

MULTI-STAGE PATTERN REDUCTION IN LOSSLESS IMAGE  
COMPRESSION

by

Mark Newman

THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

in the


GRADUATE SCHOOL

of

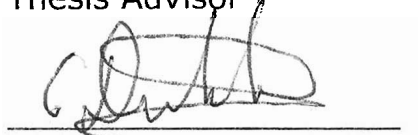
HOOD COLLEGE

May 2006


Accepted

  
W. Randolph Ford, Ph.D.


Thesis Advisor

  
George Dimitoglou, Ph.D.

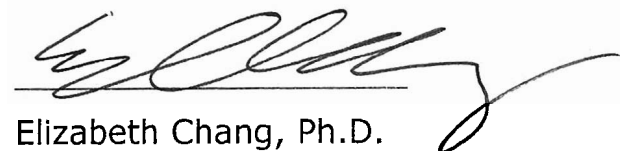
Reading committee

  
Kimber Tysdal, Ph.D.

Reading committee

  
Frank Sweeney, Ph.D.

Dean of the Graduate School

  
Elizabeth Chang, Ph.D.

Chairperson of the Computer  
Science Department

UMI Number: 1439199

Copyright 2007 by  
Newman, Mark

All rights reserved.



---

UMI Microform 1439199

Copyright 2007 by ProQuest Information and Learning Company.  
All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

---

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

### **COPYRIGHT WAIVER**

I authorize Hood College to lend this thesis, or reproductions of it, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

## **ACKNOWLEDGEMENTS**

I would like to thank my reading committee, who let me take the time I needed to take to do my thesis and still responded in a timely fashion.

I would like to thank Talbott Noyes, who was able to show me the difference between summery and summary.

I would like to thank David Gurzick, who was able to show me how to use MS Word to conform to the thesis guide lines without months of tiny edits.

## Table of Contents

Copyright Waiver .....	2
Acknowledgements .....	3
Abstract .....	10
1.0 Introduction.....	11
1.1 Need for Image Compression .....	11
1.2 Two types of Image Compression Algorithms - Lossless vs. Lossy .....	12
1.3 Current Lossless techniques.....	13
1.3.1 Entropy encoding.....	13
1.3.2 Huffman Encoding .....	14
1.3.3 Frequency Domain encoding.....	15
1.3.4 Haar Wavelet.....	16
1.3.5 Runtime Length Encoding.....	18
2.0 Multi-Stage Pattern Reduction .....	18
3.0 Methods and Materials .....	26
3.1 Hardware and software .....	26
3.2 Images .....	27
3.3 Methodology .....	27
3.4 The Algorithms .....	27
3.4.1 Shared Definitions.....	28
3.4.2 Algorithm 1: Philosophy.....	28
3.4.3 Algorithm 1: Pseudo Code.....	30
3.4.4 Algorithm 2: Philosophy.....	32
3.4.5 Algorithm 2: Pseudo Code.....	34
3.5 Algorithm Machinery .....	35
3.5.1 Approach 1-6: Philosophy .....	35
3.5.2 Approach 1-6: Pseudo code.....	37
3.5.2.1 Approach 1 .....	37
3.5.2.2 Approach 2 .....	37
3.5.2.3 Approach 3 .....	37
3.5.2.4 Approach 4 .....	38
3.5.2.5 Approach 5 .....	39

3.5.2.6 Approach 6 .....	39
3.5.3 Approach 7-12: Philosophy .....	40
3.5.4 Approach 7-12: Pseudo code.....	42
3.5.4.1 Approach 7 .....	42
3.5.4.2 Approach 8 .....	42
3.5.4.3 Approach 9 .....	42
3.5.4.4 Approach 10.....	43
3.5.4.5 Approach 11 .....	44
3.5.4.6 Approach 12 .....	44
3.5.5 Approach 13-18: Philosophy.....	45
3.5.6 Approach 13-18: Pseudo code .....	46
3.5.6.1 Approach 13 .....	46
3.5.6.2 Approach 14 .....	47
3.5.6.3 Approach 15 .....	47
3.5.6.4 Approach 16 .....	48
3.5.6.5 Approach 17 .....	49
3.5.6.6 Approach 18 .....	49
3.5.7 Approach 19-24: Philosophy.....	50
3.5.8 Approach 19-24: Pseudo code .....	51
3.5.8.1 Approach 19 .....	51
3.5.8.2 Approach 20 .....	52
3.5.8.3 Approach 21 .....	52
3.5.8.4 Approach 22 .....	53
3.5.8.5 Approach 23 .....	54
3.5.8.6 Approach 24 .....	54
3.5.9 Approach 25-30: Philosophy.....	55
3.5.10 Approach 25-30: Pseudo code .....	56
3.5.10.1 Approach 25.....	56
3.5.10.2 Approach 26.....	57
3.5.10.3 Approach 27 .....	57
3.5.10.4 Approach 28.....	58
3.5.10.5 Approach 29.....	59

3.5.10.6 Approach 30.....	60
3.5.11 Approach 31-36: Philosophy .....	60
3.5.12 Approach 31-36: Pseudo code .....	62
3.5.12.1 Approach 31 .....	62
3.5.12.2 Approach 32.....	62
3.5.12.3 Approach 33.....	63
3.5.12.4 Approach 34.....	64
3.5.12.5 Approach 35.....	65
3.5.12.6 Approach 36.....	65
3.5.13 Approach 37-42: Philosophy .....	66
3.5.14 Approach 37-42: Pseudo code .....	67
3.5.14.1 Approach 37 .....	67
3.5.14.2 Approach 38.....	68
3.5.14.3 Approach 39.....	68
3.5.14.4 Approach 40.....	69
3.5.14.5 Approach 41.....	70
3.5.14.6 Approach 42.....	71
3.5.15 Approach 43-48: Philosophy .....	72
3.5.16 Approach 43-48: Pseudo code .....	73
3.5.16.1 Approach 43.....	73
3.5.16.2 Approach 44.....	74
3.5.16.3 Approach 45.....	74
3.5.16.4 Approach 46.....	76
3.5.16.5 Approach 47.....	77
3.5.16.6 Approach 48.....	78
4.0 Data Summary.....	79
4.1 Comparative Pictorial Summary of Approaches.....	79
4.2 Data Based Summary of Approaches.....	88
5.0 Results and Discussion.....	97
5.1 Approach 1-6.....	97
5.2 Approach 7-12. ....	98
5.3 Approach 13-18. ....	98

5.4 Approach 19-24. ....	98
5.5 Approach 25-30. ....	99
5.6 Approach 31-36. ....	99
5.7 Approach 37-42. ....	100
5.8 Approach 43-48. ....	100
5.9 General Remarks for Future Research.....	100
6.0 Conclusion.....	102
Appendix 1: Code.....	103
Appendix 2: Images.....	104
Appendix 3: Screen Shots.....	108
References .....	110



## Table of Tables

Table 1. Sample of Word Table from Knowledge Base.....	21
Table 2. Sample of Phrase Table from Knowledge Base.....	22
Table 3. Sample of Concept Table from Knowledge Base .....	23
Table 4: Approach 1-6 Data Results .....	89
Table 5: Approach 7-12 Data Results.....	90
Table 6: Approach 13-18 Data Results.....	91
Table 7: Approach 19-24 Data Results.....	92
Table 8: Approach 25-30 Data Results.....	93
Table 9: Approach 31-36 Data Results.....	94
Table 10: Approach 37-42 Data Results .....	95
Table 11: Approach 43-48 Data Results .....	96

## Table of Figures

Figure 1: Approach 1-6 Pictorial Results.....	80
Figure 2: Approach 7-12 Pictorial Results .....	81
Figure 3: Approach 13-18 Pictorial Results .....	82
Figure 4: Approach 19-24 Pictorial Results .....	83
Figure 5: Approach 25-30 Pictorial Results .....	84
Figure 6: Approach 31-36 Pictorial Results .....	85
Figure 7: Approach 37-42 Pictorial Results .....	86
Figure 8: Approach 43-48 Pictorial Results .....	87
Figure 9: Fluorescence .....	104
Figure 10: Lenna .....	105
Figure 11: Aruba1 .....	106
Figure 12: Aruba2 .....	107
Figure 13: Screen Shot.....	109

## **ABSTRACT**

Lossless image compression is the process of compressing and subsequently decompressing images without the loss of data.

Historically, image compression was carried out by treating images as complex text [13]. Only in recent years have images been treated as data collections that could be processed for compression and decompression in a manner unique to images [1]. Even the best modern lossless image compression techniques, however, yield less than desirable results [5]. The biggest drawback for lossless image compression is that images can only be reduced to about one-third of their original image size. Lossy image compression algorithms, i.e., those techniques for compressing image size where image information is lost upon decompression, are capable of reducing images to one-tenth of their actual size with little or no humanly perceptual loss in image detail.

Multi-stage pattern reduction is an emerging approach for encoding data that has recently demonstrated efficient processing in the field of natural-language processing. It relies on the ability to discern small local patterns in a source, recreating a new source using these local patterns and then reapplying the technique over multiple stages.

In this thesis, the value of using multi-stage pattern reduction to compress images will be explored. The goal of this thesis is to create

a lossless image compression algorithm by employing the techniques of multi-stage pattern reduction and to determine if such an approach can provide better compression on average than the current major competing algorithms in the field.

## **1.0 INTRODUCTION**

### **1.1 NEED FOR IMAGE COMPRESSION**

As computers become faster and technological influences increase, the demand for multimedia, and more specifically visual images, becomes greater. Concordantly, the availability and use of high quality images on the Internet is increasing at a rate far outpacing the ability for users to retrieve such images in a reasonable amount of time [3, 7]. Therefore, the necessity for highly efficient image compression takes on greater importance in current technological use. Image transmitting accounts for some of the highest demands on Internet resources. It is noteworthy that one of the major shifts in Internet access is toward broadband use for many users accessing the Internet from their homes. Any advances in increasing image compression efficiency would result in reducing load on Internet traffic and improving the user experience.

## **1.2 TWO TYPES OF IMAGE COMPRESSION ALGORITHMS - LOSSLESS VS. LOSSY**

Lossless compression can be defined as the ability to uniquely retrieve a series of bits from a source image from which one can exactly duplicate the pre-compressed source image. Lossy compression can be defined as the ability to retrieve a series of bits from a compressed source that is close to, but not necessarily identical to, the pre-compressed source. If lossless compression can yield bit for bit equality then why is lossy compression even acceptable? Lossy compression is almost always more efficient, that is to say computationally faster and resulting in smaller output file, i.e. more compressed, than lossless compression. Often lossy image compression is visually indistinguishable from the original source. However, this is not always the case. In fact, for some applications, even a small amount of degradation is absolutely intolerable, such as in medical imaging. Additionally, some compression algorithms get progressively worse over many successive generations [2], where a generation is defined as one compression and decompression cycle of an image. The focus of this paper is on lossless algorithms.

### **1.3 CURRENT LOSSLESS TECHNIQUES**

Currently there are 3 categories of lossless image compression techniques. They are:

1. Entropy encoding.
2. Frequency Domain encoding.
3. Runtime length encoding (RLE)

A brief overview of each technique and its main strengths and weaknesses is given below along with a prominent algorithm from each category.

#### **1.3.1 ENTROPY ENCODING**

Entropy encoding is the process of translating all current symbols in a given source into new symbols such that the combined effects of new symbols represent an overall length reduction for the entire source.

Entropy encoding is very efficient at encoding sources that have distinct or well known frequencies, such as English language text, or show a notable frequency increase in a few symbols as compared to other symbols. Entropy encoding does not work in cases that have symbols that display very similar frequencies, such as white noise, or in cases that there are so many symbols that the translation mechanism, most likely a fully complete table, grows faster than the length reduction of valuable symbols decreases the source size.

There are two main entropy encoding algorithms: Huffman encoding and arithmetic encoding. Huffman encoding can be found in most entry level algorithm books and will be described below for completeness [6]. Arithmetic encoding, on the other hand, is patented by IBM and therefore it is left to the interested reader to delve if so desired [11].

### **1.3.2 HUFFMAN ENCODING**

The Huffman encoding algorithm is an  $O(N\log N + 2L)$  algorithm where  $N$  is the symbol count and  $L$  is the file length. Generally, this means that  $N$  is  $2^8$  or one byte. However, the algorithm is valid on any number of symbols, but the translation table tends to grow faster than the algorithm reduces the source length.

Huffman encoding first finds the frequency of all the symbols in the source. Next, the frequencies are sorted into a list in ascending order. Next, the first two entries are combined into a single node with left and right children, element one and element two respectively, whose new frequency is the combined frequency of the first two elements. The first two elements are then deleted and a new node is inserted into the still sorted list in its appropriate place determined by the sum of its left and right children's frequencies. This node is treated as any other element in the list. This process of removing two elements and replacing them as one is continued until only one element is left. The

resulting data structure is a tree. Bit string values are assigned to the leaf nodes based on their left or right path from the root. These leaf nodes are the literal symbols. The result is a translation table. A second pass is made through the file, replacing the actual byte read in from the source file with the bit string found on the translation table. Finally, the translation table is stored. Decompression is as simple as retrieving and reversing the translation table, thus restoring the original source.

### **1.3.3 FREQUENCY DOMAIN ENCODING**

Frequency Domain encoding uses a variety of mathematical techniques to reduce the source into its component form. These component pieces are generally easier to describe than the composite source and therefore take up less space, thus, achieving compression. Frequency Domain encoding works best on sources that have an underlying frequency pattern. This is difficult, however, to discern from just a cursory examination of the source. Frequency Domain encoding also has the extremely useful quality of multiple resolution. This is to say that, if less than lossless quality is desired then the computation needed to derive the lossless data produces the lossy result along the way. This is outside the purview of this thesis, but is mentioned for completeness. Frequency Domain encoding is not suited for



compression of a source that includes desirable high frequency data, such as in the case of steganography, where the noise of a image is the real desired information.

There are three main frequency domain encoding techniques: Discrete Cosine Transformation (DCT), Fast Fourier Transformation (FFT), and wavelets. In the case of real-valued sources, DCT and FFT are mathematically related [9]. Their use in many sciences is well documented and easily obtainable. Wavelets are recent additions to frequency domain encoding and come in many forms. One of these, the Haar Wavelet, will be described below. It is the easiest to understand and allows for a better grasp of more advanced wavelet concepts. The new JPEG-2000 compression uses these more advanced wavelets as part of its overall algorithm.

#### **1.3.4 HAAR WAVELET**

Wavelet compression, and in particular Haar wavelet as described below, is a group of recursive functions designed to break down a series of numerical values into multiple bands describing aggregate and detail information about the series. This allows for examination, or compression, based on overriding features of the series of local phenomena [12].

The Haar wavelet is an averaging wavelet at its base. This is to say that when you have 2 values next to each other they are averaged then the remainder is determined, i.e.  $\{6 \ 12\} \rightarrow (6+12)/2 = 9, 9-12 = -3 \rightarrow \{9, -3\}$ . The Haar wavelet provides a perfectly reversible map from Integers to Reals. Very few point this out in their documentation. The issue is that computers are geared to handle integers far better than Reals, so some of the power of the Haar wavelet is lost. A better example of what happens should be  $\{5 \ 12\} \rightarrow \text{floor}((5+12)/2) = 8, 5-12 = -7 \rightarrow \{8, -7\}$ . Using these and a small amount of simple logic we can convert from Integers to Integers. Sadly, much of the converging properties of wavelets are lost. Having discussed the problems, we can move on to the recursion aspect of wavelets. Knowing, as we do, how to manipulate the base wavelet we need to realize that recursion is intended to pull from the aggregate values only. This is done with simple reordering. If we have the original sequence  $\{2 \ 4 \ 8 \ 4 \ 10 \ 6 \ 7 \ 9\}$  performing The Haar wavelet on each of the pairs yields  $\{3 \ -1 \ 6 \ 2 \ 8 \ 2 \ 8 \ -1\}$ , reordering to pull the aggregate data to the front of the list yields  $\{3 \ 6 \ 8 \ 8 \ -1 \ 2 \ 2 \ -1\}$  the Haar wavelet can then be performed on the sub list  $\{3 \ 6 \ 8 \ 8\}$  and so on in recursive manner. Notice how I chose the more naive Haar approach and chose my data to be friendly. Specifically, note the choice of 8 elements over 7 or 9 elements.

### **1.3.5 RUNTIME LENGTH ENCODING**

Run length encoding is nearly ineffective on color images. Therefore it will not be discussed in this thesis in any detail and is mentioned only for completeness [10]. Run length encoding is used primarily as an observed optimization for the bitmap format and in fax machines.

## **2.0 MULTI-STAGE PATTERN REDUCTION**

Current lossless image compression techniques tend to achieve, on average, on the order of 1.5 to 1 to 3.0 to 1 compression ratios [8], and are often quite complex, both to code and to understand. The primary goal of this thesis is to create a compression scheme that will achieve greater compression ratios on average than the above techniques.

Multi-stage pattern reduction has recently demonstrated efficient processing in natural-language processing by limiting all possible search spaces [4]. It relies on the ability to discern small local patterns in a source, recreating a new source using these local patterns and then reapplying the technique. Since this is also a goal of image compression, the employment of this technique is a natural extension of this work. The application of this technique to natural language processing is described in detail below.

Using multi-stage pattern reduction, there are two different parts to the processor. One is the knowledge base, which is made up of three tables, and the other is the actual algorithm used for navigating the knowledge base to reduce a user's input down to a concept with parameters. Simply put, that is the goal of any natural language processor, reducing the user's input down to a concept. In the case of all the different ways that we suggested one could come up with to say, *"Throw me the ball."* In natural language processing, we are now working in the opposite direction where we are looking for a way to take all of the different ways on the list of expressing this concept (these are called surface structures) and reducing them down to a single concept, *"throw"*, with a directionality of *"to me"*, with the parameter of *"ball"* (this is called the deep structure).

For the purpose of understanding the processor, let us take the example phrase:

*"I would like to get my 401k account balance"*

On the knowledge-base side of our process we have three tables. The first table in the knowledge base contains the words and semantic symbols for those words. For example, you will notice that the first three entries all share the same semantic symbol. Likewise, *"need"* and *"want"* also share the same semantic symbol of 024. This

relationship between these two columns allows the processor to perform the first step in semantic reduction. The individual patterns, in the form of words, are identified and then reduced semantically.

In our first step we take the words in the input and arrange them into a vector.

I	Would	like	to	Get	my	401k	account	Balance
---	-------	------	----	-----	----	------	---------	---------

We then go to the first table and look up each word in the table.

WORD	SEMANTIC SYMBOL
Can	009
Could	009
Would	009
Will	010
Shall	010
Got	011
Get	012
Find	012
Obtain	012
Acquire	012
Retrieve	012
I	019
We	019
Need	024
Want	024
To	031

Do	034
Like	143
I'd	257
I've	258
My	038
401k	803
Account	695
Balance	217

**TABLE 1. SAMPLE OF WORD TABLE FROM KNOWLEDGE BASE.**

From this table we retrieve the semantic symbol for each word yielding an array such as:

I	Would	like	to	get	my	401k	account	Balance
019	009	143	031	012	038	803	695	217

This could also be expressed as a new vector of: 019 009 143 031 012 038 803 695 217. This represents the semantic reduction over words, or the first pattern reduction step in our algorithm. It is important to note that this single vector in some cases, where there are multiple words representing each semantic value, might represent twenty or more different ways to express this same deep structure. In the next step we take our new vector and begin to search through the phrase knowledge-base table from left to right for the largest represented phrase on the table.

PHRASE	SEMANTIC SYMBOL
019 010 024 031	a002
019 024 031	a002
019 013 011 031	a002
019 009 024 031	a002
019 010 013 031	a002
034 019 024 031	a002
258 011 031	a002
257 024 031	a002
019 009 143 031	a002
257 143 031	a002
012	f110
038 803 695 217	m117

**TABLE 2. SAMPLE OF PHRASE TABLE FROM KNOWLEDGE BASE**

For example in the first pass we search for a match for the whole vector, and finding no match we search for a match for the whole vector minus the last semantic representation: 019 009 143 031 012 038, and finding no match we search for a match for the whole vector minus the last two semantic representations: 019 009 143 031 012, and we repeat this process until we finally a match for 019 009 143 031 and we take the semantic symbol a002 and save it in our phrase vector. In searching for the next largest chunk we find that 012 yields f110, and finally 038 803 695 217 yields m117, creating a new vector for us of a002 f110 m117. The m117 phrase is a special type of

phrase in that it represents an object, i.e. *"401 k account balance"* It is interesting to note that it is also representing several other ways of saying this such as *"the 401k balance"*, or *"the balance in my 401k account"*.

From this table we retrieve the semantic symbol for each phrase yielding this array:

I	would	like	to	get	my	401k	account	Balance
019	009	143	031	012	038	803	695	217
A002	a002	a002	a002	f110	m117	m117	m117	m117

On average, after having gone through two stages of reduction, most objects on the list such as m117 represent about six unique ways of referral. The current demonstration application contains 46 such objects or 276 (46 x 6) different ways of saying objects referenced in the current application. This will become an important point later on. So, we are now left with the phrase vector a002 f110 and the identified object m117.

CONCEPT	SEMANTIC SYMBOL
A002 f110 m*	AA
A002 d004 f012 c001	AA
A002 f010 b001 f130	AA

**TABLE 3. SAMPLE OF CONCEPT TABLE FROM KNOWLEDGE BASE**



We go to our Concept knowledge base and find a matching value for it of AA. AA represents the deep structure for retrieval. We pass this instruction (AA) along with the parameter "*m117*" along to our application and it returns the balance in our 401k account. All this might seem a bit mundane, until we look at the last step where we matched a002 f110 of the concept table work and backwards through our tables and discover that this one line represents over three hundred difference ways that a user might ask for his/her 401k account balance. Multiply this by the six ways the "*401k balance*" is represented and we have accommodated 1800 different ways a user might ask for a specific piece of information with a single table entry and a complementary object. Crossed with all object in the current demonstration application and we are looking at 82800 different ways of asking for 46 unique pieces of information from one table entry.

From this table we retrieve the semantic symbol for each phrase yielding this array:

I	would	like	to	get	my	401k	account	Balance
019	009	143	031	012	038	803	695	217
A002	a002	a002	a002	f110	m117	m117	m117	m117
AA	AA	AA	AA	AA	AA	AA	AA	AA

There are in fact 43 different table entries in the concept knowledgebase for AA the retrieval operation. If we assume that they

represent just as many surface phrases (and in fact some of them represent hundreds of thousands), then we can conservatively say that our language processor can understand 3,560,400 different ways to ask for 46 unique objects, all from a database with fewer than 1500 table entries across words, phrases, and concepts. Of course, there is more than one deep structure in the concept table. There are in fact, thirty-one. In total, 1500 table entries that represent over 100 million user inputs, in 1.2 megabytes of space.

In the end, the algorithm as a whole may appear obvious or trivial. This, however, would be a mistaken conclusion. While no grammatical rules are visible, and no commonly seen statistical tables are exposed, what our process has effectively done is map the pathways of legitimate grammar through all of the possible combinations of words used in a specific context. The specific rules required to create the combinations of words in the form of phrases and combinations of phrases to form a concept are inherently present in the employment of the tables and are knowledge base. That is to say, these tables are the end result of what one would find after employing the grammatical rules that might be used in a parsing system under a specific application context. But the multistage pattern reduction process requires significantly less effort in all areas when compared to parsing.

The power of our algorithm is most obvious in the 300 recognized inputs yielded from a total of 31 word table entries in figure 4. Figure 4 contains both five word and four word combinations. There are 211,376 possible combinations of five word and four word phrases when drawn from a table of 31 possible words. Figure 4 represents the 300 legitimate combinations for a single table entry in the concept table. In a sense, we have reduced the search space for all combinations of words down to the legitimate representation of a grammar. In this manner, the multi-stage pattern reduction algorithm represents the best features of other approaches, i.e., parsing type natural language processors and keyword/key-phrase processors without the drawbacks. The grammatical rules required for understanding in a parsing processor are inherently present, along with the power of recognition of variability of expression inherent in keyword/key-phrase processors.

### **3.0 METHODS AND MATERIALS**

#### **3.1 HARDWARE AND SOFTWARE**

The algorithms were implemented in C# .net version 1.1, a software language developed by Microsoft. The development environment is Microsoft's Visual Studio .net 2003. The development machine used was a Dell XPS laptop running Microsoft Windows XP sp2

### **3.2 IMAGES**

The algorithm was tested on a variety of pictures. Images were standard format of 8 bits per pixel. All pictures appear in the appendix with individual citation captions.

### **3.3 METHODOLOGY**

All images began as uncompressed 256 color bitmap files. They are each then compressed into the JPG lossless compression format, Huffman compression format, and the particular pattern reduction format being tested and the compression ratios are noted. Finally, each was uncompressed and checked against the original image to verify true lossless image compression.

### **3.4 THE ALGORITHMS**

The heart of this research is fundamentally to test the value of a new image compression algorithm using recursive pattern reduction.

However, like many image compression techniques and practical algorithms in general, there is a variety of machinery surrounding the base algorithm itself. To this end I have broken out the algorithms and their accompanying machinery into separate sections. Each section follows with an accompanying description explaining the general

philosophy behind the algorithm or machinery. Finally, pseudo code is presented as an example.

### **3.4.1 SHARED DEFINITIONS**

Image: An image is a 2 dimensional  $n \times m$  array consisting of color information. Each element of the array represents a unique pixel whose value is the color at the given point. Image indexing is left to right, top to bottom starting at (1, 1).

Stream: A stream is a list whose elements can range from zero and up. Streams tend to have information deleted from them. Indexing is from left to right, starting at 1.

Table: A table is a list whose elements are ordered pairs. Tables are generally searched and appended to. Indexing is from left to right, starting at 1.

### **3.4.2 ALGORITHM 1: PHILOSOPHY**

The fundamental idea in recursive pattern recognition is that given a series of symbols, the arrangements of those symbols is the actual important piece of information as opposed to the importance of the symbols themselves. The end results of the process yields a “bubbling up” of valuable information and a virtual elimination of patterns that never, or at the least extremely infrequently, occur. To this end, a

static method was developed to take in pairs of pixels and view them as a single unit instead of 2 distinct units. This was done in hopes that there would be fewer new pixel units that actually occurred than the strict combination of pixel pairs. Then these new pixel units could be reefed recursively through the same algorithm to produce new units of information.

To do this we flatten the image to a single stream of data, selecting double pixel pairs. If we have seen the current pair we replace it with a new pair. If not we replace it with a new symbol and place the pair in a dictionary. We then apply this simple technique recursively on the new encoded stream. Clearly, when applied recursively, this could yield an end single, but this is just a minor annoyance, easily fixed with clever coding. The dictionary, which is clearly unique for each unique image, is then pre-pended to the front of the final recursively encoded stream. Decoding the stream is simply a dictionary lookup applied several times.

The real strength of this approach is that if a particular combination of pixels is never next to each other their combination will never appear and so need not be represented. This allows all the valid combinations to be stored without storing any of the useless patterns.

### 3.4.3 ALGORITHM 1: PSEUDO CODE

Please refer to the companion CD for the C# reference implementation.

Step 1: Label the image stream0. Select a number greater than one for the number of reduction steps labeled reduction1.

Step 2: Flatten the two-dimensional image source into a stream of pixel information. Create an empty stream labeled stream1. Take the pixel information from stream0 left to right, top to bottom and append it to the back of stream1.

Note: Because stream0 is a rectangular  $n*m$  matrix it must have an even number of elements, therefore stream1 will have an even number of elements.

Step 3: Create an empty table labeled table1. Create two empty streams labeled count1 and pattern1.

Step 4: Look at the first pair of elements of stream1. Create an ordered pair from those values labeled orderedpair1. If they are in table1 then go to step 5. If not then append orderedpair1 to table1.

Step 5: Count out of stream1 ordered pairs starting at the beginning of the stream, until an ordered pair doesn't have the value as orderedpair1. Delete those ordered pairs from stream1. Append the

count to count1. Append the index value of ordered pair1 to pattern1.

If stream1 is empty, then go to step 6. Otherwise, go to step 4.

Note: Count1 and pattern0 have the same number of elements. The maximum value of pattern1 is the size of table1. The sum of the elements of count1 is equal to the size of stream0.

Step6: Create an empty table labeled tableX where X is 1+ the current highest table number. Create an empty stream labeled patternX.

Step 7: Look at the first pair of elements in pattern(X-1). Create an ordered pair labeled orderedpair1. If orderedpair1 is in tableX, append its index value to patternX. If orderedpair1 is not in tableX, then append it to tableX and append add its index value to patternX.

Delete orderedpair1 from pattern(X-1).

Step 8: If pattern(X-1) is empty, then go to step 9, otherwise, go to step 7.

Step 9: If the X is equal to reduction1, then go to step 10. Otherwise, goto step 6.

Step 10: Create an empty table labeled counttableX where X is 1+ the current highest counttable number. Create an empty stream labeled countX.



Note: CounttableX starts at X equals 1 on step 10 whereas tableX starts at X equals 2 on step 6 since table1 is used on step 1.

Note: Steps 6 through 9 could be preformed in parallel to steps 10 through 13.

Step 11: Look at the first pair of elements in count(X-1). Create an ordered pair labeled orderedpair1. If orderedpair1 is in counttableX, append its index value to countX. If orderedpair1 is not in counttableX, then append it to counttableX and append add its index value to countX. Delete orderedpair1 from count(X-1).

Step 12: If count(X-1) is empty, then go to step 12, otherwise, goto step 7.

Step 13: If the X is equal to reduction1, then go to step 14.

Otherwise, go to step 10.

Step 14: Done.

#### **3.4.4 ALGORITHM 2: PHILOSOPHY**

This approach is very similar to algorithm 1 but takes features into consideration found in Huffman encoding as well. This approach differs from the first fundamentally in that it is a dynamic solution instead of a static one. This allows for a better gauge of what constitutes a worthwhile pattern. This also lets us ignore the troublesome unit one

occurrence pattern that exists only once in a source image. By its very nature, algorithm 1 must preserve these anomalies, whereas algorithm 2 allows us to skip the anomaly and continue on to worthwhile patterns.

To do this we flatten the image to a single stream of data. We then run a frequency count on the all pairs. There is a problem with pair overlap in frequency counts, but this is solved easily as long as you know to look for it. The highest frequency pattern of pixel pairs is viewed as the most worthwhile pattern and encoded to a new value, replacing all occurrences of that pair with the new value. This process is then applied several times using the pixel values and encoded values as source. The dictionary, which is clearly unique for each individual image, is then pre-pended to the front of the final recursively encoded stream. Decoding the stream is simply a dictionary lookup applied in order several times.

The strength of this algorithm, like algorithm 1, is that if a particular combination of pixels is never next to each other their combination will never appear and so need not be represented. Unlike algorithm 1, however, you need not represent existing combinations that add little information. You may instead leave them as individual pixels and not as encoded values.

### 3.4.5 ALGORITHM 2: PSEUDO CODE

Please refer to the companion CD for the C# reference implementation.

Step 1: Label the image stream0. Find the maximum value of stream0 and label it max1.

Step2: Select a number that is a power of 2  $-1$  and is greater than max1 and label this number stop1.

Step 3: Flatten the two-dimensional image source into a stream of pixel information. Create an empty stream labeled stream1. Take the pixel information from stream0 left to right, top to bottom and append it to the back of stream1.

Step 4: Examine stream1 for the pair of pixels with the greatest frequency. Label this pair symbol1.

Note: Pair overlap can occur and prevent an accurate count of frequencies. Example in the stream  $\{1,1,1,2\}$  the symbol  $\{1,1\}$  occurs two times, but could only be replaced once. Therefore, it should only be counted once.

Step 5: Search the stream for all occurrences of symbol1 and replace it with the value of  $\text{max1} + n$ , where  $n$  is the number of times step 4 has been completed.

Step 6: If the value of  $\text{max1} + n$  is equal to  $\text{stop1}$  then stop.

Otherwise, go to step 4.

Step 7: Done.

### **3.5 ALGORITHM MACHINERY**

Below are the different 48 different approaches used in conjunction with the two algorithms to determine fitness. As described above these are merely the surrounding equipment used to properly input, format, or manipulate in some way the codes going into the main algorithms. As above, they will be provided with a philosophy section as well as a pseudo code section. However, unlike above they are partitioned into logical groupings each sharing a common theme. This is done to reduce needless representation of identical thoughts, nothing more.

#### **3.5.1 APPROACH 1-6: PHILOSOPHY**

These approaches are the most basic and naive. It attempts to directly convert the image into a pattern reduction step, possibly feeding the reduction step into a Huffman compressor. Even though this process is horribly naïve it is an important step in the thesis research. It provides a baseline against which I can compare my

results and also provides a foundation on which I can expand on to come up with new surrounding processes.

Approach 1 works as a strict base line. Get the picture information in as quickly as possible in the most undiluted form and start the algorithms. Algorithm 2 expands this a little by Huffman encoding the resulting values. This is nothing special, but it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. It is important to note that Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this, however, since we are dealing with unique sources. Approach 3 manipulates JPG compression into giving us just the difference between lossless and lossy compression to feed the algorithms.

Approach 4, like approach 2, adds Huffman compression to the result of approach 3. Approach 5 attempts to derive pictorially linked units (PLUs) from the source by claiming all values that are close should really be considered the same value, synonyms for pixels as it were, then reverts back to approach 1. Approach 6 follows the theory of approaches 2 and 4 by adding Huffman compression to approach 5.

### **3.5.2 APPROACH 1-6: PSEUDO CODE**

#### **3.5.2.1 APPROACH 1**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, encode the values using one of the algorithms.

Step 4: Save the three compressed streams.

#### **3.5.2.2 APPROACH 2**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, encode the values using one of the algorithms.

Step 4: For each compressed stream, Huffman encode the values.

Step 5: Save the three double compressed streams.

#### **3.5.2.3 APPROACH 3**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: For each difference plane, encode the values using one of the algorithms.

Step 7: Save the compressed difference streams

Step 8: Save the JPG image as raw JPG data.

#### **3.5.2.4 APPROACH 4**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: For each difference plane, encode the values using one of the algorithms.

Step 7: For each compressed stream, Huffman encode the values.

Step 8: Save the double compressed difference streams

Step 9: Save the JPG image as raw JPG data.

#### **3.5.2.5 APPROACH 5**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: For each of the planes encode the values using one of the algorithms.

Step 5: Save the compressed streams.

#### **3.5.2.6 APPROACH 6**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.



Step 4: For each of the planes encode the values using one of the algorithms.

Step 5: For each compressed stream, Huffman encode the values.

Step 6: Save the double compressed streams

### **3.5.3 APPROACH 7-12: PHILOSOPHY**

These approaches are a continuation of the naiveté above. It attempts to directly convert the image into a pattern reduction step, possibly feeding the reduction step into a Huffman compressor. These six processes differ from above in that they change the color space from RGB to YCrCb (color space used by televisions). Even though this seems to be a trivial modification to the algorithms some research suggests that it improves compression [3, 10] by provably reducing the interdependence of the pixels. This can also be seen, pictorially, in the comparison between approaches 1-6 vs. 7-12, while not all achieve better compression, most do. Like the above it too provides a baseline against which I can compare my results and also provides a foundation on which I can expand on to come up with new surrounding processes.

Approach 7 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form, then convert it to the new color space and start the algorithms. Algorithm 8 expands

this a little by Huffman encoding the resulting values. This is nothing special, but it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. Although, Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this, however, since we are dealing with unique sources.

Approach 9 manipulates JPG compression into giving us just the difference between lossless and lossy formats in the new color space. This is done by first calculating the lossy jpg then converting it to the new color space. Then the lossless original is converted to the new color space. Then the difference is taken. This leaves us with a color space difference which can then be processed by the algorithms.

Approach 10, like approach 8, adds Huffman compression to the result of approach 9. Approach 11 attempts to derive pictorially linked units (PLUs) from the source by claming all values that are close should really be considered the same value, synonyms for pixels as it were, then reverts back to approach 7. Approach 12 follows approaches 8 and 10 in concept by adding Huffman compression to approach 11.

### **3.5.4 APPROACH 7-12: PSEUDO CODE**

#### **3.5.4.1 APPROACH 7**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: For each plane, encode the values using one of the algorithms.

Step 4: Save the three compressed streams.

#### **3.5.4.2 APPROACH 8**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: For each plane, encode the values using one of the algorithms.

Step 4: For each compressed stream, Huffman encode the values.

Step 5: Save the three double compressed streams.

#### **3.5.4.3 APPROACH 9**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the YCrCb stream.

Step 6: For each difference plane, encode the values using one of the algorithms.

Step 7: Save the compressed difference streams

Step 8: Save the JPG image as raw JPG data.

#### **3.5.4.4 APPROACH 10**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the YCrCb stream.

Step 6: For each difference plane, encode the values using one of the algorithms.

Step 7: For each compressed stream, Huffman encode the values.

Step 8: Save the double compressed difference streams

Step 9: Save the JPG image as raw JPG data.

#### **3.5.4.5 APPROACH 11**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: For each of the planes encode the values using one of the algorithms.

Step 5: Save the compressed streams.

#### **3.5.4.6 APPROACH 12**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: For each of the planes encode the values using one of the algorithms.

Step 5: For each compressed stream, Huffman encode the values.

Step 6: Save the double compressed streams

### **3.5.5 APPROACH 13-18: PHILOSOPHY**

These approaches are the beginning of the true research.

The above were primarily baselines; these are intended to change the underlying data into a more compatible form to accommodate a potential recursive step. It occurred to me that the simplest way to accomplish this was to change the base of the underlying data. To this end I converted the original base 256 values to base 2 values by simply converting the single byte (8bits) into eight one bit units. I have tried other combinations but omitted them from the research since they also proved fruitless.

Approach 13 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form, convert it to the base2 format, and start the algorithms. Algorithm 14 expands this a little by Huffman encoding the resulting values. While this is not unique, it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen

reduction. Conversely, Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this, since we are dealing with unique sources. Approach 15 manipulates JPG compression into giving us just the difference between lossless and lossy formats. This result is then converted to the new base2 format, then processed by the algorithms. Approach 16, like approach 14, adds Huffman compression to the result of approach 15. Approach 17 attempts to derive pictorial linked units (PLUs) from the source by claiming all values that are close should really be considered the same value, synonyms for pixels as it were, then reverts back to approach 13. Approach 18 follows approaches 14 and 16 in adding Huffman compression to approach 17.

### **3.5.6 APPROACH 13-18: PSEUDO CODE**

#### **3.5.6.1 APPROACH 13**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 4: For each plane, encode the values using one of the algorithms.

Step 5: Save the three compressed streams.

#### **3.5.6.2 APPROACH 14**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 4: For each plane, encode the values using one of the algorithms.

Step 5: For each compressed stream, Huffman encode the values.

Step 6: Save the three double compressed streams.

#### **3.5.6.3 APPROACH 15**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: Re-Code the difference planes from  $\text{base}2^8$  to  $\text{base}2^1$ .



Step 7: For each difference plane, encode the values using one of the algorithms.

Step 8: Save the compressed difference streams

Step 9: Save the JPG image as raw JPG data.

#### **3.5.6.4 APPROACH 16**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: Re-Code the difference planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 7: For each difference plane, encode the values using one of the algorithms.

Step 8: For each compressed stream, Huffman encode the values.

Step 9: Save the double compressed difference streams

Step 10: Save the JPG image as raw JPG data.

#### **3.5.6.5 APPROACH 17**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 5: For each of the planes encode the values using one of the algorithms.

Step 6: Save the compressed streams.

#### **3.5.6.6 APPROACH 18**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 5: For each of the planes encode the values using one of the algorithms.

Step 6: For each compressed stream, Huffman encode the values.

Step 7: Save the double compressed streams.

### **3.5.7 APPROACH 19-24: PHILOSOPHY**

These approaches are similar in scope and intent to approaches 7-12. They are intended to show the difference that color space makes to the algorithms. This might seem redundant since we already established numbers for changing the base as well as changing the color space. While I admit this was done primarily for completeness, the results are significantly different between 1-6 and 7-12 vs. 13-18 and 19-24.

Approach 19 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form, convert it to the new color space, then convert it to the base2 format, and then start the algorithms after all the conversions are complete. Algorithm 20 expands this a little by Huffman encoding the resulting values. This is nothing special, but it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. Conversely, Huffman encoding can yield strange results when you include the dictionary as part of the size

consideration. We must do this since we are dealing with unique sources. Approach 21 manipulates JPG compression into giving us just the difference between lossless and lossy formats in the new color space. This is done by first calculating the lossy jpg then converting it to the new color space. Then the lossless original is converted to the new color space. Then the difference is taken. This leaves us with a color space difference which can then be converted to base2 format and finally processed by the algorithms. Approach 22, like approach 20, adds Huffman compression to the result of approach 21. Approach 23 attempts to derive PLUs from the source by claiming all values that are close should really be considered the same value, synonyms for pixels as it were, then reverts back to approach 19. Approach 24 follows approaches 20 and 22 in adding Huffman compression to approach 23.

### **3.5.8 APPROACH 19-24: PSEUDO CODE**

#### **3.5.8.1 APPROACH 19**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 4: For each plane, encode the values using one of the algorithms.

Step 5: Save the three compressed streams.

#### **3.5.8.2 APPROACH 20**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 4: For each plane, encode the values using one of the algorithms.

Step 5: For each compressed stream, Huffman encode the values.

Step 6: Save the three double compressed streams.

#### **3.5.8.3 APPROACH 21**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG YCrCb stream and the YCrCb stream.

Step 6: Re-Code the difference planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 7: For each difference plane, encode the values using one of the algorithms.

Step 8: Save the compressed difference streams.

Step 9: Save the JPG image as raw JPG data.

#### **3.5.8.4 APPROACH 22**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG YCrCb stream and the YCrCb stream.

Step 6: Re-Code the difference planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 7: For each difference plane, encode the values using one of the algorithms.

Step 8: For each compressed stream, Huffman encode the values.

Step 9: Save the double compressed difference streams.

Step 10: Save the JPG image as raw JPG data.

#### **3.5.8.5 APPROACH 23**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 5: For each of the planes encode the values using one of the algorithms.

Step 6: Save the compressed streams.

#### **3.5.8.6 APPROACH 24**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: Re-Code the color planes from  $\text{base}2^8$  to  $\text{base}2^1$ .

Step 5: For each of the planes encode the values using one of the algorithms.

Step 6: For each compressed stream, Huffman encode the values.

Step 7: Save the double compressed streams

### **3.5.9 APPROACH 25-30: PHILOSOPHY**

These approaches continue the true research. As above, we have attempted to force the underlying information to reveal itself so that we might compress it. These approaches continue on the assumption that a different underlying structure may provide a better representation. Where approach 13-18 and 19-24 use a simple modification to the underlying data, approach 25-30 moves to a very different structure in the form of Huffman compression. While both converting a single byte to eight one bit values and Huffman compression yield bit streams, they are vastly different.

Approach 25 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form. Immediately run the Huffman algorithm to produce a new sequence. Finally, run the encoding algorithms. Algorithm 26 expands this a little by Huffman encoding the resulting values. This is nothing special, but it is



interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. Conversely, it is important to note that Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this since we are dealing with unique sources. I am aware that compressing a Huffman stream is considered pointless, but we have altered the intermediate values, so I believe you should run the experiment since it really costs nothing. Approach 27 manipulates JPG compression into giving us just the difference between lossless and lossy compression to feed the algorithms. Approach 28, like approach 26, adds Huffman compression to the result of approach 27. Approach 29 attempts to derive PLUs from the source then reverts back to approach 25. Approach 30 follows approaches 26 and 28 in adding Huffman compression to approach 29.

### **3.5.10 APPROACH 25-30: PSEUDO CODE**

#### **3.5.10.1 APPROACH 25**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, compress the values using Huffman.

Step 4: For each stream, encode the values using one of the algorithms.

Step 5: Save the three double compressed streams.

### **3.5.10.2 APPROACH 26**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, compress the values using Huffman.

Step 4: For each stream, encode the values using one of the algorithms.

Step 5: For each double compressed stream, Huffman encode the values.

Step 6: Save the three triple compressed streams.

### **3.5.10.3 APPROACH 27**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: For each difference plane, compress the values using Huffman.

Step 7: For each stream, encode the values using one of the algorithms.

Step 8: Save the double compressed streams.

Step 9: Save the JPG image as raw JPG data.

#### **3.5.10.4 APPROACH 28**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 6: For each difference plane, compress the values using Huffman

Step 7: For each stream, encode the values using one of the algorithms.

Step 8: For each double compressed stream, Huffman encode the values.

Step 9: Save the triple compressed difference streams.

Step 10: Save the JPG image as raw JPG data.

#### **3.5.10.5 APPROACH 29**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image.

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: For each plane, compress the values using Huffman.

Step 5: For each of the streams, encode the values using one of the algorithms.

Step 6: Save the double compressed streams.

### **3.5.10.6 APPROACH 30**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image.

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 4: For each plane, compress the values using Huffman.

Step 4: For each of the streams, encode the values using one of the algorithms.

Step 5: For each double compressed stream, Huffman encode the values.

Step 6: Save the triple compressed streams

### **3.5.11 APPROACH 31-36: PHILOSOPHY**

The remainder of the approaches starting with approaches 31-36 move into the realm of modern image compression. By this I mean the use of wavelets, namely the Haar wavelet, to compress images. It is important to note that when I refer to the Haar wavelet I do not mean the true Haar wavelet but a modified version. While it is known that the Haar wavelet is uniquely reversible, a necessary quality in lossless compression, it converts four bytes ( $4 \times 2^8$ ) to a single byte ( $1 \times 2^8$ )

and three floats (these can fit in  $3 \times 2^9$ ). This is, on its face, a bad idea when it comes to compression to first enlarge your space then try to reduce past it. To this end I converted the Haar wavelet to one that functions similarly to the original but producing slightly different numeric results.

Approach 31 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form by converting it using the modified Haar wavelet, and then starting the algorithms.

Algorithm 32 expands this a little by Huffman encoding the resulting values. This is nothing special, but it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. Conversely, Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this since we are dealing with unique sources. Approach 33 manipulates JPG compression into giving us just the difference between lossless and lossy compression. This is then converted using the modified Haar wavelet then fed into algorithms.

Approach 34, like approach 32, adds Huffman compression to the result of approach 33. Approach 35 attempts to derive PLUs from the source then reverts back to approach 31. Approach 36 follows approaches 32 and 34 in adding Huffman compression to approach 35.

### **3.5.12 APPROACH 31-36: PSEUDO CODE**

#### **3.5.12.1 APPROACH 31**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 4: For each plane, encode the values using one of the algorithms.

Step 5: Save the three compressed streams.

#### **3.5.12.2 APPROACH 32**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: For each plane, encode the values using one of the algorithms.

Step 6: For each compressed stream, Huffman encode the values.

Step 7: Save the three double compressed streams.

### **3.5.12.3 APPROACH 33**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane (both JPG and RGB), convert to the frequency domain using Haar wavelets.

Step 6: For each plane (only RGB), remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 8: For each difference plane, encode the values using one of the algorithms.

Step 9: Save the compressed difference streams

Step 10: Save the JPG image as raw JPG data.



#### **3.5.12.4 APPROACH 34**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 5: For each plane (both JPG and RGB), convert to the frequency domain using Haar wavelets.

Step 6: For each plane (only RGB), remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, calculate the difference between the JPG stream and the RGB stream.

Step 8: For each difference plane, encode the values using one of the algorithms.

Step 9: For each compressed stream, Huffman encode the values.

Step 10: Save the double compressed difference streams

Step 11: Save the JPG image as raw JPG data.

### **3.5.12.5 APPROACH 35**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 6: For each of the planes, encode the values using one of the algorithms.

Step 7: Save the compressed streams.

### **3.5.12.6 APPROACH 36**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane for the original image.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 6: For each of the planes, encode the values using one of the algorithms.

Step 7: For each compressed stream, Huffman encode the values.

Step 8: Save the double compressed streams

### **3.5.13 APPROACH 37-42: PHILOSOPHY**

These approaches are the Haar wavelet corollary to approaches 7-12 and 19-24. The literature review failed to turn up any direct reference to the combined effect of both. This set of approaches was devised to determine the impact of that combination.

Approach 37 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form. Convert that to new color space before passing the information off to the modified Haar wavelet. Finally, allow the algorithms to run. Algorithm 38 expands this a little by Huffman encoding the resulting values. This is nothing special, but interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact

seen reduction. Conversely, Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this since we are dealing with unique sources. Approach 39 manipulates JPG compression into giving us just the difference between lossless and lossy formats in the new color space. This is done by first calculating the lossy jpg then converting it to the new color space. Then the lossless original is converted to the new color space. Finally the difference is calculated. This leaves us with a color space difference which can then be converted using the modified Haar wavelet and then processed by the algorithms. Approach 40, like approach 38, adds Huffman compression to the result of approach 39. Approach 41 attempts to derive PLUs from the source then reverts back to approach 37. Approach 42 follows approaches 38 and 40 in adding Huffman compression to approach 41.

### **3.5.14 APPROACH 37-42: PSEUDO CODE**

#### **3.5.14.1 APPROACH 37**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: For each plane, encode the values using one of the algorithms.

Step 6: Save the three compressed streams.

#### **3.5.14.2 APPROACH 38**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane.

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: For each plane, encode the values using one of the algorithms.

Step 6: For each compressed stream, Huffman encode the values.

Step 7: Save the three double compressed streams.

#### **3.5.14.3 APPROACH 39**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane (both JPG and YCrCb), convert to the frequency domain using Haar wavelets.

Step 6: For each plane (only YCrCb), remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, calculate the difference between the JPG stream and the YCrCb stream.

Step 8: For each difference plane, encode the values using one of the algorithms.

Step 9: Save the compressed difference streams

Step 10: Save the JPG image as raw JPG data.

#### **3.5.14.4 APPROACH 40**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the YCrCb color plane for the compressed JPG image.

Step 4: Partition the image into the YCrCb color plane for the uncompressed RGB image.

Step 5: For each plane (both JPG and YCrCb), convert to the frequency domain using Haar wavelets.

Step 6: For each plane (only YCrCb), remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, calculate the difference between the JPG stream and the YCrCb stream.

Step 8: For each difference plane, encode the values using one of the algorithms.

Step 9: For each compressed stream, Huffman encode the values.

Step 10: Save the double compressed difference streams

Step 11: Save the JPG image as raw JPG data.

#### **3.5.14.5 APPROACH 41**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 6: For each of the planes, encode the values using one of the algorithms.

Step 7: Save the compressed streams.

#### **3.5.14.6 APPROACH 42**

Step 1: Load the image into memory.

Step 2: Partition the image into the YCrCb color plane for the original image

Step 3: For each plane, convert to the frequency domain using Haar wavelets.

Step 4: For each plane, remember and remove the first value, it is an artifact of the space domain.

Step 5: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new plane.

Step 6: For each of the planes, encode the values using one of the algorithms.



Step 7: For each compressed stream, Huffman encode the values.

Step 8: Save the double compressed streams.

### **3.5.15 APPROACH 43-48: PHILOSOPHY**

These final 6 approaches were inspired by the JPG format. They look at only a small region of any picture at a given instance. They then interpret large pictures as a mosaic of much smaller picture tiles laid side by side. The underlying concept is that while images may change drastically, in a small region they probably change very little.

Approach 43 deals with a strict base line. Get the picture information in as quickly as possible in the most undiluted form. This information can then be tiled into 8x8 blocks. For all of the tiles, convert them using the modified Haar wavelet then normalize all the tiles by averaging their values at each of the 8x8 positions. After this start the algorithms. Algorithm 44 expands this a little by Huffman encoding the resulting values. While this is nothing special, it is interesting in that Huffman is known to reduce the source toward its entropy, thus allowing us to see if we have in fact seen reduction. Conversely, Huffman encoding can yield strange results when you include the dictionary as part of the size consideration. We must do this since we are dealing with unique sources. Approach 45 manipulates JPG compression into giving us just the difference between lossless and

lossy compression. This is then converted using the modified Haar wavelet, normalized, and then fed into the algorithms. Approach 46, like approach 44, adds Huffman compression to the result of approach 45. Approach 47 attempts to derive PLUs from the source then reverts back to approach 43. Approach 48 follows approaches 44 and 46 in adding Huffman compression to approach 47.

### **3.5.16 APPROACH 43-48: PSEUDO CODE**

#### **3.5.16.1 APPROACH 43**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, break up the plane into several 8 pixel by 8 pixel units called tiles

Step 3: For each tile, convert to the frequency domain using modified Haar wavelets.

Step 4: For each tile, remember and remove the first value, as it is an artifact of the space domain.

Step 5: For each plane, normalize the values of each tile across all tiles values.

Step 6: For each plane, encode the values using one of the algorithms.

Step 7: Save the three compressed streams.

#### **3.5.16.2 APPROACH 44**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: For each plane, break up the plane into several 8 pixel by 8 pixel units called tiles

Step 3: For each tile, convert to the frequency domain using modified Haar wavelets.

Step 4: For each tile, remember and remove the first value, it is an artifact of the space domain.

Step 5: For each plane, normalize the values of each tile across all tiles values.

Step 6: For each plane, encode the values using one of the algorithms.

Step 7: For each compressed stream, Huffman encode the values.

Step 8: Save the three double compressed streams.

#### **3.5.16.3 APPROACH 45**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 4: For each plane (both JPG and RGB), break up the plane into several 8 pixel by 8 pixel units called tiles

Step 5: For each tile, convert to the frequency domain using Haar wavelets.

Step 6: For each tile, remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, normalize the values of each tile across all tiles values.

Step 9: For each tile, calculate the difference between the JPG tile and the RGB tile.

Step 10: Concatenate all tiles back into their appropriate planes.

Step 11: For each difference plane, encode the values using one of the algorithms.

Step 12: Save the compressed difference streams

Step 13: Save the JPG image as raw JPG data.

#### **3.5.16.4 APPROACH 46**

Step 1: Load the image into memory.

Step 2: Calculate the JPG compressed image from the RGB image.

Step 3: Partition the image into the RGB color plane for the compressed JPG image.

Step 4: Partition the image into the RGB color plane for the uncompressed RGB image.

Step 4: For each plane (both JPG and RGB), break up the plane into several 8 pixel by 8 pixel units called tiles

Step 5: For each tile, convert to the frequency domain using modified Haar wavelets.

Step 6: For each tile, remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, normalize the values of each tile across all tiles values.

Step 9: For each tile, calculate the difference between the JPG tile and the RGB tile.

Step 10: Concatenate all tiles back into their appropriate planes.

Step 11: For each difference plane, encode the values using one of the algorithms.

Step 12: For each compressed stream, Huffman encode the values.

Step 13: Save the double compressed difference streams

Step 14: Save the JPG image as raw JPG data.

### **3.5.16.5 APPROACH 47**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new stream.

Step 4: For each plane, break up the plane into several 8 pixel by 8 pixel units called tiles

Step 5: For each tile, convert to the frequency domain using Haar wavelets.

Step 6: For each tile, remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, normalize the values of each tile across all tiles values.

Step 8: For each plane, encode the values using one of the algorithms.

Step 9: Save the three compressed streams.

### **3.5.16.6 APPROACH 48**

Step 1: Load the image into memory.

Step 2: Partition the image into the RGB color plane.

Step 3: Reduce the number of colors in each plane by removing the least significant bit and store this bit as a new stream.

Step 4: For each plane, break up the plane into several 8 pixel by 8 pixel units called tiles

Step 5: For each tile, convert to the frequency domain using modified Haar wavelets.

Step 6: For each tile, remember and remove the first value, it is an artifact of the space domain.

Step 7: For each plane, normalize the values of each tile across all tiles values.

Step 8: For each plane, encode the values using one of the algorithms.

Step 9: For each compressed stream, Huffman encode the values.

Step 10: Save the three double compressed streams.

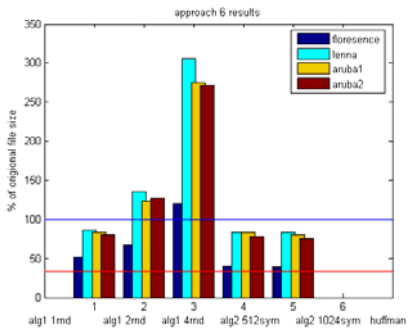
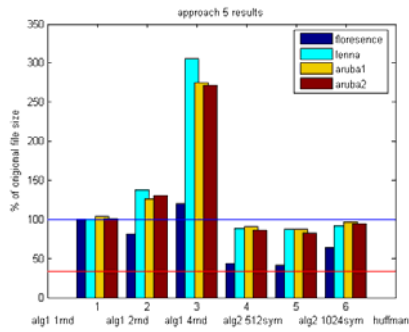
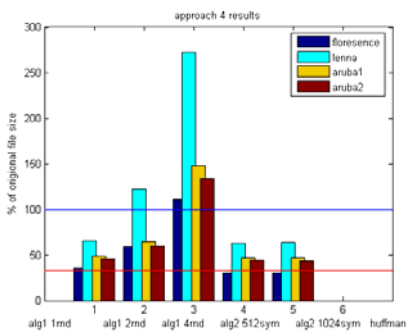
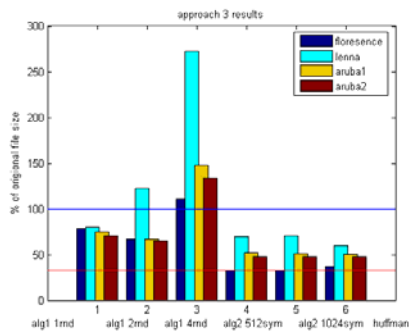
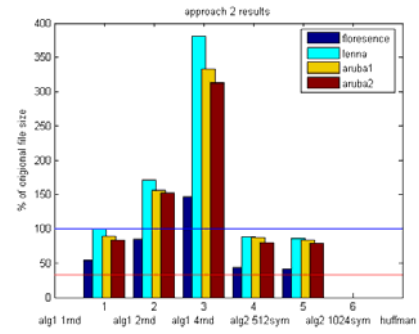
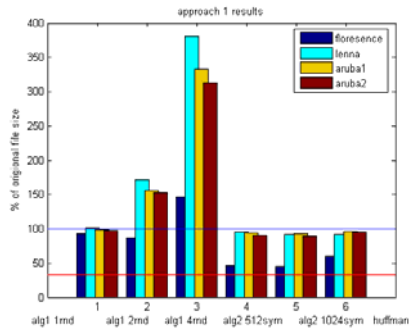
## **4.0 DATA SUMMARY**

### **4.1 COMPARATIVE PICTORIAL SUMMARY OF APPROACHES**

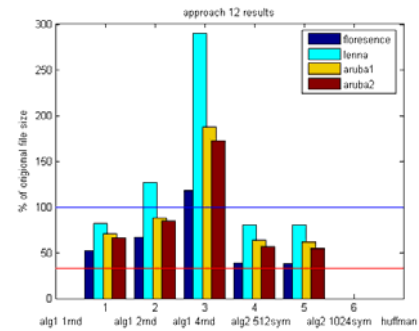
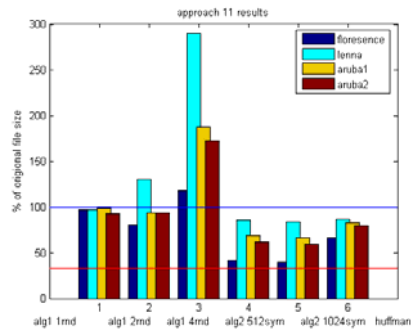
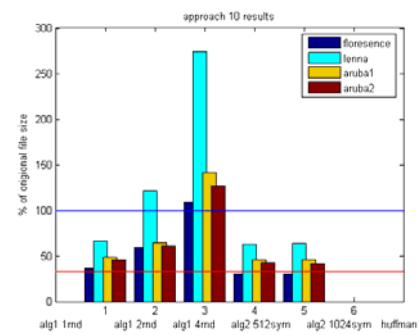
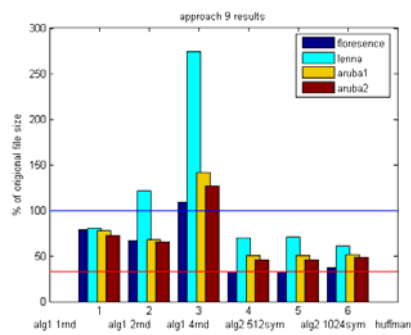
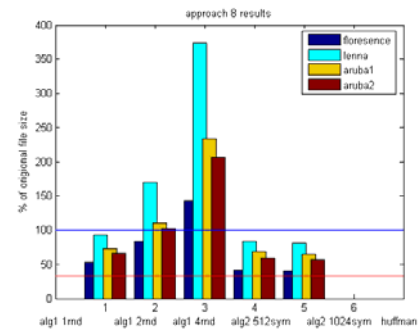
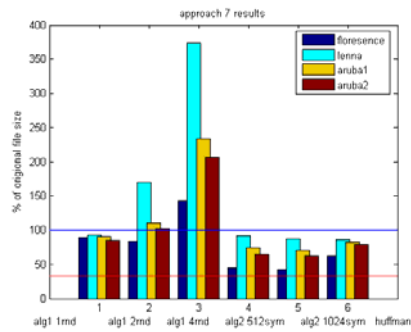
This section includes a comparative summary of all the results on a given approach. They are grouped into 6 unit blocks for easy reference to the approaches described in the previous section. This also allows for a quick glance in order to determine the value of the approach. In addition there are 2 indicator lines corresponding to normalized file size and normalized average jpeg file size. As can be seen, every even numbered approach has only 5 columns. This is because every even numbered approach automatically included Huffman encoding as a final procedure, so it is entirely redundant to include this data. The graphs were rendered in Matlab for ease of research. A diligent reader will note that the windows may not be labeled in order. This is an artifact of the way Matlab titles modal windows, nothing more. The graph title, however, is in fact correct.



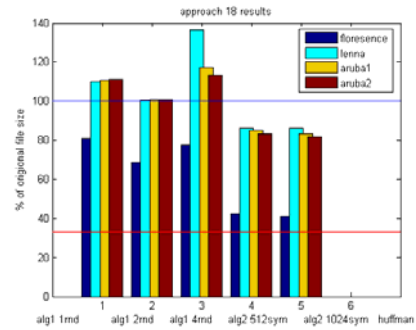
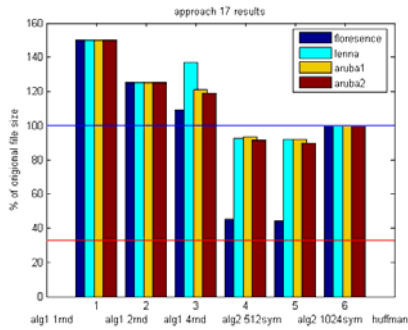
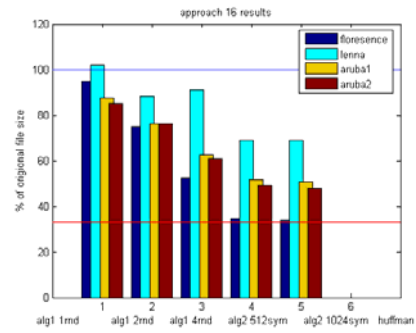
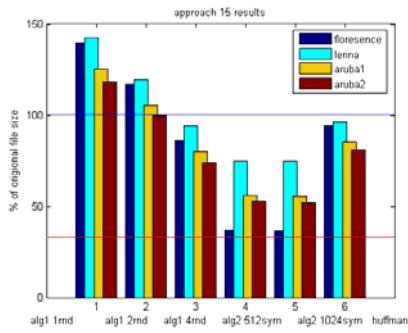
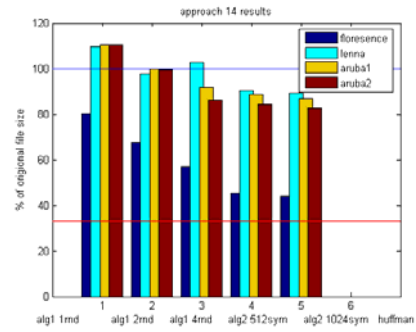
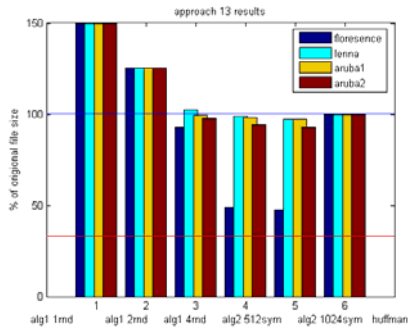
**FIGURE 1: APPROACH 1-6 PICTORIAL RESULTS**



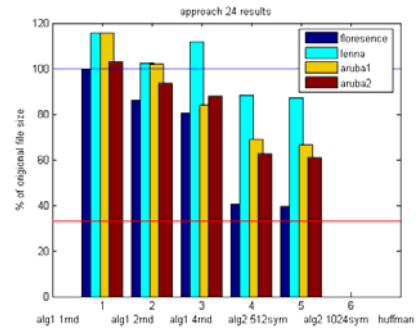
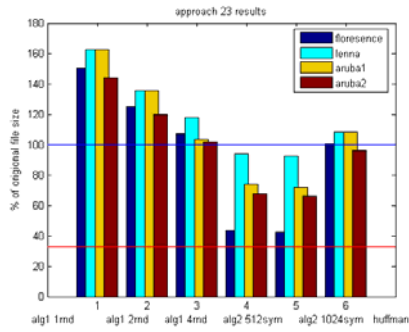
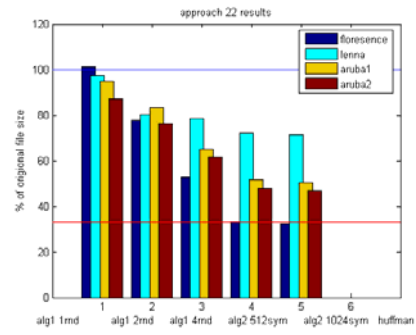
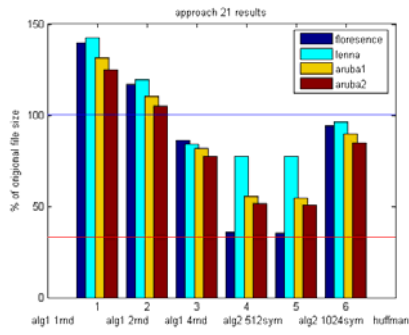
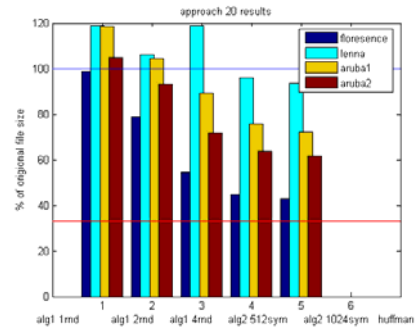
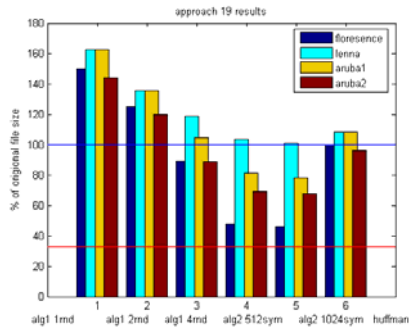
**FIGURE 2: APPROACH 7-12 PICTORIAL RESULTS**



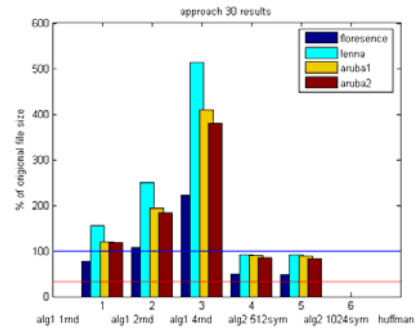
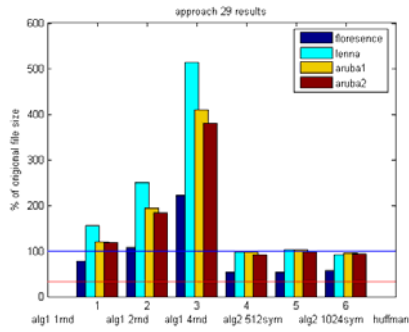
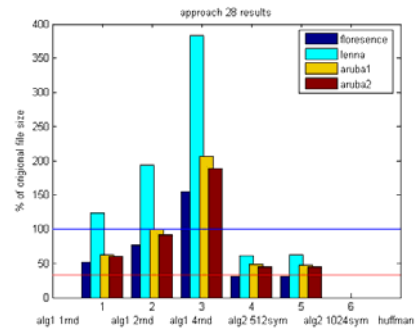
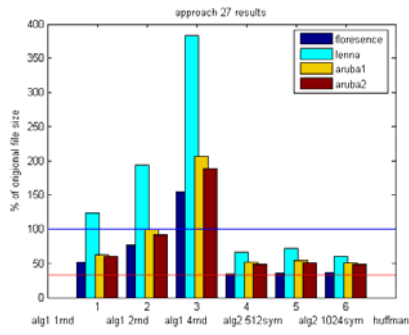
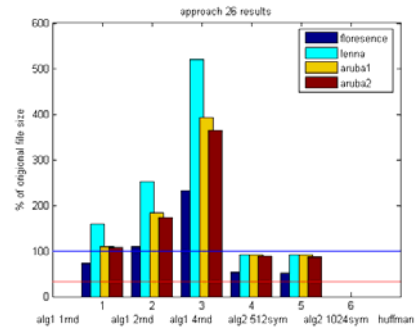
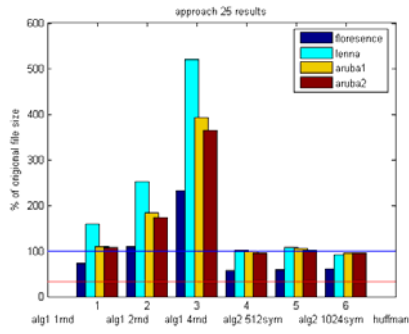
**FIGURE 3: APPROACH 13-18 PICTORIAL RESULTS**



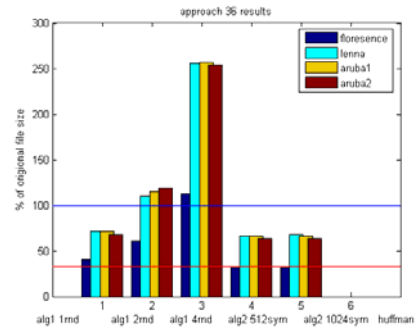
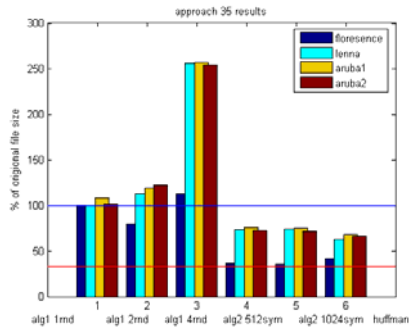
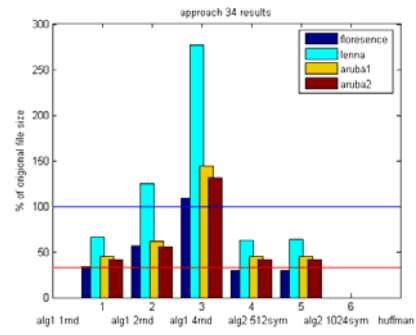
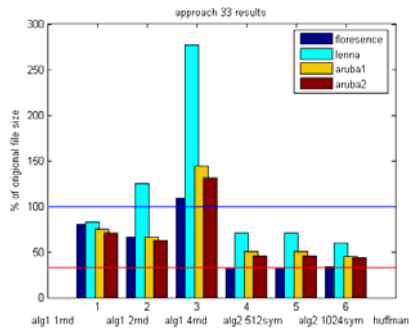
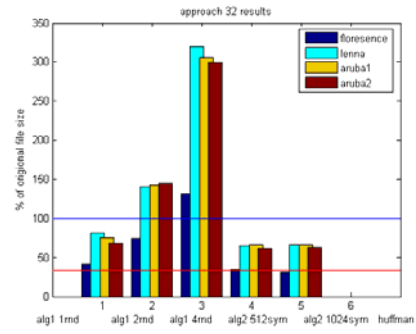
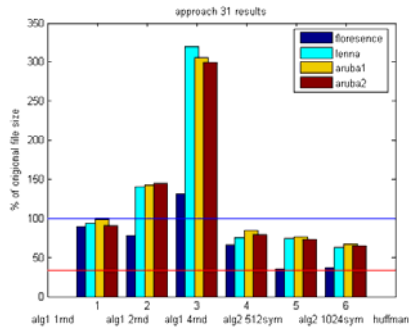
**FIGURE 4: APPROACH 19-24 PICTORIAL RESULTS**



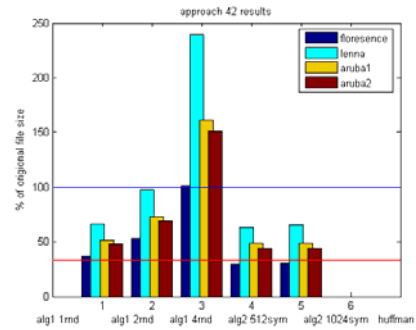
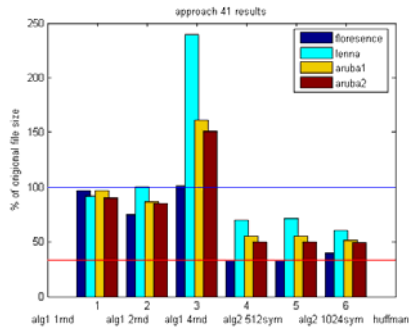
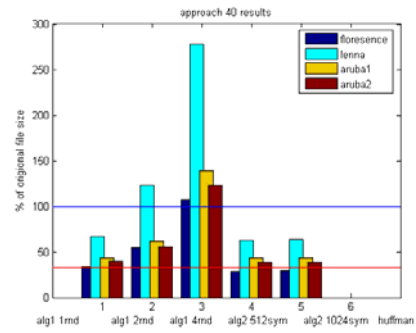
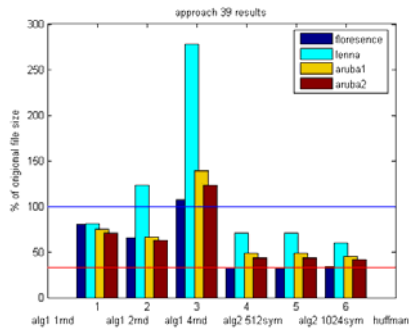
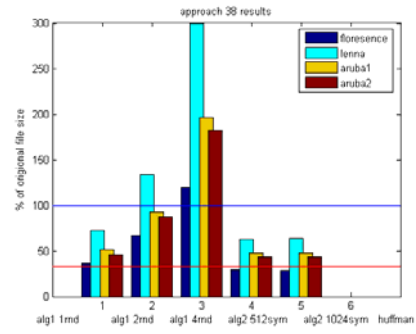
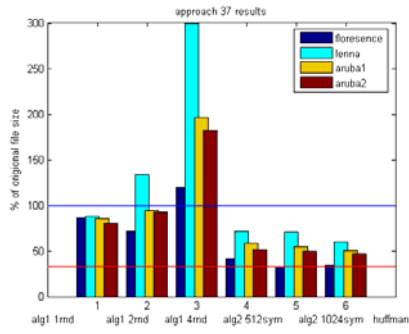
**FIGURE 5: APPROACH 25-30 PICTORIAL RESULTS**



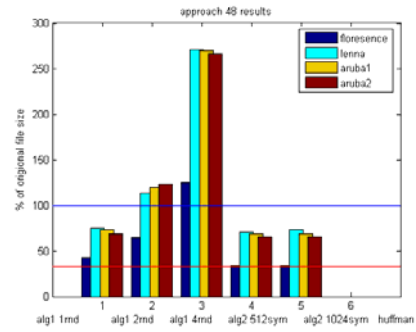
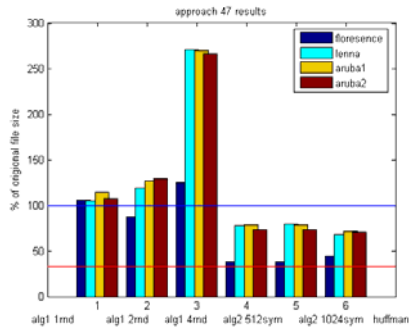
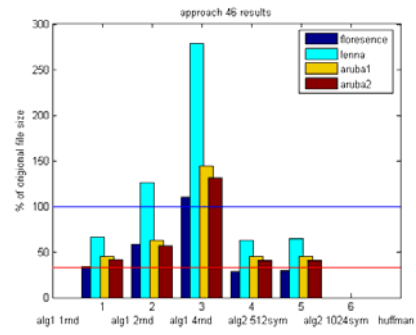
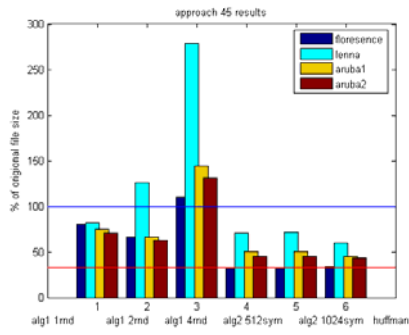
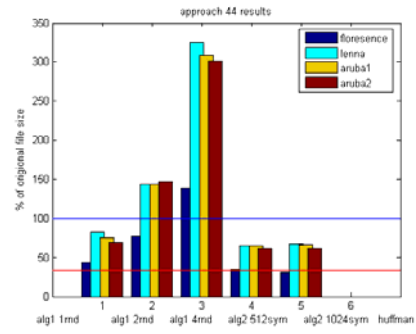
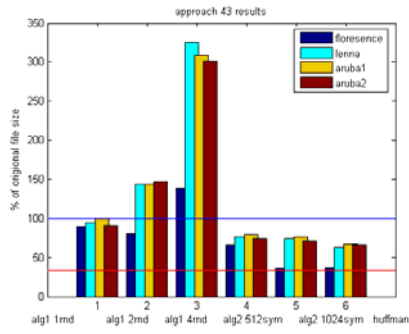
**FIGURE 6: APPROACH 31-36 PICTORIAL RESULTS**



**FIGURE 7: APPROACH 37-42 PICTORIAL RESULTS**



**FIGURE 8: APPROACH 43-48 PICTORIAL RESULTS**





## **4.2 DATA BASED SUMMARY OF APPROACHES**

This section includes the numeric results of the compression as produced by my reference implementation. Personally, I feel that it is better to display the data result as it is produced by the computer, literally, as opposed to a doctored or edited version. However, there are about 20 pages of output for each run, and as there are 48 runs, I opted, for sake of compactness, to put a chart based version of the numeric results here. They are just a replication of the graphical data, but some prefer to review hard numbers as opposed to the quick approximant information provided by a graph. As with Appendix I the actual results appear on the companion CD.

**TABLE 4: APPROACH 1-6 DATA RESULTS**

<b>Approach-Picture</b>	<b>Algorithm 1 1 round</b>	<b>Algorithm 1 2 rounds</b>	<b>Algorithm 1 4 rounds</b>	<b>Algorithm 2 512 symbols</b>	<b>Algorithm 2 1024 symbols</b>	<b>Huffman reference</b>
1-Floressence	0.92416172	0.867289365	1.469315528	0.465866811	0.445334922	0.602149734
1-Lenna	1.0120765	1.71653278	3.804966649	0.953543992	0.921594281	0.917582767
1-Aruba1	0.98984593	1.556902071	3.33174689	0.938935468	0.930718705	0.959224167
1-Aruba2	0.97224227	1.524087448	3.133189376	0.910513513	0.895581361	0.949667312
2-Floressence	0.54216691	0.843851954	1.469318542	0.440370149	0.419398213	0
2-Lenna	1.00269935	1.716544223	3.804978092	0.879916489	0.857452008	0
2-Aruba1	0.8951614	1.556903633	3.331748452	0.873498929	0.841695929	0
2-Aruba2	0.83649754	1.52408901	3.133190939	0.805067105	0.787927683	0
3-Floressence	0.78257845	0.668812989	1.108663386	0.332994088	0.329684351	0.37111584
3-Lenna	0.80365194	1.227642959	2.721763642	0.698432013	0.701943836	0.603964978
3-Aruba1	0.75325717	0.674661904	1.475347627	0.518603471	0.516602101	0.504110205
3-Aruba2	0.70634008	0.651158652	1.340082402	0.480967019	0.479273805	0.481195142
4-Floressence	0.35731863	0.590472057	1.1086664	0.300584587	0.300278496	0
4-Lenna	0.65061425	1.227654402	2.721775086	0.628873241	0.638378814	0
4-Aruba1	0.48463018	0.64553492	1.475349189	0.471900437	0.468694043	0
4-Aruba2	0.46016062	0.597744917	1.340083964	0.439909244	0.436330632	0
5-Floressence	0.99730814	0.814080062	1.20472077	0.431592967	0.417014117	0.642760407
5-Lenna	1.00261543	1.380240716	3.059052545	0.887060927	0.874330884	0.91602902
5-Aruba1	1.03549741	1.262965764	2.744885204	0.900922977	0.878966412	0.9620698
5-Aruba2	1.00917908	1.303185005	2.717963408	0.861344876	0.822826487	0.949582764
6-Floressence	0.51942407	0.669668973	1.204726799	0.407931513	0.393706979	0
6-Lenna	0.86984257	1.351005612	3.059075432	0.837550827	0.833260859	0
6-Aruba1	0.83858311	1.232876115	2.744888329	0.831914423	0.807604755	0
6-Aruba2	0.80479263	1.272303524	2.717966533	0.784281189	0.757285956	0

TABLE 5: APPROACH 7-12 DATA RESULTS

Approach-Picture	Algorithm 1 1 round	Algorithm 1 2 rounds	Algorithm 1 4 rounds	Algorithm 2 512 symbols	Algorithm 2 1024 symbols	Huffman reference
7-Floressence	0.8896812	0.8413885	1.4278234	0.4444716	0.4226222	0.6236669
7-Lenna	0.9333275	1.7048008	3.7332693	0.9212662	0.8732565	0.8650643
7-Aruba1	0.9027539	1.1109148	2.3313801	0.747119	0.7051309	0.8291292
7-Aruba2	0.8505425	1.027785	2.0637973	0.6522326	0.6311076	0.7860468
8-Floressence	0.5303121	0.8413915	1.4278264	0.4191983	0.3987772	0
8-Lenna	0.9305417	1.7048123	3.7332807	0.833875	0.814003	0
8-Aruba1	0.7261255	1.1109163	2.3313816	0.6820288	0.6460759	0
8-Aruba2	0.6638988	1.0277865	2.0637989	0.5906097	0.5708481	0
9-Floressence	0.7829271	0.6679105	1.0946348	0.3322744	0.328814	0.3764507
9-Lenna	0.8065128	1.2175296	2.7402001	0.7014632	0.7027271	0.6054386
9-Aruba1	0.7748224	0.6834254	1.4142071	0.5082275	0.5052638	0.5121501
9-Aruba2	0.7279341	0.6569135	1.2715186	0.4616212	0.4597466	0.4874529
10-Floressence	0.3598732	0.5878381	1.0946378	0.3005481	0.2999024	0
10-Lenna	0.6582749	1.2175411	2.7402115	0.6304562	0.6397266	0
10-Aruba1	0.4875994	0.6434653	1.4142086	0.462395	0.4588375	0
10-Aruba2	0.4601985	0.6107618	1.2715202	0.4223891	0.4190169	0
11-Floressence	0.9708969	0.7983482	1.1800379	0.4173781	0.4030853	0.6643773
11-Lenna	0.9699359	1.3001821	2.9024725	0.8519529	0.8407232	0.8635398
11-Aruba1	0.9874798	0.9382817	1.8732514	0.6884152	0.6652495	0.8330436
11-Aruba2	0.9215544	0.937983	1.7193613	0.6173965	0.5954689	0.7905459
12-Floressence	0.521554	0.6709905	1.1800439	0.3946326	0.381761	0
12-Lenna	0.8173674	1.2709648	2.9024954	0.7989297	0.7983066	0
12-Aruba1	0.7077114	0.8834691	1.8732545	0.6384308	0.6150487	0
12-Aruba2	0.663075	0.850562	1.7193644	0.5690983	0.5503698	0

**TABLE 6: APPROACH 13-18 DATA RESULTS**

<b>Approach-Picture</b>	<b>Algorithm 1 1 round</b>	<b>Algorithm 1 2 rounds</b>	<b>Algorithm 1 4 rounds</b>	<b>Algorithm 2 512 symbols</b>	<b>Algorithm 2 1024 symbols</b>	<b>Huffman reference</b>
13-Floressence	1.499991	1.2500176	0.9269872	0.4877533	0.4739812	0.999999
13-Lenna	1.4999657	1.2500668	1.0246718	0.9893285	0.9730752	0.9999962
13-Aruba1	1.4999953	1.2500091	0.9912629	0.9821196	0.9696308	0.9999995
13-Aruba2	1.4999953	1.2500091	0.9748308	0.9404356	0.9297715	0.9999995
14-Floressence	0.8023511	0.6743189	0.5696622	0.4541218	0.4395895	0
14-Lenna	1.0964251	0.9781916	1.0246832	0.9042717	0.8933102	0
14-Aruba1	1.1050386	0.999555	0.9163945	0.887062	0.8684292	0
14-Aruba2	1.1047806	0.9963613	0.8602645	0.8442923	0.8266266	0
15-Floressence	1.3981369	1.1689955	0.8629013	0.371085	0.3659662	0.9398099
15-Lenna	1.4230361	1.1939653	0.9400498	0.7479256	0.7470533	0.9647266
15-Aruba1	1.2509016	1.0529981	0.7996951	0.5600451	0.5536085	0.8550713
15-Aruba2	1.1838264	0.9963394	0.7398387	0.528831	0.519299	0.8088292
16-Floressence	0.9488349	0.7491606	0.526206	0.3453539	0.3389816	0
16-Lenna	1.020069	0.8825739	0.9112775	0.6907803	0.6883149	0
16-Aruba1	0.8754414	0.7628323	0.6259447	0.519033	0.5087475	0
16-Aruba2	0.8513162	0.7624152	0.6080754	0.4914619	0.4792196	0
17-Floressence	1.5000013	1.2500501	1.0940454	0.4533753	0.441774	1.0000084
17-Lenna	1.5000051	1.2501901	1.3666016	0.9259466	0.9210208	1.0000318
17-Aruba1	1.5000007	1.250026	1.2090583	0.9333458	0.9194839	1.0000043
17-Aruba2	1.5000007	1.250026	1.1887231	0.9138588	0.896359	1.0000043
18-Floressence	0.8090885	0.687161	0.7744751	0.4219638	0.411901	0
18-Lenna	1.1008689	1.0057458	1.3666245	0.8597648	0.8624451	0
18-Aruba1	1.1055973	1.0077774	1.173256	0.8498219	0.8339646	0
18-Aruba2	1.1114208	1.0100416	1.1304792	0.8326196	0.8149991	0

**TABLE 7: APPROACH 19-24 DATA RESULTS**

Approach-Picture	Algorithm 1 1 round	Algorithm 1 2 rounds	Algorithm 1 4 rounds	Algorithm 2 512 symbols	Algorithm 2 1024 symbols	Huffman reference
19-Floressence	1.499993	1.2500186	0.8925335	0.476576	0.4602986	1
19-Lenna	1.6249647	1.3542326	1.1873511	1.036232	1.0071508	1.0833302
19-Aruba1	1.6249952	1.3541757	1.0463228	0.8113344	0.7820659	1.0833329
19-Aruba2	1.4374969	1.1979268	0.8854448	0.6942539	0.6753074	0.9583337
2-Floressence	0.9877389	0.7868523	0.5439881	0.445932	0.4289225	0
20-Lenna	1.1870752	1.0615052	1.1873625	0.95923	0.9354114	0
20-Aruba1	1.1857332	1.0432855	0.8915406	0.7570919	0.7220969	0
20-Aruba2	1.049037	0.929831	0.7175395	0.6366826	0.6149003	0
21-Floressence	1.3981369	1.1689955	0.8626578	0.3563361	0.3520414	0.9398099
21-Lenna	1.4230349	1.1939653	0.8414746	0.7741676	0.7729966	0.9647266
21-Aruba1	1.3134012	1.105081	0.8195684	0.5547504	0.5453728	0.8967376
21-Aruba2	1.246326	1.0484223	0.7744346	0.5148011	0.5058145	0.8504955
22-Floressence	1.012328	0.7757855	0.5282907	0.3303066	0.32489	0
22-Lenna	0.9744013	0.8020651	0.7840941	0.719789	0.7157623	0
22-Aruba1	0.9480968	0.8333502	0.6524592	0.5167783	0.503571	0
22-Aruba2	0.8724377	0.7647503	0.6140857	0.4804733	0.4681305	0
23-Floressence	1.5000044	1.2500521	1.072059	0.4338317	0.4228195	1.0000104
23-Lenna	1.6250079	1.3543572	1.1776484	0.9393479	0.922717	1.083367
23-Aruba1	1.6250011	1.3541927	1.0319889	0.7419007	0.7190771	1.0833379
23-Aruba2	1.4375027	1.1979442	1.0208106	0.6787303	0.6608662	0.9583391
24-Floressence	0.9959106	0.8626839	0.8053658	0.4048927	0.3950553	0
24-Lenna	1.156828	1.0231307	1.1179131	0.8817767	0.8730314	0
24-Aruba1	1.1560947	1.0185878	0.8395841	0.6889489	0.6631901	0
24-Aruba2	1.0286732	0.9353454	0.8780473	0.6268427	0.6074502	0

**TABLE 8: APPROACH 25-30 DATA RESULTS**

Approach-Picture	Algorithm 1 1 round	Algorithm 1 2 rounds	Algorithm 1 4 rounds	Algorithm 2 512 symbols	Algorithm 2 1024 symbols	Huffman reference
25-Floressence	0.7350834	1.0974452	2.3237373	0.5667798	0.5834226	0.6021487
25-Lenna	1.5862457	2.5229845	5.2100559	1.0017241	1.0759874	0.9175828
25-Aruba1	1.0924189	1.8299594	3.9250665	0.9999628	1.0688058	0.9592242
25-Aruba2	1.0837732	1.7203948	3.6417001	0.9548872	1.0154071	0.9496673
26-Floressence	0.7350864	1.0974482	2.3237404	0.5293503	0.5208597	0
26-Lenna	1.5862571	2.522996	5.2100673	0.9203393	0.9280903	0
26-Aruba1	1.0924205	1.829961	3.9250681	0.9202436	0.9128586	0
26-Aruba2	1.0837747	1.7203964	3.6417016	0.8877215	0.8675926	0
27-Floressence	0.5193303	0.7616976	1.5477506	0.3402954	0.3540307	0.3711158
27-Lenna	1.2349946	1.938229	3.8256205	0.6635541	0.717372	0.603965
27-Aruba1	0.6238843	1.0037319	2.0687747	0.5180278	0.5419473	0.5041102
27-Aruba2	0.5972489	0.9181638	1.8856502	0.4822059	0.5055538	0.4811951
28-Floressence	0.5193333	0.7617006	1.5477536	0.3151822	0.3152549	0
28-Lenna	1.2350061	1.9382405	3.825632	0.6115506	0.6223874	0
28-Aruba1	0.6238858	1.0037335	2.0687763	0.4811082	0.4795037	0
28-Aruba2	0.5972505	0.9181654	1.8856518	0.4478503	0.4465614	0
29-Floressence	0.786781	1.0816363	2.2236378	0.527636	0.5308576	0.5661147
29-Lenna	1.5524714	2.5011977	5.1289279	0.9763836	1.0320336	0.9139718
29-Aruba1	1.1954805	1.9317256	4.0933002	0.9715872	1.0273836	0.9544605
29-Aruba2	1.1787077	1.8259773	3.7927464	0.9144657	0.9718879	0.937402
30-Floressence	0.7720756	1.0816423	2.2236438	0.4973698	0.4863913	0
30-Lenna	1.5524943	2.5012206	5.1289508	0.9141574	0.9265403	0
30-Aruba1	1.1954836	1.9317288	4.0933033	0.9008469	0.8941222	0
30-Aruba2	1.1787108	1.8259805	3.7927495	0.8492011	0.8389189	0

**TABLE 9: APPROACH 31-36 DATA RESULTS**

Approach-Picture	Algorithm 1 1 round	Algorithm 1 2 rounds	Algorithm 1 4 rounds	Algorithm 2 512 symbols	Algorithm 2 1024 symbols	Huffman reference
31-Floressence	0.8933125	0.778832	1.3118072	0.6588891	0.357322	0.372724
31-Lenna	0.9328265	1.4097098	3.1955699	0.7558342	0.7370176	0.6284702
31-Aruba1	0.989107	1.4238599	3.0579061	0.8422529	0.7665402	0.6721529
31-Aruba2	0.9027141	1.45171	2.9984828	0.794124	0.733227	0.6547661
32-Floressence	0.4171263	0.7433271	1.3118102	0.338624	0.3077972	0
32-Lenna	0.814696	1.4097212	3.1955814	0.6474445	0.6585432	0
32-Aruba1	0.7532363	1.4238615	3.0579076	0.6558933	0.6571817	0
32-Aruba2	0.6843653	1.4517116	2.9984844	0.6205685	0.6247434	0
33-Floressence	0.8050246	0.6602431	1.0950119	0.328748	0.3257802	0.3371739
33-Lenna	0.8296715	1.2481964	2.7640492	0.7070043	0.7054811	0.604157
33-Aruba1	0.7531318	0.6623242	1.4470213	0.5032128	0.5031876	0.454243
33-Aruba2	0.70611	0.6276561	1.3122023	0.4606363	0.462167	0.4308982
34-Floressence	0.3382864	0.5641924	1.0950149	0.2919698	0.2933503	0
34-Lenna	0.6648599	1.2482079	2.7640606	0.6294174	0.6399008	0
34-Aruba1	0.448033	0.6203454	1.4470229	0.4509706	0.4510869	0
34-Aruba2	0.4194002	0.5560175	1.3122038	0.4166265	0.4163602	0
35-Floressence	0.9963835	0.796164	1.1249214	0.3643293	0.3548491	0.4170593
35-Lenna	0.9999415	1.1299604	2.5596285	0.7315451	0.7370786	0.6277391
35-Aruba1	1.0822194	1.1883366	2.5654384	0.7618323	0.7531759	0.6754565
35-Aruba2	1.0096779	1.2229941	2.5438018	0.7278909	0.7169249	0.6656932
36-Floressence	0.4070682	0.6063145	1.1249274	0.3199005	0.3211908	0
36-Lenna	0.7182035	1.1007507	2.5596514	0.6594116	0.6788131	0
36-Aruba1	0.7110074	1.1577371	2.5654416	0.6633045	0.6633672	0
36-Aruba2	0.675652	1.1904527	2.543805	0.6354602	0.6352642	0

**TABLE 10: APPROACH 37-42 DATA RESULTS**

<b>Approach- Picture</b>	<b>Algorithm 1 1 round</b>	<b>Algorithm 1 2 rounds</b>	<b>Algorithm 1 4 rounds</b>	<b>Algorithm 2 512 symbols</b>	<b>Algorithm 2 1024 symbols</b>	<b>Huffman reference</b>
37-Floressence	0.8601987	0.7156962	1.1990149	0.420074	0.3266462	0.3461523
37-Lenna	0.8789158	1.3391923	2.997335	0.7144298	0.7072459	0.6002739
37-Aruba1	0.8556467	0.9467932	1.9623875	0.5794473	0.5484582	0.5026755
37-Aruba2	0.8037874	0.9215613	1.8243365	0.517005	0.4962033	0.4687843
38-Floressence	0.3679826	0.6688987	1.1990179	0.2961493	0.2881119	0
38-Lenna	0.7240574	1.3392037	2.9973464	0.6230651	0.6339808	0
38-Aruba1	0.5172474	0.9263601	1.9623891	0.4770136	0.4765586	0
38-Aruba2	0.4648271	0.8740401	1.8243381	0.4314151	0.4323175	0
39-Floressence	0.8053585	0.655503	1.0737027	0.3254858	0.3221908	0.3339783
39-Lenna	0.8098034	1.2346183	2.7783368	0.7068085	0.7051302	0.603155
39-Aruba1	0.7536075	0.6665073	1.3876837	0.4872855	0.4856317	0.4475705
39-Aruba2	0.7065949	0.626716	1.2358073	0.43224	0.4327755	0.4175588
40-Floressence	0.3343005	0.5514799	1.0737057	0.2889692	0.290278	0
40-Lenna	0.669338	1.2346297	2.7783482	0.628432	0.6393019	0
40-Aruba1	0.4365259	0.6163017	1.3876853	0.4360027	0.4359739	0
40-Aruba2	0.4017544	0.5552323	1.2358089	0.391064	0.3908226	0
41-Floressence	0.9692187	0.7515343	1.0110153	0.3335014	0.3306549	0.3953138
41-Lenna	0.9208772	1.0070681	2.3951437	0.7017391	0.7095473	0.5999039
41-Aruba1	0.9668149	0.8671887	1.6145514	0.5523235	0.5497488	0.5168729
41-Aruba2	0.8999942	0.8541725	1.5074873	0.5011812	0.4993028	0.4934834
42-Floressence	0.3684029	0.5272183	1.0110213	0.298596	0.3011248	0
42-Lenna	0.6613125	0.9778623	2.3951666	0.6340138	0.6540701	0
42-Aruba1	0.5161151	0.7294006	1.6145545	0.4876315	0.4877788	0
42-Aruba2	0.477286	0.6885527	1.5074904	0.4443901	0.4449231	0



**TABLE 11: APPROACH 43-48 DATA RESULTS**

<b>Approach- Picture</b>	<b>Algorithm 1 1 round</b>	<b>Algorithm 1 2 rounds</b>	<b>Algorithm 1 4 rounds</b>	<b>Algorithm 2 512 symbols</b>	<b>Algorithm 2 1024 symbols</b>	<b>Huffman reference</b>
43-Floressence	0.8967381	0.8064435	1.3831053	0.6585619	0.3608795	0.3740518
43-Lenna	0.9425432	1.441203	3.2535684	0.7608146	0.744588	0.6321231
43-Aruba1	0.9950619	1.4428139	3.0879705	0.7945814	0.7578523	0.6714468
43-Aruba2	0.9068726	1.4636472	3.0116405	0.7377851	0.7115275	0.6558449
44-Floressence	0.4300722	0.7753411	1.3831113	0.3390807	0.310269	0
44-Lenna	0.8226758	1.4412259	3.2535913	0.6529678	0.6677284	0
44-Aruba1	0.7568884	1.442817	3.0879737	0.6541609	0.6575254	0
44-Aruba2	0.6895623	1.4636503	3.0116436	0.6165569	0.6184838	0
45-Floressence	0.7991295	0.6645498	1.1031742	0.3235907	0.3218137	0.3367141
45-Lenna	0.8238786	1.2585539	2.7858042	0.7086293	0.7113744	0.6019141
45-Aruba1	0.7489313	0.6625502	1.4462071	0.500305	0.500538	0.4534841
45-Aruba2	0.702595	0.6277818	1.3109224	0.4530098	0.4534048	0.4302621
46-Floressence	0.3397378	0.5832213	1.1031802	0.2875992	0.2900944	0
46-Lenna	0.6648752	1.2585768	2.7858271	0.6314696	0.6455067	0
46-Aruba1	0.4486776	0.6271361	1.4462102	0.4487927	0.4491925	0
46-Aruba2	0.4204183	0.5659147	1.3109256	0.4095293	0.4080786	0
47-Floressence	1.0599992	0.8712397	1.2458843	0.3848293	0.3781178	0.4446079
47-Lenna	1.0440822	1.1906925	2.7070946	0.7781919	0.7904133	0.6765855
47-Aruba1	1.1448828	1.2638562	2.6965582	0.7842029	0.7807246	0.7160648
47-Aruba2	1.0720457	1.297376	2.6611598	0.7345343	0.732919	0.7023287
48-Floressence	0.4284309	0.6458997	1.2457892	0.3376039	0.3416842	0
48-Lenna	0.7521596	1.1369242	2.706881	0.7028644	0.7282838	0
48-Aruba1	0.7302103	1.1998624	2.6959815	0.6906583	0.6910524	0
48-Aruba2	0.6904982	1.2285506	2.661166	0.6533873	0.6526618	0

## **5.0 RESULTS AND DISCUSSION**

Unfortunately, none of the algorithms tested in this study yielded results that produced better compression than today's leading compression schemes. In this section, a discussion is offered as to why this may have occurred. The discussion ends with suggestions for some additional approaches to the problem of lossless compression that may be worthy of further research.

### **5.1 APPROACH 1-6.**

Pictures with highly similar colors end up producing very similar codes, thus enabling high compression rates. The issue here is that quite often, except in contrived demos, images include a wide variety of colors and in many combinations. Essentially, there were too few unique patterns to justify the cost of creating the pattern in the first place. There does not seem to be any straight forward way to proceed except to remove the tables used to translate between a pattern and its original source string from the compressed file. Even this seems a waste, since so many images have a different color makeup as to prevent any single color table from producing any real good. Furthermore, a combined approach using many tables seems equally unlikely to produce all but the most marginal improvements over a single table design.

## **5.2 APPROACH 7-12.**

As these approaches are direct extension of approaches 1-6 we expect to see a similar result for these, and indeed we do. In fact they produce nearly identical results. Any improvements we might make to approach 1-6 are likely to be applicable to 7-12 and vice versa.

## **5.3 APPROACH 13-18.**

In hind sight this approach was doomed to fail. The reason is that when you convert to base2 from base8 the length of the sequence increases 8 fold, but the file size remains the same. The cost comes in the form of doubling the file size at the time the first pattern is added. Since the value of the symbol replacing the pattern must be unique from all other symbols in the file and since the next available pattern is 2, this forces all symbols into the next highest base. Essentially, over 3 recursive steps bytes have been rebuilt with entirely new meaning, but with no compression worth noting. The only way to remedy this is to find a way to represent all numbers in a series with the lowest base available to them. To my knowledge no standard algorithm does this. However, it could be accomplished by pre-pending a marker to each symbol indicating its base.

## **5.4 APPROACH 19-24.**

Given the extremely close nature of approaches 1-6 and 7-12, and also that approaches 13-18 and 19-24 were designed to mirror these algorithms, it is startling to see a significant dissimilarity in the results sets. This begs the question as to what a re-representation of the data set looks like. Especially since the source data in the two groups are identical. The only clue is that the YCrCb color space tends to reduce the correlation between the color planes. While this certainly bears further study it seems that the absolute gain in compression is less than is gained in other approaches.

#### **5.5 APPROACH 25-30.**

Like approaches 19-24 this set of approaches defies immediate explanation. Not even bothering with the correlation, these approaches bear no resemblance whatsoever to any approach 13-24. I am at an utter loss to explain why this is so radically different from the other 12 similar approaches.

#### **5.6 APPROACH 31-36.**

These approaches most likely failed due to how the modified Haar wavelets are handled, as discussed above. Most of the narrowing information of similar wavelet pairs is lost due to the integer to integer constraint, and thus presumably, many patterns are also lost. It bears

research on how to effectively use wavelets to create similar patterns. In addition, a pattern table, as described for use in approach 1-6, seems to bear significant research. This is because, after completing the main body of research as presented here, most images have been observed to possess nearly identical Haar breakdowns for frequency.

### **5.7 APPROACH 37-42.**

These approaches, like approaches 7-12, provide similar justification for failure as well as for prospective research. As it appears, the usefulness of moving to a different color space, considering the use of the Haar wavelet, does not seem to provide any real benefit.

### **5.8 APPROACH 43-48.**

These approaches, again, seem to be of dubious benefit considering the results in approaches 31-36. However, they might benefit from a multi tabled design. Since, in these approaches only a small region is observed and compressed separately, we might be able to observe several distinct styles of patterns allowing for a different type of pattern not just a bigger pattern or pattern of smaller patterns.

### **5.9 GENERAL REMARKS FOR FUTURE RESEARCH.**

At the inception of this thesis I had only the vaguest notion of how pattern reduction worked. Having implemented several natural

language processors using multistage pattern reduction I am better able to grasp new ways to find solutions to this problem.

First, as briefly mentioned above, external tables are mandatory.

Additionally, the table or set of tables must be universal, as in the known frequency distribution of letters in English text for use in Huffman compression. If they are not universal then they must be transmitted with each file. If that is so the “cost” of any approach similar to those presented in this thesis will not be worth the trouble.

This is why I am thrilled with the observed results of the modified Haar wavelet. It seems to have a universally describable distribution that can be externally coded.

Second, and most importantly, this research brought to light the concept of generics. I was unfamiliar with this concept initially. That is to say a pattern that in whole or in part accepts data it can't deal with itself, but remembers allowing for lossless reconstruction, yet a later stage of reduction might know about, if only the unknown data is masked. The tremendous advantage of this is that a pattern with a generic can represent several similar patterns that would otherwise take up a lot of space. In extension, a single pattern at a later stage or structure of patterns, can be built that recognizes all of those previous patterns plus, perhaps, even wholly unknown data.

## **6.0 CONCLUSION**

I have produced to my satisfaction a new and innovative compression scheme. It utilizes multiple passes to gather increasingly valuable patterns and encodes them into a new stream. Even though I failed to produce better compression than today's leading compression schemes, the fact that this technique produced results at all is very promising. Moreover, in science, a failed result may provide as much new information as a positive result.

Since starting this thesis in the summer of 2003 I have used multistage pattern reduction successfully as an analysis technique. It is an incredibly powerful way to get underlying information. So the reason that I say that any compression, much less reasonable compression, is so important is that we can now begin to understand the fundamental structure of sight. When we look at any image, there is a reason we can recognize what that image is, even if we have never seen it before. This thesis allows a glimpse of insight into this ability and a promise of better understanding of vision.

## **APPENDIX 1: CODE**

I have, in this thesis, opted not to include the full source in this print document as it is a significant amount of text, 12202 lines of code across 11 files as counted by LineStats 1.0. However, I am including the entire source as well as the Visual Studio build files on the companion CD. This is, I believe, important because while the documentation describes the algorithms in a fashion I feel is sufficient to implement, a reference implementation assures cross compatibility with other vendors implementations. I came to believe this during the thesis research itself. It seems to me that a few of the authors in the references section, and many other authors in my initial literature review, believe that something can be done and the solution is well known. However, they do not seem to care if a student or someone outside their realm of experience can understand how to make this well known fact actually work.



## APPENDIX 2: IMAGES



**FIGURE 9: FLUORESCENCE**

This is florescence. It is reproduced by Special Permission of Digital Blasphemy. Copyright \* by Digital blasphemy. I chose this picture as a counter example for Dr. Ford. I wished to point out that while there are distinct shapes and conceptually very few patterns (excellent for pattern reduction) there is something to be said when black isn't really black. As it turns out, however, this was the picture that was the most compressible.



**FIGURE 10: LENNA**

This is Lenna. It is reproduced by Special Permission of *Playboy* magazine. Copyright © 1972 by Playboy. It is the sole standard image in the field of image compression. Volumes have been written on why this is an appropriate image for the standard and I will not reproduce them for sake of brevity. It is included because all image

compression work has and should include it. Therefore, and there it no better way to put this, this is Lenna.



**FIGURE 11: ARUBA1**

This is aruba1. It was taken by Dr. Ford on his 2004 trip to Aruba. I selected this picture for its size and diversity of color. These are important variables to take into consideration when evaluating image compression techniques. As we can see from the picture there are several pieces of nearly identical images. Under the assumption of pattern reduction, these should allow for space savings.



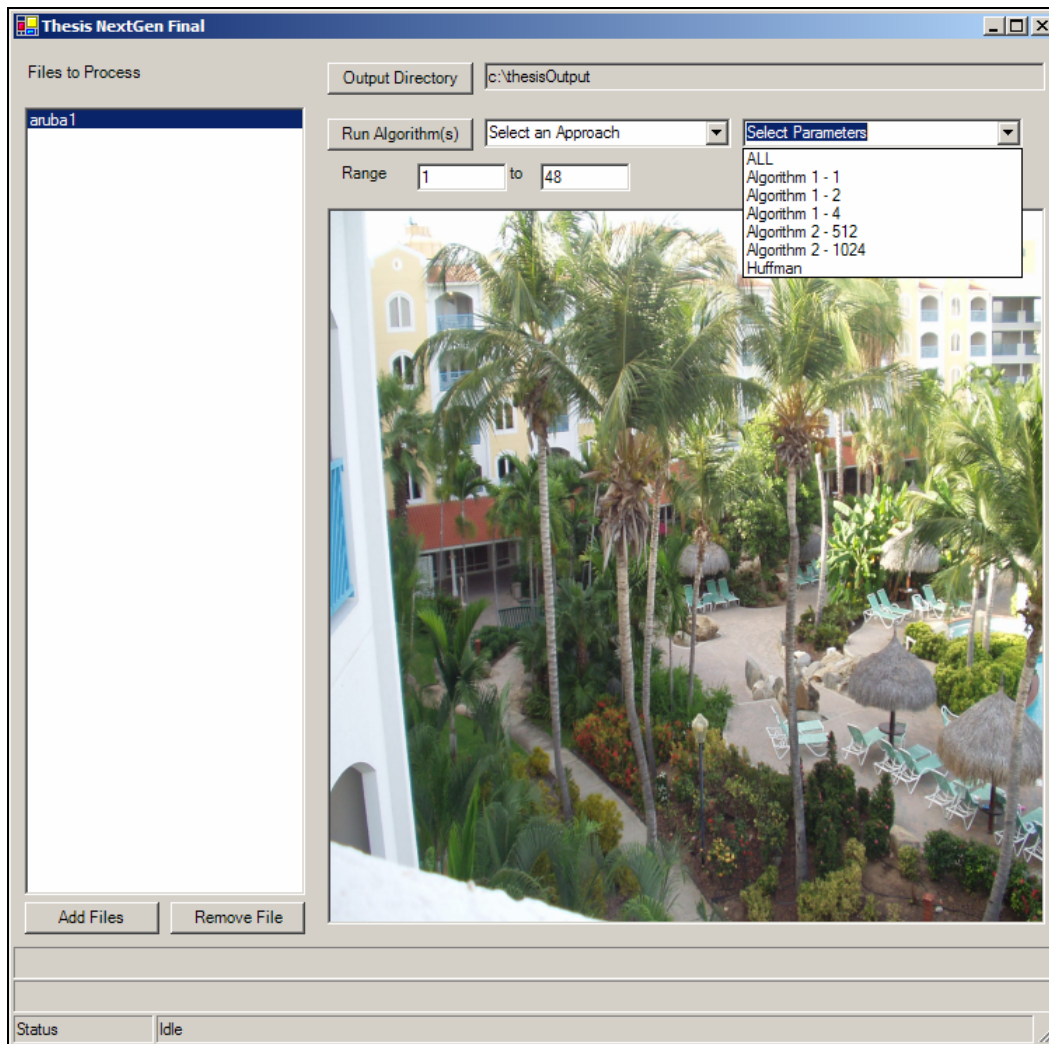


**FIGURE 12: ARUBA2**

This is aruba2. It was taken by Dr. Ford on his 2004 trip to Aruba. I selected this picture due to its lack of a color differential and high conceptual pattern level. As is obvious there are only 4 or 5 truly different things in this picture, but each of the individual pieces are themselves, highly differentiable. This demonstrates that within each pattern there are additional, or nested, sub patterns.

### **APPENDIX 3: SCREEN SHOTS**

This Section is self explanatory. It's a listing of all the screens I used in my test environment. I feel that this is a necessary inclusion to the thesis since the graphical user interface has become a ubiquitous part of computer programming even if it has just recently gained acceptance in main stream computer science research. However, I am also of the opinion that since this is a research thesis on image compression and not on graphical layout the presentation of the layout in its spartan state is sufficient.



**FIGURE 13: SCREEN SHOT**

This is a screen shot of the application. The interface is pretty cut and dry. There are no modal forms, other than the obligatory selecting files dialogs, so this is in fact the only screens the user sees.

## REFERENCES

1. The LOCO-I Lossless Image Compression Algorithm: Principals and Standardization into JPEG-LS, *in* HP Publication No. HPL-98-193R1. 1998, Hewlett Packard: Palo Alto, CA.
2. Advameg, I., JPEG image compression FAQ. 1999.
3. Cruz, D.S., et al., The JPEG-2000 still image compression. 2001, Joint Photographic Experts Group: Vancouver, BC, Canada.
4. Gurzick, D. and W.R. Ford. Using Pattern Reduction to Accommodate Variability of Expression in Natural Language Processing. *in* AVIOS 2006. 2006: Applied Voice Input Output Society.
5. Howard, P.G., The design and analysis of efficient lossless data compression systems, *in* Computer Science. 1993, Brown University: Providence, RI.
6. Huffman, D., A Method for the Construction of Minimum Redundancy Codes. *Proc. IRE*, 1952. **40**(9).
7. Kapoor, D., The div loss less image compression algorithm.
8. m.a.goldburg, et al., Application of wavelet compression to digitized radiographs. 1995.

9. *Marshall, D.*, Relationship between DCT and FFT. Available online at <http://www.cs.cf.ac.uk/Dave/Multimedia/node230.html>. 2001.
10. *Miano, J.*, Compressed image file formats: JPEG, PNG, GIF, XBM, BMP. 1999, Reading, MA, USA: ACM Press.
11. *Slattery, M.J. and J.L. Mitchell*, The Qx-coder. 1998, IBM.
12. *Wayner, P.*, Compression Algorithms for Real Programmers. 2000, San Diego, CA, USA: Academic Press. 240.
13. *Wu, X.*, An algorithmic study on lossless image compression, in Computer Science. 1996, University of Western Ontario: London, Ontario, Canada.



Yahoo! My Yahoo! Mail


 Welcome, **diputs\_mai**  
[\[Sign Out, My Account\]](#)

 Search the Web  
[Mail Home](#) - [Mail Tutorials](#) - [Help](#)

NOTEBOOKS AS LOW AS  
**\$499\*** [SHOP ONLINE NOW](#)

**LIMITED TIME ONLY**

[Mail](#) [Addresses](#) [Calendar](#) [Notepad](#)
[Mail Upgrades](#) - [Mail Options](#)






 Best MasterCard  
for bad credit

**Folders** [\[Add - Edit\]](#)
**Inbox (1)**
[Draft](#)
[Sent](#)
[Bulk](#) [\[Empty\]](#)
[Trash](#) [\[Empty\]](#)
**My Folders** [\[Hide\]](#)
[MyBulk](#)
[TAS](#)
[adnd](#)
[cmssc](#)
[digital](#)
[encode](#)
[frien](#)
[links](#)
[money](#)
[netscape](#)
[ntbugtrack](#)
[online orders](#)
[password](#)
[school](#)
[security](#)

 What's your Credit  
Score? See it FREE!

 Get unlimited calls to  
U.S./Canada

[Previous](#) | [Next](#) | [Back to Messages](#)
[Printable View](#) - [Brie](#)






 This message is not flagged. [\[Flag Message - Mark as Unread\]](#)

From tena chapman Tue Feb 15 16:43:40 2005

**X-Apparently-To:** diputs\_mai@yahoo.com via 66.218.93.119; Tue, 15 Feb 2005 16:43:39 -0800

**Authentication-Results:** mta106.mail.re2.yahoo.com from=digitalblasphemy.com; domainkeys=neutral (no sig)

**X-Originating-IP:** [207.97.221.96]

**Return-Path:** <tena@digitalblasphemy.com>

**Received:** from 207.97.221.96 (EHLO server2.digitalblasphemy.com) (207.97.221.96) by mta106.mail.re2.yahoo.com with SMTP 15 Feb 2005 16:43:39 -0800

**Received:** from [10.0.1.2] (evrtwa1-ar9-4-65-252-129.evrtwa1.dsl-verizon.net [4.65.252.129]) (authenticated bits=0) by server2.digitalblasphemy.com (8.12.11/8.12.11) with ESMTP j1G0hbit011096 for <diputs\_mai@yahoo.com>; Tue, 15 Feb 2005 19:43:38 -0500

**Mime-Version:** 1.0 (Apple Message framework v619.2)

**In-Reply-To:** <200502152308.j1FN8qaT007735@server2.digitalblasphemy.com>

**References:** <200502152308.j1FN8qaT007735@server2.digitalblasphemy.com>

**Content-Type:** text/plain; charset=US-ASCII; delsp=yes; format=flowed

**Message-Id:** <e61f551c39805b4689577ce262176661@digitalblasphemy.com>

**Content-Transfer-Encoding:** 7bit

**From:** "tena chapman" <tena@digitalblasphemy.com> [Add to Address Book](#)
**Subject:** Re: Copyright or Image Use Inquiry

**Date:** Tue, 15 Feb 2005 18:43:40 -0600

**To:** "mark newman" <diputs\_mai@yahoo.com>

**X-Mailer:** Apple Mail (2.619.2)

**Content-Length:** 971



Skip 6 mortgage  
Payments. How?

Hi Mark,

Yes, you may use Fluorescence as you described for  
your thesis.

Best wishes,  
Tena Chapman  
vp licensing & marketing  
digital blasphemy

On Feb 15, 2005, at 5:08 PM, mark newman wrote:

> Below is the result of your feedback form. It was  
submitted by  
> mark newman ([diputs\\_mai@yahoo.com](mailto:diputs_mai@yahoo.com)) on Tuesday,  
February 15, 2005 at  
> 18:08:52  
>

> -----

> email: [diputs\\_mai@yahoo.com](mailto:diputs_mai@yahoo.com)

>  
> COMMENTS: i would like to use one of ur images as a  
bentchmark for my  
> ms thesis on image compression. specifacly florence.  
may i include it  
> as a picture in my appendix? and on my thesis  
defence ppt?

>  
> ORIGIN: md  
>  
> SPEED: NOOPINION  
>  
> Submit: SEND IT!  
>  
> pleasereply: yes  
>  
>

> -----

> >  
> HTTP\_USER\_AGENT: Mozilla/4.0 (compatible; MSIE 6.0;  
Windows NT 5.1;  
> SV1; SALT 1.0.4223.1 0111 Developer; .NET CLR  
1.1.4322)  
>

Delete

Reply

Forward

Spam

Move...

[Previous](#) | [Next](#) | [Back to Messages](#)

[Save Message](#)

Check Mail

Compose

Search Mail

Search the Web

Copyright © 1994-2005 [Yahoo!](#) Inc. All rights reserved. [Terms of Service](#) - [Copyright Policy](#) - [Guidelines](#) - [Ad Feedback](#)

NOTICE: We collect personal information on this site.

To learn more about how we use your information, see our [Privacy Policy](#)

# PLAYBOY

March 24, 2005

Mr. Mark Newman  
Sonum Technologies, Inc.  
6700 Alexander Bell Drive, Suite 180  
Columbia, Maryland 21046

Dear Mr. Newman:

We are pleased to grant you one-time nonexclusive rights to include the head and shoulders "Lenna" image, which originally appeared in the November 1972 issue of *Playboy* magazine, in your thesis, Multi-Stage Pattern Reduction in lossless image compression.

This permission is contingent upon there being no derogatory references to Playboy, nor the subject matter of the photograph and neither will be depicted in an unfavorable light. The following copyright/credit notice must appear directly beneath the reprinted photograph OR on the credit page of your thesis:

Reproduced by Special Permission of *Playboy* magazine.  
Copyright © 1972 by Playboy.

Please note that permission is granted for one-time use only and that any other use or contemplated reuse will require, prior to use, further written permission from Playboy. Each reuse must be dealt with separately.

Thank you for your interest in Playboy.

Sincerely,



Diane Griffin  
Media Rights and Permissions Manager

/dg

g:Letters.05\Photo\Lenna\Newman\_Mark