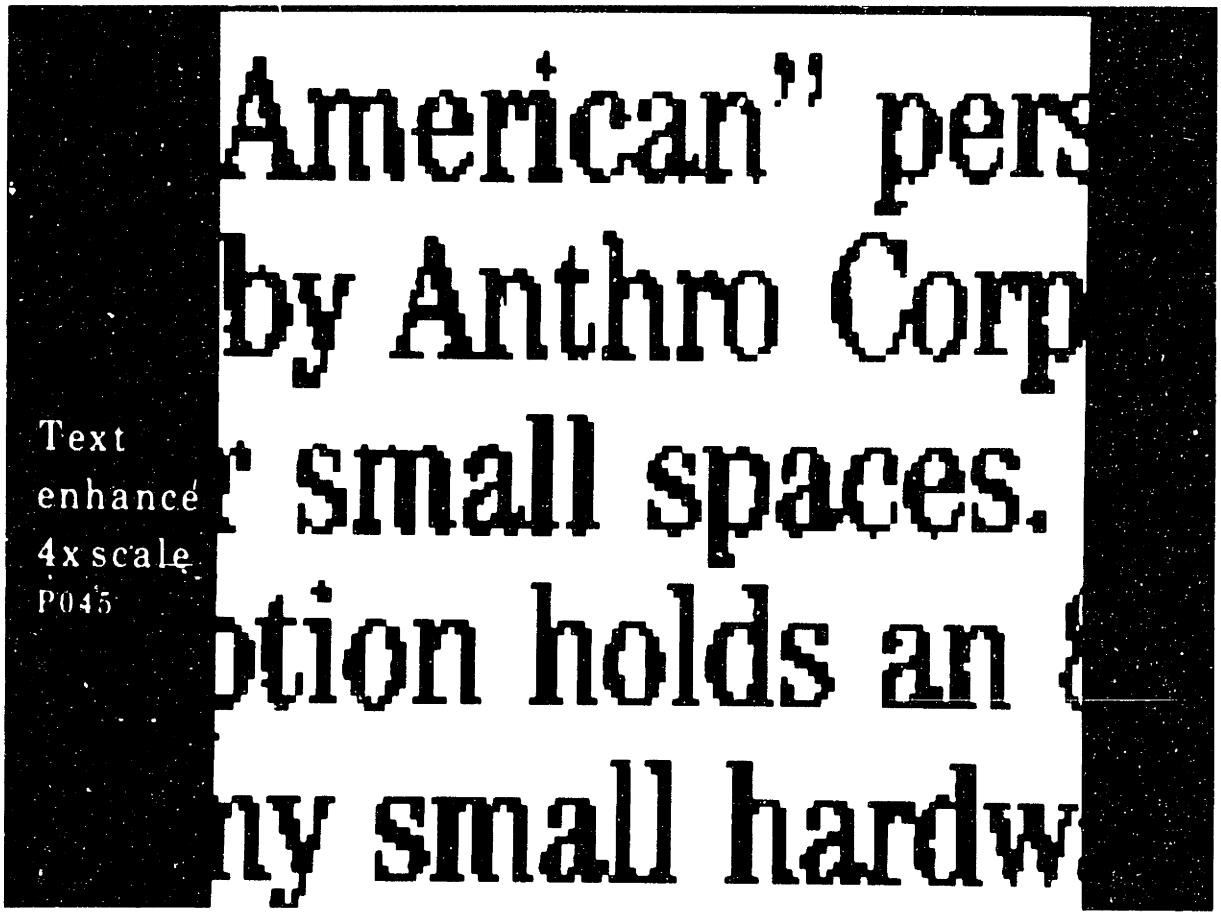


Photograph [8.1-45]: Text image scaled by four using the NSI bilinear scaling algorithm.

ID #[P042]

Photograph [8.1-46]: Eye image scaled by eight using the NSI bilinear scaling algorithm

ID #[P043]

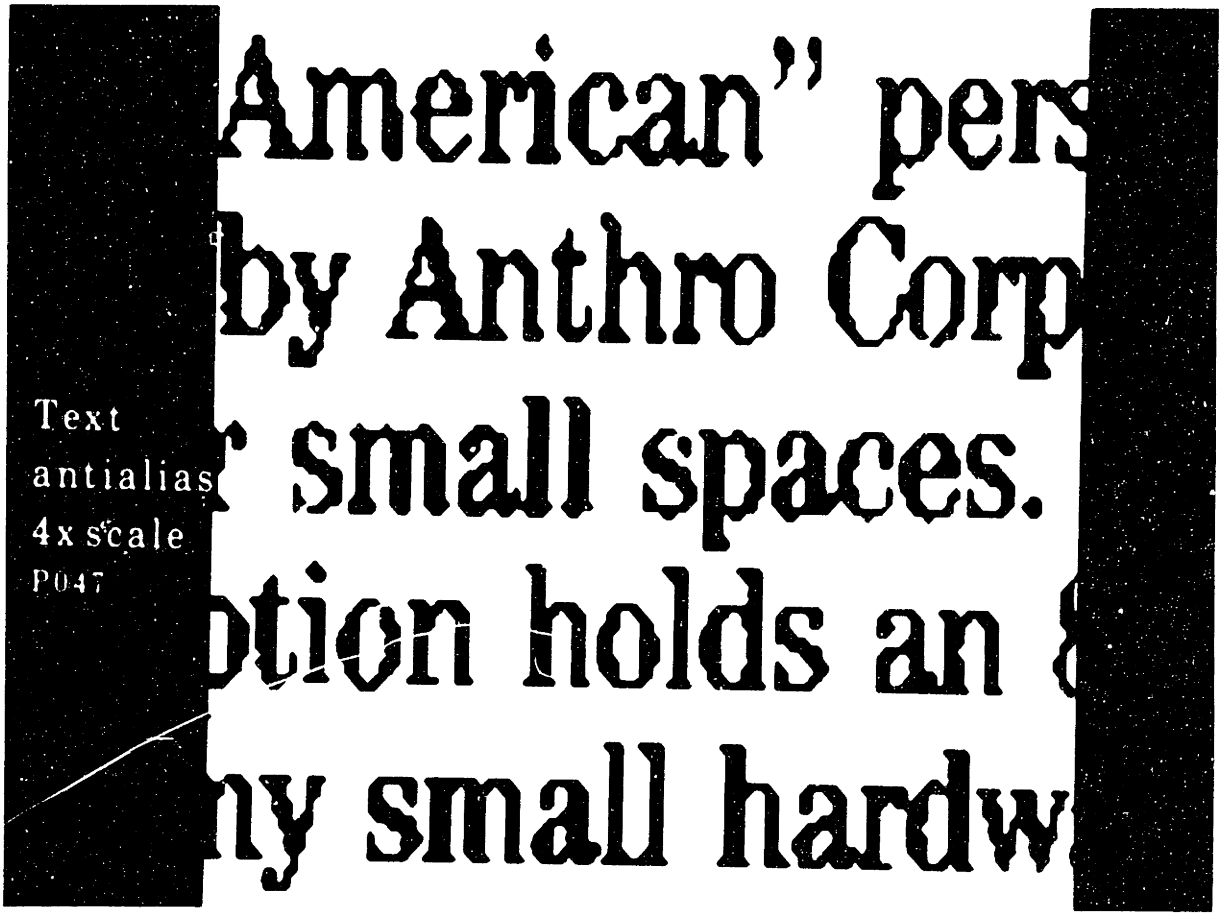


Photograph [8.1-47]: Text image scaled by four using the NSI enhanced scaling algorithm.

ID #[P045]

Photograph [8.1-48]: Eye image scaled by eight using the NSI enhanced scaling algorithm.

ID #[P044]

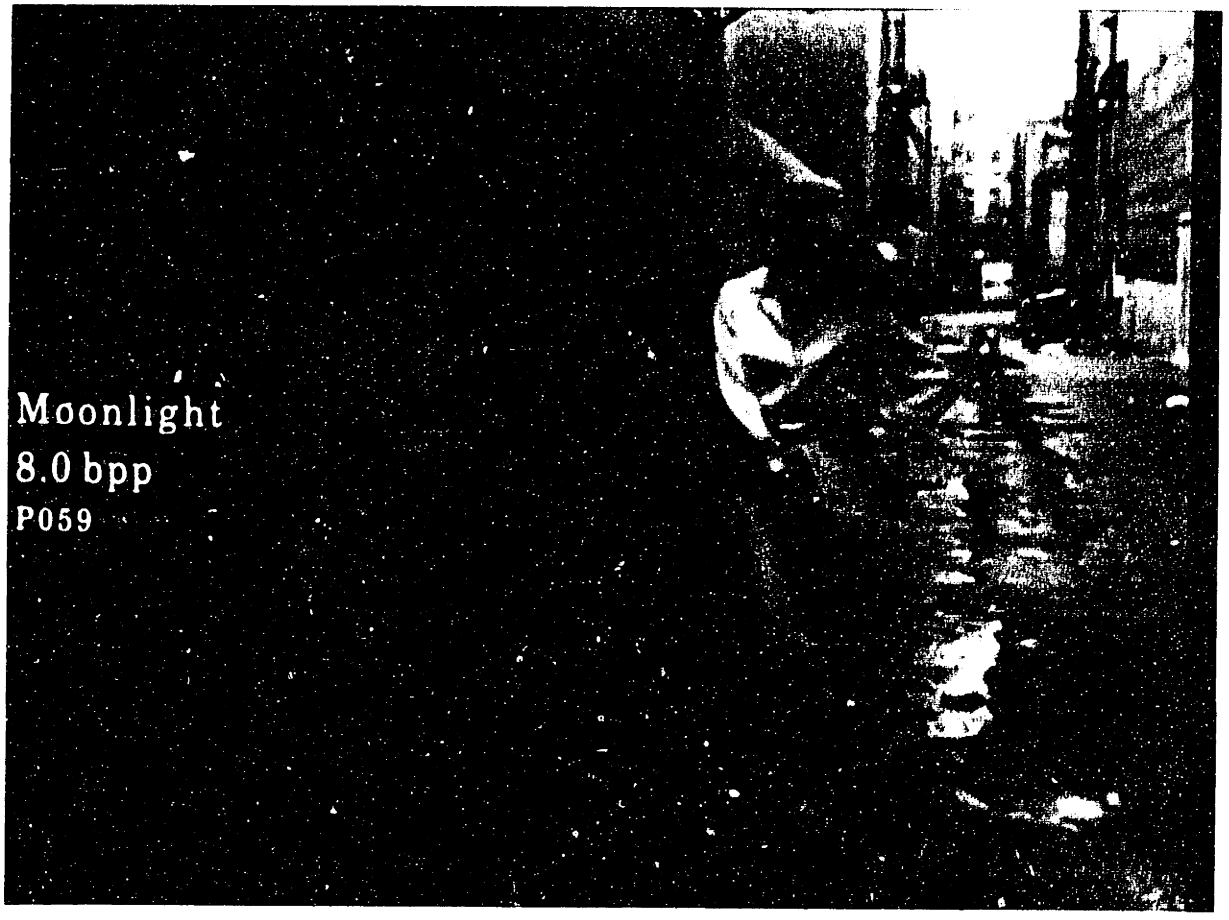


Photograph [8.1-49]: Text image scaled by four using the NSI anti-aliased scaling algorithm.

ID #[P047]

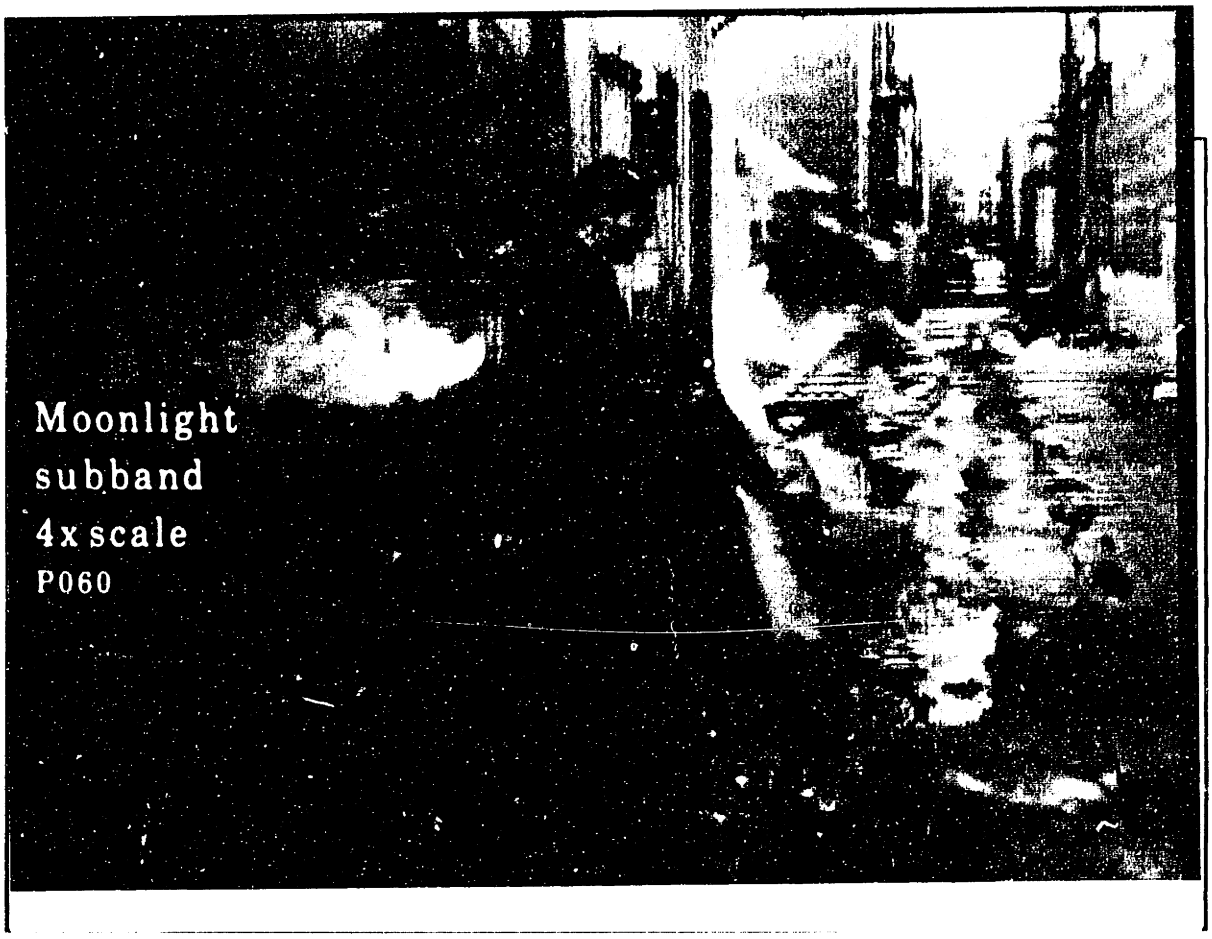
Photograph [8.1-50]: Eye image scaled by eight using the NSI anti-aliased scaling algorithm.

ID #[P046]



Photograph [8.1-51]: Original 8.0 bit per pixel Moonlight image.

ID #[P059]

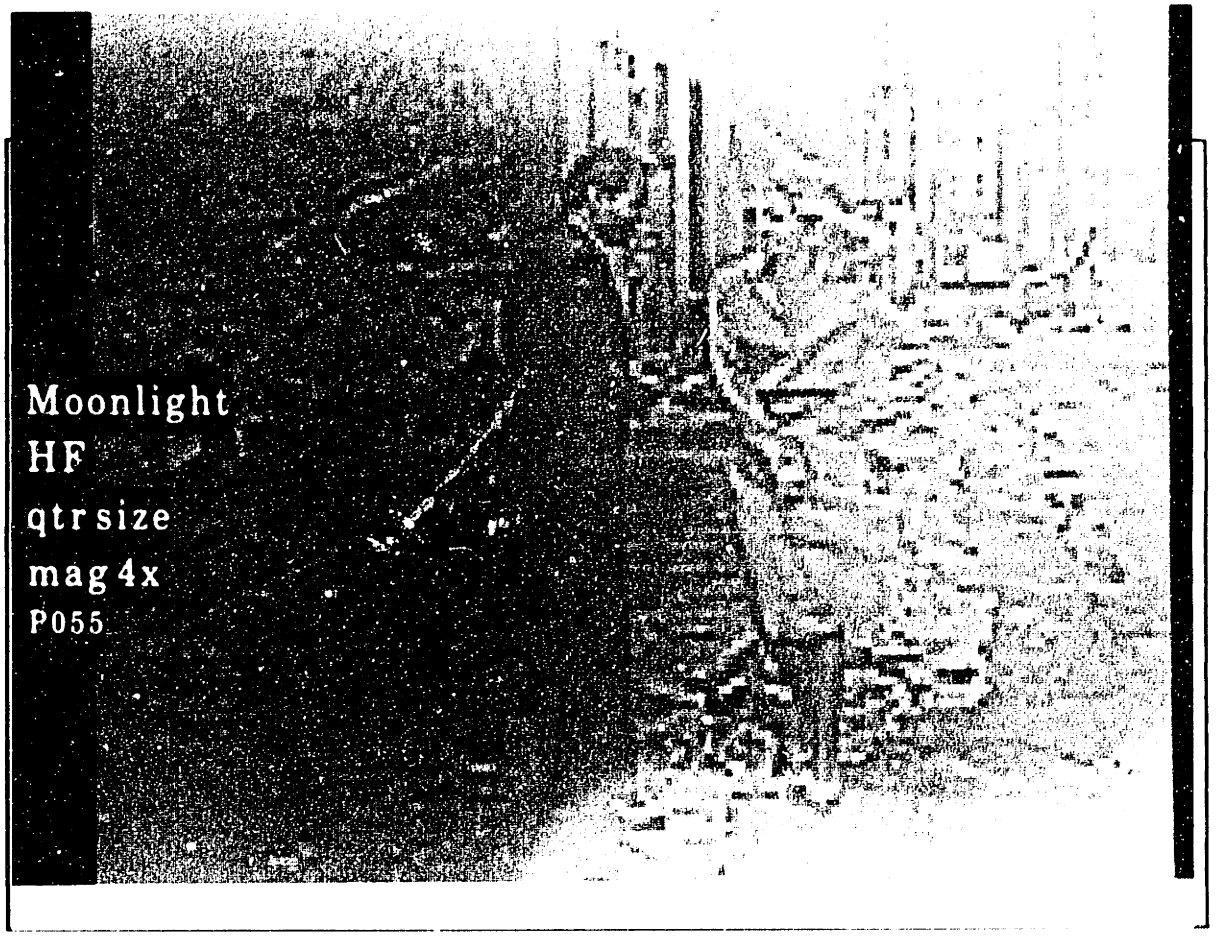


Photograph [8.1-52]: Subband compressed Moonlight image. Image is reconstructed from a three level subband decomposition. The higher level subbands were synthesized by using NSI to scale lower level subbands.

