Rate Control and Bit Allocation for JPEG Transcoding

by

Ricky D. Nguyen

S.B. E.E.C.S, M.I.T., 2006

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007

Author ....
Department of Electrical Engineering and Computer Science
May 25, 2007

Certified by ....
Shizhong Liu
Staff Engineer, QUALCOMM
VI-A Company Thesis Supervisor

Certified by ....
Vivek K. Goyal
Assistant Professor
M.I.T Thesis Supervisor

Accepted by ....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Rate Control and Bit Allocation for JPEG Transcoding

by

Ricky D. Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2007, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

An image transcoder that produces a baseline JPEG file from a baseline JPEG input is developed. The goal is to produce a high quality image while accurately meeting a filesize target and keeping computational complexity—especially the memory usage and number of passes at the input image—low. Building upon the work of He and Mitra, the JPEG transcoder exploits a linear relationship between the number of zero-valued quantized DCT coefficients and the bitrate. Using this relationship and a histogram of coefficients, it is possible to determine an effective way to scale the quantization tables of an image to approach a target filesize. As the image is being transcoded, an intra-image process makes minor corrections, saving more bits as needed throughout the transcoding of the image. This intra-image process decrements specific coefficients, minimizing the change in value (and hence image quality) while maximizing the savings in bitrate. The result is a fast JPEG transcoder that reliably achieves a target filesize and preserves as much image quality as possible. The proposed transcoder and several variations were tested on a set of twenty-nine images that gave a fair representation of typical JPEG photos. The evaluation metric consisted of three parts: first, the accuracy and precision of the output filesize with respect to the target filesize; second, the PSNR of the output image with respect to the original image; and third, the subjective visual image quality.

VI-A Company Thesis Supervisor: Shizhong Liu
Title: Staff Engineer, QUALCOMM

M.I.T Thesis Supervisor: Vivek K. Goyal
Title: Assistant Professor

Acknowledgements

Thanks to my family for always loving and supporting me.

Thanks to my friends for keeping me sane.

Thanks to Dr. Liu, Prof. Goyal, and MIT for all the opportunities.

Contents

List of Figures

Chapter 1

Introduction

From the Internet, to digital cameras, to picture messaging, the Joint Photographic Experts Group (JPEG) image compression standard has played a vital role in image transmission and storage for more than a decade. Recently, with the convergence of digital cameras and mobile communication devices, JPEG images can now be shared over mobile wireless networks. This sharing, however, may be costly in terms of network resources such as time and bandwidth. In an effort to reduce or limit this cost, carriers have imposed a filesize limit on images transmitted over the network. This filesize limit, however, should not constrain the high-quality images created or stored by a user, just the ones that require transmission. Thus, there is need for a transcoding solution, which takes a JPEG image and produces another JPEG image of smaller filesize.

Current solutions to this problem are neither efficient, nor accurate. For example, one straight-forward solution would be to use a conventional "black box" decoder and encoder to completely decode and then re-encode the image with a greater compression ratio. This method, however, requires the entire decompressed image to be stored in memory. As an example, a 5 megapixel image may be roughly 15 MB when decompresed. In addition, this solution often results in an unpredictable filesize and may require several trials before a desired target filesize is reached.

This work presents a novel JPEG transcoding solution to achieve a target filesize with dependable accuracy while also maintaining reasonable memory usage and computational complexity for implementation in mobile handheld devices. The transcoding solution presented here produces a file consistent with the JPEG baseline compression standard [4] in order to allow maximum compatibility with JPEG baseline decoders, which comprise the majority of decoders in real world usage. This work also draws upon ideas from research in the areas of image transcoding, rate control algorithms, and rate-distortion optimization.

The reader is assumed to have knowledge of the JPEG baseline compression standard. Should a detailed review be necessary, the reader may refer to Appendix A or the official document defining the JPEG standard [4]. This paper first outlines several compression options that have previously been explored. Next, the algorithms of the final transcoding solution are presented and the software architecture that implements this solution is described. Experiments were conducted, testing the performance of various algorithms on a fixed set of images. The images produced by the algorithms were evaluated by several metrics used to determine the most effective set of algorithms for the final transcoding solution.

1.1 Quantization

Quantization table selection is one of the most common ways of controlling bitrate in a compressed format. Increasing a quantization parameter decreases the number of symbols that need to be encoded. Since there are usually two 8x8 quantization tables in JPEG images, there are 128 parameters and nearly endless combinations available for both coarse and fine adjustment. But like downsampling, the tables apply to the entire image once they have been defined. There exists an extension to JPEG [5] that allows the tables to change for different parts of the image; although this extension is part of the JPEG standard, it does not fall within the baseline definitions and is not implemented in most decoders.

Much research has gone into the selection of quantization parameters. Wu and Gersho wrote one of the earliest papers using rate-distortion analysis to control JPEG bitrates [10]. The algorithm starts with a quantization table of large step sizes, producing an image with high distortion and low bitrate. The entry which produces the greatest decrease in distortion and increase in bitrate then has its step size decreased, and the process is repeated, one entry at a time, until the rate constraint is met. This algorithm is a rigorous and analytical approach to JPEG rate control but comes at a large computational cost because of the number of iterations over the entire image.

Similar research into quantization tables has attempted to model the image bitrate as a function of the quantization parameters. The Qualcomm quantization model [9] was developed after gathering empirical data from numerous test images. The model selects a scaling factor S as a function of the target bitrate R. This function was derived from images which used the "default" quantization tables $\boldsymbol{Q_0}$ and $\boldsymbol{Q_1}$ as defined in JPEG Annex K [4]. The image's new quantization tables are $S\boldsymbol{Q_0}$ and $S\boldsymbol{Q_1}$, regardless of the image's original tables. If the actual bitrate is greater than the target bitrate, S is adjusted and the entire image is reencoded.[1]

Another quantization model is described by Sherlock, Nagpal, and Monro [8]. Instead of modeling the scaling factor, Sherlock models each coefficient according to its position in the table. Let Q_{xy} be the value of the quantizer at column x and row y in an 8x8 block. Then the quantizer can be modeled as $Q_{xy} = A + D(x+y)^F$, where A and D are linear with compression ratio CR and F is quadratic with CR. This produces an 8x8 block that is symmetric over the main diagonal and nondecreasing with higher AC frequencies. The Sherlock Formula Model [8] fills the entries of the quantization tables according to the following definitions:

$$Q_{xy} = \lfloor A + D(x+y)^F \rfloor \tag{1.1}$$

$$A = \max(0, 2.04CR - 1.29) \tag{1.2}$$

$$D = \max(0, 0.315CR - 2.37) \tag{1.3}$$

$$F = \max(0, 4.33 - 0.218CR + 0.00384CR^2) \tag{1.4}$$

$$CR = \frac{3 * \text{height} * \text{width}}{\text{targetbytes}} \tag{1.5}$$

where Q_{xy} is the entry on row x and column y of an 8x8 block

The biggest drawback to these last two solutions is that several reencodings may be necessary in a slow and costly feedback loop. And since one model was derived from experimental data, it is still significantly probable that an image will not fit the model and the requantization may perform poorly. Also, neither model takes into account the statistics of the image in question, and therefore, is not guaranteed to reach the target bitrate after just one encoding.

[1]The details of the Qualcomm Model are not described for legal reasons.

1.2 Thresholding

While quantization table selection is a macroscale method of bitrate control that affects the entire image, thresholding is a microscale method that affects individual blocks of DCT coefficients. Ramchandran and Vetterli discuss a method of zeroing DCT coefficients according to rate-distortion analysis, called thresholding [7]. For each DCT block of 63 AC coefficients, the algorithm retains the set of coefficients that minimizes a rate-distortion cost function. This can be done independently for each block. The cost function depends on parameter λ, so for a given fixed λ, the algorithm returns an optimal solution after processing the entire image once. However, λ is usually not optimal, so additional iterations are required to find the optimal λ. Again, the drawback of this solution is that its complexity requires several passes through the entire image to reach an optimal solution, even though the act of thresholding itself is independent from block to block.

1.3 ρ-Domain Analysis

While studying rate control algorithms for video processing, He and Mitra identify a nearly linear relationship between the bitrate and the number of zero-valued quantized DCT coefficients [6] [2] [1]. They call the percentage of these zeros ρ and expressed the rate R as an approximately linear function of ρ. Using ρ-domain analysis, their algorithm can achieve desired bitrates by adjusting quantization parameters to produce the desired number of zeroes. These algorithms, however, were designed for video compression standards, such as MPEG-2 and H.263, which allow quantization parameters to change throughout a single image frame. This is called variable quantization, and although the JPEG standard defines an extension for variable quantization, this extension does not fall under the baseline definition and cannot be used in this work. Despite this fact, ρ-domain analysis and the linear relationship identified by He and Mitra is incredibly useful for bitrate control via quantization.

1.4 Other Compression Options

Downsampling appears to be the first step where compression may be improved. There are two approaches to downsampling: the image may have its dimensions reduced, downsampling all color components uniformly; or only the chroma components may be downsampled, maintaining the original image size. Uniform downsampling may produce significant savings but is an extremely coarse method of filesize control in terms of accuracy. Also, since most JPEG decoders only handle downsampling ratios of H1V1, H2V1, and H2V2, we are limited to these options to be assured that our transcoded images remain compliant. In addition, heavy chroma downsampling may produce more than 10 data units per MCU, exceeding the limit defined in the baseline standard.

Entropy encoding may or may not already be optimized in a given JPEG image. Often the default Huffman tables are used and further compression may be achieved if these tables are replaced with optimized ones. However, Huffman optimization reaches a theoretical limit, and if we desire smaller filesizes than this limit allows, another compression technique must be used in conjunction with Huffman optimization.

1.5 Summary of Motivations and Goals

Wireless carriers have imposed a filesize limit on images transmitted over their wireless network. Given a JPEG baseline image that exceeds this filesize limit, we wish to produce the best quality image that accurately meets the limit. Since the target application is cell phone usage, the solution must be efficient in terms of execution time and memory usage. The solution must also produce a JPEG baseline image that is compliant with the majority of decoders. This paper presents a transcoding solution that accurately and efficiently controls the bitrate of JPEG images.

Chapter 2

Implementation

The proposed transcoding framework implements two tiers of rate control on JPEG images. The first tier performs on a macroscale, where global operations affect the entire image equally and uniformly. The second tier performs on a microscale, where the effect of each operation is localized to specific regions of the image. This two-tiered strategy allows the transcoder to first coarsely approache the target filesize and then make fine adjustments as necessary.

2.1 Macroscale Rate Control

The primary means of achieving macroscale rate control is modification of the quantization tables. The DC components of the quantization tables should not change, since this may result in severe loss of features, intensities, and colors. Thus, we may only modify the AC quantization step sizes. We would like to scale the quantization tables uniformly, such that every entry of the table is scaled by a factor s. However, the JPEG bitrate R does not vary linearly with s. For this reason, we use the ρ-domain analysis approach suggested by He and Mitra [1] to first find a linear relationship between ρ, the percentage of zero-valued quantized DCT coefficients, and bitrate R. Then, all that remains is to find the relationship between s and ρ. We will show that this relationship is easily determined through the use of a histogram of quantized coefficient values.

A secondary means of macroscale rate control is optimization of the Huffman codes used for entropy encoding of the quantized coefficient values. Since the DC components will remain unchanged, the Huffman DC tables can easily be optimized if statistics on the whole image are gathered during a first pass decoding. However, since we will be modifying the AC components of the quantization tables, the AC statistics are no longer relevant for optimizing the AC Huffman tables. AC Huffman optimization may occur only if a third decoding pass collected relevant statistics, but for speed efficiency reasons, we will limit ourselves to two decoding passes. To make this point clear, let us follow an example. The default AC tables defined in the JPEG standard provide Huffman codes for all 162 distinct possible values to encode. An image may have defined its own custom optimized table that has codes for only 100 out of those 162 values. After modification of the AC quantization table entries, the image may only have 80 values that require codes, which are not necessarily a subset of the original 100 values. Because there is no way to predict which values will arise out of the new quantization tables, we must always use the default AC tables that provide full coverage of all 162 possible values.

2.1.1 ρ-Domain Analysis

Recall that the relationship between bitrate and ρ, the percentage of zero-valued quantized DCT coefficients, is nearly linear. When four images had their quantization tables scaled by various factors, their AC bits R and zero coefficient percentages ρ were plotted. The graph in Figure 2-1 verifies this conclusion.

Thus, the bitrate R can be modeled roughly as $\theta(1-\rho)$, where θ is a constant slope and is a property of the image. The model from He and Mitra requires several refinements before becoming useful in the transcoding framework. First, let z equal the number of zeroes; this just simplifies the arithmetic by using an integer rather than a ratio. Then z_{max} is the maximum value of z, or the total number of AC coefficients in the image. Second, the equation should only reflect the bits that are affected by the scaling of AC quantization entries and should not include end-of-block (EOB) symbols. An adjustment R_{min} is defined as the minimum number of AC bits. Stated differently, R_{min} is the value of R if every block in the image contained all zeroes, or equivalently, just an EOB symbol. The new relationship becomes

$$R - R_{min} = \theta(z_{max} - z). \tag{2.1}$$

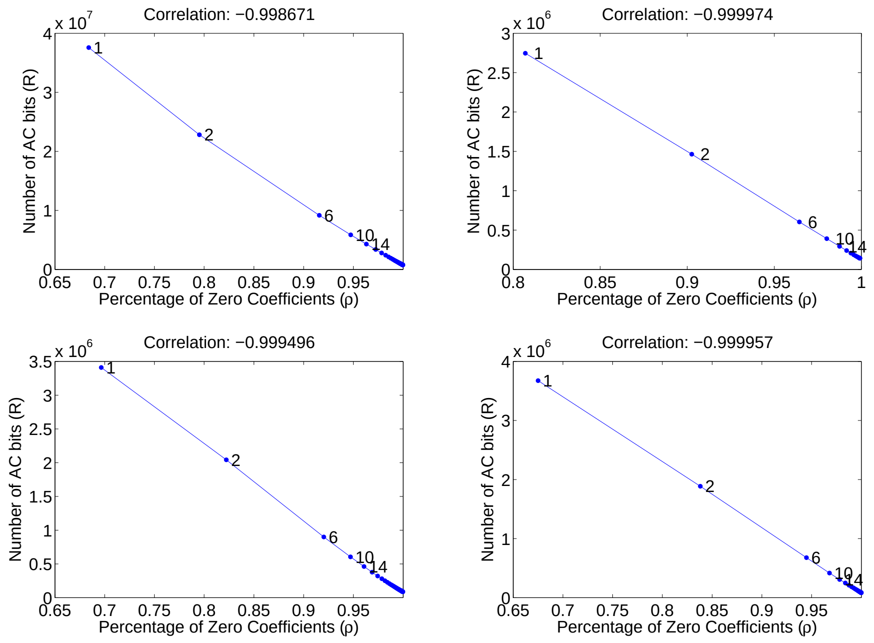


Figure 2-1: The relationship between the number of AC bits (R) and the number of zeroes as a percentage (ρ) of quantized AC coefficients is plotted for several images. The number next to each point indicates the scaling factor used on the quantization table of the image. Every image exhibits a correlation of nearly unity.

Given a target bitrate R_T and the original image statistics R_o and z_o, the target zeroes z_T can be calculated using Equation 2.1:

$$\theta_o = \frac{R_o - R_{min}}{z_{max} - z_o} \tag{2.2}$$

$$R_T - R_{min} = \theta_o(z_{max} - z_T) \tag{2.3}$$

$$z_T = z_{max} - \frac{R_T - R_{min}}{\theta_o} \tag{2.4}$$

Note that R_T is the target number of AC bits. A user would provide a target filesize F_T in bytes, which includes JPEG header information and DC bits. To get R_T, these bits are subtracted: $R_T = 8F_T - H - DC$, where H is the expected number of header bits in the new image, and DC is the number of DC bits in the original image.

With a histogram keeping count of the values of quantized DCT coefficients, z for an image can be easily recalculated for any modification of the quantization table. Define the histogram of quantized DCT coefficients as follows:

$$H(q,i,v) = \text{the number of occurrences of absolute quantized coefficient value } |v| \text{ in zig-zag scan entry } i \text{ (zero-indexed) using quantization table } q \text{ (zero-indexed)} \tag{2.5}$$

Note that $z_o = \sum_{q=0}^{1}\sum_{i=1}^{63} H(q,i,v=0)$. As another example, suppose that the 17th entry of the first quantization table is scaled by 5. Then z increases by exactly $\sum_{v=1}^{4} H(q=0, i=16, v)$, because all quantized coefficients with magnitude less than 5 in the 17th entry of the luma zig-zag scan will be requantized to zero.

This example is depicted graphically in Figure 2-2. The first plot shows the histogram of absolute values of quantized coefficients in the 17th entry of the luma zig-zag scan. The second plot shows the new histogram after rescaling the table entries by 5. The original counts for values 0 through 4, inclusive, are accumulated into the new count for value 0. Note that the original histogram only counts values up to 49. This is a practical space-saving consideration for the histogram data structure. We are only interested in counting zeroes produced by rescaling quantization parameters. Thus, a histogram with 50 bins can count zeroes for a scaling factor $s \leq 50$. We assume this is adequate for our transcoding purposes. A full histogram is shown in Figure 2-3.

Using this analysis, a scaling factor s and position index k can be chosen, such that

$$z_T \geq z_o + \sum_{q=0}^{1}\sum_{i=1}^{63}\sum_{v=1}^{s-1} H(q,i,v) + \sum_{n=k \ (n\neq 64)}^{127} H\left(q = \left\lfloor \frac{n}{64} \right\rfloor, i = n \bmod 64, v = s\right). \tag{2.6}$$

Equation 2.6 scales the chroma table more than the luma table. Conceptually, the quantization entries of both tables are concatenated into one long sequence: the luma entries are ordered in a zig-zag scan, then the chroma zig-zag scan is appended (still omitting the DC components). The complete scan progresses from low frequency luma to high frequency luma and continues from low frequency chroma to high frequency chroma. The entries of this complete scan before k are scaled by s, the remaining by $s+1$. Figure 2-4 provides a visual example of scaling factors applied to the quantization tables using this technique.

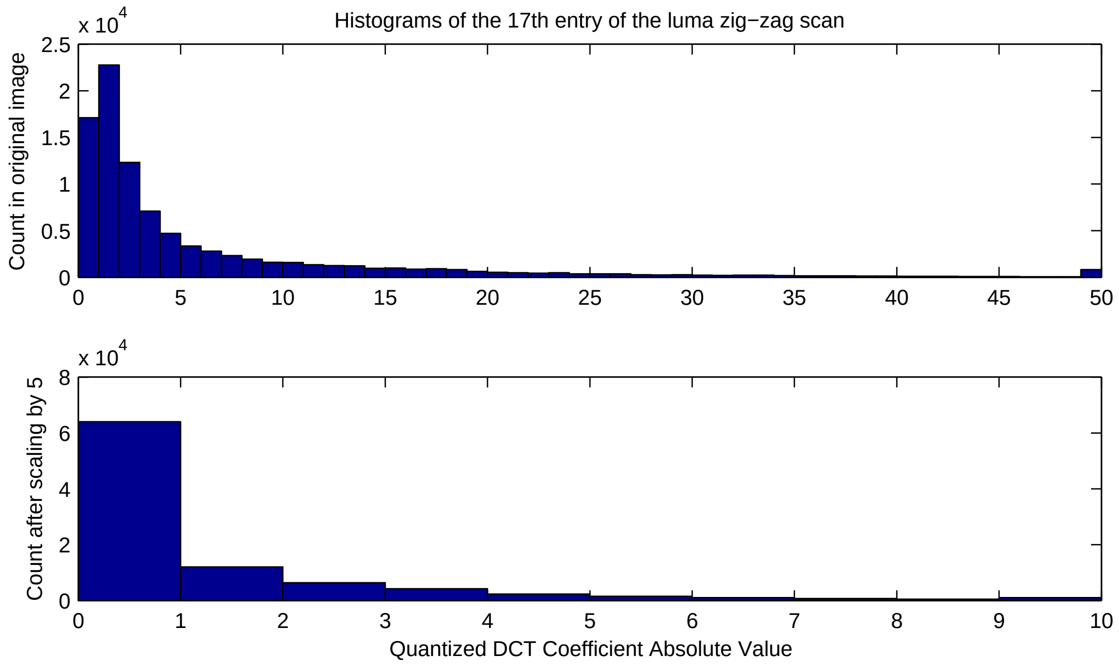
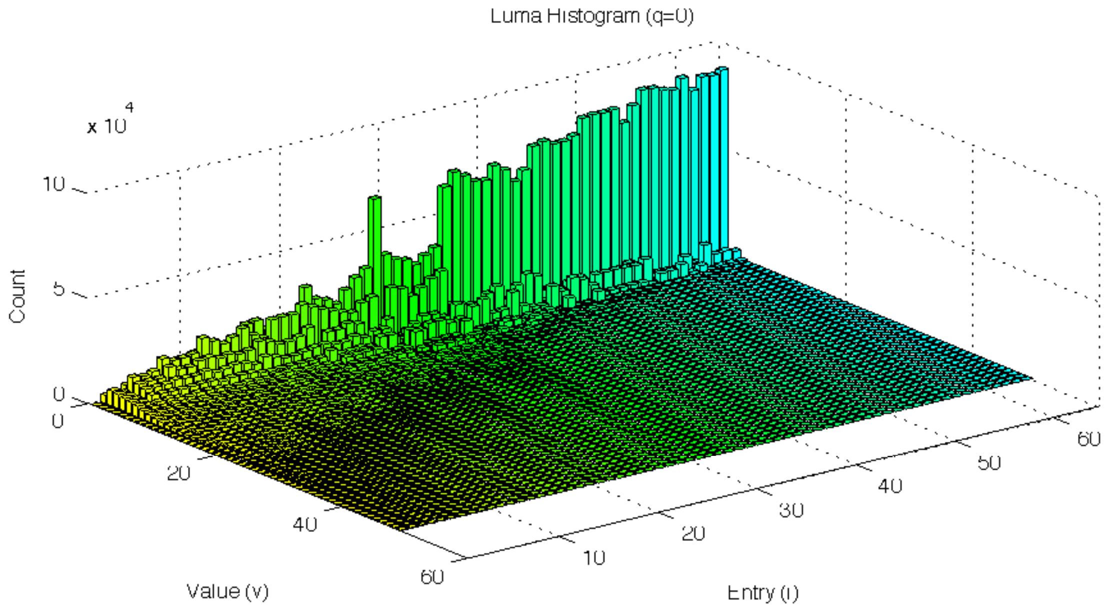


Figure 2-2: An example histogram of quantized DCT coefficients in the 17th entry of the luma

Thus, we have the first k entries of the concatenated scan scaled by s, and the remaining $128 - k$ entries scaled by $s + 1$ (omitting DC components). If we had scaled every entry of both tables by $s+1$, the zeroes produced by the new tables would have overshot z_T. Likewise, if we had scaled both tables by s, the new tables would undershoot z_T. This scheme of two scaling factors s and $s + 1$ acting on partitions determined by index k provides fine control over the number of zeroes produced by the new quantization tables.

2.2 Microscale Rate Control

While macroscale rate control modifies the quantization tables that uniformly affect the entire image, microscale rate control makes appropriate localized adjustments to meet the target bitrate. For fast transcoding, we would like microscale rate control to proceed in an online fashion, in midstream, during the process of decoding and encoding an image. Implementing this kind of microscale rate control requires four basic components. First, an indicator determines when bit-reducing action is necessary. Second, a quantifier calculates how much action is necessary. Third, a selector chooses which parts of the image should receive this action. Fourth, an actuator removes bits from the JPEG bitstream.



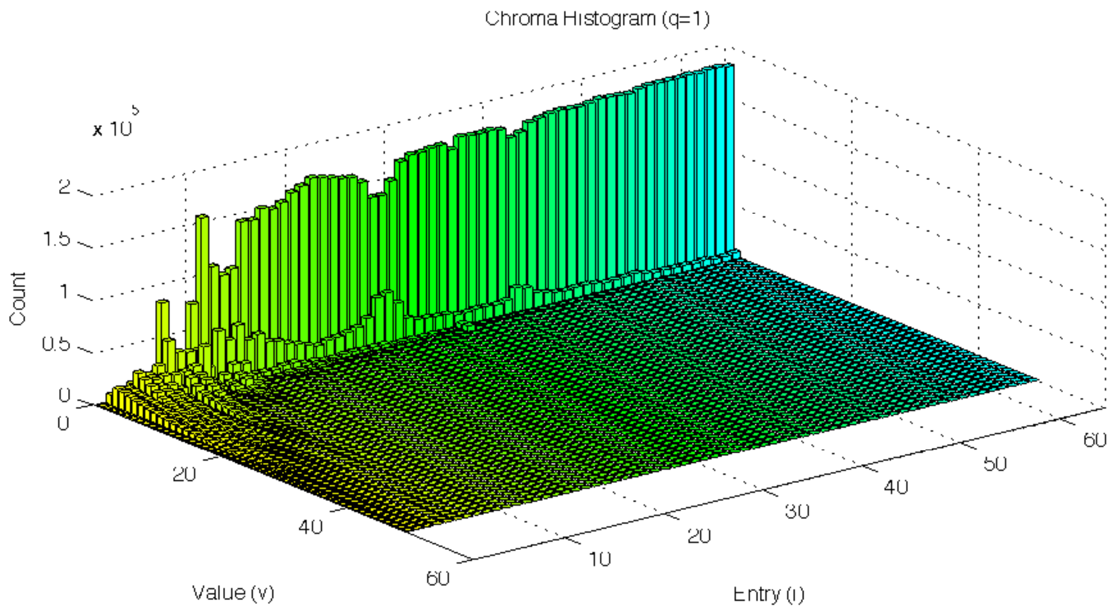


Figure 2-3: The full histogram from image 1.jpg. The luma histogram $H(q = 0, i, v)$ is shown above, and the chroma histogram $H(q = 1, i, v)$ is shown below it.

1	s	s	s	s+1	s+1	s+1	s+1	1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s	s	s	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s	s	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1
s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1	s+1

Figure 2-4: Example quantization scaling factors. Luma on left, chroma on right, $k = 10$. Note that the DC components are not scaled.

2.2.1 Indicator and Quantifier

Suppose we wish to transcode an image with original bitrate R_o down to a target bitrate of R_T. Ideally, we might expect that at the n^{th} 8x8 block of coefficients, the ratio of bits in the decoded original image $R_d(n)$ to the bits in the encoded new image $R_e(n)$ is equal to $R_o : R_T$. This leads to the following assumption:

$$\frac{R_d(n)}{R_e(n)} \approx \frac{R_o}{R_T} = \frac{R_d(N)}{R_T}, \tag{2.7}$$

where $R_d(n)$ and $R_e(n)$ denote the number of bits already decoded and encoded, respectively, at the n^{th} block; R_o is the original image bitrate, which is equal to $R_d(N)$ if the image has a total of N blocks; and R_T is the target bitrate of the newly encoded image.

The proportions in Equation 2.7 can be made more accurate if we remove the constant offset of the minimum number of bits, just as we did in the case of ρ-domain analysis in Equation 2.4.

$$\frac{R_d(n) - R_{min}(n)}{R_e(n) - R_{min}(n)} = \frac{R_d(N) - R_{min}(N)}{R_T - R_{min}(N)} \tag{2.8}$$

Since we want the target bitrate at the n^{th} block to reflect this proportion, substitute $R_t(n)$ for $R_e(n)$ in Equation 2.8 and solve for the error $\epsilon(n) = R_e(n) - R_t(n)$.

$$R_t(n) = \frac{R_T - R_{min}(N)}{R_d(N) - R_{min}(N)} \left(R_d(n) - R_{min}(n)\right) + R_{min}(n) \tag{2.9}$$

$$\epsilon(n) = R_e(n) - R_t(n) \tag{2.10}$$

Thus, we can use this error measure for both our indicator and quantifier. When $\epsilon(n) > 0$, the indicator signals that action must be taken. The magnitude of $\epsilon(n)$ quantifies how many bits should be removed from the bitstream.

2.2.2 Bit Allocation with the Selector and Actuator

The bit allocation problem of JPEG transcoding is determining which parts of the bitstream should receive bitrate reduction and how should bitrate reduction occur. These are the aspects of bit allocation that the selector and actuator address. Keep in mind that we would like to achieve the greatest reduction in bitrate for the smallest distortion cost. The solution implemented here exploits the JPEG specifications of entropy encoding to obtain large savings in bits for small distortions.

The JPEG bitstream is comprised mostly of Huffman codes and residue strings for the AC coefficients. The length of the residue string is dictated by the value returned by the Huffman code. Thus, if we can reduce the length of the Huffman code, we can directly save several bits. Since smaller coefficient values are usually more frequent than larger values, the small values are usually assigned shorter Huffman codes. Consequently, we can achieve savings in both the Huffman code and the residue if the code is shortened. Codes for AC coefficients are assigned based on the run-length of zeroes and the binary size of the coefficient. We cannot change the run-length of zeroes without radically altering the block, but we can subtly change the binary size of a coefficient. If the value to be encoded is a power of 2, it can be decremented by 1, and its binary size will be reduced by 1. The Huffman code shortens, the residue shortens, and several bits are saved as a consequence of merely decrementing a single coefficient value. Bit savings are maximized while coefficient distortion is minimized.

The selector in this case selects all quantized DCT coefficients that are a power of 2. However, since 1 is also a power of 2, all 1's will decrement to 0, eliminating that particular component of the DCT entirely. To avoid this, the selector omits values of 1 when selecting powers of 2. The actuator in this case is the action of decrementing the value.

Another source of bit savings may be achieved if the number of stuff bytes is reduced. Recall that a stuff byte is a 0x00 inserted after the occurence of 0xFF in the entropy coding bitstream. In other words, if a Huffman codes and residues form a byte-aligned sequence of eight 1's, then eight 0 bits are inserted. Bits may be saved if we could avoid the use of stuff bytes by avoiding the creation of a byte-aligned sequence of eight 1's. However, we cannot arbitrarily flip any of the eight 1's. If we alter a Huffman code, the decoder will reach an error, thus we can only change residues. When altering a residue, we do not want to change its value by a large amount, so we should flip the least significant bit that still prevents the writing of 0xFF to the bitstream.

Suppose we have a bit buffer in which the leftmost 8 bits will determine the next byte written to the JPEG bitstream. Huffman codes and residues are alternately appended on the right, with their most significant bit being the leftmost bit. The leftmost 8 bits can contain any of the following:

1. only bits from Huffman code (H)
2. only bits from residue (R)
3. end of Huffman code, start of residue (HR)
4. end of residue, start of Huffman code (RH)
5. end of Huffman code, entire residue, start of Huffman code (HRH)
6. end of residue, entire Huffman code, start of residue (RHR)
7. Any of items 3–6 with HR or RH pairs inserted in the middle of the sequence.

We will now go through this list of cases and determine those which may produce 0xFF. If a case produces 0xFF, we must determine which of the 8 bits to flip, while perserving Huffman codes and minimizing the change in residue. Immediately, we see that case 1 is unmodifiable.

No Huffman code may be all 1 bits because the JPEG standard has declared that "codes shall be generated such that the all-1-bits code word of any length is reserved as a prefix for longer code words" [4]. Using this information, cases 6 and 7 will include an entire Huffman code and are guaranteed to never produce 0xFF.

In all of cases 2–5, we can just flip the least significant bit of the residue that still lies within the leftmost 8 bits of the bit buffer. The selector determines cases 2–5, and the actuator flips the residue bit; together, this is the bit allocation scheme for stuff-byte avoidance.

The final transcoding solution implements both the decrementing of powers of 2 and stuff-byte avoidance. These two selector/actuator pairs are both activated when the indicator/quantifier returns a value greater than zero. Otherwise, encoding proceeds normally.

Thus, the decrementing and stuff-byte avoidance are either both active or both inactive. Since stuff bytes are typically rare occurrences, most of the bit reduction will arise from decrementing.

2.3 Software Architecture

The complexity and efficiency constraints on the transcoding process necessitated the development of an entirely new decoding and encoding framework. The popular open-source Independent JPEG Group libraries [3] are extremely versatile, handling many JPEG formats (in addition to baseline), performing complex postprocessing, and offering dynamic memory management. Although the memory management options may seem beneficial for applications in embedded systems, the non-baseline and postprocessing routines are superfluous features that inflate the code base and keep state information that this project will never utilize. A custom-built framework allows fine control over every aspect of the transcoding process, operating on only the necessary phases of decoding and encoding. This chapter outlines the design choices made for the transcoding framework, including design patterns, modules, and data flows.

Efficiency, in terms of speed and memory usage, is a primary concern in the design of this framework. Processing time is reduced by eliminating all DCT calculations and operating entirely within the frequency domain. While decoding, the inverse DCT is unnecessary because the forward DCT will be immediately applied during encoding. In the framework, decoding proceeds until just before the inverse DCT, the frequency domain coefficients are manipulated, and encoding writes the coefficients to the JPEG bitstream. Rather than store all the coefficients of the entire image in memory before encoding, memory can be saved if the transcoding operates on smaller units, such as an MCU or an 8x8 block of DCT coefficients. Figure 2-5 shows the block-by-block processing in this transcoding framework.

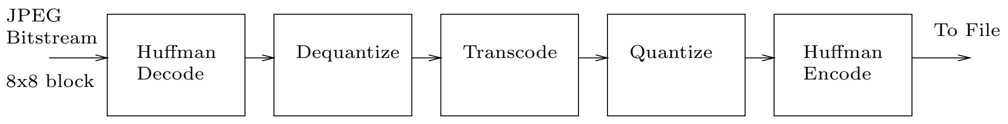


Figure 2-5: The block-by-block transcoding in this framework. Note that no DCT or inverse DCT is performed, and all transcoding operations are in the frequency domain of DCT coefficients.

The transcoding framework uses the observer design pattern. In the observer design pattern, when an object changes its state, it raises an event and notifies any observers who are listening for that particular event. The object *publishes* the event to its *subscribers*. Each subscriber then performs actions relevant to information contained in the event. With the observer pattern, the design becomes very modular. Several subscribers may respond to the same event, each with completely different functionality, and related functions can be packaged into one module. Also, the observer pattern allows the transcoding to proceed in an online fashion without keeping too much state in memory. As soon as a data unit is decoded, a subscribed encoder can immediately encode it.

There are four basic modules in the transcoding framework: the decoder, the encoder, the statistics collector, and the transcoder. The decoder reads the original JPEG bitstream and produces data for manipulation. The decoder is the main publisher of information to subscribers. The other three modules may subscribe to decoder events. As mentioned earlier, if an encoder subscribes to the decoder, it immediately encodes the recently decoded data. If a statistics collector subscribes to the decoder, it gathers information such as histograms of DCT coefficients and counts of Huffman symbols. If a transcoder subscribes to the decoder, it manipulates the decoded data and re-publishes the new data. Here, the observer pattern becomes very handy. The encoder and collectors can subscribe to the transcoder to encode and count the new data just as easily as if they were subscribed to the decoder directly. Different collectors may also be customized to perform specifically for the decoder, transcoder, or encoder.

Data flow throughout the framework is outlined in Figure 2-7. A first pass decoding is allowed to collect global image statistics. In this first pass, statistics collectors subscribe to the decoder. Transcoding occurs on the second pass. The transcoder subscribes to the decoder, the encoder subscribes to the transcoder, and another collector subscribes to the encoder to track encoder statistics. Thus, the decoder is run twice, and only the subscriptions are changed. Since statistics are only useful to the transcoder when manipulating coefficients, collectors are encapsulated in the transcoder. Pseudocode for the main program appears in Figure 2-8.

The modules in this framework are data structures that occupy space in the memory heap. The largest data structures in this program are shown in Figure 2-9 and include the histogram of quantized coefficients, the read and write buffers of the decoder and encoder,

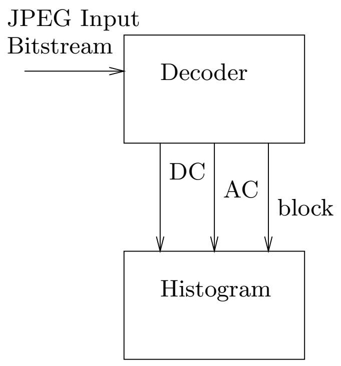


Figure 2-6: Data flow in the JPEG transcoding framework during the decoding pass (first pass). The JPEG bitstream is decoded block by block. The decoder publishes block data and DC and AC symbols to the histogram collector.

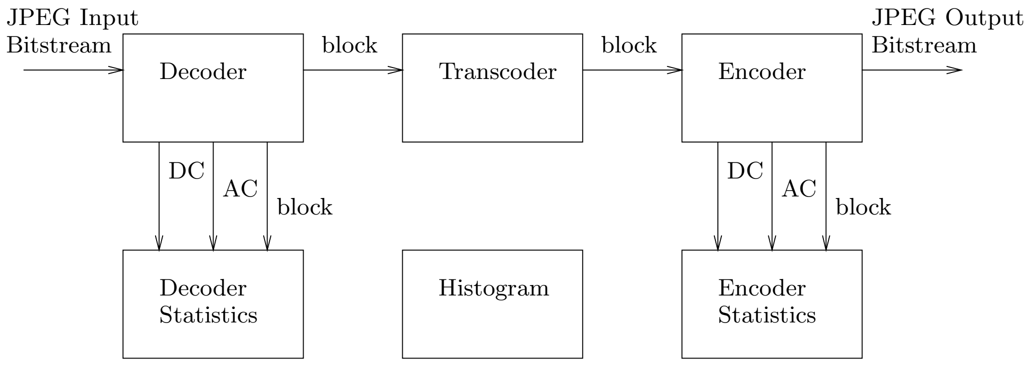


Figure 2-7: Data flow in the JPEG transcoding framework during the transcoding pass (second pass). The JPEG bitstream is decoded, transcoded, and encoded block by block. In addition, both the decoder and encoder publish block data and DC and AC symbols to their respective statistics collectors. The transcoder has access to the statistics in both collectors and the histogram when manipulating the JPEG bitstream.

```
create decoder from original file
create transcoder
link transcoder to decoder
decode entire image (collect statistics)

reset decoder
reset transcoder
create encoder
link transcoder to decoder and encoder
decode entire image (transcode and write new image)

destroy decoder
destroy encoder
destroy transcoder
```

Figure 2-8: Pseudocode for the main program of the proposed JPEG transcoder. Because of the observer design pattern, the main program can simply link publishers to subscribers, then decode the input JPEG image in two passes. The output image is produced automatically by the subscribing transcoder and encoder.

Data Structure		**Bytes**
Transcoder		256
Decoder	Read buffer	4096
	Huffman decoders	968
	Quantization tables	256
Encoder	Write buffer	4096
	Huffman encoders	2304
	Quantization tables	256
Histogram		25K
Heap (optional)		80K
Total with heap		117K
Total without heap		37K

Figure 2-9: Memory usage of data structures in the transcoding framework.

the quantization tables, the Huffman decoders and encoders, and an optional heap data structure used to keep track of blocks with high AC energy (see Section 3.2). The histogram keeps 32-bit unsigned integer counts for 2 quantization tables, 64 zig-zag scan entries, and 50 values, requiring 25KB of memory. The read and write buffers each store 4KB of data. The quantization tables are different for the decoder and encoder, so there are 2 luma tables and 2 chroma tables of 8-bit parameters, requiring 512 bytes of memory. The Huffman decoder, as implemented by the JPEG standard [4], uses 4 arrays totalling 242 bytes of storage. The Huffman encoder, implemented as 2 arrays of codes and code lengths, uses 576 bytes of storage. Four Huffman decoders and four huffman encoders require 3.2KB of memory. The optional heap can store approximately 10000 nodes, each containing a block number (32-bit unsigned integer) and a corresponding AC energy value (32-bit unsigned integer), totalling about 80KB. The transcoder module itself takes up very little memory, holding pointers to all of these other data structures.

Another design pattern is implemented in the transcoder called the strategy pattern. This pattern simply means that several algorithms (or strategies) may implement one function. The strategy pattern becomes useful when evaluating and comparing different algorithms for their effectiveness in achieving accurate bitrate control. The framework implements every algorithm in question, and the main program simply needs switches to plug in the correct algorithm. The implementation and results of individual algorithms are each discussed in detail in the next chapter.

Chapter 3

Experimental Results

A data set of 29 JPEG photos was collected, containing diverse scenery that included indoor and outdoor environments, people, faces, objects, large smooth regions, large detailed regions, text, numerous vivid colors, and nearly monochrome images. The intent was to provide the gamut of photos taken by ordinary users, and at the same time test the transcoder's response to variations in frequency and color.

As a control, the quantization tables of each test image were uniformly scaled by eight constants to produce eight files of different size for each test image. These eight constants were chosen as powers of 2, linear in the exponent: $2^{0.5}, 2^{1}, 2^{1.5}, ..., 2^{4}$. The image $f(i,j)$ is the j^{th} scaling of image i. We use the filesize of $f(i,j)$ as the target filesize for our experiments. The resulting images are then compared against image $f(i,j)$ for filesize accuracy, subjective visual quality of the image, and objective peak signal-to-noise ratio (PSNR) against the original image.

The experiments approached the two tiers of rate control separately. First, several quantization models were tested for macroscale rate control. After completing the macroscale experiments, one quantization model was chosen and subsequently used in all microscale experiments. Several combinations of indicator / quantifier / selector / actuator were tested for microscale rate control. Taking into account filesize accuracy, visible compression artifacts, and PSNR, the most effective models and algorithms for JPEG bitrate control were determined. Throughout all experiments, we limit our transcoding algorithms to two passes over the image: the first for gathering statistics and the second for transcoding operations on the output image.

3.1 Macroscale Results

Macroscale rate control experiments were used to determine the best method of modifying the quantization tables to accurately achieve a target filesize. Chapter 1 introduced two quantization models, the Qualcomm Model and the Sherlock Formula Model. These two models were implemented as described, however, this implementation of the Qualcomm Model only encoded the image once because of our two-pass restriction on the transcoder. The ρ-domain analysis described in Section 2.1 was also used in this set of experiments. Figure 3-1 shows the results of macroscale rate control experiments.

From Figure 3-1, the Sherlock Formula tends to overcompress the image beyond the target filesize, while the Qualcomm Model often does not compress enough. The ρ-domain analysis, however, provides encouraging results, as the resulting output filesizes are within $\pm 10\%$ error. Because the Qualcomm Model and Sherlock Formula produce highly inconsistent results, the subsequent microscale rate control experiments use ρ-domain analysis to scale the quantization tables.

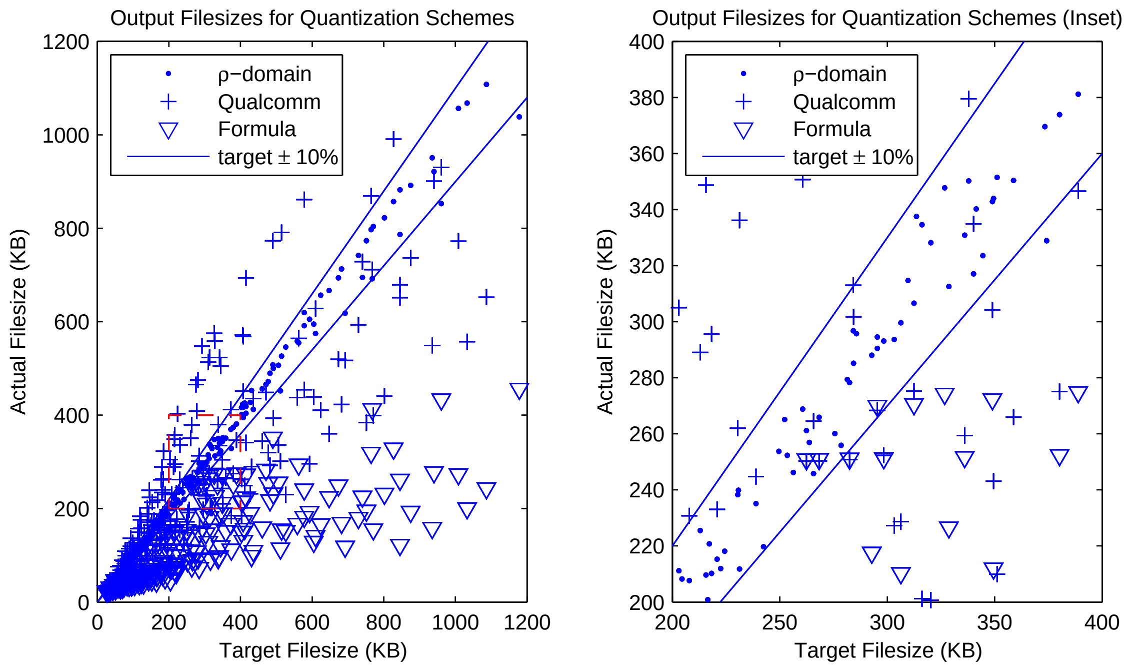


Figure 3-1: The test images had their quantization tables uniformly scaled by several constants. These filesizes were used as targets for the three quantization schemes: Qualcomm Model, Sherlock Formula Model, and ρ-domain analysis. Scatter plots of output filesize vs. target filesize are shown. The output filesizes for the ρ-domain analysis usually fall within $\pm 10\%$ of the target, while the Qualcomm Model undercompresses and the Sherlock Formula overcompresses.

Macroscale rate control is achieved through modification of the quantization tables. The methods that scale the original tables obtain better filesize accuracy. The relationship between bitrate and the number of zero-valued quantized coefficients was verified to be linear. Through the use of a histogram, this linear correlation is exploited to achieve filesize control accurate to within $\pm 10\%$. Because the human visual system is more sensitive to brightness and low frequencies, scaling the quantization entries favors these components and discards more information in high frequencies and color components.

3.2 Microscale Rate Control

Microscale rate control experiments were used to determine the best method of modifying the DCT coefficients intra-image to accurately achieve a target filesize. Because our transcoding framework was built to encode one 8x8 block of coefficients immediately after decoding it, we can modify the coefficients between the decode and encode steps. Given the statistics of the first pass and the statistics gathered during progress of decoding and encoding the current (second) pass, the transcoder module must determine how to change the coefficients a block before it is encoded. Recall from Section 2.2 that there are four basic parts to microscale rate control: the indicator determines when action is necessary, the quantifier calculates how much action is necessary, the selector chooses which parts of the image should receive action, and the actuator takes bitrate-reducing action. These four parts must operate intra-image, when a portion of the image has been transcoded and the remainder is unknown.

This section first approaches microscale rate control from a ρ-domain perspective. When this is discovered to produce unacceptable distortion artifacts in the image, we redefine the indicator. This new approach also proves unsatisfactory, and the algorithm described in Section 2.2 is finally tested and shown to produce the desired results.

3.2.1 The ρ-Cumulative Thresholding Algorithm

Keeping with the ρ-domain analysis that proved so successful for macroscale control, one could ask the question, "During the transcoding, given the cumulative ρ-domain slope in the decoded original image so far, and the cumulative number of zeroes encoded so far, what should the current number of bits in the encoded image be?" From this question, the definition of the indicator and quantifier is as follows:

$$\theta = \frac{R_d(n) - R_{min}(n)}{z_d(n) - z_{max}(n)} \tag{3.1}$$

$$R_{\rho t}(n) = \theta(z_e(n) - z_d(n)) - R_d(n) \tag{3.2}$$

$$\epsilon_\rho(n) = R_e(n) - R_{\rho t}(n), \tag{3.3}$$

where, at the n^{th} block in the image, $R_d(n)$ and $R_e(n)$ are the cumulative number of bits decoded and encoded, $z_d(n)$ is the cumulative number of zeroes, $z_{max}(n)$ is the maximum number of zeroes, and $R_{\rho t}(n)$ is the target intra-image bitrate. The error term $\epsilon_\rho(n)$ is both the indicator and quantifier and is called the ρ-domain cumulative indicator.

Because the target intra-image bitrate $R_{\rho t}(n)$ is dependent on the number of zeroes $z_e(n)$, the actuator should produce zeroes. This is called a thresholding actuator, which nullifies coefficients in reverse zig-zag scan order as long as the quantifier $\epsilon_\rho(n)$ is positive. The highest frequencies in a block will be zeroed before the low frequencies. However, it is possible that an entire DCT block may be thresholded to zero values. To prevent this, the selector employs the block mask in Figure 3-2 to preserve low-frequency DCT coefficients because the human eye is more sensitive to these components.

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

Figure 3-2: The block mask that preserves low-frequency components in a block of DCT coefficients. Coefficients eligible for thresholding are indicated by 1's in the mask.

Within a block, the block mask determines which coefficients are eligible for thresholding. The selector must also choose which blocks are eligible for thresholding. Blocks with high AC energy[1] in their DCT coefficients require many bits to encode and have many bits that we can discard. The selector picks the top one-eighth of all blocks with the highest AC energy using a heap data structure to keep track of these blocks. As one might imagine,

[1]AC energy is defined as the sum of the squares of the dequantized AC coefficients.

this heap may incur significant computation time and memory requirements, since there may be tens of thousands of blocks in a given JPEG image.

Together, the ρ-cumulative indicator, the high-energy high-frequency selector, and the thresholding actuator are called the ρ-cumulative thresholding microscale rate control algorithm.

3.2.2 The Bit-Cumulative Thresholding Algorithm

Note that the ρ-cumulative indicator depends entirely on the intra-image statistics of the image; in other words, the error $\epsilon_\rho(n)$ is only a function of the statistics up to the n^{th} block. We have not exploited knowledge of the original image statistics gained during the first decoding pass. For this reason, the ρ-cumulative indicator is prone to gross overestimations of the intra-image bitrate error term $\epsilon_\rho(n)$, especially in the early stages when n is small.

While it is unreasonable to store the decoded bits $R_d(n)$ for all n in the original image because that would require significant storage space, it is entirely reasonable and very beneficial to store the end-state of the image $R_d(N)$, the number of AC bits decoded at completion. From this logic, the bit-cumulative indicator is defined with an expected intra-image bitrate $R_t(n)$ that is proportional to $\frac{R_T}{R_d(N)}$, where R_T is the target number of AC bits of the newly encoded image at completion. The error term is $\epsilon(n) = R_e(n) - R_t(n)$, where $R_e(n)$ is the actual number of AC bits encoded at the n^{th} block. See Section 2.2 for a more detailed explanation of this indicator.

The next set of experiments uses this bit-cumulative indicator, keeping the high-energy high-frequency selector and the thresholding actuator.

3.2.3 The Huffman/Stuff Byte Algorithm

Zeroing coefficients causes a large change in value for the thresholded frequency component. We may need an actuator with a more subtle effect on the coefficient values that still produces savings in bits. We can exploit the entropy encoding of the JPEG bitstream to reduce the bitrate by merely decrementing certain DCT coefficients. The idea is that we reduce the size of the Huffman code and the corresponding residue string by decrementing a single coefficient, saving several bits in the process. In addition, the extraneous stuff bytes in the JPEG bitstream can be avoided, saving eight bits by flipping one bit. This defines a new selector and actuator, as described in Section 2.2. We continue to use the bit-cumulative indicator for these experiments.

3.2.4 Microscale Results

As with the macroscale experiments, we begin evaluating the microscale algorithms by examining their accuracy in reaching a target filesize. Figure 3-3 shows these results using macroscale ρ-domain analysis in combination with the three different microscale algorithms. The filesizes of images without microscale rate control are also plotted. Note that all microscale algorithms cause filesizes to fall below the corresponding filesize created without microscale rate control.

The ρ-cumulative algorithm tends to overcompress, producing errors greater than -10%. This is due to the fact that the indicator is poorly projecting the target intra-image bitrate and inducing more thresholding than necessary.

The bit-cumulative and Huffman/stuff byte algorithms perform remarkably better by using the bit-cumulative indicator. By using more information from the decoded image, a better estimate of the intra-image bitrate is projected, allowing the actuator to save bits at more necessary times and in more accurate quantities. Interestingly, the Huffman/stuff byte algorithm never exceeded the target filesize for all 29 images, while the bit-cumulative algorithm occasionally did not compress enough.

Looking at the filesize accuracy of the three options for microscale rate control, the bit-cumulative and Huffman/stuff byte algorithms produce the best results with most errors roughly between +5% and -10%. These two algorithms share the bit-cumulative indicator, implying the bit-cumulative approach produces more accurate results than the ρ-cumulative approach.

Intra-image AC bitrates are examined in Figures 3-4 and 3-5. Test image 14.jpg had its quantization tables scaled by 4, and the resulting filesize, 252KB, was used as a target for the encoded image. Intra-image bitrates as a function of the block number n were obtained during the transcoding of this image and are plotted in these figures.

The plot on the left of Figure 3-4 shows the intra-image bitrate of the original test image as a solid line. The dashed line represents the intra-image bitrate of the transcoded image without using an actuator—only macroscale rate control was performed by scaling the quantization tables with ρ-domain analysis. Note that the dashed line is roughly a scaled version of the solid line, demonstrating our assumption that the encoded image has

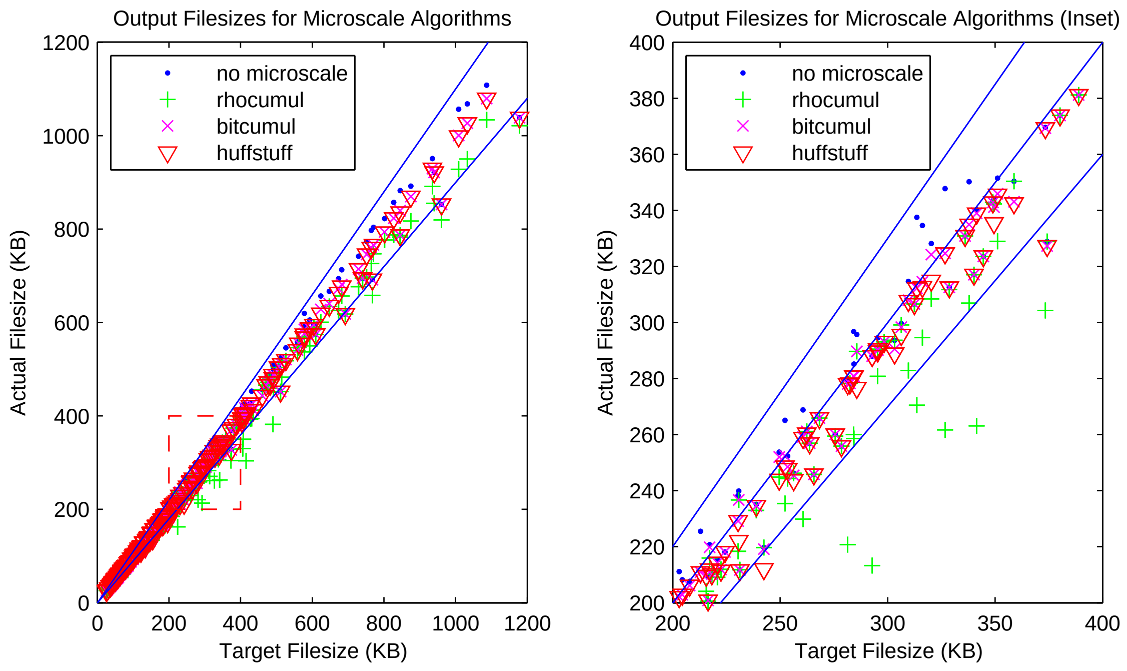


Figure 3-3: Scatter plots of output filesize vs. target filesize are shown. The ρ-cumulative algorithm tends to overcompress. The bit-cumulative and Huffman/stuff byte algorithms produce filesize errors roughly between 0% and -10%.

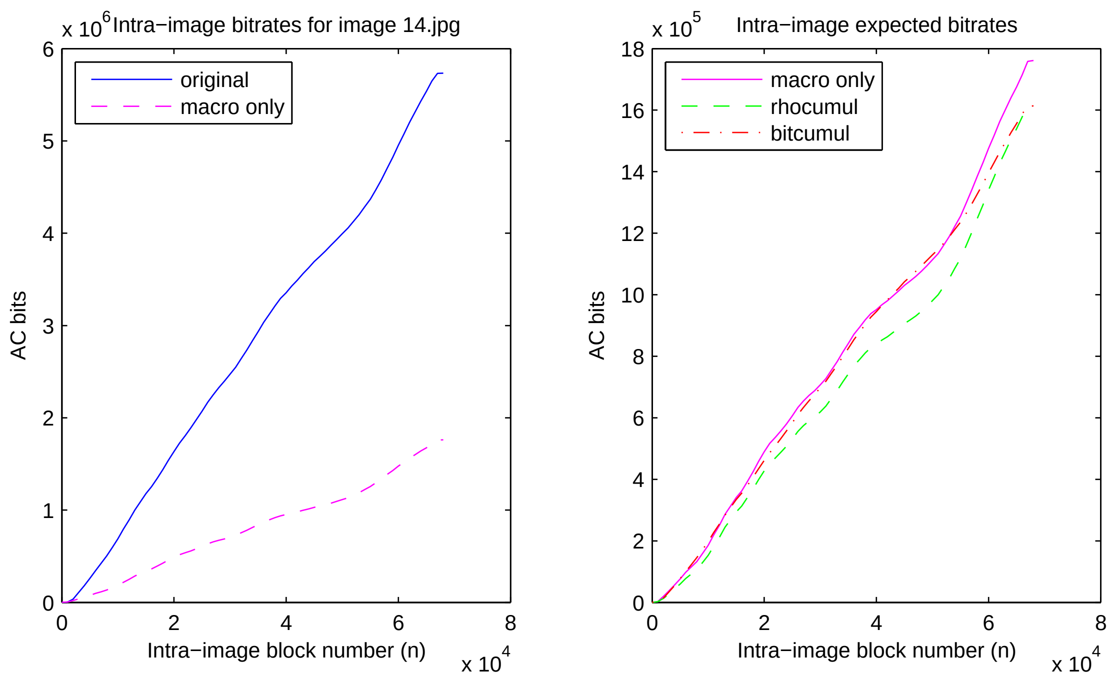


Figure 3-4: The actual and expected intra-image bitrates for image 14.jpg. The actual bitrates on the left plot show that the compressed image has bitrate roughly proportional to the original image. The expected bitrates on the right plot converge to the same final value.

intra-image bitrates proportional to the original image.

Figure 3-4 shows the expected bitrates using the two indicators we have defined. On the right, the same macroscale-only bitrate line is now shown as a solid line. The dashed line plots the expected bitrate $R_{\rho t}(n)$ as defined in Equation 3.2. The dash-dotted line plots the expected bitrate $R_t(n)$ as defined in Equation 2.9. Both the expected bitrates converge to the same value at the last block, namely the target bitrate. Convergence to the target bitrate should be a necessary condition for every indicator, since we would like the transcoded image to meet the target bitrate at completion. Note that the ρ-cumulative indicator (dashed line) in this case is always less than the actual number of bits encoded (solid line), and that the bit-cumulative indicator (dash-dotted line) is sometimes greater than the actual bits encoded. The errors between actual and expected bitrates are plotted in Figure 3-5.

The first plot in Figure 3-5 shows the error $\epsilon_\rho(n)$ as defined in Equation 3.3 when only macroscale rate control has been applied. The dash-dotted line plots $\epsilon_\rho(n)$ with the selector and actuator applied. Since the error is positive, the ρ-cumulative microscale rate control algorithm does its best to reduce the error but cannot eliminate it at any point in the image.

The second plot in Figure 3-5 shows the error $\epsilon(n)$ as defined in Equation 2.10 when only macroscale rate control has been applied. The dash-dotted line plots $\epsilon(n)$ with the selector and actuator applied. Thresholding is applied only when the error is positive, returning the error to zero, as is the case around the $20{,}000^{th}$ block. Thereafter, the two plots with and without thresholding are parallel until the error becomes positive again. The bit-cumulative microscale algorithm cannot completely eliminate the error by the end of the transcoding process.

The third plot in Figure 3-5 shows the same dashed line as the second plot, since the Huffman/stuff byte algorithm uses the same indicator as the bit-cumulative algorithm. The dash-dotted line plots $\epsilon(n)$ with the selector and actuator applied. This line is similar to the dash-dotted line of the second plot, but the error at the final block is much smaller.

By examining the intra-image expected bitrates and error terms, we can get a sense of the behavior of each microscale rate control algorithm. The ρ-cumulative algorithm tends to overestimate the error. In addition, since the intra-image target bitrate $R_{\rho t}(n)$ depends on $z_e(n)$ from Equation 3.2, any actuator that changes $z_e(n)$ will affect future values of $R_{\rho t}(n)$. The intra-image target bitrate should not change as a consequence of the actuator,

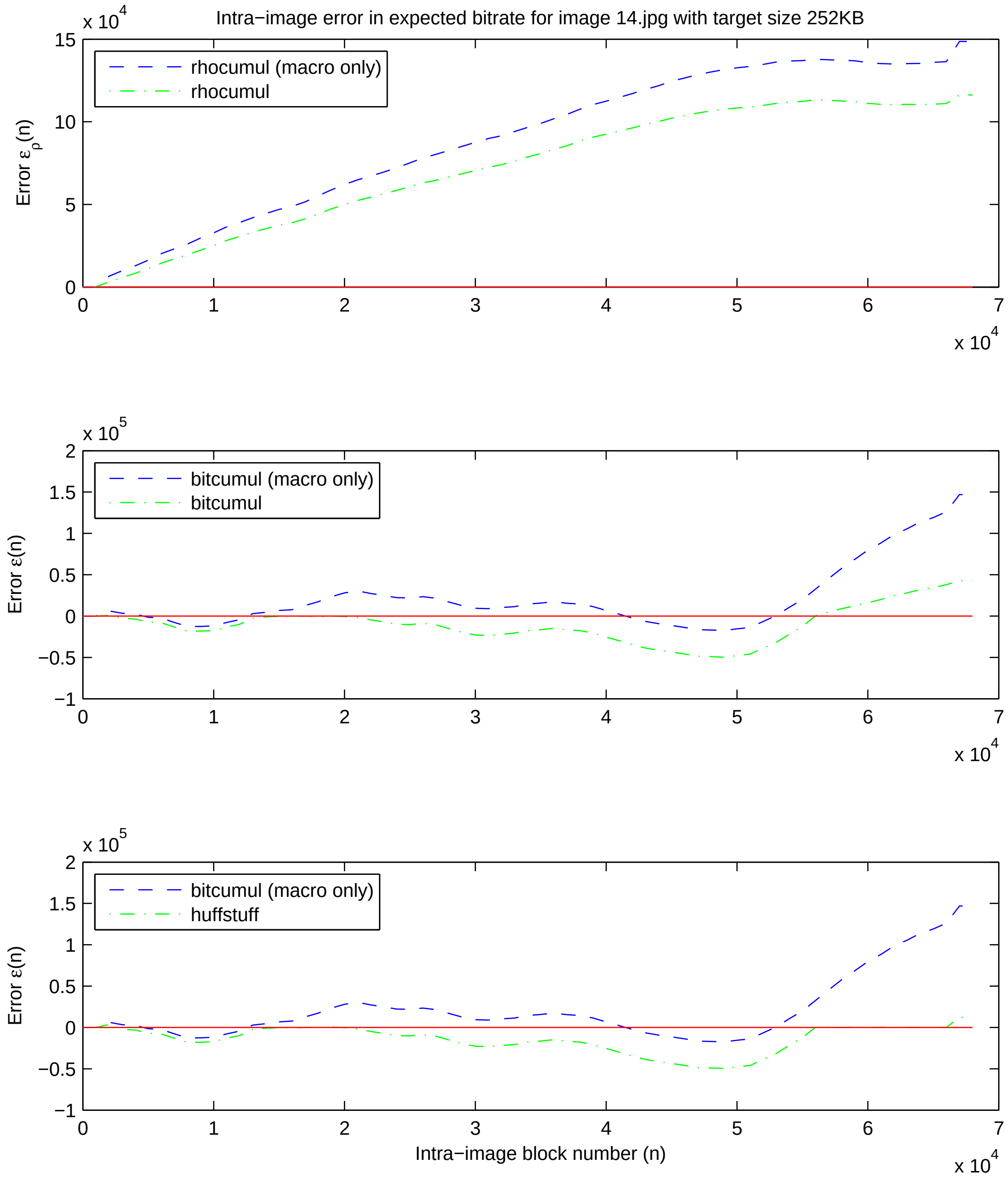


Figure 3-5: The intra-image error calculated as the difference between actual bitrate and expected bitrate. The bit-cumulative indicator produces error values near zero in the second and third plots. The Huffman/stuff byte actuator has the smallest final error in bitrate.

otherwise we are not guaranteed to converge to the final target bitrate. The bit-cumulative target bitrate $R_t(n)$ from Equation 2.9 is unaffected by anything the actuator can modify, guaranteeing convergence to the final target bitrate. The Huffstuff/stuff byte actuator appears to obtain faster convergence than the thresholding actuator.

Next, the image quality produced by each microscale option is analyzed using peak signal-to-noise ratio (PSNR). The luma component receives special attention when calculating PSNR because the human visual system is more sensitive to brightness rather than color. Given the luma components of the original image I_Y, a transcoded image i_Y, and a maximum pixel value of 255, the PSNR is defined as:

$$\begin{aligned} MSE &= (I_Y - i_Y)^2 \\ PSNR &= 20 \log_{10}\left(\frac{255}{\sqrt{MSE}}\right), \end{aligned} \tag{3.4}$$

Figure 3-6 samples four images out of the test set of 29 and plots PSNR against filesize for each microscale rate control algorithm. The control image (uniform quantization) and macroscale-only image have their PSNR plotted for reference.

In most cases, the Huffman/stuff byte algorithm performs slightly better than the bit-cumulative algorithm. This result verifies the claim the decrementing DCT coefficients usually produces less overall distortion than zeroing the coefficients.

The ρ-cumulative option regularly produces the lowest PSNR, as can be expected from the greater compression seen in Figure 3-3. Likewise, the macroscale-only image has the best PSNR because every microscale option shaves bits from the macroscale-only image. In addition, the macroscale-only image has better luma PSNR than the control image because the quantization tables have been scaled to favor the luma component.

The anomaly in image 14.jpg at filesize 430KB is probably explained by a circumstantial occurrence of many powers of 2. This creates many opportunities for the actuator to decrement coefficients and, therefore, increase the mean squared error (MSE).

Quantitatively, the Huffman/stuff byte algorithm appears to produce images with good PSNR. Qualitatively, this algorithm also shows fewer compression artifacts than the other two options. Figure 3-7 shows a cropped portion of image 14.jpg using the three microscale rate control algorithms. Note the discontinuities and compression artifacts on the strokes

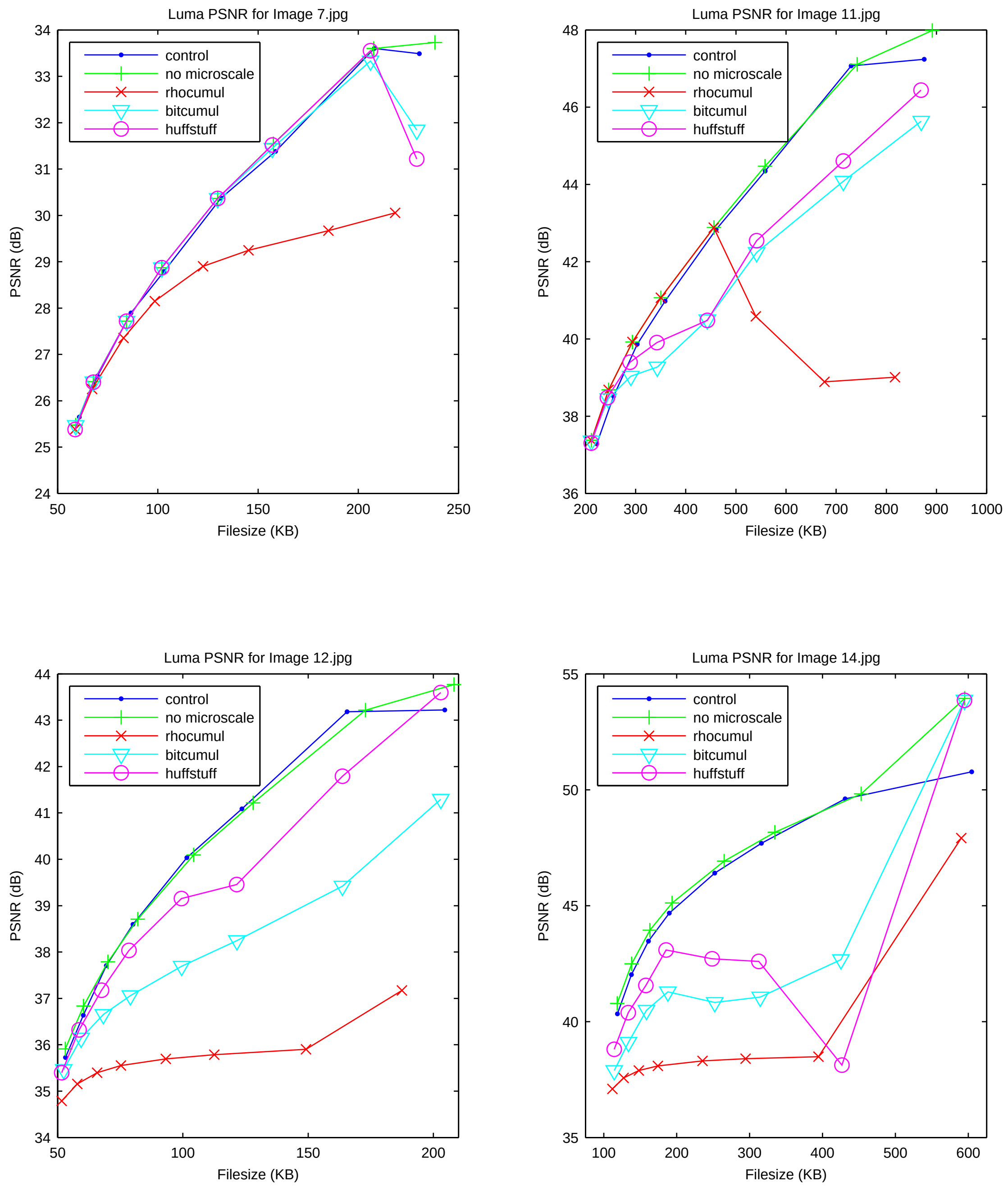


Figure 3-6: PSNR of the luma component vs. output filesize for four images using various miocroscale rate control algorithms. In most cases, the highest PSNR was produced without microscale control. Images from the Huffman/stuff byte algorithm usually had higher or comparable PSNR to images from the bit-cumulative algorithm. The ρ-cumulative algorithm consistently produced the lowest PSNR.

of the ‘M’. These distortions appear less severe in the image produced by the Huffman/stuff byte algorithm. This compression artifacts shown in this cropped image is representative of those seen in many other images in the test set.

The Huffman/stuff byte algorithm appears to have many advantages over the other microscale rate control options. First, because the actuator performs on powers of 2, there is no need for a large heap data structure to keep track of blocks with high AC energy, as would be required for both the ρ-cumulative and bit-cumulative options. Second, the filesizes produced are reliably accurate and reasonably precise. Third, the PSNR of the transcoded images are comparable to or better than the PSNR produced with other algorithms. And fourth, the subjective image quality has fewer perceived compression artifacts.

3.3 Summary

Using a test set of 29 images for a representative sample of JPEG images that exist, we conducted experiments on various rate control algorithms. Control images were created by uniformly scaling the quantization tables of an image by eight known constants. All subsequent experiements used the filesizes of these control images as their target filesize. All subsequent comparisons between algorithm results are with reference to these control images as well.

First, experiments on scaling the quantization tables were conducted to determine the best macroscale rate control algorithm. The Qualcomm Model, Sherlock Formula Model, and ρ-domain analysis were tested, and ρ-domain analysis achieved, by far, the most accurate output filesizes.

Second, experiments on intra-image bit reduction were conducted to determine the best microscale rate control algorithm. The ρ-cumulative, bit-cumulative, and Huffman/stuff byte algorithms were tested, and the last performed very well on several fronts. The Huffman/stuff byte algorithm has low memory requirements, achieves accurate and precise output filesizes, produces high PSNR, and exhibits minimal compression artifacts.

Together, ρ-domain analysis for macroscale rate control and the Huffman/stuff byte algorithm make up the best solution for the rate control and bit allocation problem of JPEG transcoding, achieving a target filesize accurately and efficiently, and minimizing distortion and compression artifacts.

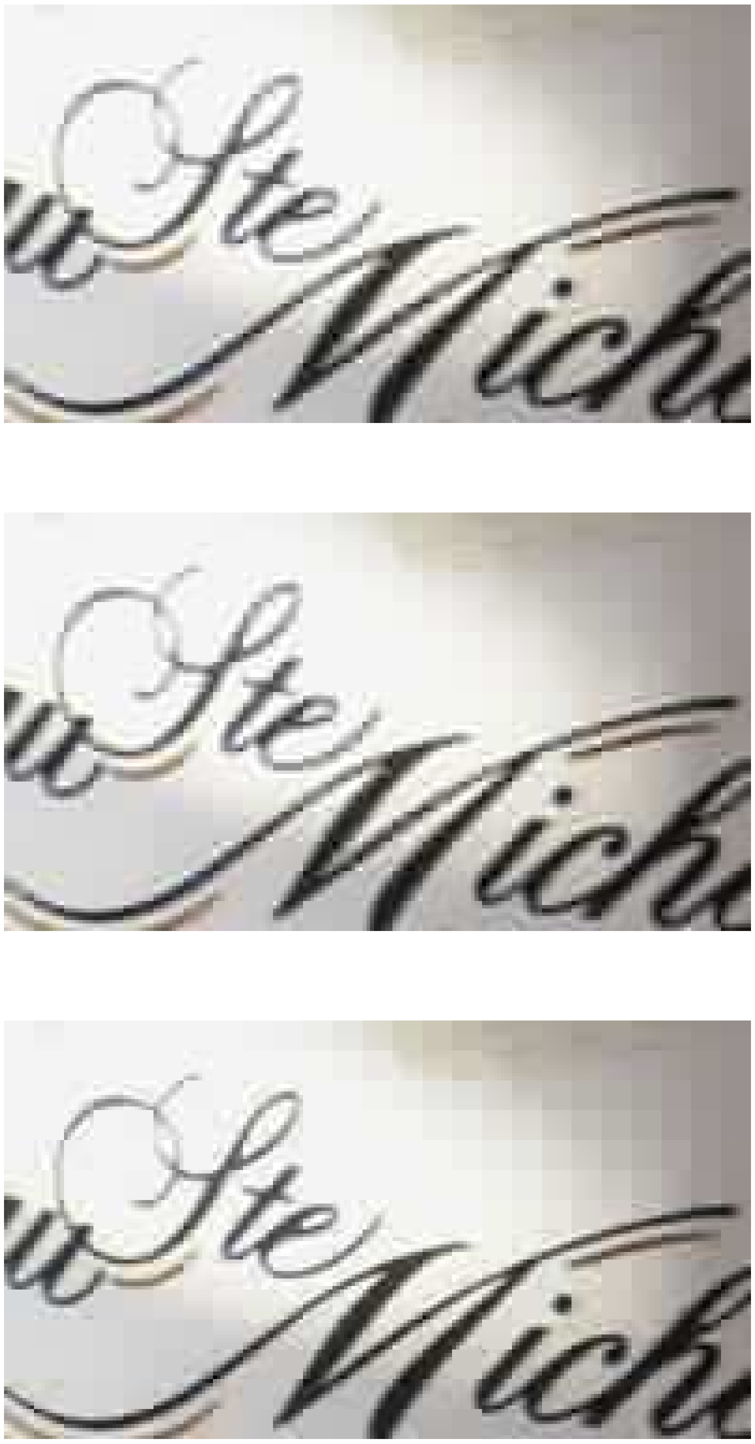


Figure 3-7: This image was cropped from image 14.jpg after it was transcoded with the ρ-cumulative (top), bit-cumulative (middle), and Huffman/stuff byte (bottom) algorithms. Compare the discontinuities and compression artifacts on the strokes of the ‘M’.

Chapter 4

Conclusion

Digital images have become a ubiquitous part of everyday life, becoming as much a means of communication as letters, phone calls, or emails. Wireless carriers and handset manufacturers are engineering solutions to allow picture messaging over wireless networks. In such applications, resources such as bandwidth, airtime, processor speed, and device memory are limited. At the same time, camera phones are becoming more capable of producing high quality images. Thus, the need arises to store and share these images without consuming scarce resources. The JPEG transcoder designed in this paper addresses this issue by providing a fast and memory-efficient utility for compressing a given JPEG image to a target filesize while maintaining the best quality possible.

To develop this solution, an entirely new decoding and encoding framework was built to avoid the pitfalls of bloated third-party code. First and foremost, the framework provides only essential functionality, which means decoding and encoding JPEG baseline images without DCT calculations and only operating on the frequency domain DCT coefficients. To gather statistical information before transcoding, the framework allows a first pass for decoding before a second pass for transcoding and encoding.

Furthermore, the framework needed the flexibility to experiment with different algorithms, so a modular design was used. The decoder publishes blocks to a statistics collector and to the transcoder. The transcoder uses information from all statistics collectors to publish a transformed block to the encoder. The encoder finally writes the block to a file, publishing the block to another statistics collector. This publishing technique uses the observer design pattern. Also, the strategy design pattern allowed the substitution of various algorithms to perform the same task. The transcoder conceptually operates on two tiers of rate control: macroscale rate control by modifying the quantization tables; and microscale rate control by modifying the DCT coefficients intra-image.

Numerous experiments were performed using this transcoding framework on a test set of 29 images that gave a fair representation of a variety of typical JPEG photos. Control images were created by uniformly scaling the quantization tables of each test image by eight known constants, producing eight smaller files for each test image. The two tiers of rate control were approached separately. The evaluation metric consisted of three parts: first, the accuracy and precision of the output filesize with respect to the target filesize; second, the PSNR of the output image with respect to the original image; and third, the subjective visual image quality.

Building upon the work of He and Mitra [1], the JPEG transcoder exploits a linear relationship between the number of zero-valued quantized DCT coefficients and the bitrate. Using this relationship and a histogram of coefficients, it is possible to determine the best way to scale the quantization tables of an image to approach a target filesize. As the image is being transcoded, an intra-image process makes minor corrections, saving more bits as needed throughout the transcoding of the image. This intra-image process cleverly decrements specific coefficients, minimizing the change in value while maximizing the savings in bitrate. The result is a JPEG transcoder that is fast, reliable, accurate and efficient at achieving a target filesize and preserving as much image quality as possible.

Although much progress has been made and great results were achieved, there is still much room for future work in every aspect of the transcoder. For macroscale rate control, one could imagine changing the strategy for picking scaling factors in the quantization tables to achieve a target number of quantized zeroes.

For microscale rate control, the bit-cumulative algorithm showed much promise. Perhaps a better selector could be designed. Instead of using a heap to keep track of the highest energy blocks in the image, just determine a threshold as a percentage of the highest AC energy; if a block has more AC energy, it is eligible for bitrate reduction. This eliminates the memory requirement and computational complexity needed for a heap. The selector may also use a different block mask, perserving more or fewer coefficients, or even preserving the set of horizontal, vertical and diagonal frequency components. A new indicator could be defined, but it appears that the bit-cumulative indicator was robust enough to produce accurate results for two different actuators.

The PSNR is a commonly used objective metric for image quality. But its does not accurately reflect the human visual system. Measuring luma PSNR does not consider color information, which still contributes significantly to image quality. Perhaps weighting the components before computing PSNR would address this. Also, the perception of low- or high-frequency components was not considered in the PSNR metric. Maybe an entirely new metric should be designed, perhaps something like joint entropy.

Finally, this work has proven that a JPEG transcoder can produce a high-quality image of a desired filesize quickly, reliably, predictably, accurately and efficiently. The parts for designing such a transcoder have been identified and a framework was established for verifying the performance of these parts. A good solution was found, and the stage is set for a better solution to be discovered.

Appendix A

JPEG Baseline Compression

A raw image to be encoded is first converted to the YCbCr colorspace. Because the human visual system is more sensitive to brightness (luminance Y) than chrominance (color along the blue Cb or red Cr axis), the YCbCr colorspace allows compression to retain more important luma information and discard less important chroma information. For the remainder of compression, the components each go through the following steps.

Each component may be optionally downsampled. Downsampling is usually done by a factor of two in either or both of the horizontal and vertical directions. Since chroma information is less important, these components are usually downsampled. Common downsampling ratios are expressed as H1V1 (no downsampling), H2V1 (chroma downsampled in horizontal direction), and H2V2 (chroma downsampled in both vertical and horizontal directions).

The component is then tiled into 8x8 pixel blocks so that a 2-dimensional 8x8 discrete cosine transform (DCT) may be applied. Specifically, the pixel values are level-shifted to produce positive and negative values centered at 0, then the DCT type-II is applied. This is a common transform used in image and video compression because most of the energy of the 8x8 block is compacted into a few coefficients.

After the DCT, the 64 coefficients are scaled by entries in an 8x8 quantization table. This allows the encoding of a coefficient "level" as opposed to the actual value itself. Since the human eye is more sensitive to low frequency components, typical quantization tables provide many levels for low frequencies and few levels for high frequencies. This quantization is the most significant lossy step in JPEG compression. Usually there exists one quantization table for the luma component and one for both chroma components, where the chroma table often has larger step sizes to discard more chroma information. The JPEG standard provides recommended default tables; encoders may use their own custom tables.

The DCT produces one DC component and 63 AC components for each 8x8 block. The DC component is differentially encoded. Only the difference between the current DC value and the last DC value for a component is encoded. For continuous-tone images, these differences are usually small in magnitude. The AC components in the 8x8 block are ordered according to a zig-zag scan so that the low frequency coefficients are grouped at the beginning of the scan and the high frequencies at the end. The AC coefficients are then zero run-length encoded because most AC values quantize to zero.

The entropy encoding of DC and AC coefficients are similar in that first a "value" is Huffman encoded and then a "residue" is appended. For DC, the "value" is the size, in bits, of the magnitude of the DC difference. The "residue" is a unique identifier for DC differences of that size. For example, the DC difference -6 has size 3 because it takes 3 bits to represent 6. Since a difference of size 3 can by any of (-7, -6, -5, -4, 4, 5, 6, 7), the number -6 is identified by the 3-bit residue "010", which is binary for 2 because -6 is the second-lowest difference of size 3. So to encode a DC difference of -6, the encoder emits the Huffman code for size 3, and then emits three bits "010".

For AC run-lengths, the "value" is the concatenation of a "run" and a "size". The "run" is the number of consecutive zeros in the zig-zag order of AC coefficients. The "size", as before, is the size, in bits, of the subsequent nonzero coefficient. The residue is the same as the DC case. For example, suppose the zig-zag scan contains the sequence "0003". This produces a run of 3 and a size of 2 with residue "11". The encoder emits the Huffman code for a run/size of 3/2, and then emits two bits "11".

The encoding of AC run-lengths also has two special symbols. The end-of-block (EOB) symbol means the remainder of the block consists of only zeroes and is designated with a run/size of 0/0. Since the size is 0, no residue is emitted. The other special symbol is the zero-run-length (ZRL) value, which means there are 16 consecutive zeroes in the zig-zag scan. Since run and size are each represented by up to 4-bits, the maximum value of run is 15. It makes sense, then, that the ZRL run/size symbol is 15/0 (i.e., 15 zeroes followed by a 0). Again, since size is 0, no residue is required.

Four standard Huffman tables are defined in the JPEG standard, one each for DC luma, DC chroma, AC luma, and AC chroma components. These standard tables define a code for every possible symbol, and although they are suboptimal, they provide complete coverage.The entropy decoding and encoding of an image may use the standard Huffman tables, or other tables which may be optimized for the statistics of the image. The Huffman tables used to encode a JPEG image are always included in the headers of the JPEG file.

The entropy encoding of components is usually interleaved into what is called a minimum coded unit (MCU). An MCU consists of up to 10 data units, the term for a block of 64 DCT coefficients. (The terms “block” and “data unit” will be used interchangeably.) For example, if the downsampling is H1V1, one Y data unit is encoded, followed by one Cb data unit and one Cr data unit; the three data units together form one MCU. Likewise for H2V1, two Y data units, one Cb unit and one Cr unit comprise one MCU; and for H2V2, four Y, one Cb and one Cr.

Metadata that stores information about the image, such as dimensions, downsampling ratios, quantization tables, and Huffman tables, are also stored in a specific format in the JPEG file. These regions of metadata are identified by two-byte markers that begin with the byte 0xFF. In case the entropy encoding of coefficients happens to produce a byte-aligned 0xFF, a zero byte 0x00 is “stuffed” to disambiguate the entropy bitstream from a JPEG marker (since no marker 0xFF00 is defined).

The standard defines optional restart markers intended to help decoders recover from errors. After encoding a specified interval of MCUs, the encoder fills the last byte with 1’s, and emits a restart marker, informing the decoder to restart the DC difference calculations at 0. Often the restart interval is the number of MCUs in one row of the image. This allows DC differences to start anew for each row.

Appendix B

Test Set of Images

1.jpg (3008x2000)

2.jpg (3008x2000)

3.jpg (1000x750)

4.jpg (2877x2304)

5.jpg (1600x1069)

6.jpg (3008x2000)

7.jpg (1600x1047)

8.jpg (1000x800)

9.jpg (1944x2592)

10.jpg (1024x768)

11.jpg (2006x1532)

12.jpg (1024x664)

13.jpg (1280x853)

14.jpg (1051x1378)

15.jpg (800x590)

16.jpg (800x600)

17.jpg (1024x1024)

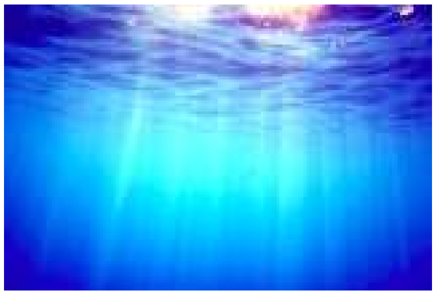
18.jpg (1165x776)

19.jpg (1024x768)

20.jpg (1389x1281)

21.jpg (1024x641)

22.jpg (2160x1440)



23.jpg (3000x1024)

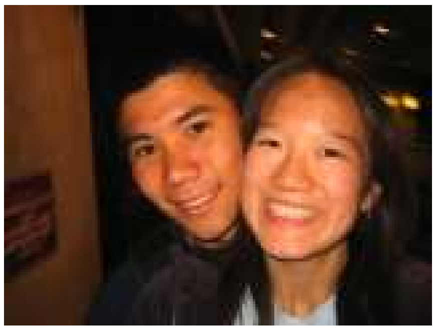
24.jpg (1280x960)

25.jpg (1280x853)

26.jpg (2393x1836)

27.jpg (1280x960)

28.jpg (2592x1944)



29.jpg (2592x1944)

Bibliography

[1] Zhihai He and Sanjit K. Mitra. A linear source model and a unified rate control algorithm for DCT video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(11):970–982, November 2002.

[2] Zhihai He and Sanjit K. Mitra. Optimum bit allocation and accurate rate control for video coding via ρ-domain source modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(10):840–849, October 2002.

[3] Independent JPEG Group. *JPEG Image Compression Library.* Available at `http://www.ijg.org`.

[4] ISO/IEC 10918-1, ITU-T Rec. T.81. *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*, 1993.

[5] ISO/IEC 10918-3, ITU-T Rec. T.84. *Information Technology – Digital compression and coding of continuous-tone still images: Extensions*, 1997.

[6] Yong Kwan Kim, Zhihai He, and Sanjit K. Mitra. A novel linear source model and a unified rate control algorithm for H.263 / MPEG-2 / MPEG-4. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1777–1780, May 2001.

[7] Kannan Ramchandran and Martin Vetterli. Rate-distortion optimal fast thresholding with complete JPEG/MPEG decoder compatibility. *IEEE Transactions on Image Processing*, 3(5):700–704, September 1994.

[8] B. G. Sherlock, A. Nagpal, and D. M. Monro. A model for JPEG quantization. In *International Symposium on Speech, Image Processing and Neural Networks*, volume 1, pages 176–179, April 1994.

[9] Qualcomm CDMA Technologies. *Model driven JPEG file size control.* QUALCOMM Incorporated, March 2004.

[10] Siu-Wai Wu and Allen Gersho. Rate-constrained picture-adaptive quantization for JPEG baseline coders. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 389–392, April 1993.