12-2008

An Evolutionary Approach to Image Compression in the Discrete Cosine Transform Domain

Benjamin E. Banham
Utah State University

AN EVOLUTIONARY APPROACH TO IMAGE COMPRESSION IN THE DISCRETE COSINE TRANSFORM DOMAIN

by

Benjamin E. Banham

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Xiaojun Qi Major Professor	Stephen J. Allan Committee Member
Daniel W. Watson Committee Member	Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

An Evolutionary Approach to Image Compression in the

Discrete Cosine Transform Domain

by

Benjamin E. Banham, Master of Science

Utah State University, 2008

Major Professor: Dr. Xiaojun Qi
Department: Computer Science

This paper examines the application of genetic programming to image compression while working in the frequency domain. Several methods utilized by JPEG encoding are applied to the image before utilizing a genetic programming system. Specifically, the discrete cosine transform (DCT) is applied to the original image, followed by the zig-zag scanning of DCT coefficients. The genetic programming system is finally applied to the one-dimensional array resulting from the zig-zag scan. The research takes an existing genetic programming system developed for the spatial domain and develops DCT domain functionality. The results from the DCT domain-based genetic programming system are compared with those from the spatial domain-based system, and show improvements to the image quality with a reduction up to half of the evolved image's average error. The results show that working in the frequency domain has advantages over the spatial domain. Several methods to exploit these advantages are proposed and evaluated. (55 pages)

ACKNOWLEDGMENTS

There are many people who contributed to the success of this research. First of all, many thanks go to Dr. Xiaojun Qi for serving as my major professor and for providing so much helpful advice, suggestions, and improvements to this paper. Thanks also go to my other committee members, Dr. Dan Watson and Dr. Steve Allan, for serving on my committee, and for many quality courses throughout my education. I also wish to thank Dr. Don Cooley for serving as my advisor, helping to outline my coursework, and answering an endless stream of questions. To all of my professors at Utah State University, who helped me to learn a great deal over the years, I extend my appreciation. To all of these I extend my appreciation. Finally, I would like to thank my wife, Jennifer, for herding the kids away from me during study, and whose patience, support, and gentle reminders were indispensable to the success of my education.

I would also like to acknowledge and thank Matteo Frigo and Steven G. Johnson for making their powerful discrete Fourier transform library publicly available, as well as Tamas Szalay whose C# wrappers made it easy to use the library from C#.

Benjamin Banham

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Overcoming memory or disk space limitations is a familiar challenge in computing and will continue to be so, since disk space and memory cannot be unbounded. Adding hardware ultimately does not answer this growing need, since applications continue to demand increasing amounts of space. There is a cycle: as more space becomes available, data and media that previously could not be stored use all of the newly available space. To help alleviate space limitations, software allies with hardware by way of compression. Compression is a means of reducing the space requirements of data at the cost of increased computation time needed to access the data.

Several different approaches to compression exist, with a variety of strengths and computational requirements. The use of genetic programming (GP) to accomplish compression requires large amounts of computation power and time to achieve the compression. However, the time spent on GP compression is acceptable for many applications [8] since most compression techniques are asymmetric, and decompression time is minimal. The hope is that the large increase in time over conventional compression methods will pay dividends of better compression.

The basic principle behind GP is to have the computer evolve a program to perform the task that needs to be done, without telling it exactly how to do it [9]. For image compression, the task that needs to be done is to generate the approximation values to represent an image. The motivation behind applying the genetic programming technique to image compression is to generate a program which represents an image and then compress the program, potentially achieving better compression results over simply compressing the actual image [8].

1.2 Research Goal and Contributions

The goal of this paper is to add to this body of research with the unique contribution of using a GP system that works with a frequency domain representation of an image. This research also utilizes the advantages of a discrete cosine transform (DCT) for handling and reducing the impact of error. The experimental results show that applying GP in the frequency domain produces results that are at least as good as those produced by applying GP in the spatial domain, with the added benefit of more flexibility in how the data are handled. The following bulleted items list the major contributions of this research.

- Development of subdivision methods that apply the DCT on subimages, converting the image data into a frequency domain representation, and zig-zag gather the DCT coefficients prior to applying the GP.
- Development of a fitness calculation method which evaluates the GP population functions specifically on their ability to represent DCT coefficient data.
- Development of configurable GP parameters which take advantage of a frequency domain representation to improve the performance of the GP system.

1.3 Thesis Outline

Chapter 2 presents a brief literature review, wherein the existing approaches to image compression are summarized and the problems associated with these systems are also addressed.

Chapter 3 describes the GP system developed by Galloway [5] and the general framework of the proposed compression system in detail. A description of the experiments is also presented.

Chapter 4 presents the experimental results, wherein the quality of the approximated images generated by the proposed system is compared with that of the peer system developed by Galloway to illustrate the effectiveness of the proposed system.

Chapter 5 concludes the thesis with a brief discussion and summary of the proposed approach and some directions for future work.

CHAPTER 2

RELATED WORK

Many GP compression systems have been developed. Several relevant image compression systems and the techniques deployed in these systems are briefly reviewed here.

2.1 The Genetic Programming Technique

The genetic programming (GP) technique is a specialized genetic algorithm (GA) developed by John Koza [9]. In GP, each of the solutions or chromosomes in the population is a computer program or function. Each program of the population is run and the results generated are evaluated and assigned a fitness value according to how well the program performed the desired task. For example, a chromosome for GP that is trying to reconstruct an image might be assigned a fitness value that is the inverse of the difference between the original and reconstructed image. A chromosome that recreates an image with less difference from the original (less error) is given a higher fitness value. The symbolic representations (genes) of a chromosome in GP are the individual instructions that make up a program or function.

Genetic programming has good potential and applicability to the reconstruction and compression of images. As Koza points out, the problem space for GP may be highly nonlinear [9]. This feature is important for working with images, since a group of pixel values in an image to be reconstructed by GP may not have a linear relationship. The nonlinear problem space of GP is even more important in the frequency domain, wherein the frequency components of pixel values are less likely to have a linear relationship.

2.2 Applying GP to the Raw Image Data

Some methods of applying GP for image compression work directly with the image data, such as the approach proposed in [10]. The GP the authors developed creates functions whose input is simply the X and Y coordinates of a pixel, and the output was the pixel intensity. A GP often works with just a grayscale image so that there is only one intensity value. However, any such GP could be extended to a multidimensional image by working on each of the different color planes individually [10].

Nordin and Banzhaf [10] present several techniques to make their GP more effective. One method they use, which they term “chunking,” is to break up the image into equal-sized chunks and to develop programs to represent each chunk. As they pointed out, in all likelihood, GP do not converge on a solution that produces an acceptable image quality for a full-sized image since large entropy prevents a single program from adequately reproducing each pixel value. By chunking, the authors are able to work with smaller sets of data containing less entropy and converging on an acceptable solution more quickly. Another technique presented in [10] is termed a compiling genetic programming system (CGPS). This is different from a GP system that is interpreted at each step (for instance, using LISP). Instead, it directly manipulates the machine code to represent each program in a generation. Such representation reduces the time required to converge on a solution and, therefore, addresses the time-consuming issue of GP implementation. Finally, Nordin and Banzhaf observe that better compression results are achieved when the image data are represented in one dimension instead of two dimensions [10].

Many GP implementations seek to reduce execution costs [1, 4, 5, 7, 10, 11]. Galloway [5] developed a GP system similar to that proposed by Nordin and Banzhaf, but he applied the current technology of C# to get the speed benefits of using a compiled rather than interpreted language. The aforementioned methods all demonstrate that applying GP to the raw image data achieves promising compression results. Each of the methods show that getting the data in a form that is manageable to the GP technique is integral to the success of the system.

2.3 Improving Existing Methods

Improving a key compression source of an existing algorithm should lead to overall improved compression and/or quality. Therefore, several GP-based image compression techniques try to optimize a component of an existing, proven image compression system to improve the efficiency of the whole system. One example is Wu's approach [13], wherein he optimizes a single aspect of the proven and well used JPEG compression algorithm. Wu points out that the quantization table is responsible for a significant portion of JPEG's compression capabilities and control over quality. Consequently, he developed a GP algorithm to find an optimal quantization table for a specific image. Specifically, his proposed technique replaces the generic JPEG quantization table with the optimal one found by the GP and runs the otherwise untouched JPEG compression algorithm.

Jiang and Butler [11] present similar work in finding an optimal quantization table. Their research focused around the vector quantization in general. Their goal was to apply GP to converge on an optimal codebook representing the target image. They started with a small population of five randomly generated codebooks, and then measured how well each codebook represented the target image. The codebook with the smallest error was determined as the most fit. Their results show that the idea holds promise: it produced a 15% improvement over many of the common compression techniques at the time the experiments were performed.

Fukanaga and Stechert also apply GP to an aspect of a developed compression technique [4]. Their work optimizes a predictive coding compression algorithm. Working from a nonlinear predictive model, the value of each pixel is predicted based upon the values of its eight neighbors. The error image (the image formed from the difference between the original image and the predicted image) is Huffman encoded and stored. The pixel values around the edges of the image are also stored so that there are some initial "neighbors" to work with. This algorithm lends itself very well to GP, as it finds an optimal predictor such that the error image contains minimal entropy and can be highly compressed.

These methods illustrate how the utilization and consideration of high performing compression methods increase potential for achieving good compression results. Each of these methods attains promising results by a combining current compression technology with genetic programming.

CHAPTER 3

PROPOSED GENETIC PROGRAMMING SYSTEM

The genetic programming system used for this research was built upon Galloway's engine. A detailed overview of Galloway's system can be found in [5]. Here, a brief overview of Galloway's system is presented along with details of the changes incorporated for this research.

There are several advantages to building on Galloway's system. First, it is written in C#, which combines the flexibility of an interpreted language for generating code on the fly with the speed of compiled code. Second, the image data are broken up and represented in a similar manner as Nordin's chunking scheme [10]. This representation is highly suitable to the necessary preliminary steps for applying the DCT and working in the frequency domain. Third, the current design allows for adding new functionality without requiring an overhaul of existing functions and classes. That is, it allows for the definition of new functionally by utilizing the appropriate hooks. Fourth, building upon an existing system allows for a fair comparison between the results of the two data representation domains.

3.1 Overview of Galloway's System

There are three main classes for the system: Generation, GPUtilities, and GPConfiguration. The Generation class provides the functions to initialize the first generation of functions, generate the next generation (including mating the programs to create offspring), evaluate, and compile the generation. GPUtilities provides many utility functions to the system including gathering statistics and creating the image represented by a generation. It also provides the functions for representing an image in multiple quad-tree-based blocks, the functions for evaluating the fitness of a generation, and even the signatures for the functions to be evolved. GPConfiguration is a collection of all the different configurable options, including mate selection methods, mutation probability, and the individual operations from which the evolved functions are constructed.

From these three major classes, the system is able to perform the core signature steps of any genetic programming system. It first sets up all of the desired GPConfiguration options, instantiates a Generation object and calls its initialization method, and finally sets up a loop inside of which the Generation object's function for generating the next generation is called. During the loop, the GPUtilities object is used to generate statistics for the evaluation and is called upon by the Generation object to evaluate the population fitness.

3.1.1 Image Subdivision

Before actually jumping into the loop of generating new populations and evaluating fitness, Galloway's system breaks up the input image into smaller blocks. This is done either by creating fixed-size blocks of some maximum size, or by generating a quad-tree [5], which breaks up the image into blocks based on the complexity of the block. That is, areas of an image with lower complexity are broken into larger-sized blocks. The image is next represented in a tree form, with individual nodes storing the subimage data, which have been either uniformly divided or quad-tree divided. The fixed tree form (i.e., the subimage data have been uniformly divided) is just a quad-tree with every single node being the same size. In this case, all of the terminal nodes are at the same level on the tree.

This subdivision of the image is important for the application of the DCT. It is a distinction of the JPEG encoding algorithm, and provides a more manageable set of data to be reconstructed by the genetic programs.

Unlike most existing GP methods, which generally work on a small-sized single image or only a portion of a larger image, Galloway's system applies the GP to a collection of subimages. These subimages, which have varying sizes based on the uniformity measure, are further represented by the same GP program. In addition, Galloway's system develops a population of functions that can represent multiple images. Some side effects of this feature are discussed in Section 3.2.1.4.3.

3.1.2 Calculating Fitness

As mentioned in the previous subsection, the GP system evolves a population of functions to multiple target subimages. This design requires a unique method to calculate the function fitness. Galloway tackles this issue by assigning two fitness values to each function. The first fitness value is a priority, which is simply a count of the number of nodes that use the function as its best fit. That is, if a node chooses a function as the most capable of representing its data, the priority of the function is increased by one. The second fitness value is the function fitness, which is calculated as the function's best representation of any subimage measured by the mean square error between the subimage and its estimation. A function's overall fitness is first calculated by its priority, and the best node fitness is considered in the case of equal priorities. Specifically, if a function is the best representation of more nodes than another function, it is kept.

3.1.3 Configurable Function Signatures

Galloway's system tests different forms of evolving functions to comprise the population. One function takes two parameters, the *x* and *y* coordinates of the pixel values. The other function takes seven parameters, the *x* and *y* coordinates of the pixel value, the four corner pixel values of the subimage block, and the size of the image block. As a result, his system allows the population functions to be configurable. This configurable feature is a benefit to the future development of the system. One of the goals of this research is to test the effectiveness of using a one-dimensional array of data based upon 1) the fact that the zig-zag-gathered frequency components are in that form and 2) Nordin's [10] observation that compression results are better when the data are represented one-dimensionally.

3.2 The Proposed System

In this research, we propose to apply a configurable GP method to the one-dimensional zig-zag DCT coefficients of varying subimages to achieve comparable compression results. An illustration of how the different components of the proposed system are connected is presented in Figure 3.1. The following subsections detail each component of the proposed system.

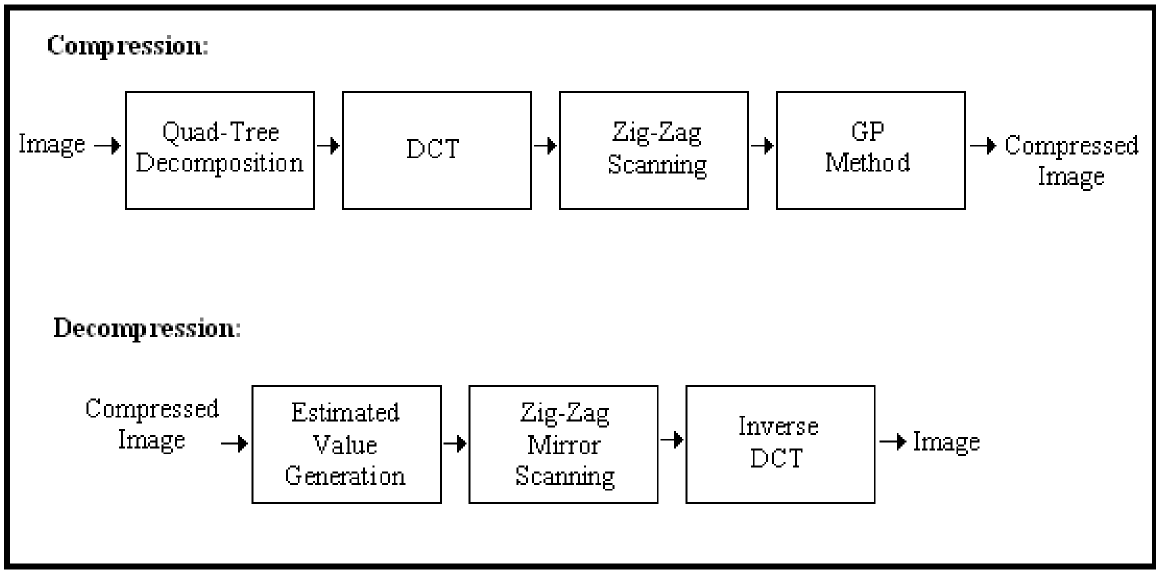


Figure 3.1. Block diagram of DCT-based GP system compression and decompression.

3.2.1 Compression Scheme

3.2.1.1 Quad-Tree Decomposition. The first step of the compression scheme, quad-tree decomposition, is identical to Galloway's system. The quad-tree decomposition can be configured to either break up the image into fixed-sized subimages, or to break it up into varying-sized subimages based on image complexity.

3.2.1.2 Discrete Cosine Transform (DCT). Once the image has been broken up into more manageable subimages, the data are ready for the second step which transforms the data into the frequency domain. We apply the DCT to get the data into the frequency domain. This transform was chosen because it is the one used by the JPEG compression algorithm [6, 11, 13], and because of its ability to obtain fewer data, thus allowing for more efficient encoding [6].

Like other transforms, DCT is based on Fourier's work showing that a function can be represented as a sum of sines or cosines of different frequencies [6]. Each of the

DCT coefficients produced by applying the transform represent factors that can be multiplied to cosines of differing frequencies and summed up to regain the original spatial domain data. This frequency domain representation has the beneficial quality of compacting information into the lower frequency components for image data, which is a central reason it is used in image compression techniques. Figure 3.2 shows an example of pixel intensity values from a 4-by-4 subimage, along with the coefficients produced by applying the DCT. Lower frequency information is located at the upper left corner, and higher frequency information is located at the bottom right corner, as explained in Section 3.2.1.3.

Unfortunately, like most programming languages, C# does not provide a built in function for this method. As a result, we created and implemented the DCT class to fill this language void. The DCT class makes use of the fastest Fourier transform in the West (FFTW) library, which is written in C [2, 3]. The choice to use this library was based on the fact that it is free, it is licensed by the Gnu Public License (GPL), and it is fast. The FFTW achieves performance comparable to commercial transform libraries by dynamically tuning the transforms to the hardware upon which it is being executed; hence, the program runs as quickly and efficiently as possible [2]. As pointed out

```
175 169 169 172    304 -1 12  0
171 167 167 171     25  3  0  0
166 162 163 168     -4  0  0  0
160 156 157 163      1  0  0  0
       (a)                 (b)
```

Figure 3.2. Example DCT Application. (a) The original subimage pixel intensity data. (b) Its frequency domain representation by applying the DCT.

previously, speed is a very important feature when working with a genetic programming system.

The DCT class also makes use of a library developed by Tamas Szalay [12] which provides C# wrappers around the FFTW C functions. This library makes utilizing the C library from C# painless, and keeps the code in the DCT class cleaner.

The main components of the DCT class are comprised of two functions and a constructor. The constructor takes an integer argument that specifies the size N of an N-by-N array of data, allocates the memory, and sets up the plans required by the FFTW library. An FFTW plan is a specification of a transform type and sets up the chosen transform to perform the dynamic tuning of the transform to the hardware. Once the memory and plan have been set up, the transform is ready for use. The other two functions perform the forward and backward DCT. Each function has to copy the data from the C# managed arrays to unmanaged arrays useable by FFTW, invoke FFTW's forward or reverse (inverse) transform, and finally copy the values out of the unmanaged array to a C# array. Both of these functions take the array of values as arguments to be transformed as well as the total number of elements in the two-dimensional array (i.e., the size of one dimension of the two-dimensional array). A DCT transform is performed by instantiating a DCT object and calling an execute function on the target data to perform either the forward or reverse transformation. All these make performing the DCT as easy and fast as if the DCT were built into the language.

The forward DCT is performed right after the image has been broken up into smaller blocks in the quad-tree decomposition step. This choice was made to reduce the amount of times the DCT operation needs to be performed, since it is a relatively time consuming operation. Otherwise, the DCT would need to be performed for each node every time the generation of functions was evaluated for fitness.

3.2.1.3 Zig-Zag Scanning. Once the data is transformed into the frequency domain by the DCT, it is next reordered into a one-dimensional representation. As previously mentioned, one benefit this research sets out to take advantage of is the improved results given by a one-dimensional representation of the data [10]. The JPEG encoding scheme provides the perfect method to attain a one-dimensional representation by using a zig-zag scanning order, as illustrated in Figure 3.3, which starts at the direct current (DC) coefficient (i.e., the coefficient located at the top left corner) of the DCT transform. This scanning order of collecting the data takes advantage of the long runs of zeros that typically occur in higher frequency DCT coefficients [6]. These runs of zeroes are a major contributor to JPEG's high compression ratios. They also allow a frequency-based GP system to reduce the amount of work required for evolving the population. Section 3.2.1.4.4 discusses some potential ways to take advantage of this representation.

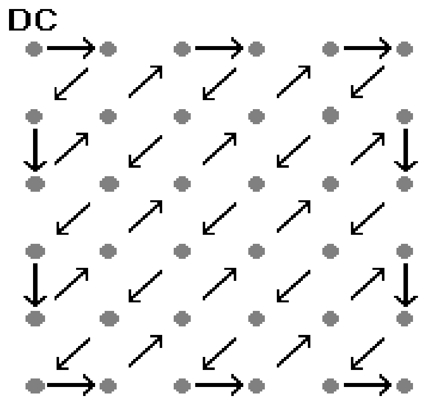


Figure 3.3. Illustration of the zig-zag scanning order.

A function for gathering data using the zig-zag scanning order was implemented and added to the GPUtilities class. This function takes a two-dimensional array of data and its size as arguments and outputs a one-dimensional array of the zig-zag-collected data.

3.2.1.4 GP Method. Once the data is in a one-dimensional array, the GP algorithm begins evolving functions to represent it. The GP algorithm contains the basic sequence of population initialization, evaluation, and mating common to any GP system. The core details of the algorithm are detailed in [5]. Here, the modifications important to this research are detailed.

3.2.1.4.1 Design Differences for the Frequency Domain. Evolving GP functions in the spatial domain is different from evolving GP functions in the frequency domain. In the spatial domain, each subimage is almost homogeneous. That is, all the data in the spatial domain are closely related to each other. However, all the data in the frequency domain show less correlation and display more entropy. As a result, we removed two interpolation functions (e.g., Interpolate and InterpolantFunc) from Galloway's system and added three functions, which are described in Table 3.1. These functions are among the simple building blocks used by the GP system to generate functions. Besides the inclusion of these building block functions, the GP system's population functions are composed of simple arithmetic operations and comparison operators as detailed in [5].

3.2.1.4.2 Bounding Possible Output. It is intuitive that allowing the individual GP programs to generate values which are unbounded or at least bounded only by the upper and lower ranges of a certain data type would lead to a very inefficient GP system

Table 3.1. Added GPConfiguration Building Block Functions.

Function name	**Possible Values**	**Description**
`Sin256`	`0-255`	Takes amplitude, period, and position parameters and returns a truncated integer sine value. The modulus 256 is performed on the amplitude parameter to attain values below 256.
`Modit256`	`0-255`	Returns modulus 256 of the input parameter.
`Modit128`	`0-127`	Returns modulus 128 of the input parameter.

design. In other words, it is not reasonable to allow the generation of values in the range of 0 to 65535 if the actual range of values is 0 to 255, as in the case of an 8-bit grayscale image. In Galloway's research, 0 to 255 was the allowable range for the grayscale pixel values, so his bound was immediately visible. The bound for the DCT coefficients was not as readily available, so we needed to determine a reasonable maximum for DCT coefficients.

To determine the bound on the DCT values, we randomly generated millions of two-dimensional arrays of varying size of M-by-M, where M is divisible by 4 due to the use of the quad-tree subdivision, and the values in the two-dimensional array range from 0-255 to simulate the case of an 8-bit grayscale image. We then applied the DCT to them, and analyzed the range of the DCT coefficients. The largest magnitude encountered was 653. This occurred when the size of the array was larger than 64-by-64. Since the block size of larger than 64-by-64 will not likely occur in our quad-tree subdivision, and since a value over 600 was rarely encountered even in larger block sizes, the initial bound was set to 600. We also added precaution code to pop up a message window to alert if an actual DCT coefficient value from a subimage exceeded this amount. As a result, a new configuration variable was added to the GPConfiguration class to set the maximum bound for DCT coefficients to be 600. This value was used in the code as a modulus to the output of the evolved functions. Thus, the range of values producible by the evolved functions was -600 to 600 (the modulus is taken with 1 plus the bound so that this top value is also attainable).

