7. Appendix

In order to give the reader a flavor of how the JPEG algorithms affect the reconstructed images, we attached three pairs, an original image and its difference image generated from our simulation. In the beginning, we tried to take photographs of the original images and the decoded images. But it turns out that it is difficult to see on the photographs the differences between the original image and the decoded image, although they are visible on the screen. Therefore, instead, we have attached pictures of the *difference* between the original image and the decoded image. The original images are visually better on the screen than on the photograph due to limitations of the photographic techniques.

Most of the reconstructed images are blurry compared to the original ones. The example luminance quantization table (see Table 3.2) has a better impact on the images *Tree*, *Sailboat*, *Airplane* and *Aerial* than the images *House*, *Splash*, *Girl*, *Lenna* and *Tiffany*. There are visible block edge effects and obvious blurring in the latter decoded images. Since the absolute difference images are too obscure to show on the photographs, we *exaggerated* these differences in the following way:

- For each pixel compute a *diff*, the absolute value of the difference between the original image and the decoded image.
- In a given image, there are very few *diff*s in the range 128~255. Map those values to pixel value 255. On the screen those will show up as very bright spots, but on the photographs they are not so obvious.
- Find the minimum and maximum of all the *diff*s in a given image.
- Exaggerate the differences. If a *diff* is in the range 0~127, then map it to 0~255 as follows:

$$255 * (diff - min)/(max - min)$$

In the exaggerated difference images it is easy to see the effects of the JPEG algorithms.

[TLR85] S. Todd, G. G. Langdon, Jr., and J. J. Rissanen. Parameter reduction and context selection for compression of gray-scale image. *IBM J. Res. Develop.*, 29(2), March 1985.

[Wal91] G. K. Wallace. The jpeg still picture compression standard. *CACM*, 34, April 1991.

[Wel84] T. A. Welch. A technique for high performance data compression. *IEEE Computer*, 17(6):8 – 19, June 1984.

[Wil89] Ross. N. Williams. *Adaptive Data Compression.* PhD thesis, Department of Computer Science, University of Adelaide, June 1989. (It is going to be published by Kluwer Academic Publishers in 1991).

[WNC87] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. *CACM*, pages 520 – 540, June 1987.

[(Zh90] Manyun Chang (Zhang). The jpeg and image data compression algorithms. Master's thesis, UCSC, 1990.

References

[Cha90] G. K. Wallace JPEG Chair. Overview of the jpeg (iso/ccitt) still image compression standard. *SPIE Image Processing Algorithm and Techniques*, 1244, 1990.

[CL91] Manyun Chang and Glen G. G. Langdon, Jr. Effects coefficient coding on jpeg baseline image compression. In J. A. Storer and J. H. Reif, editors, *Proc. DCC'91*, Snowbird, Utah, April 8-11 1991. IEEE Computer Society. Poster session, abstract only.

[CW84] J. Cleary and I. Witten. A comparison of enumerative and adaptive codes. *IEEE Trans. on Info. Theory*, IT-30(2):306 – 315, March 1984.

[Hua90] A. Huang. Image compression: The emerging standard for color images. *Computer Future*, 13, 1990.

[ISO90] ISO/IECJTCI/SC2/WG8 – CCITT SGVIII. *JPEG Technical Specification Revision 5.2*, jpeg-8-r5.2 edition, May 1990. JPEG stands for Joint Photographic Experts Group.

[Lan] G. G. Langdon, Jr. Further developments in lossless gray-scale image compression. IBM Internal Conference on Pattern Recognition and Image Processing, Yorktown Heights New York, November, 1984. (Available as IBM Research Report RJ-6426).

[Lan90] G. G. Langdon, Jr. On adaptation to multiple-context binary sources. Technical Report RJ 7845, IBM Research Division, Almaden Research Center, San Jose, California 95120-6099, November 1990.

[Lig91] Adriaan Ligtenberg. Jpeg++: Selective compression for high quality color desktop publishing. In *COMPCON SPRING 91*, February 1991.

[Nga84] K. N. Ngan. Image display technique using the cosine transform. *IEEE Trans. Acoust., Speech, Sig. Proc*, ASSP-32(1), 1984.

[Ore81] T. I. Ored. Concepts and criteria to assess acceptability of simulation studies: A frame of reference. *CACM – Special Issue on Simulation Modeling and Statistical Computing*, 24, April 1981.

[PBW79] O. Pahlm, P. O. Borjesson, and O. Werner. Compact digital of ecg's. *Compact Programs Biomed*, 9, 1979.

[PM89] W. B. Pennebaker and J. L. Mitchell. Standardization for color image data compression part i: Sequential coding. Preprint of Summary of Invited Talk to be Given at Electr. Imaging '89 East Boston, MA, October 1989.

[PMLA88] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps. An overview of the basic principles of the q-coder adaptive binary arithmetic coder. *IBM Journal Research and Development*, pages 717 – 726, November 1988.

[RL81] J. J. Rissanen and G. G. Langdon, Jr. Universal modeling and coding. *IEEE Trans. Infor. Theory*, IT-27(1):12 – 23, January 1981.

[ST79] K. R. Sloan and S. L. Tanimoto. Progressive refinement of images. *IEEE Trans. Comput.*, C-28(11), 1979.

6. Future Work

Additional studies and research can build upon the results from our simulation. Here we list a few of them:

- Identify the image type and compression gain. From the simulation we can see that different kinds of image affect the performance of the JPEG algorithms. If we can further explore this phenomenon, we might be able to classify the image types.
- Improve the example quantization table that included in the JPEG according to the image type. Once we understand the image type, we can design a suitable quantization table for those types image.
- Further study on the arithmetic coding. We adopted the code from the paper [WNC87] and got fairly good results. In the JPEG [ISO90], it proposes the QM-coder as the implementation for its extended system. Since we did not have time to implement the QM-coder, we used [WNC87] instead. The simulation of the QM-coder and a comparison of the performance [WNC87] and UNIX *compress* is worthwhile.
- The compression gain calculation for the Huffman code assumed the table had already been transmitted to the decoder. The results should be calculated for the signaling overhead of transmitting the table, to see the effect on the gain.
- Rather than generating our own Huffman codes, we adopted the Huffman example tables from the draft [ISO90]. It would be valuable to determine the gain for using image-specific Huffman tables.
- The simulation method applies to studying the interaction of statistical compression gain and the quantization Table 3.2, with a comparison of image quality between the original and reconstructed image. This application should be explored relative to image types.
- Progressive image transmission algorithms based on the DCT algorithm can be built based on our simulation. They are an important extension of the JPEG algorithms for certain applications.

6.1 Summary

A method of decomposing and analyzing the compression of the JPEG algorithm was presented. Using counting methods, estimates on enumerative code lengths for various quantization tables can be readily obtained over typical images for a particular application by a simulation without actually compressing the images. Based on our simulation we are convinced that the JPEG is fairly good in terms of its compression performance. Our experiments also show that although the JPEG algorithm was selected over many candidate algorithms, how to apply the JPEG algorithms to a particular application by selecting the quantization levels and coding tables with some "cleverness" is still a challenging task.

5.3 Performance of the JPEG Sequential Baseline

In the previous section, we mentioned the compression performance that JPEG claims in the draft 5.2 [ISO90]. The results that we obtained from this simulation are provided in Table 5.5. The notation bits/p denotes the compression result measured in bits per pixel for the compressed image.

Table 5.5 shows that that following quantization, run-length and Huffman coding, the results are close to what JPEG expects, which is less than 1 bit/pixel. (see Table 5.5).

