N94-33829

The Importance of Robust Error Control in Data Compression Applications

S.I.Woolley
Dept. Electrical Engineering
University of Manchester
Oxford Road, M13 9PL, U.K.
Phone: 061-275-4538
Fax: 061-257-3902
sandra@hpc.ee.man.ac.uk

Abstract

Data compression has become an increasingly popular option as advances in information technology have placed further demands on data storage capacities. With compression ratios as high as 100:1 the benefits are clear, however; the inherent intolerance of many compression formats to error events should be given careful consideration.

If we consider that efficiently compressed data will ideally contain no redundancy, then the introduction of a channel error must result in a change of understanding from that of the original source. Whilst the prefix property of codes such as Huffman enables resynchronisation, this is not sufficient to arrest propagating errors in an adaptive environment. Arithmetic, Lempel-Ziv, discrete cosine transform (DCT) and fractal methods are similarly prone to error propagating behaviours. It is, therefore, essential that compression implementations provide sufficient combatant error control in order to maintain data integrity. Ideally, this control should be derived from a full understanding of the prevailing error mechanisms and their interaction with both the system configuration and the compression schemes in use.

Introduction

Data compression is essentially the process of identifying and extracting source redundancy in order to reduce storage requirements. Since the nature of encountered redundancy is dependent upon the type of source, e.g. image, audio, video, text, program source, database, instrumentation, etc., the best compression performance is achieved by a source-specific algorithm. For example, an image source may contain a large amount of positional redundancy (e.g. a raster scan of a vertical line) which could be efficiently exploited. However, an algorithm of this type would be of little benefit if applied to a textual source. Accordingly, each of the main source categories have their own families of compression algorithms.

The following text provides an introduction to some important compression concepts, methodologies and behaviours, with the aim of demonstrating the effects of compression on data integrity and the need for adequate error control. Where possible, examples have been used to illustrate this information such that a prior understanding of compression methods and behaviours is not required.

Advantages and Disadvantages of Data Compression

In addition to the increase in data storage capacity, real-time compression implementations can enhance data transfer rates, improve network performance by reducing traffic loads on lines and, via the encryption process, provide additional data security against unauthorised access [1].

There are, however, disadvantages involved in data compression. Firstly, the compressed data is significantly more vulnerable to error, as will be demonstrated below. Secondly, the majority of software compressors do not work in real-time such that data retrieval and storage can be significantly delayed. This is particularly noticeable with most image compression methods, although the delay is, of course, dependent on system performance and source size. In addition, the plethora of algorithms currently available can result in compatibility problems when transferring data between systems. For example, the absence of an early compression standard for DAT (Digital Audio Tape) drives resulted in two competing and incompatible implementations, such that compressed DAT tapes cannot be freely transferred between systems which would otherwise be compatible [2]. Compatibility is also deteriorated, to some extent, by the existence of a variety of patents which make bespoke compressors more attractive.

Channel Errors and Data Integrity

There are two types of channel errors; amplitude errors and phase errors. In digital form amplitude errors appear as inversions of bit values whilst phase errors appear as the insertion or extraction of one or more bits in the data stream. As shown in Figure 1, amplitude errors affect only those bits which are directly afflicted, whereas phase errors result in a stream of amplitude errors which propagate to the end of the information source or until the next resynchronisation marker.

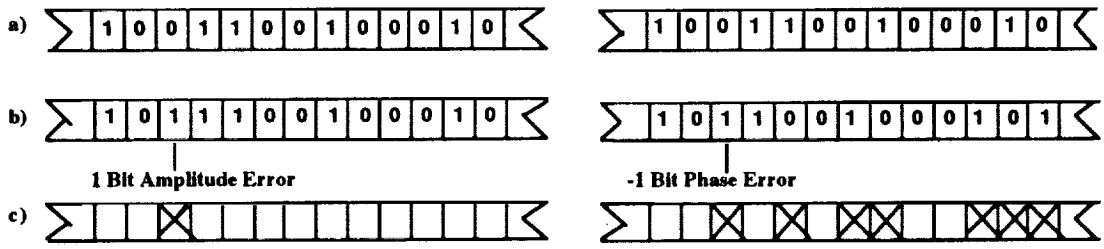


Figure 1 : Example of Channel Errors
a) Source data, b) Source data with channel error imposed, c)Resultant bits in error.

All systems are prone to error mechanisms of one sort or another, and with an understanding of these phenomena the subsequent, or secondary, effects on data integrity can be ascertained. For example, in magnetic tape recording systems a primary source of errors is the existence of embedded asperities, introduced either via the manufacturing process or via external contamination. The secondary effect of these mechanisms is the burst error syndrome caused directly by a departure of head-to-tape contact which attenuates signal amplitude. This phenomenon is referred to as a dropout, and where the attenuation results in effective signal loss then, as well as a burst of amplitude errors, the system may also experience a loss of synchronisation, i.e. a phase error. Therefore, although these events are comparatively rare their potential for error propagation makes them the dominant contributors to the overall error rate. So whilst bit/byte error rates are important indicators of system performance, the distribution and characterisation of error events are similarly important, and can be used to determine suitable error control measures i.e. interleaving, resynchronisation markers and error control coding.