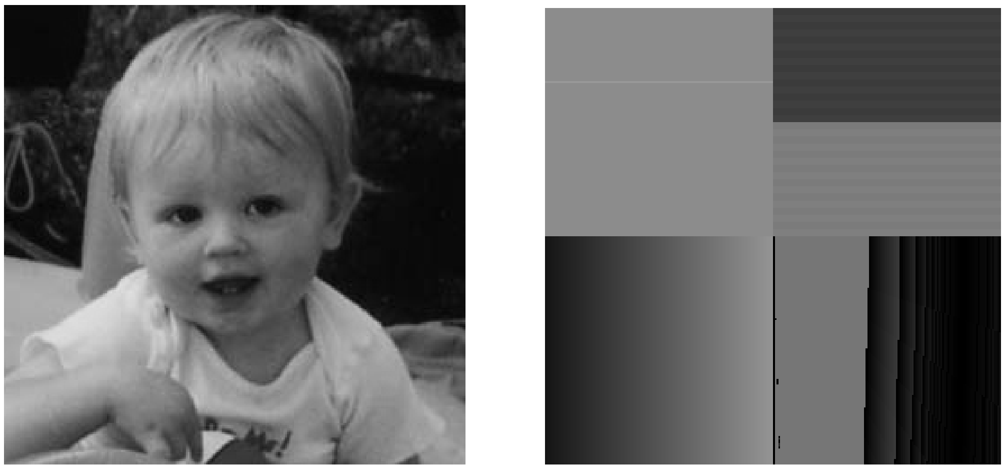
This worked well for most cases. However, there was some wasted effort in the GP evolution process, since the range from -600 to 600 was not the actual range for the DCT coefficients. To resolve this, the maximum DCT coefficient magnitude was calculated as the DCT was applied to each of the subimages. The largest DCT coefficient of all subimages was recorded, which required an extra 16 bits of data to be stored with the final results. However, these extra bits were considered inconsequential compared to the potential of a gain in quality or compression.

3.2.1.4.3 Improving System Performance. Testing the Efficiency of the GP System. In order to uncover any shortcomings in Galloway's system, a simple experiment was performed, wherein we tested whether Galloway's GP system could perfectly evolve an image that it had previously created. In other words, a test was performed to see whether the GP system could evolve the same functions it had evolved in a previous run.

An image was put into Galloway's GP system with the constraint that it could only be subdivided into four fixed-sized blocks before it was evolved. This constraint was designed to overwhelm the systems capabilities, and therefore create an image significantly less complex than the original. Figure 3.4 shows the original "Julie1.bmp" image and the resulting image evolved under this constraint.

Once the test image had been created, it was put back into the GP system to see if it could recreate its previous product. The system failed to recreate the image. An examination of the population's function fitness values revealed that nearly all of them represented the bottom left subimage of Figure 3.4(b) without error. This shortcoming of Galloway's system is due to the following reasons: 1) The system wastes time evolving multiple functions for a subimage that has already found one function to perfectly represent it; and 2) The system causes the evolving functions to converge onto the least complex subimages, and therefore the more complex subimages are not well represented. The next section discusses the code changes we made to fix this problem.

Fitness Calculation Improvements. In view of the outcome of the simple test described above, we proposed two improvements to the GP system. Previous GP systems that evolve only a single, small image can effectively use the fitness calculation designed by Galloway. However, Galloway's system needs a more representative



(a) (b)

Figure 3.4. Results of the GP efficiency test. (a) The original "Julie1.bmp". (b) Its evolved representation by applying the GP system on four equal-sized subimages.

calculation. First, if a subimage is being perfectly represented by a function of the population, the remaining population should not be applied to it. Not applying the remaining population decreases the number of calculations necessary during the population fitness evaluation, and frees up the rest of the population functions to converge on representations of the other subimages. Second, the function should not be assigned a fitness that measures how well it represents a single node. Instead, it should be assigned a fitness that measures how well it represents all of the nodes. In other words, the GP system should prefer functions representing many nodes over those representing only a few. This fitness calculation requirement results from the complex burden the GP system has of evolving many functions to represent many subimages.

To implement the first improvement, a small code change was made to the GPUtilities population evaluation function to short circuit further evaluation of functions on a subimage when the data was already perfectly represented by one of the functions. The second improvement was also implemented in the GPUtilities population function and was made by summing the fitness calculations for a function as it was applied to each of the subimages in the quad-tree.

3.2.1.4.4 Reducing the Workload. We made two major observations made during research and experimentation that led to a few ideas for reducing the amount of work the GP performs when finding functions to represent subimage DCT coefficients. First, when introducing an error into a set of DCT coefficients after function estimation, after taking the inverse DCT to regain the pixel values, the error is spread out among the pixel values and not just centralized to one place. Second, the DCT usually results in long runs of zeros in the higher frequency components [6]. Table 3.2 summarizes the added three parameters, namely DCTQuantize, DCTEvolvePercent, and AdjustErrorByAvgDifference, to reduce the workload of the GP system. The optimal values for these parameters were determined by experimentation, which is described and discussed in later sections.

Parameter DCTQuantize. The JPEG encoding algorithm takes advantage of both of the above observations when applying the quantization step. Here, each of the DCT coefficients is divided by the corresponding value in a defined quantization table. This does introduce some round off error, but it is virtually unnoticeable in the reconstructed image. The algorithm is also able to create even longer runs of zero by eliminating very small DCT coefficient values.

In this research, we utilized the concept of the quantization as employed by the JPEG encoding algorithm. However, we were not able to make direct use of the same quantization method due to our varying block sizes and the fixed 8-by-8 quantization

Table 3.2. Added GPConfiguration Parameters.

Property Name	**Possible Values**	**Description**
`DCTQuantize`	`> 0`	This is an integer value that specifies the amount to quantize each DCT coefficient.
`DCTEvolvePercent`	`0..1`	This value will vary the number of DCT values to be evolved. Only the first n percent of the values will be worked on, while the rest are just set to 0’s.
`AdjustErrorByAvgDifference`	`True / false`	This tells the fitness evaluation function whether or not to generate and adjust by an average error. Doing so will reduce the overall error between the original subimage values and the estimated values, bringing the estimated values closer to the actual values.

table used by JPEG. Instead, we used the quantization idea as a base to develop a simpler quantization method. It works in this way. After applying the DCT on the original subimage, a configurable quantization amount is applied to quantize each DCT coefficient. The resultant DCT coefficients have a smaller range for the GP to estimate. For example, if the configurable quantization value is set to be 10 and the original range of DCT coefficients is from -600 to 600, the new quantized DCT coefficients range from -60 to 60. It is easy to see that the GP takes less effort to estimate this smaller range. As a result, we added a new integer parameter (i.e., DCTQuantize) to the GPConfiguration class to specify the amount of quantization to perform. A value of 1 will performs no quantization.

Parameter DCTEvolvePercent. Introducing the DCTEvolvePercent parameter was motivated by the observation that the majority of high frequency components are zero at the far end of the array of DCT coefficients. Therefore, it would be better to skip estimating them, given that they are zeroes, and the evolved functions might assign unneeded error.

A configuration parameter (i.e., DCTEvolvePercent) was added to specify the percentage of DCT coefficients to evolve. The DCT coefficients falling into the specified percent are evolved by the GP functions, and the remaining DCT coefficients are set to 0. This gives a performance boost in terms of execution speed since fewer values need to be evaluated. In addition, it has the potential of reducing errors that might be introduced in the runs of zeros.

Parameter AdjustErrorByAvgDifference. Another observation we made was the possibility of some good estimators being thrown away because they were off by large estimation errors even though they represent the subimage well (i.e., the estimated and the actual images resemble each other, except one of them is much brighter or darker than the other). To address this, we stored the average error between the best function's estimation and the subimage data. This value is calculated and stored for each subimage of the quad-tree. Figure 3.5 illustrates this idea, wherein an estimation of the values is off by roughly the same amount for each pixel. The figure clearly shows that adjusting by the average error potentially helps the GP more quickly find good representative functions, and helps increase the number of subimages represented per function. For example, if one function output has values 3, 3, 3 and another output has values 7, 7, 7, only one of those functions is necessary when using an adjustment value, since the same output can be achieved by the appropriate adjustment.

As a result, we added a boolean parameter (i.e., AdjustErrorByAvgDifference) to the GPConfiguration class to signal whether the GPUtilities fitness evaluation function should calculate and store the average error to adjust the estimation.

3.2.2 Decompression Scheme

Technically, decompression is not a part of the GP compression system. However, when presenting a compression scheme, it is important to describe how the decompression is done as well. Also, the decompression steps are called by the last instruction of the GP engine's next generation/evaluation loop to update the GUI, thus showing the progress of the compression. As mentioned in the introduction, the

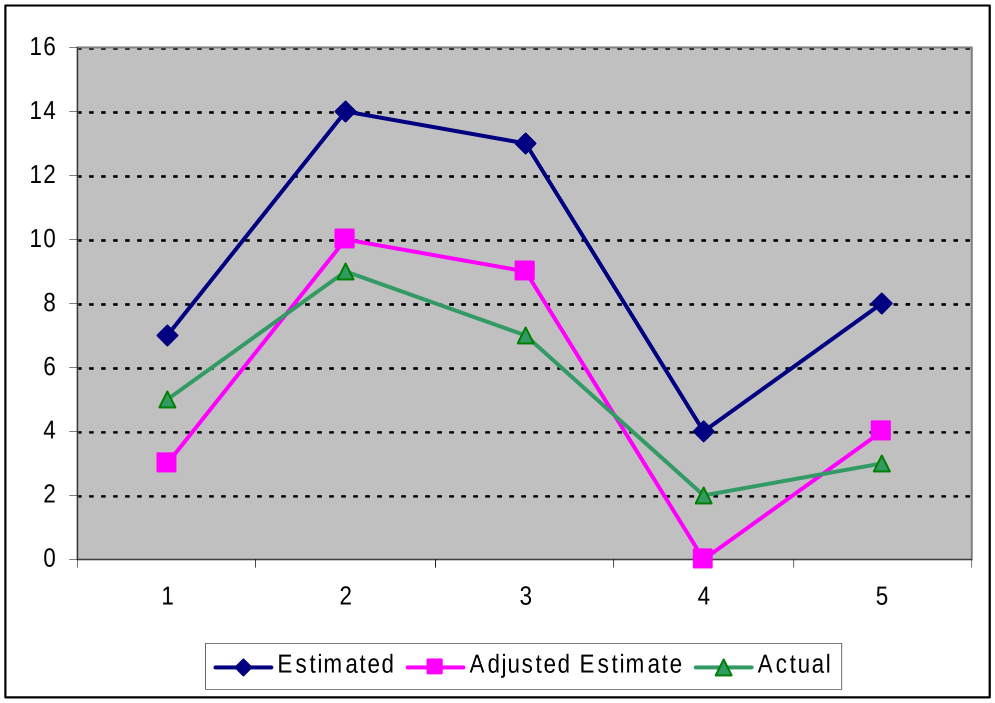


Figure 3.5. Illustration of adjusting by average error to more closely match actual data.

application of GP to image compression is an asymmetric technique. The time it takes to compress an image is measureable in hours, while the time to decompress is measureable in milliseconds. The previous subsections detailed several time intensive steps required for the compression. However, the decompression scheme is significantly faster and simpler. It performs the reverse of the compression scheme steps. The following subsections detail each component of the decompression.

3.2.2.1 Estimated Value Generation. The first step of the decompression scheme is to generate the estimated values by calling the evolved function with the appropriate parameters. For this research, the data was represented as a one-dimensional array while the functions were evolved so the input parameter is the index into the one-dimensional array. To generate the estimated values, an integer is looped from 0 to 1 less than the number of pixels in the subimage and passed into the function which returns the estimated DCT coefficient at the index corresponding to the input integer.

3.2.2.2 Zig-Zag Mirror Scanning. The next step of the decompression scheme is to get the DCT coefficients from the one-dimensional array back into a two-dimensional array. In order to do this, a “mirror” function is necessary to un-zig-zag the collected data. This function does the inverse of the zig-zag function, taking a one-dimensional array as an argument and outputting a two-dimensional array of the restored data.

3.2.2.3 Inverse DCT. The inverse DCT is next performed to restore (i.e., transform) the DCT coefficients to pixel intensities. This last step takes the un-zig-zagged values produced by the evolved functions to produce an estimation of the original image, which should very closely resemble the original image.

CHAPTER 4

TEST RESULTS

The goal of this paper is to show that applying GP in the frequency domain can improve compression results. Further, the frequency domain provides flexibility in how the data can be handled, which could provide further performance gains. To support this thesis, two sets of experiments were performed which we refer to as experiment set A and experiment set B. Experiment set A evaluated the performance of the new configuration parameters introduced in Subsection 3.2.1.4.4 for reducing the GP workload. Experiment set B compared the performance of the modified frequency domain GP system against the original spatial domain system. All experiments were performed with the same images and configuration options outlined by Galloway [5], in order to provide a more accurate and fair comparison of the GP systems. All of the experiments evaluated the improvement in image quality, which was measured by calculating the mean error between original and approximated images, and by visual inspection where appropriate. A description of experiment set A is provided in Section 4.1 with the results immediately following in Section 4.2. Similarly, a description of experiment set B is provided in Section 4.3 with the results immediately following in Section 4.4.

4.1 Experiment Set A: Configuration Parameter Tests

The hypotheses and their implementations presented in Section 3.2.1.4.4 were proposed as ways to boost the performance of the frequency domain-based GP system. Two of the configuration parameters, DCTQuantize and DCTEvolvePercent, take advantage of the characteristics of a frequency domain representation. These parameters have the potential to improve performance by reducing the data the GP system needs to process and reduce the likelihood of error. The third parameter works off the principle that some good functions in the population can be saved by adjusting their output to more accurately estimate the target data.

The value combinations listed in Table 4.1 were created to test the effectiveness of these three parameters. These combinations were developed to test several values within each parameter's range, including the value that essentially turns each parameter on or off (i.e., enabled or disabled). A disabled parameter means its value has no effect on the system. For example, the value 1 for DCTQuantize indicates that the quantization is not used and all the DCT coefficients are kept the same. The value 1 for

Table 4.1. GPConfiguration Values for Experiment Set A.

AdjustErrorByAvgDifference = false		**AdjustErrorByAvgDifference = true**	
DCTQuantize	**DCTEvolvePercent**	**DCTQuantize**	**DCTEvolvePercent**
1	0.4	1	0.4
1	0.6	1	0.6
1	0.8	1	0.8
1	1.0	1	1.0
2	0.4	2	0.4
2	0.6	2	0.6
2	0.8	2	0.8
2	1.0	2	1.0
5	0.4	5	0.4
5	0.6	5	0.6
5	0.8	5	0.8
5	1.0	5	1.0
10	0.4	10	0.4
10	0.6	10	0.6
10	0.8	10	0.8
10	1.0	10	1.0

DCTEvolvePercent indicates that all the DCT coefficients are evolved.

In order to provide a wider range of conditions and to test the robustness of these parameters, four sets of experiments were created by using two images of different sizes as well as using both the fixed and varying-sized subimage representations of those images. These tests are summarized in Table 4.2. For all experiments in this set, the number of generations and the population size were both set to 500. This value was chosen as a good balance between enough functions to effectively represent the image and a shorter running time to facilitate running many experiments.

4.2 Experiment Set A Results

Figures 4.1 through 4.4 show the results of experiment set A. Each of the image subdivision methods of each input image are presented individually. Specifically, Figure 4.1 presents the experimental results for image Julie1 represented in a fixed block

Table 4.2. Parameters for Experiment Set A.

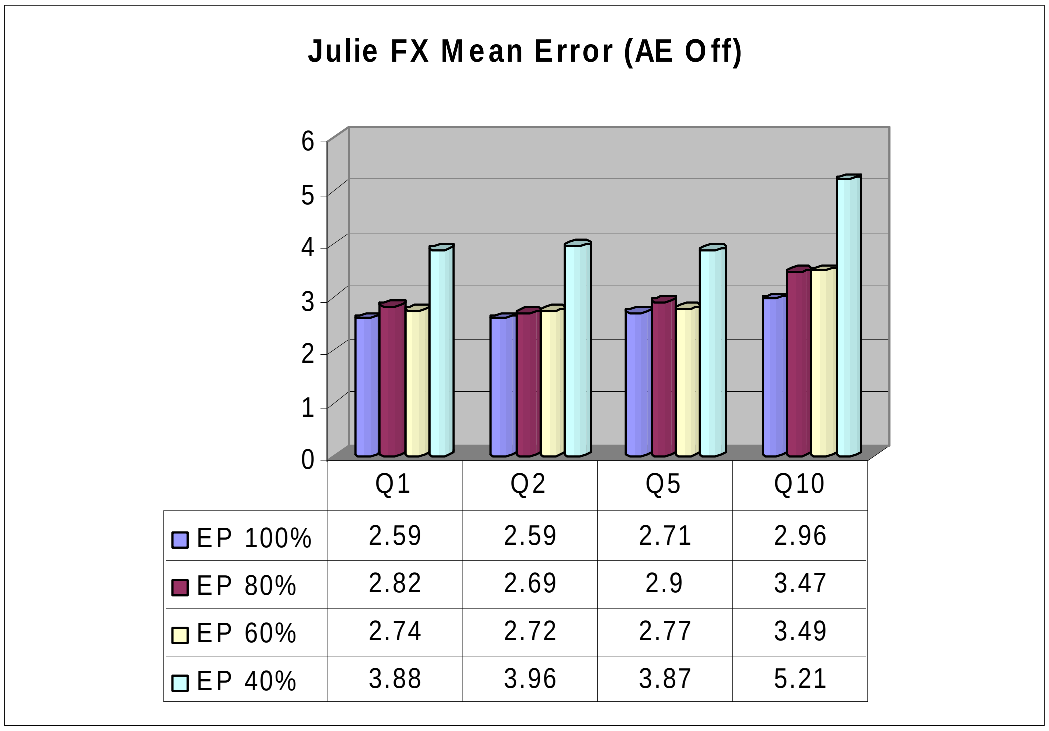
Test Name	**Image**	**Subdivision Method**	**Number of Generations**	**Population Size**
A_Julie1_FX	Julie1 (256-by-256)	Fixed blocks 4x4 block size (4096 blocks)	500	500
A_Julie1_QT	Julie1	Quad-tree 4x4 min block size (<4096 blocks)	500	500
A_F16_FX	F16 (512-by-512)	Fixed blocks 4x4 block size (16384 blocks)	500	500
A_F16_QT	F16	Quad-tree 4x4 min block size (<16384 blocks)	500	500

subdivision form. Figure 4.2 presents the experimental results for image Julie1 represented in a variable size quad-tree form. Similarly, Figures 4.3 and 4.4 shows the experimental results for image F16 represented in a fixed block form and in a variable size quad-tree form, respectively.

The data presented for the experimental result is the mean error between the original and the approximated image. Both a table of the data and its graphical depiction are provided. The labels are abbreviations for evolution percent (EP) representing the DCTEvolvePercent parameter, quantization (Q) representing the DCTQuantize parameter, and adjust error (AE) for the AdjustErrorByAvgDifference parameter.

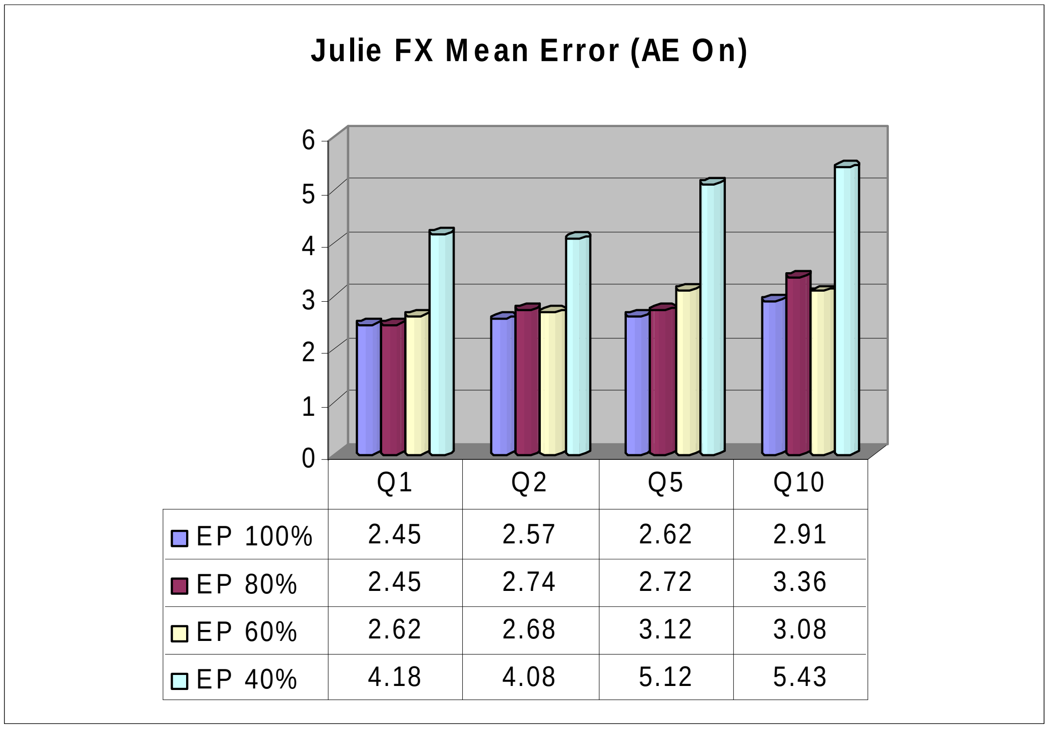
4.3 Experiment Set B: Frequency Domain Versus Spatial Domain

We conducted an experiment comparing the results of the frequency domain-based GP system against the spatial domain-based system to justify the premise of this research. This experiment set, experiment set B, was designed to produce results directly comparable to Galloway's final results. The same original images, population sizes, and number of generations in Galloway's experiments were used in this experiment set, and are summarized in Table 4.3. Table 4.4 summarizes the values of the added configuration parameters we set. These values ensure that the choices for these parameters are essentially disabled. That is, the quantization is not used, all the DCT coefficients are used for evolving, and no adjustable errors are recorded.



	Q1	Q2	Q5	Q10
🟦 EP 100%	2.59	2.59	2.71	2.96
🟥 EP 80%	2.82	2.69	2.9	3.47
🟨 EP 60%	2.74	2.72	2.77	3.49
⬜ EP 40%	3.88	3.96	3.87	5.21

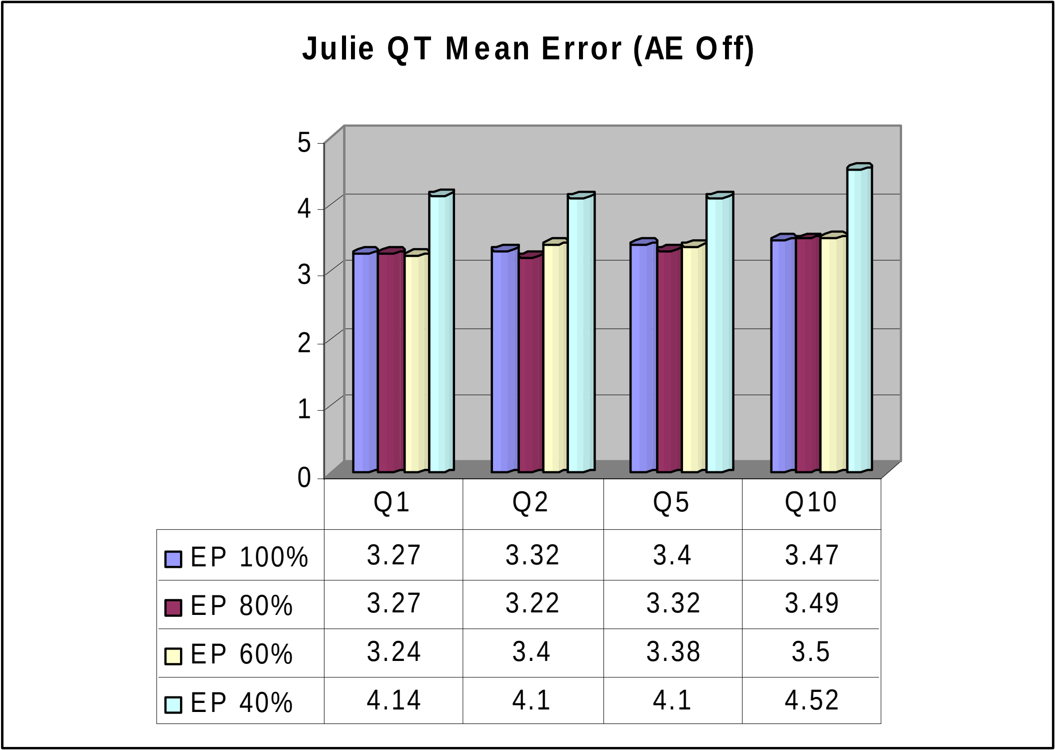
.



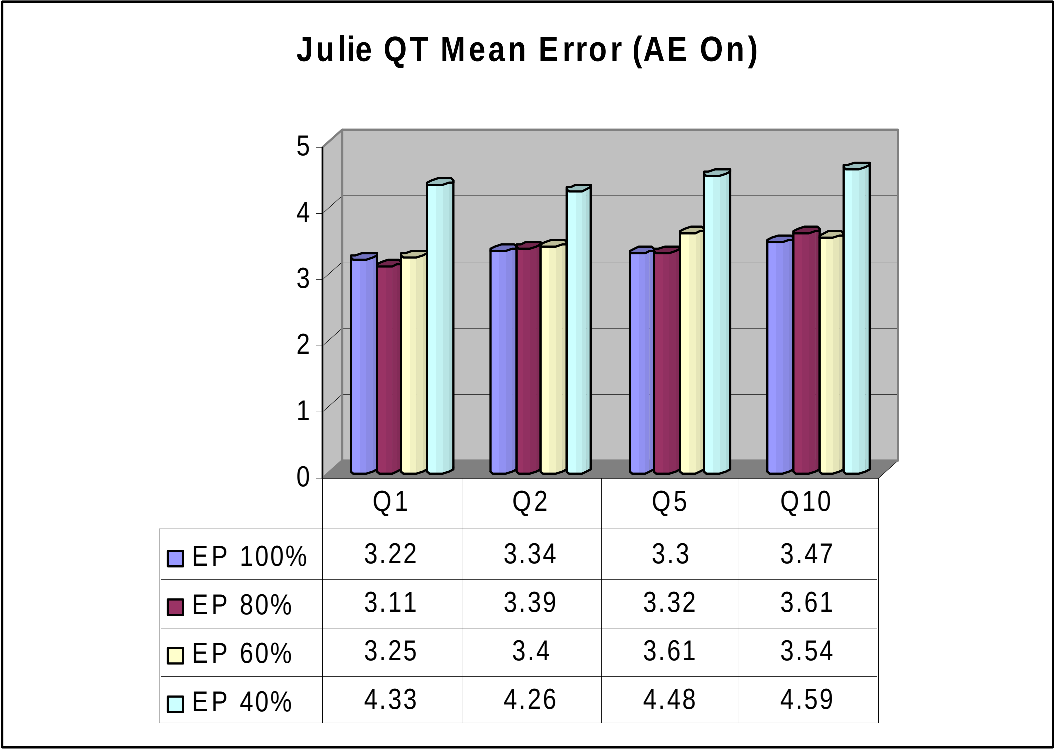
	Q1	Q2	Q5	Q10
🟦 EP 100%	2.45	2.57	2.62	2.91
🟥 EP 80%	2.45	2.74	2.72	3.36
🟨 EP 60%	2.62	2.68	3.12	3.08
⬜ EP 40%	4.18	4.08	5.12	5.43

.

Figure 4.1. Results for experiment A_Julie1_FX.

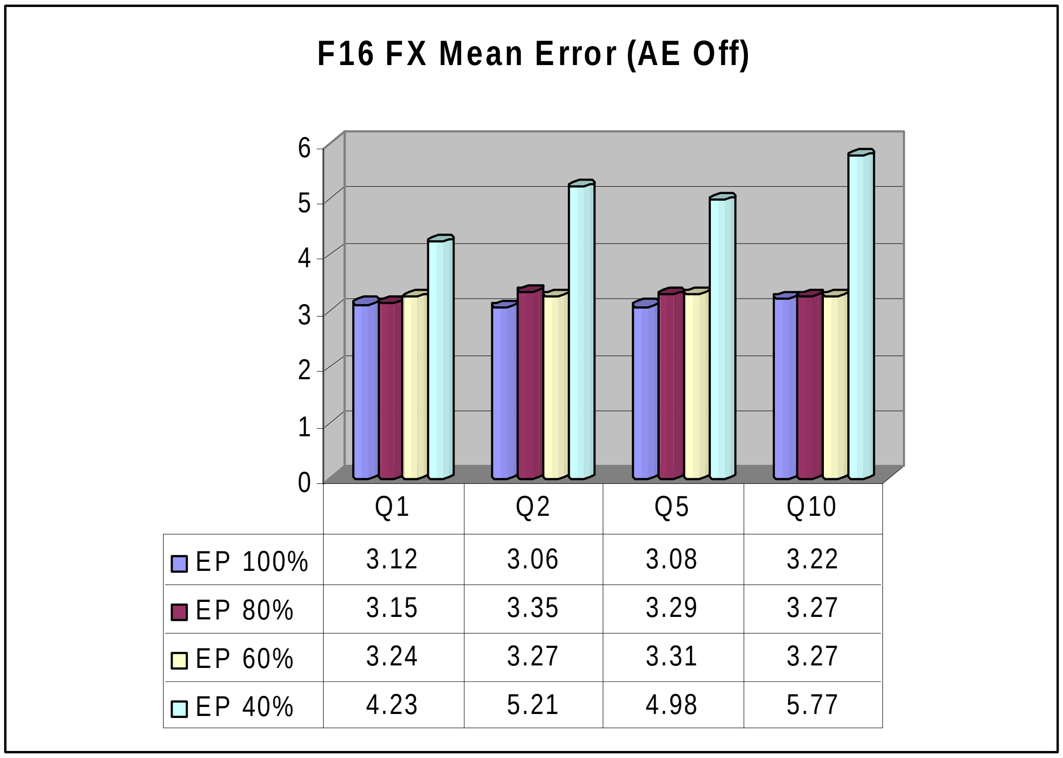


	Q1	Q2	Q5	Q10
🟦 EP 100%	3.27	3.32	3.4	3.47
🟥 EP 80%	3.27	3.22	3.32	3.49
🟨 EP 60%	3.24	3.4	3.38	3.5
⬜ EP 40%	4.14	4.1	4.1	4.52

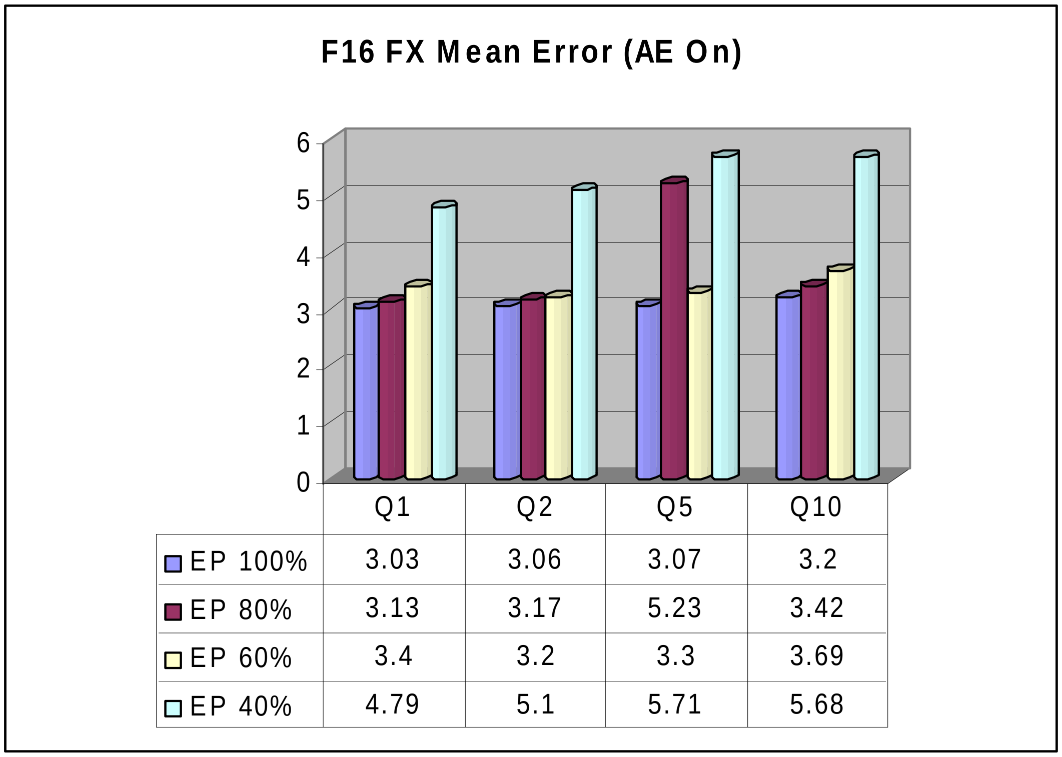


	Q1	Q2	Q5	Q10
🟦 EP 100%	3.22	3.34	3.3	3.47
🟥 EP 80%	3.11	3.39	3.32	3.61
🟨 EP 60%	3.25	3.4	3.61	3.54
⬜ EP 40%	4.33	4.26	4.48	4.59

Figure 4.2. Results for experiment A_Julie1_QT.

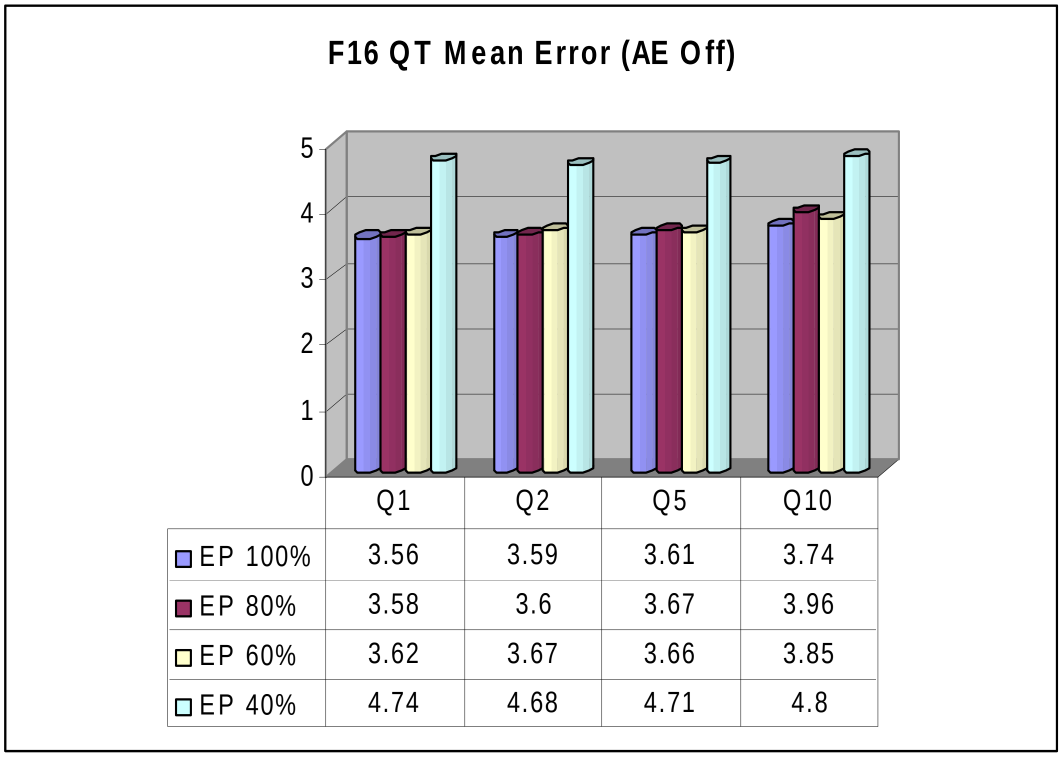


	Q1	Q2	Q5	Q10
🟦 EP 100%	3.12	3.06	3.08	3.22
🟥 EP 80%	3.15	3.35	3.29	3.27
🟨 EP 60%	3.24	3.27	3.31	3.27
⬜ EP 40%	4.23	5.21	4.98	5.77