5.4 Conclusions

Table 5.1 shows the coefficient quantization provides the least gain, and the run length coding the most. However, it is the application of the DCT followed by quantizing that many coefficients are zero, which accounts for the success of run length encoding. It is interesting to note that the compression gain of the run length code may serve as a measure of image smoothness, the lower the gain the greater the fine detail of the image. The gain is as high as 12.25 for *splash* and as low as 4.02 for the small *sailboat* image. The statistical step of the categories and extra bits does not seem to vary as much from image to image, remaining in the vicinity of 2.2 to 2.5.

We also observe the following "conclusions" from this simulation: for the larger images, the JPEG algorithms have a better performance on the average, perhaps due to a finer granularity of the image covered by each pixel. On the average, the arithmetic coding is about 7 $\sim$ 8% better than the Huffman coding, depending on the image.

For examining the quantization errors, we generated the difference images of our image test set. As expected from higher quantization levels of AC coefficients, the *lost* information is mainly distributed in the edges of the image.

Quantization table 3.2 that JPEG provides filters out most of the high-frequency coefficients, although we should say it is a fairly good quantization table for the images on the average. In the simulation we realize that this quantization table results in some of the images in our test set losing their sharpness on certain parts with high details. A customized quantization table would give us a high quality reconstructed image. Our conclusion is that the JPEG baseline system algorithm with the example quantization table causes the majority of our tested images to lose their sharpness. We also noticed that for the natural scene images such as *sailboat, chemical_plant, and tree*, the performance of the JPEG algorithms is almost the same. For the low-detail image such as *splash* (the content of the image is just the water splashed from a container). compression performance is better than the high-detail image, the reconstructed image has the visible block effect.

5.2.1 Application of the [WNC87] Arithmetic and LZW Lossless Coding Algorithms

Table 5.5: The Results of Lossy and Lossless Compression

Image Name	Original File	Run (byte)	Rate (b/p)	Huffman (byte)	Rate (b/p)	Compress (byte)	Rate (b/p)	Arithmetic (byte)	Rate (b/p)
Sailboat	16384	3090	1.51	1846	0.90	2137	1.04	1861	0.90
Sailboat	65536	9949	1.21	5909	0.72	6399	0.78	5759	0.70
Chemical	65536	9776	1.19	5705	0.70	6213	0.76	5528	0.67
Tree	65536	9912	1.21	5841	0.71	6363	0.78	5718	0.70
Girl.1	65536	5922	0.72	3294	0.40	3620	0.44	3146	0.38
House	65536	5412	0.66	3213	0.39	3211	0.39	2944	0.36
Clock	65536	5830	0.71	3477	0.42	3528	0.43	3240	0.40
Splash	262144	16212	0.49	9295	0.28	8099	0.25	7900	0.24
Tiffany	262144	19723	0.60	10934	0.33	10712	0.33	9884	0.30
Lenna	262144	22564	0.69	12806	0.39	12980	0.39	11922	0.36
Airplane	262144	25002	0.76	14393	0.44	14108	0.43	13404	0.40
Aerial.9	262144	42532	1.30	24937	0.76	26073	0.80	23962	0.73

A program to implement a one-pass adaptive arithmetic coding appears in the paper [WNC87]. The adaptive statistics routine (`update_model.c`) is called both in encoding and decoding processes. In `model.h` (see the following code) the variable `No_of_chars` defines the maximum possible number of unique symbols during the adaptive coding process. Since the code is designed for the symbol with 8 bits, originally this variable is 256. In the sequential baseline system we know that the data ranges for the DC and AC coefficients are less than 256. Therefore, we set `No_of_chars` to 12 for the DC coefficients, and 162 for the AC coefficients. The compression performance is improved significantly over the settings of 256, due to the faster adaptation. For example, for the image *sailboat*, if `No_of_chars` is 256 for the DC coefficient, the result is 3328 bits; if it changes to 16, the result is 2944 bits; furthermore, if it changes to 12, the result is 2920 bits.

```
#define No_of_chars 256  /* Number of bits in a code value */
#define EOF_symbol (No_of_chars+1) /* Index of EOF symbol */
#define No_of_symbols (No_of_chars+1) /* Total number of symbols */
#define Max_frequency 16383 /*Maximum allowed frequency count, 2^14-1*/
int char_to_index[No_of_chars]; /*To index from character*/
unsigned char index_to_char[No_of_symbols+1]; /*To character from index*/
int cum_freq[No_of_symbols+1]; /*Cumulative symbol frequencies*/
```

<`model.h`>

The simulation results on the performance of the arithmetic coding and LZW algorithm for the statistical coding, are listed in Table 5.4. In Table 5.4, column *Enu* and *Ideal* show that better than 2:1 compression gain is available for the statistical coding. From the simulation, we can see that adaptive arithmetic coding performs better than the LZW algorithm for the image compression.

Table 5.3: Performance Evaluation of Huffman coding (No Extra Bits)

Image Name	DC_huff (bit)	DC_enu ((bit)	DC_id (bit)	AC_huff (bit)	AC_enu (bit)	AC_id (bit)
Sail128	771	699	682	8829	8318	8279
Sail256	3102	2895	2866	27988	26094	26022
Chemical	2999	2682	2656	27815	25806	25738
Tree	3024	2893	2866	27468	25720	25652
Girl.1	3006	2683	2658	14967	13257	13214
House	2806	2629	2603	15398	12651	12595
Clock	2827	2694	2669	16486	13943	13885
Splash	11615	10471	10437	42347	31338	31263
Tiffany	11789	10515	10481	50029	41870	41788
Lenna	12095	11295	11258	57237	49909	49832
Airplane	11457	10758	10725	67842	59573	59474
Aerial.9	12086	10663	10629	120044	112461	112347

- DC_enu and AC_enu: the enumerative code lengths for the DC and AC coefficients.
- DC_id and AC_id: the ideal code lengths for the DC and AC coefficients.

Table 5.4: The Performance of Three Lossless Compression Algorithms
(Not Including Extra Bits)

Image Name	After Step 2	Huffman (byte)	Compress (byte)	Arithmetic (byte)	Enu (byte)	Ideal (byte)
Sailboat	2444	1200	1491	1215	1127	1120
Sailboat	7926	3886	4376	3736	3624	3611
Chemical	7923	3852	4360	3675	3561	3549
Tree	7883	3812	4334	3689	3576	3565
Girl.1	4875	2247	2573	2098	1993	1984
House	4475	2276	2274	2007	1910	1900
Clock	4767	2414	2465	2177	2080	2069
Splash	13753	6745	5550	5350	5226	5213
Tiffany	16516	7727	7505	6677	6547	6534
Lenna	18425	8667	8841	7783	7651	7636
Airplane	20522	9912	9628	8923	8791	8775
Aerial.9	34112	16516	17653	15541	15390	15372

In Table 5.1, the DCT column is obtained by the number of blocks of each image times 388 bits. Based on Table 3.3 of the previous section, we calculate the maximum bits left per block after DCT computation is 338 bits. The column DCT+ Run is the number of bits for each compressed image, after scanning into zigzag pattern and applying run length coding algorithm. Note that the extra bits are included in step 2 and step 3.

Following this step the image is uniquely described by a sequence of bytes pointing to the Huffman codeword for the run/category, *plus* the extra bits. The gain from step 3 is obtained by using the Huffman code on the run/category event for the AC coefficients, and a DCPM encoding of the error for the DC coefficients. The same number of extra bits are included in the results of the Huffman and the arithmetic coding.

From Table 5.1, we note that the image statistics play an important role in the compression gain. For a low-detail image such as *splash, house, girl.1*, a higher compression performance is achieved. For a high-detail image such as aerial.9, which is a photograph of an urban area taken from an airplane, more bits required for the compressed file and performance is lowered as the cost of maintaining image quality.

Table 5.2: The DC and AC Extra Bits During the Coding Process

Image Name	Original file	DC Extra (bit)	AC Extra (bit)	Total Extra (bit)
Sail128	128×128	837	4331	5168
Sail256	256×256	3163	13019	16182
Chemical	256×256	2675	12147	14822
Tree	256×256	2881	13351	16232
Girl.1	256×256	2685	5694	8379
House	256×256	1898	5596	7494
Clock	256×256	2055	6447	8502
Splash	512×512	7863	12531	20394
Tiffany	512×512	9440	16212	25652
Lenna	512×512	11357	21754	33111
Airplane	512×512	8318	27523	35841
Aerial.9	512×512	11918	55444	67362

The categories, run lengths, and extra bits are treated as equally likely events in step 2. Therefore, step 3 is the step where the relative frequencies of these events are taken into account, and is subject to Huffman coding (baseline system) or to arithmetic coding (extended system).

5.2 Huffman and Count-Based Code Length Measures

The symbols generated from the DCT quantization and run length compression step are entropy coded for further compression using statistical techniques. The performance of the example Huffman code is evaluated by the ideal code length and enumerative code length measures.

In table 5.3, we use the following notations:

- DC_huff and AC_huff: the total bits required for coding the DC and AC coefficients using the Huffman coder proposed for the JPEG sequential baseline system.

5. Performance Results – Baseline

The routines for the simulation are written in the C programming language. The simulations were done on a Sun Sparcstation 1 running UNIX SunOS 4.1. Since the simulation is independent of any image library or utilities, it is easy to run it on other systems. The routines for the simulation have been checked by the decoding routines. A display routine, completely separate from the simulation routines, enables the original and the reconstructed images to be displayed under the other environments, for example, under X windows.

The test images are displayed on a 386 machine, with a 512×512 multisync monitor. Pictures were taken by a camera connected to a Multicolor Image Recording System (made by DUNN Instruments), connected to a Silicon Graphics IRIS workstation with display resolution 1280×1024.

5.1 Coefficient Quantization

To precisely evaluate the compression gain from DCT transform coding, Table 5.1 describes the compression from various steps using the example quantization Table 3.2. The data in Table 5.2 is used to calculate the compression gains displayed in the Table 5.1. The extra bits are listed in the Table 5.2 in order to compare the later results to the Huffman and the arithmetic coding. We can see from the Table 5.2 that the high-detail images have higher extra bits. The column DCT of the table is the lossy part (or step 1). The remaining number of bits per coefficient following quantization was calculated from the respective quantization value found in luminance Table 3.2, see the description of Table 3.3. The column denoted as step 2 shows the gain obtained for each image for the lossless compression of the run length coding of the AC coefficients, as the number of bits after quantization divided by the number of bits after converting the sequence for AC coefficients to a sequence of 8-bit addresses used to look up a Huffman codeword.

Table 5.1: The Performance of JPEG Baseline System

Image Name	Original File	Size (bit)	DCT (bit)	Step1 Gain	DCT+Run (bit)	Step2 Gain	Huffman (bit)	Step3 Gain
Sailboat	128×128	131072	99328	1.32	24720	4.02	14768	1.67
Sailboat	256×256	524288	397312	1.32	79590	4.99	47272	1.68
Chemical	256×256	524288	397312	1.32	78206	5.08	45636	1.71
Tree	256×256	524288	397312	1.32	79296	5.01	46724	1.70
Girl.1	256×256	524288	397312	1.32	47379	8.39	26352	1.80
House	256×256	524288	397312	1.32	43294	9.18	25698	1.68
Clock	256×256	524288	397312	1.32	46638	8.52	27815	1.68
Splash	512×512	2097152	1589248	1.32	129698	12.25	74356	1.74
Tiffany	512×512	2097152	1589248	1.32	157780	10.07	87470	1.80
Lenna	512×512	2097152	1589248	1.32	180511	8.80	102443	1.76
Airplane	512×512	2097152	1589248	1.32	200017	7.95	115140	1.74
Aerial.9	512×512	2097152	1589248	1.32	340258	4.67	199492	1.71

also listed in the file. This line also indicates the beginning of the block.

- AC coefficient information. The zero run number *nnnn*, category number *ssss* and the AC coefficient are listed.

The simulation output file provides the contexts for the Huffman coding, the arithmetic coding [WNC87], and the LZW algorithms.

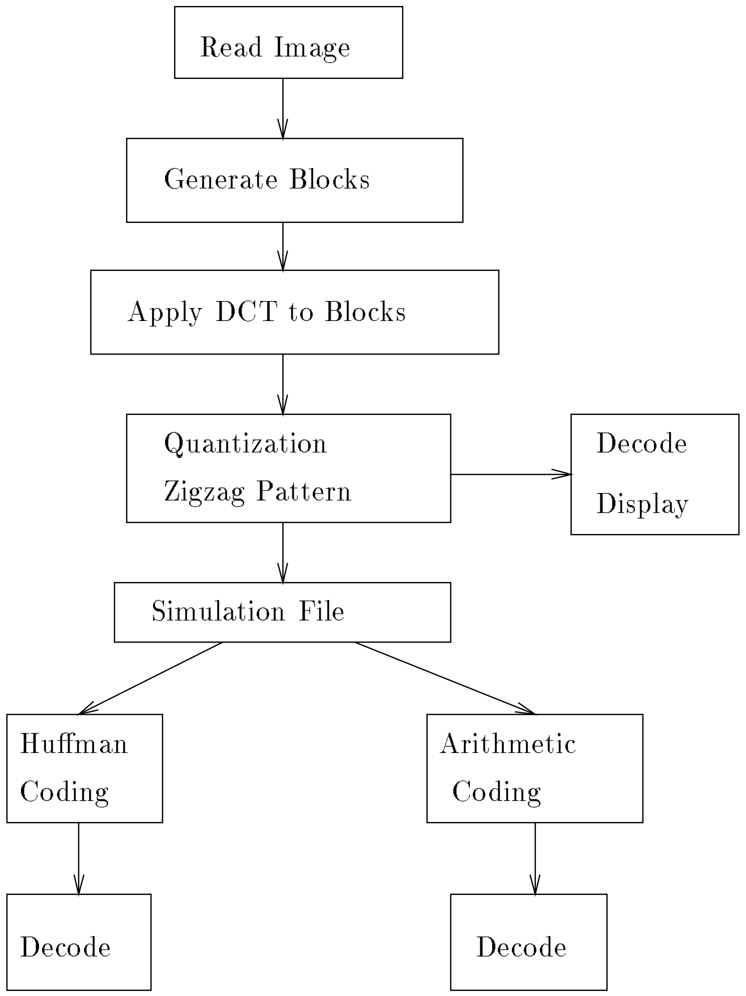

Figure 4.1: Simulation Flow Chart

$f(i,j)$ – shifted pixel value (from -128 to $+127$ for the JPEG baseline system)
$F(u,v)$ – DCT coefficient.

4. Quantize the transform coefficients and reorder them in a zigzag pattern.
   From the simulation, we can see that most blocks of the tested images consist of many zeros, and under most situations, the nonzero coefficients are often clustered around the upper left corner .

5. Create a simulation output file for each test image.
   The simulation output file consists of all the information of the quantized DCT coefficients of each block of the image. The following is the detailed description:

   - Image header. This consists of the width, height, length and depth of the original image.
   - DC coefficient information. The difference between the current block and the previous one is stored. The corresponding category number of the difference is

An important requirement is to ensure the simulation *accurately* represents the system being studied. The sequential baseline system simulation faithfully follows the documentation of JPEG draft version 5.2 [ISO90]. Even if the experimental frame is different, for example the test image set or the display device is different, the simulation *demonstrates* the performance of the JPEG algorithms. (An experimental frame is defined as a limited set of circumstances under which the system is to be observed or subjected to experimentation [Ore81]).

Figure 4.1 flow charts the main steps of the simulation. Counters are initialized to zero, and the following statistical information is collected.

- The number of bytes required for the compressed image after applying the DCT, quantization and run-length coding.
- The number of bytes required for the image after applying lossless Huffman coding algorithm to the result of step 1.
- The number of bytes required for the image after applying an arithmetic coding algorithm to the result of step 1.
- The count of the occurrence of each unique Huffman codeword during the coding process.
- From the above counts, one calculates three values for each compressed image: the ideal code length, the enumerative code length, and the *bits/pixel* rate for encoding the symbols that were generated.

The JPEG baseline simulation is done in the following steps:

1. Read in the image file.
   We assume that each image in our test set is a square image (image width is the same as the image height). All the images have 8 bits/pixel, which provides 256 grey-scale levels. (If it is not a square image, it is proposed in the JPEG draft that the last column (row) be duplicated in order to obtain a square image).
2. "Block" the image into 8×8 blocks.
   The first block is obtained by reading in the first 8 pixel elements of the first scan line, and the first 8 pixel elements of the second scan line, $\cdots$, the first 8 pixel elements of the 8th scan line, *etc.*
3. Apply the DCT algorithm to each 8×8 block.
   Since the DCT is computed for each block, a correct and fast algorithm is desirable. Because the block size is small, the improved version of the implementation of equation (4.4) is comparable to the DCT implementation using FFT (Fast Fourier Transform). The improvements include some common techniques such as precomputing the cosine functions, factoring out the fixed-variable in the inner loop, *etc.* All the computations are done in double precision floating point (64 bits).

$$F(u,v) = (\frac{1}{4})C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos((2i+1)u\frac{\pi}{16})\cos((2j+1)v\frac{\pi}{16}) \qquad (4.4)$$

   where

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{otherwise} \end{cases} \qquad C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } v = 0 \\ 1 & \text{otherwise} \end{cases}$$

4. The Simulation Technique

In this section, we present our analysis procedure for simulating the JPEG sequential DCT baseline system. The compression performance of the lossy and lossless algorithms are compared. We also present the compression performance of the Huffman coder, arithmetic coder [WNC87] and LZW algorithm based on our simulation of the DCT baseline system. In Appendix **A** the reader can find some of the reconstructed images that are generated from our simulation.

The SMI (SUN Microsystems, Inc.) raster file format is employed for this simulation. Since the information for a color image is conveyed mainly by the luminance component, we only worked with the gray-scale images. Most of these gray-scale images were converted from color images. Among the twelve images tested, three of the images were originally monochrome; *Chemical_plant, Aerial.9 and Lenna.*

The test images have variety, involving *Sailboat* (different size), *Aerial.9* (high detail) and *Splash* (low detail), *Tiffany* (low contrast). Most of the images, for example *Lenna, Sailboat and Girl.1*, appear in textbooks and the literature. The test set also includes images with different sizes. From the simulation, we can see that the compression performance varies according to the algorithm used. At the same time it also depends heavily on the statistical redundancy of a particular image.

A useful measure of the compression available in a file is the *ideal code length* (IL), defined in [RL81] for a model structure and individual finite data strings by using the context-dependent relative frequencies obtained from a posteriori occurrence counts. A one-pass measure, *enumerative code length* (EL), is similarly defined using a closed form formula [CW84]. Both measures are reviewed in [Lan90], and are defined for a single conditioning context z and 8 bits per symbol (256 values) as follows:

$$IL(z) = N(z)\log_2 N(z) - \sum_{i=0}^{255} n(i|z)\log_2 n(i|z) \tag{4.1}$$

$$EL(z) = [1 + \log_2 N(z)! - \sum_{i=0}^{255}(\log_2(n[i|z] - 1)!] \tag{4.2}$$

In equation (4.2), the notation z denotes a single context. Value $n(i|z)$ is the occurrence count of symbol i in context z. $N(z)$ is the total count of all symbol values in context z. The ideal code lengths (IL) and enumerative code length (EL) for the given image under the given model structure is the *sum* of the length contributions, respectively IL or EL, summed over all the contexts z. In equation (4.2), for large values of n we can use Stirling's formula to estimate $\log_2(n!)$:

$$\log_2 n! = (1/\log_e(2.0))[0.5\log_e(2\pi) + (n + 0.5)\log_e(n)] - n \tag{4.3}$$

In general, the term *compression ratio* is ambiguous, because of the many ways the term is defined. The method of reporting compression performance varies greatly from researcher to researcher. In [Wil89], Williams lists all the measures that are used in the literature, and suggests descriptive names. Defining α and β as the length of the uncompressed message and the length of the compressed message respectively, then value α/β defines the *compression gain.*

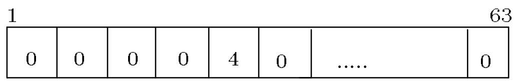

Figure 3.4: The AC Coefficient in the DCT Vector in the Example

The Huffman code table for *nnnn* and *ssss* concatenated together is in Table 3.7. Once *nnnnssss* is determined, this 8-bit value looks up the corresponding Huffman codeword. For example, if the number of zeros in the AC vector is 4 and the coefficient is in category A, then the codeword for 4/A (or 0100 1010 in binary) is 1111111110011101.

The convention for the transmission is the same as for the DC coefficients. If the coefficient is positive, then it is transmitted as it is; otherwise $C - 1$ is transmitted, where C is the AC coefficient. The decoding rule is the same as for the DC coefficient.

There are two special *nnnnssss* codewords, namely ZRL (Zero Run Length) and EOB (end-of-block). When 15 consecutive zeros are found in the ZZ vector and are followed by a zero coefficient, F0 (ZRL) is used. In other words, there are 16 consecutive zeroes in the ZZ vector. The 00 (EOB) represents the last nonzero coefficient. If the 63^{rd} coefficient is nonzero, then EOB is bypassed.

Example: suppose part of the DCT vector is defined as in Figure 3.4. Observe that run *nnnn* is 4, and that according to Table 3.6, and 4 lies in category ssss = 3, so in binary *nnnnssss* = 01000011. This value may be used as an address to look up the codeword.

By looking in Table 3.7 for 4/3, we get codeword 11111111 10010110. So the bits for transmission are: $\underbrace{1111111110010110}_{HuffmanCode}\ \underbrace{100}_{extra\ bits}$.

baseline system, the category number will not exceed 10 for the sequential baseline system.