Lossy and Lossless Compression Techniques

The amount of compression that can be achieved by a given algorithm depends on both the amount of redundancy in the source and the efficiency of its extraction. The very high compression ratios often quoted generally relate to high-redundancy sources such as databases or to lossy compressed formats such as fractal image representations.

Lossless techniques are, of course, required by many applications, such as computer disk compression, where exact replication of the original is essential. Alternatively, lossy compression techniques, where only a close approximation of the original is reconstructed, can be successfully applied to many image, video and audio applications where losses outside visual/aural perception can be tolerated, and where the additional compression achieved is highly desirable. For example, the philosophy behind the lossy DCT (Discrete Cosine Transform) compression of images is that the human eye is less sensitive to high-frequency information. Further compression can, therefore, be achieved by more coarsely quantising high-frequency components of an image without significant visual deterioration.

There are, however, some image compression applications where certain quality and/or legal considerations dictate the use of lossless compression. For example, medical imaging [3] and deep space communication of imagery data [4].

Static and Adaptive Implementations

Compression algorithms remove source redundancy by using some working definition of the source characteristics, i.e. a source model. Compression algorithms which use a pre-defined source model are referred to as static, whilst algorithms which use the data itself to fully or partially define this model are referred to as adaptive.

Static implementations can achieve very good compression ratios for well defined sources; however, their inability to respond to changes in source statistics limits their usage. If applied to a source significantly different from that modelled, a static algorithm could result in source expansion rather than compression.

In contrast, adaptive algorithms are more versatile, and will update their working source model according to current source characteristics. However, adaptive implementations have lowered compression performance, at least until a suitable model is properly generated. In addition,

"A major problem with any adaptive compression method is that a single error on the communication channel or storage medium can cause the sender and receiver to lose synchronisation and can result in error propagation that in the worst case can corrupt all data to follow."[5]

Compression Methodologies

The following text presents a selection of commonly used compression methods and investigates the effects of channel errors on data integrity. Image sources and bit error maps are used to illustrate these phenomena since the effects of any errors are more readily appreciated.

Huffman Coding

Huffman compression techniques are examples of statistical coding, where character frequency statistics are used to allocate appropriate codewords for output. The basic philosophy is that compression can be achieved by allocating shorter codewords to the more frequently occurring characters (a simple example of this technique is the Morse Code where, for example, E= • and Y= -•--). As shown in Figure 2, by arranging the source alphabet in descending order of probability, then repeatedly adding the two lowest probabilities and resorting, a Huffman tree can be generated. The resultant codewords are formed by tracing the tree path from the root node to the codeword leaf.

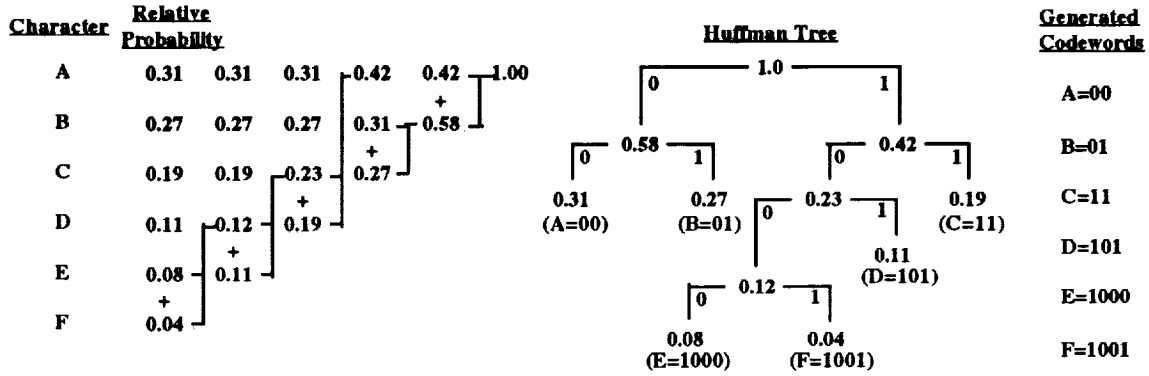


Figure 2 : An Example of Huffman Coding

The relative probabilities and, hence the Huffman tree, can be derived by the compressor in three ways. Firstly, in static implementations the probabilities are predefined. This enables rapid and efficient compression and decompression but only for sources whose character frequencies are accurately described by those of the algorithm. Secondly, the probabilities can be generated by an initial pass of the data to count the character frequencies, but this requires an additional second pass to perform the compression. Thirdly, the probabilities (and hence the codewords) can be adjusted dynamically during the compression process. This adaptive Huffman method, although more complex, can still perform quite rapid compression and decompression and has the advantage of being more versatile as well as being able to respond dynamically to changes in the source.