ID #[P060]

Photograph [8.1-53]: Low frequency components of the quarter sized Moonlight image in Photograph [8.1-55]. Magnified four times.

ID #[P066]



Photograph [8.1-54]: High frequency components of the quarter sized Moonlight image in Photograph [8.1-55]. Magnified four times.

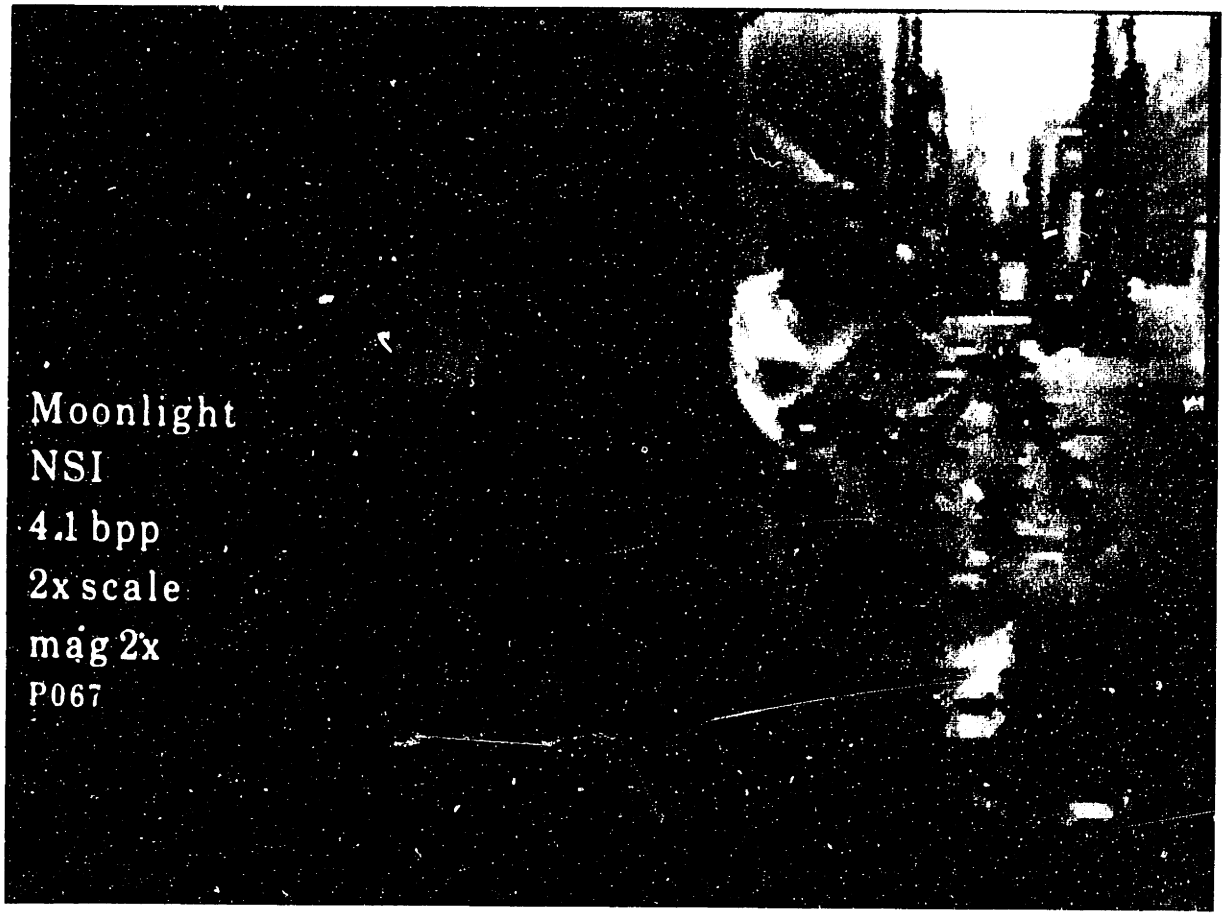
ID #[P055]

Photograph [8.1-55]: Quarter sized Moonlight image. Magnified four times.

ID #[P052]

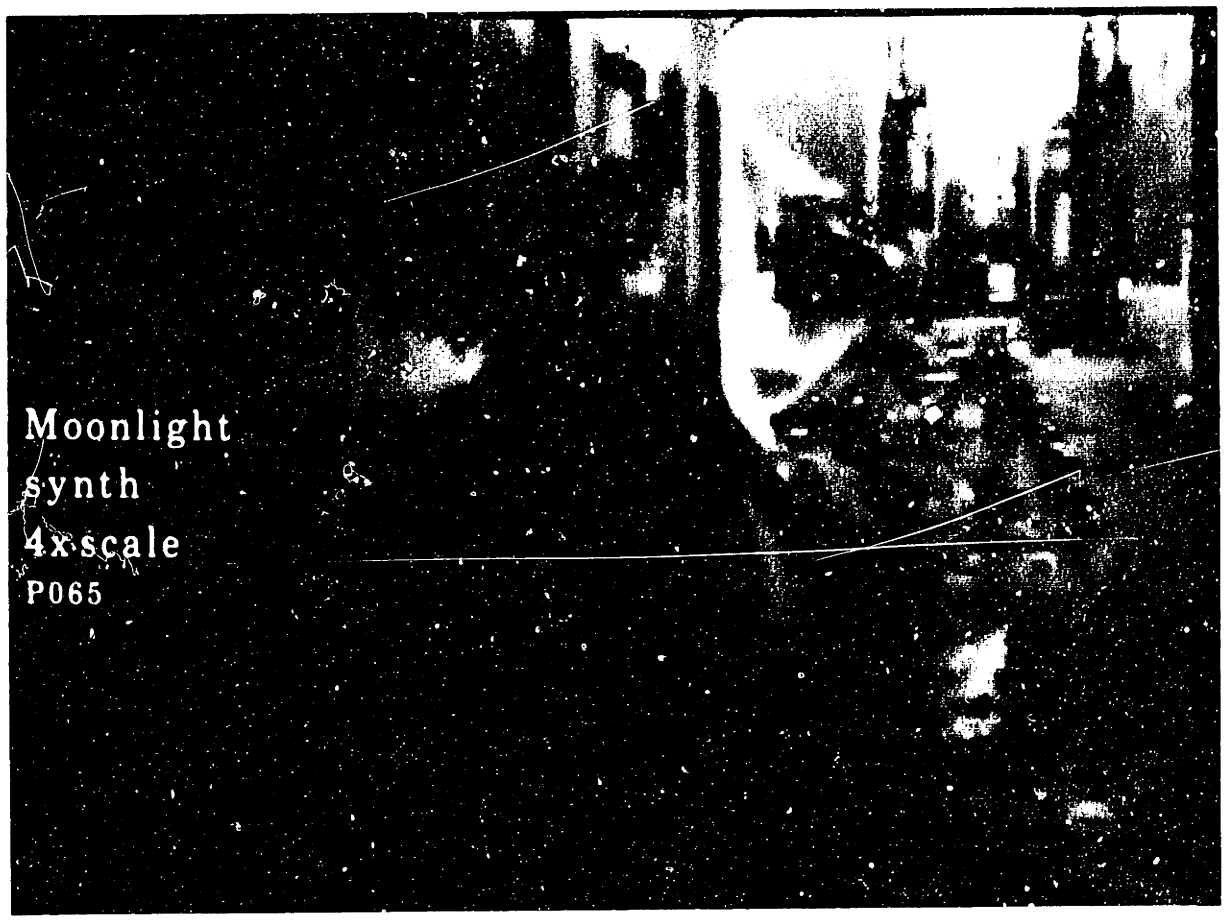
Photograph [8.1-56]: Moonlight image reconstructed at half size using a Laplacian pyramid and NSI to synthesize the scaled high frequency information. Magnified two times.

ID #[P064]



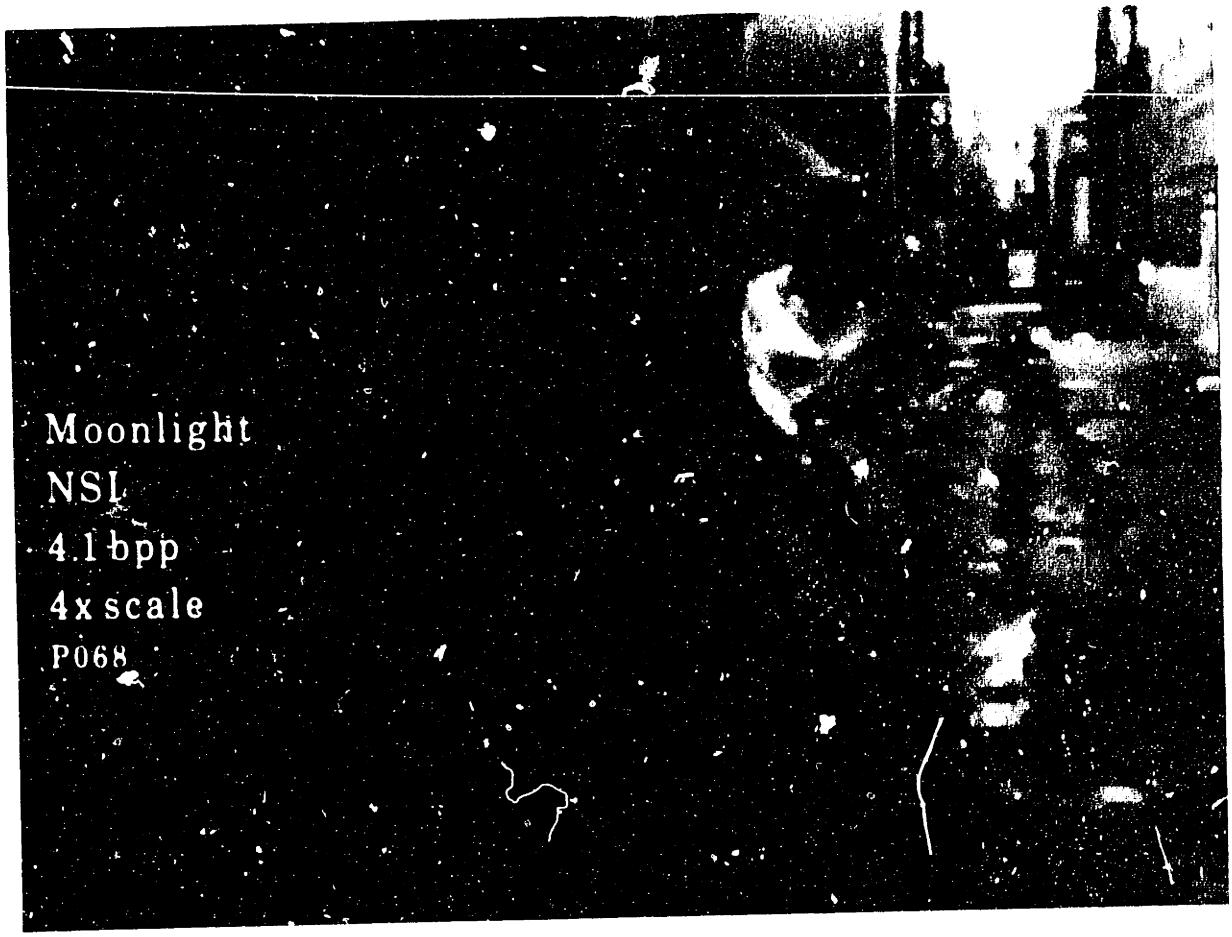
Photograph [8.1-57]: Moonlight image scaled to half size using the NSI anti-aliased scaling algorithm. Magnified two times.

ID #[P067]



Photograph [8.1-58]: Moonlight image reconstructed at full size using a Laplacian pyramid and NSI to synthesize the scaled high frequency information.

ID #[P065]



Photograph [8.1-59]: Moonlight image scaled to full size using the NSi anti-aliased scaling algorithm.

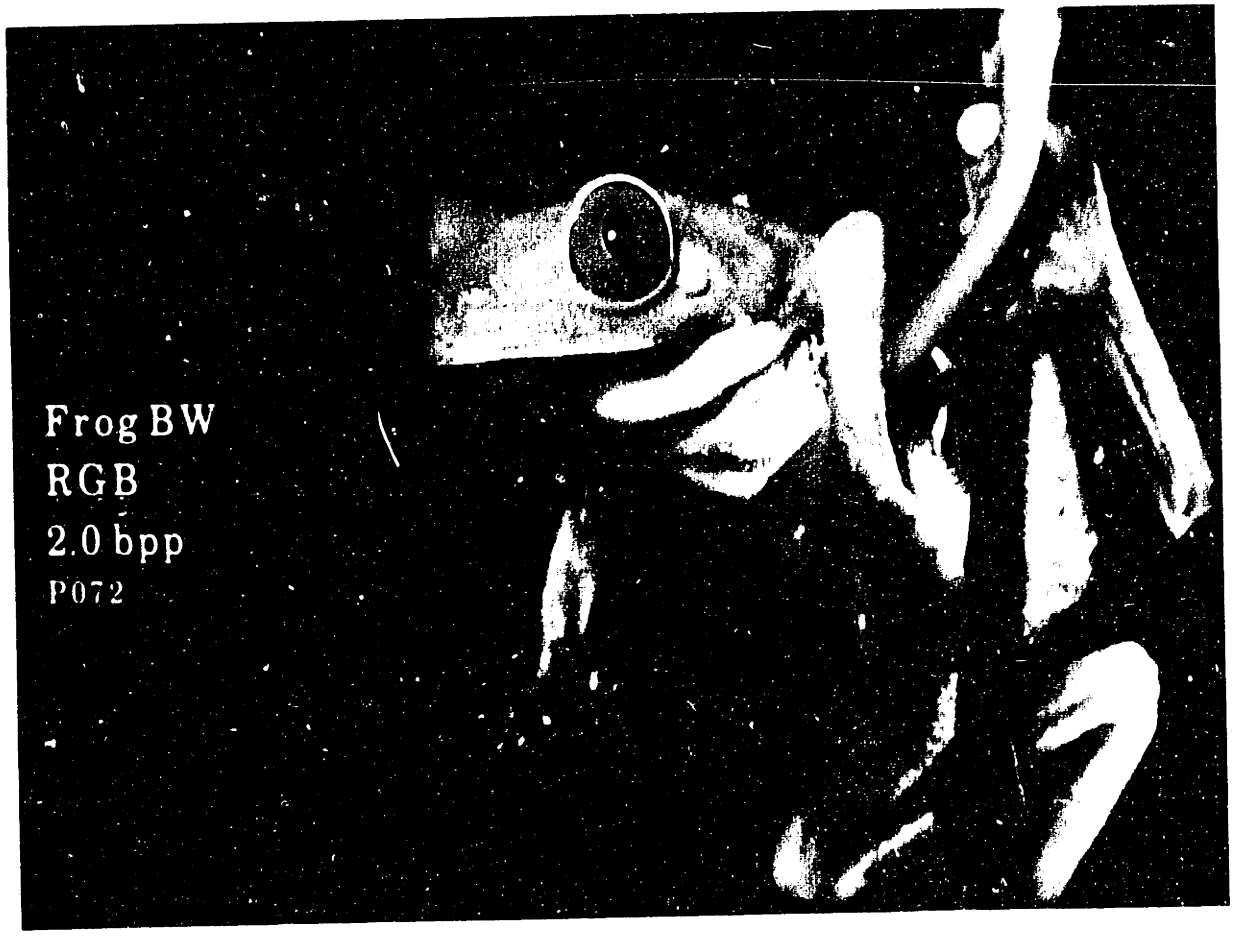
ID #[P068]

Photograph [8.1-60]: Black and white version of the color Frog image in Photograph [8.1-36].

ID# [P069]

Photograph [8.1-61]: Black and white version of the color Lena image compressed by NSI to 2.0 bits per pixel by compressing the R, G, and B channels individually. The color original of this image can be seen in Photograph [8.1-37].

ID #[PC70]



Photograph [8.1-62]: Black and white version of the color Frog image compressed by NSI to 2.0 bits per pixel by compressing the R, G, and B channels individually. The color original of this image can be seen in Photograph [8.1-38].

ID #[P072]

Photograph [8.1-63]: Black and white version of the color Lena image compressed by NSI to 2.0 bits per pixel by compressing the Y, I, and Q channels individually. The color original of this image can be seen in Photograph [8.1-39].

ID #[P071]

Photograph [8.1-64]: Black and white version of the color Frog image compressed by NSI to 1.9 bits per pixel by compressing the Y, I, and Q channels individually. The color original of this image can be seen in Photograph [8.1-40].

ID #[P073]

8.2 Subjective evaluation data

8.2.1 Statistical analysis of the data

An important part of the analysis of the data from the subjective image quality tests described in this thesis is the statistical analysis performed to evaluate which images were ranked statistically differently from each other. This analysis was performed using the Wilcoxon signed ranks test as described in [Siegel and Castellan 88]. A confidence level of 95% was used with a corresponding z-statistic of 1.641. Table [8.2-1] lists the calculated z-statistics from the experimental subjective ranking data. The actual calculations were performed by Joyce Farrell at Howlett-Packard Laboratories. Values for images pairs which were ranked equivalently are shown in bold.

Pic	D6	D5	D4	D3	D2	D1	N6	N5	N4	N3	N2	N1
D6	**0.00**	-3.72	-3.72	-3.68	-3.72	-3.72	-3.27	-3.72	-3.72	-3.72	-3.72	-3.72
D5	3.72	**0.00**	-3.72	-3.66	-3.72	-3.72	3.01	**1.09**	-2.81	-3.66	-3.72	-3.72
D4	3.72	3.72	**0.00**	-2.94	-3.72	-3.72	3.72	3.72	3.46	50.0	-3.68	-3.72
D3	3.68	3.66	2.94	**0.00**	-2.05	10.0	3.68	3.68	3.46	2.94	**-0.5**	27.0
D2	3.72	3.72	3.72	2.05	**0.00**	24.0	3.72	3.72	3.72	3.72	**1.45**	39.0
D1	3.72	3.72	3.72	110.	96.0	**0.00**	3.72	3.72	3.72	3.72	2.66	59.5
N6	3.27	-3.01	-3.72	-3.68	-3.72	-3.72	**0.00**	-3.64	-3.72	-3.72	-3.72	-3.72
N5	3.72	**-1.1**	-3.72	-3.68	-3.72	-3.72	3.64	**0.00**	-3.72	-3.72	-3.72	-3.72
N4	3.72	2.81	-3.46	-3.46	-3.72	-3.72	3.72	3.72	**0.00**	-3.62	-3.72	-3.72
N3	3.72	3.66	41.0	-2.94	-3.72	-3.72	3.72	3.72	3.62	**0.00**	-3.72	-3.72
N2	3.72	3.72	3.68	**0.52**	**-1.5**	-2.66	3.72	3.72	3.72	3.72	**0.00**	-2.20
N1	3.72	3.72	3.72	93.0	81.0	31.5	3.72	3.72	3.72	3.72	2.20	**0.00**

Table [8.2-1]: Z statistics calculated by the Wilcoxon signed rank test. Values for image pairs which are ranked equivalently are shown in bold.

8.3 Rate-distortion data

8.3.1 Overview

This section contains rate-distortion data comparing the final version of the algorithm developed in this thesis (NSI) to the discrete cosine transform (DCT) for three images Lena, Tiff and Smag. These original versions of these images can be seen in Photographs [8.1-1], [8.1-2], and [8.1-3], respectively. This data is displayed as a set of rate-distortion graphs as well as tables for the data actually contained in these graphs. (The equation for calculating peak signal to noise ratio (PSNR) is described in the body of the thesis.)

8.3.2 Data for "Lena" Image

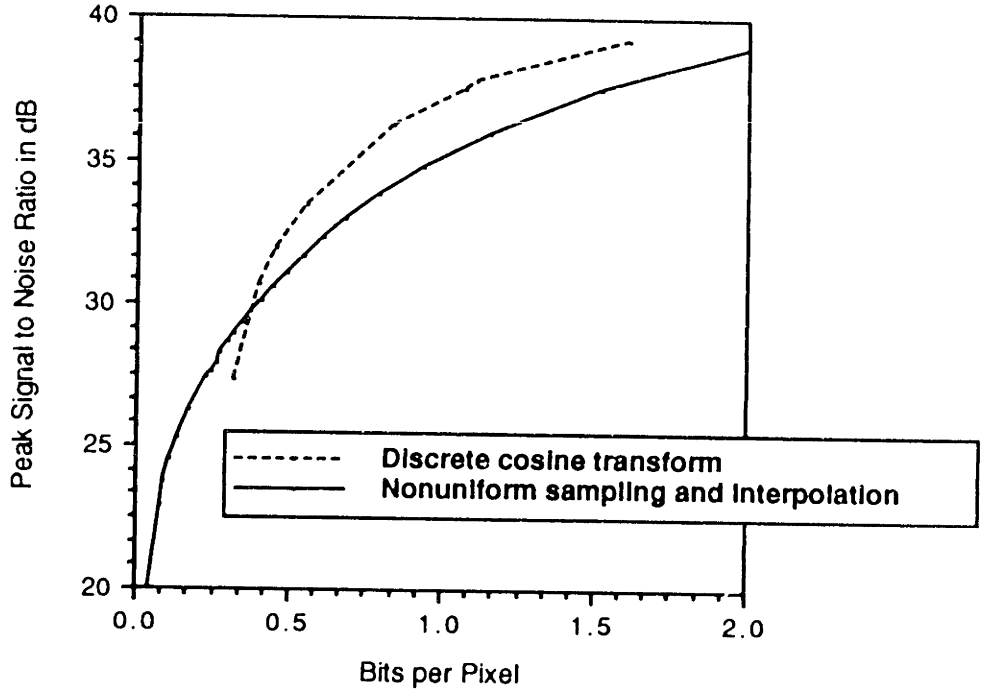
Bit rate	PSNR in dB
4.201	44.761
2.281	39.834
1.520	37.539
1.157	36.004
0.934	34.822
0.789	33.806
0.686	32.999
0.613	32.279
0.546	31.636
0.494	31.056
0.450	30.566
0.406	30.114
0.377	29.714
0.346	29.299
0.320	28.958
0.299	28.621
0.276	28.251
0.260	27.905
0.243	27.632
0.230	27.356
0.176	26.298
0.136	25.263
0.110	24.519
0.094	23.757
0.080	22.910
0.049	19.978
0.033	18.021
0.028	17.009

Table [8.3-1]: Data for NSI. Compression rate in bits per pixel versus image quality in PSNR for "Lena."

Bit rate	PSNR in dB
1.610	39.245
1.106	37.742
1.075	37.486
0.836	36.205
0.553	33.506

0.457	31.965
0.400	30.664
0.319	27.298

Table [8.3-2]: Data for the DCT. Compression rate in bits per pixel versus image quality in PSNR for "Lena."



Graph [8.3-1]: Comparative rate-distortion curves for the DCT and NSI for "Lena."

8.3.3 Data for "Tiff" Image

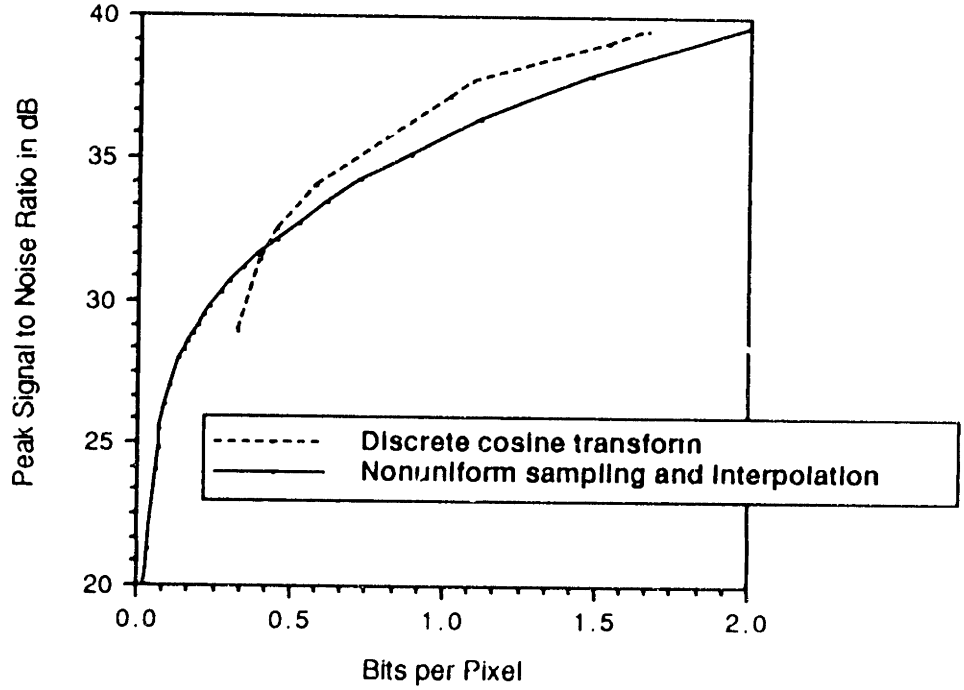
Bit rate	PSNR in dB
3.981	44.970
2.210	40.254
1.485	37.930
1.119	36.321
0.891	35.144
0.731	34.206
0.616	33.415
0.526	32.721
0.454	32.124
0.393	31.619
0.344	31.108
0.303	30.670
0.273	30.253
0.239	29.829
0.214	29.472
0.196	29.163
0.179	28.864
0.164	28.564

0.150	28.280
0.139	28.021
0.108	27.032
0.090	26.341
0.076	25.502
0.069	24.806
0.063	24.025
0.039	21.238
0.027	19.582
0.020	18.621

Table [8.3-3]: Data for NSI. Compression rate in bits per pixel versus image quality in PSNR for "Tiff."

Bitrate	PSNR in dB
1.665	39.484
1.539	39.029
1.087	37.645
1.015	37.139
0.583	34.079
0.459	32.490
0.403	31.429
0.377	28.972

Table [8.3-4]: Data for the DCT. Compression rate in bits per pixel versus image quality in PSNR for "Tiff."



Graph [8.3-2]: Comparative rate-distortion curves for the DCT and NSI for "Tiff."

8.3.4 Data for "Smag" Image

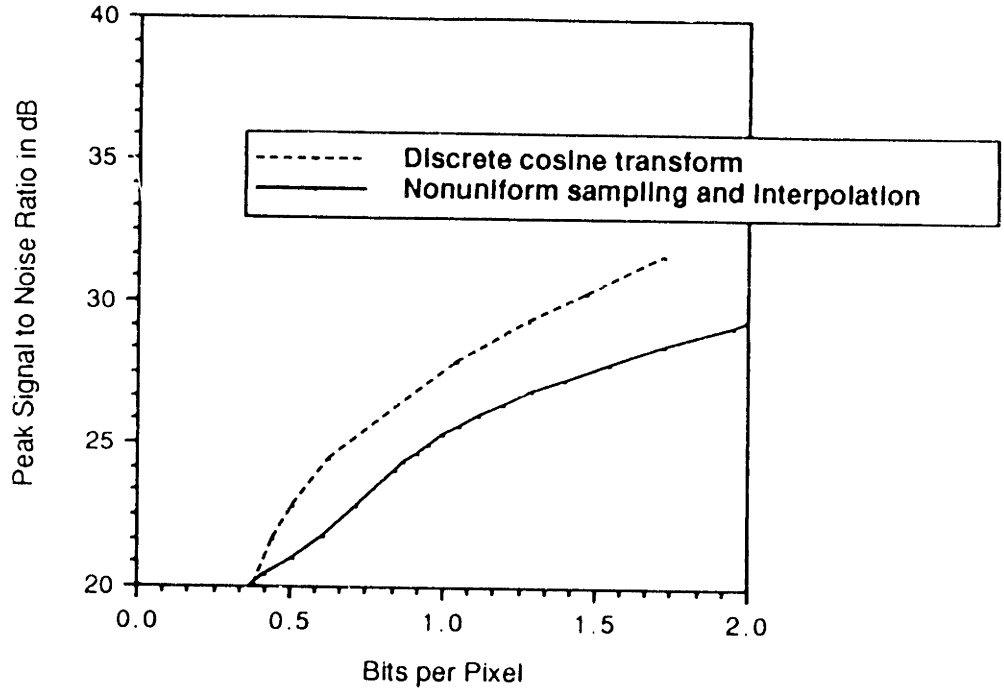
Bit rate	PSNR in dB
6.531	46.477

5.092	40.250
4.249	37.071
3.626	34.680
3.088	32.780
2.628	31.310
2.247	30.130
1.954	29.182
1.724	28.415
1.543	27.780
1.403	27.252
1.291	26.812
1.200	26.366
1.121	25.962
1.056	25.591
1.004	25.248
0.957	24.931
0.914	24.610
0.877	24.319
0.842	24.003
0.719	22.828
0.612	21.688
0.511	20.956
0.421	20.365
0.353	19.819
0.134	17.666
0.069	16.304
0.036	15.225

Table [8.3-5]: Data for NSI. Compression rate in bits per pixel versus image quality in PSNR for "Smag."

Bit rate	PSNR in dB
1.725	31.588
1.477	30.327
1.285	29.328
1.043	7.907
0.629	24.462
0.506	22.797
0.445	21.608
0.332	18.567

Table [8.3-6]: Data for the DCT. Compression rate in bits per pixel versus image quality in PSNR for "Smag."



Graph [8.3-3]: Comparative rate-distortion curves for the DCT and NSI for "Smag "