Title: FAST LOSS LESS IMAGE COMPRESSION BASED ON NEIGHBORHOOD COMPARISONS

Abstract: A real-time video compression method compares pixel value differences with neighboring pixels and replaces such pixel values with the minimum of the differences. A marker is attached to a block of pixels, such that all the pixels in that block are compared with neighbors of one direction. Such marker indicates how all the pixels in that block are compared. Interim Huffmann-tree construction is used such that one tree is used for several frames. Huffman coding is used to compress the resulting frame. A single Huffman-tree is constructed once every q-number of frames. Since the frames do not change frequently, "q" can be set, for example, to thirty and still not cause much deterioration in compression ratio. When used for real-time video compression, the frequency of Huffman-tree construction is done according to the instantaneous availability of processor time to do the construction. When more processing time is available, the Huffman-trees are computed more frequently. Such frequency variation can be implemented by first checking the input video frame buffer. If it's empty, processor time for Huffman-tree construction is available.
Fast Loss Less Image Compression Based on Neighborhood Comparisons

BACKGROUND OF THE INVENTION

TECHNICAL FIELD

The present invention relates to data compression methods, and more specifically to fast loss-less image compression based on neighborhood comparisons.

DESCRIPTION OF THE PRIOR ART

Video data is typically encoded in large blocks of memory. Even a single frame 1K x 1K can require a megabyte to describe it. So video data requires large amounts of memory to store it, and high communication bandwidths to transfer it. It therefore is highly advantageous to compress video data. But some compression methods lose information in the process. Some applications can tolerate various degrees of loss, but still others require loss-less compression.

The typical loss-less video data compression methods use differential pulse code modulation (DPCM). A current pixel value is replaced by the difference from the previous pixel. Since pixel values change rather slowly along scan lines, the difference values will near zero, making compression algorithms particularly effective. But many times the slow change between pixel values occurs in vertical or diagonal bands. So prior art methods that can only take advantage of horizontal bands in the image are compromised.

Huffman coding can be used to compress real-time video data, and such builds a Huffman-tree, and then uses the tree to recode the data. The basic idea is that frequently occurring video codes are replaced by a Huffman-code that requires fewer bits. The occasional video codes that occur infrequently can be
described with Huffman-codes that actually need more bits. The combination is an overall gain in compression.

It has been conventional to compute the Huffman-tree once regardless of any real difference resulting from the previous tree, and regardless of available process time to handle the job that instant. The computational time needed to compute the Huffman-tree can equal that needed to run the compression for one image with that tree.

SUMMARY OF THE INVENTION

In general, method embodiments of the present invention compare the pixel value differences with the neighboring pixels and replace the pixel values with the smallest differences. A marker is attached to a n-by-m block of pixels, such that all the pixels in that block are compared with neighbors of one direction. Such marker indicates how all the pixels in that block are compared. Intermittent Huffman-tree construction is used. Huffman coding is used to compress the resulting frame. A single Huffman-tree is constructed once every q-number of frames. Since the frames do not change frequently, "q" can be set to thirty and still not cause much deterioration in compression ratio. When used for real-time video compression, the frequency of Huffman-tree construction is selected according to the instantaneous availability of processor time to do the construction. When more processing time is available, the Huffman-trees are computed more frequently. Such frequency variation can be implemented by first checking the input video frame buffer for video frames to be compressed. If it's empty, processor time for Huffman-tree construction is available.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a flowchart of a video data compression embodiment of the present invention that uses four delta computations for improved performance where the image has horizontal, vertical, and/or diagonal bands;
Fig. 2 is a flowchart of a video data compression embodiment of the present invention that uses two delta computations for improved performance where the image has horizontal and/or vertical bands.

DETAILED DESCRIPTION OF THE INVENTION

The particular description below uses a block of four by four. It can be generalized to a block of \(n\) by \(m\), for any numbers \(n\) which is smaller than the number of scan lines, and \(m\) which is smaller than the number of pixels in one scan line.