Table 3.6: Values Assigned to AC Coefficient Range

ssss	AC coefficients
1	$-1, 1$
2	$-3, -2, 2, 3$
3	$-7, \ldots, -4, 4, \ldots, 7$
4	$-15, \ldots, -8, 8, \ldots, 15$
5	$-31, \ldots, -16, 16, \ldots, 31$
6	$-63, \ldots, -32, 32, \ldots, 63$
⋮	⋮
E	$-16383, \ldots, -8192, 8192, \ldots, 16383$
F	not used

Table 3.7: Huffman Code Table for Luminance AC Coefficients

Run/Size(*nnnn/ssss*)	CodeLength	Huffman Codeword
0/0	4	1010
0/1	2	00
0/2 ∼ 0/9 omitted	⋮	⋮
⋮	⋮	⋮
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3 ∼ 1/9 omitted	⋮	⋮
⋮	⋮	⋮
1/A	16	1111111110001000
2/1 ∼ 3/A omitted	⋮	⋮
⋮	⋮	⋮
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4 ∼ 4/9 omitted	⋮	⋮
⋮	⋮	⋮
4/A	16	1111111110011101
5/1 ∼ E/A omitted	⋮	⋮
⋮	⋮	⋮
F/0 (ZRL)	11	11111111001
⋮	⋮	⋮
F/A	16	1111111111111110

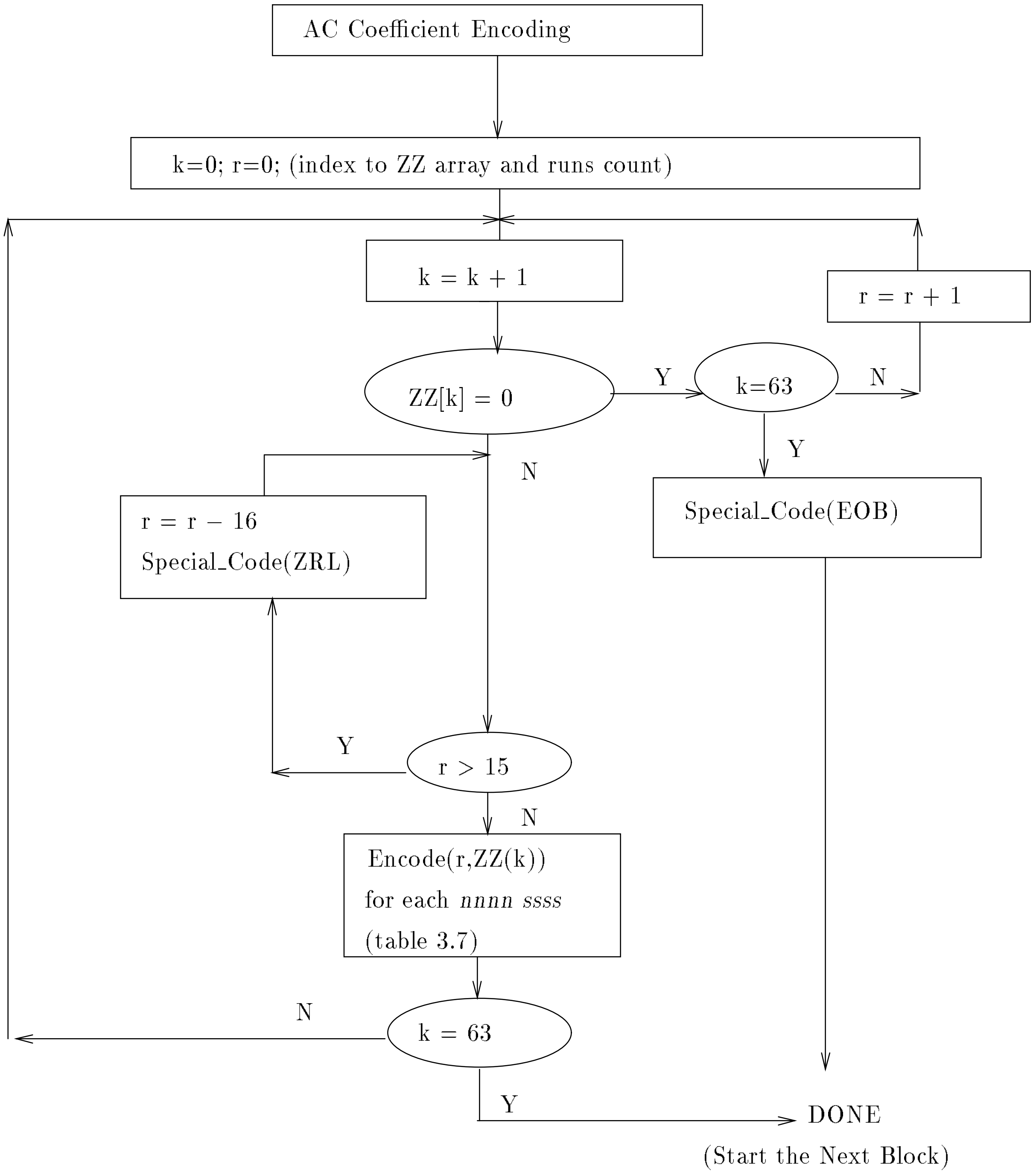

Figure 3.3: AC Coefficients Encoding Algorithm

Given the AC coefficients ZZ(1, . . ., 63), run length coding is done by creating a single byte *nnnnssss*, where *nnnn* is the run length of zeros before hitting the next nonzero AC coefficient and *ssss* is the category number that represents the size of the nonzero AC coefficient (see Table 3.6). The precision of the quantized DCT coefficients can be up to 15 bits, but with the input of only 8 bits and the quantized DCT coefficient 11 bits for the

4-bit value *ssss* (see Table 3.4). It is also the size of the appended bits. As in [PBW79] and [TLR85], the differences are losslessly encoded by a two-step process, where the first step is the error bucket, or category in JPEG terminology. The second step encodes the exact error value within the category.

Using *ssss* as an index, a Huffman codeword can be found in the code table (see Table 3.5). This example Huffman code table used for the baseline system was developed, according to JPEG, from "the average statistics of a large set of video images with 8-bit precision [PM89]". Tables 3.4 and 3.5 employ a Huffman coding technique similar to that described in [PBW79].

Table 3.5: Example Table for Luminance DC Coefficients

ssss	HuffCodeLength	Huffman Codeword
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

The length of the resulting data that are ready for the transmission is *HuffCodeLength* + *ssss*, where *ssss* is the number of bits required by the *diff* value. It turns out the value of index *ssss* is also the number of appended extra bits. The Huffman code followed by the required number of extra bits is transmitted. By transmission convention, if *diff(k)* is positive, then the low order coefficient is transmitted; otherwise *diff(k)*-1 is transmitted. The most significant bit of the extra-bit sequence is 0 for negative coefficients and 1 for positive coefficients in this convention. By this convention the sign of the prediction error is incorporated in the extra bits. The decoder adds 1 to the extra bits if the sign is negative.

Example: suppose the difference is minus eleven (-11): 11111110 101. Note that the maximum quantized DCT coefficient is up to 11 bits long. The category number *ssss*, the position of the most significant bit, is 4 according to Table 3.4. Using 4 as the index, Huffman codeword 101 is the corresponding entry in Table 3.5. Since 11111110 101 is negative, the sign bit is 0 and the extra bits are 100 which by the transmission convention is one less than 101. Therefore, $\underbrace{101}_{HuffmanCode} \quad \underbrace{0100}_{-11}$ is transmitted.

3.5 Coding Model for AC Coefficients

The quantized AC coefficients usually contain runs of consecutive zeros, which are encoded with the run-length coding technique. The flow chart for the run-length coding algorithm appears in Figure 3.3.

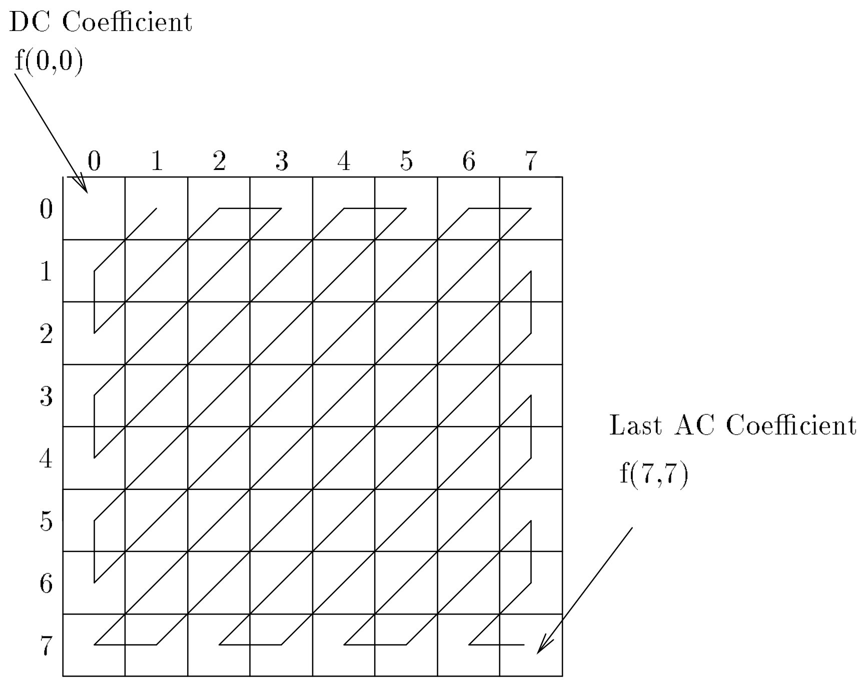

Figure 3.2: An Example of Zigzag Pattern for Reordering DCT Coefficients

diff(k) = DC_*CurrentBlock(k)* − DC_*PreviousBlock(k-1)*

Table 3.4: Difference Categories *ssss* for DC Coding of Difference *diff*

ssss	Difference Value (*diff*)
0	0
1	−1, 1
2	−3, −2, 2, 3
3	−7, ..., −4, 4 , ..., 7
4	−15, ..., −8, 8, ..., 15
5	−31, ..., −16, 16, ..., 31
⋮	⋮
15	−32767, ..., −16384, 16384, ..., 32767

In the decoding stage, an inverse DPCM is applied to the k^{th} block to retrieve the original DC coefficient. It is defined as

DC_*CurrentBlock(k)* = *diff(k)* + DC_*PreviousBlock(k-1)*

3.4 Huffman Coding Model for DC Coefficients

The symbols (difference values) *diff(k)'s* are further encoded using the Huffman coder. All possible differences are segmented into 16 categories and the category is denoted by

Table 3.2: Luminance Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 3.3: Quantization Bits Assigned to Each DCT Coefficient

7	8	8	7	7	6	6	6
8	8	8	7	7	6	6	6
8	8	7	7	6	6	7	6
8	7	7	7	6	5	5	6
7	7	6	6	5	5	5	5
7	6	6	5	5	5	5	5
6	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5

Note that Table 3.3 contains eight 8's, fourteen 7's, sixteen 6's and twenty-six 5's, resulting in 388 bits/block. Therefore, following the lossy coding step in the baseline system using example Table 3.2, the 64$\times$11 bits has been reduced to 388 bits per block.

3.3 Coding Model for DC coefficients

The quantized coefficients are packed into a one-dimensional array in a zigzag (ZZ) pattern as shown in Figure 3.2. The purpose of ordering the transform coefficients with the zigzag pattern is to place the lower frequencies before the higher frequencies. It has been confirmed that this simple scheme does result in near-optimal performance since the zigzag pattern almost matches the order of coefficient variance value [Nga84]. The zigzag pattern usually generates more consecutive zeros for quantized DCT coefficients, and permits the efficient use of the run-length coding.

The baseline system uses only the Huffman coding model for the DC and AC coefficients. Two Huffman coding tables (luminance and chrominance) are provided in [ISO90] as examples for coding the DC coefficients. Another two Huffman tables can be used for coding the AC coefficients. These tables appear in each version of JPEG drafts. Although there are no default Huffman tables, these example tables proved to be useful for many applications.

Because the difference between the DC coefficients of two neighboring blocks is usually small, DPCM (Differential Pulse Code Modulation) exploits this property. The coding model for the DC coefficients is the one-dimensional lossless DPCM. For each 8$\times$8 block, the coding model for block k is defined as

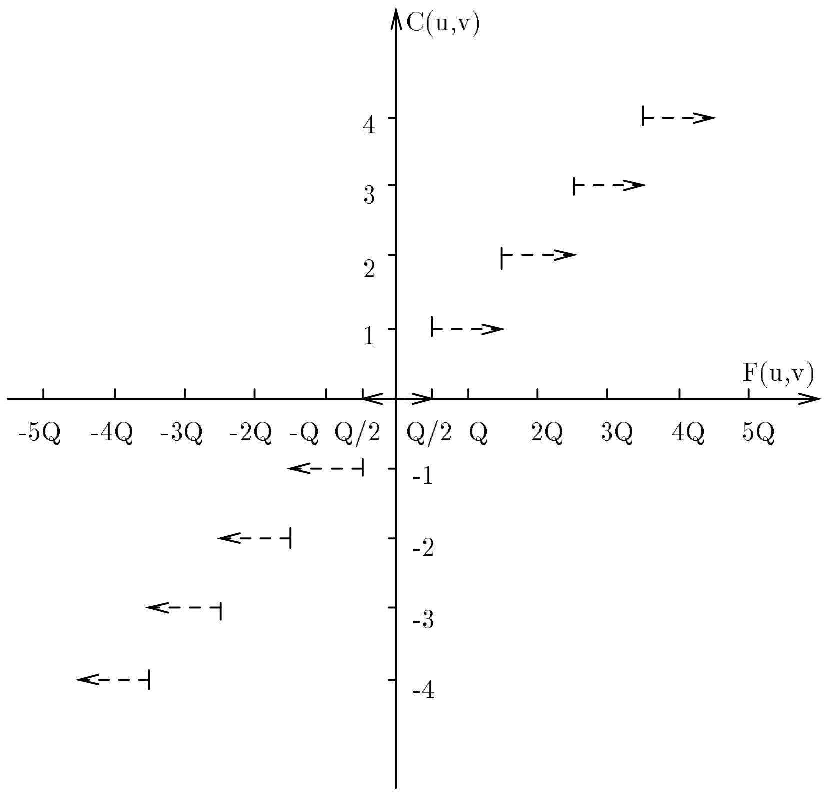

Figure 3.1: The Quantization Rules

Table 3.1: Chrominance Quantization Table

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

coefficient. The precision is rounded up to the next integer value. The sign is positive, if the DCT coefficient $F(u,v) \geq 0$; otherwise it is negative. Equation (3.4) can be rewritten as

$$C(u,v) = [F(u,v)/Q(u,v) \pm 1/2] \tag{3.5}$$

From equation (3.5), it is clear that the quantized DCT coefficient is rounded up to the next closest integer.

The quantization step loses information by representing the coefficients with fewer bits. Suppose the DC component is 11 bits long, following quantization the coefficient has 7 bits after dividing by 16. Table 3.3 shows the maximum bits in the respective coefficients by applying the DCT and then the quantization step using Table 3.2.

Furthermore, the DCT coefficients usually have the following properties:

- The maximum is usually the DC coefficient. The remainder of the coefficients become smaller and smaller.
- They vary slightly. This is because the image basis varies slowly.
- After the FDCT transformation, in most cases, the majority of the AC coefficients are small.

3.2 Quantization

In the baseline system, the input and output data are limited to an input precision P of 8 bits/pixel and the quantized DCT coefficients need at most 11 bits ($-128 \times 8 \sim 127 \times 8$). This is because the input pixel range is -128 to $+127$ after the level-shift, and FDCT increases the range of $F(u,v)$ by a maximum factor of 8. The equation (3.1) is defined in such a way that the coefficients before the quantization have a precision of $P+3$, or 11 for 8-bit pixel values. This expands the data by 3 bits per pixel. After the quantization the quantized DCT coefficients have a precision of

$$P + 3 - \log_2(Q(u,v)) \tag{3.3}$$

where $Q(u,v)$ is the value in the (u,v) position of the quantization matrix.

The quantization of each of the 64 coefficients is determined by a quantization table that contains 64 independent values $Q(u,v)$, which is the step size varying according to coefficient location (u,v) and the color system used. JPEG [ISO90] provides a chrominance (see Table 3.2) and a luminance (see Table 3.1) quantization table, with the claim that these tables provide good results for 8 bits/pixel for gray-level and Y, C_b, C_r color image formats. The quantization tables use finer quantization for the lower frequency coefficients, and use coarser quantizers for the higher frequency coefficients. Consequently, most of the higher frequency coefficients of each block become zeros after quantization. These tables are provided in the JPEG document as *example* quantization tables. Users can design their own quantization tables according to their applications.

In the case that an image contains text, the compressed text using the example tables might be unreadable; whereas the non-text part has excellent quality. JPEG++ [Lig91] provides a scheme to vary the quantization rate *within* the image. This scheme can also be used when one part of the image is more important than the other part (for example, the background of the image is normally considered as insignificant).

All the images used for the simulation in this report are gray-scale and thus only the luminance table is used. Experiments confirm that the luminance component of a color image conveys the majority of the information.

A uniform midstep quantizer (see Figure 3.1) is used in the JPEG baseline system. The quantization rules are

$$C(u,v) = [F(u,v) \pm (Q(u,v)/2)]/Q(u,v) \tag{3.4}$$

In equation (3.4), $F(u,v)$ is the DCT transform coefficient. $Q(u,v)$ is the quantization value obtained from quantization Table 3.1 or 3.2. Each $Q(u,v)$ ($1 \sim 255$) is the quantizer step-size for its respective quantized coefficient $C(u,v)$. $C(u,v)$ is the quantized DCT

3. The Sequential DCT Baseline System

3.1 FDCT

The input and output data for the baseline system are 8-bit unsigned integers representing pixel intensity values. The input data is level shifted to a signed representation by subtracting 2^{P-1} from each pixel value, where P is the precision of the input data. For the sequential baseline system P is 8, and the range of the input data is 0 to 255. The new zero after shifting is at 128. The level-shift makes the DC coefficient of each block smaller, and it is consistent with an example quantization table which appears in JPEG draft documentation [ISO90].
The normalized forward DCT (FDCT) and the inverse DCT (IDCT) defined in the sequential baseline system are

$$F(u,v) = (\frac{1}{4})C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos((2i+1)u\frac{\pi}{16})\cos((2j+1)v\frac{\pi}{16}) \qquad (3.1)$$

$$f(i,j) = (\frac{1}{4})\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos((2i+1)u\frac{\pi}{16})\cos((2j+1)v\frac{\pi}{16}) \qquad (3.2)$$

where

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{otherwise} \end{cases} \qquad C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } v = 0 \\ 1 & \text{otherwise} \end{cases}$$

$f(i,j)$ – shifted pixel value (from -128 to $+127$ for the baseline system)
$F(u,v)$ – DCT coefficient.

The equations (3.1) and (3.2) define the infinite-precision DCT. The specification for DCT accuracy and implementation has been discussed within JPEG [Cha90]. The FDCT is applied to each 8×8 block and results in 64 DCT coefficients. If the 8×8 block input contains N-bit integers, then the integer part of the DCT coefficients can grow by at most 3 bits. This is also the largest possible size of a quantized DCT coefficient. Since the round-off and truncation effects depend on the way the calculations are performed, the different IDCT (Inverse Discrete Cosine Transform) implementations may produce slightly different output images.

The *DC coefficient* is the (0,0) coefficient representing the vertical and horizontal spatial frequencies of zero. The DC coefficient is the average value of the 64 pixels times 8. The other coefficients have at least one nonzero spatial frequency and are called *AC coefficients.* An appealing fact is that many of the remaining 63 AC coefficients, under most cases, are close to zero. The resulting coefficients of the two-dimensional DCT coefficients are ordered in the following way:

- The DC coefficient is placed in the upper left corner of an 8×8 matrix. *i.e.* location (0,0).
- The higher vertical frequencies correspond to higher row numbers.
- The higher horizontal frequencies correspond to higher column numbers.

In the sequential baseline system, only two Huffman tables (one for the DC coefficient, and the second for the AC coefficients) are needed for each color component. In addition to providing an example Huffman code table for its sequential baseline system, the JPEG draft [ISO90] also includes (section 13) the implementation details on how to generate a Huffman code table. This allows users to customize their Huffman code table to their application.

The Huffman codewords are generated in such a way that every codeword consists of two or more bits and the longest codeword is never all 1's. The maximum length of the Huffman codeword is 16 bits.

2.4 The Extended Sequential System

In order to meet the needs for higher precision coding systems, JPEG defines several extensions as optional enhancements to the baseline system. A one-pass adaptive binary arithmetic coder, familiarly known as the QM-coder, is proposed as an alternative to the Huffman coder. The QM-coder is based on the Q-coder [PMLA88]. (Note that the arithmetic coding simulation done for this paper is based on [WNC87]).

The input and output data precision can be extended to 12 bits per pixel per color component. The quantized DCT coefficients are limited to 15 bits. The Huffman coding tables may be inherited from the previous image, or be specially optimized for each individual image. The specification of the Huffman code table is included in the signaling parameters.

The conventional line-by-line transmission technique does not permit an early recognition of the picture content. An approach to solve this problem is to use *progressive transmission* techniques that transmit a crude representation of the image first and then transmit a better and better approximation. This is particularly useful for communication using low-bit-rate channels, browsing through remote stored pictures in the database, or interactive graphics from a remote computer.

The hierarchical progressive mode is similar to the pyramid-structure progressive transmission [ST79]. The pyramids of an image are successively reduced in size through certain decimation processing such as pixel average, filtering or simply discarding the pixels.

The independent lossless extension is useful, for example, in the space satellite and medical imaging applications. The lossless algorithm is JPEG extension using a variation of the DPCM coding model for the DC coefficients of the DCT, and resembles a lossless compression algorithm called *Sunset* [Lan].

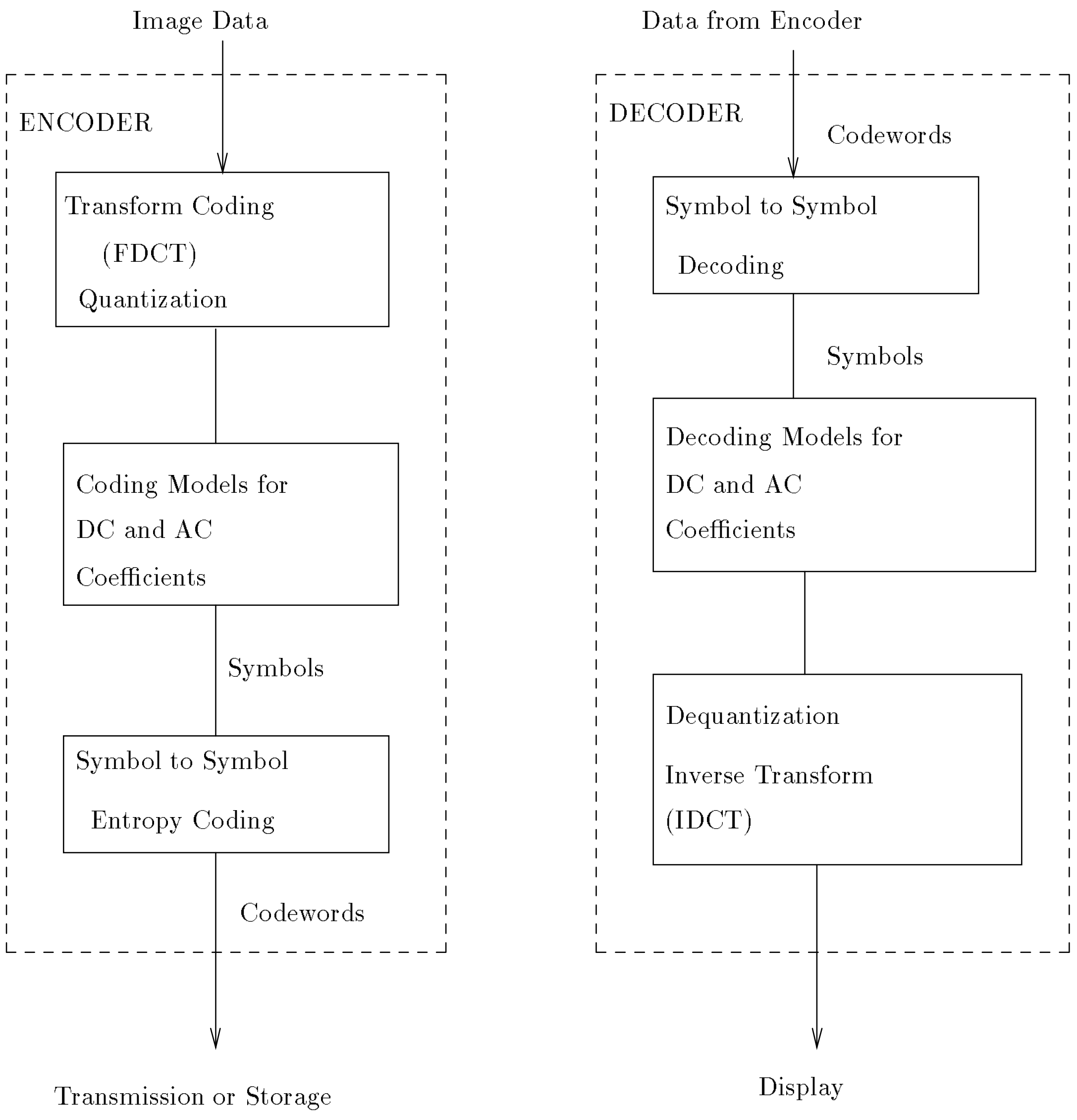

Figure 2.1: JPEG Encoder and Decoder for the Baseline

The baseline system is the minimum portion of the standard that all products claiming conformance must implement.

2.3 The Sequential Baseline System

The sequential baseline system is based on the DCT algorithm on 8$\times$8 blocks, and appears to guarantee good images with compression ratios of at least 10:1. Figure 2.1 shows the minimal system required for all JPEG implementations.

The transform coding stage consists of the Forward DCT (FDCT) transformation that actually expands the amount of data, but is followed by a lossy scalar quantization process. The coding model converts the quantized DCT coefficients into a set of values, which are then further encoded using a Huffman coder. The input and output data precision is 8 bits per pixel. The DCT coefficients each has 11 bits prior to using the quantization matrix.

2. JPEG

2.1 Historical Background

Inspired by the success of the current bi-level fax standardization, which was set by the CCITT (Consultative Committee on International Telegraph and Telephone), members of a CCITT Study Group VIII Special Rapporteur Group on Common Components for Image Communication and an ISO (International Standards Organization) working group on coded representation of picture and audio information (ISO/IEC JTCI/SC2/WG8) have been working together as a subcommittee for several years. The mission of this Joint Photographic Experts Group (JPEG) is to standardize a set of algorithms for general-purpose, continuous-tone (gray-scale or color), still-image compression [PM89]. This dual reporting structure ensures that ISO and CCITT produce a joint image compression standard. The design of the algorithms aims at (1) compression performance that produces excellent image quality at around 1 bit/pixel, and (2) compression speed that keeps pace with (at least) 64 Kbps channels. For example, in a photovideotex system, a 720×576 pixel image at 1 bit/pixel after compression can be sent over 64 Kbps line in 6.5 seconds, which is tolerable for occasional image viewing [Cha90]. The JPEG standard may be employed in various widely-used applications such as color printers, gray-scale and color scanners, facsimile, photo-videotex, and still-frame teleconferencing. The JPEG technical experts expect to resolve the remaining issues soon and complete the technical documentation. A good overview can be found in [Wal91].

2.2 Overview

The JPEG data compression algorithms has four basic functional parts:

1. The sequential baseline system.
2. The extended sequential system for better compression, higher precision.
3. The progressive modes.
4. An independent lossless coding system.

1. Introduction

This report (also see [(Zh90] and [CL91]) concerns the compression of images in digital form. Image data compression is the process of transforming the digital image data to a representation employing a *smaller* amount of data from which the original (*lossless* case) or some approximation to the original (*lossy* case) can be recovered. Image data compression applies to data communication and storage. The transmission of a picture consisting of 512×512 picture elements (pixels) at 8 bits/pixel over a 1200-baud line can take 30 minutes! Reducing the volume of data by a factor of two or three significantly improves transmission time. Image data compression is an important component of video systems.

The dramatic increase of image processing applications has made image compression standardization desirable. An international standardization subcommittee, familiarly known as the Joint Photographic Experts Group (JPEG), has been defining a set of algorithms and formats for image compression since 1986. The JPEG subcommittee was presented with proposals for many algorithms. The working group balanced the system parameters such as the cost and speed, the quality of the image and compression performance, the transmission scheme, *etc.* The forthcoming standard (familiarly known as the JPEG standard), can be used with devices such as printers and scanners with applications to videotex, interactive video [Hua90], *etc.* This standardized image algorithm is already implemented in firmware for many computers such as the Macintosh and NeXT workstations.

We investigate both the lossy and the statistical coding aspects for the baseline JPEG image compression approach. The main contribution of this report is to demonstrate a method and then analyze the independent components of the JPEG algorithm and its performance. We simulate the JPEG sequential DCT (Discrete Cosine Transform) baseline system, and show the JPEG compression performance at each step. The beauty of the JPEG standard, we think, is the room allowed within the standard, for the users to customize the algorithm steps to their own applications.

This report is organized as follows:

- Section 2 provides an overview of JPEG.
- Section 3 describes the JPEG baseline system.
- Section 4 is the simulation approach.
- Section 5 presents compression results for the JPEG baseline system. It also includes performance of the Huffman coding, the UNIX[1] *compress* algorithm (based on LZW (Lempel-Ziv-Welch) [Wel84]), and an adaptive arithmetic coding algorithm [WNC87].
- Section 6 summarizes our conclusions and observations. We also suggest some future work.

[1]UNIX is a trademark of AT&T

Effects of Coefficient Coding on JPEG Baseline Image Compression

Manyun Chang
Glen G. Langdon, Jr., Jim Murphy

UCSC-CRL-91-07
April 1991

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA