Journal of Digital Imaging

VOL 7, NO 1 FEBRUARY 1994

IMAGES ON PERSONAL COMPUTERS

Displaying Radiologic Images on Personal Computers: Image Storage and Compression—Part 2

Thurman Gillespy III and Alan H. Rowberg

This is part 2 of our article on image storage and compression, the third article of our series for radiologists and imaging scientists on displaying, manipulating, and analyzing radiologic images on personal computers. Image compression is classified as lossless (nondestructive) or lossy (destructive). Common lossless compression algorithms include variable-length bit codes (Huffman codes and variants), dictionary-based compression (Lempel-Ziv variants), and arithmetic coding. Huffman codes and the Lempel-Ziv-Welch (LZW) algorithm are commonly used for image compression. All of these compression methods are enhanced if the image has been transformed into a differential image based on a differential pulse-code modulation (DPCM) algorithm. The LZW compression after the DPCM image transformation performed the best on our example images, and performed almost as well as the best of the three commercial compression programs tested. Lossy compression techniques are capable of much higher data compression, but reduced image quality and compression artifacts may be noticeable. Lossy compression is comprised of three steps: transformation, quantization, and coding. Two commonly used transformation methods are the discrete cosine transformation and discrete wavelet transformation. In both methods, most of the image information is contained in a relatively few of the transformation coefficients. The quantization step reduces many of the lower order coefficients to 0, which greatly improves the efficiency of the coding (compression) step. In fractal-based image compression, image patterns are stored as equations that can be reconstructed at different levels of resolution.

KEY WORDS: personal computer, image compression, differential pulse code modulation, Huffman codes, Lempel-Ziv-Welch compression, discrete cosine transformation, discrete wavelet transformation, fractal compression, tagged interchange file format.

THIS IS PART 2 of our review article on image storage and compression. Part 1 reviewed image storage and the fundamental principles of information theory and image compression. In this part, we review different methods of lossless and lossy image compression. This is the third article of our series for radiologists and imaging scientists on displaying, manipulating, and analyzing radiologic images on personal computers.

LOSSLESS IMAGE COMPRESSION

Run-Length Encoding

The run-length encoding (RLE) compression algorithm replaces a sequence of pixels with the same value with an RLE code and the number of occurrences. For example, 20 consecutive zero-value pixels can be encoded by the sequence: ⟨RLE code⟩ ⟨20⟩. RLE is commonly used for bilevel or bit-mapped images in which long stretches of white and black pixels occur in series. This algorithm is not very useful for continuous-tone gray scale or color images, but is sometimes used in conjunction with other compression techniques. In addition, RLE may be useful for compressing the edges of radiologic images that are very dark (pixel values = 0) and do not contain any anatomically useful information.

From the Department of Radiology, University of Washington, Seattle, WA.

Address reprint requests to Thurman Gillespy III, MD, Department of Radiology, SB-05, University of Washington, Seattle, WA 98185.

0897-1889/94/0701-0010$3.00/0

Image-Specific Compression Techniques

Knowledge of the specific characteristics of certain image types can be useful in designing unconventional compression techniques. For example, computed tomographic (CT) scans from some vendors are reconstructed using a roughly circular image matrix (Fig 1). A circular matrix occupies approximately 80% of the nominal 512 × 512 pixel CT image matrix. When stored as an uncompressed raster data file, the nonimage pixels outside of the circular matrix are set to zero (black) or some other value. These nonimage pixels can be greatly compressed using the RLE method (see above), or by a simple two-column packing/unpacking table that distinguishes the image and nonimage portions of each row (Table 1).

Huffman Coding

The Huffman algorithm uses the probabilities of each pixel's occurrence in an image to construct a table of codes that has the following properties: (1) the codes have varying length; (2) the codes can be uniquely decoded because they have a unique prefix; and (3) the codes for pixels with higher probabilities have fewer bits than codes for pixels with lower probabilities.[1-3]

The algorithm is easiest to show using a text example, but the principle is the same for radiologic images. Take the following message as an example:

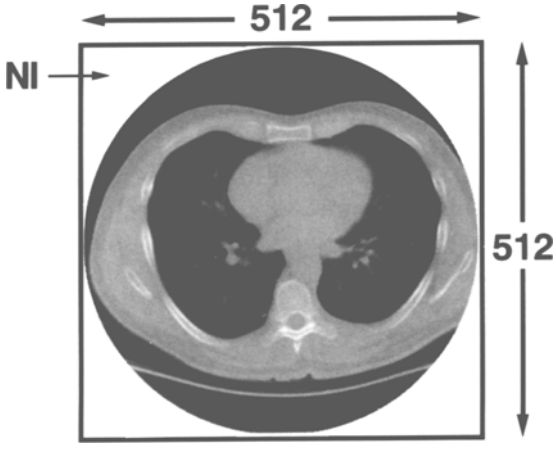

Fig 1. CT circular image matrix. The actual image data occupies a roughly circular region inside the nominal 512 × 512 pixel CT image matrix. The image data occupies approximately 79% (π/4) of the square image matrix. Abbreviation: NI, nonimage.

Table 1. Packing/Unpacking Table for Compression of Nonimage Regions of CT Image Files

Row	Column 1 (nonimage pixels)	Column 2 (image pixels)	Uncompressed*	Compressed†
1	233	46	932	4
2	224	64	896	4
3	217	78	868	4
4	211	90	844	4
5	206	100	824	4
6	201	110	804	4
7	197	118	788	4
8	193	126	772	4
9	189	134	756	4
10	185	142	740	4
256	0	512	0	4
503	185	142	740	4
504	189	134	756	4
505	193	126	772	4
506	197	118	788	4
507	201	110	804	4
508	206	100	824	4
509	211	90	844	4
510	217	78	868	4
511	224	64	896	4
512	233	46	932	4
Total			55,772	2,048

A simple 512-pixel row two-column table (column 1, column 2) defines the image and nonimage portions of the CT image (Fig 1). For each row, column 1 specifies the number of nonimage pixels from the "left edge" of the 512 × 512-pixel matrix to the beginning of the circular image matrix. Column 2 specifies the number of image pixels for that row. The number of nonimage pixels to the right of the image pixels is equal to [512 − (column 1 + column 2)].

*Number of bytes to store the nonimage pixels.

†Number of bytes to store the packing/unpacking table.

NOW IS THE TIME FOR ALL GOOD COMPUTER USERS TO COME TO THE AID OF THEIR OPERATING SYSTEM

First, we construct a sorted table of the symbols and their frequencies (Table 2). Then we can calculate the probabilities of each symbol, and the information content per symbol (see equation 1 in part 1 of this article [Journal of Digital Imaging, November 1993, p 202]). This yields a total of 340.5 bits of information for this message, whereas the message is normally stored using 704 bits (88 characters × 8 bits per character). The difference between these two numbers is an estimate of the maximum possible compressibility of this message using an order-0 statistical model (51.6% in this example).

Table 2. Entropy (Information Content) of the Text Example

Symbol	Frequency	Storage Bits*	Probability†	Entropy‡	Information Content§
' '	17	136	.1932	2.37	40.29
'O'	10	80	.1136	3.16	31.60
'E'	9	72	.1023	3.29	29.61
'T'	9	72	.1023	3.29	29.61
'I'	5	40	.0568	4.14	20.70
'R'	5	40	.0568	4.14	20.70
'S'	5	40	.0568	4.14	20.70
'M'	4	32	.0455	4.46	17.84
'A'	3	24	.0341	4.87	14.61
'H'	3	24	.0341	4.87	14.61
'C'	2	16	.0227	5.46	10.92
'D'	2	16	.0227	5.46	10.92
'F'	2	16	.0227	5.46	10.92
'G'	2	16	.0227	5.46	10.92
'L'	2	16	.0227	5.46	10.92
'N'	2	16	.0227	5.46	10.92
'P'	2	16	.0227	5.46	10.92
'U'	2	16	.0227	5.46	10.92
'W'	1	8	.0114	6.45	6.45
'Y'	1	8	.0114	6.45	6.45
Total	88	704	1.0000		340.53

The probabilities for all the tables in this article are calculated using an order-0 statistical model.

*Bits required to store each symbol in the message as ASCII text (8 bits × frequency of symbol).

†Probability = symbol frequency/total symbols.

‡Entropy: see equation 1. Note that the entropy is lowest for the symbols with the highest probability.

§Information content = (entropy per symbol) × (symbol frequency).

The Huffman codes are calculated in the following manner. A binary tree (Fig 2) is constructed, beginning from the bottom and progressing to the top (root) of the tree. The tree begins with an ordered list of all the symbols and their frequencies (Fig 3A). Each symbol is assigned to a free node. The two free nodes with the lowest frequencies are combined

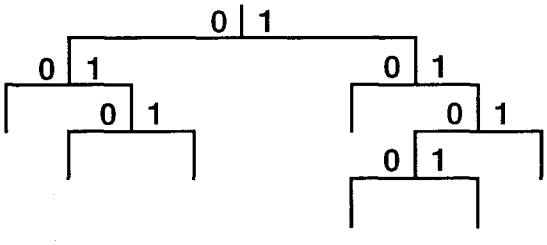

Fig 2. Binary tree. Binary trees are commonly used data structures. The top of the tree is known as the root. Below the root are two nodes. Each node has exactly two branches consisting of either another node or a free leaf. By assigning a binary 0 and 1 to each node and leaf, a unique code can be assigned to each leaf of the tree.

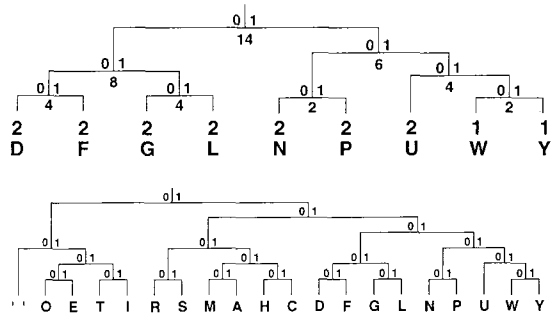

Fig 3. Huffman tree. (Top) Constructing the Huffman tree. The free nodes with lowest weights are combined into a parent node. Only the right half of the tree is shown. (Bottom) The completed Huffman tree. The symbols with the highest frequency have the shortest codes (see Table 3).

into a parent node, and then removed from the list of free nodes. The combined weight of the two old nodes are assigned to the new node. Then a binary 0 and 1 are assigned to the two leaves, which become the least significant bits in the Huffman code for these symbols. The process is continued until all the nodes are assigned (Fig 3B). The Huffman codes for each symbol are calculated by traversing the tree from the root to the leaf containing that symbol, accumulating the binary 0's and 1's. The codes with the fewest bits are assigned to the symbols with the highest frequency (space, O, E, and T in this example) (Table 3). This message can be encoded with 348 bits (50.6% compression), which is only slightly worse than the 340.5 bits (51.6% compression) predicted from Table 3 (ignoring the overhead of storing the code table).

There are several disadvantages to this form of compression. First, the coding table must be stored with the message because the frequencies of symbols will differ with different messages. The coding table is fairly small using an order-0 model as we have done in this example, but it becomes unacceptably large if higher-order models are used. Furthermore, the Huffman codes have an integral number of bits, whereas the entropy equation usually calculates a fractional number of bits for a given symbol. Therefore, the Huffman codes are usually not optimal for a given symbol. This inefficiency appears in the slightly less-than-optimal compression ratio. However, Huffman codes have been shown to be the most efficient fixed-length coding method.

Huffman codes are commonly used in conjunc-

Table 3. Huffman Coding of the Text Example

Symbol	Frequency	Huffman Code	Huffman Bits	Total Bits*
' '	17	00	2	34
'O'	10	0100	4	40
'E'	9	0101	4	36
'T'	9	0110	4	36
'I'	5	0111	4	20
'R'	5	1000	4	20
'S'	5	1001	4	20
'M'	4	10100	5	20
'A'	3	10101	5	15
'H'	3	10110	5	15
'C'	2	10111	5	10
'D'	2	11000	5	10
'F'	2	11001	5	10
'G'	2	11010	5	10
'L'	2	11011	5	10
'N'	2	11100	5	10
'P'	2	11101	5	10
'U'	2	11110	5	10
'W'	1	111110	6	6
'Y'	1	111111	6	6
Total	88			348

*Total Huffman bits = (Huffman bits per symbol) × (frequency of symbol).

tion with differential pulse code modulation (DPCM) images because the altered-image statistics improves the Huffman coding. Although maximum compression occurs if a unique set of Huffman codes are used for each image, a combined set of codes for CT and magnetic resonance imaging (MRI) can offer very acceptable performance.[4] An example of a Huffman code book for differential images is shown in Table 4.[5]

Adaptive Huffman Coding

One enhancement that works with Huffman and other forms of coding is adaptive coding.[2] Instead of using a fixed table of probabilities, adaptive coding builds and alters the Huffman tree as each symbol is encountered. This scheme eliminates the need to store probability tables with the compressed file. In addition, adaptive coding also handles local changes in symbol probabilities better than fixed probability tables that are built for an entire file. Adaptive coding is commonly used with many other forms of coding.

Simplified Huffman Codes

This algorithm can be considered a simplified variant of Huffman encoding. A difference image is encoded with codes that are 1 to 3 bytes long (Table 5). A unique prefix distinguishes the different codes. This algorithm is easily implemented because the codes occur on byte boundaries, and the prefixes can be quickly decoded. Although not as effective as some of the other compression algorithms considered so far, this technique permits very rapid compression and decompression of images.

Arithmetic Coding

Unlike the Huffman coding that replaces a series of pixels with a series of codes, arithmetic coding generates a single floating point number. This number is less than 1 and is greater than 0, and can be uniquely decoded to regenerate the original file.[6,7]

Using the same text example listed above, the arithmetic coding process begins with a probability table of the input symbols (Table 6). A "generic" probability table for certain image types can also be used. Each symbol (pixel) is

Table 4. Huffman Code Book for DPCM Images

Code	Length	Value	Range
00	2	0	0
01xxxx*	6	1 + xxxx	1 to 17
10xxxx	6	−1 − xxxx	−1 to −17
1100xxxxx	9	17 + xxxxx	17 to 49
1101xxxxx	9	−17 − xxxxx	−17 to −49
11100xxxxxx	11	49 + xxxxxx	49 to 113
11101xxxxxx	11	−49 − xxxxxx	−49 to −113
111101xxxxxxxxxxxxxx	20	xxxxxxxxxxxxxx	0 to 16,384
111100xxxxxxxxxxxxxx	20	−xxxxxxxxxxxxxx	0 to −16,384

Compression occurs because most of the DPCM pixel values are tightly clustered around 0. For an example of a more optimized code book for CT images, see Rabbani.[4]

Table adapted from Wilson.[5]

*x, bits that encode the DPCM pixel value.

Table 5. Simplified Huffman Codes

1st Byte	2nd Byte	3rd Byte	Pixel Value (2's complement*)
0xxxxxxx†			−64 to +63
10xxxxxx	xxxxxxxx		−8192 to +8191
110##### ‡	xxxxxxxx	xxxxxxxx	−32768 to +32767

A DPCM image is encoded with codes that are 1, 2, or 3 bytes long. A unique prefix distinguishes the code types. Data compression occurs because the majority of the DPCM pixel values are within the range −64 to +63 (see Table 3 in part 1 of this article).

*2's complement is the binary method of coding signed integers used by many different computers systems. For 8-bit signed integers, the sequence −3, −2, −1, 0, 1, 2, 3 is represented by the bit codes 11111101, 11111110, 11111111, 00000000, 00000001, 00000010, 00000011.

†x, bits that encode the DPCM pixel value.

‡#, these bits are not defined.

given a portion of the "probability range" between 0 and 1 that is as wide as the probability of that symbol. Collectively, the consecutive probability ranges of all the symbols extend from 0 to 1. To encode a message (or image), a running table of two floating point numbers is kept, the lower limit and upper limit (Table 7). These two numbers represent the lower and upper bounds of the final output code. At the

Table 6. Arithmetic Coding of the Text Example

Symbol	Frequency	Probability*	Probability Range†
' '	17	.1932	.0000 ≤ P < .1932
'O'	10	.1136	.1932 ≤ P < .3068
'E'	9	.1023	.3068 ≤ P < .4091
'T'	9	.1023	.4091 ≤ P < .5114
'I'	5	.0568	.5114 ≤ P < .5682
'R'	5	.0568	.5682 ≤ P < .6250
'S'	5	.0568	.6250 ≤ P < .6818
'M'	4	.0455	.6818 ≤ P < .7273
'A'	3	.0341	.7273 ≤ P < .7614
'H'	3	.0341	.7614 ≤ P < .7995
'C'	2	.0227	.7995 ≤ P < .8182
'D'	2	.0227	.8182 ≤ P < .8409
'F'	2	.0227	.8409 ≤ P < .8636
'G'	2	.0227	.8636 ≤ P < .8863
'L'	2	.0227	.8863 ≤ P < .9090
'N'	2	.0227	.9090 ≤ P < .9317
'P'	2	.0227	.9317 ≤ P < .9544
'U'	2	.0227	.9544 ≤ P < .9771
'W'	1	.0114	.9771 ≤ P < .9885
'Y'	1	.0114	.9885 ≤ P < 1.0000

This probability-calculating table would only be valid for this particular example.

*Symbol probability = symbol frequency/total symbols.

†Probability range is the region of probability from 0 to 1.0 assigned to that symbol. The width of the probability range is the same as the probability for each symbol.

Table 7. Arithmetic Coding Example: Encoding the Beginning of the Text Message

New Symbol	Lower Limit	Upper Limit
	.0000	1.0000
'N'	.9090	.9317
'O'	.91338564	.91596436
'W'	.915905307312	.915934704720
' '	.915905307312	.915910986891
'I'	.915908211849	.915908534449
'S'	.915908413474	.915908431798
Output code	.91590842	—

Note how the lower limit and the upper limit converge as more symbols are encoded.

start, the lower limit is set to 0 and the upper limit is set to 1. For the first symbol in our text example (N), the lower limit is set to the lower number of that symbol's probability range (.9090, from Table 6), and the upper limit is set to the higher number of the symbol's probability range (.9317). For the next symbol (O), the probability range of the new symbol (.1932 ≤ P < .3068) is assigned to the difference of the current lower and upper limit as follows:

$$LL_{new} = LL_{old} + (P_{low} \times R_{out}) \quad (1)$$

and

$$UL_{new} = LL_{old} + (P_{high} \times R_{out}) \quad (2)$$

where LL_{new} is the new lower limit, LL_{old} is the old lower limit, UL_{new} is the new upper limit, UL_{old} is the old upper limit, P_{low} is the lower probability number, P_{high} is the higher probability number. R_{out}, the output range, is defined as the difference between the old lower limit and upper limit:

$$R_{out} = LL_{old} - UL_{old} \quad (3)$$

The symbol (O) is coded by adding .00438564 to .9090 to calculate a new lower limit, and .00696436 is added to .9090 to calculate a new upper limit:

$$UL_{new} = .9090 + (.1932 \times .0227) = .91338564 \quad (4)$$

$$UL_{new} = .9090 + (.3068 \times .0227) = .91596436 \quad (5)$$

As the coding progresses, the lower and upper limits converge. After the last symbol is coded, any number between the upper limit and

the lower limit will be uniquely decoded as the original message.

An arithmetic coded message is decoded by reversing the process that produced the original code (Table 8). Of course, the same table that was used for compression must be used for decompression. In the example in Table 8, the code .91590842 falls in the range of the probability for the symbol (N), which becomes the first character of the message. The next symbol is calculated by removing the effect of the first symbol from the code:

$$\text{Code}_{\text{new}} = \frac{(\text{Code}_{\text{old}} - \text{LL}_{\text{old}})}{\text{P}_{\text{old}}}$$

$$= \frac{(.91590842 - .9090)}{.0227} = .304335682 \quad (6)$$

where LL_{old} is the lower limit of the old symbol and P_{old} is the probability of the old symbol.

In this example, the new code .304335682 falls in the probability range of the symbol (O). The next symbol is decoded by removing the effect of the symbol (O) from the code. The process is continued until all the symbols have been decoded. An end of message condition can be detected by either using a special end-of-message code, or keeping track of the number of characters in the original message.

Although the example is shown using floating point numbers, the algorithm can also be implemented using integers,[7] which makes implementation practical on personal computers.

Arithmetic coding can be very efficient in compression, but tends to be slower than Huffman and other forms of lossless compression. Arithmetic coding is especially effective when the probability of certain symbols is very high. In an example cited by Nelson,[7] a file consisting of 100,000 zeros and one end-of-file marker could be encoded with only 4 bytes. For radiologic images, arithmetic coding would most typically be used for coding DPCM images (in which the pixel values are tightly clustered around 0) or the transform coefficients of a lossy compression method after quantization (see below).

Table 8. Decoding the Arithmetic Example

Code	Symbol	Lower Probability Number*	Probability*
.91590842	'N'	0.9090	0.227
.304335682	'O'	0.1932	0.1136
.978307067	'W'	0.9711	0.0114
.105883076	' '	0.0	0.1932
.548049048	'I'	0.5144	0.0568
.592412822	'S'	—	—

*Number refers to the current symbol (Table 6).

Dictionary-Based Compression: Lempel-Ziv and Variants

The previously considered compression methods used a statistical model to encode fixed-length symbols with smaller bit strings. Dictionary-based compression instead encodes variable-length strings (or groups of pixels) as tokens. The tokens point to phrases in a dictionary. Compression occurs because the tokens are smaller than the phrases they represent.

Dictionary-based compression can have either fixed or adaptive dictionaries. Fixed dictionaries are useful for large databases that remain relatively constant over time, like the parts numbers in an inventory database or merchandise catalog. The dictionary must be present for both the compressor and the decompressor programs. Fixed dictionaries are generally not useful for general data or image compression in which the objects to be compressed differ markedly.

Practical adaptive dictionary-based compression began with the seminal work of Jacob Ziv and Abraham Lempel.[8,9] The adaptive dictionary-based compression algorithms begin either without a dictionary or with a small predefined dictionary. As compression proceeds, new phrases are added to the dictionary to be encoded later as tokens. The program outputs tokens (pointers to the dictionary) and plain-text phrases that have not yet been encountered and placed in the dictionary. Various methods have been used to distinguish between the tokens and plain text. The compression dictionary does not have to be stored with file because the decompression step creates the same dictionary as the file is decompressed.

The first version of the Lempel-Ziv compression algorithm is known as LZ-77.[8,10,11] This algorithm uses a dictionary that is a "sliding window" into the previously seen input file (Fig 4). The window may be 2 to 16 Kbytes long. A much smaller look-ahead buffer (often 100 to

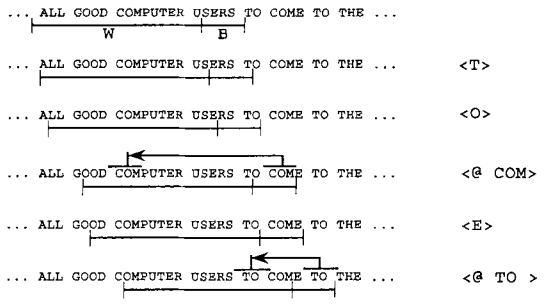

Fig 4. LZSS Compression. The algorithm uses a look-ahead buffer (B) that tries to match phrases in the sliding-window dictionary (W). In actual use, both the window and the buffer would be much larger. The coder outputs either text (if no phrase matches can be made) or a token that points to a matching phrase in the window (arrows).

200 bytes) tries to match incoming phrases with phrases in the dictionary. The program either outputs plain text or a token (pointer to the dictionary). The LZ-77 algorithm has been enhanced by improving the data structure associated with the dictionary window and improving the format of the output tokens. These enhancements are known as the LZSS compression algorithm. LZSS is a popular compression technique that is found in many commercial and shareware compression programs, for example, PKZIP and LHarc. An LZ-77 variant-compression method has also been implemented on a computer chip by Stac Electronics (Carlsbad, CA), and is widely used for compressing data for tape drives and other storage devices. This compression method is known by the standard which defines it: QIC-122.

The second major variant of Lempel-Ziv compression is known as the LZ-78 algorithm.[9,11] Instead of using a fixed width sliding window for the dictionary, the LZ-78 dictionary is built from all of the previously encountered symbols in the file. The dictionary is built a single token at a time, which allows the creation of very long dictionary strings. The algorithm outputs a token that consists of a pointer to a phrase in the dictionary and a single character. Each time a token is emitted, a new phrase entry is added to the dictionary.

A popular adaptation of the LZ-78 algorithm is the Lempel-Ziv-Welch (LZW) algorithm developed by Terry Welch[12-14] (Table 9). The LZW algorithm preloads the dictionary with all the symbols that will be encountered in the input file. Then, as the file is compressed, there are no symbols encountered that do not have an entry in the dictionary. Therefore, only tokens are emitted by the program. Like LZ-78, each time a token is emitted, a new entry is added to the dictionary. Either fixed-width or variable-width tokens can be used. The LZW algorithm is a popular compression method found in the UNIX COMPRESS program and in the tagged image file format (TIFF). The TIFF standard[14] specifies a variable-width 9- to 12-bit token. Compression is improved if the LZW algorithm is applied to a DPCM image, and this transfor-

Table 9. LZW Compression of the Sample Phrase "THAT THAT IS, IS, THAT THAT IS NOT, IS NOT."

		Input	Output	Dictionary			
Input	Output	Bits*	Bits†	Code	Parent	Character	String‡
'T'		8					
'H'	'T'	8	9	257	'T'	'H'	'TH'
'A'	'H'	8	9	258	'H'	'A'	'HA'
'T'	'A'	8	9	259	'A'	'T'	'AT'
' '	'T'	8	9	260	'T'	' '	'T '
'TH'	' '	16	9	261	' '	'T'	' T'
'AT'	257	16	9	262	257	'A'	'THA'
' '	259	8	9	263	259	' '	'AT '
'I'	' '	8	9	264	' '	'I'	' I'
'S'	'I'	8	9	265	'I'	'S'	'IS'
','	'S'	8	9	266	'S'	','	'S,'
' I'	','	16	9	267	','	' '	', '
'S,'	264	16	9	268	264	'S'	' IS'
' T'	266	16	9	269	266	' '	'S, '
'HA'	261	16	9	270	261	'H'	' TH'
'T '	258	16	9	271	258	'T'	'HAT'
'THA'	260	24	9	272	260	'T'	'T T'
'T '	262	16	9	273	262	'T'	'THAT'
'IS'	260	16	9	274	260	'I'	'T I'
' '	265	8	9	275	265	' '	'IS '
'N'	' '	8	9	276	' '	'N'	' N'
'O'	'N'	8	9	277	'N'	'O'	'NO'
'T'	'O'	8	9	278	'O'	'T'	'OT'
','	'T'	16	9	279	'T'	','	'T,'
'IS '	267	24	9	280	267	'I'	', I'
'NO'	275	16	9	281	275	'N'	'IS N'
'T'	277	8	9	282	277	'T'	'NOT'
'.'	'T'	8	9	283	'T'	'.'	'T.'
<EOF>	'.'		9				
	<EOF>		9				
Total		344	261				

The dictionary entries 0 to 255 are preloaded with all possible input characters. Entry 256 is the end-of-file (EOF) marker. As characters are input into the program, new entries are added to the dictionary. The coder outputs tokens, which either represent an original or new dictionary entry. In one implementation,[7] the dictionary is a data structure with three entries: the dictionary code, the parent code, and the recently added character. The parent code can either be an original or new dictionary entry.

mation is explicitly allowed in the TIFF standard. LZW compression is also the basis for the V.42*bis* data-communication standard that is widely available on most high-performance modems.

Compression of the Example Images

The example images (see Fig 1 in part 1 of this article) have been compressed using Huffman codes, arithmetic coding, and the LZW method (Table 10). Three commercial compression programs for the Macintosh (Apple Computers, Cupertino, CA) are included for comparison purposes. Three versions of each image were compressed: the original image, the image after the DPCM transformation (see part 1 of this article), and a smoothed + DPCM image. The smoothed image shows the effect of image noise on image compression. In general, the LZW compression was the most effective of the algorithms, especially after the DPCM transformation, and it performed almost as well as the best of the commercial programs. However, these examples should be considered only illustrative, because none of the compression methods used were optimized for 16-bit images. In addition, the CT scans can be compressed further if the "nonimage" pixels were encoded separately (Fig 1).

LOSSY IMAGE COMPRESSION

Image compression is considered lossy if the compressed file is not numerically identical to the original file after it has been restored. Lossy image compression is capable of much higher compression ratios than lossless compression. However, image-compression artifacts may be visible, and diagnostically useful information may be lost. There is a growing consensus that lossless compression is best used for short-term storage, and lossy compression may be useful for long-term storage. There is no consensus on

Table 10. Compression of the Example Images

	Compression Algorithm*			Commercial Compression Software (Macintosh)		
Image	Huffman†	Arithmetic‡	LZW§	Auto-Doubler‖	Disk-Doubler¶	StuffIt Deluxe**
Blank	87.5	99.93	99.75	89.7	99.89	99.96
Random noise	14.4	14.6	1.47	−0.03	3.8	13.9
DPCM	11.0	11.2	—	0.0	0.0	10.6
Smooth + DPCM	23.4	23.6	9.9	7.1	18.1	23.0
CT chest	36.5	37.5	44.4	34.8	49.1	51.8
DPCM	44.7	47.3	55.2	41.7	54.6	55.8
Smooth + DPCM	48.5	51.5	62.6	49.6	60.8	62.9
CT abdomen	35.2	36.1	44.0	33.9	47.7	51.2
DPCM	44.9	44.7	51.9	37.1	51.7	53.0
Smooth + DPCM	46.4	49.6	61.1	47.0	58.9	60.7
MRI knee	27.4	27.7	23.1	16.5	31.5	35.1
DPCM	35.5	35.6	33.0	21.4	35.4	37.5
Smooth + DPCM	42.5	42.5	45.6	33.3	45.1	47.4
MRI brain	34.3	35.4	34.8	27.0	39.7	43.2
DPCM	41.3	41.4	43.4	31.7	43.6	45.3
Smooth + DPCM	47.8	47.9	54.9	43.3	53.3	56.0
MRI spine	33.1	34.8	36.1	26.6	39.6	42.7
DPCM	41.8	42.0	42.9	31.8	43.0	44.8
Smooth + DPCM	48.8	48.7	54.2	42.7	52.5	54.8

Three versions of each image are compressed (except the blank image): normal (16-bit raster file without a header); difference image computed by DPCM (see equation 4 in part 1 of this article); and a DPCM image calculated after a 3 × 3 smoothing kernel applied to the image. The smoothing operation shows the effect of image noise on compression. Compare the actual compression achieved by the various algorithms and commercial packages with the image entropy in Table 2 in part 1 of this article.

*All compression figures expressed as percent compression. Huffman, arithmetic, and LZW code derived from Nelson.[7]

†Static order-0 8-bit model with Huffman coding.

‡Static order-0 8-bit model with arithmetic coding.

§9- to 16-bit variable width tokens, dictionary flushed when full.

‖Version 2.0.1. Fifth Generations Systems (Baton Rouge, LA).

¶Version 3.7.7. Fifth Generations Systems.

**Version 3.0.2. Aladdin Systems (Aptos, CA).

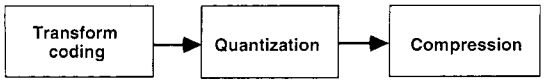

Fig 5. Lossy compression.

what constitutes an acceptable level of lossy image compression. Furthermore, lossy image compression is undesirable if any quantitative image measurements are required because the original pixel values are altered.

Generally, lossy compression is done in three stages: transform coding, quantization, and compression (Fig 5).

The transformation step converts the image into a format that is more suitable for compression and is usually lossless. Typically, the transformation step converts the image into a sequence of transformation coefficients. The specific transform coding method is useful for image compression if most of the image information is contained in relatively few of the coefficients. Two commonly used transform-coding methods include discrete cosine transformation (DCT) and discrete wavelet transformation (DWT) (see below). The quantization step, on the other hand, alters the coefficient values, usually by dividing each coefficient by an integer. The quantization step actually reduces the number of different output values in the transformation image and is the "lossy" step in this process. Frequently, many of the smaller coefficients are reduced to 0, which greatly improves the compressibility of the image. Finally, the actual compression step compresses the quantized coefficients. Modified Huffman codes and RLE are commonly used in this step.

Discrete Cosine Transform

The most common type of transform coding used for image compression is the DCT.[4,5,14,15]

743	966	1059	1041	984	943	922	923
932	1043	1037	994	947	922	915	941
1039	1032	990	948	926	924	943	977
1033	989	937	913	918	941	979	1016
998	945	919	910	929	966	1004	1034
970	932	927	939	969	1008	1033	1047
940	925	934	964	1001	1022	1047	1054
953	945	967	999	1030	1044	1052	1050

Fig 6. DCT example. 8 × 8 pixel block of CT image values before transformation. (Data from Rabbani.[4])

1946.13	-26.63	7.96	-13.96	-5.38	-4.15	2.72	-2.80
-23.19	48.32	-13.33	-25.10	-11.26	-9.33	-1.30	-1.41
11.98	-24.35	-55.82	-20.50	-8.98	-3.89	-2.98	-0.74
-15.01	-34.94	-22.40	-8.48	-3.87	-1.15	0.53	-0.66
-4.75	-15.16	-11.67	-6.01	-1.75	0.03	-0.24	0.69
-5.31	-10.72	-5.42	-0.36	0.42	0.43	1.57	0.75
-1.45	-3.93	-3.48	-0.19	0.20	-0.83	1.19	0.47
-0.97	-3.61	-1.32	-0.24	0.31	0.65	0.46	0.73

Fig 7. CT image values after DCT. The high-frequency coefficients are in the lower right of the matrix.

The DCT belongs to a family of transform equations that include the fast Fourier transform (FFT). Like the FFT, the DCT converts a matrix of pixel values from the spatial domain into the frequency domain (Figs 6-9). However, the computation time required increases markedly with the matrix size, so the image is typically divided into a series of N × N pixel blocks before the transform, where N is either 8 or 16 pixels. After the transformation, most of the energy in the image is contained in relatively few of the low-frequency transform coefficients. The low frequencies in the image are in the upper left of the transform matrix, and the high frequencies are in the lower right. The weighted average of all pixel values is the first value of the matrix. Most of the perceptible information content in radiologic images (and real world images in general) is contained in the low frequencies, and the high-frequency information can be reduced with little impact on image quality.

Significant data compression is possible after quantization of the DCT image. In the simplest method of quantization, each value of the DCT matrix is divided by an integer, and then rounded to the nearest integer. The quantization step reduces the possible values of the transform

122	-2	0	-1	0	0	0	0
-1	3	-1	-2	-1	-1	0	0
1	-2	-3	-1	-1	0	0	0
-1	-2	-1	-1	0	0	0	0
0	-1	-1	0	0	0	0	0
0	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig 8. DCT values after quantization. Each coefficient was divided by 16 and rounded to the nearest integer. Note how most of the high-frequency coefficients have been set to 0.

122	-2	-1	1	3	0	-1	-1
-2	-1	0	-2	-3	-2	0	0
-1	-1	-1	-1	0	0	-1	0
-1	-1	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig 9. DCT matrix after coefficients reordered by zig-zag sequence (Fig 10). Note that most of the 0 coefficients are now in sequence, which improves compressibility of the block.

coefficients, and many of the smaller coefficients are set to 0 (Fig 8). Then the quantified coefficients are rearranged in sequence, with the upper-left values first and the lower-right values last. This sequence is termed the zig-zag sequence (Fig 10), and puts the higher frequency coefficients in each encoded block last.

Different strategies for the quantization step can be used. Options include using a single integer for all the coefficients (as mentioned above), setting a predetermined number of the high frequency coefficients to 0, and using a quantization matrix. A quantization matrix is an integer matrix that is divided into the corresponding value of the DCT matrix. The values of the quantization matrix are typically larger for the high-frequency coefficients than the low-frequency coefficients. The goal of the quantization matrix is to preferentially reduce the high-frequency information content of the image. Finally, the actual compression of the data is performed. Huffman codes are commonly used, sometimes after run length encoding the 0 coefficients. The steps are summarized in Fig 11.

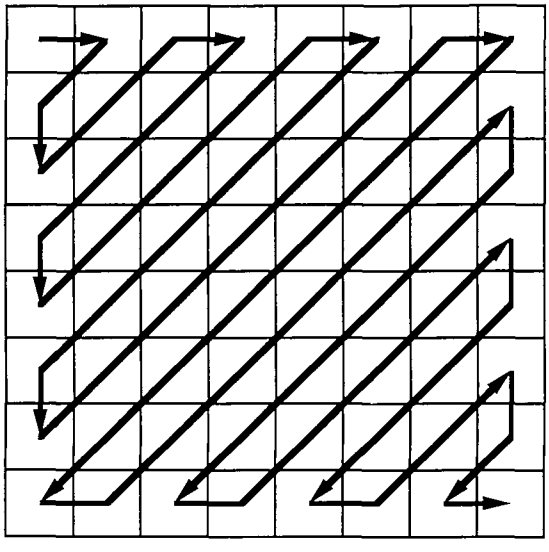

Fig 10. The zig-zag sequence. The transformation coefficients are reordered so that the low-frequency components precede the high-frequency components.

The DCT is an important part of the Joint Photographic Experts Group (JPEG) lossy image-compression standard.[16] This standard is an increasingly popular method for compressing images on personal computers. Fortunately, radiology will benefit from the large of amount of research effort devoted to implementing and improving the JPEG standard.

The major disadvantages of the DCT image-compression method include the considerable computation time and image artifacts at high-compression ratios. Computation times can be significantly reduced by using special hardware optimized for this technique.[17] At higher compression ratios, a "blocking" artifact may be noticeable. This artifact occurs when the boundaries of the DCT N × N matrix become visible at higher compression ratios.

Wavelet Compression

Wavelet compression uses "wavelet functions" as the basis for the DWT. These functions satisfy certain mathematical criteria, and are all

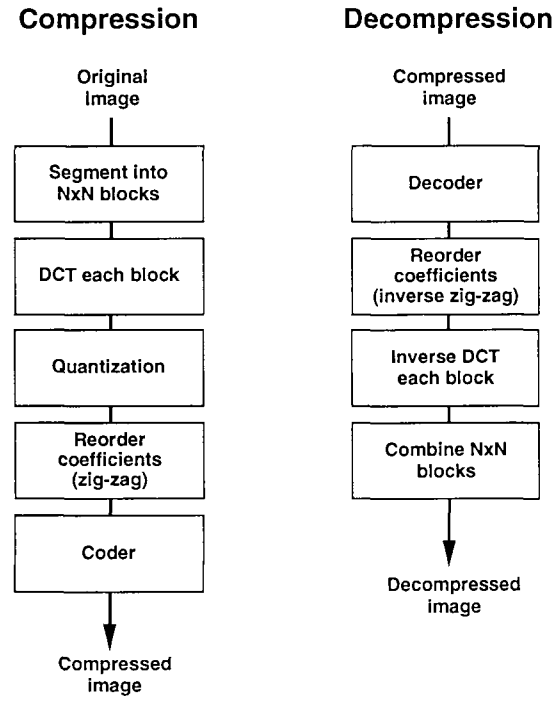

Fig 11. DCT overview.

Fig 12. Wavelet compression. (A) The classic Lena image. (B) The subimages of the DWT for this image. Reprinted with permission from Manduca.[18]

translations and scalings of each other.[18] Unlike the DCT or FFT, which convert image spatial data into frequency data, the wavelet functions are partially localized in both space and frequency. The DWT is applied repeatedly to the image, which results in a multilevel wavelet hierarchy (Fig 12). At each higher level of the hierarchy, the transform coefficients are less localized in space and more localized in frequency. Like the DCT described above, image compression is possible because most of the image information is contained in relatively few of the transform coefficients. There are an infinite number of possible wavelet functions, and different functions offer tradeoffs in image fidelity and compression. Preliminary results suggest wavelet compression results in better image quality than DCT/JPEG image compression for a given level of compression.[18]

Fractal Compression

Fractals are geometric structures generated from simple formulas that have the peculiar property of looking similar at different levels of magnification. Many natural structures such as trees and clouds have been shown to have fractal characteristics and these objects can be closely mimicked using fractal geometry.[19] Fractal-based image compression is performed by the fractal transform, which translates fractal-like patterns in an image into a series of fractal formulas.[20,21] The image is compressed because the formulae are much smaller than the image they describe. The formulae are also resolution independent, which allows the image to be decompressed at higher or lower resolution than the original with little loss in image quality. The process is markedly asymmetric, ie, the compression step is much more computationally intensive than the decompression step. Advocates of fractal compression claim fractal decompression is much faster than JPEG compression, and image quality is superior, especially at higher levels of compression.[22] The suitability of fractal compression for radiology is largely unknown, but the technology appears especially attractive for compressing images for teaching and reference programs.

REFERENCES

1. Lynch TJ: Data Compression: Techniques and Applications. Belmont, CA, Lifetime Learning Publications, 1985, pp 55-56

2. Held G, Marshall TR: Data Compression: Techniques and Applications, Hardware and Software Consideration (ed 2). New York, NY, Wiley, 1987, pp 96-107

3. Huffman DA: A method for construction of minimum-redundancy codes. Proc Institute for Radio Engineers 40:1098-1101, 1952

4. Rabbani M, Jones PW: Image compression techniques for medical diagnostic imaging systems. J Digit Imaging 4:65-78, 1991

5. Wilson DL: Compressed radiologic images and workstation viewing. J Digit Imaging 5:168-175, 1992

6. Witten IH, Neal RM, Cleary JG: Arithmetic coding for data compression. Communications Association for Computing Machinery 30:520-540, 1987

7. Nelson M: The Data Compression Book. San Mateo, CA, M&T Books, 1992

8. Ziv J, Lempel A: A universal algorithm for sequential data compression. IEEE Trans Info Theory 23:337-343, 1977

9. Ziv J, Lempel A: Compression of individual sequences via variable-rate coding. IEEE Trans Info Theory 24:530-536, 1978

10. Fiala ER, Greene DH: Data compression with finite windows. Communications ACM 32:490-505, 1989

11. Storer JA: Data Compression: Methods and Theory. Rockville, MD, Computer Science, 1988

12. Welch T: A technique for high-performance data compression. IEEE Computer 17:8-19, 1984

13. Nelson MR: LZW data compression. Dr Dobb's J 14:29-87, 1989

14. TIFF, revision 6.0. Aldus Developers Desk, Aldus Corporation, Seattle, WA

15. Embree PM, Kimble B: C Language Algorithms for Digital Signal Processing. Englewood Cliffs, NJ, Prentice Hall, 1991

16. Wallace GK: The JPEG still picture compression standard. Communications Association for Computing Machinery 34:30-44, 1991

17. Siegel EL, Templeton AW, Hensley KL, et al: Image data compression using a new floating-point digital signal processor. J Digit Imaging 4:188-195, 1991

18. Manduca A: Interactive wavelet-based 2-D and 3-D image compression. SPIE Med Imaging 1897:307-318, 1993

19. Mandelbrot BB: The Fractal Geometry of Nature. New York, NY, WH Freeman and Co, 1982

20. Barnsley MF: Fractals Everywhere. Boston, MA, Academic, 1988

21. Barnsley M: Image Compression Using the Fractal Transform. Image Processing 90—the Key Issues. Conference Proc. London, UK, Blenheim Online, 1990

22. Banet B: Is fractal worth holding out for? The cold fusion of image compression begins to get some respect, in: Digital Media: A Seybold Report. Media, PA, Seybold, 1992, p 9