Project Report
ATC-261

The Weather-Huffman Method of Data Compression of Weather Images

J. Gertz

31 October 1997

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Federal Aviation Administration,
Washington, D.C. 20591

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. ATC-261	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The Weather-Huffman Method of Data Compression of Weather Images		5. Report Date 31 October 1997
		6. Performing Organization Code
7. Author(s) Jeffrey L. Gertz		8. Performing Organization Report No. ATC-261
9. Performing Organization Name and Address MIT Lincoln Laboratory 244 Wood Street Lexington, MA 02173-9108		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTFA01-93-Z-02012
12. Sponsoring Agency Name and Address Department of Transportation Federal Aviation Administration Systems Research and Development Service Washington, DC 20591		13. Type of Report and Period Covered Project Report
		14. Sponsoring Agency Code

15. Supplementary Notes

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, under Air Force Contract F19628-95-C-0002.

16. Abstract

Providing an accurate picture of the weather conditions in the pilot's area of interest is a highly useful application for ground-to-air datalinks. The problem with using data links to transmit weather graphics is the large number of bits required to exactly specify the weather image. To make transmission of weather images practical, a means must be found to compress the data to a size compatible with a limited datalink capacity.

The Weather-Huffman (WH) Algorithm developed in this report incorporates several subalgorithms in order to encode as faithfully as possible an input weather image within a specified datalink bit limitation. The main algorithm component is the encoding of a version of the input image via the Weather Huffman runlength code, a variant of the standard Huffman code tailored to the peculiarities of weather images.

If possible, the input map itself is encoded. Generally, however, a resolution-reduced version of the map must be created prior to the encoding to meet the bit limitation. In that case, the output map will contain blocky regions, and higher weather level areas will tend to bloom in size.

Two routines are included in WH to overcome these problems. The first is a Smoother Process, which corrects the blocky edges of weather regions. The second, more powerful routine, is the Extra Bit Algorithm (EBA). EBA utilizes all bits remaining in the message after the Huffman encoding to correct pixels set at too high a weather level. Both size and shape of weather regions are adjusted by this algorithm.

Pictorial examples of the operation of this algorithm on several severe weather images derived from NEXRAD are presented.

17. Key Words Data Compression Weather Images	18. Distribution Statement This document is available to the public through the National Technical Information Service, Springfield, VA 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	88	

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

LIST OF TABLES

1. INTRODUCTION

It is often said that "a picture is worth a thousand words". This is certainly the case when it comes to giving the pilot of an aircraft a description of the weather. Providing an accurate picture of the weather conditions in the pilot's area of interest could be a highly useful application of a ground-to-air datalink. Even those pilots that have onboard weather radars could profit from the ability to see the weather in regions beyond the range of their radar, or seeing the nearby region with the greater clarity that a ground-based weather radar can provide.

1.1 WEATHER RADAR INFORMATION

The present TDWR (Terminal Doppler Weather Radar), ASR-9 (Airport Surveillance Radar), and NEXRAD (NEXt generation weather RADar) radars are designed to provide hazardous weather information to controllers located at the tower or at an enroute center. While the voice message pilots receive from the terminal controllers will facilitate the avoidance of hazardous weather, it will not provide "escape" information. The lack of vectoring instructions will leave the choice of alternate flight paths to the pilot. With such a system, the possibility of a pilot encountering weather situations as dangerous, or even more dangerous, than those in the verbal warning, or his not understanding the message enough to avoid the hazard, becomes a possible scenario.

During the enroute portions of flight, graphical weather information could be used in a number of applications to increase flight safety. This information would be useful as warnings of immediate weather hazards that need be avoided, as information of possible escape maneuvers once hazardous weather has been encountered, and for long range strategic flight path planning information. Because NEXRAD images are so important to in-flight decision making, they are used as examples throughout this report.

1.2 GROUND-TO-AIR DATA LINKS

To provide pilots with the information needed to make informed decisions on the avoidance of hazardous weather, and to supply them with the same information the controllers have, the FAA is actively developing the capability to provide real time graphical information of hazardous weather conditions to aircraft by use of datalink. Several forms of ground-to-air datalinks are currently under development.

The problem with using any datalink to transmit weather graphics is the large number of bits required to specify a weather image. For the purposes of this report, a weather image is defined as an array of 256x256 (65,536) pixels. This array size is typical of a graphic, such as a NEXRAD image, that could be transmitted and displayed in an aircraft using modern equipment. Each pixel can indicate either clear sky or one of the six National Weather Service weather reflectivity levels (see Table 1 below), and thus requires 3 bits to specify. Hence, a weather image, such as the example shown in Figure 1-1, will consist of 196,608 bits of information, not counting any datalink overhead. This size will impose a severe load upon any practical ground-to-air datalink.

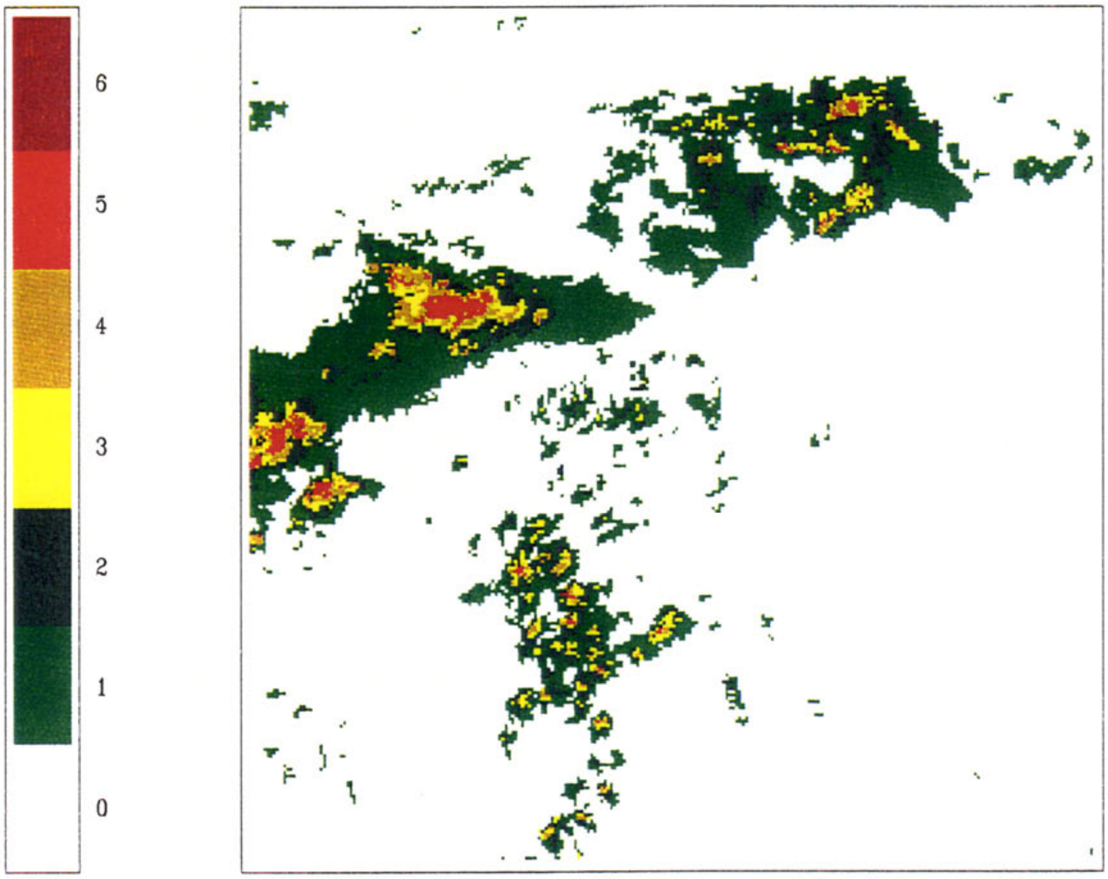


Figure 1-1. Sample Stormy Weather Image.
Mobile AL, 11 *May* 1995, 6 *PM*.
256 *x* 256 *Pixels*, 512 *km x* 512 *km*.

Table 1. National Weather Service Reflectivity Levels

NWS Level	Precip. Intensity	Rainfall in/hr	Possible Turbulence	Hail	Lightning
LEVEL 6	Extreme	≥ 7.1	Severe	Large	Yes
LEVEL 5	Intense	4.5-7.1	Severe	Likely	Yes
LEVEL 4	Very Strong	2.2-4.5	Severe	–	Yes
LEVEL 3	Strong	1.1-2.2	Severe	–	Yes
LEVEL 2	Moderate	0.2-1.1	Light/ Moderate	–	No
LEVEL 1	Weak	< 0.2	Light/ Moderate	–	No

To make transmission of weather images using datalinks practical, a means must be found to significantly compress the image; a typical aeronautical datalink would require a 65-to-1 compression if it were to limit the uplinked image transmission to 3600 bits. In addition, the algorithms used to perform the compression and decompression must be computationally efficient: the ground computer may have to handle many aircraft requests in a given scan, and the airborne computer may not be particularly powerful or have extensive memory capacity.

1.3 STANDARD COMPRESSION APPROACHES

There are many data compression techniques that are well-developed and documented. These techniques include runlength encoding, select encoding, Huffman encoding, Lempel-Ziv encoding, and arithmetic encoding [1]. These algorithms are drawn from a variety of applications including text-file archiving, compression of Fax images, and transmission of general pictures. All of these approaches make use of the redundancies in the data to achieve compression. They are general techniques, independent of the nature of the data. They also maintain exact fidelity under compression — the decompressed image exactly matches the raw image. However, as a result of this property, they cannot guarantee a maximum bit limit.

Several of these general approaches were tested on sample weather images taken from operating weather radars as well as from weather service providers [2]. The best of these approaches, the Huffman [3], could only produce compressions of the order of 3-to-1 or 4-to-1 on the sample images. None ever achieved the compressions required to fit weather images into typical datalink limitations. It became clear that an algorithm based on the unique attributes of weather images would be required to achieve the necessary compression.

It also became apparent that exact fidelity under compression could not always be achieved for this application; some amount of distortion in the transmission of the weather images over the datalink would often have to be tolerated. This opened the door to modern data compression approaches such as fractals, cosine transforms, and wavelets. These algorithms were found to be inappropriate for the types of graphics represented by weather images (few bits, low redundancy), and also require too great an encoding and decoding overhead for the target class of computers, especially those aboard general aviation aircraft.

As a result of these observations, two methods of compression tailored to weather images were developed at Lincoln Laboratory: Polygon-Ellipse [4] and Weather-Huffman. Polygon-Ellipse was implemented and field tested first, but human factors evaluation indicated that the images produced were unacceptable at high compression levels. Later evaluations indicated that Weather-Huffman removed the drawbacks noted by subject pilots.

The algorithm described in the remainder of this report, specially tailored for weather images, maximizes the number of images that can be sent exactly. When lossless representation is impossible, it trades off controlled amounts of distortion for increased compression. The ultimate test of this algorithm is that it produces useful and accurate images when bit-limited by representative ground-to-air datalink limits.

1.4 WEATHER IMAGE COMPRESSION ALGORITHM

Weather images exhibit a great deal of structure — they are far from random sets of pixels. If the compression algorithm can make use of this structure, the result will be the achievement of greater compression than would be possible for a "general" algorithm which simply depended on the basic redundancy of the image.

One useful property of weather-map images is their "contiguity". Weather regions tend to form smooth, continuous areas instead of isolated random spots. Abrupt edges or disconnected segments of weather are rare as seen in Figure 1-2.

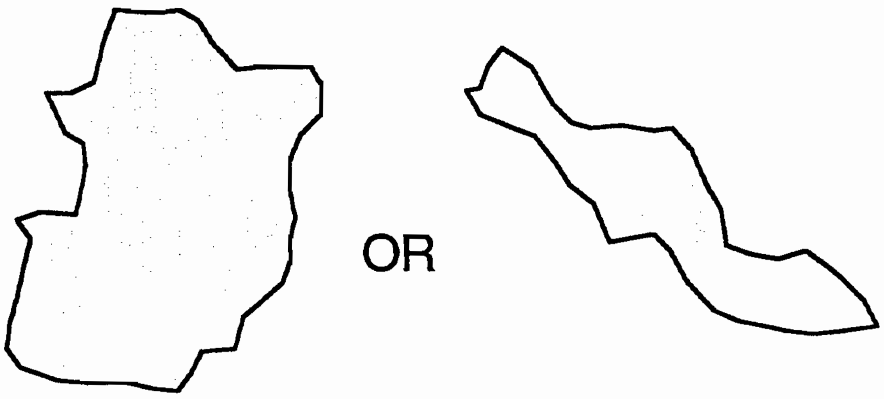


Figure 1-2. Sample weather regions showing the "contiguity" of weather images.

A second property of weather images might be termed "continuity" — regions of intense (higher-level) weather usually are wholly contained within regions of less intense weather. In fact, weather transitions are generally restricted to plus or minus one level as seen in Figure 1-3.

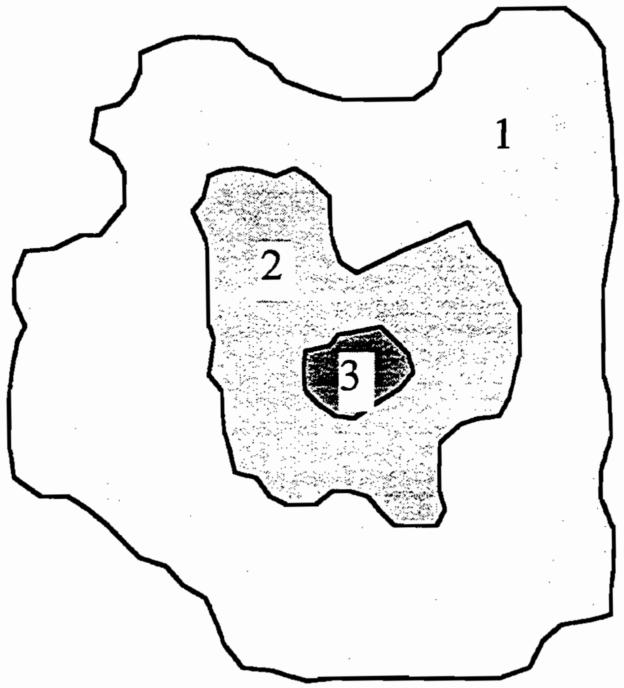


Figure 1-3. Sample weather region showing weather transition levels.

These two properties are exploited in the compression technique developed in this project: The Weather-Huffman (WH) Algorithm. The key element of the WH algorithm is a Huffman-type runlength encoding scheme that has built-in assumptions tied to typical weather images; when these assumptions hold, the bit requirements of WH are far less than the theoretically optimum Huffman code.

The WH algorithm contains procedures for gracefully degrading the resolution of the transmitted image when necessary to meet a specified bit limit. It also contains an auxiliary Extra Bit Algorithm (EBA) which makes use of any bits remaining after the WH compression step to restore as much as possible of the fidelity lost by this resolution reduction.

1.5 REPORT OUTLINE

Section 2 of this report provides an overview of the Weather-Huffman (WH) approach to weather image compression. This section also includes results of applying the algorithm with a bit limitation (3500 bits) to sample complex real-life weather images. Finally, it estimates the ground computer and onboard avionics requirements for implementing the WH encoding and decoding routines.

Section 3 provides an overview of the theory of the Huffman runlength algorithm, the foundation of the WH approach. It also presents the principles of the decoding table, which must be included with the encoded image in the uplink message in order to permit the decoder to decipher the sequence of codewords that constitute the image representation.

Section 4 then details the complex procedures used to convert the general Huffman method to one that applies specifically to the weather image situation. In particular, the techniques employed to compress the bit requirements of the decoding table are described in depth. Without these modifications, the decoding table alone could use up all the bits of the uplink message.

Section 5 then presents the methods needed to simplify the input weather image when its exact representation exceeds the datalink bit limit. Both filtering and resolution reduction of the weather image are employed. The former eliminates isolated pixels to reduce the number of runs on the image, while the latter maps a block of pixels into a single "superpixel" (2x2->1, or 4x4->1, or 8x8->1) to reduce the number of pixels to encode.

Section 6 then presents the smoothing algorithm employed by the decoder to eliminate the blockiness of the decoded image that results whenever resolution reduction was required. That is, the edges of a 2x2, 4x4, or 8x8 expansion are modified to better match the true shapes of weather regions.

Section 7 provides the details of the Extra Bit Algorithm (EBA), which utilizes all message bits remaining after the Huffman encoding of the image to improve the accuracy of the decoded image whenever resolution reduction was required. In particular, it specifies which quadrants of a 1->2x2 expansion should be set to a weather level other than the level of the expanded superpixel.

Finally, Sections 8 and 9 present the complete set of steps used by the encoder and decoder respectively in the Weather-Huffman Algorithm, making use of the information provided in the earlier sections. They also provides a detailed description of the bit format of the encoded data stream sent on the datalink. The level of detail provided in these sections is sufficient for a programmer to understand the C-coded subroutines developed as part of this project.

Sample results of the Weather-Huffman algorithm are presented in Section 10. The effects of including or omitting each part of the algorithm are illustrated, so that the relevant tradeoffs can be realized. Also various bit limitations are explored, so that the graceful degradation of the approach will be apparent.

Appendix A presents, for each weather level, the three default codeword sets available for the Huffman encoding step. Appendix B lists the set of user-specified parameters that currently exist in the WH software. For each, an explanation is provided of the parameter's effect and its default setting.

2. WEATHER-HUFFMAN ALGORITHM OVERVIEW

The Weather-Huffman (WH) Algorithm incorporates several subalgorithms in order to encode as faithfully as possible an input weather image within a specified datalink bit limitation. The main algorithm component is the encoding of a version of the input image via the Weather-Huffman runlength code, a variant of the standard Huffman code tailored to the peculiarities of weather graphics. In addition, this code has features designed to significantly reduce the bits needed to transmit the decoding table, an overhead concern of all Huffman codes.

If possible, the input image itself is encoded. Generally, however, a resolution-reduced version of the image must be created prior to the encoding to meet the bit limitation. In that case, the output image will contain blocky regions, and higher weather level areas will tend to bloom in size.

Two routines are included in WH to overcome these problems. The first is a Smoother Process, which corrects the blocky edges of weather regions. The second, more powerful routine, is the Extra Bit Algorithm (EBA). EBA utilizes all bits remaining in the message after the Huffman encoding to correct pixels set at too high a weather level. Both size and shape of weather regions are adjusted by this algorithm.

2.1 WEATHER IMAGE HUFFMAN CODE

The standard Huffman code is optimal for a random sequence of pixel values. However, a weather image usually has specialized properties that violate the randomness assumption; these include the 2-dimensional contiguity and continuity properties illustrated in Section 1. These properties have been exploited to produce a code, named the Weather-Huffman code, specifically tailored to weather images, that requires fewer bits on average than the "optimal" Huffman code.

This code, instead of being a strict runlength code (with a codeword indicating a specific run at a specific level, such as 3 pixels at level 2), is rather a runlength/transition code. In particular, the algorithm assigns a complete Huffman code to each weather level, 0-6, to encode runs at that level. Transitions between weather levels, whenever a choice of direction exists (see example below), are encoded by a single bit:

0: go up to the next higher level

1: go down to the next lower level

As long as most transitions are a single change in level, this code will significantly exceed in performance the standard single Huffman runlength code. In particular, for 6 level weather images, the savings is 2 bits per run, while for 3 level weather images the savings is 1 bit per run.

Guided by the contiguity assumption, the scanning mechanism used to produce the string of pixels for runlength encoding is the Hilbert scan, rather than the traditional row-by-row raster scan. The Hilbert scan is a 2-dimensional space-filling scan with the property that all pixels in a binary 2^n square are visited before that square is left. The Hilbert scan for a 32*32 square can be seen in Figure 2-1. Note that, as just stated, each quadrant is completed before the next quadrant is entered.

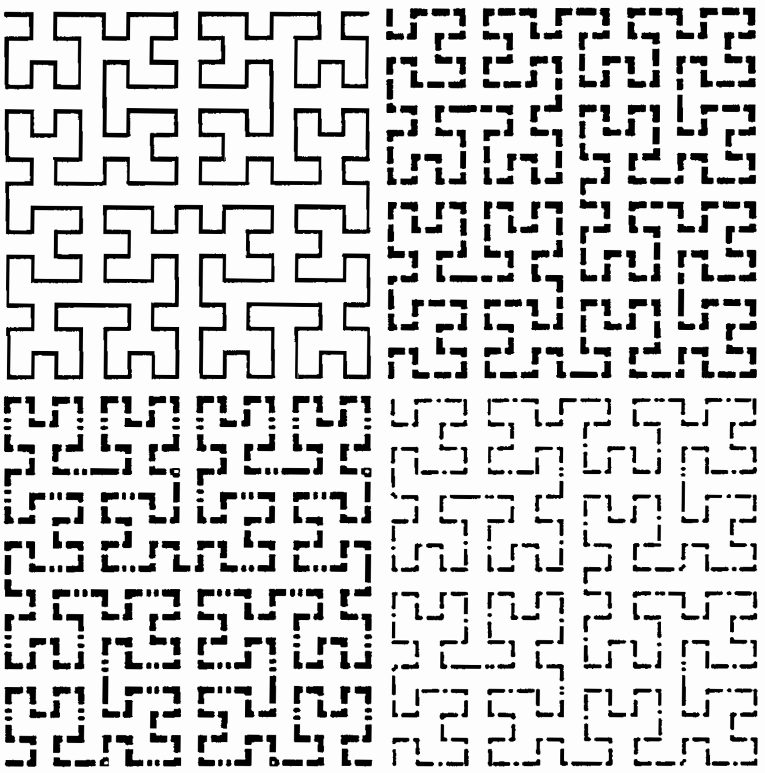


Figure 2-1. The Hilbert scan for a 32∗32 square. Each quadrant (different line type) is completed before another is entered.

The advantage of a Hilbert scan for 2-dimensional objects such as weather regions is that a few long runs are produced while the scan meanders within the region, combined with several very short runs as the scan nicks the region's edges. Although the total number of runs produced is about the same as would result from a raster scan, the dichotomy of lengths is advantageous to a Huffman coding: since the large majority of runs are concentrated in a few (short) lengths, short Huffman codewords can be assigned to most runs. In addition, if filtering of the runs is required to meet a message bit limit, numerous single length runs are available to the filtering process.

The codes for the various weather levels are all runlength in nature. Codewords are provided for each possible length run:

0 pixels at the level

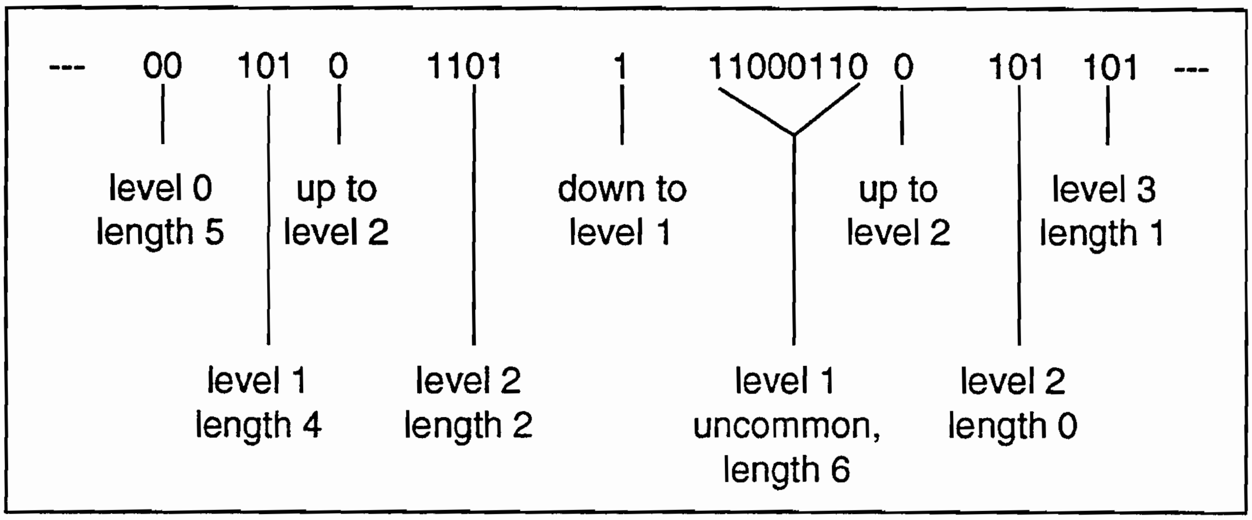
1,2,3 ... n consecutive pixels at the level (common lengths)

63 consecutive pixels (longest encoded run)

"other" (uncommon) length runs

The codeword for 0 length runs is required to cover the case of transitions of more than one weather level. The uncommon "other" length runs all share the same initial codeword to provide a manageable number of codewords and minimize the decoding table length. The bits after this codeword then specify the actual runlength.

As an example of the Weather-Huffman approach, a typical codeword sequence for a given segment of the image might be as follows:



Note the following facts illustrated by this example:

1. Implied level transitions are not explicitly stated:
    a. after any run of 0's, such as the initial one, level 1 must follow
    b. after a 0 length run, such as that at level 2 near the end, a transition in the same direction as the previous one must occur
    c. (not shown) after any run at the highest weather level on the image, the next lower level must follow
2. Since each level uses its own set of codewords, the same codeword (such as 101) can have different interpretations on different levels: length 4 for level 1, length 0 for level 2, and length 1 for level 3 in this example
3. Uncommon lengths require a codeword signifying that fact (11 for level 1) followed by the length in binary (6 bits are needed when the longest run is 64 pixels)
4. As with any Huffman code, codewords vary in length

A drawback to the Huffman encoding technique is that the Huffman code tables must be sent along with the data, and the overhead of the tables may outweigh the benefits of the data compression. The WH algorithm uses two techniques to minimize the Huffman table transmission drawback. First, a set of default tables is defined for each weather level. If one of these tables proves to be suitable, only the default table index need be transmitted for that level.

A second technique uses techniques to significantly compress any Huffman table that must be transmitted. In particular, only the code lengths are sent, never the codes themselves. The decoder can infer the actual codes from the list of lengths. Furthermore, the code lengths are sent via very few bits, often only 1 bit per codeword even when the lengths are as great as 7. Finally, a special codeword represents all uncommon long runlengths; this codeword is then followed by a specification of the actual length of the run. This feature removes the need for a large section of the coding table.

2.2 RUNLENGTH FILTERING AND RESOLUTION REDUCTION

If the Huffman runlength encoding of the exact input weather image requires more than the allowable number of bits, some form of detail reduction must be performed on the image to lower the encoding load. Two types of image simplification are used as part of the WH approach: runlength filtering and resolution reduction.

Isolated weather pixels add substantially to the Huffman runlength encoding requirement. For example, the following sequence of pixels — 221222 — produces code for 3 runs and 2 level changes. By changing the isolated pixel to a 2, the encoding reduces to that for a single run. Altering the weather level of such isolated pixels is in some cases deemed acceptable distortion for the large benefit in data compression that is achieved.

In the example of the NEXRAD radar graphic, for pilot safety considerations, no reduction in level is permitted for moderate (level 2) or greater pixels in the filtering operation; only level 1 pixels may be reduced. Any weather level pixel, however, may be increased a single increment by filtering, and in addition level 0 pixels may be set to level 2. Thus, examples of the application and rejection of filtering operations are:

(a)	11011	—>	11111	0 can increase
(b)	33433	—>	33433	4 can't be reduced
(c)	12033	—>	12233	0 can be raised to 2
(d)	11233	—>	11233	no gain by changing the 2 to 3
(e)	101010	—>	111111	prefer raising 0 to lowering 1

Another, more severe, way to decrease the Huffman runlength encoding bit requirements of a weather image is to reduce the resolution of the image from 256x256 pixels to 128x128 pixels, with each new pixel representing 4 of the old ones (a 2x2 square). This change produces a blockier image, but maintains reasonable fidelity. If this reduction is not sufficient to satisfy the bit limit, further reduction to 64x64 pixels, or even 32x32 pixels, is utilized in the WH algorithm.

The approach chosen to produce the various size superpixels emphasizes higher levels of weather, in that the new pixel setting is influenced most by the highest weather level in the superpixel square. The level chosen also depends upon the weather in the neighboring superpixels, in that no region of weather of level 3 or above, even if only 1 pixel in size, will ever be permitted to be lost via resolution reduction.

2.3 SMOOTHING PROCESS

Whenever the Weather-Huffman encoding requires resolution reduction of the input image to meet the datalink bit limitation, the decoder must expand the received image back into its full resolution. Without any other information about the original image to go by, the decoder is reduced to replicating each received superpixel into 4 quadrants of same-level pixels at each step of the regeneration. Thus, if a received superpixel were of size 4x4, the decoder would generate 4 2x2 superpixels, and then 16 1x1 actual pixels, all of the same weather level. The result of this operation is the generation of an output image with weather regions that are larger and blockier than those on the input image.

The Smoothing Process attempts to mitigate this effect by using knowledge of weather region shapes to reduce in level some of the pixels at each regeneration step. For example, weather regions do not often have sharp corners. The Smoothing Process, for reasons of pilot safety, is not allowed to reduce the extent of weather regions. It is only permitted to reshape the edges of regions to drive them closer to expected reality.

2.4 EXTRA BIT ALGORITHM (EBA)

Should the Huffman coding of the Hilbert-scanned input image produce too many bits to meet the datalink limit, the Huffman encoding will need to be applied to a reduced resolution version of the input image, with pixels replaced by superpixels (2x2, 4x4, or 8x8). Then the decoder will expand each received superpixel into an array of pixels, each with the same level. This will result in output weather regions being noticeably larger and more blocky than those on the input image. While the smoothing algorithm attempts to reduce this effect, it has two major drawbacks:

1. only a few pixels are subject to level reduction, and
2. the smoother makes its reduction without any knowledge of the actual weather contours

In effect, the smoothing algorithm uses knowledge of typical weather region shapes to produce a more esthetically pleasing result, which is usually, but not guaranteed to be, more accurate than the decoder output.

The Extra Bit Algorithm (EBA) utilizes the bits remaining in the uplink message to relate to the decoder correction information for selected quadrants of as many superpixels as possible. The information, one bit per quadrant, says either to leave the quadrant at the superpixel level or to reduce the quadrant by 1 weather level. Successive applications to the same quadrant, via more than one pass through EBA, can achieve greater than 1 level reductions, and quadrants that were incorrectly smoothed down can be returned to their original correct values.

2.5 SAMPLE IMAGE RESULTS

Weather images representing light weather conditions require no compression to meet a 3500 bit limit. Figures 2-2a and 2-2b illustrate the performance of the Weather-Huffman Algorithm on images that do require compression, namely severe weather images (the worst we could find) produced at various U.S. locations. The first and last images were taken in the New York area, the third at Mobile, and the second represents hurricane Allison at Tallahassee.

Each image is 256x256 pixels, with a pixel representing 2 kilometer by 2 kilometers; thus each image covers a region approximately 300 nautical miles on a side. The figure in each case compares the input weather image to the output image provided the pilot.

It was required in each case that the WH algorithm input at least a 4x4 resolution-reduced image to the runlength encoder (the last image required 8x8 reduction because of its extremely large number of small heavy weather regions). This level of reduction appears typical for severe weather images. However, the output images were in each case quite reasonable representations of "truth". Some blockiness and region blooming are apparent in each output, though the fidelity was judged operationally acceptable in each case.

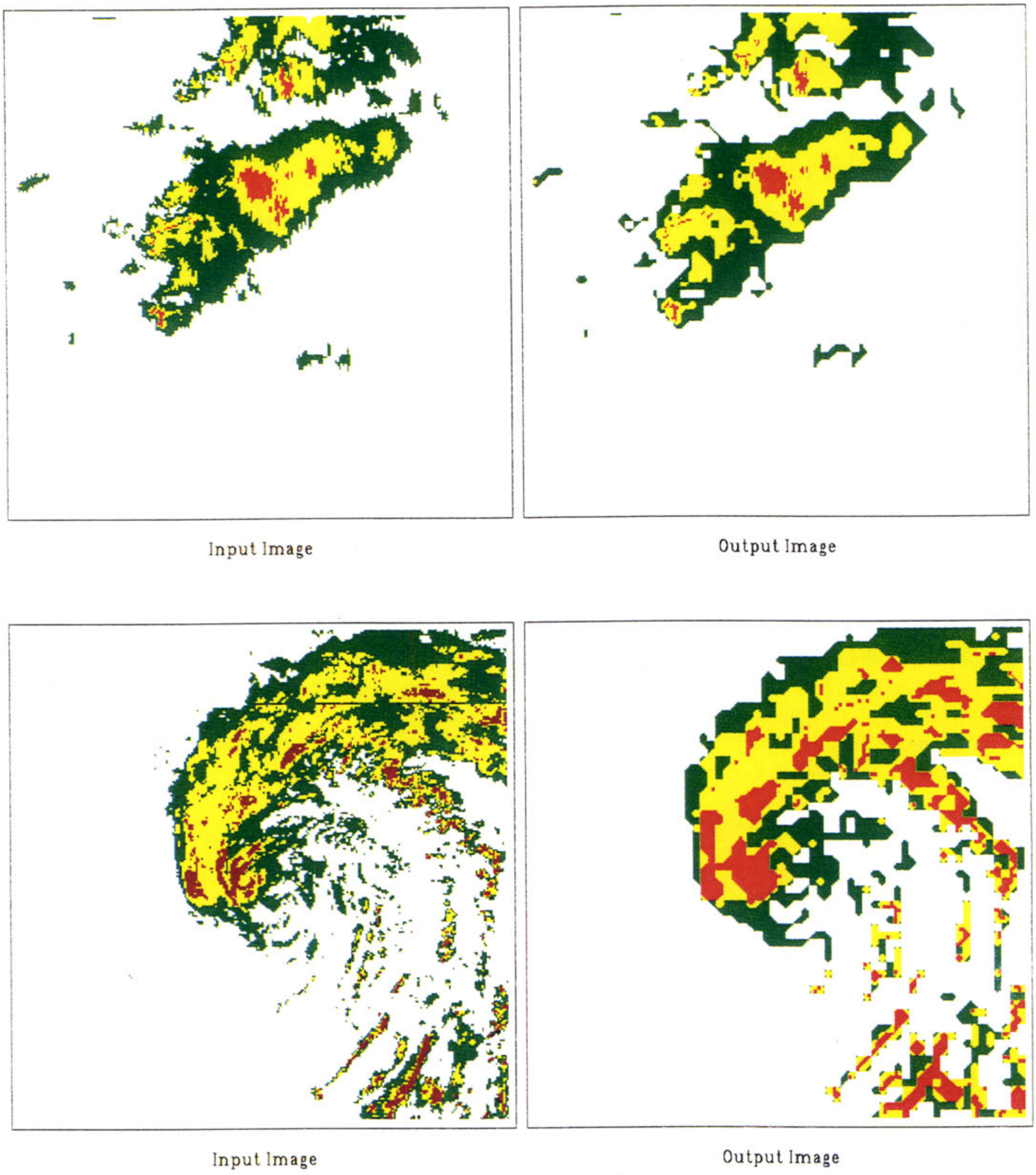


Figure 2-2a. Compression Performance at 3500 *Bits.*
Top: New York NY, 31 *July* 1992, 9 *PM.*
Bottom: Hurricane Allison, Tallahassee FL, 5 *June* 1995, 7 *PM.*

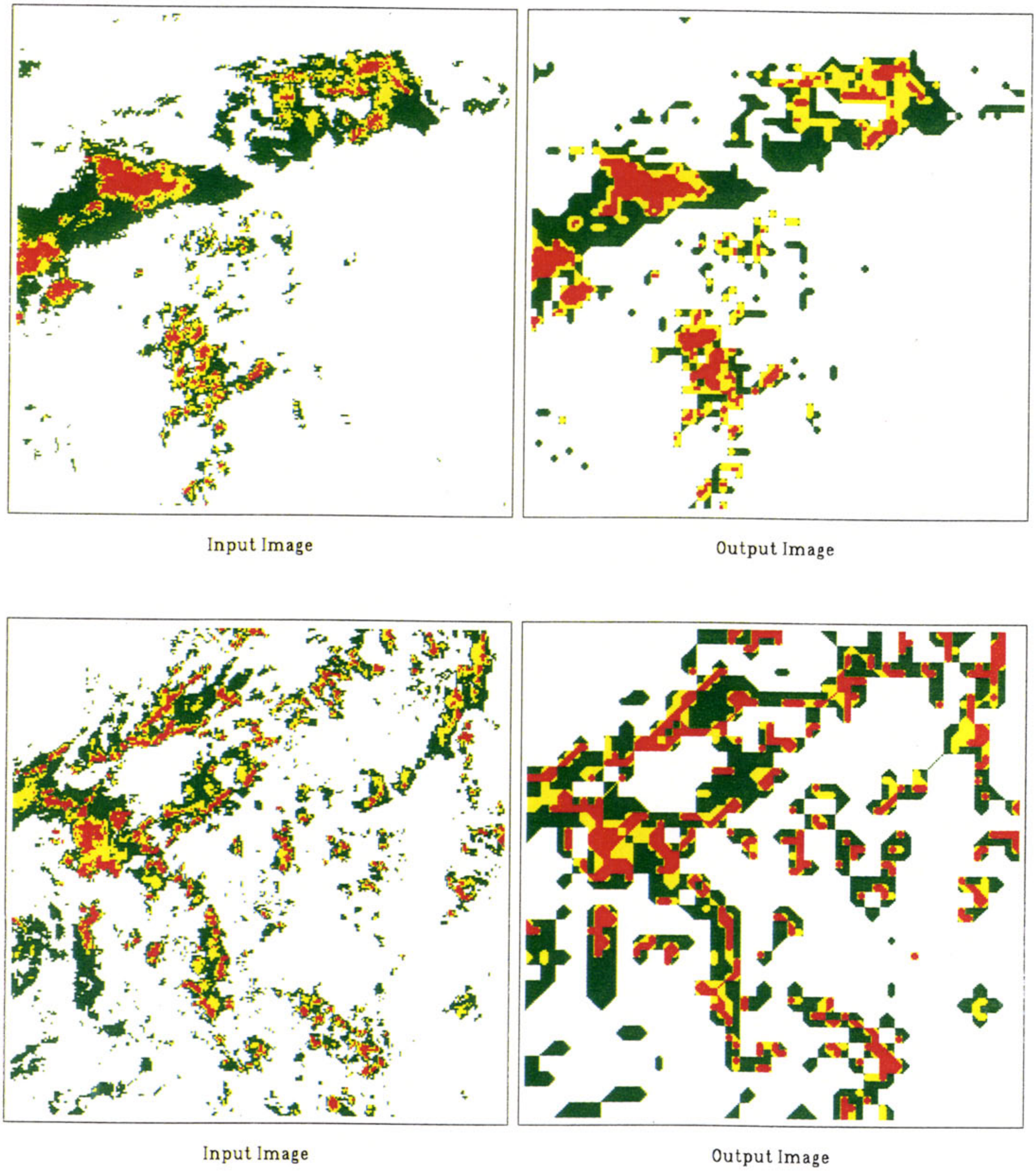


Figure 2–2b. Compression Performance at 3500 Bits.
Top: Mobile AL, 11 May 1995, 6 PM.
Bottom: Bridgeport CT, 11 May 1995, 6 PM.

Based on the results seen to date on a large number of weather images, it appears that the Weather-Huffman Algorithm is successful in providing accurate weather images to pilots over bit-limited datalinks.

2.6 PROGRAM SIZE AND TIMING MEASURES

The Weather-Huffman algorithm has been fully coded in the C language and tested on a Digital Equipment Corporation MicroVAX 3500 Color Graphics Workstation. (The 3500 is a minicomputer whose performance can be matched or exceeded by modern high-capability 32-bit microcomputers). Considerable effort has been made to optimize the coding for computational efficiency.

The compression routines start with a 256x256 pixel image array (read from a disk file) and produce a bit-string in memory. The decompression routines perform the inverse transformation. The WH algorithm requires about 2 megabytes of memory to execute its compression procedure and about 1 megabyte of memory to execute its decompression procedure. This includes all code, data areas, and C libraries used.

The algorithm was tested on a set of severe weather images derived from data supplied by weather services. Timing the procedures was done with the VAX system clock, accurate to a 10 millisecond quantization. All testing assumed a 3500 bit limit. The algorithm, as presently coded and run, requires about 3 seconds on average to encode these severe images. (Tests on more typical weather images required significantly less time per image). The decoding routines, on the other hand, require only 0.35 seconds processing per image.

It is clear that the processing requirements for the airborne datalink computer to perform decoding for the WH algorithm is quite reasonable:

onboard memory : 1 megabyte

onboard processing : 0.35 seconds per image

The speed-optimized WH algorithm requires more computer resources to do its encoding procedure, but, its requirements are still reasonable for modern ground-based computer systems.

3. HUFFMAN RUNLENGTH CODING

Coding theory tells us that if successive runlengths are independent random variables, the most efficient codes for representing them are entropy codes. Entropy codes are those whose codeword lengths are based on the frequency of occurrence of the symbol being encoded; more common runlengths will be assigned shorter codewords. In order for variable length codes to be uniquely decodable, their codewords must satisfy the prefix constraint: no codeword can be the prefix of any other codeword.

The most efficient entropy prefix code is the Huffman code. This code assignment is optimum in the sense that the total number of bits required to transmit a set of runlengths describing a given random image is minimum. Unfortunately, since the code assignment will be different for each image, the Huffman approach requires the overhead of transmitting a decoding table as part of each encoded message. This overhead can easily eliminate the optimality of the Huffman approach

The Weather-Huffman algorithm is a modification of the standard Huffman encoding scheme, with weather runs being the entities to be encoded. It introduces variations to reduce the number of bits required for encoding such images, particularly concentrating on methods of reducing the decoding table overhead. In order to help the reader understand the WH approach, this section presents the details of the general Huffman algorithm. The next section then provides details of the Weather-Huffman modifications.

3.1 IMAGE SCANNING PROCEDURES

A run on a weather image is a number of successive pixels having the same weather level. In order to determine a run, the term "successive" must be defined. The "normal" method of scanning a 2-dimensional picture is the raster scan, which proceeds left-to-right, row-by-row down the picture. The raster scan takes advantage of the continuity, or smooth edge, property of weather regions. That is, successive rows will tend to have similar run characteristics, which makes the Huffman coding job more efficient.

An alternative scanning method, the Hilbert scan, exploits a different property of weather regions, namely their contiguity, or tendency to form large 2-dimensional areas. The Hilbert scan is a 2-dimensional space-filling scan with the property that all pixels in a binary 2^n square are visited before that square is left. The Hilbert scan for a 32∗32 square can be seen in Figure 3-1. Note that, as just stated, each quadrant is completed before the next quadrant is entered.

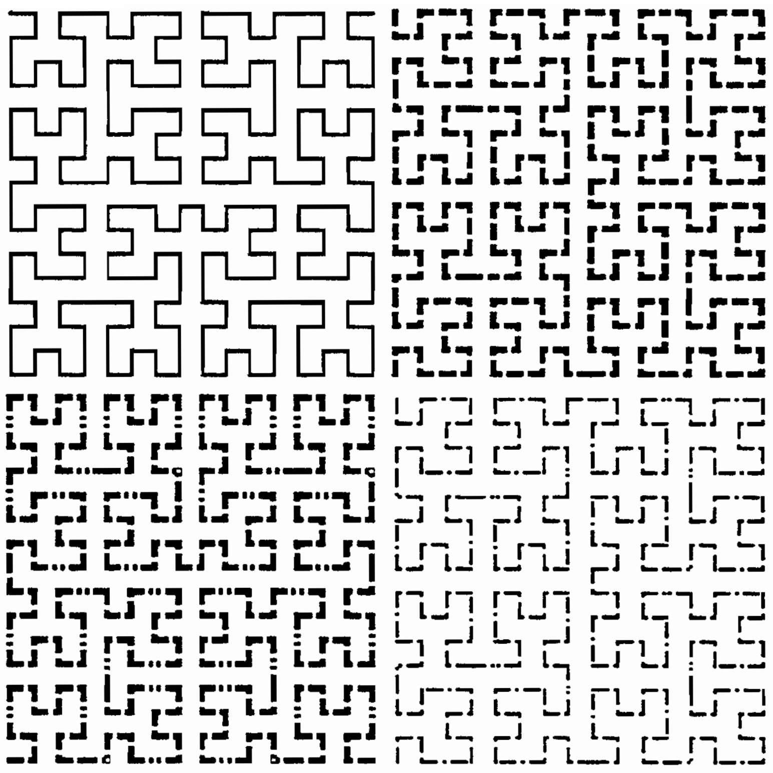


*Figure 3-1. The Hilbert scan for a 32*32 square. Each quadrant (different line type) is completed before another is entered.*

The advantage of a Hilbert scan for 2-dimensional objects such as weather regions is that a few long runs are produced while the scan meanders within the region, combined with several very short runs as the scan nicks the region's edges. This is illustrated by Figure 3-2, where the runs produced by the Hilbert scan of a typical weather region are given. Although the total number of runs produced is about the same as would result from a raster scan, the dichotomy of lengths is advantageous to a Huffman coding: since the large majority of runs are concentrated in a few (short) lengths, short Huffman codewords can be assigned to most runs. In addition, if filtering of the runs is required to meet a message bit limit, numerous single length runs are available to the filtering process.

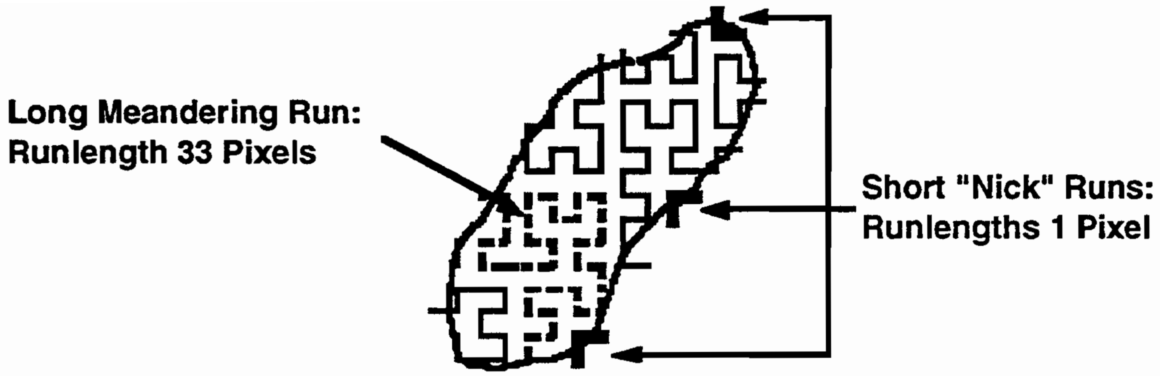


Figure 3-2. The Hilbert scan of a typical weather region. Several long meandering runs and several short "nick" runs are produced.

This assertion was tested by processing 11 sample NEXRAD weather images via both scanning methods. The ensemble average number of bits required by the Huffman runlength encoding were as follows:

Scanning Method	Input Image	4x4 Filtered Image
Raster	22546	2444
Hilbert	20179	2168

Thus, the Hilbert scan saved an average of 10% for the unedited images, and 11% for the filtered images. Considering the fixed decoding table overhead, these savings, particularly the latter one, are quite substantial.

The generation program for the Hilbert scan can be expressed in the following iterative form:

```
Hilbert(i)                              /* main routine */
{
hilbert1('e','s','w','n',i);
}

hilbert1(r,u,l,d,i)                     /* subroutine */
{
if(i>0)
  {
   hilbert1(u,r,d,l,(i-1));       /* iterative calls to itself */
   move(r);
   hilbert1(r,u,l,d,(i-1));
   move(u);
   hilbert1(r,u,l,d,(i-1));
   move(l);
   hilbert1(d,l,u,r,(i-1));
   }
}
```

where 'e','s','w','n' are the directions of the compass, the move function tells you to move in the direction of its argument, and

$i = \log_2(n)$ (i = 8 for a 256x256 image).

Note that the iterative calls to hilbert1 have various permutations of the original arguments; this is what causes the reflections and rotations of the curve components.

3.2 CODEWORD DETERMINATION

The Huffman algorithm takes as its input a non-increasing list of runlength probabilities. At each step, it combines the two lowest entries on the list into a single new entry. The two merged entries are removed, and the new entry is sorted into its proper location to create a new shorter ordered list. The process terminates when only a single entry remains. The record of pair mergings is then used to construct a binary tree which specifies the Huffman codewords in the manner now to be explained.

First, the lone entry in the final list is mapped into the root node of the tree. Thereafter, for every node that represents a pair merging, the two joined entries that created it are placed in the tree as that node's children, with the entry representing the fewer levels of joinings becoming the left child. (The selection of which node to be the left child is irrelevant to the Huffman algorithm; this rule is required for our decoding table tricks introduced below). On the other hand, every node that represents a single entry from the original starting list is made a leaf node labeled by the runlength it represents. Finally, when the tree is complete, the codeword for each runlength is read by traversing the tree from the root to that runlength's leaf, with a left branching labeled as a 0 and a right branching a 1.

This process is best understood through an example. Figure 3-3 presents from left to right the list generation procedure for an ensemble of six runlengths and their associated probabilities. In each column, the two entries joined are shown, as is the position of the combined entry in the subsequent list. For record keeping, each joined entry is "boxed"; if either or both of the paired entries already has a box, another one is added beyond the present maximum of the entries.

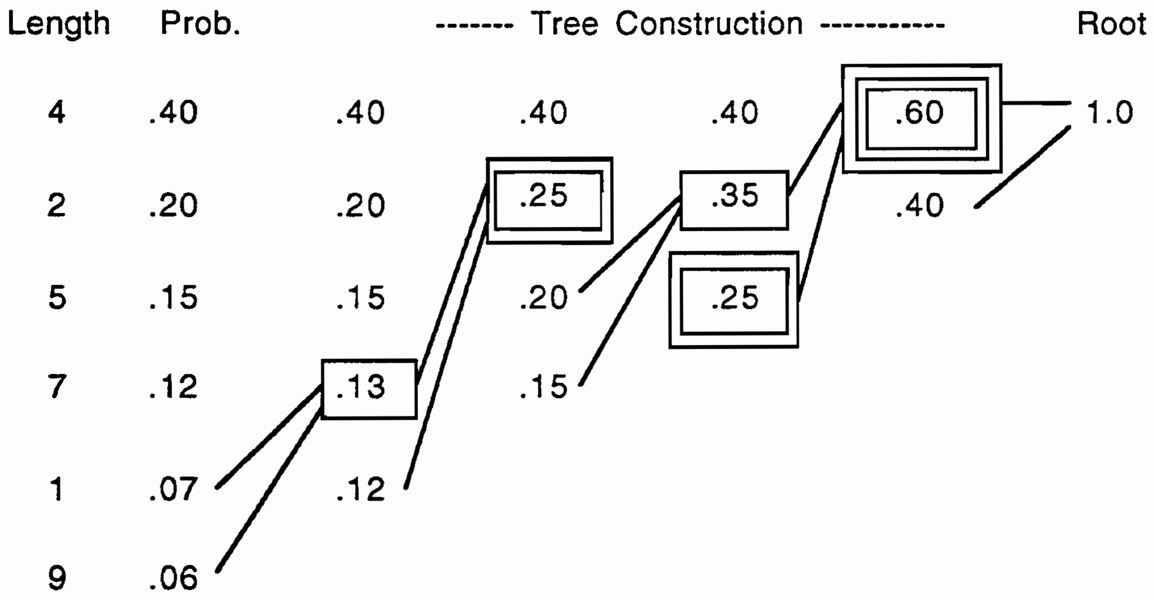
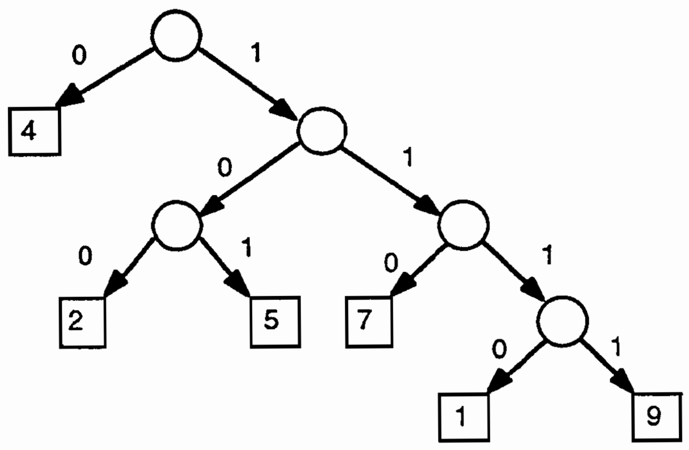


Figure 3-3. List generation procedure for an ensemble of six runlengths and their associated probabilities.

Next, the tree shown below is constructed from these lists in the manner described above. The root node has two children, one that represents a pairing (boxed) and the other an original list entry (unboxed). Thus, the latter is placed on the left and made a leaf node, the former is placed on the right and made an intermediate node. Then, this new intermediate node is given its two children. Since both of these entries represent pairings, they both become intermediate nodes; the one with the lower level of pairings (single box) is placed on the left. Continuing in this manner, the tree is completed as shown. Finally, the codewords shown in Figure 3-4 are read from the tree.



Runlength	Codeword
4	0
2	100
5	101
7	110
1	1110
9	1111

Figure 3-4. A complete binary tree specifying the Huffman codewords.

3.3 HUFFMAN DECODING TABLE REPRESENTATION

Whenever the Huffman codes are determined individually for each message, rather than being pre-determined on an ensemble average, the encoded message must explicitly contain the decoding table. Since this table represents an overhead on the message, its bit representation must be minimized. There are many possible methods for encoding this table. This section presents the fundamental concepts of table representation, while the next section describes the method actually employed in the Weather-Huffman approach.

One representation method that is often optimum, and the one generally presented in the literature, is to transmit the tree structure of the Huffman code. With this approach, both the tree shape and the list of runlengths in the tree are specified. The tree shape is encoded by scanning the tree left to right, level by level from root node down, sending a 0 to indicate a leaf node and a 1 for an intermediate node. Thus, for example, the tree shown above is encoded as 10111000100. Then the list of runlengths is presented in the order they are encountered in the tree; for this example 4,2,5,7,1,9.

Upon reception, the decoder constructs the tree from the shape specification, knowing that every intermediate node will have two children. Every time a leaf node is encountered, the next runlength is read from the runlength list and its code read by traversing the tree to its position. This approach to decoding table presentation requires at most 2n bits for the tree specification, where n is the number of runlengths, plus $n*m$ bits for the runlength list, where the longest runlength L satisfies $L \leq 2^m$.

A more straight-forward, though more bit-consuming, method of representing the decoding table is to list the n triplets of runlength, number of codeword bits, and codeword. Thus, the table for the above example would be the following:

number of runlengths n = 6

runlength	number of bits	codeword
1	4	1110
2	3	100
4	1	0
5	3	101
7	3	110
9	4	111

This manner of representation can be shortened considerably by leaving out the entire third column. This "trick" can be performed because it is possible, with our tree rules described above, to infer the codeword from its number of bits through use of the following algorithm:

1. Reorder the runlengths in increasing order of number of bits; if two or more have the same value, list them in the order in which they appear in the decoding table.

2. Initialize:

   b_1 = number of bits of first runlength

   $code_1 = b_1$ bits of 0

3. Compute each other codeword from the previous one:

   b_i = number of bits of runlength i

   $code_i = (code_{i-1} + 1)$ followed by $(b_i - b_{i-1})$ 0's

Applying this algorithm to the above example, the re-ordered list of runlengths would become 4,2,5,7,1,9. Then, for instance, the code for a runlength of 1 (i=5) is generated from the code for a runlength of 7 (i=4) as follows:

$code_5 = (code_4 + 1)$ followed by $(b_5 - b_4)$ 0's

$= (110 + 1)$ followed by $(4 - 3)$ 0's

$= 1110$

Proceeding in this manner, all the codes shown above would be produced.

A further modification to this method of presenting the table is to leave out the first column as well. To permit this approach, the number-of-bits entries must be listed for each runlength in order from 0 to n*, where n* is the largest value of the original n values. The above example with this approach thus becomes:

longest runlength n* = 9

(inferred runlength)	number of bits
(0)	null
(1)	4
(2)	3
(3)	null
(4)	1
(5)	3
(6)	null
(7)	3
(8)	null
(9)	4

Note that a special symbol, null, is used to indicate a non-coded runlength.

Finally, the longest runlength n^* also can be eliminated and need not be specified. That is because it is possible to know when all the codewords have been presented. In particular, let

$$w_i = \frac{1}{2^{bi}} \qquad b_i > 0$$

Then, the code has been completely specified after the jth entry if and only if:

$$\sum_{i=1}^{j} w_i = 1.$$

This last approach requires the fewest bits of any method, including the tree approach, if the set of coded runlengths is reasonably dense; that is, if n* is not much larger than n. Otherwise, the overhead of listing all the extraneous null entries can cause it to be less efficient than other approaches.

To overcome this efficiency problem, a hybrid approach is possible when many short runlengths, but only a few scattered longer runlengths, exist. The hybrid approach starts with a sequence of inferred-length number-of-bit entries from 0 to $n^{/}$, followed by the remaining entries given by a series of explicit runlength and number-of-bits pairings. The above example might then be presented as follows:

longest sequential runlength $n^{/} = 5$

(inferred runlength)	number of bits
(0)	null
(1)	4
(2)	3
(3)	null
(4)	1
(5)	3

runlength	number of bits
7	3
9	4

Note that the number of explicit runlengths need not be specified because of the end determination rule given above. A modification of this last method, described in Section 4 , is the approach used by the Weather-Huffman algorithm.

4. WEATHER-HUFFMAN CODING TABLE

The Huffman theory section presented general methods for creating the codewords to be used in encoding weather runlengths, as well as alternatives for transmitting to the decoder the tables specifying the chosen runlength/codeword pairings. This section develops the specialized techniques developed for the Weather-Huffman (WH) application that result in a major reduction in the number of bits needed to transmit the Huffman tables. In fact, it is these enhancements that make the WH approach feasible as a data compression algorithm for the Mode S data link; with the "normal" approach, the decoding tables alone could on occasion exceed the bit capacity of the Mode S message.

4.1 REPRESENTATION OF LONG RUNLENGTHS

Providing a separate codeword for every runlength existing for a given image level would generally be very inefficient because of the large decoding table overhead, even if the hybrid approach to table representation described in the theory section were adopted. Thus, the Weather-Huffman algorithm makes use of two special codewords to represent "other" length runs, where "other" is defined as any length beyond the longest one explicitly given a codeword. These words obviate the need to list numerous explicit runlengths and their codeword lengths.

The first special codeword (denoted by S1), when encountered, informs the decoder that the next set of bits will state the actual length of the run; in the simplest implementation these extra bits would just be the binary value of the length (i.e.: 000110 = 6). Since the longest possible runlength is 65536 (for a 256x256 image), 16 bits would be required for the extra field every time an "other" length occurs.

To reduce this overhead to only 6 bits, only lengths up to 63 are represented by the special codeword S1. Longer lengths are encoded by the use of a second special codeword (S2); this codeword tells the decoder that a run of greater than 63 pixels is present. The decoder then records the initial length 63 part of the run, doesn't change level, and reads the length of the remainder of the run in the normal manner. For example, a run of 149 would be encoded as follows:

(word S2) (word S2) (word S1) (binary 23)

63 + 63 + 23 = 149

if a run of 23 were an "other" length.

4.2 ENCODING "OTHER" LENGTH RUNS

Even using only 6 bits for specifying the length of "other" runs is inefficient when these runs are generally at the short end of the spectrum. Thus, the Weather-Huffman implementation considers the use of shorter binary fields on a level-by-level, image-by-image basis. It does this either by uniformly reducing the number of bits if the longest runlength is sufficiently short to make this possible, or by employing a "long-short" dichotomy for the set of runlengths otherwise.

With a "long-short" scheme, short runs use fewer specification bits than do long runs; a single extra bit preceding the binary field indicates which bitlength applies. The long-short approach makes sense when most of the "other" runs are at the short end of the spectrum, as then the loss encountered by having to include the extra bit is more than offset by the gain of having numerous short specifications.

The full set of options utilized by the Weather-Huffman algorithm, including both uniform bit reduction and long-short approaches, for the representation of "other" lengths for a given image level are as follows:

Option	Bit Field Lengths		Extra Bit Needed
0	6		no
1	6	3	yes
2	6	2	yes
3	5		no
4	5	2	yes
5	4		no
6	3		no
7	2		no

For example, with option 1, a long-short scheme, "other" runs that are 8 pixels or fewer greater than the longest explicitly specified runlength will require a 3-bit binary field (plus the single extra bit); longer runlengths will require 6 + 1 bits. Conversely, with option 5, a uniform reduction method, all runlengths are specified by 4 bits, but the longest "other" run must not exceed the longest specified length by more than 16 for this option to be applicable. The option in use for each level will, of course, have to be included in the transmitted decoding table.

Once the longest runlength R_{max} to be explicitly coded is determined (by the method discussed below), selecting the proper long-short option to employ is quite simple: try all eight, and choose the one requiring the fewest bits to encode all the "other" runs. In particular, the "other" bit requirement for option i is given by:

$$B_i = \sum_{j=R_{max}+1}^{R_{max}+S_i} R_j*(E_i+BS_i) \quad + \quad \sum_{j=R_{max}+S_i+1}^{R_{max}+L_i} R_j*(E_i+BL_i)$$

where R_j is the number of runs of length j, S_i and L_i are the number of runlengths accommodated by the short and long bit fields, respectively, for option i, BS_i and BL_i are the number of bits in the short and long fields, and E_i is 1 if the option uses a long-short dichotomy:

Option	BS_i	S_i	LS_i	L_i	E_i
1	—	0	6	64	0
2	3	8	6	64	1
3	2	4	6	64	1
4	—	-	5	32	0
5	2	3	5	32	1
6	—	0	4	16	0
7	—	0	3	8	0
8	—	0	2	4	0

Note that an option i is not a viable candidate if the longest runlength for the image level exceed $R_{max} + L_i$, which is the longest runlength that can be represented by the option.

4.3 ELIMINATION OF LONG CODEWORDS

Since for any given image level there are 66 potential runlengths to be assigned codewords (0-63, S1, S2), it is quite possible for the Huffman algorithm to produce codewords that exceeds the desired 7 bit limitation. (The theoretical limit is 65 bits, but a weather image does not contain enough pixels to ever approach that value). In these cases, the codewords must be modified to satisfy the imposed limit. In particular, some of the shorter codewords must be lengthened to permit the outlier codewords to be brought down within the 7 bit limit.

The algorithm adopted is, when a choice exists, to start by modifying codewords of length 6, then proceeding if necessary to codewords of length 5, etc. until all the longer codewords have been brought down to length 7. This approach, of not lengthening any codeword for common runlengths, minimizes the number of bits required for image encoding.

When the longest codeword is restricted to have at most 7 bits, there are $2^7 = 128$ bit sequences in the code space. Each codeword of length m, because of the prefix nature of the Huffman code, exhausts $2^{(7-m)}$ of these sequences. For example, the codeword 01001 removes the sequences 0100100, 0100101, 0100110, and 0100111 from possible assignment, while any 7 bit codeword removes only its own sequence from the available list.

This realization has led to the following algorithm for the reassignment of Huffman codes when the longest codeword produced by the Huffman algorithm exceeds the 7 bit limit, where the codewords are ordered from the shortest to the longest:

```
1.  Initialize $i = 0$, and let $n$ = number of codewords.
2.  Initialize the number of sequences used to $U = 0$.
3.  Increment i by 1; if $i > n$, quit.
4.  Increase U by $2^{(7-bi)}$, where $b_i$ = number of bits currently assigned to codeword i.
5.  If $(128 - U) = (n-i)$, set $b_j = 7$ for all $j = i+1, \dots, n$; quit.
6.  Otherwise, if $(128 - U) < (n-i)$, repeat the following actions:
    a. increment $b_i$ by 1
    b. adjust $U = U - 2^{(7-bi+1)} + 2^{(7-bi)}$
       until $(128 - U) \geq (n-i)$, then return to step 3.
7.  Otherwise, if $(128 - U) > (n-i)$, repeat the following actions:
    a. compute $\Delta = 2^{(7-bi+1)} - 2^{(7-bi)}$
    b. if $(128 - U - \Delta) \geq (n-1)$,
       decrement $b_i$ by 1 and adjust $U = U + \Delta$
       until the test in b fails, then return to step 3.
```

Step 5 recognizes the fact that if the number of remaining sequences and remaining runlengths are equal, each remaining runlength can be assigned a 7-bit codeword. Step 6 handles the case that fewer sequences than runlengths remain, so that more available sequences have to be generated by increasing the length of the codeword assigned to the current runlength. Finally, step 7 recognizes the case when previous applications of step 6 have freed up more sequences than were required, so that the codeword for runlength i may be able to be decreased to absorb some of the extra sequences.

An example of the application of this algorithm is given by the following before and after set of codeword length assignments:

Runlength Order	Huffman Length	Adjusted Length
1	1	1
2	2	2
3	3	3
4	4	5
5	6	5
6	6	7
7	6	7
8	7	7
9	8	7
10	9	7
11	10	7
12	11	7
13	11	7

Note that the 4th runlength had to have its codeword length increased to accommodate the codewords over the limit, but that by doing so the number of sequences freed up also allowed the 5th codeword to be shortened.

4.4 WEATHER-HUFFMAN (WH) TABLE ENCODING

The actual Huffman table encoding employed by WH can now be described. The general method utilized is a variant on the implicit runlength scheme presented in the previous section, in which a list of codeword lengths (with null meaning no codeword because that runlength never occurred) is presented. The order of the implicit runlengths is S1, S2, 0, 1, 2, 3, ... , n, where n is the longest encoded runlength, and thus (n+1) is the first "other" length. Since the null length is expected to be the most common encoded value, the runlength codes are encoded in two parts: a single bit indicating null (0) or not null (1), and a field for the non-null cases which presents the actual length of the Huffman code for that runlength.

The simplest way to present the latter field would be to set a limit on the acceptable length of a Huffman codeword, and then use enough bits in the field to cover all possible values of that maximum length. For example, if the limit is set at 7, 3 bits would be required for the field. Note that limiting the codeword length required the modification to the codeword generating algorithm presented above to cover the cases when the algorithm produces longer codewords.

Rather than use a constant length field, the Weather-Huffman algorithm saves bits by computing, for each runlength, the set of codeword lengths that could possibly still exist. This set can be determined from the previously listed codeword lengths if the longest codeword for the level is known and the rule given earlier,

$$\sum_{i=1}^{j} w_i = 1$$

where $w_i = \frac{1}{2^{bi}}$ $b_i > 0$ is used. The specific algorithm for encoding the codeword lengths is provided below.

Putting together all of the above pieces, the Huffman table for each image level is presented as the following set of fields:

Field	No. of Bits
Table present (i.e., non-default)	2
Longest codeword used	3
S1 encoded?	1
if yes, S1 codeword length	variable
S2 encoded?	1
if yes, S2 codeword length	variable
Length 0 encoded?	1
if yes, 0 codeword length	variable
Length 1 encoded?	1
if yes, 1 codeword length	variable
-------	-----
Length n encoded?	1
if yes, n codeword length	variable
if S1 encoded, option used	3

As described below, three default codeword sets are available for each image level. The first two bits above either specify one of these default sets, in which case the table specification terminates, or state that the actual table description is to follow. The codeword-reading process is known to be over when the completion rule above is satisfied, that is, when the length n has been read such that:

$$\sum_{i=S1}^{n} \frac{1}{2^{bi}} = 1$$

Also, for image levels 0 or mapmax, no runs of length 0 can exist; thus that length is not encoded.

4.5 ENCODING CODEWORD LENGTHS

The concept of counting the number of "used" sequences presented earlier can also be applied advantageously to the problem of determining how many bits are required to encode the lengths of codewords in the decoding table. If the length of the longest codeword used for a given image level is stated to be L_{max}, then the length L_1 of the first codeword presented in the table is restricted to the range:

$L_1 = 0$ if $L_{max} = 0$

$1 \leq L_1 \leq L_{max}$ if $L_{max} > 0$

Note that if a 0-length codeword is used, it must be the only codeword for that level since it would use up all 128 available sequences; thus knowing another length L_{max} exists precludes the 0 length.

After each codeword length is specified in the decoding table, the number of available unused sequences decreases. Thus the range of possible codeword lengths also decreases as one-by-one the shorter lengths drop out as possibilities. In particular, by the formulas of the last section, length L is still an option only if:

$$2^{(7-L)} \leq (128 - U)$$

As the set of codeword length options decrease, of course, the number of bits needed in the decoding table to specify the actual length will also decrease (in fact, if only 1 option remains, no bits at all are needed!). For example, consider the codeword length assignments in the following list of runlengths when the longest codeword is stated to be 4:

Sequences Used So Far	Runlength	Code Length Options	Bits Needed to Specify	Actual Code Length Used
0	1	1-4	2	3
16	2	1-4	2	2
48	3	1-4	2	2
80	4	2-4	2	2
112	5	3-4	1	4
120	6	4	0	4

where the sequences used so far entry is increased at each step according to the length used in the last column of the previous row. The total of the bits needed to specify column represents a savings of 50% over the 3 bits per codeword length that would be required by the "normal" approach when a 7-bit codeword limitation is enforced.

The bit requirement for specifying the codeword length can be reduced even further by employing a Huffman approach (Huffman encoding the entries of the Huffman decoding table!). For example, if the code length options available at a given point are 5, 6, and 7, it takes 2 bits for the specification by the above logic. But it is possible to shorten the specification for length 7 by choosing the following set of decoding table entries:

5:	11
6:	10
7:	0

In particular, whenever the number of options is not fully 2^m, some of the options can be assigned m-1 bits for their specification. The WH algorithm in such cases assigns the 1-bit shorter m-1 bit specifications starting with length L_{max}, then with L_{max}-1, etc. until only m bits specifications are available.

Complete details of the encoding and decoding of the decoding table are presented in Sections 8 and 9, respectively.

4.6 DETERMINING THE "OTHER" LENGTH BREAKPOINT

The final piece of the decoding table puzzle is how to determine the longest runlength to explicitly give its own codeword, with the remaining longer runlengths relegated to the "other" category. In a pure Huffman coded system, the "optimum" codewords would be determined as outlined in the theory section, with each length assigned its own codeword. However, the requirement of transmitting the decoding table in the message changes the problem from minimizing the total codeword bits to instead:

Problem: Minimize T = { (Table bits) + (Codeword bits) }

With the table architecture described earlier, the only variable that affects the minimization problem is L_{max}, the longest explicitly encoded runlength. Since providing each runlength with its own codeword will always minimize the number of bits needed to encode the weather runs, the tradeoff when changing L_{max} can be expressed by:

as L_{max} is raised, the bits for encoding runs decreases, but

as L_{max} is raised, the decoding table size increases.

The optimum value of L_{max} is the value that minimizes the sum of encoding bits and table bits.

Because of all the "tricks" used to represent the decoding table, there is no simple calculation that will produce the optimum value of L_{max}. If the optimum is truly sought, all values of L_{max} from 0 to the longest runlength must be processed through both the Huffman codeword assignment and the table generation algorithm to learn the number of bits required for that value; the smallest total found then determines L_{max}.

For a given value of L_{max}, the computation of T proceeds as follows:

1. Let $R_{65} = \sum_{i=n+1}^{63} R_i$ be the number of "other" runs.
2. Determine the Huffman codewords C_i for the lengths i = S1, S2, 0, 1, ..., n using the frequencies R_i.

3. Let $T_1 = \sum_i R_i C_i$

4. Determine the "other" option number that minimizes the bits T_2 required to state the explicit second codewords for the "other" runlengths (if $R_{65} = 0$, $T_1 = 0$).

5. Determine T_3, the number of bits required to encode the Huffman table for this value of n.

6. $T = T_1 + T_2 + T_3$.

If this calculation is performed for all values of L_{max}, the minimum value T_{min} will be determined.

Clearly, this process is very time consuming, and a shortcut that eliminates many of the L_{max} values from consideration would be highly desirable. The method adopted to attain this goal is to not test any value i if its number of runs R_i satisfies either

(a) $R_i < R_{i+1}$, or

(b) $R_i \leq 1$

If the first condition applies, the next runlength value will probably be a better stopping point than this one; if the second condition applies, this runlength value will probably not beat the current optimum stopping point. The length R_{max} is always tested even if it has only 1 run, as the elimination of the need for "other" runs is a significant savings of bits.

4.7 DEFAULT CODE ENSEMBLES

The above "tricks" for encoding the Huffman decoding table have severely reduced the bit requirements of these tables. In fact, tests on sample images show that often as few as 20 bits are needed for a table for a particular weather level.

Alternatively, a default Huffman codeword set can be employed for the level. The predefined codewords will generally not be optimal for the level's actual runlength statistics, but the elimination of the need for the decoding table can result in fewer total encoded bits. The Weather-Huffman algorithm has defined 3 default codeword sets for each weather level, matched to statistics found on a set of test images. These default sets, which vary by level, are presented in Appendix A. Since each default set has a defined value of L_{max}, the approach given above can be followed to determine for each set the number of bits $T_{def,i}$ that are required if it is selected for the encoding. The only differences are that step 2 is avoided (the default codewords are used) and in step 5, $T_3 = 5$ (2 bits for the default number and 3 bits for the option selection). The default set with the smallest result is the winner, with T_{def} being its calculated value.

Finally, the proper encoding scheme for each level is found by comparing the table-driven value of T_{min} to the optimum default-driven value of T_{def}. If the latter if smaller, the encoder simply lists the number, 0, 1, or 2, of the optimum default option; if the former is smaller, the encoder lists the value 3 followed by the decoding table.

5. WEATHER IMAGE SIMPLIFICATION

If the Huffman runlength encoding of the exact input weather image requires more than the allowable number of bits, some form of detail reduction must be performed on the image to lower the encoding load. Two types of image simplification are used as part of the Weather-Huffman approach: runlength filtering and resolution reduction.

5.1 RUNLENGTH FILTERING

Isolated weather pixels add substantially to the Huffman runlength encoding requirement. For example, the following sequence of pixels — 221222 — produces code for 3 runs and 2 level changes. By changing the isolated pixel to a 2, the encoding reduces to that for a single run. Altering the weather level of such isolated pixels is in some cases deemed acceptable distortion for the large benefit in data compression that is achieved.

For pilot safety considerations, no reduction in level is permitted for significant weather pixels (level 2 and above) in the filtering operation; only level 1 pixels may be reduced. Any weather level pixel, however, may be increased a single increment by filtering, and in addition level 0 pixels may be set to level 2. Thus, examples of the application and rejection of filtering operations are:

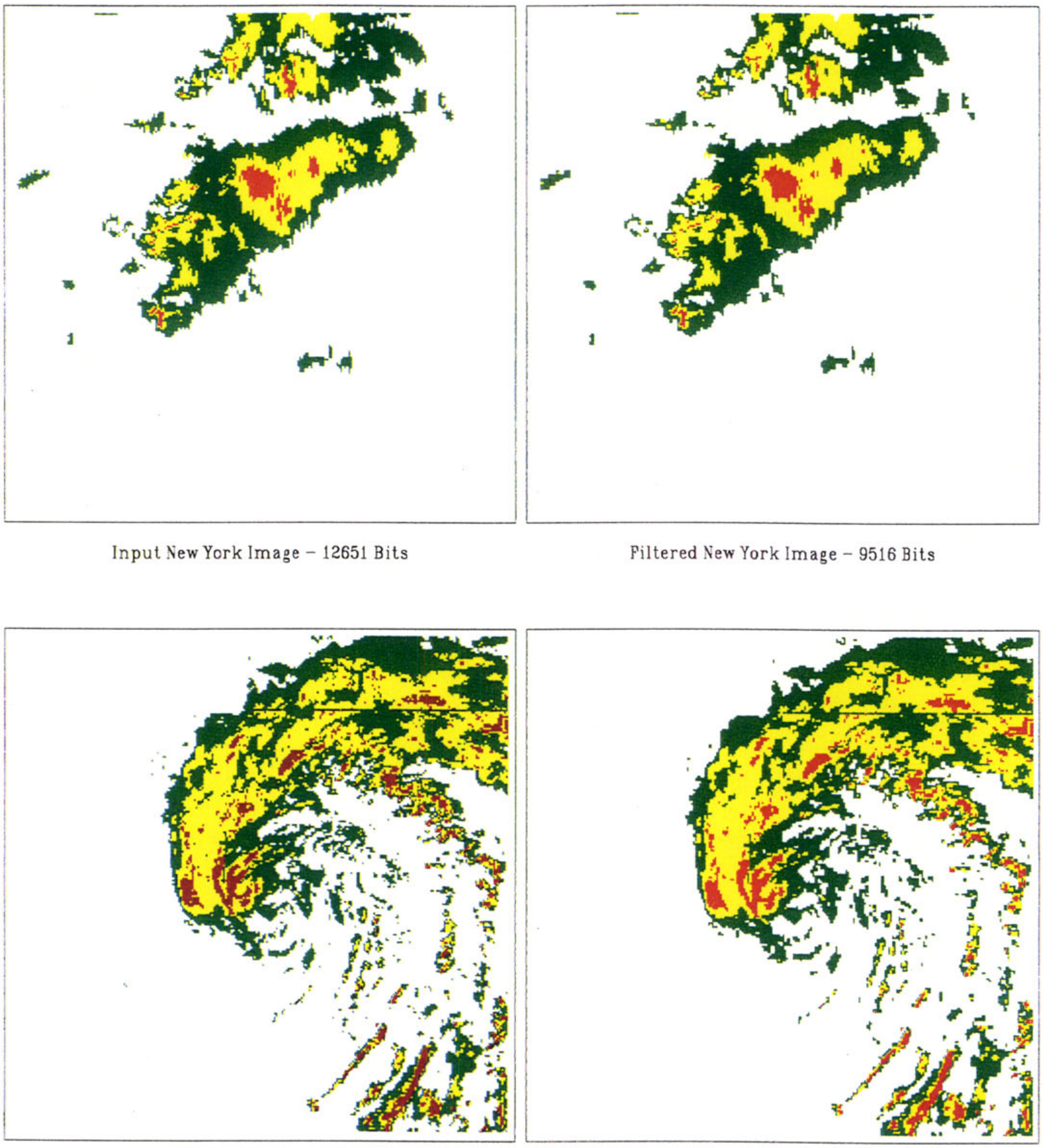
(a)	11011	—>	11111	0 can increase
(b)	33433	—>	33433	4 can't be reduced
(c)	12033	—>	12233	0 can be raised to 2
(d)	11233	—>	11233	no gain by changing the 2 to 3
(e)	101010	—>	111111	prefer raising 0 to lowering 1
(f)	33133	—>	33233	1 can only be raised to 2

Example (d) recognizes the fact that a sequence of 11333 would still require a 0 length run specification for level 2, and hence no gain would be achieved by making the change. Similarly, example (f) saves the two 0 length runs that would have been required for level 2, one on the descent and the other on the subsequent ascent.

Figure 5-1 presents before and after views of the filtering operation for two typical weather images. Note that the significant reduction in bit requirement for the Huffman encoding is achieved at a very minor distortion in the weather regions on the image.

5.2 RESOLUTION REDUCTION

Another, more severe, way to decrease the Huffman runlength encoding bit requirements of a weather image is to reduce the resolution of the image from 256x256 pixels to 128x128 pixels, with each new pixel representing 4 of the old ones (a 2x2 square). This change produces a blockier image, but maintains reasonable fidelity. If this reduction is not sufficient to satisfy the bit limit, further reduction to 64x64 pixels, or even 32x32 pixels, is utilized in the Weather-Huffman algorithm. To generalize the notation to apply to any size image, not just 256x256 ones, the various levels of resolution reduction are denoted by the size of the encoded "superpixel". Thus the original image is labeled 1x1, the first reduction in resolution becomes the 2x2 image, followed if needed by the 4x4 and 8x8 images.



Input New York Image – 12651 Bits

Filtered New York Image – 9516 Bits

Input Allison Image – 29582 Bits

Filtered Allison Image – 22530 Bits

Figure 5–1. Effect of Filtering Weather Images. Small Change in Fidelity, Significant Reduction in Bit Requirement.

The approach chosen to produce the various size superpixels emphasizes higher levels of weather, in that the new pixel setting is influenced most by the highest weather level in the superpixel square. The level chosen also depends upon the weather in the neighboring superpixels, in that no region of weather of level 3 or above, even if only 1 pixel in size, will ever be permitted to be lost via resolution reduction.

To determine the proper weather level setting of a superpixel, the number of its constituent pixels at or above each level is counted (that is, for example, the level 2 count includes all pixels of levels 2, 3, 4, 5, and 6). Then, highest level to lowest, each level count is compared to its two threshold values. If the level count satisfies the "mandatory" threshold, the superpixel components by themselves are sufficient to justify setting the superpixel to that weather level.

Otherwise, if the level count satisfies the "maybe" threshold, the superpixel components by themselves are insufficient to warrant that weather level setting. However, the count is enough that the weather region must be represented on the output image. Thus, if not enough of the superpixel's neighbors will be set to that weather level to adequately represent the region, this superpixel must assume the responsibility by being set to that level.

To determine if such a case exists, the superpixel's 8 horizontal, vertical, and diagonal neighbors are checked to determine how many M of them have already, or will when checked, assume responsibility for representing the weather region. M is determined by comparing the neighbor setting (if already processed) or the neighbor counts (if not yet processed) of each of the 8 neighbors. For each neighbor of the former class, M is incremented if its setting is the level in question or higher, while for each neighbor of the latter class, M is incremented if its count for this level is at least the "mandatory" value. Finally, if the resulting value of M is less than 3, the superpixel is set to the level being tested. For this operation, a left-to-right, top-to-bottom row-by-row scan is assumed, so for a superpixel, its W, NW, N, and NE neighbors will have already been processed, while its E, SE, S, and SW neighbors will not have been processed as yet.

As an example, assume the "mandatory" and "maybe" thresholds for level 2 for a 4x4 image are 5 and 2, respectively. This means that any 4x4 superpixel with 5 or more level 2 pixels out of 16 is automatically set to level 2. Otherwise, any superpixel with a level 2 count between 2 and 4 is set to level 2 only when the neighboring superpixels cannot adequately represent the weather region. Applying the above rules, the following superpixel settings are generated from the indicated level 2 counts as seen in Figure 5-2.

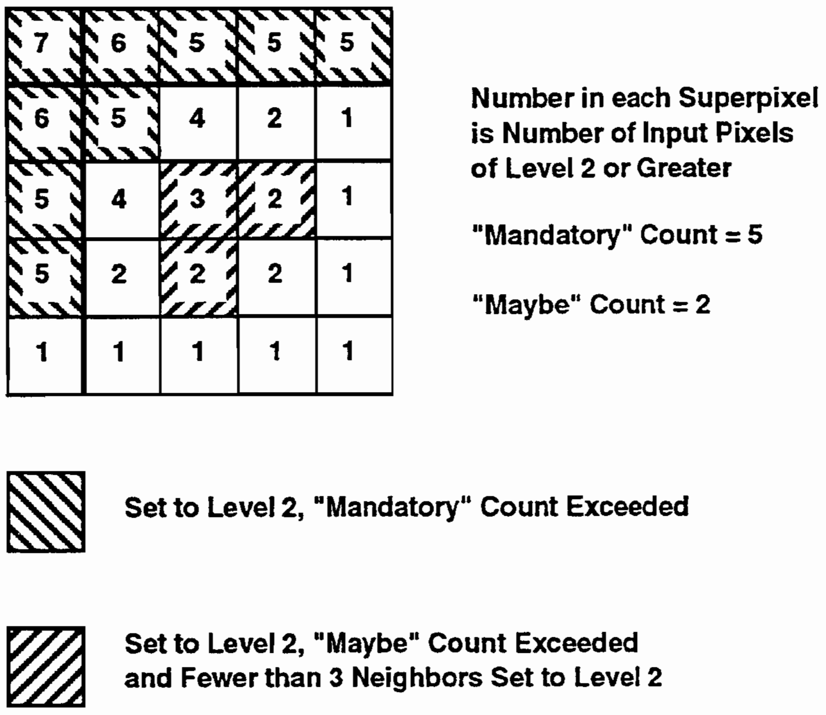

7 6 5 5 5
6 5 4 2 1
5 4 3 2 1
5 2 2 2 1
1 1 1 1 1
Number in each Superpixel is Number of Input Pixels of Level 2 or Greater
"Mandatory" Count = 5
"Maybe" Count = 2
Set to Level 2, "Mandatory" Count Exceeded
Set to Level 2, "Maybe" Count Exceeded and Fewer than 3 Neighbors Set to Level 2

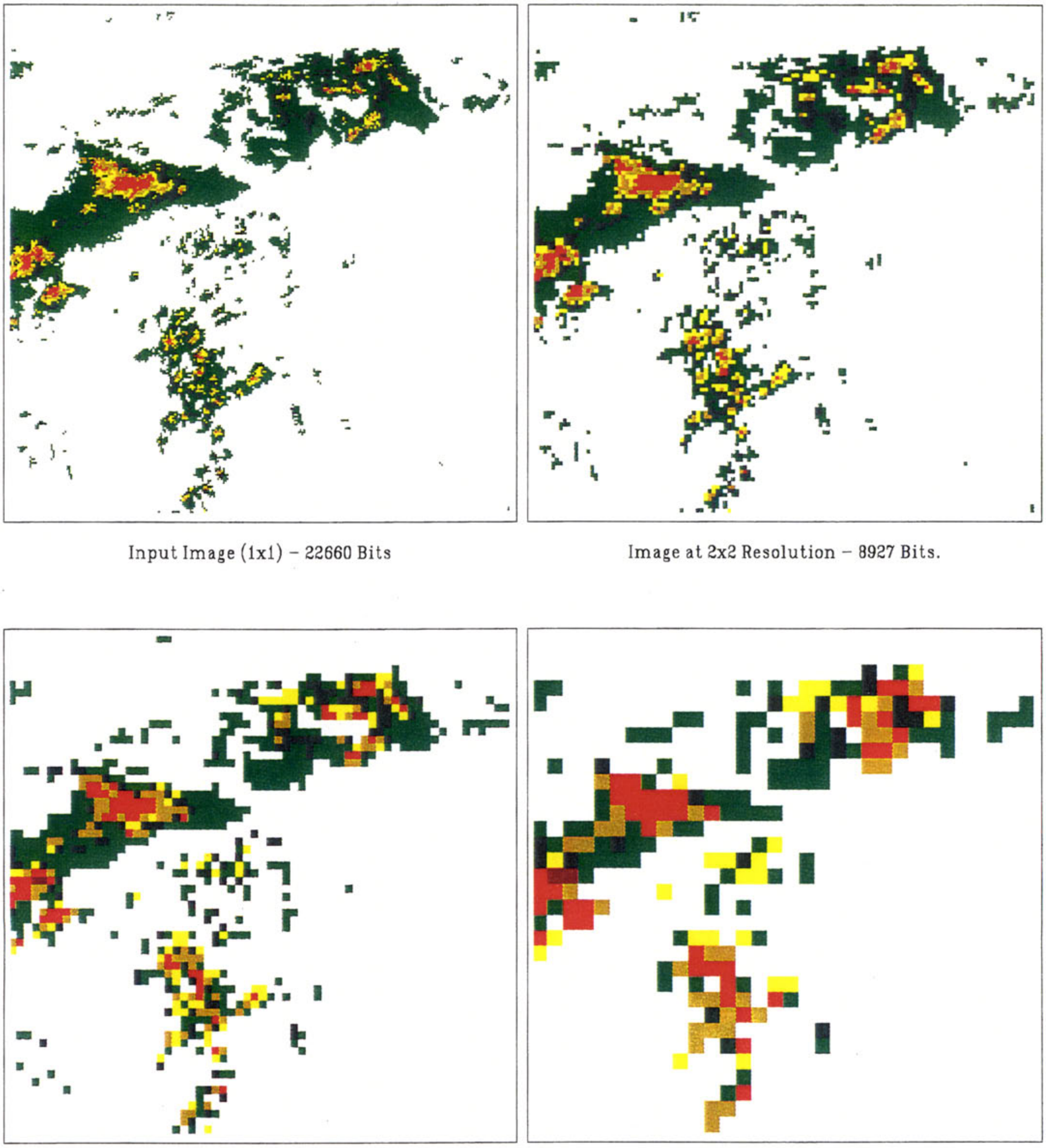
Figure 5-2. Superpixel settings generated from the indicated level 2 counts.

The shaded area is a reasonable representation of the extent of the level 2 weather region. Without the "maybe" superpixels, the region would appear much smaller than the reality of the weather.

The "maybe" count for levels 3 and above, for any degree of resolution reduction, is always 1. This guarantees that no region of intense weather can ever be lost, even if only 1 pixel in extent. Appendix B contains the full set of default parameter settings.

Figure 5-3 presents the changes in image that result from 2x2, 4x4, and 8x8 resolution reductions for a typical weather image. Note that higher weather levels tend to bloom in size, while the effect for levels 1 and 2 is to provide an average representation of the true contours. The blooming of higher levels is desirable for pilot safety, as any reduction in size could lead to flying into the edge of a severe storm.

It is clear from this figure that substantial bit reduction occurs with the Huffman runlength encoding as the degree of resolution reduction increases.



Input Image (1x1) - 22660 Bits

Image at 2x2 Resolution - 8927 Bits.

Image at 4x4 Resolution - 3245 Bits.

Image at 8x8 Resolution - 1182 Bits

Figure 5-3. Mobile Image Encoded at Various Levels of Resolution Reduction.

6. SMOOTHING PROCESS

Whenever the Weather-Huffman encoding requires resolution reduction of the input image to meet the link bit limitation, the decoder must expand the received image back into its full resolution. Without any other information about the original image to go by, the decoder is reduced to replicating each received superpixel into 4 quadrants of same-level pixels at each step of the regeneration. Thus, if a received superpixel were of size 4x4, the decoder would generate 4 2x2 superpixels, and then 16 1x1 actual pixels, all of the same weather level. The result of this operation is the generation of an output image with weather regions that are larger and blockier than those on the input image.

The Smoothing Process attempts to mitigate this effect by using knowledge of weather region shapes to reduce in level some of the pixels at each regeneration step. For example, weather regions do not often have sharp corners. The Smoothing Process, for reasons of pilot safety, is not allowed to reduce the extent of weather regions. It is only permitted to reshape the edges of regions to drive them closer to expected reality.

6.1 SMOOTHING PROCESS RULES

The rules of the Smoother Process are quite simple. When a superpixel is reproduced, 4 corner quadrant pixels are generated. A quadrant may be smoothed down 1 level only when:

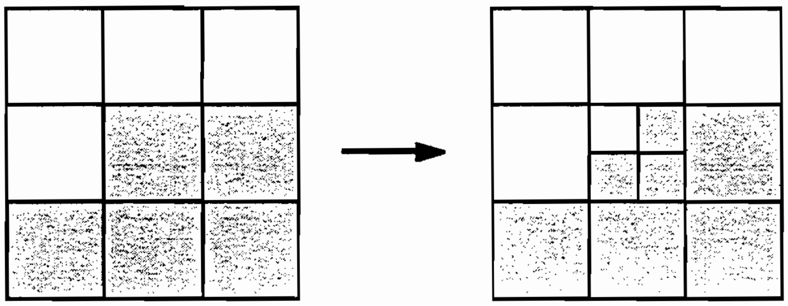
1. All 3 neighboring superpixels of its parent superpixel are of lower weather level than it is, and
2. Neither the horizontal nor vertical neighbor quadrant in its superpixel meets rule 1.

Figure 6-1 illustrated these rules pictorially, while Figure 6-2 presents sample applications of these rules. The effect of rule 1 is to smooth the edges of weather regions by eliminating sharp corners. Rule 2, on the other hand, precludes the losing of the full extent of the weather region that would occur by smoothing away two neighboring regions. The bottom two examples of Figure 6-2 show how rule 2 prevents compromising the size of the weather region.

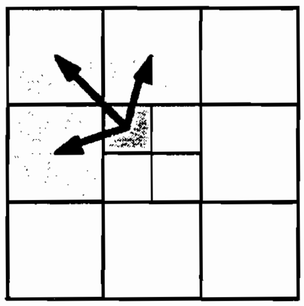
A simple cartoon example of smoothing is presented in Figure 6-3, while a full-scale example of the application of smoothing when a 4x4 superpixel image is expanded first to 2x2 superpixels, and then to 1x1 output pixels, is provided by Figure 6-4. It should be clear that the result is more indicative of the actual underlying weather region than is the unsmoothed post-Huffman picture.

6.2 SMOOTHING PROCESS IMPLEMENTATION

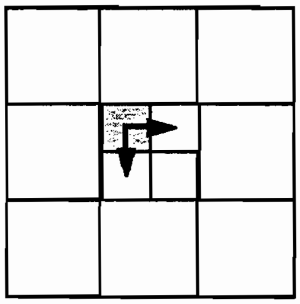
Implementation of the Smoothing Process is simplified via use of the four pre-computed neighbor value lookup tables UL(score), UR(score), LL(score), and LR(score), one for each of the four quadrants of the superpixel. The tables are generated by assigning the following values to the 8 neighbor positions of a superpixel (see Figure 6-1).



In Order to Smooth a Corner Quadrant



a) All 3 Neighboring Blocks Must be Lower Level, and



b) Neither Neighbor Quadrant May be Smoothable

Figure 6-1. Rules of smoothing process algorithm.

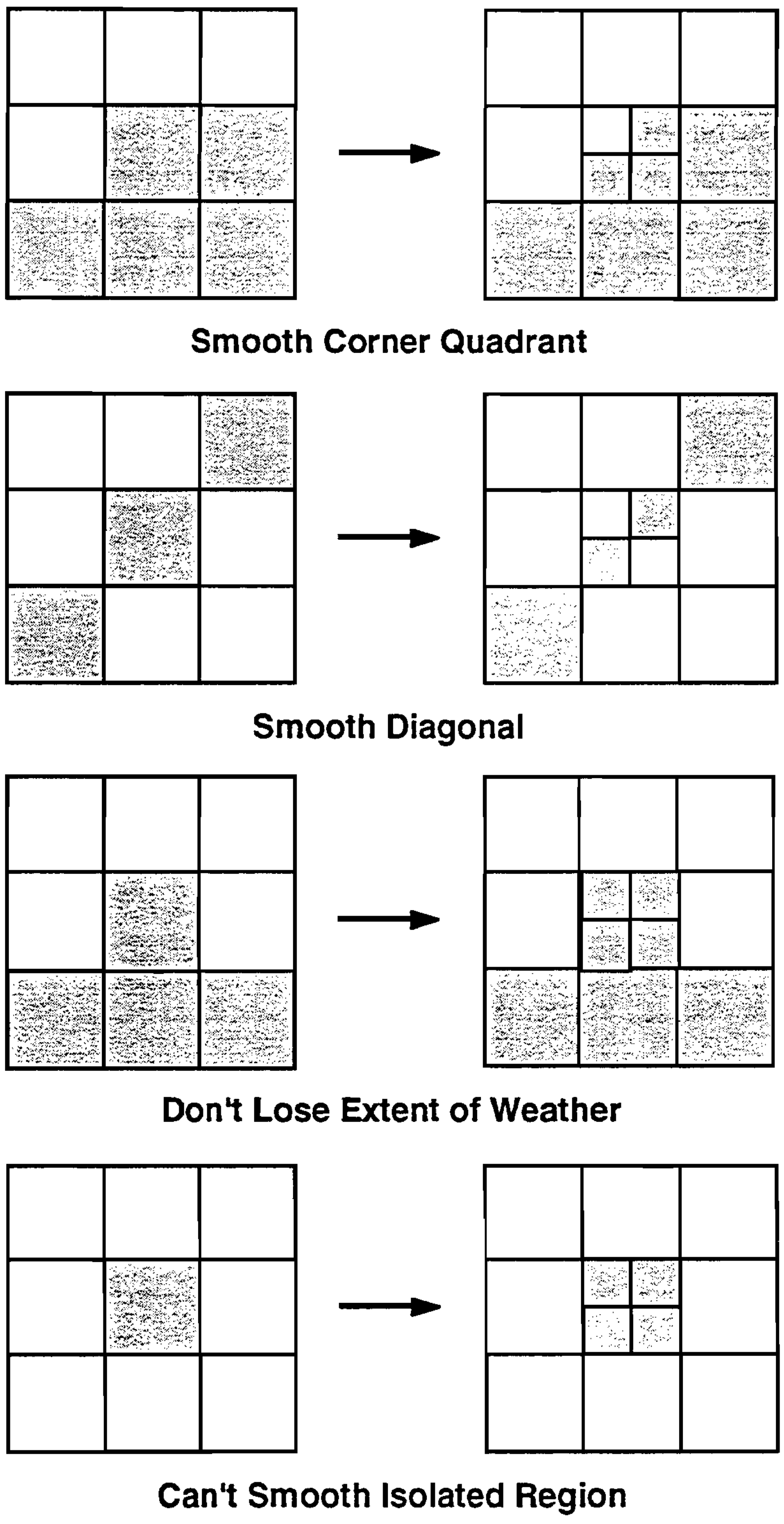
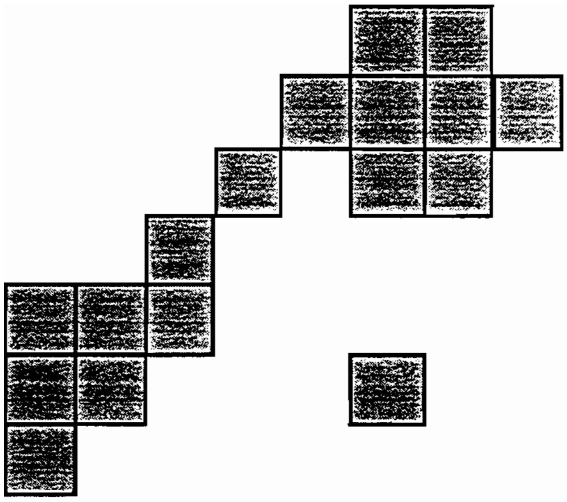
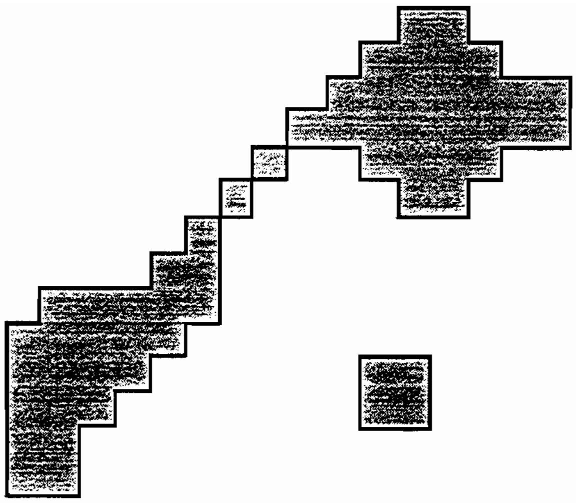


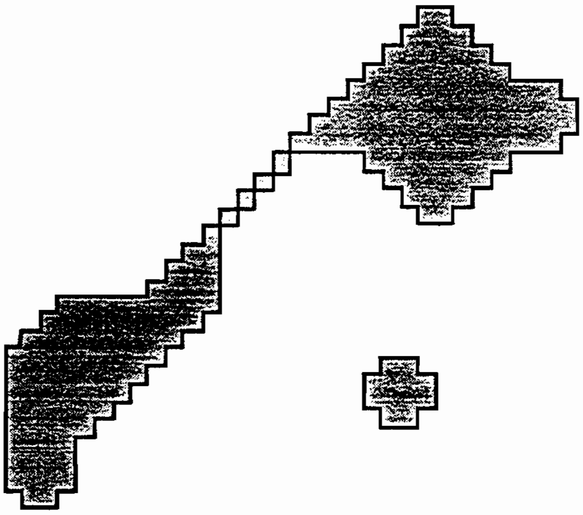
Figure 6-2. Examples of application of smoothing process rules.



(a) Original 4x4 Decoding.



(b) After Smoothing to 2x2.



(c) Final Smoothing to 1x1.

Figure 6-3. Example of smoothing process algorithm applied to sample weather region.

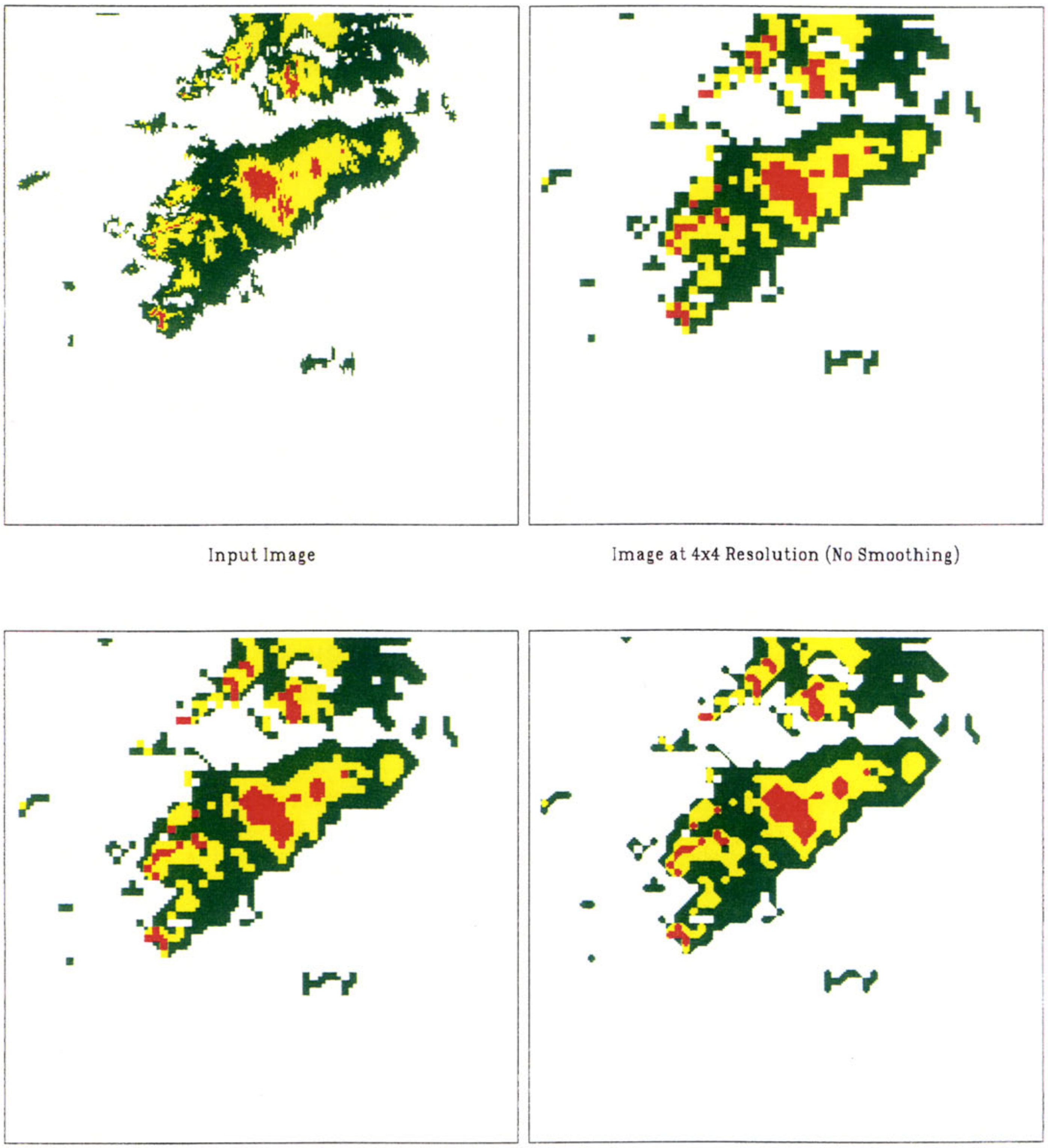
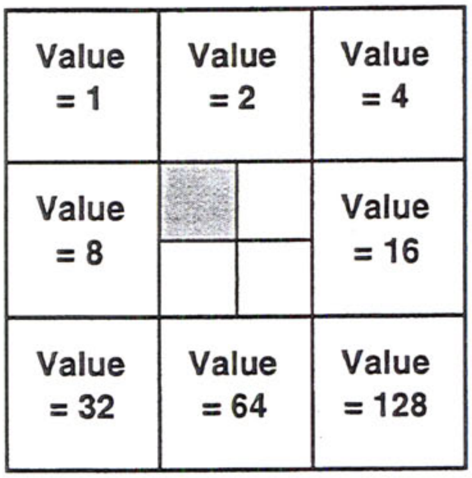

Input Image

Image at 4x4 Resolution (No Smoothing)

Image at 4x4 Resolution after Smoothing at 2x2

Image at 4x4 Resolution after Full Smoothing (2x2 and 1x1)

Figure 6–4. Effect of Various Levels of Smoothing on Output New York Image.



For the shaded upper left quadrant of the center superpixel to be smoothable, by the first rule stated above, the neighbors of values 1, 2, and 8 must all be of lower weather level. But, by the second rule, either neighbor 4 or 16, and either neighbor 32 or 64, must be of equal or higher weather level. Thus, if a score is generated for the center superpixel by adding the values of all neighbors that are of lower weather level, the set of scores that permit smoothing of the UL pixel are:

UL(s) = 1 for s = 11, 15, 27, 43, 47, 59, 75, 79, 91,
139,143,155,171,175,187,203,207,219

Similar tables are generated for the upper right, lower left, and lower right quadrants.

6.3 SMOOTHING PROCESS OPERATION

When the smoothing algorithm is run, two images are input: the lower resolution image MS already decoded, and the higher resolution image MR to be created by the Smoothing Process. For example, the MS image may be resolution 4x4, in which case the MR image will be 2x2. For each superpixel of image MS, each of the 8 neighboring superpixels are checked to determine the smoothing score by noting which set of them has lower weather levels.

Then, using the UL table, the upper left quadrant on MR is set equal to the superpixel weather level if UL(score)=0, or instead set one weather level lower if UL(score)=1. Similar actions are taken for the three other quadrants of the superpixel on image MR. To save time, superpixels of weather level 0 are not checked, as all of its quadrants on MR must also become level 0.

7. THE EXTRA BIT ALGORITHM (EBA)

Unless there is only sparse weather on the image, the Huffman encoding of the image will need to be applied to a reduced resolution version of the input image, with pixels replaced by superpixels (2x2, 4x4, or 8x8). Then the decoder will expand each received superpixel into an array of pixels, each with the same level. This will result in output weather regions being noticeably larger and more blocky than those on the input image. While the smoothing algorithm of the previous section attempts to reduce this effect, it has two major drawbacks:

1. only a few pixels are subject to level reduction, and
2. the smoother makes its reduction without any knowledge of the actual weather contours

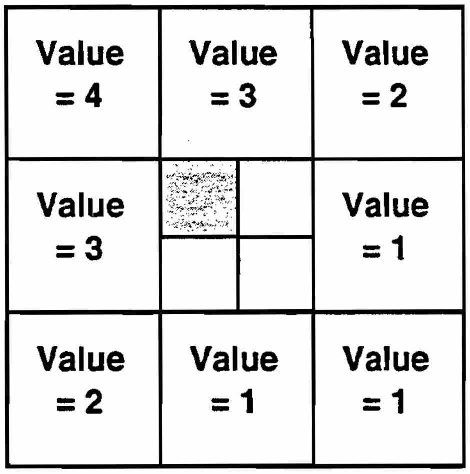
In effect, the smoothing algorithm uses knowledge of typical weather region shapes to produce a more esthetically pleasing result, which is usually, but not guaranteed to be, more accurate than the decoder output.

The Extra Bit Algorithm (EBA) utilizes the bits remaining in the uplink message to relate to the decoder correction information for selected quadrants of as many superpixels as possible. The information, one bit per quadrant, either says to leave the quadrant at the superpixel level or to reduce the quadrant by 1 weather level. Successive applications to the same quadrant, via more than pass through EBA, can achieve greater than 1 level reductions, and quadrants that were incorrectly smoothed down can be returned to their original correct values.

Examples of the improvement in output image fidelity produced by applications of the Extra Bit Algorithm are presented later in Section 10.

7.1 SCORING OF QUADRANTS FOR EXTRA BIT APPLICATION

Since there will never be sufficient bits left to specify all quadrants (if there were, a lower amount of resolution reduction would have succeeded), the quadrants are ranked in order of predicted probability of requiring reduction. The weather levels of the neighboring superpixels are used to divide the set of quadrants into a hierarchy of classes from most likely to least likely needing reduction; the four quadrants of a single superpixel may fall into four different classes. The following is the scoring values for the neighbors of the upper left quadrant:



where the number indicated for each neighbor superpixel is added to the score of the upper left quadrant if that superpixel is at or above the level of the quadrant.

Thus, a quadrant totally within a weather region will receive a score of 17 and be the least likely to require reduction, while a quadrant of an isolated superpixel will receive a score of 0. The score values were determined by examining a set of sample images and computing the percent of the time a quadrant required reduction when a particular neighbor was below its level; the greater the percentage, the higher the assigned score.

7.2 SELECTION OF CLASSES FOR EXTRA BIT APPLICATION

Once all quadrants are scored, the EBA algorithm determines, for each score class, the fraction of its quadrants that require level adjustment. Adjustment is required if either:

(a) the quadrant has not been smoothed down, but its correct value is below that of its superpixel, or

(b) the quadrant has been incorrectly smoothed down.

For each weather level, a minimum fraction parameter P_{level} exists; if a class adjustment fraction exceed this parameter, it is deemed worth using extra bits for that class.

In theory, the adjustment fraction will be a monotonically decreasing function of score, so that a "highest worthwhile score" can be found. All scores up to and including this score would then be encoded by EBA, and the rest ignored. In practice, however, the fraction for one score can easily exceed that of its next lower neighbor. Thus, the algorithm selected to find the highest score H to use is as follows, where N_T and N_A are the total number of quadrants of score N and the number of quadrants requiring adjustment of score N, respectively:

```
1.  Initialize winning index H= -1.
2.  Initialize search index J= -1.
3.  Increment J.
4.  If $J_T = 0$ (no quadrants exist at score J), go to step 3.
5.  If $\frac{J_A}{J_T} \geq P_{level}$ (score J is worthwhile), set H=J, go to step 3.
6.  Set temporary index K = J.
7.  Increment K.
8.  Increment $J_A$ by $K_A$ and $J_T$ by $K_T$ (combine scores J through K).
9.  If $\frac{J_A}{J_T} \geq P_{level}$, set H=K, J=K, go to step 3.
10. If K<17, go to step 7.
11. H is the desired score.
```

Thus, when a score fails the requirements test, and combining that score with successive scores fails to correct the situation, the last successful score is used by EBA. Should the final value of H be -1, no successful score exists, and so no EBA is used for this weather level.

Of course, the number of bits remaining in the message may not be sufficient to accommodate all the quadrants for scores through H. In that case, only a number of quadrants equal to the number of remaining bits can be specified. The value of H_{actual} used by EBA will then by necessity be less than the desired value of H. In particular, if B is the number of bits available, H_{actual} is the smallest value such that:

$$\sum_{i=1}^{H_{actual}} i_T \geq B$$

The values specified to the decoder in the encoded message are the score H_{actual} and the number N_{max} of quadrants of level H_{actual} that can be accommodated:

$$N_{max} = B - \sum_{i=1}^{H_{actual}-1} i_T$$

If no bit limitation is encountered, the score $H_{actual} = H$ and N_{max} is unlimited. 5 bits are used to transmit H_{actual}, followed by 1 bit to indicate whether N_{max} is limited (0=no, 1=yes), followed by 9 bits for N_{max} if a limitation exists.

7.3 ENCODING ALGORITHM FOR EXTRA BIT APPLICATION

Extra Bit Algorithm (EBA) encoding is performed on one combination of image resolution and image weather level per call to the routine. The first series of calls is at one level less resolution reduction than was used for the Weather-Huffman encoding, and occurs after smoothing was applied to generate the new higher resolution image. The levels are processed in decreasing order, starting at the image maximum L_{max}. If sufficient bits exist for a full pass through EBA to occur, so that all weather levels are processed, smoothing is again applied to generate the next higher resolution image, after which EBA starts again at the highest image level. In the most extreme case (8x8 Weather-Huffman resolution), three passes can occur through EBA.

The encoder employs three images in its processing:

1. Image MS, the superpixel image that exists after Weather-Huffman decoding and any previously completed EBA passes are performed.
2. Image MR, the image at the resolution of the routine call, which is one level less resolution reduction than MS, that exists after the latest smoothing application.
3. Image MA, the input image at the same resolution as image MR.

Image MS is used to determine the neighbor score discussed above for each of the quadrant pixels on image MR, the image to be corrected by EBA. Image MA provides "truth", so that the algorithm will know which pixels are to have change bit indications sent to the decoder.

The first step in the decoding process is to scan image MS row by row. For each superpixel whose level is at or above the level of the subroutine call, its four quadrants (NW, NE, SW, SE) on image MR are scored and evaluated as to whether a change is required. A change is needed either when the MS pixel is at the same level as the MS superpixel but the true pixel of MA is lower level, or when the MS pixel has been incorrectly smoothed below the MS level.

After this scan is complete, the selection of classes for EBA application is determined as described above. Using this selection, and the number of message bits remaining, the values of H_{actual} and N_{max} are calculated and added to the message, also as described above.

The remaining part of the EBA is the listing in the encoded message of the information bits themselves, one for each applicable quadrant on image MR. The superpixels of image MS are once again scanned row by row, and for each quadrant on MR of an applicable superpixel, its neighbor score is checked (it was computed and saved in the class scoring procedure). If the score is less than H_{actual}, or is equal to H_{actual} and is one of the first N_{max} of that score, a '0' bit is added to the message if the quadrant should be lower level than the MS superpixel, or a '1' bit is added if the quadrant should be at least as high as the MS superpixel.

In order for the encoder to process successive EBA calls, it must modify its MR image to track the actions to be taken by the decoder. Thus whenever a '0' bit is created, if the pixel on MR is above the level of its MS superpixel, the pixel must be set one level below that value. Similarly, whenever a '1' bit created, and the pixel is of lower weather level than its superpixel, the pixel on MR must be set to the superpixel level. These actions ensure that further applications of EBA will refer to the same image as that used by the decoder.

7.4 DECODING ALGORITHM FOR EXTRA BIT APPLICATION

The EBA decoder is responsible for modifying the weather levels of a set of the superpixel quadrants on the output image that exists after Weather-Huffman decoding. The actions to be taken for the set of quadrants judged worthy of consideration are listed via the message bits remaining after the Weather-Huffman decoding is completed.

The EBA decoder is called separately for each weather level processed by the encoder. It may be called two or three times for the same level if more than one pass through the levels has occurred. Of course, successive passes occur at different degrees of resolution reduction.

The EBA decoder uses two weather images. The first is the reference superpixel image MS that exists after Weather-Huffman decoding and any previously completed EBA passes are performed, while the second is the currently active image MR, of one level less resolution reduction than MS, that exists after the latest smoothing application. For example, if MS is a 4x4 superpixel image, MR will be a 2x2 image. Each superpixel of MS corresponds to 4 elements of MR; these elements (which can be pixels or superpixels) are referred to as the quadrants of the MS superpixel.

The scoring classes employed by EBA have been discussed above. The encoder determined the highest class to be adjusted by EBA, and if a bit limitation existed, the number of quadrants at this class that were processed. The decoder's first action is to read these values. The first 5 unread message bits specify the highest class H. If H=31, this is a signal that no EBA is to occur at this level, and the decoder simply returns. The next bit indicates whether a limit exits (1) or not (0); if yes, the next 9 bits specify the quadrant count C for score H (if no limit, C is set to 100000).

The remainder of the EBA decoder simply scans image MS row by row, one superpixel at a time. For each superpixel whose level is at least as high as the level of the subroutine call, its four quadrants (NW, NE, SW, SE) on image MR are processed. For each, the decoder computes the neighbor superpixel score, referencing image MS, using the scoring algorithm detailed above.

If the quadrant score S satisfies $S < H$, or if $S = H$ and the count of such occurrences is one of the first C, a message bit is read to determine the action to take for the quadrant pixel on image MR. If the bit is '0', the quadrant should end up lower than the level of the EBA call; thus if it is not, it is set to one level less than the calling level. On the other hand, if the bit is '1', the quadrant should end up at least as high as this level. If it is lower, due to being incorrectly reduced by the smoothing algorithm, it is returned to the calling level.

8. WEATHER-HUFFMAN ALGORITHM ENCODING

This section provides the complete message encoding process utilized by the Weather-Huffman Algorithm. This process incorporates two sub-algorithms:

1. encoding, via the Weather-Huffman runlength algorithm, the highest resolution version possible, given a specified data link bit limit, of the original input image
2. using the Extra Bit Algorithm on some or all weather levels to improve the fidelity of the output image generated by the runlength decoder

The details of the encoding procedures for the Extra Bit Algorithm have already been provided in Section 7, so that information will not be repeated here. Also, many of the details of the Weather-Huffman runlength encoding algorithm have been provided in Sections 3 and 4. On the other hand, the full set of steps of the Weather-Huffman encoding logic, including when each of the already discussed routines are utilized, is fully specified below.

8.1 SEQUENCE OF ENCODING METHODS

Unless the input weather image has only very sparse weather, the attempt to losslessly encode the image via Weather-Huffman runlength encoding will generally exceed the bit limitation imposed on the link. In such cases, a sequence of progressively more reduced resolution and filtered approaches are applied; the first to meet the bit requirement is used to transmit the image over the link to the decoder. This section describes in order the sequence of combination runlength, smoothing, and extra bit algorithm (EBA) selected for WH. For each method, the size of the superpixel (1x1, 2x2, 4x4, or 8x8) used for runlength encoding is listed first. Section 5 presented the method employed to construct these superpixels from the original input pixels.

1. 1x1 encoding

This is the attempt to losslessly encode the input image using the actual pixels. No smoothing or extra bits are required.

2. 2x2 encodingThis case attempts to encode the reduced resolution 2x2 image. The result is smoothed to 1x1, after which any remaining message bits are used by the EBA to correct as many as possible of the incorrect 1x1 pixels.

3. 4x4 encoding

Experience with sample complex weather images has revealed two facts: most such images require at least 4x4 resolution reduction to have a chance of meeting typical link bit limitations, and any further resolution reduction to 8x8 produces a decoded image with severe loss of fidelity. Thus, the WH algorithm pursues several subcases in an attempt to encode the image as accurately as possible within the 4x4 resolution reduction restriction.

3.a. 4x4 encoding using unfiltered 4x4 image

This case attempts to encode the reduced resolution 4x4 image produced as described earlier. If successful within the bit limit, the output image is smoothed to 2x2 resolution and the EBA algorithm is applied. The lowest level L acted upon by EBA is then noted, and the following actions taken:

$L \geq 3$ or failed bit limit — convert processing to case 3.b.

$L \leq 2$ and $L < L_{max}$ — convert processing to case 3.c.

If processing remains in case 3.a., the encoding is completed. The decoder will smooth the 2x2 image version existing after EBA to 1x1 resolution to form the output image.

3.b. 4x4 encoding using filtered 4x4 image

In Weather-Huffman encoding, the original 4x4 image in case 3.a., either the attempt failed to meet the bit limit or there were insufficient bits remaining for EBA to correct the high weather levels on the 2x2 image. Since these levels are more critical to pilots, a tradeoff is made by pre-filtering the 4x4 image as described in Section 5, which has the effect of slightly distorting the lower weather levels. When the filtered image is runlength encoded, more bits are available for EBA; hopefully, the higher weather levels can now be completed. If sufficient bits are generated to complete all weather levels at 2x2 resolution, EBA can be continued on the smoothed 1x1 image to further refine the high weather levels.

3.c. 4x4 encoding using modified 4x4 image

In Weather-Huffman encoding, the original 4x4 image in case 3.a., there were sufficient bits for EBA to fully correct one or more higher weather levels on the 2x2 image. Unfortunately, the 4x4 image may not have had the data it required for its job. For example, consider the following situation in Figure 8-1.

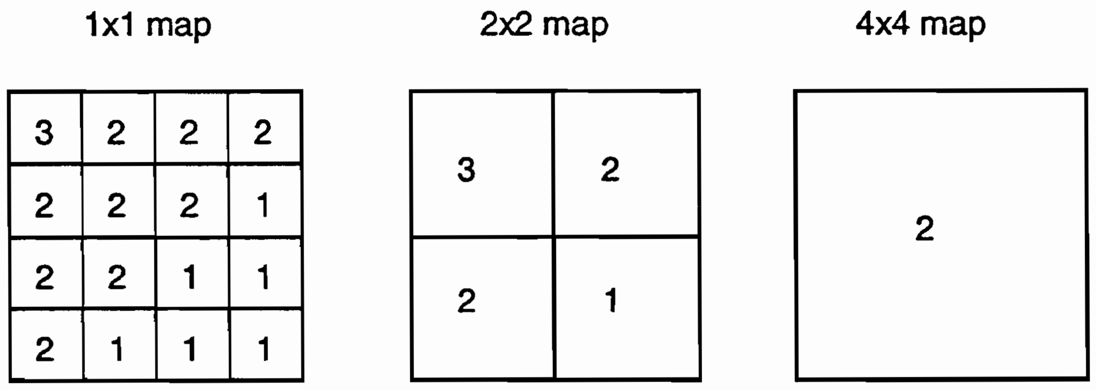


Figure 8-1. An example of Weather-Huffman resolution reduction (numbers are weather levels).

When the 4x4 superpixel is expanded to 2x2 via smoothing, all quadrants will become level 2. The EBA algorithm cannot create the higher level 3 quadrant seen on the 2x2 image; it can only reduce quadrant levels.

The solution to this defect is to modify the 4x4 image prior to encoding. Specifically, this 4x4 superpixel is set to the maximum of its 2x2 image constituents (which is 3 in this example). Then the EBA can operate on the four level 3 quadrants produced in the expansion to leave one at level 3 and reduce the others back to level 2 in order to produce a 2x2 image with the desired set of values. The 4x4 superpixels that are modified in this process are any that have:

a) a constituent 2x2 quadrant of higher level than the superpixel level,

b) that quadrant is of level L+1 or greater, as case 3.a. has already indicated that these are the levels that will be completed by EBA, and either

c1) this quadrant is of level 3 or higher (severe weather), or

c2) two or more quadrants of the superpixel exist with this level

4. 8x8 encoding

If 4x4 resolution reduction was insufficient to Weather-Huffman encode the image, a final attempt using 8x8 reduction is made. First, the 8x8 image generated as described in Section 5 is employed; if this image requires too many bits, the pre-filtered version of the 8x8 image is created. No tradeoff case as used for 4x4 is employed here, as the lower weather levels have already been strongly distorted by the resolution reduction.

4.a. 8x8 encoding using unfiltered 8x8 image

This case attempts to encode the reduced resolution 8x8 image produced as described earlier. If successful within the bit limit, the output image is smoothed to 4x4 resolution and the EBA algorithm is applied. If bits still remain, the new 4x4 image is smoothed to 2x2 and the EBA algorithm is continued. In the most extreme case, EBA can even be applied to the smoothed 1x1 image.

4.b. 8x8 encoding using filtered 8x8 image

This case is identical to step 4.a. except that the filtered 8x8 image is the starting entity. Again, EBA is applied to each new resolution level in succession until the remaining bits are all used.

5. Increase the message bit limitation

If even 8x8 resolution reduction with filtering fails to produce a image that can be encoded within the message bit limitation, this limit must be increased and the encoding process restarted.

8.2 WEATHER-HUFFMAN ENCODING OVERVIEW

The image encoding process of the Weather-Huffman algorithm has four main phases: scanning the image to build the runlength statistics, determining the codewords to use for each weather level, encoding the decoding tables corresponding to these sets of codewords, and using these codewords to sequentially encode the runs of the image. Scanning can be performed by either a row-by-row raster scan or by a Hilbert scan. As discussed in Section 3, the Hilbert scan will in general lead to fewer encoding bits and as such has been selected.

Three preparatory operations are required prior to the image encoding itself. The first is generating the input image at the resolution reduction level that matches the sequence step being pursued. The details of the creation of the superpixels for this degree of reduction were provided in Section 5. Next, the linear 1-dimensional string of superpixels corresponding to the Hilbert scan of this image must be generated by the code provided in Section 3. During this process, the maximum level superpixel L_{max} resident on the image is noted.

Finally, the maximum level superpixel in each 16x16 image quadrant (corresponding to 256 consecutive Hilbert-scanned superpixels) is found and stored in the qmax array. These values are encoded in the message so as to be available to the decoder; 3 bits are employed if $L_{max}>3$, 2 bits otherwise. Using these quadrant maximums, the maximum level applicable to each pixel is stored in the quadval array. To ensure proper end of image processing, the arbitrary value 13 is stored in the lastpixel+1 array element.

8.3 BUILDING THE RUNLENGTH STATISTICS

The general procedure for building the level-by-level runlength statistics is quite straightforward: examine the superpixels one-by-one, noting every time the level changes. At each such occurrence, increment the array element corresponding to the count of consecutive pixels just completed for the level just exited.

Two minor complications must be added to this process. First, if the new level is not contiguous to the previous level, the count of 0 length runs must be incremented for each level passed over. Second, the longest length run explicitly permitted is 63 pixels. Therefore, whenever a longer run is encountered, the count for length "S2" must be incremented and a new count initiated. Specifically, when the 64th consecutive pixel at a level is noted, the count for "S2" is incremented and the count reset to 1.

One alteration to the "S2" process occurs if the run is at level 0 and the 64th pixel occurs in a quadrant having qmax=0 (that is, a quadrant of all 0 pixels). In this case, the "S2" run is extended to the next-to-last pixel in the quadrant prior to the first quadrant for which qmax is no longer 0 (see Figure 8-2).

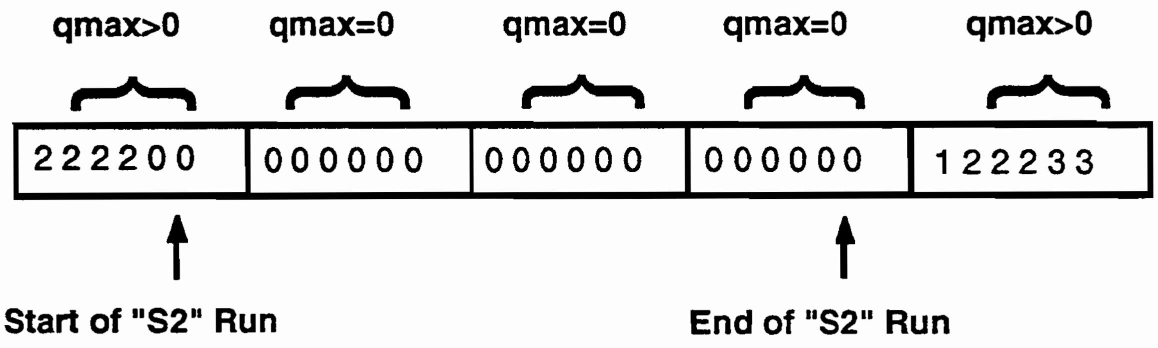


Figure 8-2. Example case indicating extension of S2 run.

This action saves the bits that would be required if several consecutive "S2" runs had to be specified (since a quadrant is 256 pixels, 6 runs of "S2" would be needed to traverse it normally). The decoder, of course, will know the values of qmax, and hence can follow this action.

The specific action taken is to extend the run until the next pixel has a value of qmax>0, then increment the count for "S2" at level 0, and finally reset the count of level 0 pixels to 1. Note that the "S2" run cannot include the last pixel in the quadrant, as then no method would exist to encode a higher level at the first pixel in the new quadrant (the pixel after an "S2" run is always assumed to be at the same level, and length 0 runs are not permitted for level 0).

When the last superpixel of the image is reached, a final explicit length run is recorded. This last run cannot be a long "S2" run due to the setting of quadval(lastpixel+1)=13.

The result of this step is the generation of the array Run(L,G), where L ranges from weather level 0 to level L_{max} and G ranges from runlength 0 to runlength 63 and then the 64th entry is used for runlength "S2". For each level, the longest normal non-S2 runlength R_{max} is noted for later use.

8.4 DETERMINING THE CODEWORDS

Once the Run array is determined, for each image level the encoder must determine the codewords that will correspond to each non-null runlength. The resulting decoding table must then be encoded and added to the message for use by the decoder. The winning codeword set can be one of the three default sets defined for the level, or be a set tailored specifically for this image.

Before the codewords can be generated for a level, two parameters must be calculated: the longest explicitly encoded runlength G_{max}, and the option O to use for encoding "other" longer runlengths. The methods employed for indicating the incremental runlength of an "other" run were presented in Section 4. To review, eight options were defined, five of which always use the same number of bits for all runlengths (a different number for each option), while the other three are long-short options which use two fixed numbers of bits: the smaller for runlengths near G_{max} and the larger for longer runlengths. Depending upon the distribution of "other" runlengths for a level, one option will require the fewest message bits for the encoding of the set of "other" runs.

The first step in determining the codeword set to use for a level is calculating the number of bits that would be required for encoding its runs with the default settings. Each default set specifies the value of G_{max} and the codewords for runs of length 0 through G_{max} and S1 (the "other" length) and S2 (length exceeding 64). The "other" option to use for runlengths beyond G_{max}, however, is open. Thus, all eight are tested against the set of "other" runs to select the best, using the procedure detailed in Section 4. The number of bits assigned to each default option is then the sum of:

1. 3 bits for specifying the "other" option selected
2. the number of bits required by the winning "other" option
3. number of S1 runs x length of S1 codeword
4. number of S2 runs x length of S2 codeword
5. $\sum_{j=0}^{G_{max}}$ (Run(L,j) x length of runlength j codeword)

The index of the winning default set, and the number of bits it requires for level L runs, is stored. Should there be no runs at level L (winning total =3), the first default set is automatically used for the level.

Next, the number of bits required for level L by a tailored set of codewords is computed for comparison. This total would always beat the default case if it weren't for the fact that the bits for the decoding table now has to be included.

With a tailored set, the best value of G_{max} to use has to be determined. Section 4 explained the method employed in this search to avoid trying all possible values, which is to skip any runlength i for which the number of runs R_i satisfies either:

(a) $R_i < R_{i+1}$, or

(b) $R_i \leq 1$

For any other potential value of G_{max}, from -1 (all runs are "other" runs) to R_{max}, the number of bits required to encode the level is calculated, and the minimum-bit value of G_{max} is noted. For each tested value of G_{max}, the steps are as follows:

1. Compute the number of "other" runs:

$$R_{other} = \sum_{j=G_{max}+1}^{R_{max}} Run(L,j)$$

2. If $R_{other} > 0$, determine the number of bits required by the winning "other" option as described above for the default cases.

3. Order in decreasing order of number of runs the non-zero runlengths from 0 to G_{max} plus S1 and S2.

4. Calculate the Huffman codeword lengths for each of these runlengths (see Sections 3 and 4 for the method).

5. Calculate the number of bits required for the decoding table (see Section 4 for the method).

The number of bits corresponding to this value of G_{max} then becomes the sum of:

1. if $R_{other} > 0$, 3 bits for specifying the "other" option selected

2. the number of bits required by the winning "other" option

3. number of S1 runs x length of S1 codeword

4. number of S2 runs x length of S2 codeword

5. $\sum_{j=0}^{G_{max}}$ (Run(L,j) x length of runlength j codeword)

6. number of bits to specify the decoding table

Finally, the bit requirement for the winning default option is compared to the bit requirement of the tailored codeword set with the optimum value of G_{max}. If the tailored set is victorious, the ensemble of codeword lengths produced by the Huffman algorithm call corresponding to that G_{max} value will be employed in the weather image decoding. The actual codewords themselves are determined in the next step during the encoding of the decoding table.

To save encoding time, if the bit requirement of the selected alternative, when added to those of the lower levels already processed, exceeds the message bit limitation, the encoding procedure is terminated, and the next method of the encoding sequence pursued instead.

8.5 ENCODING THE DECODING TABLE

If a default codeword set is selected, the encoding of the decoder table is trivial:

1. Use 2 bits to encode the number (0-2) of the default set selected.
2. Use 3 bits to encode the winning "other" option.

Then the codewords lengths and codewords themselves of the default ensemble are read from storage and transferred to the encoding arrays.

The process is more complicated if a tailored codeword set is selected. In that case, the encoding of the decoder table proceeds as follows:

1. Use 2 bits to encode the number '3', indicating a tailored set of codewords is being used.
2. Use 3 bits to encode the longest Huffman codeword H_{max} being used (maximum length allowed is 7).
3. Initialize runlength $G = -1$.
4. Increment G.
5. If $G > G_{max}$, go to step 10.
6. If no runs of length G exist for this level, that is if $R(L,G) = 0$, add a '0' to the message, and return to step 4.
7. Add a '1' to the message.
8. Determine, as explained in Section 4, the number of bits required to encode the length of the codeword H_G for runlength G.
9. Encode $(H_{max} - H_G)$ using that many bits, and return to step 4.
10. If "other" runs exist, that is if $R_{other} > 0$, use 3 bits to encode the winning "other" option.

The last step prior to encoding the image itself is determining the actual codewords for each runlength, including S1 and S2. This process, which uses the codeword lengths already known from the call to the Huffman algorithm, was presented in detail in Section 3.

8.6 ENCODING THE IMAGE

The last phase of the encoding process, encoding the image itself, is the simplest. The image's superpixels are traversed once again from beginning to end. As each new run termination is encountered, the codeword (or codewords if an "other" length run is found), followed by the level transition bit (if needed), corresponding to the previous runlength and next level combination, are added to the encoded message. The method of determining the end of a run is the same as above, including the special "S2" cases.

If the terminated runlength has an explicit codeword assigned to it (always true for "S2" runs), that codeword is added to the message. Otherwise, the codeword for "S1", the "other" length run, is added to the message. This "S1" codeword must then be followed by the bit sequence that spells out the differential length $G-G_{max}$ of the run, where G_{max} is the longest explicitly coded runlength.

The steps taken by the encoder for an "other" length run of length G on level L are as follows, where the option to use for level L was determined above in the previous step:

1. If level L does not use a long-short option, go to step 4.
2. Determine if G exceeds the short field maximum length for the option.
3. If it does, add a '1' to the message, else add a '0' to the message.
4. Add the binary value $G-G_{max}$ to the message using the appropriate number of bits for the option in effect.

After the completed runlength has been encoded, the encoder must check to see what type of level transition message must be generated. The normal case consists of adding a '0' to the message if the new level is higher than the old one, or adding a '1' if the new level is lower. No bit is required, however, if the old level was 0 (must go up) or if it was at or above the quadrant maximum of the new runlength's starting pixel (must go down).

Another problem arises whenever the level change is greater than a single unit, such as going from level 1 to level 3. The decoder must be informed of the levels skipped over, by including codewords for length 0 runs at each such intermediate level. The codewords are determined as just specified; length 0 can be either an explicit runlength or an "other" runlength. However, if the transition is downward, and any intermediate level is above the quadrant maximum of the new runlength's starting pixel, no length 0 run need be given for such a level, as seen in Figure 8-3.

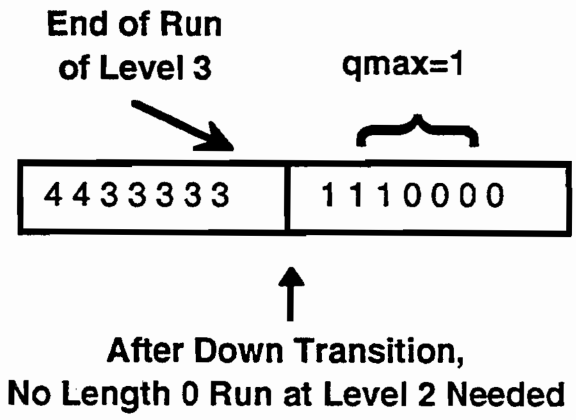


Figure 8-3. Example where 0 length run specification is not required.

No level transition bit is required after a length 0 run, as the same direction as the one last specified is known to apply.

9. WEATHER-HUFFMAN ALGORITHM DECODING

This section provides the complete message decoding process utilized by the Weather-Huffman Algorithm. This process incorporates three sub-algorithms:

1. decoding the Weather-Huffman runlength-encoded image, which may be resolution reduced from the original input image
2. applying the Smoothing operation sufficient times to regenerate a image with the original resolution
3. using the Extra Bit Algorithm on some or all weather levels to improve the fidelity of the smoothed images

The details of the decoding procedures for the Smoothing and Extra Bit Algorithm have already been provided in their respective Sections 5 and 6, so that information will not be repeated here. The Weather-Huffman decoding logic, on the other hand, is fully specified below.

9.1 DECODER CONTROL LOGIC

Depending upon the degree of input image resolution reduction required to enable application of the Weather-Huffman (WH) runlength encoding, and the number of weather levels that the Extra Bit Algorithm (EBA) was successful in completing, the decoder will process one of seven possible cases. The first 3 bits of the message specify the applicable case. The next three bits provide the highest weather level L_{max} of the input image. Finally, the next 3 bits specify the lowest weather level L_{EBA} processed by the Extra Bit Algorithm (EBA) on its final pass; different cases are used to represent different numbers of EBA passes. For the last pass, EBA progresses in order from the highest weather level L_{max} to level L_{EBA}; on all prior passes it progresses all the way down to level 1.

The actions taken by the decoder for the 7 possible cases are as follows:

Case 1: 1x1 WH resolution

The WH decoding will produce directly the Hilbert scanned version of the output image. This image is then transformed back to the raster scan version which constitutes the final image.

Case 2: 2x2 WH resolution

The WH decoding will produce the Hilbert scanned version of the 2x2 resolution output image. This image is transformed back to the raster scan version of the 2x2 image, which is then smoothed to a 1x1 resolution image. For each weather level in order, from the highest level L_{max} down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce the final output image.

Case 3: 4x4 WH resolution with a single 2x2 EBA pass

The WH decoding will produce the Hilbert scanned version of the 4x4 resolution output image. This image is transformed back to the raster scan version of the 4x4 image, which is then smoothed to a 2x2 resolution image. For each weather level in order, from the highest level L_{max}

down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce a corrected 2x2 image. This image is then smoothed to 1x1 resolution to generate the final output image.

Case 4: 4x4 WH resolution with 2x2 and 1x1 EBA passes

This case starts the same as case 3, except that all levels of the 2x2 image receive EBA treatment to produce the corrected 2x2 image, which is then smoothed to generate a 1x1 image. For each weather level in order, from the highest level L_{max} down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce a final 1x1 output image.

Case 5: 8x8 WH resolution with a single 4x4 EBA pass

The WH decoding will produce the Hilbert scanned version of the 8x8 resolution output image. This image is transformed back to the raster scan version of the 8x8 image, which is then smoothed to a 4x4 resolution image. For each weather level in order, from the highest level L_{max} down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce a corrected 4x4 image. This image is then smoothed to 2x2 resolution, and then further smoothed to 1x1 resolution, to generate the final output image.

Case 6: 8x8 WH resolution with 4x4 and 2x2 EBA passes

This case starts the same as case 5, except that all levels of the 4x4 image receive EBA treatment to produce the corrected 4x4 image, which is then smoothed to generate a 2x2 image. For each weather level in order, from the highest level L_{max} down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce a corrected 2x2 image prior to the final smoothing to 1x1 resolution.

Case 7: 8x8 WH resolution with 4x4, 2x2, and 1x1 EBA passes

This case starts the same as case 6, except that all levels of the 2x2 image receive EBA treatment to produce the corrected 2x2 image, which is then smoothed to generate a 1x1 image. For each weather level in order, from the highest level L_{max} down to the level L_{EBA} specified in the message, EBA is applied to correct this image and produce the final 1x1 output image.

9.2 WEATHER-HUFFMAN (WH) DECODING PROCESS

The basic Weather-Huffman decoding process consists of iteratively reading a runlength at a known weather level, followed by determining the direction of transition to the next weather level. In order for the decoder to know the codewords employed by the encoder to specify the runlengths, the message must contain the specifications of the decoding table.

The first step of WH decoding is to check for a maximum weather level of 0; in that case, an empty image is immediately returned. Otherwise, the decoder next reads the weather level maximums for each image quadrant. These are used to reduce the number of weather transitions that require explicit specification: if a run is at the quadrant maximum, a down transition must follow. The number of bits for each quadrant maximum is either 2 or 3, depending upon the maximum weather level for the image (3 bits if $L_{max} > 3$).

After these preliminaries are completed, the process of reading the decoding table and decoding the weather runs commences.

9.3 DECODING TABLE REPRESENTATION

The first step, and by far the most complex step, of the image decoding operation is the construction of the decoding tables for each weather level. As explained in the decoding table Section 4, numerous "tricks" are employed by the encoder to compress the number of bits required to transmit these tables. The job of the decoder is to read the message bits allocated to decoder table specifications, undo these tricks, and produce the array that expresses the decoding tables.

The decoding table array is a 3-dimensional entity of the form:

dec_run(level, #bits, codeword) —> length of run

For example, the entry

dec_run(3, 4, 5) —> 2

says that if you know that the next run consists of level <u>3</u> pixels, and the next <u>4</u> message bits to be decoded are 0101 (binary <u>5</u>), then the run is of length <u>2</u>.

Most of the entries in the decoder array will set to -1, which is a signal that the entry does not correspond to a valid codeword. Since the Huffman code is a prefix code, the following entries are mandated by the above example:

dec_run(3, 1, 0) —> -1,

dec_run(3, 2, 1) —> -1, and

dec_run(3, 3, 2) —> -1

since the codewords 0, 01, and 010 are all prefixes of the assumed codeword 0101. When the decoder accesses a decoder array entry of -1, it knows that it must read another message bit in its search for the next runlength codeword.

In addition to the decoding array, the decoder must know how to read the actual length of the "other" length runs that are signaled by the presence of an S1 entry in the decoding array. These lengths are those beyond the longest length explicitly assigned a codeword for the level. As detailed in the decoding table theory Section 4, there are 8 possible options for the number of bits used to encode these runlengths; in 3 of the options, two different lengths, a "long" length and a "short" length, are used. Since the length parameters of each option are fixed, the decoder need only be told the option number in effect for each level.

9.4 DECODING TABLE CREATION

The actual operation of decoding the message bits that encode a given level's decoding table commences with the determination of whether a default table or a tailored table has been selected. For the default cases, the total decoding table generation process is extremely simple:

1. Read T, the value of the next 2 message bits.

2. If T = 3, a tailored table specification follows; go to step 5.

3. Copy the default array init(T, i, j, k) into dec_run(i, j, k).

4. Read V, the value of the next 3 message bits;

   Set up the parameters for "other" option V for this level.

Appendix A lists the default codeword assignments for each level's default tables.

When a tailored table has been constructed for the weather level, the decoder must read the specification of that table from the message bits. As already described in the encoding Section 8, this specification has been severely compressed to minimize its bit requirement. To review, the basic form of the specification is a list, in order of runlength, of the number of bits in the codeword assigned to that runlength. The specific runlength ordering is as follows:

$$S1, S2, (0), 1, 2, \cdots, R_{max}$$

where R_{max} is the longest explicitly encoded runlength, S1 is the "other" length designation used to signal runs between R_{max} and 63, S2 signals a run of length greater than 63, and the length of 0 is not required for weather levels 0 and L_{max}. Once the set of codeword lengths are known, the actual codewords themselves can be determined by the method described in the Huffman theory Section 3.

For each runlength assigned its own explicit codeword, the number of bits required to state the length of that codeword is a function of the number of lengths still possible. The process connected with determining this number was detailed in the Huffman table Section 4. That section also explained how a Huffman coding scheme could result in the number of bits actually employed varying with the codeword length being utilized.

The actual decoding algorithm continuation for determining a level's tailored decoding table thus becomes:

5. Initialize the sum of the number of codewords used so far to S = 0, and the runlength whose codeword length is being read to R = S1.

6. Read G, the longest codeword length, as the value of the next 3 message bits.

7. Read the next message bit;

   if it is 0, no codeword exists for length R, go to step 13.

8. Use S and G to look up D, the range of possible codeword lengths for R, and then use D to look up the fewest message bits B that could provide the actual codeword length for runlength R.

9. Read V, the value of the next B message bits.

10. If V is not a valid Huffman code for range D, read the next message bit M and set $V = 2*V + M$.

11. Look up the offset F corresponding to V;

    the codeword length for R is $L_R = G - F$.

12. Increase S by the number of codewords covered by L_R;

    if S = 128, all explicit lengths have been specified, go to step 14.

13. Set R to the next runlength in the list;

    return to step 7.

14. Using the set of values L_R, determine the actual codewords for S1, S2, and each explicit length run and create the decoding array dec_run.

15. If a codeword exists for S1, the tailored decoding table uses an "other" option;

    read V, the value of the next 3 message bits;

    set up the parameters for "other" option V for this level.

9.5 IMAGE RUNLENGTH DECODING

After the decoding array dec_run has been established, the actual decoding of the runlengths of the Weather-Huffman encoded Hilbert-scanned image can commence. The next 3 bits of the message specify the level L_1 of the first image superpixel. Each runlength in turn is then read. The basic idea is to read message bits until a valid codeword is located.

Level transitions are read via one message bit (up or down) after each run except when the choice is predetermined. In particular, up is assumed after level 0, down is assumed after a run at the quadrant maximum level, and transition to the new quadrant maximum level is assumed after a run above that level. Multiple level transitions are handled by having a runlength of length 0 for each skipped over level, after which a transition in the same direction is assumed.

The sequence of steps for determining each runlength and transition pair is as follows:

1. Initialize last decoded pixel P to 0, level L to L_1.

2, Initialize codeword C to 0, number of codeword bits N to 0.

3. If R = dec_run(L, N, C) is ≥ 0, the runlength has been found;

   go to step 5.

4. Increment N and read the next message bit B;

   set C = 2*C + B;

   return to step 3.

5. If R = S1 ("other" length), follow rules of the "other" option in effect for level L to determine actual R (see below).

6. If R = S2 (run > 63), set R = 63 unless L = 0 and the quadrant max level is 0, in which case extend R through the next to last superpixel for which the quadrant max is 0 (if extended to last one, a change from level 0 at the next superpixel would be impossible to specify, as no length 0 run at level 0 is permitted).

7. Set superpixels P+1 through P+R to L.

8. Increase P to P+R.

9. If P = end of image, quit.
10. If level transition is not assumed to be known, read next message bit B;

    B=0 -> up, B=1 -> down.
11. Set new value of L;

    return to step 2.

As described earlier, two types of "other" options exist. The first type always employs a fixed number of bits F to specify the actual runlength. When such an option is in effect, the next F message bits are read to produce the value V, and the true runlength R is given by

$$R = R_{max} + V$$

where R_{max} is the longest explicitly encoded runlength for level L.

The second type of option employs two different numbers of bits for runlength specification. Thus the next message bit must be read to determine which length to use: the shorter length (bit = '0') or the longer length (bit = '1'). Once the length is known, that many bits are read, and the procedure for computing R is then the same as for the single length case.

10. WEATHER-HUFFMAN ALGORITHM RESULTS

This section presents more detailed results (beyond those provided in Section 2) of applying the Weather-Huffman Algorithm to various sample weather images. It also highlights the effectiveness of each part of the algorithm by showing incremental results. Various bit limitations are also applied to indicate the graceful degradation properties of the algorithm. For ease of viewing results, all images have been converted to 3-level weather: green for light rain, yellow for moderate rain and turbulence, and red for heavy rain and severe turbulence.

10.1 DEGREES OF RESOLUTION REDUCTION

The greater the degree of resolution reduction forced on the algorithm by the bit limitation, the greater the loss of fidelity that results in the encoded image. Figure 10-1 presents, for a sample image, the four levels of reduction considered by the Weather-Huffman Algorithm: none (or 1x1 superpixels), 2x2, 4x4, and 8x8. It is clear that as resolution decreases, two significant effects occur: the weather regions become blockier, and higher weather level regions bloom in size. The blooming alternative is to lose small regions of high level weather, which for pilot safety reasons is unacceptable.

The figure also provides information on how the encoder bit requirement decreases with greater resolution reduction. The ratio for each step is approximately 2.5 : 1, rather than the 4 : 1 that might have been expected by just counting pixels to be encoded. The reason for this is two-fold: the decoding table overhead increases in importance as the message decreases in size, and the Huffman efficiency decreases as the number of runs decreases.

The algorithm also allows image filtering at both the 4x4 and 8x8 resolution steps, in order to decrease the encoding bit requirement and thus possibly meet the bit limitation without needing further resolution reduction. Figure 10-2 indicates the output images, and bit requirements, produced both with and without filtering for the 4x4 and 8x8 levels. Two results are clear from this figure:

1. Filtering provides a significant reduction in encoding bits for a minor change in image.
2. A filtered 4x4 image is strongly preferable to an 8x8 image, and thus is worth trying first.

10.2 INCREMENTAL RESULTS

The Weather-Huffman Algorithm contains three separate subalgorithms:

1. Weather-Huffman runlength encoding a resolution-reduced version of the input image.
2. Smoothing the resolution-reduced output image back to the original resolution.
3. Applying the Extra Bit Algorithm to correct errors introduced by the resolution reduction.

Figure 10-3 illustrates the effectiveness of each step by presenting the results of skipping some of the subalgorithms. The upper left picture is the input image. The upper right picture is the output of Weather-Huffman runlength encoding when a 3500 bit limit is imposed. The algorithm was forced to encode a reduced version of the input with 4x4 superpixels, which required 1962 bits. The blockiness of this resolution reduction is quite apparent. The lower left picture shows the effect of employing smoothing on the previous image: the edges of the regions have become more realistic, but unfortunately the regions are still as large as before. Finally, the lower right picture includes the Extra Bit Algorithm processing, using the bits remaining in the 3500-bit limit. It is clear that the red regions have been significantly altered, removing most of the blooming that resulted from the original resolution reduction. In addition, the shapes of the yellow regions have been significantly improved.

10.3 ALGORITHM TRADEOFFS

It is possible to trade off resolution reduction for increased extra bit coding. That is, a possible alternative is 2x2 runlength encoding with no Extra Bit Algorithm versus 4x4 runlength encoding with 2x2 EBA correction employed in the freed up bits. Figure 10-4 considers these alternatives. The upper left image is the input image for the example. The upper right image presents the result of 2x2 Huffman encoding only, while the lower left image is the output from 4x4 Huffman encoding followed by a complete pass of EBA at the 2x2 level.

The first thing to note is that 4x4 with EBA requires significantly fewer bits. This is because EBA only attempts to correct the edges of existing regions. Thus, as seen in the comparison of the images, several more green regions, with superior internal structure, exist in the 2x2 image. The red areas, however, are very similar between the images.

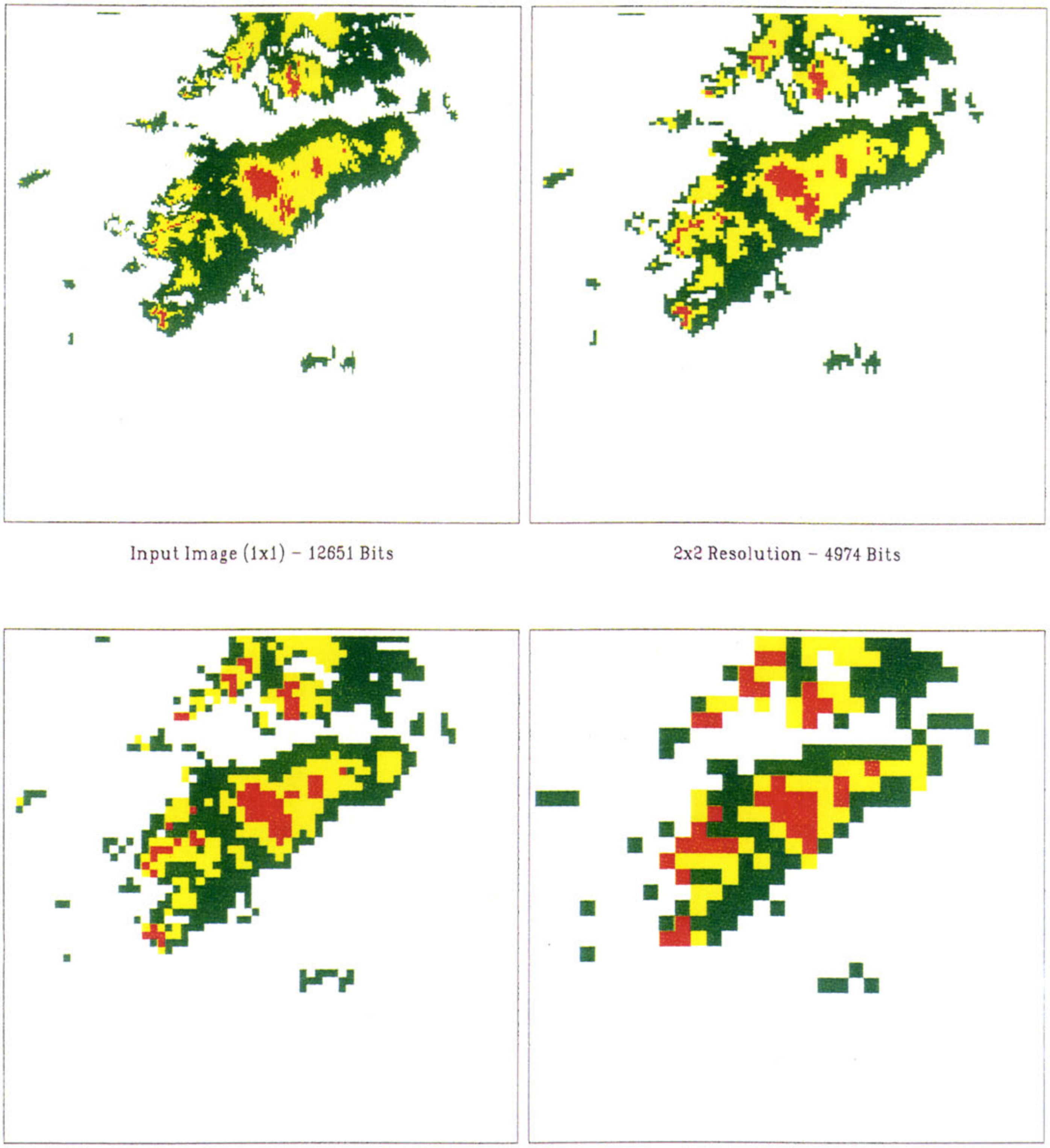
A more fair comparison would be to allow EBA to utilize all the bits freed up by going to 4x4 resolution. The lower right image presents this result. Comparing it with the 2x2 image above it, it is seen that, although the green regions are still inferior, the red regions are significantly more accurate. This results from the EBA being able to correct the red at the 1x1 resolution in its second pass through the weather levels.

Thus, a real tradeoff exists for operational evaluation: is it better to concentrate on accuracy of higher or lower level weather regions. The resulting opinion could alter the decision currently used in the Weather-Huffman algorithm, which is to use the least possible resolution reduction for the Huffman encoding step. It should be noted, however, that if the tradeoff for a severe weather image is between 4x4 reduction versus 8x8 with EBA, the loss of fidelity with 8x8 reduction makes it the losing alternative.

10.4 VARIATION IN LINK BIT LIMIT

Obviously, the more bits allocated to image encoding, the more faithful will be the output result. Figures 10-5a and 10-5b illustrate this effect by presenting the output images produced for two sample images at three different bit limits: 2300, 3500, and 4700.

As the bit limit is increased, the shape and sizes of the various weather regions increase in accuracy. Note that because of the priority scheme employed by the Extra Bit Algorithm, higher weather regions are better preserved than lower ones with lower bit limits. Put another way, the degradation of the output images as the bit limit is tightened is moderate for the most critical weather information. In fact, both of the 2300-bit images are probably acceptable for strategic flying.



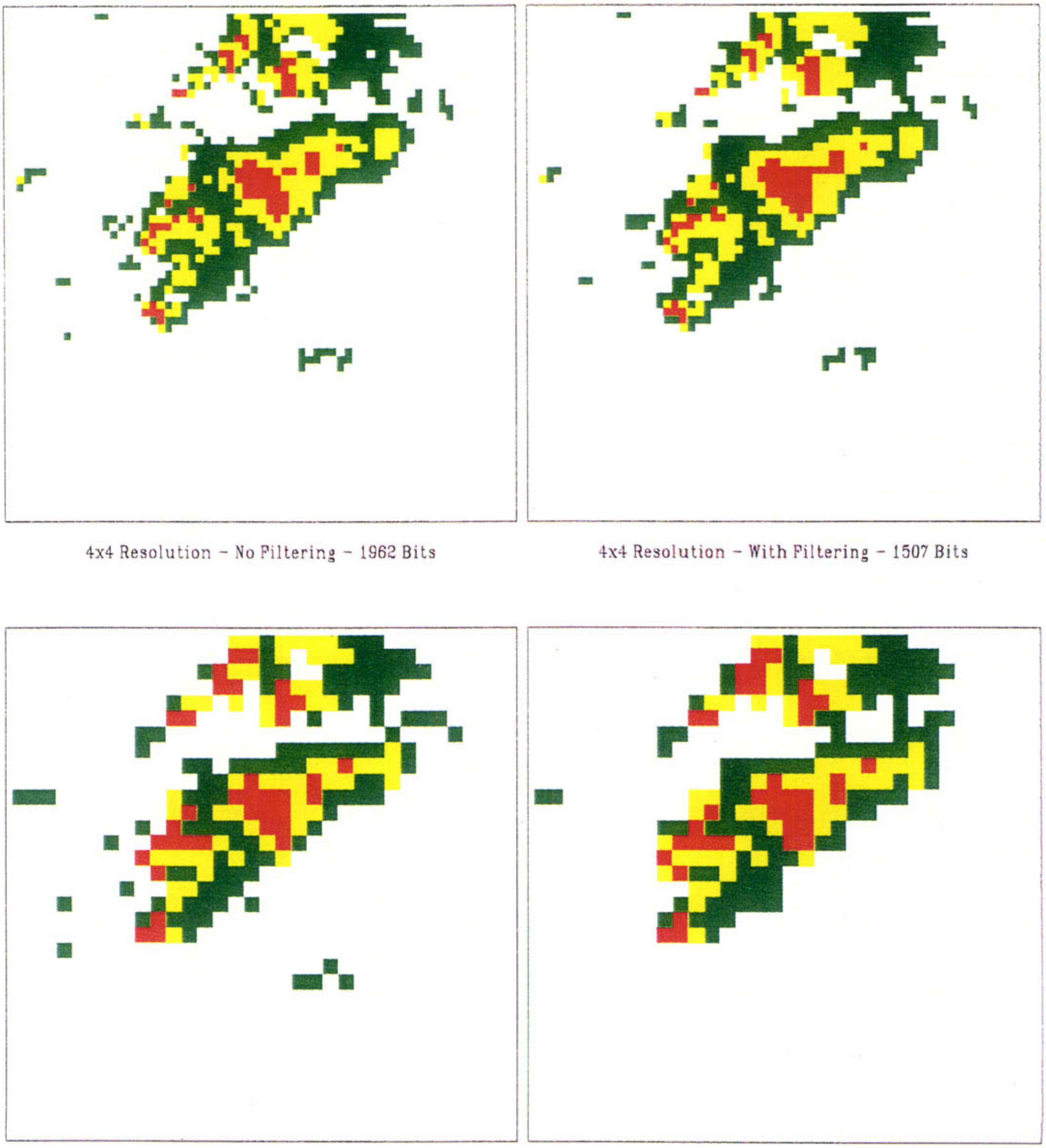
Input Image (1x1) - 12651 Bits

2x2 Resolution - 4974 Bits

4x4 Resolution - 1962 Bits

8x8 Resolution - 731 Bits

Figure 10-1. Effect of Different Resolution Reduction Levels on New York Image. As Resolution Decreases, Image Fidelity Degrades but Bit Requirement Drops.

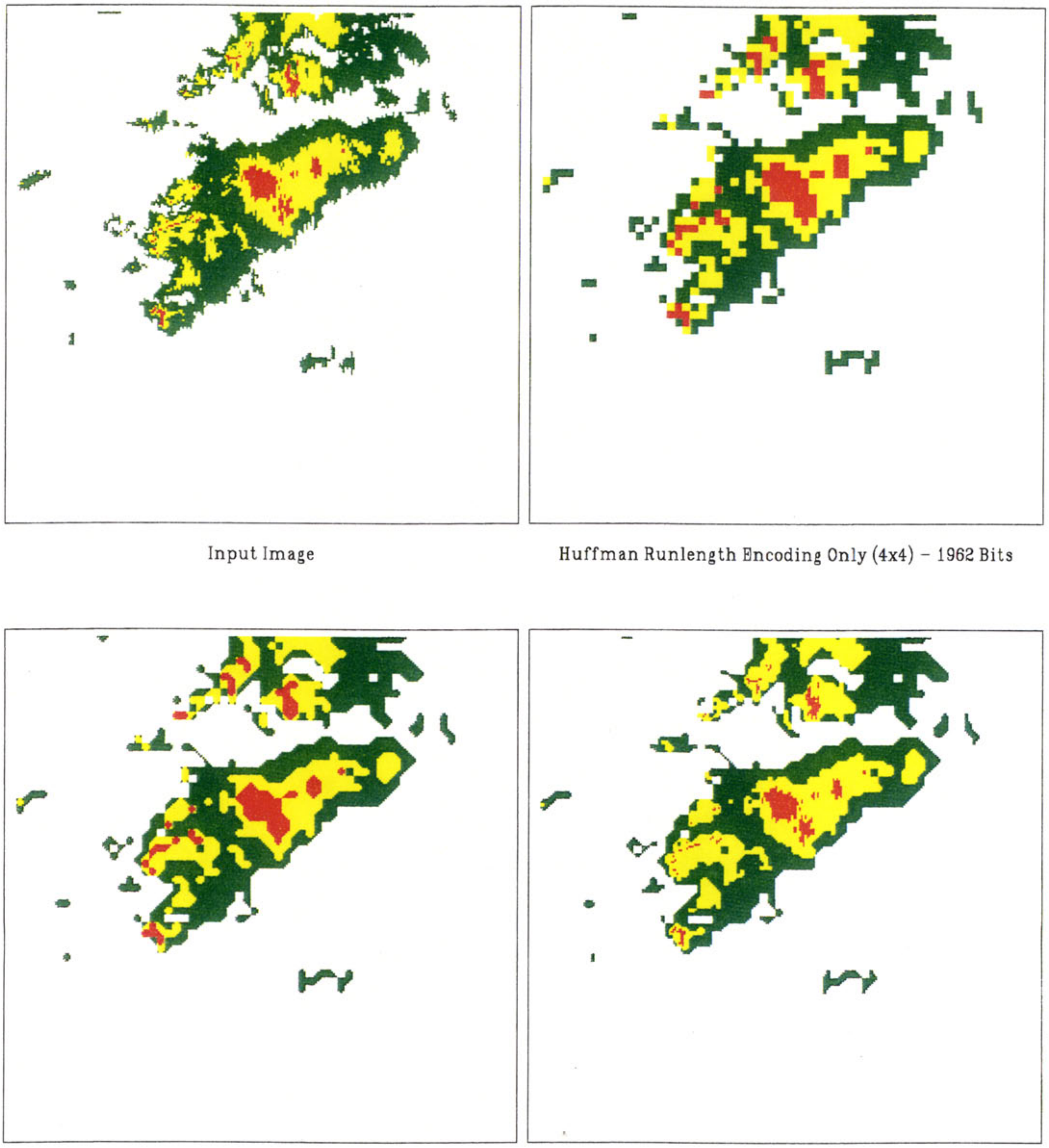

4x4 Resolution – No Filtering – 1962 Bits

4x4 Resolution – With Filtering – 1507 Bits

8x8 Resolution – No Filtering – 731 Bits

8x8 Resolution – With Filtering – 538 Bits

Figure 10–2. Effect of Runlength Filtering at Lower Resolution Levels on New York Image. Fidelity Undergoes Minor Change while Bit Requirement Drops Significantly.


Input Image

Huffman Runlength Encoding Only (4x4) – 1962 Bits

Smoothing Process Included – Still 1962 Bits

Extra Bit Algorithm Applied to 3500 Bit Limit

Figure 10–3. Output New York Image Existing after Each Incremental Step of the WH Algorithm.

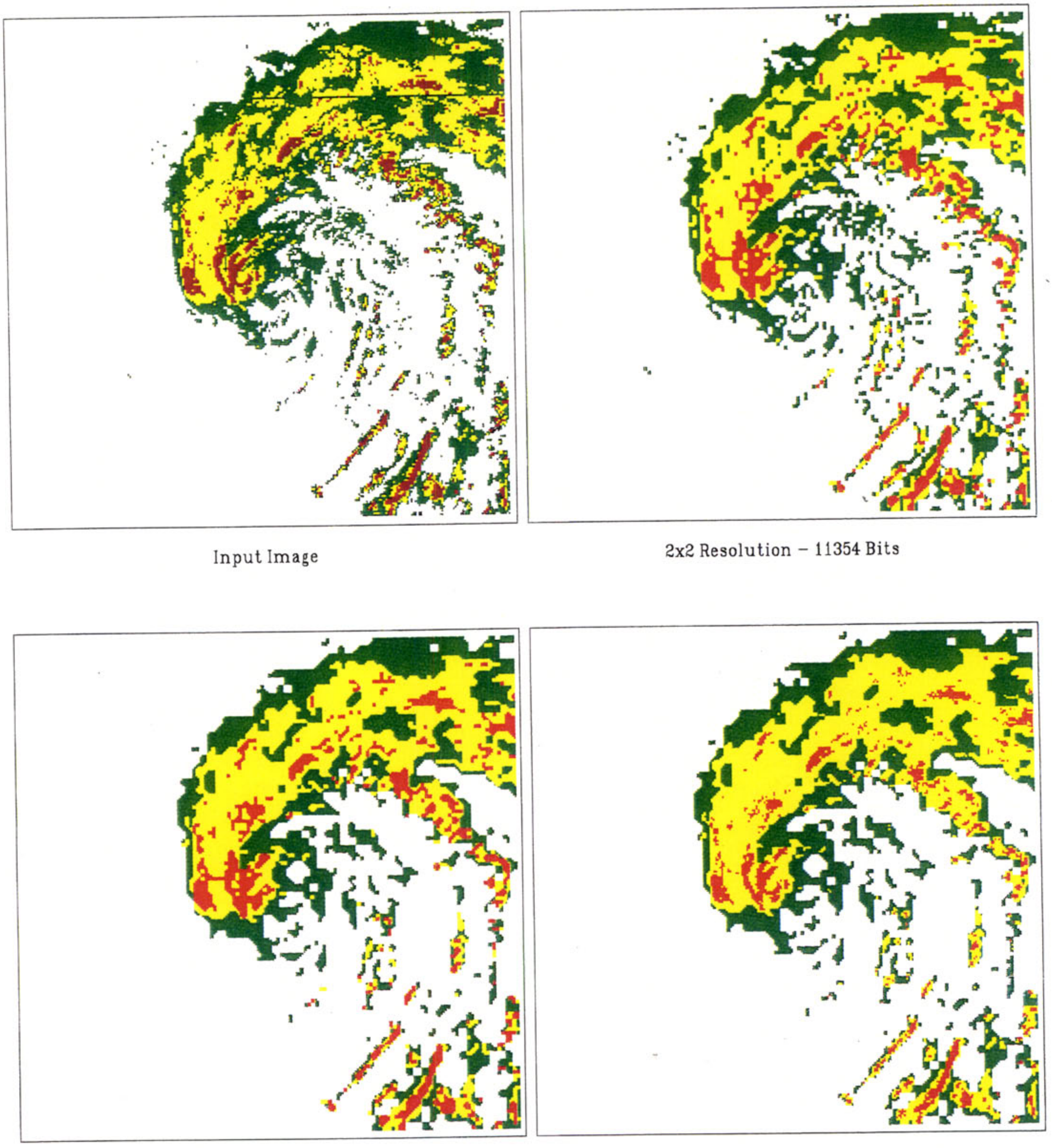


Figure 10–4. Tradeoff for Allison Image Between Resolution and Extra Bits. Extra Bit Algorithm Compensates for Resolution Reduction by Using Bits Freed Up by Application of Lower Resolution.

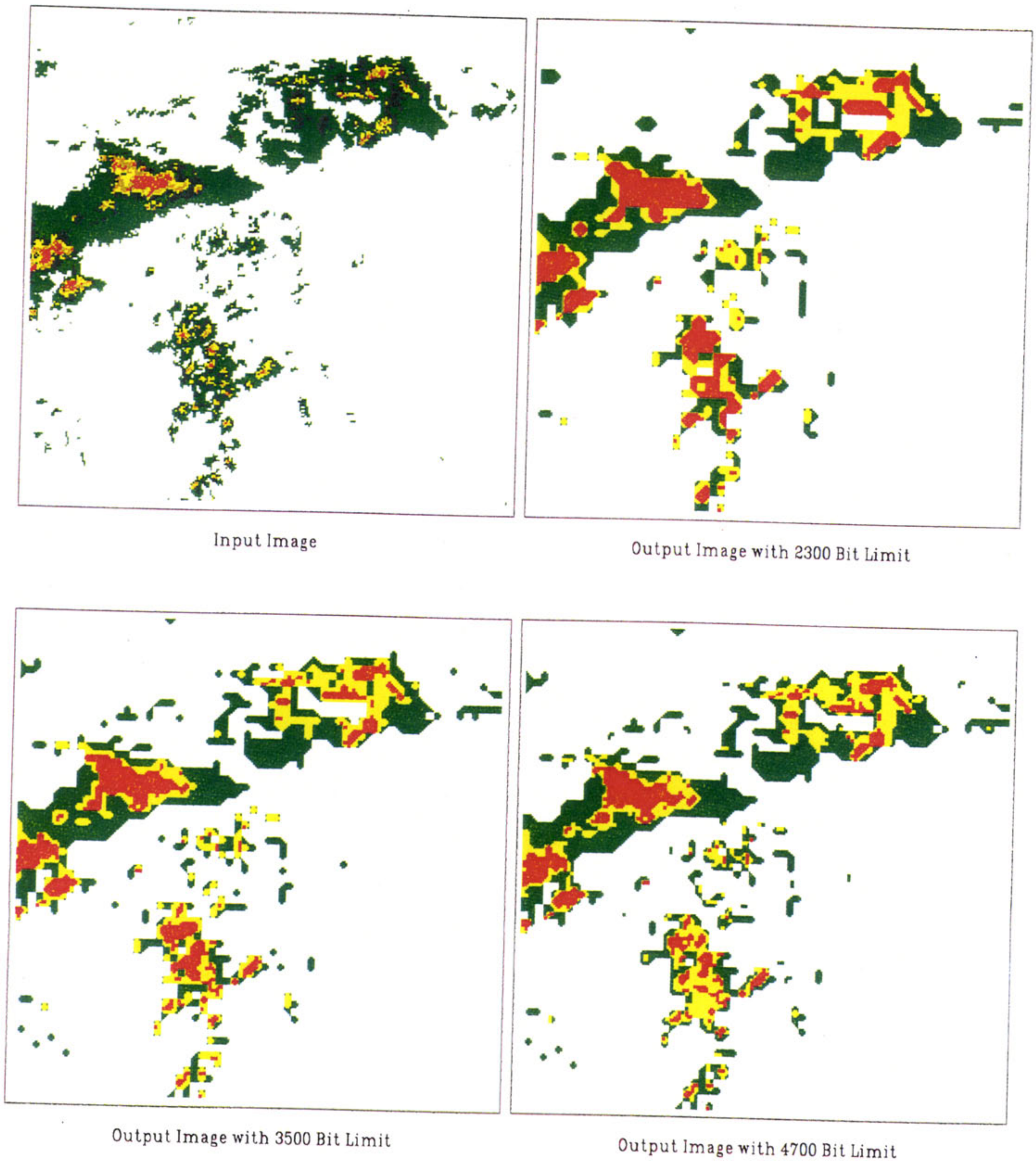

Input Image
Output Image with 2300 Bit Limit
Output Image with 3500 Bit Limit
Output Image with 4700 Bit Limit

Figure 10–5a. Output Mobile Image as a Function of Datalink Bit Limit.

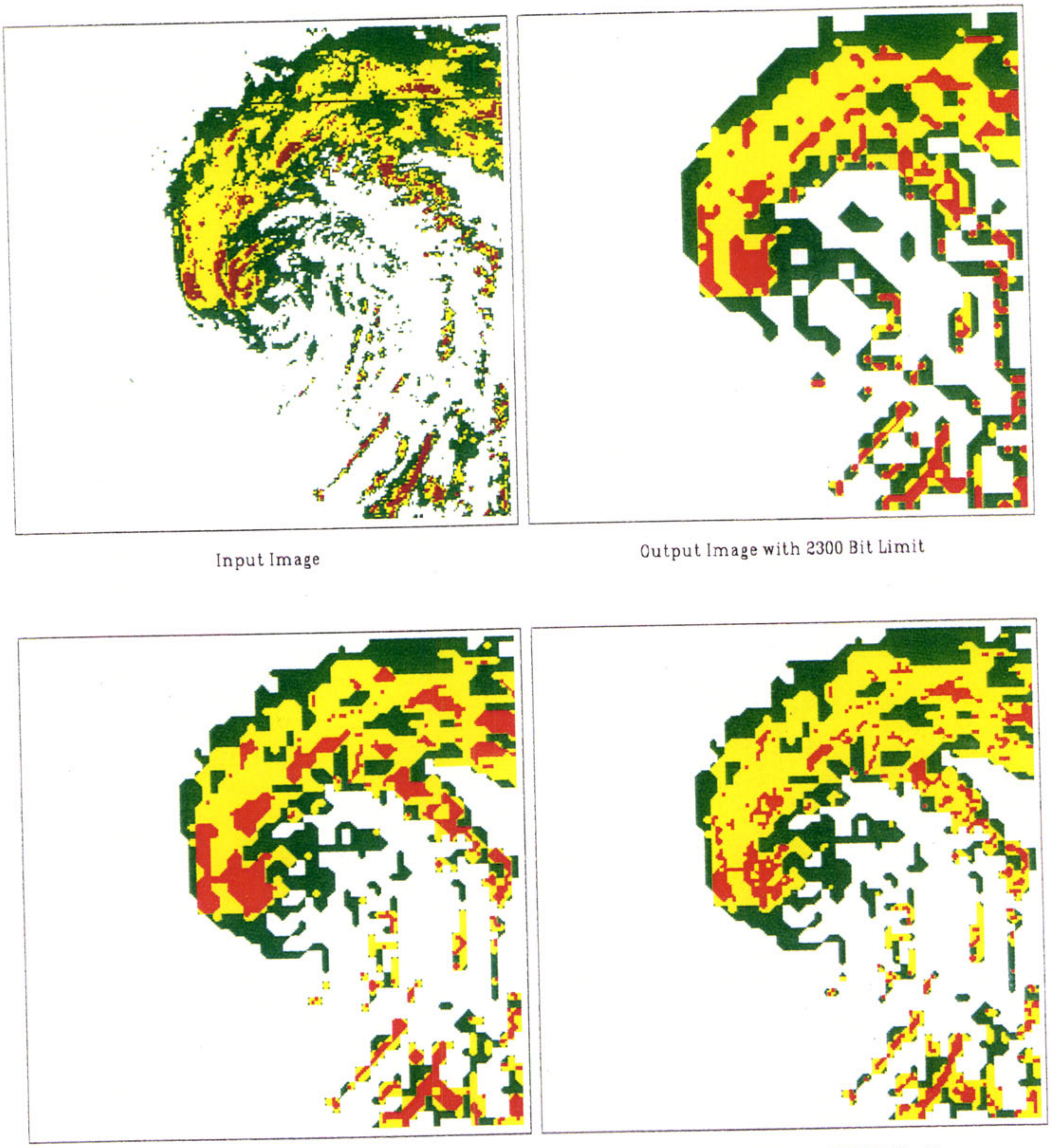

Input Image
Output Image with 2300 Bit Limit
Output Image with 3500 Bit Limit
Output Image with 4700 Bit Limit

Figure 10-5b. Output Allison Image as a Function of Datalink Bit Limit.

APPENDIX A
DEFAULT HUFFMAN ENCODING ENSEMBLES

This Appendix presents, for each image level, the 3 options available for default Huffman codeword ensembles. For each option, the codeword bitlength L and the actual codeword assignment are listed for each applicable runlength. The special "runlengths" S1 and S2 are also included, where as described in the report:

S1 indicates an "other" length run, length > L_{max}

S2 indicates a long run, length > 63

where L_{max}, also listed for each option, is the longest runlength assigned its own codeword.

Level 0

	Option 0		Option 1		Option 2	
	$L_{max} = 7$		$L_{max} = S2$		$L_{max} = 8$	
Runlength	L	Code	L	Code	L	Code
S1	4	1000	1	0	2	01
S2	1	0	1	1	2	00
1	4	1001			4	1000
2	4	1010			4	1001
3	4	1011			4	1010
4	4	1100			4	1011
5	4	1101			4	1100
6	4	1110			4	1101
7	4	1111			4	1110
8					4	1111

Level 1 and 2

	Option 0		Option 1		Option 2	
	$L_{max} = 5$		$L_{max} = S2$		$L_{max} = 7$	
Runlength	L	Code	L	Code	L	Code
S1	3	000	1	0	2	00
S2	3	001	1	1	2	01
0	3	010			4	1000
1	3	011			4	1001
2	3	100			4	1010
3	3	101			4	1011
4	3	110			4	1100
5	3	111			4	1101
6					4	1110
7					4	1111

Levels 3, 4, 5, and 6

	Option 0		Option 1		Option 2	
	$L_{max} = 2$		$L_{max} = 2$		$L_{max} = 4$	
Runlength	L	Code	L	Code	L	Code
S1	3	110	1	0	2	00
S2	3	111	3	111	3	010
0	2	00	3	100	3	011
1	2	01	3	101	3	100
2	2	10	3	110	3	101
3					3	110
4					3	111

Level 7

	Option 0		Option 1		Option 2	
	$L_{max} = 2$		$L_{max} = 2$		$L_{max} = 4$	
Runlength	L	Code	L	Code	L	Code
S1	3	110	1	0	2	00
S2	3	111	3	111	3	010
1	2	00	3	100	3	011
2	2	01	3	101	3	100
3	2	10	3	110	3	101
4					3	110
5					3	111

Note that, for levels 0 and 7, no codeword need be assigned to a runlength of 0. A runlength of 0 is used to cover multi-level weather changes through the given level. For example, a change from level 3 to level 1 requires a codeword for a length 0 run at level 2. Clearly, no change "through" level 0 or 7 is possible. For an actual image, no codeword for length 0 would be required for the highest weather level existing on that image. Thus, whenever the highest image level is 3 or greater, the default ensemble of level 7 is used for that highest level.

APPENDIX B
WEATHER-HUFFMAN (WH) PERFORMANCE-TRADEOFF PARAMETERS

This Appendix lists and defines each of the performance-tradeoff parameters currently existing in the WH algorithm software. It also lists the current default settings of each parameter, and discusses the performance effects of changing to other values.

Image Bit Limit (maxbits) Default = 3500

This parameter is the maximum number of bits that can be used to encode the weather image. Changes in the setting will increase or decrease the distortion produced in the image representation output to the pilot.

Level Renumbering (remapp) Default = 0

This parameter specifies whether the standard weather level numbering scheme should be maintained on the input image, or whether some of the levels should be grouped. The meanings of the possible values are:

0 = no renumbering of weather levels

1 = renumber the weather levels

the user must enter in order the new numbers for levels 1 through 6.

The remapping feature is useful when fewer than 6 levels of weather are desired on the output display. For example, a 3 level display — light rain, heavy rain, storms — would be produced by entering 1 followed by the 6 values 1,2,3,3,3,3, which combines together all pixels of levels 3 through 6.

Minimum Level Count for 8x8 (wt8g) Default = 32,20,6,1,1,1

These parameters specify, for each level 1 through 6, the minimum count of pixels at or above that level that require the 8x8 resolution reduced superpixel to be set to that level. For example, if 20 or more of the 64 pixels in the superpixel are of level 2 or above, and if none of the higher level thresholds are satisfied (i.e., there are fewer than 6 at or above level 3, and none at levels 4, 5, or 6), the superpixel is set to level 2. If the values are decreased, weather regions will be "averaged up" in size, and more weather "spots" will be maintained on the output image; if they are increased, the larger regions will be more faithfully represented as more bits are freed up for encoding them.

Minimum Level Count for 4x4 (wt4g) Default = 8,5,2,1,1,1

The analog for 4x4 superpixels.

Minimum Level Count for 2x2 (wt2g) Default = 2,1,1,1,1,1

The analog for 2x2 superpixels.

Provisional Level Count for 8x8 (wt8) Default = 12,6,1,1,1,1

These parameters specify, for each level 1 through 6, the provisional count of pixels at or above that level that require the 8x8 resolution reduced superpixel to be set to that level if not enough neighboring superpixels are available to adequately represent the weather region. For example, if 6 or more of the 64 pixels in the superpixel are of level 2 or above, and if none of the higher level thresholds are satisfied (i.e., there are none at levels 3, 4, 5, or 6), the superpixel is set to level 2 if less than two neighboring superpixels are set to level 2. This parameter works as a protection against the previous parameter (wt2g) causing the loss of a significant weather region. If the values are decreased, weather regions will be further "averaged up" in size, and additional weather "spots" will be maintained on the output image; if they are increased, the larger regions will be more faithfully represented as more bits are freed up for encoding them.

Provisional Level Count for 4x4 (wt4) Default = 4,2,1,1,1,1

The analog for 4x4 superpixels.

Provisional Level Count for 2x2 (wt2) Default = 1,1,1,1,1,1

The analog for 2x2 superpixels.

Extra Bit Usefulness Multiplier (mult) Default = 4,5,7,7,7,7

This parameter specifies, for each weather level, the fraction of the quadrants in an extra bit class that must require level adjustment for the class to be encoded by the Extra Bit Algorithm. For example, for level 2, a given extra bit class must satisfy:

$$\text{mult}[2] * (\text{adjusted quadrants}) \geq (\text{total quadrants})$$

for the class to be encoded. By reducing the value for a level, that level will be more accurately represented on the output image at the expense of lower levels. (Due to the possibility of multiple extra bit passes, reducing the value for a level can in some cases degrade that level as well).

REFERENCES

[1] Lelewer, D. E., and D. S. Hirschberg, "Data Compression," *ACM Computing Surveys* , 19, 3 (September 1987): 261-296.

[2] Gertz, J. L., and R. D. Grappel, "Encoding Approaches for Data Link Transmission of Weather Graphics," Lexington, MA, M.I.T. Lincoln Laboratory Project Report ATC-205, 10 December 1993.

[3] Huffman D. A., "A method for the construction of minimum-redundancy codes," *Proceedings IRE* , 40,9 (September 1952): 1098-1101.

[4] Gertz, J. L., "The Polygon-Ellipse Method of Data Compression of Weather Maps," Lexington, MA, M.I.T. Lincoln Laboratory Project Report ATC-213, 28 March 1994.