Fig. 1 represents a best-neighborhood compression method embodiment of the present invention, and is referred to herein by the general reference numeral 100. A typical video stream for compression by method 100 comprises a series of video frames, each video frame comprises a number of scan lines, and each scan line includes a number of pixels. The method 100 comprises a step 102 that sets a current-scan-line address counter to the top scan line of a frame. A step 104 checks if there are less than four \((n)\) scan lines left below the current scan line. If not, a step 106 lowers the current-scan-line address counter by four \((n)\). A step 108 sets a cursor address counter to the first, or left-most pixel of the current scan line. A step 110 checks to see if there are less than four \((m)\) pixels to the right of the cursor. If so, control returns to step 104. Otherwise, a step 112 advances the cursor address counter to the right by four \((m)\). A step 114 selects, for example, a four-by-four \((n\times m)\) pixel matrix with a cursor pixel at a bottom-right corner.

A step 116 computes four pixel-differential (delta) values for the selected pixel matrix, i.e., left delta, left-above delta, above delta, and right-above delta. An evaluation measure is computed for each delta matrix resulting from four delta operations. The delta matrix with the best (lowest) evaluation measure is selected. One marker is added to indicate which delta matrix was selected. The original matrix is replaced with the selected delta matrix and its marker. Huffman coding is applied to the marker.

A step 118 applies pixel left-delta computation to the remaining scan lines below current and top most scan line. Pixel left-delta computation replaces the pixel values of the entire scan line except the left most pixel. Each pixel is
replaced by the difference between the value of that pixel and the original value of the immediately left neighbor pixel. A step 122 applies run-length coding. A step 124 Huffman codes each pixel of the whole frame.

In method embodiments of the present invention, a left delta operation returns a four-by-four (n-by-m) matrix with pixel values computed from the original matrix. For each pixel in the original matrix, a difference is calculated between a current pixel value and a left neighbor pixel value.

A left-above delta operation returns a four-by-four (n-by-m) matrix with pixel values computed from the original matrix. For each pixel in an original matrix, a difference is calculated between a current pixel value and a diagonally left and above neighbor pixel value.

An above delta operation returns a four-by-four (n-by-m) matrix with pixel values computed from the original matrix. For each pixel in the original matrix, a difference is calculated between a current pixel value and an above-neighbor pixel value.

A right-above delta operation returns a four-by-four (n-by-m) matrix with pixel values computed from the original matrix. For each pixel in the original matrix, a difference is calculated between a current pixel value and the diagonally right and above neighbor pixel value.

The Huffman coding comprises Huffman-tree building, and encoding based on the Huffman-tree. The Huffman-tree building takes substantial amount of time. So if we make the Huffman-tree for every image frame, the computation time is too long so that the algorithm does not run in real time. So compression like MPEG uses Huffman coding but uses one Huffman-tree for the entire video. This is not satisfactory in many cases since the compression ratio deteriorates. Our invention is to create the Huffman-tree once in a while. One approach is to create the Huffman-tree for once every fixed number of frames. The frequency of tree generation is computed so that the computation time for tree generation per frame is about one tenth of encoding time. Another approach is an adaptive approach. If the computer is fast and a lot of cycles are left, then we should run Huffman-tree generation more frequently to get better compression ratio. We approach this problem by looking at the input buffer. Whenever input buffer becomes empty or near empty, that means the compression system is catching up with the video input, and therefore time is left to do other things, such as tree generation. So the algorithm is to look at the input buffer
size and if the size becomes below a threshold, then Huffman-tree generation is invoked.

Fig. 2 represents a simple best-neighbor compression method embodiment of the present invention, and is referred to herein by the general reference numeral 200. Method 200 is similar to method 100, but it only computes the left-delta and above-delta. A typical video stream for compression by method 200 comprises a series of video frames, each video frame comprises a number of scan lines, and each scan line includes a number of pixels. The method 200 comprises a step 202 that sets a current-scan-line address counter to the top scan line of a frame. A step 204 checks if there are less than four (n) scan lines left below the current scan line. If not, a step 206 lowers the current-scan-line address counter by four (n). A step 208 sets a cursor address counter to the first, or left-most pixel of the current scan line. A step 210 checks to see if there are less than four (m) pixels to the right of the cursor. If so, control returns to step 204. Otherwise, a step 212 advances the cursor address counter to the right by four (m). A step 214 selects, for example, a four-by-four (n-by-m) pixel matrix with a cursor pixel at a bottom-right corner.

A step 216 computes two pixel-differential (delta) values for the selected pixel matrix, i.e., left delta and above delta. An evaluation measure is computed for each delta matrix resulting from the two delta operations. The delta matrix with the best (lowest) evaluation measure is selected. One marker is added to indicate which delta matrix was selected. The original matrix is replaced with the selected delta matrix and its marker. Huffman coding is applied to the marker.

A step 218 applies pixel left-delta computation to the remaining scan lines below current and top most scan line. A step 222 applies run-length coding. A step 224 Huffman codes each scan line of the whole frame.

The evaluation measures mentioned for steps 116 and 216 are preferably equal to the sum of the absolute values of all the pixels in a four-by-four (n-by-m) resulting delta matrix. If some pixel value differences cannot be computed because the pixel is at the edge of a frame, the evaluation value does not exist or is indicated. Consider the compression example represented in Table I.
A first matrix to be processed is presented in Table II.

In the second row and second column there is a current value of 17. A left delta computes 17-12=5. An above-left delta computes 17-14=3. An above delta computes 17-18=-1. An above-right delta computes 17-22=-5. Each of the sixteen current pixel positions are similarly computed to generate a left delta matrix, a left-above delta matrix, an above delta matrix, and a right-above delta matrix. A completed left delta matrix produces an evaluation measure = nil for the first matrix in Table II. The computation of a left-above delta matrix also produces an evaluation measure = nil.

An above delta matrix results in an evaluation measure = 27, as in Table III.

A right-above delta matrix, as in Table IV, results in an evaluation measure = 83.
TABLE IV
Right-Above Delta Matrix

<table>
<thead>
<tr>
<th>-2</th>
<th>-1</th>
<th>9</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6</td>
<td>-5</td>
<td>-7</td>
<td>7</td>
</tr>
<tr>
<td>-7</td>
<td>-1</td>
<td>4</td>
<td>-2</td>
</tr>
<tr>
<td>-10</td>
<td>-4</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Therefore, the evaluation measure comparison shows that the above delta (A) is best for the first 4x4 pixel matrix. A second matrix is represented in Table V.

TABLE V
Second Exemplary 4x4 Matrix

<table>
<thead>
<tr>
<th>10</th>
<th>12</th>
<th>15</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>11</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

A left delta matrix of such second matrix, as in Table VI, results in evaluation measure = 38.

TABLE VI
Left Delta Matrix

<table>
<thead>
<tr>
<th>-3</th>
<th>2</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>-2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>-4</td>
<td>2</td>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>-4</td>
<td>-1</td>
<td>-5</td>
<td>3</td>
</tr>
</tbody>
</table>

A left above delta matrix results in an evaluation measure = 59.

TABLE VII
Left Above Delta Matrix

<table>
<thead>
<tr>
<th>-3</th>
<th>4</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-10</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
<td>-7</td>
<td>-1</td>
</tr>
</tbody>
</table>
An above delta matrix results in an evaluation measure = 51.

TABLE VIII
Above Delta Matrix

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-6</td>
<td>-2</td>
<td>-8</td>
<td>-8</td>
</tr>
<tr>
<td>1</td>
<td>-2</td>
<td>-4</td>
<td>-3</td>
</tr>
</tbody>
</table>

A right above delta matrix results in an evaluation measure = nil.

Therefore, the evaluation measure comparison shows that the left delta (L) is best for the second 4x4 pixel matrix. Pixel left delta of the first scan line is represented in Table IX.

TABLE IX
Pixel Left Delta Of The First Scan Line

| 15 | 1  | 3  | -6 | -5 | 1  | 3  | 2  |

Table X represents a result.

TABLE X
Result

<table>
<thead>
<tr>
<th>15</th>
<th>1</th>
<th>3</th>
<th>-6</th>
<th>-5</th>
<th>1</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>-3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>4</td>
<td>-4</td>
<td>-2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-6</td>
<td>-4</td>
<td>2</td>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>-4</td>
<td>-1</td>
<td>-5</td>
<td>3</td>
</tr>
</tbody>
</table>

Neighbor matrix markers are:

A | L
Huffman compression processing results in the Huffman-tree of Table XI.

<table>
<thead>
<tr>
<th>pixel value</th>
<th>Huffman code</th>
<th># of occurrences</th>
<th>total bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00010</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>-1</td>
<td>0110</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>-2</td>
<td>100</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>-3</td>
<td>0111</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>00001</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>-4</td>
<td>0011</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>-5</td>
<td>0010</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>-6</td>
<td>00011</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>00000</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>40</td>
<td>133</td>
</tr>
</tbody>
</table>

The compression ratio for this example is $(200-133)/200=34\%$

In general, method embodiments of the present invention compare the pixel value differences with the neighboring pixels and replace the pixel values with the smallest of the several differences. A marker is attached to a block of pixels, such that all the pixels in that block are compared with neighbors of one direction. Such marker indicates how all the pixels in that block are compared. Intermittent Huffman-tree construction is used. Huffman coding is used to compress the resulting frame. A single Huffman-tree is constructed once every q-number of frames. Since the frames do not change frequently, "q" can be set to thirty and still not cause much deterioration in compression ratio. When used for real-time video compression, the frequency of Huffman-tree construction is selected according to the instantaneous availability of processor time to do the construction. When more processing time is available, the Huffman-trees are computed more frequently. Such frequency variation can be implemented by first checking the input video frame buffer. If it's empty, processor time for Huffman-tree construction is available.

Loss-less compression of images has been implemented using the fact that color and brightness of the images change gradually on the surface so the
values of neighboring pixels do not change drastically. In other words, each pixel is constrained by its neighbors because the image data is not completely arbitrary.

Many compression methods are based on differential pulse code modulation (DPCM), which creates a "predictor" from some linear function of the pixels preceding a point in a scan line. The actual pixel value is replaced with a difference between the actual pixel value and the predictor. Such predictor often equals the value of the immediately preceding pixel. In this way the whole pixel values of the images are close to zero, and can be compressed either by using fewer bits to represent pixels or using Huffman coding to compress.

If each pixel were assigned its own marker, the total amount of data needed for such markers can be substantial, e.g., one quarter of the total. Such overhead can be significantly reduced by dividing the image into a number of small n-by-m blocks. All the pixels in a block are assigned one marker. The one marker is attached to each block that indicates which of the four comparisons is selected. So all the pixels in each block are compared their neighboring pixels in a single way. The shape and size of the block can take any form. In one embodiment of the present invention, the n-by-m block dimension was set to be a four-by-four square matrix.

After all the absolute pixel values have been reduced to small-numeric differential values, Huffman coding is used to compress the frame. But the Huffman coding of video can be time-expensive. Huffman coding consists of two phases, one to create the Huffman-tree for encoding, and a next to encode the pixels. The first-phase Huffman-tree-generation processing time can require as much as the second-phase time needed to encode the whole image. In order to reduce the time we perform tree generation once for several frames. Since the images do not change drastically between sequential frames, this often does not degrade the compression ratio.

Two different schemes can be used to set the frequency of Huffman-tree generation, one fixed and one variable. A fixed number of frames can be made to share a single Huffman-tree, e.g., thirty frames. The variable method generates Huffman trees according to the processor power available at the time. The availability of processing power is assessed by looking at the number of entries in an input buffer that need to be compressed. Little or no
work to do means more time can be spent on generating more frequent Huffman trees.

Although the invention is preferably described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.
CLAIMS

1. A video compression method wherein a video stream comprises a series of video frames, each video frame comprises a number of scan lines, and each scan line includes a number of pixels, the method comprising the steps of:
   finding a left-delta value that represents a difference between a current pixel value and a left-adjacent pixel value;
   finding a left-above-delta value that represents a difference between said current pixel value and a value of an adjacent pixel above and to the left in a preceding scan line;
   finding an above-delta value that represents a difference between said current pixel value and a value of an adjacent pixel above in said preceding scan line;
   finding a right-above-delta value that represents a difference between said current pixel value and a value of an adjacent pixel above and to the right in said preceding scan line;
   calculating a minimum of an absolute value for each of said left-delta value, left-above-delta value, above-delta value, and right-above-delta value;
   replacing said current pixel value with one of said left-delta value, left-above-delta value, above-delta value, and right-above-delta value found to be closest to a zero value in the step of calculating a minimum; and
   marking a replaced pixel value with a marker to indicate which one of said left-delta value, left-above-delta value, above-delta value, and right-above-delta value was found to be closest to a zero value in the step of calculating a minimum.

2. The method of claim 1, further comprising the step of:
   repeating the preceding steps for each pixel in a video frame.

3. The method of claim 2, further comprising the step of:
   compressing the replaced pixel values using run-length coding.

4. The method of claim 3, further comprising the step of:
   Huffman-encoding the pixel values compressed by run-length coding and markers.
5. The method of claim 4, wherein:
   the Huffman-encoding includes the steps of computing a Huffman-
   tree, and using said tree.

5. The method of claim 5, wherein:
   the Huffman-encoding computes said Huffman-tree once only q-
   number of video frames.

7. The method of claim 5, wherein:
   the Huffman-encoding computes said Huffman-tree at a frequency
   that is dependent on available processor overhead.

8. A video compression method wherein a video stream comprises a series
   of video frames, each video frame comprises a number of scan lines, and each
   scan line includes a number of pixels, the method comprising the steps of:
   finding a left-delta value that represents a difference between a
   current pixel value and a left-adjacent pixel value;
   finding an above-delta value that represents a difference between
   said current pixel value and a value of an adjacent pixel above in a preceding
   scan line;
   calculating a minimum of an absolute value for each of said left-delta
   value and above-delta value;
   replacing said current pixel value with one of said left-delta value
   and above-delta value found to be closest to a zero value in the step of
   calculating a minimum; and
   marking a replaced pixel value with a marker to indicate which one
   of said left-delta value and above-delta value was found to be closest to a zero
   value in the step of calculating a minimum.

9. The method of claim 8, further comprising the step of:
   repeating the preceding steps for each pixel in a video frame.

10. The method of claim 9, further comprising the step of:
    compressing the replaced pixel values using run-length coding.

11. The method of claim 10, further comprising the step of:
Huffman-encoding the pixel values compresses by run-length coding and markers.

12. The method of claim 11, wherein:
   the Huffman encoding includes steps of computing a Huffman tree.

13. The method of claim 12, wherein:
   the Huffman-encoding computes said Huffman-tree once only q-
   number of video frames.

14. The method of claim 12, wherein:
   the Huffman-encoding computes said Huffman-tree at a frequency
   that is dependent on available processor overhead.

15. A video compression method wherein a video stream comprises a series
    of video frames, each video frame comprises a number of scan lines, and each
    scan line includes a number of pixels, the method comprising the steps of:
    finding a left-delta n-by-m pixel matrix for a current n-by-m pixel
    matrix;
    finding a above-delta n-by-m pixel matrix for said current n-by-m
    pixel matrix;
    calculating an evaluation measure for each of said n-by-m delta
    matrices, said evaluation measure is equal to the sum of the absolute values of all
    the pixels in an n-by-m delta matrix;
    replacing said current n-by-m pixel matrix with one of said left-delta
    and above-delta pixel matrices whose evaluation measure was found to be
    closest to a zero value; and
    marking a replaced n-by-m pixel matrix with a marker to indicate
    which one of said left-delta and above-delta evaluation measures was found to
    be closest to a zero value.

16. The method of claim 15, further comprising the step of:
    repeating the preceding steps for each pixel in a video frame.

17. The method of claim 16, further comprising the step of:
    compressing the replaced pixel values using run-length coding.
18. The method of claim 17, further comprising the step of:
   Huffman-encoding the pixel values compresses by run-length coding and markers.

19. The method of claim 18, wherein:
   the Huffman encoding includes steps of computing a Huffman tree.

20. The method of claim 19, wherein:
   the Huffman-encoding computes said Huffman-tree once only q-number of video frames.

21. The method of claim 19, wherein:
   the Huffman-encoding computes said Huffman-tree at a frequency that is dependent on available processor overhead.

22. A video compression method wherein a video stream comprises a series of video frames, each video frame comprises a number of scan lines, and each scan line includes a number of pixels, the method comprising the steps of:
   finding a left-delta n-by-m pixel matrix for a current n-by-m pixel matrix;
   finding a left-above-delta n-by-m pixel matrix for a current n-by-m pixel matrix;
   finding an above-delta n-by-m pixel matrix for a current n-by-m pixel matrix;
   finding a right-above-delta n-by-m pixel matrix for a current n-by-m pixel matrix;
   calculating an evaluation measure for each of said n-by-m delta matrices, said evaluation measure is equal to the sum of the absolute values of all the pixels in an n-by-m delta matrix;
   replacing said current n-by-m pixel matrix with one of said left-delta, left-above-delta, above-delta, and right-above-delta pixel matrices whose evaluation measure was found to be closest to a zero value; and
   marking a replaced n-by-m pixel matrix with a marker to indicate which one of said left-delta, left-above-delta, above-delta, and right-above-delta evaluation measures was found to be closest to a zero value.
23. The method of claim 22, further comprising the step of:
   repeating the preceding steps for each pixel in a video frame.

24. The method of claim 23, further comprising the step of:
   compressing the replaced pixel values using run-length coding.

25. The method of claim 24, further comprising the step of:
   Huffman-encoding the pixel values compressed by run-length
   coding and markers.

26. The method of claim 25, wherein:
   the Huffman-encoding includes the steps of computing a Huffman-
   tree, and using said tree.

27. The method of claim 26, wherein:
   the Huffman-encoding computes said Huffman-tree once only q-
   number of video frames.

28. The method of claim 26, wherein:
   the Huffman-encoding computes said Huffman-tree at a frequency
   that is dependent on available processor overhead.
Fig. 1

100

102

current ← top line

104

less than 4 scan lines below current

Y

N

106

108

current ← current - 4

cursor ← first pixel of the current scan line

110

less than four more pixels?

Y

N

112

114

cursor ← cursor + 4

select 4x4 pixel matrix

116

compute four deltas

118

apply pixel left-delta operation to top-most scan line and all the scan lines below current

122

run length coding

124

Huffman code each scan line in whole frame
Fig. 2

current ← top line

less than 4 scan lines below current?

Y

N

current ← current - 4

cursor ← first pixel of the current scan line

less than four more pixels?

Y

N

cursor ← cursor + 4

select 4x4 pixel matrix

compute two deltas

apply pixel left-delta operation to top-most scan line and all the scan lines below current

run length coding

Huffman code each scan line in whole frame