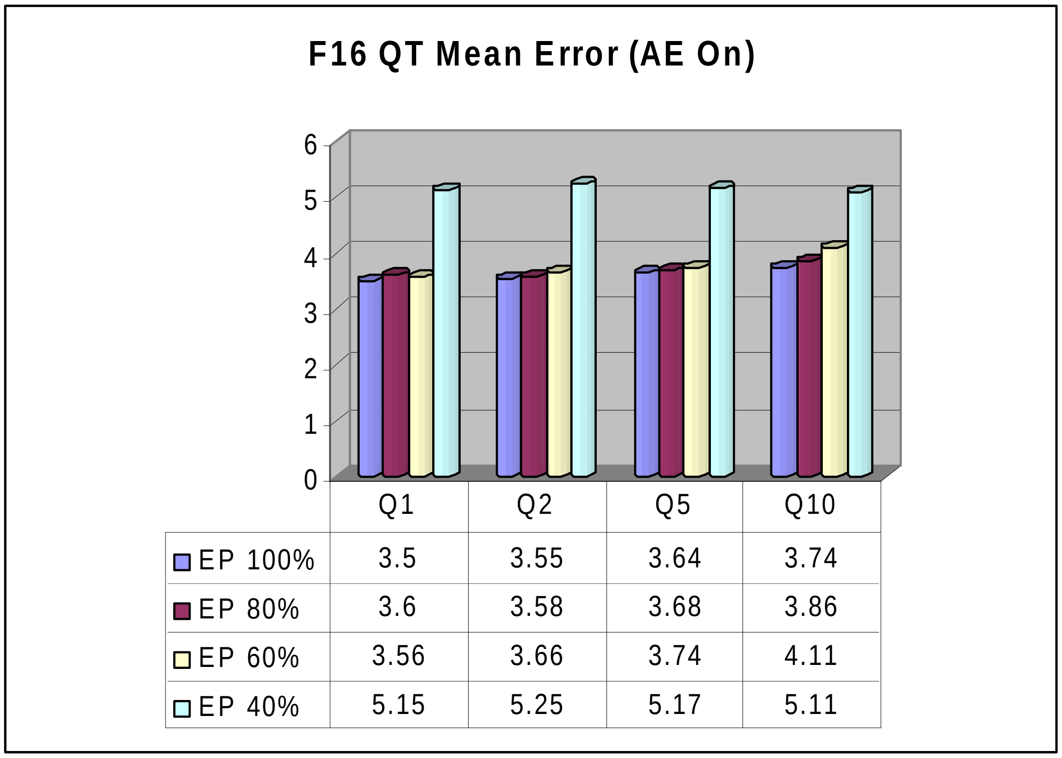


	Q1	Q2	Q5	Q10
🟦 EP 100%	3.03	3.06	3.07	3.2
🟥 EP 80%	3.13	3.17	5.23	3.42
🟨 EP 60%	3.4	3.2	3.3	3.69
⬜ EP 40%	4.79	5.1	5.71	5.68

Figure 4.3. Results for experiment A_F16_FX.



	Q1	Q2	Q5	Q10
🟦 EP 100%	3.56	3.59	3.61	3.74
🟥 EP 80%	3.58	3.6	3.67	3.96
🟨 EP 60%	3.62	3.67	3.66	3.85
⬜ EP 40%	4.74	4.68	4.71	4.8



	Q1	Q2	Q5	Q10
🟦 EP 100%	3.5	3.55	3.64	3.74
🟥 EP 80%	3.6	3.58	3.68	3.86
🟨 EP 60%	3.56	3.66	3.74	4.11
⬜ EP 40%	5.15	5.25	5.17	5.11

Figure 4.4. Results for experiment A_F16_QT.

Table 4.3. Parameters for Experiment Set B.

Test Name	Image	Subdivision Method	Number of Generations	Population Size
B_Julie1_FX	Julie1 (256-by-256)	Fixed blocks 4x4 block size (4096 blocks)	500	128
B_Julie1_QT	Julie1	Quad-tree 4x4 min block size (<4096 blocks)	500	128
B_F16_FX	F16 (512-by-512)	Fixed blocks 4x4 block size (16384 blocks)	200	128
B_F16_QT	F16	Quad-tree 4x4 min block size (<16384 blocks)	200	128

Table 4.4. GPConfiguration Parameters for Experiment Set B.

Parameter Name	Value
DCTQuantize	1
DCTEvolvePercent	1.0
AdjustErrorByAvgDifference	false

4.4 Experiment Set B Results

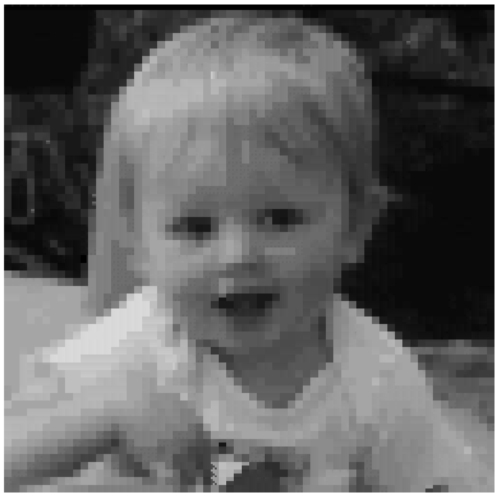
As mentioned in Section 3.1.3, Galloway tested two types of evolvable functions in his research. One type of function took more parameters and required more storage per subimage. The extra parameters were the values of the four corner pixels and the size of the image block. Galloway concludes that the additional information was useful to the GP, since the results of the GP when run with this function type were superior to the other function type which requires both x and y coordinates of a pixel [5]. In our system, the proposed evolvable function requires more data storage per subimage than

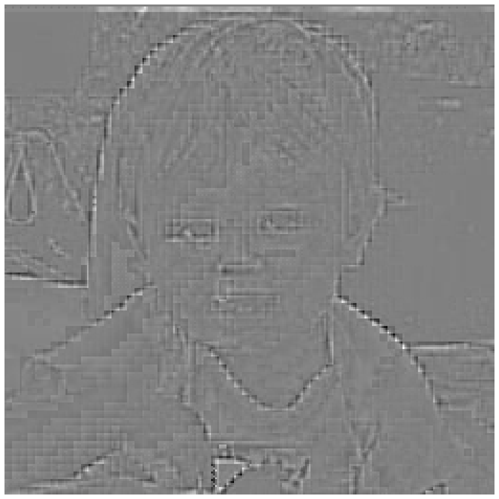
Galloway’s two-parameter function, but less than his seven-parameter function. For these reasons, the results of experiment set B are compared against the results of both function types proposed in Galloway’s system.

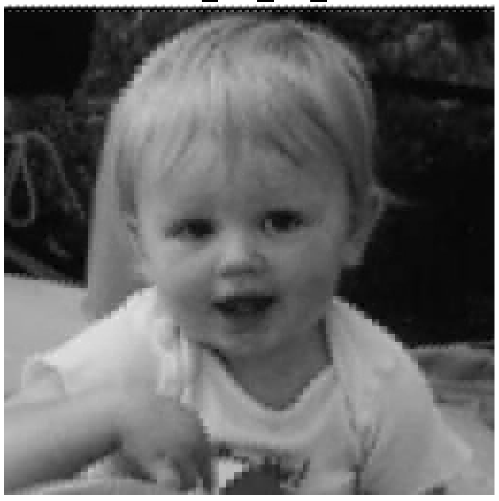
Table 4.5 shows comparisons in quality between the results of each of the four experiments in experiment set B and their equivalent experiments in Galloway’s research. The numerical data in each cell is the average mean error, standard deviation, and highest error between the pixel values of the original and approximated image.. The results from Galloway’s experiments are italicized. The abbreviations used in Galloway’s system are directly adopted here for the convenience of comparison. Specifically, FX represents the compression scheme using a fixed block subdivision, QT represents the compression scheme using a variable quad-tree subdivision, XY represents the compression scheme using the two-parameter (i.e., x and y coordinates of a pixel) evolvable function, and XYC4Z represents the compression scheme using the seven-parameter evolvable function. The evolved images and their calculated error images are shown in Figures 4.5 through 4.8. The first row shows the evolved and error images generated by Galloway’s 2-parameter evolvable function. The second row shows the evolved and error images generated by our approach with all of the new GPConfiguration parameters disabled. The third row shows the evolved and error images generated by Galloway’s seven-parameter evolvable function.

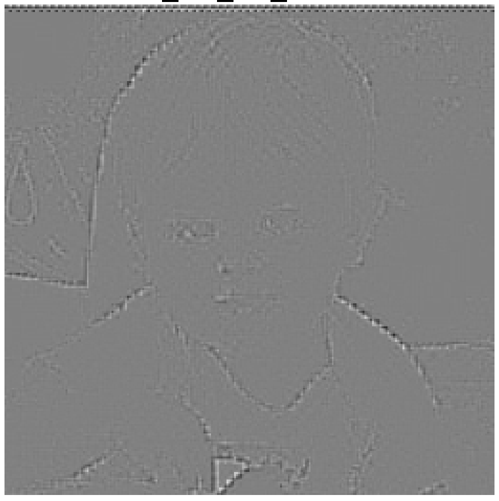
Table 4.5. Mean Error Comparisons Between GP Systems.

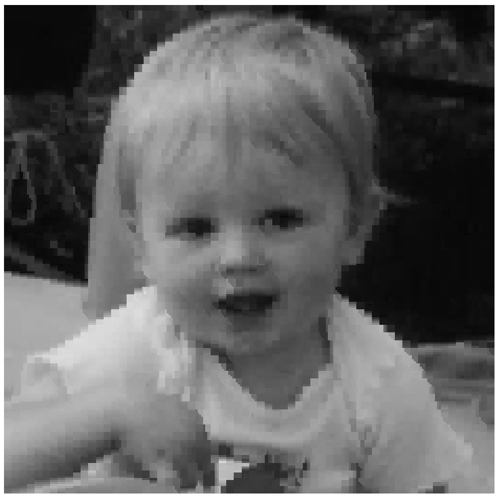
Test Name	**Mean Error**	**Std Dev**	**High**
B_Julie1_FX	3.18	7.64	86
Julie_FX_XY	*5.85*	*11.36*	*124*
Julie_FX_XYC4Z	*3.32*	*7.36*	*124*
B_Julie1_QT	3.81	7.54	78
Julie_QT_XY	*6.80*	*12.63*	*122*
Julie_QT_XYC4Z	*4.14*	*7.88*	*103*
B_f16_FX	3.83	8.44	88
F16_FX_XY	*6.11*	*14.01*	*108*
F16_FX_XYC4Z	*4.56*	*11.13*	*126*
B_f16_QT	4.10	7.78	85
F16_QT_XY	*6.36*	*13.41*	*119*
F16_QT_XYC4Z	*4.52*	*9.33*	*127*


Julie1_FX_XY_128


Julie1_FX_XY_128 Error


B_Julie1_FX


B_Julie1_FX Error


Julie1_FX_XYC4Z_128

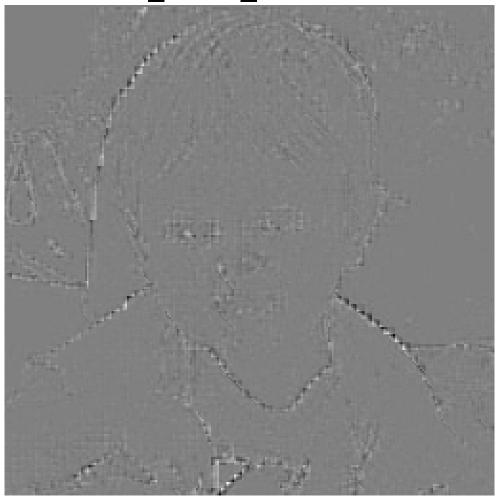
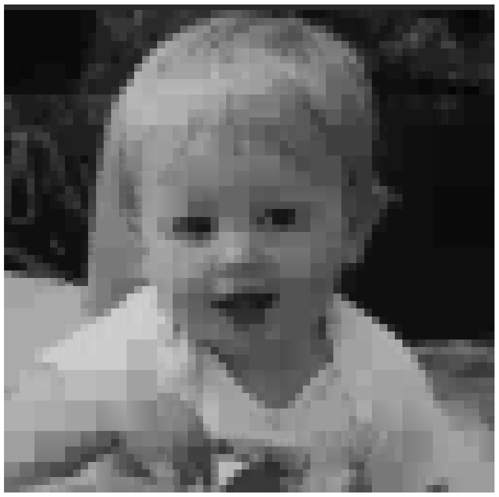
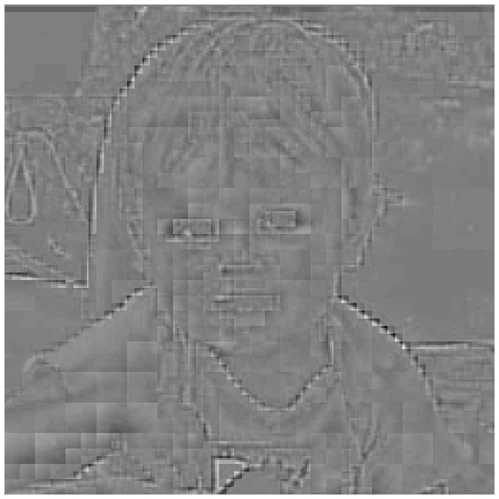

Julie1_FX_XYC4Z_128 Error

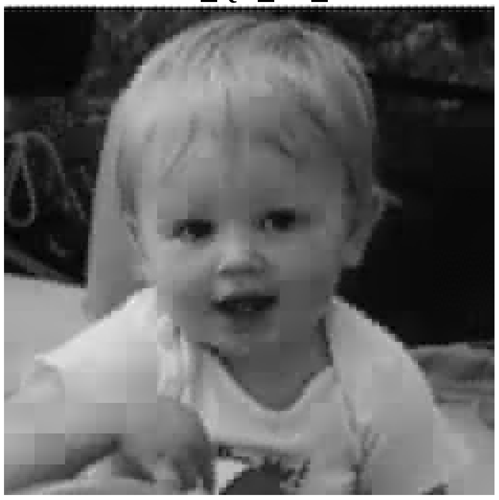
Figure 4.5. Results for experiment B_Julie1_FX



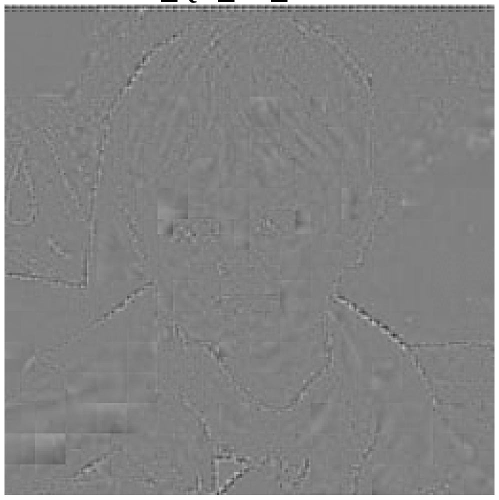
Julie1_QT_XY_128



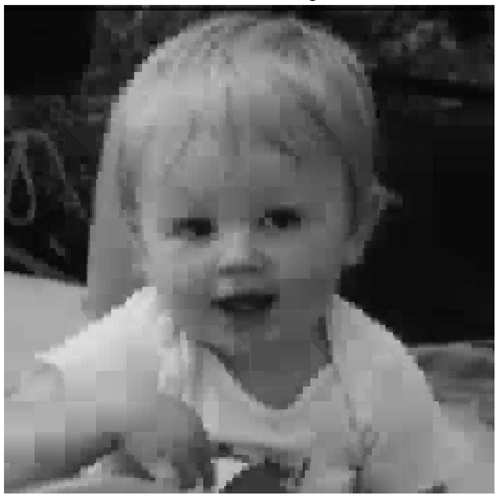
Julie1_QT_XY_128 Error



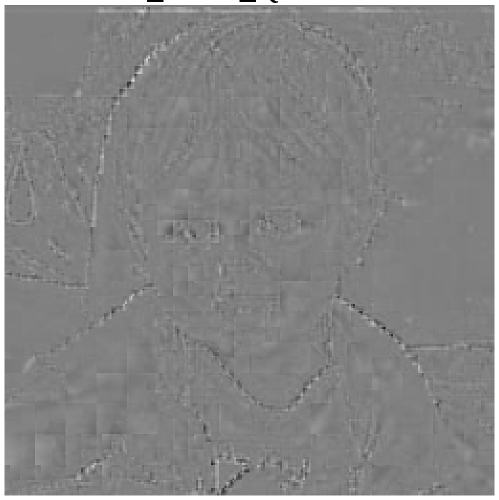
B_Julie1_QT



B_Julie1_QT Error

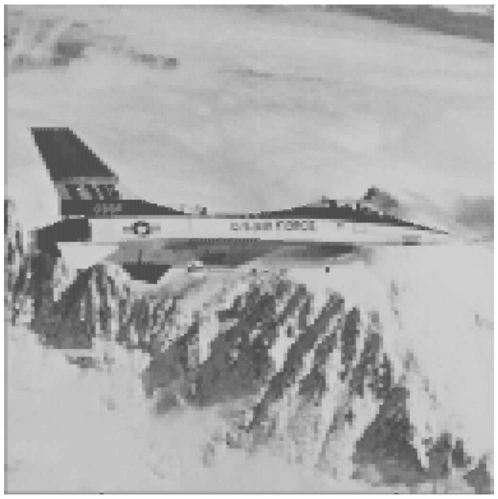


Julie1_QT_XYC4Z_128

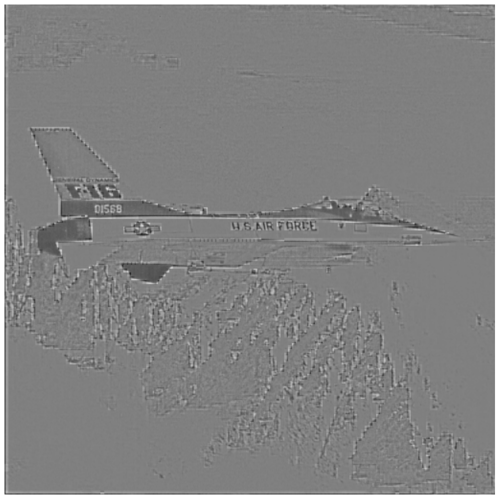


Julie1_QT_XYC4Z_128 Error

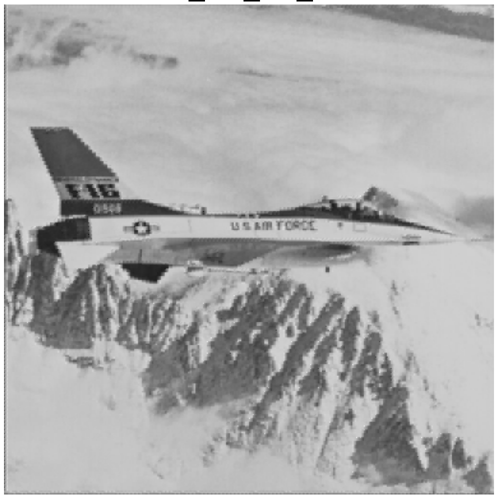
Figure 4.6. Results for experiment B_Julie1_QT.



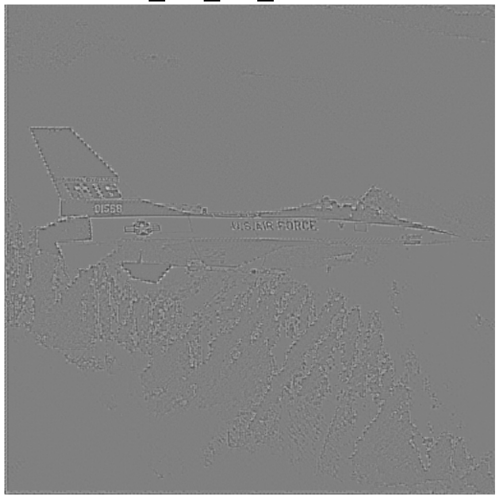
F16_FX_XY_128



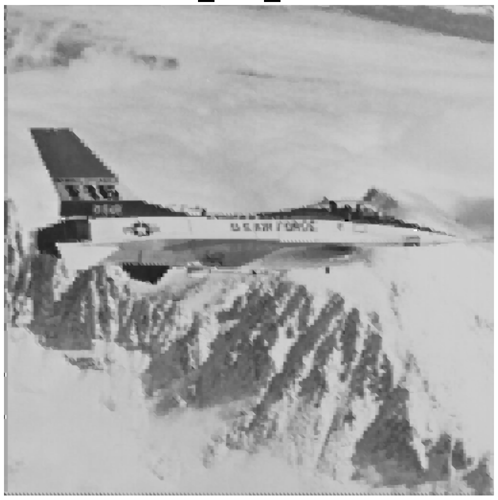
F16_FX_XY_128 Error



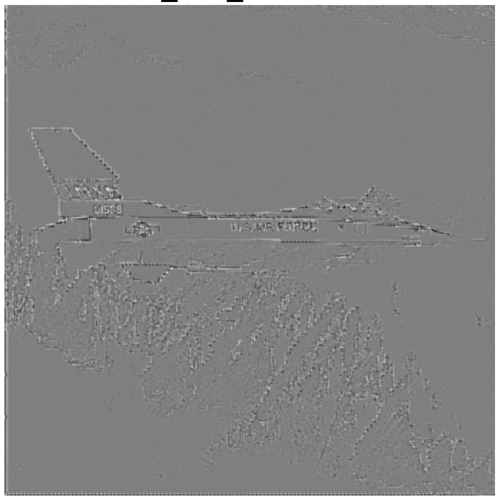
B_F16_FX



B_F16_FX Error

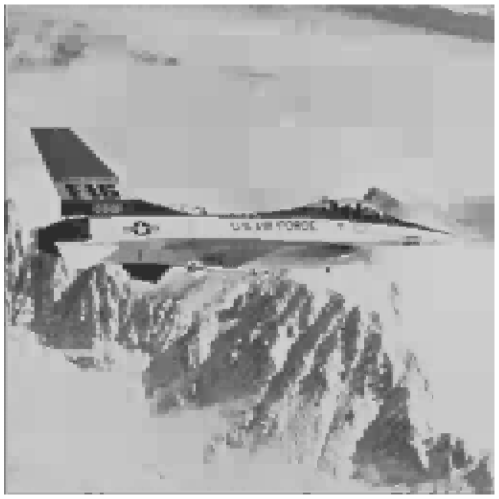


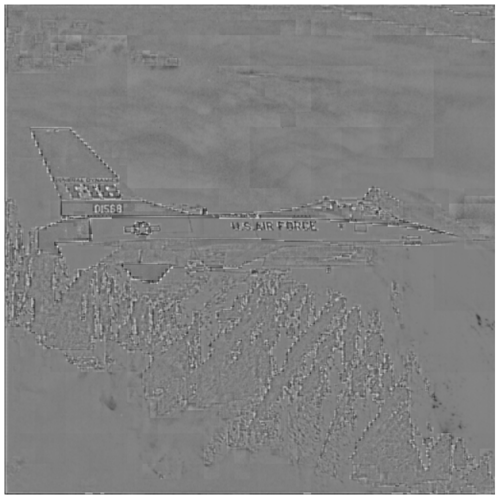
F16_FX_XYC4Z_128

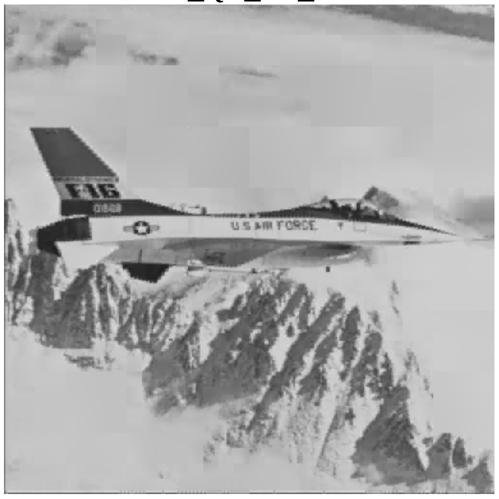


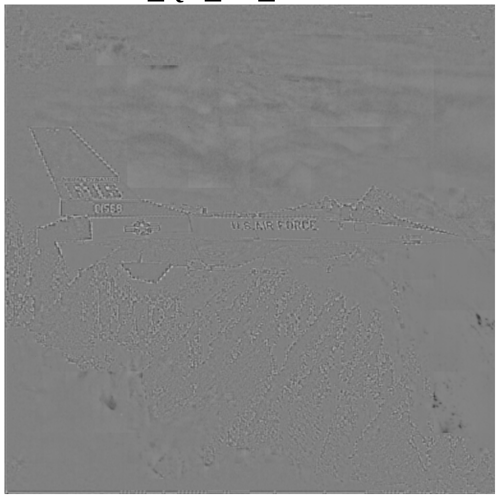
F16_FX_XYC4Z_128 Error

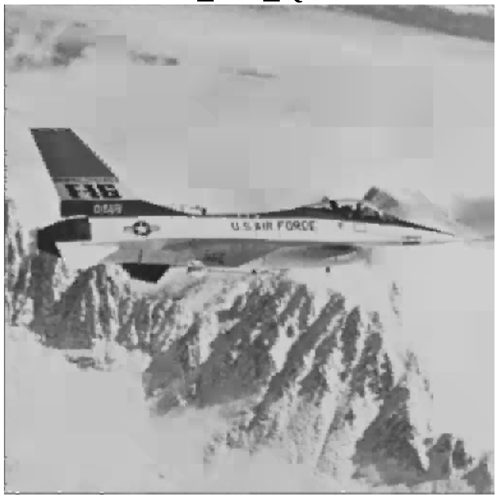
Figure 4.7. Results for experiment B_F16_FX.


F16_QT_XY_128


F16_QT_XY_128 Error


B_F16_QT


B_F16_QT Error


F16_QT_XYC4Z_128

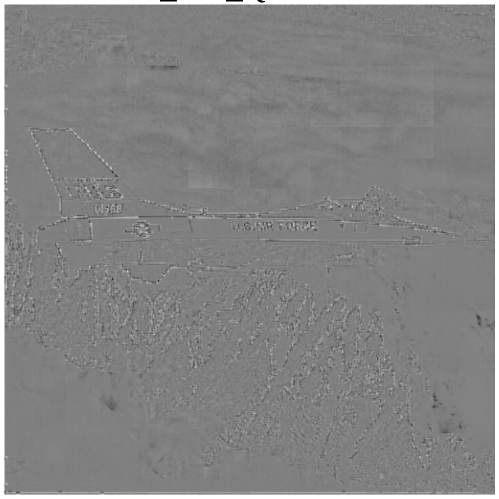

F16_QT_XYC4Z_128 Error

Figure 4.8. Results for experiment B_F16_QT.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The results shown in Chapter 4 are both exciting and surprising. The experiment B results show visible improvement over the spatial domain counterparts. The experiment A results prove that at least one of the proposed ideas has merit. They also show that the other proposed ideas fail to provide a benefit for this system. Below, we discuss the results of each test set individually.

5.1 Strengths and Weaknesses

The results of experiment set A show some definite trends. Comparing the graphs in Figures 4.1 through 4.4 across the AdjustErrorByAvgDifference parameter, we see an improvement when the parameter is set to true. This is understandable. This parameter helps the system recognize a function that provides a better fitness when adjusted errors are used and an unadjusted error would make the estimation unfit. Even when the same population function is the most fit with or without the error adjustment, this operation reduces the search space, thus allowing for a better estimation.

The DCTQuantize and DCTEvolvePercent parameters are less effective than the AdjustErrorByAvgDifference parameter. Figures 4.1 through 4.4 demonstrate that there is an increase in error as either the DCTQuantize value increases or the DCTEvolvePercent parameter decreases. The experiments show that our methods of quantization and evolving only a percentage of the DCT coefficients actually degrade the system’s performance.

Examining the DCT coefficients of one subimage during evolution reveals why these two parameters are less effective. Figure 5.1 presents a comparison between a subimage's DCT coefficients at the end of the GP system when run with DCTQuantize set to 1 and the same subimage's DCT coefficients at the end of the GP system when run with DCTQuantize set to 10. The first row of values for each of the data sets is the actual DCT coefficients or quantized actual DCT coefficients, the second row is the estimated DCT coefficients or estimated quantized DCT coefficients, and the third row is the error between the actual and estimated DCT coefficients or the error between the quantized actual and estimated DCT coefficients. A subimage's fitness value is calculated by the mean square error (MSE) between the estimated and actual DCT coefficients or the MSE between the quantized estimated and actual DCT coefficients. However, when the quantization is used in the compression, we need to take the estimated quantized DCT coefficients and multiply them by the quantization amount (10 in our example) in order to reverse the quantization. This unquantized data should be subtracted from the actual, original DCT coefficients to get the true error and the true subimage fitness. As a result,

```
No Quantization (fitness 1346.9375)
-453   0 453 346   0   0   0   0   0 188   0   0   0   0   0   0
-453 -21 381 414   2  35  34 -14  38 114 -24  35   0   0   0   0
   0  21  72 -68  -2 -35 -34  14 -38  74  24 -35   0   0   0   0

Quantized by 10 (fitness 21.9375)
 -45   0  45  35   0   0   0   0   0  19   0   0   0   0   0   0
 -45   3  42  40   2   1   2   2   2   2   1   1   0   0   0   0
   0  -3   3  -5  -2  -1  -2  -2  -2  17  -1  -1   0   0   0   0
```

Figure 5.1. DCT coefficient data for a 4-by-4 sub image with and without quantization.

the true fitness of the quantized subimage is revealed to be 2239.3125 instead of 21.9375, which is 892.375 greater than the fitness when it is not quantized at all. That is, even though the estimation errors are less when the subimage DCT coefficients are quantized, the true estimation error after the quantization is reversed is greater than the estimation error if quantization had never been done at all.

We can also understand from Figure 5.1 why it is problematic to evolve only a percentage of DCT coefficients. The example in this figure shows only a few runs of zeroes at the end of the actual DCT coefficients. Evolving less than 63% of the DCT coefficients in this example would produce significant error when the value 188 is simply set to the 0 default instead of evolving an estimate for it. Depending on the image, the DCTEvolvePercent parameter may actually work well if it is set to a value that does not neglect any significant DCT coefficients, as in this example. Every image would likely have a different optimal value for the DCTEvolvePercent parameter. This value could be calculated after the zig-zag scanning step of the GP system by recording the minimum length of the runs of zeros among all subimages. However, as a suggestion for future research, Section 5.3 presents a solution that is likely to be more beneficial than this one.

5.2 Improved Results

It is obvious from our experiments that quality of both images in experiment set B created by the frequency domain-based GP system is better than quality of both images created by the spatial domain version. Table 4.5 clearly shows this improvement, which is indicated by the small MSE. On average, the results of our system have an MSE that is 59% of the MSE of Galloway's system's results when using his two-parameter function, and 91% when using his seven-parameter function. The highest pixel error for our results is typically 70% of Galloway’s, with a smaller spread of error in all but one case. Figures 4.5 through 4.8 also illustrate these improvements in a more impressive and visible manner. The error images resulting from our proposed system are visibly the smoothest. In particular, in the case of the Julie1 image, a quality difference can even be visually detected in the estimated image from our system. As stated before, the proposed DCT domain GP system could potentially require more storage space than Galloway’s system using two-parameter population functions. However, it still requires less storage than using seven- parameter population functions. In both comparisons, our system achieves better results. Working in the frequency domain has a lot of potential, especially if the flexibility of the frequency domain representation of the data can be appropriately employed.

5.3 Future Research

As discussed in Section 3.2.1.4.1, the evolving functions need to produce output that has more entropy than the spatial domain GP system, since the values in the frequency domain are less correlated than their spatial domain representations. Introducing building block functions for the evolving GP that would work well on data with more entropy was an important part of this research. Similarly, removing functions that introduced homogeneity was another important part of this research. However, much more consideration could be made to developing building block functions that provide better suited output for the evolving functions. One possibility is to incorporate some form of a pseudo-random number generator. These functions are typically very simple in implementation and might prove to be a good fit for this system requirement due to their randomness and simplicity.

Section 3.2.1.4.3 explains how improvements to the population fitness calculation method were able to squeeze better performance out of the system, as well as better image quality. However, there is still potential for improvement in this part of the system. One possibility is to un-zig-zag scan the data of the inverse DCT before calculating the fitness. In this way, the error calculations would be for the actual pixel data, which ultimately is what the GP system is trying to estimate anyway. Additionally, this idea might benefit more from the AdjustErrorByAvgDifference parameter than our system did, since the data the system is trying to correctly estimate would be directly adjusted.

Run length encoding is a powerful compression tool utilized by the JPEG algorithm that could also be an effective tool in this research. The run length encoding method takes runs of identical values and represents them as a single value and a count. In some respects, the DCTEvolvePercent parameter is just a naïve form of run length encoding, since it is essentially assuming that some percentage of the values were non-zero and the rest were zero. A more intelligent approach of applying this idea might be to actually store runs of zeros and non-zeros. This would leave only a run of non-zero values for the GP system to evolve, since a large percentage of the DCT coefficients are zero. This implementation could also be combined with the AdjustErrorByAvgDifference parameter. Fewer zeroes (or in this case no zeros) between non-zero values it would increase the accuracy of the error adjustment.

Galloway was able to get better results from his seven-parameter population functions than his simpler two-parameter functions which accept only the x and y coordinates of pixels in the image. He concluded that these better results were a direct consequence of providing data that held useful information about the image. These results could be tested to see if they are extensible to our system. Potential data to be provided might be the minimum and maximum DCT coefficient values, the number of non-zeros, or even some form of run length encoding information.

Image subdivision is an area of the original GP system that was not directly modified for the frequency domain system. The original system uses a complexity calculation to determine if a subimage is simple enough to be evolved. Modifying this subdivision method to take into account the entropy features of the frequency domain representation of the data would likely improve performance. For example, zig-zag scan the DCT of a subimage and measure the length of the ending runs of zeros. This could make the subdivision better suited for other parts of the system wherein exploiting these characteristics of a frequency domain representation would prove beneficial.

REFERENCES

[1] Feiel, H., and Ramakrishnan, S. A genetic approach to color image compression. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, 1997.

[2] Frigo, M. and Johnson, S.G. The Design and Implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005), 216-231.

[3] Frigo, M., and Johnson, S.G. The FFTW web page. 2006. http://www.fftw.org. June 2008.

[4] Fukunaga A., and Stechert A. Evolving nonlinear predictive models for lossless image compression with genetic programming. In *Proceedings of 3rd Annual Genetic Programming Conference,* 1998.

[5] Galloway, J. *An Evolutionary Approach to Image Compression in C#.* Master's Thesis, Utah State University, 2003.

[6] Gonzalez, R., and Woods, R. *Digital Image Processing*, Prentice Hall, 2002.

[7] Jiang, J., and Butler, D. A genetic algorithm design for vector quantization. In *Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1995.

[8] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[9] Koza, J. *Genetic Programming II.* MIT Press, 1994.

[10] Nordin, P., and Banzhaf, W. Programmatic Compression of Images and Sound. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, 1996.

[11] Saha, S. Image compression – from DCT to wavelets: A review. *ACM Crossroads* 6, 3 (Spring 2000), 12-21

[12] Szalay, T. Using FFTW in C#. 2006. http://www.sdss.jhu.edu/~tamas/bytes/fftwcsharp.html. June 2008.

[13] Wu Y.-G. GA-based DCT quantisation table design procedure for medical images. *Vision, Image and Signal Processing, IEEE 151*, 5 (October 2004), 353-359.