Compression performance can be improved by increasing the order of the source model (i.e. considering the context of incoming characters). For example, when the letter "q" is parsed from a stream of english text there is a very high probability (approx. 95%) that the letter "u" will follow.

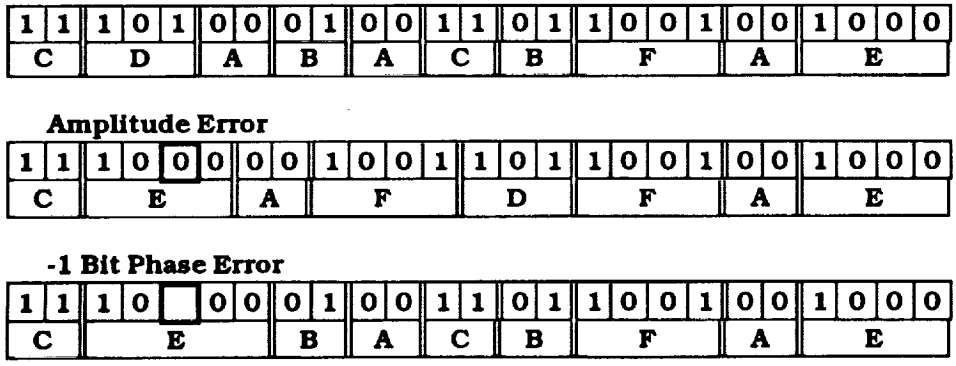


Figure 3 : Resynchronisation of Huffman Coded Data in the Event of Channel Errors

There is, therefore, no single Huffman method, but rather a whole family varying in adaptivity and order[6]. The specific effects of any transmission errors on Huffman compressed data would therefore depend on the particular implementation as well as the source. Fortunately, however, some generalisations can be made. Firstly, static implementations have an in-built resistance to error propagation. This is due to the prefix property common to all Huffman codes, whereby no code is a prefix of any other. For example, using the generated codewords in Figure 1, if 00 is a permitted codeword then no other codeword may begin with the sequence 00, similarly if 101 is a permitted codeword then no other codeword may begin with the sequence 101, and so on. Figure 3 illustrates the effects of an amplitude and a phase error, demonstrating the code's self-synchronising ability.

Adaptive Huffman implementations, however, are not protected by this facility since; *"The fact that the sender and receiver are dynamically redefining the code indicates that by the time synchronisation is regained, they may have radically different representations of the code"* [7].

Figure 4b below shows the effects of a single bit error on an image source compressed by an adaptive Huffman scheme.

a) Original

b) Adaptive Huffman with 1 Bit Error

c) Arithmetic Compression with 1 Bit Error

Figure 4 : Examples of Bit Errors on Arithmetic and Adaptive Huffman Formats

Arithmetic Coding

The method of compression employed by Huffman compression coding involves the allocation of shorter codewords for more frequently occurring characters. It is, however, unable to allocate fractional codeword lengths, so that a character must be allocated at least a one-bit codeword no matter how high its frequency. Huffman coding cannot, therefore, achieve optimal compression.

Arithmetic coding [8] offers an alternative to Huffman coding, enabling characters to be represented as fractional bit lengths. This is achieved by representing the source as a real number, greater than or equal to zero, but less than one, denoted as the range [0,1). As shown in Figure 5, each character of the source alphabet is allocated a proportion of this range according to its probability. As data symbols are parsed from the source, the working range is reduced and a real number is generated which can then be transmitted and decompressed via a parallel process. The source sequence shown in column four of Figure 5 illustrates this process. As can be seen the initial range is [0,1) which is reduced to [0.2,0.5) when a "B" is encountered. The next symbol, an "A", requires the range [0,0.2), i.e., the first 20% of the working range, [0.2,0.5), hence the new range [0.2,0.26).

Source Characters	Relative Probabilities	Allocated Range	Source Sequence	Denoted Range
A	0.2	[0,0.2)	---	[0,1)
B	0.3	[0.2,0.5)	B	[0.2,0.5)
C	0.1	[0.5,0.6)	A	[0.2,0.26)
D	0.2	[0.6,0.8)	C	[0.23,0.236)
E	0.1	[0.8,0.9)	C	[0.233,0.2336)
F	0.1	[0.9,1.0)	F	[0.23354,0.2336)

Figure 5 : An Example of Arithmetic Coding

The handling of the necessary floating point arithmetic and the need for special terminating sequences makes arithmetic coding more complex than Huffman coding. However, the algorithm achieves close to optimal compression, and, like Huffman coding, can be increased in order and adaptivity.

Unfortunately, arithmetic coding, whether static or adaptive, is particularly vulnerable to errors, whereby a single bit error can result in a complete scrambling of all subsequent data [**Teuhola91**], as demonstrated in Figure 4c.

DATA SEQUENCE : "THE THREE TREES"					
Previous Character or String	New Character	Generated Dictionary Codeword	Meaning of Dictionary Codeword	Code Output	Meaning of Output
---	T	---	---	---	---
T	H	D1	TH	T	T
H	E	D2	HE	H	H
E	T	D3	ET	E	E
T	H	The string "TH" already exists so a new dictionary codeword is not generated.	---	---	---
TH	R	D4	D1+R =THR	D1	TH
R	E	D5	RE	R	R

E	E	D6	EE	E	E
E	T	The string "ET" already exists so a new dictionary codeword is not generated.	---	---	---
ET	R	D7	D3+R =ETR	D3	ET
R	E	The string "RE" already exists so a new dictionary codeword is not generated.	---	---	---
RE	E	D8	D5+E =REE	D5	RE
E	S	D9	ES	E	E
S	End of data	---	---	S	S

Figure 6 : Simplified Example of LZ78 Compression Process
The original source contains 13 8-bit characters (=104 bits) and the compressed output contains 10 9-bit codewords (=90 bits).

Lempel-Ziv Methods

There are two Lempel-Ziv compression algorithms: LZ77 [10] and LZ78 [11], both of which compress data by replacing repeated strings by defined codewords. LZ77 uses a so-called sliding window from which repeated strings are identified and referenced by a coded block indicating the position and length of the string. In this way LZ77 creates a compressed format which comprises uncompressed data interspersed with pointers to recognised strings. This simple format enables rapid decompression; however, compression is comparatively slow since continuous searches of the sliding-window are required.

Alternatively, the LZ78 algorithm (and similarly LZW [12], an improved and patented version) creates a dynamic embedded dictionary designed with a self-referencing structure. The algorithm parses the data source for unique strings (i.e., strings not previously encountered) for which it allocates dictionary codewords that can be used to replace the string if it occurs again. The implementation of this method is best explained by means of a simple example as shown in Figure 6 (reading left to right, row by row). In its simplest form all of the output (characters and dictionary codewords) are in 9-bit form, but can be increased to 10, 11 or 12 bits as required by sending reserved codewords (control flags). These are also used to fully or partially reset the dictionary when it fills or when significant deterioration in compression performance is detected. The simplified example shown in Figure 6 describes dictionary entries as D1, D2, D3 etc., and a high-redundancy data sequence ("THE THREE TREES", with spaces ignored) is used in order to demonstrate the compression of repeated strings.

During compression, the self-referencing nature of the dictionary enables longer and longer strings to be replaced by just one dictionary codeword, and there is no need to explicitly write the dictionary contents since it can be regenerated via the same process on decompression. Initially the compression performance of the algorithm is poor, but, as strings are re-encountered and replaced with dictionary codewords performance increases rapidly.

The speed and versatility of the Lempel-Ziv implementations have made them particularly popular and have led to the development of a large and growing family of related algorithms

([13] refers to 12 variants of Lempel-Ziv methods). Implementations of Lempel-Ziv type algorithms can be found in most computer disk compressors and on tape drives including DAT.

Channel Errors in LZ78 Compressed Data Formats

The introduction of amplitude errors into the compressed format can have the following effects:

1. a small number of amplitude errors restricted to the locality of the original,
2. a propagation of amplitude errors from the point of error location to the end of the entity, and
3. a complete loss of data either occurring abruptly or following a variable-length burst of amplitude errors.

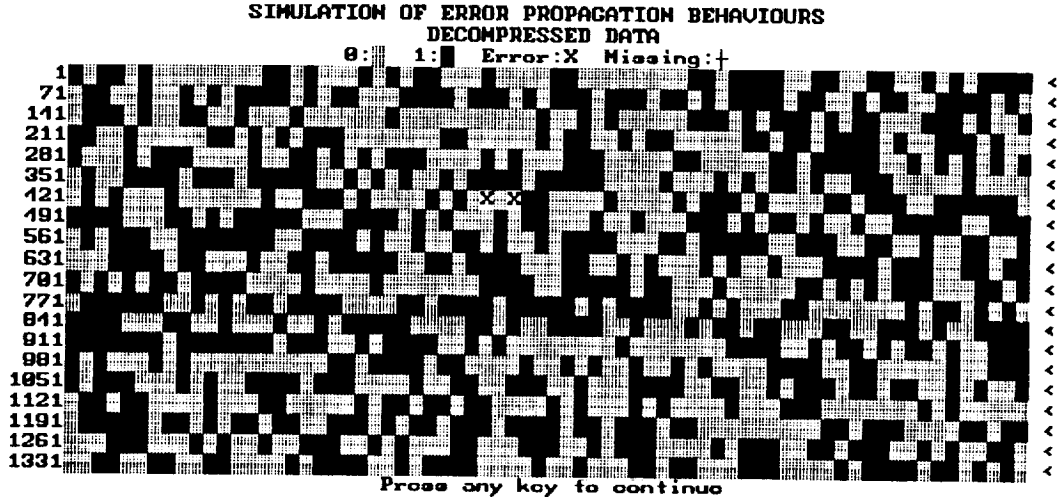
These outcomes are explained in Figure 7 by considering the types of compressed source involved.

The Effects of Channel errors on LZ78 Compressed Data		
Location of Error	**Description**	**Resultant Effect on Data Integrity**
a) In an output character.	(i) The character value is altered.	The error would remain in the decompressed output and be replicated for each repeating string in which the character was parsed.
	(ii) The character is transposed into a dictionary codeword.	*The difference in length alone will produce a synchronisation loss.
	(iii) The character is transposed into a control flag.	**The source will be corrupted or lost since control flags adjust byte lengths, manipulate the dictionary, and indicate the end of data.
b) In an output dictionary codeword.	(i) The error transposes the dictionary codeword into another [recognised] dictionary codeword.	The referenced string is replaced by an erroneous string. If these are of different lengths then synchronisation will be lost, and any future references to the newly created dictionary codeword will be affected in the same way.
	(ii) The error transposes the dictionary codeword into an unrecognised dictionary codeword.	The decompressor will be unable to identify the codeword and subsequent data will be lost.
	(iii) The error transposes the dictionary codeword into an output character.	As *.
	(iv) The error transposes the dictionary codeword into a control flag.	As **.

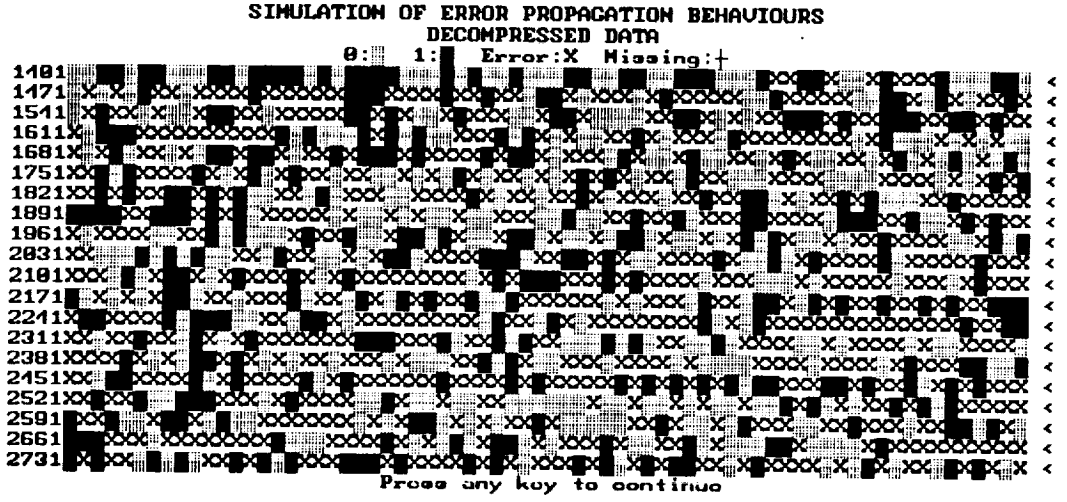
c) In a control flag.	(i)The control flag is mistaken for an output character or a dictionary codeword.	As **.

Figure 7 : A Description of the Potential Effects of Channel Errors on an LZ78 Compressed Source.

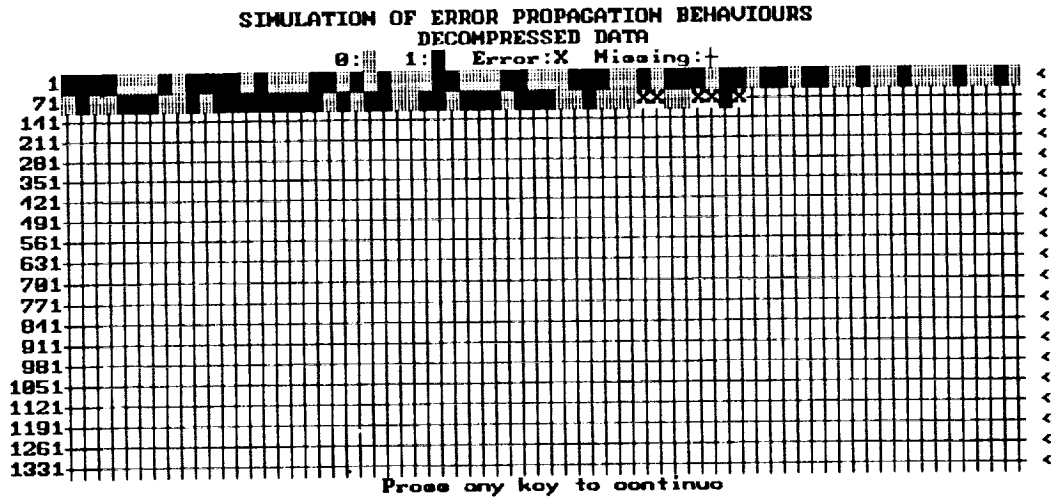
Figure 8 below shows 3 different bit error maps: a), b) and c), of LZ78-type compressed sources which correspond to **1.**, **2.** and **3.** above. These results were generated via the comparison of the original source with the decompressed output of the same source, but with a single-bit error introduced into the compressed form.



a)



b)



c)

Figure 8 : The effects of single amplitude errors on LZ78 compressed data.
a) Recovery, b) Error Propagation and c) Data Loss

Discrete Cosine Transform (DCT) Image Compression

The philosophy behind DCT image compression is that the human eye is less sensitive to high-frequency information (and also more sensitive to intensity than to colour), so that compression can be achieved by more coarsely quantising the large amount of high-frequency components usually present. Firstly, the image must be transformed into the frequency domain. Since it would be computationally prohibitive to transform even a low resolution image in one full block, the image is subdivided. The developing JPEG (Joint [CCITT and ISO] Photographic Experts Group) standard algorithm **[14]** for full-colour and grey-scale image compression uses 8x8 blocks.

The DCT itself does not achieve compression, but rather prepares the image for compression. Once in the frequency domain the image's high-frequency coefficients can be coarsely quantised so that many of them (>50%) can be truncated to zero. The coefficients can then be arranged so that the zeroes are clustered and Run-Length Encoding (RLE), whereby repeated values are referenced and followed by a counter indicating the number of successive occurrences, can be applied. The remaining data is then compressed with Huffman coding (arithmetic coding has also been proposed but implementation has been hampered by patent issues).

DCT compression, therefore, involves a number of processes, all of which combine to allow extensive error propagation. An example of the effects of a single bit error is shown in Figure 9g. The RLE coding alone is prone to error propagation since the length or value of referenced symbol repetitions can be altered by a single amplitude error. The JPEG standard addresses this problem by providing "restart" markers which allow the decoder to resynchronise after a transmission error **[JPEG Version 4:USAGE]**. This facility does not correct errors, but arrests their propagation. The number of markers used is user defined and should be determined by the channel error statistics, the error tolerance of the system, and the reduction in compression which can be tolerated as a result of their insertion.

The lossy nature of the DCT method is shown in Figure 9. As can be seen in Figure 9b, significant compression can be achieved without any visible loss in picture clarity. However,

by increasing the compression performance further losses are seen to develop (particularly lossy examples were chosen since the effects were required to be visible after reduction for inclusion in this document).

The presence of artifacts around sharp edges is referred to as Gibb's phenomenon **[15](pg225)**. These are caused by the inability of a finite combination of continuous functions to describe jump discontinuities. As shown in Figure 9e, at higher compression ratios these losses become mores apparent, as does the blocked nature of the compressed form. Figure 9f shows a difference mapping of the original and the highly lossy decompressed image in which the loss of edge clarity can be observed.

This type of lossyness makes JPEG and other DCT-based algorithms unsuitable for non-realistic images, e.g. line drawings, cartoons, etc., as can be seen by the large amount of deterioration in the geometric example used in Figure 10.

Fractal Image Compression

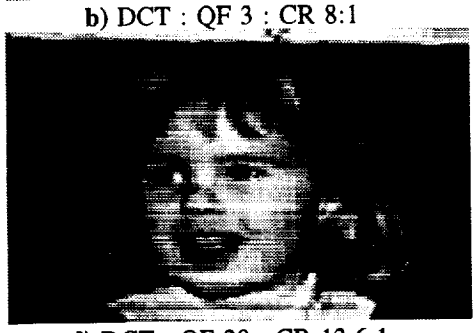
A fractal, in simplest terms, is an image of a texture or shape expressed as one or more mathematical formulae. It is a geometric form whose irregular details recur at different locations, scales and angles, and which can be described in terms of formulae called affine or fractal transformations **[16]**. Fractal image compression is achieved by dividing an image into sub-blocks, each of which is then compared to scaled and rotated versions of the other sub-blocks in the image. When sufficiently similar sub-blocks have been found for all of the sub-blocks in the image, they can be referenced geometrically so that a fractal description is obtained.

Unlike other image compression techniques, the fractal transform is a very asymmetric process. The exhaustive searching for self-similarity requires intensive computational power, but once expressed in terms of fractal transforms the image can be quickly decompressed. Fractal compression is therefore best suited to WORM applications. For example, the Microsoft multimedia encyclopaedia Encarta makes use of fractal compression to store hundreds of colour maps and thousands of photographs on a single CD which also contains extensive audio, animation and textual data. **[15]**.

a) Original

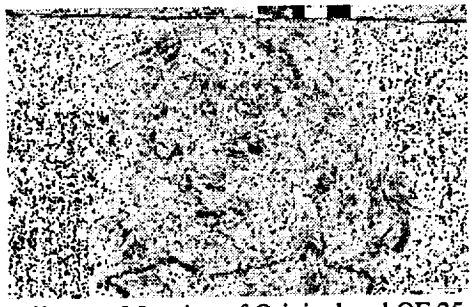
b) DCT : QF 3 : CR 8:1

c) DCT : QF 10 : CR 11.6:1



d) DCT : QF 20 : CR 13.6:1

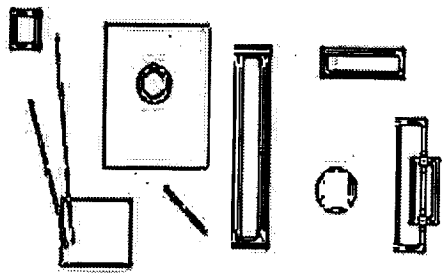
e) DCT : QF 25 : CR 14.2:1



f) Difference Mapping of Original and QF 25

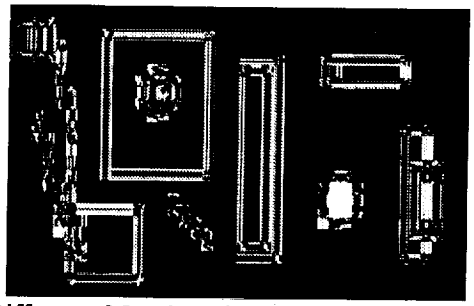
g) DCT (8:1) with 1 Bit Error

Figure 9: **b)-e)** DCT Compressed Images of **a)** : QF (Quality Factor) in the range [1-25] (best-worst)
f) Comparison of a) and e)
g) Data Loss due to a Single Bit Error



Original

DCT : QF 25 : CR 11.6:1



Difference Mapping of Original and QF 25

Figure 10 : Edge Distortions Produced by DCT Compression

The large amount of self-similarity in "real-world" images enables fractal image compression to achieve very high compression ratios, usually significantly higher than those for DCT compression. But like DCT compression, fractal compression is unsuited to geometrical images where self-similarity is not evident [17].

The effects of small numbers of bit errors on fractal compressed images can result in severe degradation of the afflicted sub-blocks and propagate into the referenced self-similar sub-blocks. However, the iterative nature of fractal image generation means that these distortions will have reduced contributions.

Since fractal compressed images are represented as mathematical structures they are size-independent. Compression ratios can therefore be increased by comparing the size of the fractal form to the size of the enlarged original, rather than the original itself.

Compression of Instrumentation Data

The compression of instrumentation data is a comparatively neglected area. This can be explained by the variance in byte lengths and also the application specific nature of generated sources. However, one method which can be more generally employed is difference modulation, whereby data symbols are encoded as the difference (positive or negative) from the previous symbol. For example:

42 41 43 43 45 44 42 39 40

could be represented as;

42 -1 +2 0 +2 -1 -2 -3 +1

which, of course, could be easily compressed by using fewer bits to represent the differences. Instrumentation data is suited to difference modulation since it often involves either visible trends relating to gradual changes in a monitored process, or relatively small perturbations

centred about some mean value. Difference modulation is also useful in audio compression applications, where trends can be identified in the source waveform. In applications were losses can be tolerated, the quantisation of differences can achieve further compression.
In the presence of channel errors difference modulation will result in error propagation since each of the symbols measures itself against its predecessor. The resultant shift caused by a bit error will be replicated in all subsequent symbols on decompression, propagating until the end of the data sequence or until the next true measurement is parsed.

Conclusion

In the absence of data compression many systems, when afflicted by uncorrected channel errors, will suffer only localised losses in data integrity, i.e. will fail gracefully.. However, similar errors in systems using data compression can have disastrous results. For this reason users of computer disk compression are advised to take regular backups since: *"if something does go wrong, it is likely to be major"*.[18].

The results of transmission errors on different compression methodologies has been demonstrated, and the need for robust error control emphasised. This control should be determined by the channel error statistics and the error tolerance of the system. In addition to error control coding, piece-wise compression, resynchronisation markers and deep interleaving can also be employed to limit the propagation of errors and reduce the correction burden placed on the error control coding.

Acknowledgements

Much of the work presented here was sponsored by British Gas Plc, whose financial and technical support has been greatly appreciated. I am also very grateful for the interest and support given by ICI Imagedata which has enabled the publication of this work. Finally, I would like to thank to my research supervisor, Prof.B.K.Middleton, and Dr.B.Bani-Eqbal (Dept. Computer Science, University of Manchester) for his technical advice on fractal and DCT compression implementations.

References

[1] **The Hidden Benefits of Data Compression**
D.Powell
Networking Management, Vol.7, No.10, October 1989, pp46-54

[2] **The Compression Wars**
J.McLeod
Electronics, Vol.64, September 91, pp27-28

[3] **Performance Analysis of Reversible Image Compression Techniques for High-Resolution Digital Teleradiology**
G.R.Kuduvalli and R.M.Rangayyan
IEEE Transactions on Medical Imaging, Vol.11, No.3, September 1992, pp430-445

[4] **Recent Advances in Lossless Coding Techniques**
G.S.Yovanof
Proc-27th Int. Telemetric Conf. ITC91, pp7-19

[5] **Data Compression**
A.Bookstein and J.A.Storer
Information Processing and Management, Vol 28, No.6, 1992, pp675-680

[6] **Overview of Huffman Encoding as [a] Compression Technique**
K.Anderson
Computer Technology Review, Vol.11, No.6, 1991, pp97-101

[7] **Data Compression**
D.A.Lelewer and D.S.Hirschberg
ACM Computing Surveys, Vol.19, No.3, September 1987, pp261-296

[8] **An Introduction to Arithmetic Coding**
G.G.Langdon
IBM Journal of Research and Development, Vol.28, No.2, March 1984, pp135-149

[9] **Piecewise Arithmetic Coding**
J.Leuhola and T.Raita
Proceedings of DCC'91, pp33-42

[10] **A Universal Algorithm for Sequential Data Compression**
J.Ziv and A.Lempel
IEEE Trans IT, IT-32, No.3,May 1977,pp337-343

[11] **Compression of Individual Sequences via Variable-Rate Coding**
J.Ziv and A.Lempel
IEEE Trans IT, IT-24, No.5,Sept 1978, pp530-536

[12] **A Technique for High-Performance Data Compression**
T.A.Welch
IEEE Computer, Vol.17, No.6, June 1984, pp8-19

[13] **Text Compression**
T.C.Bell, J.G.Cleary and I.H.Witten
Published by Prentice Hall, Englewood Cliffs,NJ. 1990

[14] **The JPEG Still Picture Compression Standard**
G.K.Wallace
Communications of the ACM, Vol.34, No.4, April 1991, pp31-44

[15] **Fractal Image Compression**
M.F.Barnsley and L.P.Hurd
Published by AK Peters Ltd, 1993, ISBN 1-56881-000-8

[16] **Advances in Digital Image Compression Techniques**
G.Lu
Computer Communications, Vol 16, NO.4, April 1993, pp202-214

[17] **Fractals Transform Image Compression**
A.Wright
Electronics World and Wireless World, March 1992, pp208-211

[18] **Data Compression Software**
R.Milton
Computing , 22 July 1993, pp19-20

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
April 1993

3. REPORT TYPE AND DATES COVERED
Conference Publication, October 19-21, 1993

4. TITLE AND SUBTITLE
Third NASA Goddard Conference on Mass Storage Systems and Technologies

5. FUNDING NUMBERS
505

6. AUTHOR(S)
Benjamin Kobler and P. C. Hariharan, Editors

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER
94B00057

9. SPONSORING / MONITORING ADGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING ADGENCY REPORT NUMBER
NASA CP-3262

11. SUPPLEMENTARY NOTES
Kobler: Goddard Space Flight Center, Greenbelt, Maryland;
Hariharan: Systems Engineering and Security, Inc., Lanham, Maryland.

12a. DISTRIBUTION / AVAILABILITY STATMENT
Unclassified - Unlimited
Subject Category 82

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report contains copies of nearly all of the technical papers and viewgraphs presented at the Goddard Conference on Mass Storage Systems and Technologies held in October 1993. Once again, the conference served as an informational exchange forum for topics primarily relating to the ingestion and management of massive amounts of data and the attendant problems involved. Discussion topics include the necessary use of computers in the solution of today's infinitely complex problems, the need for greatly increased storage densities in both optical and magnetic recording media, currently popular storage media and magnetic media storage risk factors, data archiving standards including a talk on the current status of the IEEE Storage Systems Reference Model (RM). Additional discussion topics addressed System performance, data storage system concepts, communications technologies, data distribution systems, and finally, a talk on data compression and error detection and correction.

14. SUBJECT TERMS
Magnetic tape, magnetic disk, optical disk, mass storage, software storage, digital recording, data compression

15. NUMBER OF PAGES
514

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL