Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 3014052 C 2020/12/15

(11)(21) 3 014 052

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 2011/07/14

(41) Mise à la disp. pub./Open to Public Insp.: 2012/01/19

(45) Date de délivrance/Issue Date: 2020/12/15

(62) Demande originale/Original Application: 2 934 184

(30) Priorités/Priorities: 2010/07/14 (US61/364,322);
2010/09/30 (US61/388,541)

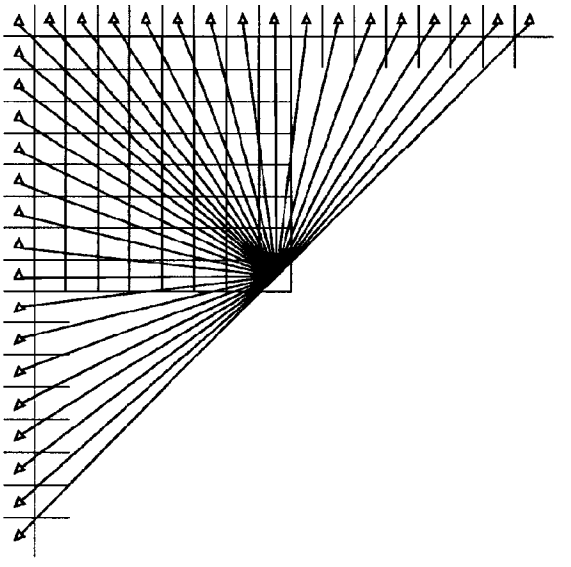
(51) Cl.Int./Int.Cl. *H04N 19/159* (2014.01),
H04N 19/174 (2014.01), *H04N 19/182* (2014.01),
H04N 19/593 (2014.01)

(72) Inventeurs/Inventors:
BOSSEN, FRANK JAN, US;
TAN, THIOW KENG, SG

(73) Propriétaire/Owner:
NTT DOCOMO, INC., JP

(74) Agent: SMART & BIGGAR LLP

(54) Titre : PREDICTION INTERNE A FAIBLE COMPLEXITE POUR CODAGE VIDEO
(54) Title: LOW-COMPLEXITY INTRA PREDICTION FOR VIDEO CODING



(57) Abrégé/Abstract:
The present invention provides a unique intra prediction process which improves the efficiency of video coding. H.264/AVC uses reference pixels in a horizontal boundary located immediately above a target block to be predicted and reference pixels in a vertical boundary located immediately left of the target block. In the present invention, at least some of one of an array of horizontal boundary pixels and an array of vertical boundary pixels are retrieved. Then, the retrieved pixels are added to the other boundary pixels to extend the array thereof. Intra prediction is performed, based solely on the extended array of boundary pixels.

ABSTRACT

The present invention provides a unique intra prediction process which improves the efficiency of video coding. H.264/AVC uses reference pixels in a horizontal boundary located immediately above a target block to be predicted and reference pixels in a vertical boundary located immediately left of the target block. In the present invention, at least some of one of an array of horizontal boundary pixels and an array of vertical boundary pixels are retrieved. Then, the retrieved pixels are added to the other boundary pixels to extend the array thereof. Intra prediction is performed, based solely on the extended array of boundary pixels.

LOW-COMPLEXITY INTRA PREDICTION FOR VIDEO CODING

The present application is a divisional of Canadian Patent Application Serial No. 2,934,184, which is a divisional of Canadian National Phase Patent Application Serial No. 2,804,762 filed July 14, 2011.

BACKGROUND OF THE INVENTION

1. Technical Field text

[0001] The present invention relates to video coding and in particular to intra-frame prediction in which a block of sample is predicted, using previously encoded and reconstructed pixels from the same video frame.

2. Background Information

[0002] Digital video requires a large amount of data to represent each and every frame of a digital video sequence (e.g., series of frames) in an uncompressed manner. It is not feasible for most applications to transmit uncompressed digital video across computer networks because of bandwidth limitations. In addition, uncompressed digital video requires a large amount of storage space. The digital video is normally encoded in some manner to reduce the storage requirements and reduce the bandwidth requirements.

[0003] One technique for encoding digital video is inter-frame prediction, or inter prediction. Inter prediction exploits temporal redundancies among different frames. Temporally adjacent frames of video typically include blocks of pixels, which remain substantially the same. During the encoding process, a motion vector interrelates the movement of a block of pixels in one frame to a block of similar pixels in another frame. Accordingly, the system is not required to encode the block of pixels twice, but rather encodes the block of pixels once and provides a motion vector to predict the other block of pixels.

[0004] Another technique for encoding digital video is intra-frame prediction or intra prediction. Intra prediction encodes a frame or a portion thereof without reference to pixels in other frames. Intra prediction exploits spatial redundancies among blocks of pixels within a frame. Because spatially adjacent blocks of pixels generally have similar attributes, the efficiency of the coding process is improved by referencing the spatial correlation between adjacent blocks. This

correlation may be exploited by prediction of a target block based on prediction modes used in adjacent blocks.

SUMMARY OF THE INVENTION

[0004a] According to an aspect of the present disclosure, there is provided a video encoding method comprising computer executable steps executed by a processor of a video encoder to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation comprising: obtaining a value of an inverse angle parameter, corresponding to the intra prediction angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles; identifying at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array; adding the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*, where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

[0004b] According to an aspect of the present disclosure, there is provided a video decoding method comprising computer executable steps executed by a processor of a video decoder to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation comprises: obtaining a value of an inverse angle parameter, corresponding to the intra prediction angle,

from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles; identifying at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array; adding the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*, where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

[0004c] According to an aspect of the present disclosure, there is provided a video encoder comprising a processor of a computer system and a memory that stores programs executable by the processor to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation implemented by the processor to: obtain a value of an inverse angle parameter, corresponding to the intra predicition angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles; identify at least some of one of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array; add the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and use only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*, where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

[0004d] According to an aspect of the present disclosure, there is provided a video decoder comprising a processor of a computer system and a memory that stores programs executable by the processor to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation implemented by the processor to: obtain a value of an inverse angle parameter, corresponding to the intra predicition angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles; identify at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array; add the identified at least some of the vertical pixels as horizontal boundary pixels to the extension of the extended horizontal array; and using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*, where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

[0005] The present invention provides a unique intra prediction process which improves the efficiency of video coding. H.264/AVC uses reference pixels in a horizontal boundary located immediately above a target block to be predicted and reference pixels in a vertical boundary located immediately left of the target block. In the present invention, at least some of either an array of horizontal boundary pixels or an array of vertical boundary pixels are retrieved. Then, the retrieved pixels are added to the other boundary pixels to extend the array thereof. Intra prediction is performed, based solely on the extended array of boundary pixels. In an embodiment of the present invention, at least some of the vertical boundary pixels are retrieved and added to the horizontal boundary pixels to extend the array thereof.

[0006] The present invention eliminates the decision process of selecting either the horizontal boundary or the vertical boundary from which reference pixels are retrieved. The present invention also eliminates the recurring process of calculating a position of the vertical

boundary intersecting with a prediction direction, wherein the recurring calculation process typically includes a divisional operation. Elimination of these processes enables the intra prediction process to be implemented on Single-Instruction Multiple Data (SIMD) architectures, thereby improving the computational efficiency of video coding.

[0007] In an embodiment according to the present invention, at least some of the pixels among the vertical boundary pixels are retrieved, using a vertical pixel identifier which is expressed by

$$[\frac{size \times col}{angle}],$$

where *size* represents a size of a target block to be predicted, *angle* represents a prediction direction and *col* is a counter which is decremented by 1 from -1 to the angle. The retrieved pixels are added to the horizontal pixels at a location identified by a horizontal pixel identifier [*col*].

[0008] In another embodiment, in retrieving at least some of vertical boundary pixels, *InvAngle* is calculated from

$$\frac{N \times size}{angle},$$

where N is an integer power of 2. Then, at least some of the pixels among the vertical boundary pixels are retrieved, using a vertical pixel identifier which is expressed by $[col \times InvAngle >> \log_2 N]$. The retrieved pixels are added to the horizontal pixels at a location identified by a horizontal pixel identifier [*col*].

[0009] In another embodiment, *InvAngle* is obtained from a look-up table which lists values of *InvAngle* in relation to the values of *angle*.

[0010] In another embodiment, a pixel is identified among the vertical boundary pixels, using a vertical pixel identifier [*row*], where *row* is a counter which is incremented by 1 from 0 to *size*. The retrieved pixel is added to the horizontal boundary pixels at a location identified by a horizontal pixel identifier [*int* + 1], where *int* is an integer representation of a position of a pixel intersecting with a prediction direction.

[0011] The present invention also provides an encoder and a decoder which implement an intra prediction operation in which at least some of either an array of horizontal boundary pixels or an array of vertical boundary pixels are retrieved. Then, the retrieved pixels are added to the other boundary pixels to extend the array thereof. Intra prediction is performed, based solely on the extended array of boundary pixels.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] FIG. 1 is a block diagram showing an exemplary hardware architecture on which the present invention may be implemented.

[0013] FIG. 2 is a block diagram showing a general view of a video encoder to which the present invention may be applied.

[0014] FIG. 3 is a block diagram showing a general view of a video decoder to which the present invention may be applied.

[0015] FIG. 4 is a block diagram showing the functional modules of an encoder according an embodiment of the present invention.

[0016] FIG. 5 is a flowchart showing an intra prediction process performed by an intra prediction module of the embodiment of the present invention.

[0017] FIG. 6 is a block diagram showing the functional modules of a decoder according to an embodiment of the present invention.

[0018] FIG. 7 is a diagram showing prediction directions illustrating Intra_4×4 prediction modes supported in H.264/AVC.

[0019] FIG. 8 is a diagram showing the prediction directions proposed in Document No. JCT-VC A119 (Authors: Kemal Ugur, et al.; Title: Description of video coding technology proposal by Tandberg, Nokia, Ericsson; JCT-VC of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 1st Meeting at Dresden, Germany, 15-23 April, 2010).

[0020] FIG. 9 is a flowchart showing the process, proposed in JCT-VC A119, of generating a predicted block along one of the prediction directions shown in FIG. 7.

[0021] FIG. 10 is a flowchart showing the process of low complexity intra prediction performed according to an embodiment of the present invention.

[0022] FIG. 11A is a schematic view showing a prediction block and arrays of horizontal and vertical boundary pixels.

[0023] FIG. 11B is a schematic view showing an array of horizontal boundary pixels extended with vertical boundary pixels.

[0024] FIG. 12 is a flowchart showing the process of extending an array of horizontal boundary pixels performed according to an embodiment of the present invention.

[0025] FIG. 13 is a flowchart showing another embodiment of extending an array of horizontal boundary pixels.

[0026] FIG. 14 a flowchart showing the process of low complexity intra prediction performed according to another embodiment of the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS AND THE PRESENTLY PREFERRED EMBODIMENTS

[0027] FIG. 1 shows an exemplary hardware architecture of a computer 100 on which the present invention may be implemented. Please note that the hardware architecture shown in FIG. 1 may be common in both a video encoder and a video decoder which implement the embodiments of the present invention. The computer 100 includes a processor 101, memory 102, storage device 105, and one

or more input and/or output (I/O) devices 106 (or peripherals) that are communicatively coupled via a local interface 107. The local interface 105 can be, for example, but not limited to, one or more buses or other wired or wireless connections, as is known in the art.

[0028] The processor 101 is a hardware device for executing software, particularly that stored in the memory 102. The processor 101 can be any custom made or commercially available processor, a central processing unit (CPU), an auxiliary processor among several processors associated with the computer 100, a semiconductor based microprocessor (in the form of a microchip or chip set), or generally any device for executing software instructions.

[0029] The memory 102 comprises a computer readable medium which can include any one or combination of volatile memory elements (e.g., random access memory (RAM, such as DRAM, SRAM, SDRAM, etc.)) and nonvolatile memory elements (e.g., ROM, hard drive, tape, CDROM, etc.). Moreover, the memory 102 may incorporate electronic, magnetic, optical, and/or other types of storage media. A computer readable medium can be any means that can store, communicate, propagate or transport the program for use by or in connection with the instruction execution system, apparatus or device. Please note that the memory 102 can have a distributed architecture, where various components are situated remote from one another, but can be accessed by the processor 101.

[0030] The software 103 in the memory 102 may include one or more separate programs, each of which contains an ordered listing of executable instructions for implementing logical functions of the computer 100, as described below. In the example of FIG. 1, the software 103 in the memory 102 defines the computer 100's video encoding or video decoding functionality in accordance with the present invention. In addition, although not required, it is possible for the memory 102 to contain an operating system (O/S) 104. The operating system 104 essentially controls the execution of computer programs and provides scheduling, input-output control, file and data management, memory management, and communication control and related services.

[0031] The storage device 105 of the computer 100 may be one of many different types of storage device, including a stationary storage device or portable

storage device. As an example, the storage device 105 may be a magnetic tape, disk, flash memory, volatile memory, or a different storage device. In addition, the storage device 105 may be a secure digital memory card or any other removable storage device 105.

[0032] The I/O devices 106 may include input devices, for example, but not limited to a touch screen, a keyboard, mouse, scanner, microphone or other input device. Furthermore, the I/O devices 106 may also include output devices, for example, but not limited to a display or other output devices. The I/O devices 106 may further include devices that communicate via both inputs and outputs, for instance, but not limited to a modulator/demodulator (modem; for accessing another device, system, or network), a radio frequency (RF), wireless or other transceiver, a telephonic interface, a bridge, a router or other devices that function both as an input and an output.

[0033] As is well known by those having ordinary skill in the art, video compression is achieved by removing redundant information in a video sequence. Many different video coding standards exist, examples of which include MPEG-1, MPEG-2, MPEG-4, H.261, H.263, and H.264/AVC. It should be noted that the present invention is not intended to be limited in application of any specific video coding standard. However, the following description of the present invention is provided, using the example of H.264/AVC standard. H.264/AVC is the newest video coding standard and achieves a significant performance improvement over the previous coding standards such a MPEG-1, MPEG-2, H.261 and H.263.

[0034] In H.264/AVC, each frame or picture of a video can be broken into several slices. The slices are then divided into blocks of 16×16 pixels called macroblocks, which can then be further divided into blocks of 8×16, 16×8, 8×8, 4×8, 8×4, down to 4×4 pixels. There are five types of slices supported by H.264/AVC. In I slices, all the macroblocks are coded using intra prediction. In P slices, macroblocks can be coded using intra or inter prediction. P slices allow only one motion compensated prediction (MCP) signal per macroblock to be used. In B slices, macroblocks can be coded using intra or inter prediction. Two MCP signals may be used per prediction. SP slices allow P slices to be switched

between different video streams efficiently. An SI slice is an exact match for an SP slice for random access or error recovery, while using only intra prediction.

[0035] FIG. 2 shows a general view of a video encoder to which the present invention may be applied. The blocks shown in the figure represent functional modules realized by the processor 101 executing the software 103 in the memory 102. A picture of video frame 200 is fed to a video encoder 201. The video encoder treats the picture 200 in units of macroblocks 200A. Each macroblock contains several pixels of picture 200. On each macroblock a transformation into transform coefficients is performed followed by a quantization into transform coefficient levels. Moreover, intra prediction or inter prediction is used, so as not to perform the coding steps directly on the pixel data but on the differences of same to predicted pixel values, thereby achieving small values which are more easily compressed.

[0036] For each slice, the encoder 201 generates a number of syntax elements, which form a coded version of the macroblocks of the respective slice. All residual data elements in the syntax elements, which are related to the coding of transform coefficients, such as the transform coefficient levels or a significance map indicating transform coefficient levels skipped, are called residual data syntax elements. Besides these residual data syntax elements, the syntax elements generated by the encoder 201 contain control information syntax elements containing control information as to how each macroblock has been encoded and has to be decoded, respectively. In other words, the syntax elements are dividable into two categories. The first category, the control information syntax elements, contains the elements related to a macroblock type, sub-macroblock type and information on prediction modes both of a spatial and temporal types, as well as slice-based and macroblock-based control information, for example. In the second category, all residual data elements, such as a significance map indicating the locations of all significant coefficients inside a block of quantized transform coefficients and the values of the significant coefficients, which are indicated in units of levels corresponding to the quantization steps, are combined and become residual data syntax elements.

[0037] The encoder 201 comprises an entropy coder which encodes syntax elements and generates arithmetic codewords for each slice. When generating the arithmetic codewords for a slice, the entropy coder exploits statistical dependencies among the data values of syntax elements in the video signal bit stream. The encoder 201 outputs an encoded video signal for a slice of picture 200 to a video decoder 301 shown in FIG. 3.

[0038] FIG. 3 shows a general view of a video decoder to which the present invention may be applied. Likewise, the blocks shown in the figure represent functional modules realized by the processor 101 executing the software 103 in the memory 102. The video decoder 301 receives the encoded video signal and first entropy-decodes the signal back into the syntax elements. The decoder 301 uses the syntax elements in order to reconstruct, macroblock by macroblock and then slice after slice, the picture samples 300A of pixels in the picture 300.

[0039] FIG. 4 shows the functional modules of the video encoder 201. These functional modules are realized by the processor 101 executing the software 103 in the memory 102. An input video picture is a frame or a field of a natural (uncompressed) video image defined by sample points representing components of original colors, such as chrominance ("chroma") and luminance ("luma") (other components are possible, for example, hue, saturation and value). The input video picture is divided into macroblocks 400 that each represent a square picture area consisting of 16×16 pixels of the luma component of the picture color. The input video picture is also partitioned into macroblocks that each represent 8×8 pixels of each of the two chroma components of the picture color. In general encoder operation, inputted macroblocks may be temporally or spatially predicted using inter or intra prediction. It is however assumed for the purpose of discussion that the macroblocks 400 are all I-slice type macroblocks and subjected only to intra prediction.

[0040] Intra prediction is accomplished at an intra prediction module 401, the operation of which will be discussed below in detail. The intra prediction module 401 generates a prediction block 402 from horizontal and vertical boundary pixels of neighboring blocks, which have previously been encoded, reconstructed, and stored in a frame memory 403. A residual 404 of the prediction block 402, which

is the difference between a target block 400 and the prediction block 402, is transformed, scaled and quantized at a transformation/quantization module 405, using methods and techniques known to those of skill in the video coding field. Quantized transform coefficients 406 are then entropy-coded at an entropy coding module 407 and transmitted (together with other information relating to the intra prediction) as an encoded video signal 408.

[0041] The video encoder 201 contains decoding functionality to perform intra prediction on target blocks. The decoding functionality comprises an inverse quantization/transformation module 409, which performs inverse quantization and inverse transformation on the quantized transform coefficients 406 to produce the decoded prediction residual 410, which is added to the prediction block 402. The sum of decoded prediction residual 410 and prediction block 402 is a reconstructed block 411, which is stored in the frame memory 403 and will be read therefrom and used by the intra prediction module 401 to generate a prediction block 402 for decoding of a next target block 400.

[0042] FIG. 5 is a flowchart showing processes performed by the intra prediction module 401. In accordance with the H.264/AVC Standard, intra prediction involves predicting each pixel of the target block 400 under a plurality of prediction modes, using interpolations of boundary pixels ("reference pixels") of neighboring blocks previously encoded and reconstructed. The prediction modes are identified by positive integer numbers 0, 1, 2... each associated with a different instruction or algorithm for predicting specific pixels in the target block 400. The intra prediction module 401 runs intra prediction under the respective prediction modes and generates different prediction blocks. Under a full search ("FS") algorithm, each of the generated prediction blocks is compared to the target block 400 to find the optimum prediction mode, which minimizes the prediction residual 404 or produces a lesser prediction residual 404 among the prediction modes. The identification of the optimum prediction mode is compressed and sent to the decoder 301 with other control information syntax elements.

[0043] Each prediction mode may be described by a general direction of prediction as described verbally (i.e., horizontal up, vertical and diagonal down left). A prediction direction may be described graphically by an angular direction

which is expressed through a diagram with arrows such as shown in FIG. 7. In this type of diagram, each arrow may be considered representing a prediction direction or a prediction mode. The angle corresponding to a prediction mode has a general relationship to the direction from the weighted average location of the reference pixels used to predict a target pixel to the target pixel location. Please note that the prediction modes include a DC prediction mode which is not associated with any prediction direction and, thus, cannot be described graphically in the diagram unlike the other prediction modes. In the DC prediction mode, the prediction block 402 is generated such that each pixel in the prediction block 402 is set uniformly to the mean value of the reference pixels.

[0044] Turning back to FIG. 5, the prediction mode is initialized in Step 501. It is then determined in Step 502 whether the prediction mode indicates the DC prediction. If it does, the flow advances to Step 503, where a DC prediction block 402 is generated with the mean value of the reference pixels in Step 503. If the prediction mode indicates otherwise, a prediction block 402 is generated according the instruction or algorithm associated with the prediction mode in Step 504, whose process will be discussed below in detail. After Step 503 or 504, the flow advances to Step 505, where it is determined whether the prediction blocks are generated for all of the prediction modes. If intra prediction is run under all of the prediction modes, the flow advances to Step 506. Otherwise, the prediction mode is incremented in Step 507 and the flow returns to Step 502. In Step 506, each of the generated prediction blocks is compared to the target block 400 to determine the optimum prediction mode, which minimizes the prediction residual 404.

[0045] FIG. 6 shows the functional modules of the video decoder 301. These functional modules are realized by the processor 101 executing the software 103 in the memory 102. The encoded video signal from the encoder 201 is first received by an entropy decoder 600 and entropy-decoded back into quantized transform coefficients 601. The quantized transform coefficients 601 are inversely quantized and transformed by an inverse quantization/transformation module 602 to generate a prediction residual 603. An intra prediction module 604 is notified of the prediction mode selected by the encoder 201. According to the selected prediction mode, the intra prediction module 604 performs an intra prediction process similar

to that performed in Steps 502, 503 and 504 of FIG. 5 to generate a prediction block 605, using boundary pixels of neighboring blocks previously reconstructed and stored in a frame memory 606. The prediction block 605 is added to the prediction residual 603 to reconstruct a block 607 of decoded video signal. The reconstructed block 607 is stored in the frame memory 606 for use in prediction of a next block.

[0046] A detailed description will be given as follows on the process of Step 504 performed by the intra prediction modules 401 and 604 to generate a prediction block under one of the prediction modes, except the DC prediction mode. H.264/AVC supports Intra_4×4 prediction, Intra_8×8 prediction and Intra_16×16 prediction. Intra_4×4 prediction is commonly used when there is significant detail in the picture. Intra_4×4 prediction predicts the sixteen 4×4 luma blocks within one macroblock individually. Intra_4×4 prediction is performed under nine prediction modes, including one DC prediction mode. Spatial prediction directions along which Intra_4×4 prediction is performed are shown in FIG. 7. Intra_8×8 prediction is performed under nine prediction modes, including one DC prediction mode. Intra_16×16 prediction is performed under four prediction modes, including one DC prediction mode.

[0047] Recent studies show that an increase in the number of prediction directions, or an increase in the number of prediction modes, generally contributes to improving the compression efficiency in video coding. See, for example, Document Nos. JCT-VC A119 ("Angular Intra Prediction") and JCT-VC A124 ("Arbitrary Direction Intra" - Authors: Ken McCann, et al.; Title: Samsung's Response to the Call for Proposals on Video Compression Technology; JCT-VC of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 1st Meeting at Dresden, Germany, 15-23 April, 2010) submitted to Joint Collaborative Team on Video Coding (JCT-VC). An increase in the number of prediction directions leads to an increase in the number of angular intervals of available prediction directions and, thus, to an increase in the number of prediction block candidates. The increased number of prediction block candidates simply increase chances to have a prediction block which is nearly the same as a target block to be encoded. FIG. 8 is a diagram showing the prediction directions proposed in Document No. JCT-VC A119. In FIG. 8, the reference pixels consist of seventeen (17) horizontal pixels and seventeen (17) vertical pixels, where the upper left pixel is common to both horizontal and

vertical boundaries. Therefore, 33 different prediction directions are available to generate prediction pixels in an 8×8 block. JCT-VC A124 proposes arbitrary directional intra prediction in which the number of prediction directions is adjusted according the size of a block to be predicted.

[0048] FIG. 9 is a flowchart showing the process, proposed in JCT-VC A119, of generating a prediction block along one of the prediction directions shown in FIG. 8. In the following description of the process, some algorithms are simplified for ease of explanation. Also, the described process is limited to intra prediction along a prediction direction that is mainly vertical. Intra prediction along a prediction direction that is mainly horizontal can be implemented symmetrically to the process shown in FIG. 9, as demonstrated in the software provided by JCT-VC A119. Although FIG. 8 shows an 8×8 block to be predicted, the process shown in FIG. 9 is expandable to be applied to various numbers of pixels in different configurations. For example, a block to be predicted may comprises a 4×4 array of pixels. A prediction block may also comprise an 8×8 array of pixels, a 16×16 array of pixels, or larger arrays of pixels. Other pixel configurations, including both square and rectangular arrays, may also make up a prediction block.

[0049] In Step 900 in FIG. 9, reference pixels in horizontal and vertical boundaries, which lie immediately above and left of a target block, respectively, are read from neighboring blocks which have been previously encoded, reconstructed and stored in a frame memory, such as the memory 403 shown in FIG. 4. The pixels from the horizontal boundary are stored in a memory area called "*refH*". The pixels from the vertical boundary are stored in another memory area called "*refV*". Returning to FIG. 8, the reference pixels are identified by their coordinates in a coordinate system having the origin at the upper left pixel position in the 8×8 block. Thus, the horizontal boundary pixels have coordinates expressed by p[x, y] with x = 0, 1...16 and y = 0. The vertical boundary pixels have coordinates expressed by p[x, y] with x = 0, y = 0, -1, -2...-16.

[0050] It is assumed that the horizontal boundary pixels stored in the memory area *refH* are identified by a logical address (x) with x = 0, 1...16 and that the vertical boundary pixels stored in the memory area *refV* are likewise identified by

a logical address (y) with y = 0, -1, -2...-16, where each pixel is stored in the address having the number in the coordinate from which it is read. Therefore, as the horizontal and vertical pixels are graphically represented in FIG. 8, the memory areas *refH* and *refV* may be considered extending linearly and orthogonal to each other and each having a length of 2 × *size* + 1, where "*size*" is a parameter representing the size of the target block. It is assumed that *size* has a value equal to an integer power of 2, such as 4, 8, 16... A low-pass filter as described in Section 8.3.2.2.1 in H.264/AVC may optionally be applied to the pixels in *refH* and *refV*.

[0051] In Step 901, a counter called "*row*" is set to zero ("0"). The counter *row* takes a value from 0 to *size* and indicates a row position of a prediction pixel in the prediction block. In Step 902, a parameter called "*pos*" is calculated by *angle* × (*row* + 1). *angle* is a parameter having a fractional number in a fixed-point representation. As such, *angle* is formed with an integer part and a fraction part, and the fraction part consists of a fixed number of binary digits. *angle* represents one of the prediction directions shown in FIG. 8. For instance, "*angle* = -*size*" identifies the prediction direction which goes through the coordinates [x = 0, y = 0] in FIG. 8. *angle* having a positive value identifies a prediction direction which intersects only the horizontal boundary, whereas *angle* having a negative value identifies a prediction direction which intersects both the horizontal and vertical boundaries. *angle* varies within a range determined by the number of prediction directions desired to be used. As proposed in JCT-VC A124, the number of prediction directions to be used may be determined according the size of a block to be predicted. In the following description, it is assumed that *angle* takes a fractional number which varies within a range from "-*size*" to "*size*". Please note that the range limits of *angle* may be defined with other values.

[0052] Like *angle*, the parameter *pos* consists of an integer part and a fraction part, and the fraction part thereof consists of a fixed number of binary digits, which is equal to the logarithm in base 2 of the range limit of *angle*, which may be expressed by *log2_size* according to the above assumption that the range limit of *angle* is set to *size*. *pos* identifies the position of an intersection between the horizontal boundary and the prediction direction represented by *angle*. Returning

to Step 902, the operation "*pos* >> *log2_size*" identifies an integer number in *pos*, which is stored in a parameter "*int*", and the operation "*pos* & (*size* - 1)" identifies a fraction number in *pos*, which is stored in a parameter "*frac*". The operator ">>" calls for an arithmetic right shift of binary digits. The operator "&" calls for bit-wise "and" operation.

[0053] In Step 903, it is determined whether *angle* has a value equal to or larger than zero ("0"). If *angle* has a value equal to or larger than zero, the flow proceeds to Step 904. The flow otherwise proceeds to Step 913. *angle* equal to or larger than zero suggests that only the reference pixels located in the horizontal boundary, or stored in *refH*, can be relied upon to derive prediction pixels in a prediction block. On the other hand, *angle* smaller than zero suggests that reference pixels located in the vertical boundary, or stored in *refV*, are needed to derive prediction pixels in the prediction block.

[0054] In Step 904, it is determined whether *frac* is not zero. If *frac* is not zero, the flow proceeds to Step 905. If *frac* is zero, the flow proceeds to Step 906. *frac* equal to zero suggests that a prediction pixel in the prediction block can be copied directly from a reference pixel in the horizontal boundary. Non-zero *frac* suggests that the prediction direction intersects the horizontal boundary at at non-integer position, and an interpolation of more than one reference pixel is needed to derive a prediction pixel in the prediction block.

[0055] In Step 905, a counter called "*col*" is set to zero ("0"). The counter *col* is used to address a reference pixel in *refH*. In Step 907, two reference pixels indentified by "*int* + *col* + 1" and "*int* + *col* + 2" are retrieved from *refH*. These two reference pixels are weight-averaged or interpolated with *frac* to derive a prediction pixel *v*. Specifically, a reference pixel in *refH* identified by "*int* + *col* + 1" is multiplied by "*size* - *frac*" and stored in a parameter *a*. A reference pixel in *refH* identified by "*int* + *col* + 2" is multiplied by "*frac*" and stored in a parameter *b*. The parameters *a* and *b* are then added and divided by *size*, i.e., (*size* - *frac*) + *frac*. The division by *size* can be replaced with right shift by *log2_size*. The derived prediction pixel *v* is stored in an array of memory areas called "*pred,*" which represents a prediction block for the target block under a particular prediction direction. Each memory area in *pred* is identified by the parameters

row and *col*. Then, *col* is incremented by 1 in Step 908 and compared to *size* in Step 909. As long as *col* is smaller than *size*, Steps 907 and 908 are repeated. When *col* becomes equal to *size*, the flow proceeds to Step 920.

[0056] If *frac* is determined zero in Step 904, the counter *col* is set to zero in Step 906. In Step 910, the prediction pixel *v* is copied directly from *refH* (*int* + *col* + 1) and then stored in the corresponding memory area in *pred*. *col* is then incremented by 1 in Step 911 and compared to *size* in Step 912. As long as *col* is smaller than *size*, Steps 910 and 911 are repeated. When *col* becomes equal to *size*, the flow proceeds to Step 920.

[0057] Returning to Step 903, *angle* smaller than zero requires reference pixels from *refV* to derive prediction pixels in the prediction block. The counter *col* is set to zero in Step 913. It is then determined in Step 914 whether "*int* + *col* + 1" is lower than zero. "*int* + *col* + 1" equal to or larger than zero suggests that only the reference pixels stored in *refH* can still be relied upon to derive prediction pixels in the prediction block, and the flow proceeds to Step 915. The process performed in Step 915 is similar to that of Step 907, and description thereof will not be repeated here. *col* is then incremented by 1 in Step 916 and compared to *size* in Step 917. As long as *col* is smaller than *size*, Steps 914, 915 and 916 are repeated. When *col* becomes equal to *size*, the flow proceeds to Step 920.

[0058] If "*int* + *col* + 1" is determined smaller than zero in Step 914, reference pixels stored in *refV* are needed to derive prediction pixels in the prediction block. In Step 918, the position of an intersection between the vertical boundary and a prediction direction is first determined. In Step 918, the position is represented by *pos2*. Please note that in Step 902, *pos*, i.e., the position of an intersection between the horizontal boundary and a prediction direction, is determined by "*angle* × (*row* + 1)". Given that *angle* represents a ratio of horizontal and vertical differences, "$angle^{-1}$ × (*col* + 1)", instead of "*angle* × (*row* + 1)", is calculated to determine the position of an intersection between the vertical boundary and a prediction direction. As assumed above, *angle* is within the range of -*size* to *size* (-*size* ≤ *angle* ≤ *size*). Therefore, a ratio α between *angle* and *size* is defined by:

$$\alpha = \frac{angle}{size} \ (-1 \leq \alpha \leq 1).$$

Then, $angle^{-1}$ is defined by:

$$angle^{-1} = \frac{size}{\alpha} \text{ or } \frac{size^2}{angle}.$$

As such, *pos2* is determined in Step 918 with the square of *size* multiplied by *col* + 1 and then divided by the absolute value of *angle* as follows:

$$pos2 = \frac{size^2 \times (col+1)}{|angle|}.$$

[0059] Like *pos*, *pos2* has a fractional number in a fixed-point representation being formed of an integer part and a fraction part. The fraction part consists of the number of binary digits determined by *log2_size*. The integer part of *pos2* is stored in a parameter *int2*, and the fraction part of *pos 2* is stored in a parameter *frac2*. In Step 919, two reference pixels identified by "*int2* + *row* + 1" and "*int2* + *row* + 2" are retrieved from *refV*. These two reference pixels are weight-averaged or interpolated with *frac2* to derive a prediction pixel *v*. Specifically, a reference pixel from *refV* (*int2* + *row* + 1) is multiplied by "*size* - *frac2*" and stored in a parameter *a*. A reference pixel from *refV* (*int2* + *row* + 2) is multiplied by "*frac2*" and stored in a parameter *b*. The parameters *a* and *b* are then added and divided by *size* or right shifted by *log2_size*. The derived prediction pixel *v* is stored in the corresponding memory area of *pred*. Steps 914, 918, 919 and 916 are repeated until *col* becomes equal to *size* in Step 917.

[0060] In Step 920, *row* is incremented by 1. It is then determined in Step 921 whether *row* is smaller than *size*. As long as *row* is smaller than *size*, the Steps from Step 902 are repeated to derive a prediction pixel in the prediction block. The flow ends when *row* becomes equal to *size* in Step 921.

[0061] As mentioned above, an increase in the number of prediction block candidates contributes to improving the coding efficiency, whereas an increase in the number of prediction block candidates leads to an increase in the computational workload. Therefore, in order to increase the number of prediction block candidates to thereby improve the coding efficiency, the process of generating a prediction block candidate needs to be reviewed to further achieve the efficiency of the process. In reviewing the process shown in FIG. 9, two computational bottlenecks may be identified. The first computational bottleneck

is the comparison and branching operation of Step 914, which is repeated within the loop. The second computational bottleneck is the divisional operation of Step 918, which is also repeated within the loop.

[0062] In these days, Single-Instruction Multiple Data (SIMD) is available for efficient computing. SIMD enables computers with multiple processing elements to perform the same operation on multiple data simultaneously. However, typical SIMD architectures do not support implementation of division and computation/branching in a loop and, thus, cannot be used to implement the process shown in FIG. 9 because of inclusion of Steps 914 and 918 in the loop, although the loops starting from Steps 907 and 910 are robust enough to be implemented with SIMD. It is therefore an object of the present invention to remove the computational bottlenecks from the process shown in FIG. 9 and provide low complexity intra prediction, which enables typical SIMD architectures to implement parallel processing along all of the prediction directions shown in FIG. 8.

[0063] FIG. 10 is a flowchart showing the process of low complexity intra prediction according to an embodiment of the present invention, which is designed to replace the process of FIG. 9 in implementation of the process in Step 504 of FIG. 5. In FIG. 10, the same process steps as performed in FIG. 9 are identified by the same step numbers as used in FIG. 9, such as Steps 900, 901, 902, 904, 905, 906, 907, 908, 909, 910, 911, 912, 920 and 921. Description of these common steps is not repeated here. Steps 1000 and 1001 are steps peculiar to the process of FIG. 10. As is apparent from comparison to the process shown in FIG. 9, the process of FIG. 10 eliminates the comparison step of Step 903 and all of the steps branched to the left from Step 903, which are performed when *angle* is smaller than zero, thereby eliminating the computational bottlenecks of Steps 914 and 918.

[0064] In added Steps 1000 and 1001, it is determined whether *angle* is equal to or larger than -1. When *angle* is equal to or larger than -1, reference pixels located in the horizontal boundary are sufficient to generate a prediction pixel in the prediction block, and reference pixels in the vertical boundary are not needed. On the other hand, *angle* is smaller than -1, reference pixels in the vertical boundary are needed to generate a prediction pixel in the prediction block. In Step

1001, reference pixels stored in *refH* are extended in the negative direction, using at least some of the pixels stored in *refV*. FIGS. 11A and 11B are schematic representations showing extension of *refH* performed in Step 1001. In FIG. 11A, reference pixels 1102 stored in *refH* are from the horizontal boundary located above the target block 1101. Reference pixels 1103 stored in *refV* are from the vertical boundary located left of the target block 1101. As shown in FIG. 11B, after Step 1001 of FIG. 10, some of the reference pixels in *refV* are copied into *refH*, and *refH* has an extended part 1104 extending in the negative direction.

[0065] FIG. 12 is a flowchart showing details of the process performed in Step 1001. In Step 1201, a counter *col* is set to -1. *col* is used to identify an address of the extended part of *refH*. In Step 1202, a reference pixel in *refV* to be copied into the extended part of *refH* is identified by:

$$\frac{size \times col}{angle}.$$

The division in the above equation is an integer division, and the equation yields an integer number. The equation functions similarly to the process of Step 918 shown in FIG. 9. In Step 918, an integer value of *pos2* is calculated by:

$$\frac{(size^2 \times (col+1))}{angle} >> \log 2\_size.$$

Please note that right shift by *log2_size* is equivalent to division by *size*.

[0066] In Step 1203, *col* is decremented by 1. It is then determined in Step 1204 whether *col* is equal to *angle*. If *col* is not equal to *angle*, the flow returns to Step 1202. Steps 1202 and 1203 are repeated until *col* becomes equal to *angle*. Thus, reference pixels are read from *refV* in the ascending order, or from the top to the bottom of the vertical boundary, and copied into the *refH* also in the descending order, or from the right to the left of the horizontal boundary. Also, not all of the reference pixels in *refV* are copied into *refH*. Only the reference pixels located within the range from the top to the intersection of a prediction direction are copied from *refV* into *refH*.

[0067] Retuning to FIG. 10, the process steps starting from Step 902 are copied from FIG. 9, and includes the steps for generating prediction pixels branched to the right from the comparison step of Step 903 in FIG. 9. Please note, however,

that the steps in FIG. 10 for generating prediction pixels use extended *refH* (a sum of parts 1102 + 1104 in FIG. 11B), whereas the corresponding steps in FIG. 9 use original *refH* (part 1102 in FIG. 10A). Since *refH* is extended in the negative direction, a separate intra prediction operation designed specifically to use reference pixels stored in *refV*, such as branched to the left from Step 903 in FIG. 9, is not needed regardless of the sign of *angle*.

[0068] FIG. 13 is a flowchart showing another embodiment of the process for extending *refH*, using reference pixels in *refV*. The process shown in FIGS. 11 and 12 eliminates the bottleneck steps of Steps 914 and 918 shown in FIG. 9 and, thus, is expected to improve the efficiency of the intra prediction process. The process shown in FIG. 13 eliminates the divisional operation performed in Step 1202 of FIG. 12 from the loop for copying reference pixels from *refV* into *refH*. By eliminating the divisional operation from the loop, the process shown in FIG. 13 is expected to further improve the efficiency of the intra prediction process.

[0069] The process shown in FIG. 13 replaces Step 1202 of FIG. 12 with Steps 1301 and 1302. Step 1302 is within the loop for copying reference pixels from *refV* into *refH*, whereas Step 1301 is outside the loop. Step 1301 introduces a new parameter called "*InvAngle*". *InvAngle* is defined by:

$$256 \times \frac{size}{angle}.$$

Multiplication by 256 is equivalent to left shift by 8 and makes sure that every bit resulting from the operation of "*size/angle*" accounts for the calculation of identifying a reference pixel in *refV*. In Step 1302, the address of a reference pixel in *refV* to be copied into the extended part of *refH* is identified by:

$$col \times InvAngle >> 8 \ .$$

The result of "*col* × *InvAngle*" is right-shifted by 8 to undo the left shift operation performed in Step 1301. Please note that the right shift operation in Step 1302 functions to round down the result of "*col* × *InvAngle*". To round towards a nearest integer, a rounding offset of 128 may be added to the result of "*col* × *InvAngle*" before the right shift operation is performed. It should be noted that the number "256" is just an example, and Step 1301 can adopt another offset number,

preferably an integer power of 2, as long as the number is large enough to preserve all the bits resulting from the operation of "*size/angle*". For instance, the number may be 64 in Step 1301, instead of 256, and the number of right shifts is 6 in Step 1302, instead of 8. If 64 is adopted, the rounding offset should be 32.

[0070] The calculation performed in Step 1301 may be replaced with a look-up operation to further reduce the computational workload. In other words, a look-up table is prepared which stores values of *InvAngle* in relation to the values of *angle*. Table 1 provided below is an exemplary table for look up in Step 1301:

Table 1

angle	4	5	6	7	8
InvAngle	512	410	341	293	256

It is assumed that in the above table, *size* is 8, and *angle* takes integer values from 4 through 8. It should however be noted that *size* is not limited to 8 and may take another value, such as 4 and 16. Also, *angle* may be a fractional number in a fixed-point representation as defined above.

[0071] When a reference pixel is copied from *refV* to *refH* in Step 1202 of FIG. 12 or Step 1302 of FIG. 13, the reference pixel may go through a low-pass filter to reduce possible aliasing in the prediction block. The strength of the low-pass filter may vary according to the value of *angle*. For example, when *angle* is equal to *-size*, weak low-pass filtering may be applied, and when *angle* is equal to -2, strong low-pass filtering may be applied.

[0072] As explained above, not all of the reference pixels in *refV* are copied into *refH*. Since not all of the reference pixels in *refV* are copied, some information is lost when pixels are copied. To mitigate the loss of information, the resolution of reference pixels in *refH* and *refV* may be doubled so that *refH* and *refV* contain not only pixels from previously encoded and reconstructed blocks but also one pixel between two adjacent reconstructed pixels, which is generated by interpolating two adjacent pixels. Two adjacent pixels may simply be averaged to generate an interpolation pixel. The interpolation process may be performed when reference pixels are read in Step 900 of FIG. 9. When the resolution of pixels is doubled in *refH* and *refV*, identifications of the addresses of reference pixels

stored in *refH* and *refV*, such as performed in Steps 907, 910, 915 and 919 in FIG. 9, and Step 1001 in FIG. 10, need to be scaled. For instance, "*int* + *col* + 1" performed in Steps 907, 910 and 915 needs to be changed to "*int* + 2×*col* + 2". "*int* + *col* + 2" performed in Steps 907, 910, 915 needs to be changed to "*int* + 2×*col* + 3". "*int2* + row + 1" and "*int2* + row + 2" performed in Step 919 need to be changed to "*int2* + 2×row + 2" and "*int2* + 2×row + 3", respectively.

[0073] In another embodiment, the process of Step 1202 in FIG. 12 may be changed simply to "*refH* [*col*]←*refV* [-*col*]" to further simply the copying process. Although degrading the accuracy of prediction, this embodiment provides the lowest complexity to the intra prediction operation.

[0074] FIG. 11B shows the extended part 1104 added to *refH*. The extended part 1104 does not need to be formed with reference pixels from *refV*. The extended part 1104 may be formed with pixels from an area of previously reconstructed block, which spatially corresponds to the location of the extended part 1104. In FIG. 11B, since extended in the negative direction, extended *refH* (parts 1102 and 1104) ranges from -*size* + 1 to 2×*size*. The range of extended *refH* may be rescaled to range from 0 to 3×*size* - 1 by adding an appropriate offset when addressing reference pixels in extended *refH*. The same holds true for rescaling the range of *refV*.

[0075] In another embodiment, the range limit of *angle* may be freely chosen. In the above embodiments, it is assumed that *angle* takes a value within a range from -*size* to *size* (-*size* ≤ *angle* ≤ *size*). In other words, in the above embodiments, the range limits of *angle* are defined with the size of the target block. Please note that the range limits of *angle* may be defined independently from the size of the target block, although it is still preferable that the range limit be defined with an integer power of 2, so that *log2_rangelimit* is a positive integer, and the equation "*rangelimit* = 1 << *log2_rangelimit*" holds true. By choosing a suitable large number for *rangelimit*, a large number of prediction directions can be established and represented by values of *angle* at sufficiently wide angular intervals.

[0076] If the range limit of *angle* is defined independently from the size of the target block, *size* appearing in FIGS. 9 and 10 needs to be replaced with *rangelimit*, and *log2_size* needs to be replaced with *log2_rangelimit*, except for Steps 909,

912, 917 and 921. The comparison of "*angle* ≥ -1" performed in Step 1000 of FIG. 10 also needs to be replaced with "*angle*×*size/rangelimit* ≥ -1" or "*angle*×*size* ≥ - *rangelimit*". Further, *size* appearing in Steps 1202 and 1301 in FIGS. 12 and 13 needs to be replaced with *rangelimit*, and the comparison of "*col* = *angle*?" performed in Step 1204 needs to be replaced with "*col* = *angle* × *size/rangelimit*?".

[0077] If *rangelimit* is introduced as a range limit of *angle*, Table 1 (provided above) may be changed as follows:

Table 2

*angle**	13	17	21	26	32
InvAngle	630	482	390	315	256

In Table 2, *rangelimit* is set to 32. *angle** is equal to an integer approximation of "*rangelimit* × tan (π/4 × *angle*/8)", where *angle* = 4, 5, 6, 7 and 8. *InvAngle* is equal to 256 × *rangelimit*/*angle**. The values in Table 2 are all integers which are derived by rounding up. Instead of being rounded up, the numbers may be rounded down. In Table 3 provided below, *InvAngle* is equal to 32 × *rangelimit*/*angle**. Since "32" is used, instead of "256", the accuracy of prediction is necessarily lower than that of Table 2.

Table 3

*angle**	13	17	21	26	32
InvAngle	78	60	48	39	32

[0078] FIG. 14 is a flowchart showing another embodiment which further simplifies the process shown in FIG. 10. The process shown in FIG. 10 of copying reference pixels from *refV* into *refH* is performed before the flow enters the main prediction loop, whereas the copying process shown in FIG. 14 is performed within the main prediction loop. Also, the process shown in FIG. 14 eliminates the variable *InvAngle*. Steps 900, 902 and 921 shown in FIG. 14 are from the corresponding steps in FIG. 10.

[0079] In Step 1401, a counter *lastInt* is initialized to -1. *lastInt* represents the index of the last pixel which was added to *refH*. In Step 902, *pos* is calculated by *angle* × (*row* + 1). As explained above, *pos* identifies the position of an

intersection between the boundaries and the prediction direction represented by *angle*. In the context of FIG. 9, Step 902 yields *pos*, which identifies the position of an intersection between the horizontal boundary and the prediction direction represented by *angle*. Further in Step 902, an integer part in *pos* is stored in *int*, and a fraction part in *pos* is stored in a parameter "*frac*". In Step 1402, it is determined whether *int* is smaller than *lastInt*. If *int* is smaller than *lastInt*, a reference pixel in *refV* identified by *row* is copied into *refH* at an address identified by "*int* + 1". Step 1404 consists of Steps 904, 905, 906, 907, 908, 909, 910, 911 and 912 shown in FIGS. 9 and 10, whose description is not repeated here. In Step 1405, *int* is copied to *lastInt*. The operation of copying *int* to *lastInt* may be performed in Step 1403, instead of Step 1405.

[0080] The copying operation in Step 1403 results in copying the same pixel as copied in Steps 1202 and 1302, where rounding down is used in these steps. Step 1403 can be modified to round to a nearest integer by conditionally using "*row* + 1", instead of "row", in Step 1403 when the fractional position *frac* computed in Step 902 is larger than *offset*, which is defined by *rangelimit* + (*angle* >> 1). Please note that *angle* is -ve, and *frac* is +ve. The use of "*row* + 1" results in rounding up. To effect the conditional increment of *row* by 1, the process performed in Step 1403 is changed to *refH*[*int* + 1] ← *refV*[*row* - ((*offset* - *frac*) >> 31)], assuming that in 32 bit arithmetic, right shift of "*offset* - *frac*" results in -1 when *frac* is larger than *offset* and results in 0 otherwise. Thus, the address identifier "*row* - ((*offset* - *frac*) >> 31)" becomes "*row* + 1" when *frac* is larger than *offset* and becomes "*row*" otherwise. If *offset* is set to *rangelimit*, "*offset* - *frac*" will always be positive and thus no rounding will occur.

[0081] The source code developed in the C++ programming language, which implements the process shown in FIG. 14, is listed below. The source code is modified from the TComPrediction::xPredIntraAng function found in the TComPrediction.cpp file which is part of the TMuC 0.7 software developed by JCT-VC.

```
// Function for deriving the simplified angular intra predictions
```

```
Void TComPrediction: :xPredIntraAng( Int* pSrc, Int iSrcStride, Pel*& rpDst,
Int iDstStride, UInt iWidth, UInt iHeight, UInt uiDirMode, Bool bAbove, Bool
bLeft ){
  Int k,l;
  Int deltaInt, deltaFract, refMainIndex;
  Int intraPredAngle       = 0;
  Int absAng        = 0;
  Int signAng              = 0;
  Int blkSize              = iWidth;
  Bool modeDC              = false;
  Bool modeVer             = false;
  Bool modeHor             = false;
  Pel* pDst         = rpDst;

// Map the mode index to main prediction direction and angle
  if (uiDirMode == 0)
    modeDC = true;
  else if (uiDirMode < 18)
    modeVer = true;
  else
    modeHor = true;

  intraPredAngle = modeVer ? uiDirMode - 9 : modeHor ? uiDirMode - 25 : 0;
  absAng              = abs(intrapredAngle);
  signAng      = intraPredAngle < 0 ? -1 : 1;

  // Set bitshifts and scale the angle parameter to size2
  Int iAngTable[9]  = {  0,  2,  5,  9,  13,  17,  21,  26,  32};
  absAng = iAngTable[absAng];
  intraPredAngle = signAng * absAng;

  // Do the DC prediction
  if (modeDC){
    Pel dcval = predIntraGetPredValDC(pSrc, iSrcStride, iWidth, iHeight,
bAbove, bLeft);
```

```
        for (k=0;k<blkSize;k++){
          for (l=0;l<blkSize;l++){
            pDst(k*iDstStride+1] = dcval;
          }
        }
      }

      // Do angular predictions
      else {
        Pel tmp;
        Int *pSrcTL = pSrc - iSrcStride - 1;
        Int iStepMain = (modeVer) ? 1 : iSrcStride;
        if (intraPredAngle == 0){
          for (k=0;k<blkSize;k++){
            for (l=0;l<blkSize;l++){
              pDst [k*iDstStride+1] = pSrcTL[(l+1) * iStepMain];
            }
          }
        }
        else {
          Int iStepSide = (modeVer) ? iSrcStride 1;
          int lastDeltaInt = -1;
          Int iOffset = 32 + (intraPredAngle >> 1);       // enables rounding to
nearest side reference
        // Int iOffset = 32;                          // no rounding.
          Pel ref [2*MAX_CU_SIZE];
          Pel* refMain =  ref + ((intraPredAngle < 0) ? blkSize : 0);
          if (intraPredAngle > 0){
            for (k = 0; k  <  2*blkSize; k++)
              refMain[k] = pSrcTL[(k+1)  *  iStepMain];
          }
          else {
            for (k = -1; k < blkSize; k++)    // the rest are copied later in step
1403, as and when required
```

```
            refMain[k] = pSrcTL[(k+1) * iStepMain];
        }

        for (k = 0; k < blkSize; k++){
        Int deltaPos = (k+1) * intraPredAngle;
        deltaInt       = deltaPos >> 5;
        deltaFract = deltaPos & (32 - 1);
        if (deltaInt < lastDeltaInt) { // step 1402
          lastDeltaInt = deltaInt;
          refMain[deltaInt] = pSrcTL[(k-((iOffset-deltaFract)>>31))*iStepSide];
// step 1403
        }

        // step 1404
        if (deltaFract){
          // Do linear filtering
          for (l=0;l<blkSize;l++){
            refMainIndex = l+deltaInt;
            pDst[k*iDstStride+l] = (Pel) ( ((32-deltaFract) *
refMain[refMainIndex] + deltaFract * refMain[refMainIndex+1] + 16) >> 5 );
          }
        }
        else {
          // Just copy the integer samples
          for (l=0;l<<blkSize;l++) {
            pDst[k*iDstStride+l] = refMain[l+deltaInt];
          }
        }
      }
    }
    // Flip the block if this is the horizontal mode
    if (modeHor){
      for (k=0;k<blkSize-1;k++){
        for (l=k+1;l<blkSize;l++){
```

```
            tmp = pDst[k*iDstStride+1];
              pDst(k*iDstStride+1] = pDst(1*iDstStride+k];
              pDst[1*iDstStride+k] = tmp;
          }
        }
      }
    }
```

[0082] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims, which in themselves recite only those features regarded as essential to the invention.

CLAIMS:

1. A video encoding method comprising computer executable steps executed by a processor of a video encoder to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation comprising:

obtaining a value of an inverse angle parameter, corresponding to the intra prediction angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles;

identifying at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array;

adding the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and

using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and

wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*,

where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

2. The method according to claim 1, wherein identifying the at least some of the vertical boundary pixels comprises an arithmetic right shift by 8 binary digits to undo multiplication by *N*, which takes a value of 256.

3. The method according to claim 1, wherein the look-up table comprises at least the following inverse angle parameters in relation to the following intra prediction angles

Intra prediction angle	13	17	21	26	32
Inverse angle parameters	630	482	390	315	256

.

4. The method according to claim 1, wherein the values of the inverse angle parameters listed in the look-up table are calculated from $(N \times rangelimit)/angle$,

where *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32, *angle* represents one of the plurality of different intra prediction angles and *N* takes a value of 256.

5. A video decoding method comprising computer executable steps executed by a processor of a video decoder to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation comprises:

obtaining a value of an inverse angle parameter, corresponding to the intra prediction angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles;

identifying at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the

inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array;

adding the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and

using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and

wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*,

where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

6. The method according to claim 5, wherein identifying the at least some of the vertical boundary pixels comprises an arithmetic right shift by 8 binary digits to undo multiplication by *N*, which takes a value of 256.

7. The method according to claim 5, wherein the look-up table comprises at least the following inverse angle parameters in relation to the following intra prediction angles

Intra prediction angle	13	17	21	26	32
Inverse angle parameters	630	482	390	315	256

.

8. The method according to claim 5, wherein the values of the inverse angle parameters listed in the look-up table are calculated from (*N* × *rangelimit*)/*angle*,

where *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32, *angle* represents one of the plurality of different intra prediction angles and *N* takes a value of 256.

9. A video encoder comprising a processor of a computer system and a memory that stores programs executable by the processor to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation implemented by the processor to:

obtain a value of an inverse angle parameter, corresponding to the intra predicition angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles;

identify at least some of one of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array;

add the identified at least some of the vertical boundary pixels as horizontal boundary pixels to the extension of the extended horizontal array; and

use only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and

wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*,

where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

10. The video encoder according to claim 9, wherein the processor is further programmed to perform an arithmetic right shift by 8 binary digits to undo multiplication by *N*, which takes a value of 256.

11. The video encoder according to claim 9, wherein the look-up table comprises at least the following inverse angle parameters in relation to the following intra prediction angles

Intra prediction angle	13	17	21	26	32
Inverse angle parameters	630	482	390	315	256

12. The video encoder according to claim 9, wherein the values of the inverse angle parameters listed in the look-up table are calculated from $(N \times rangelimit)/angle$,

where *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32, *angle* represents one of the plurality of different intra prediction angles and *N* takes a value of 256.

13. A video decoder comprising a processor of a computer system and a memory that stores programs executable by the processor to implement an intra-prediction operation that derives a prediction block of a target block with boundary pixels of the target block interpolated along an intra prediction angle, wherein the boundary pixels comprise a horizontal array of horizontal boundary pixels and a vertical array of vertical boundary pixels, the intra-prediction operation implemented by the processor to:

obtain a value of an inverse angle parameter, corresponding to the intra predicition angle, from a look-up table which lists values of inverse angle parameters in relation, respectively, to a plurality of different intra prediction angles;

identify at least some of the vertical boundary pixels located in the vertical array at positions which are a function of multiplication between the obtained value of the inverse angle parameter and a value of a horizontal location identifier which is a variable representing positions in an extension of an extended horizontal array;

add the identified at least some of the vertical pixels as horizontal boundary pixels to the extension of the extended horizontal array; and

using only the horizontal boundary pixels in the extended horizontal array, without using the vertical boundary pixels, to derive the prediction block of the target block, and

wherein the horizontal location identifier takes values of -1 ... (*size* × the intra prediction angle)/*rangelimit*,

where *size* represents a size of a target block to be predicted and *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32.

14. The video decoder according to claim 13, wherein the processor is further programmed to perform an arithmetic right shift by 8 binary digits to undo multiplication by *N*, which takes a value of 256.

15. The video decoder according to claim 13, wherein the look-up table comprises at least the following inverse angle parameters in relation to the following intra prediction angles

Intra prediction angle	13	17	21	26	32
Inverse angle parameters	630	482	390	315	256

.

16. The video decoder according to claim 13, wherein the values of the inverse angle parameters listed in the look-up table are calculated from (*N* × *rangelimit*)/*angle*,

where *rangelimit* represents a range limit of the plurality of intra prediction angles, which is fixed to a constant of 32, *angle* represents one of the plurality of different intra prediction angles and *N* takes a value of 256.

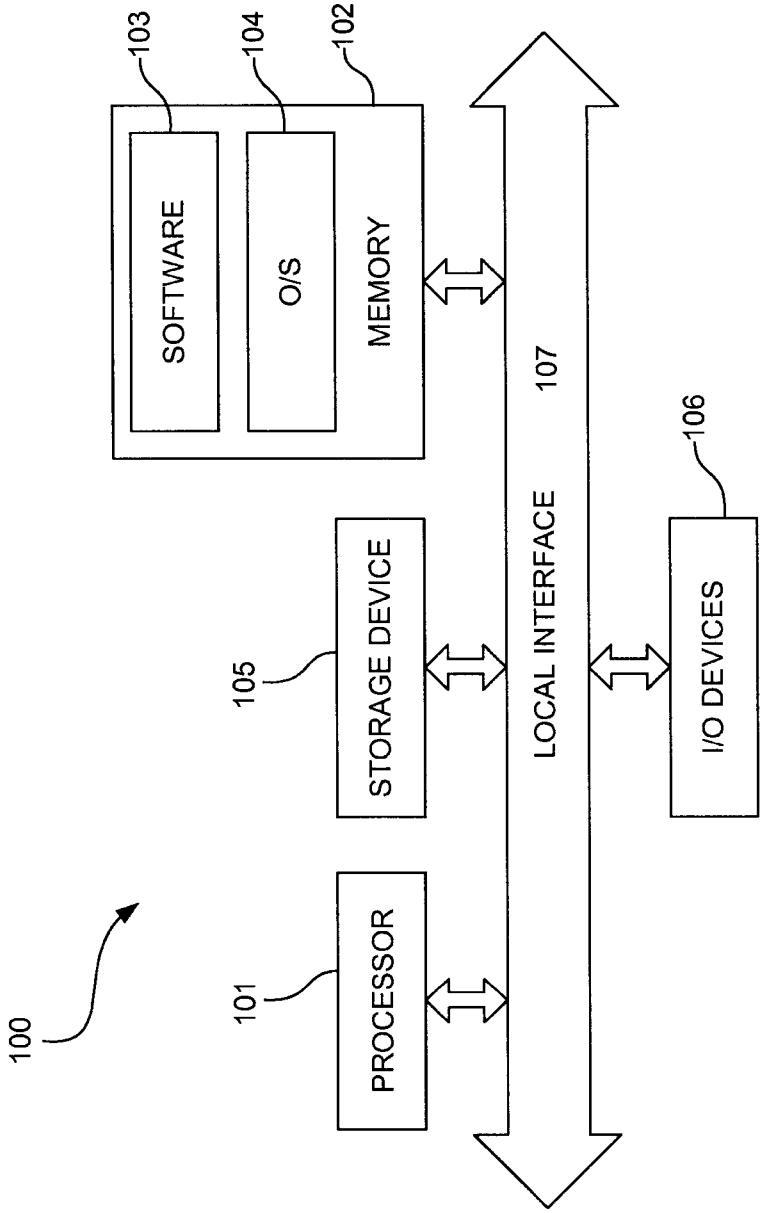

100
101
PROCESSOR
105
STORAGE DEVICE
LOCAL INTERFACE
107
106
I/O DEVICES
102
MEMORY
103
SOFTWARE
104
O/S

Fig. 1

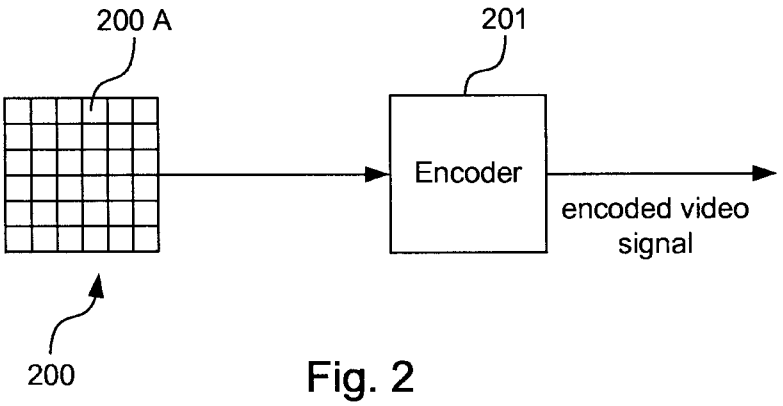

200 A
201
Encoder
encoded video signal
200

Fig. 2

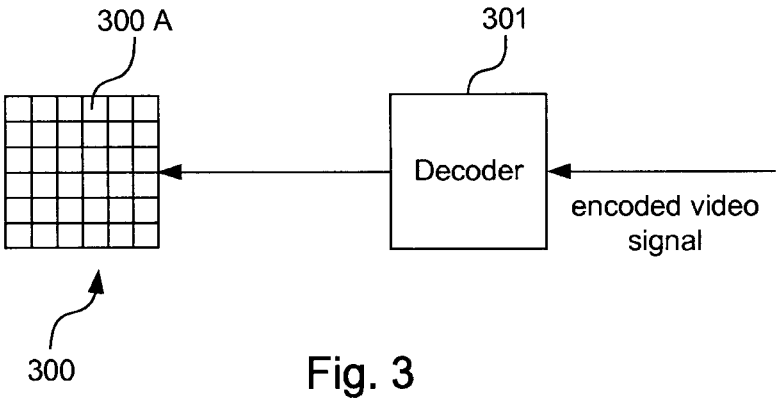

300 A
301
Decoder
encoded video signal
300

Fig. 3

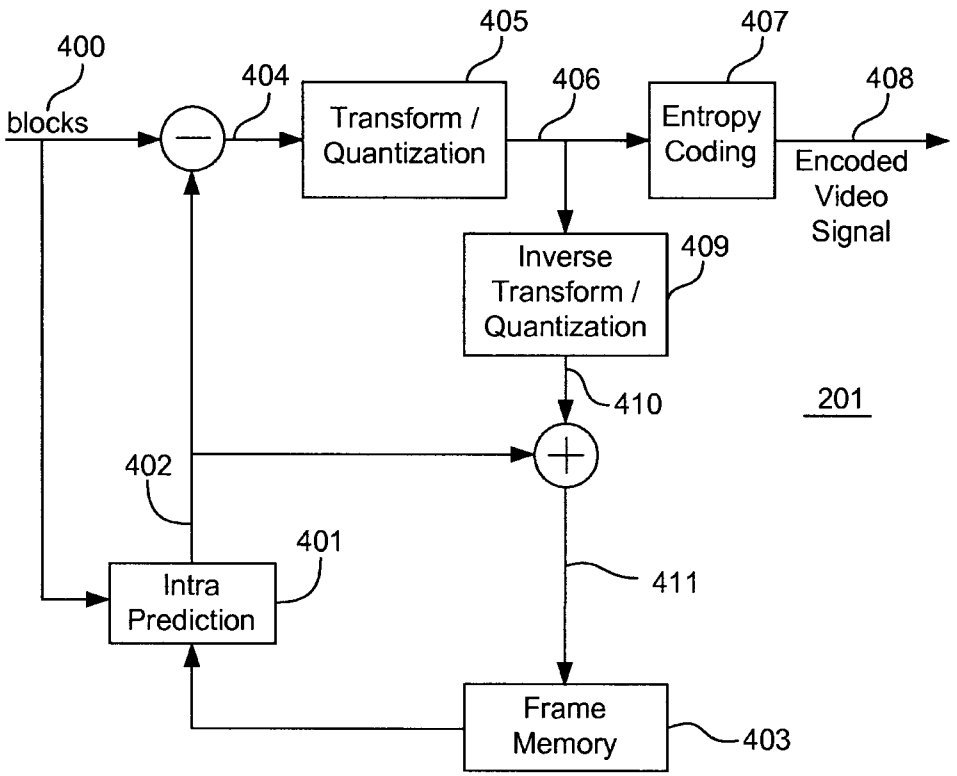

400
blocks
404
405
Transform / Quantization
406
407
Entropy Coding
408
Encoded Video Signal
Inverse Transform / Quantization
409
410
201
402
401
Intra Prediction
411
Frame Memory
403

Fig. 4

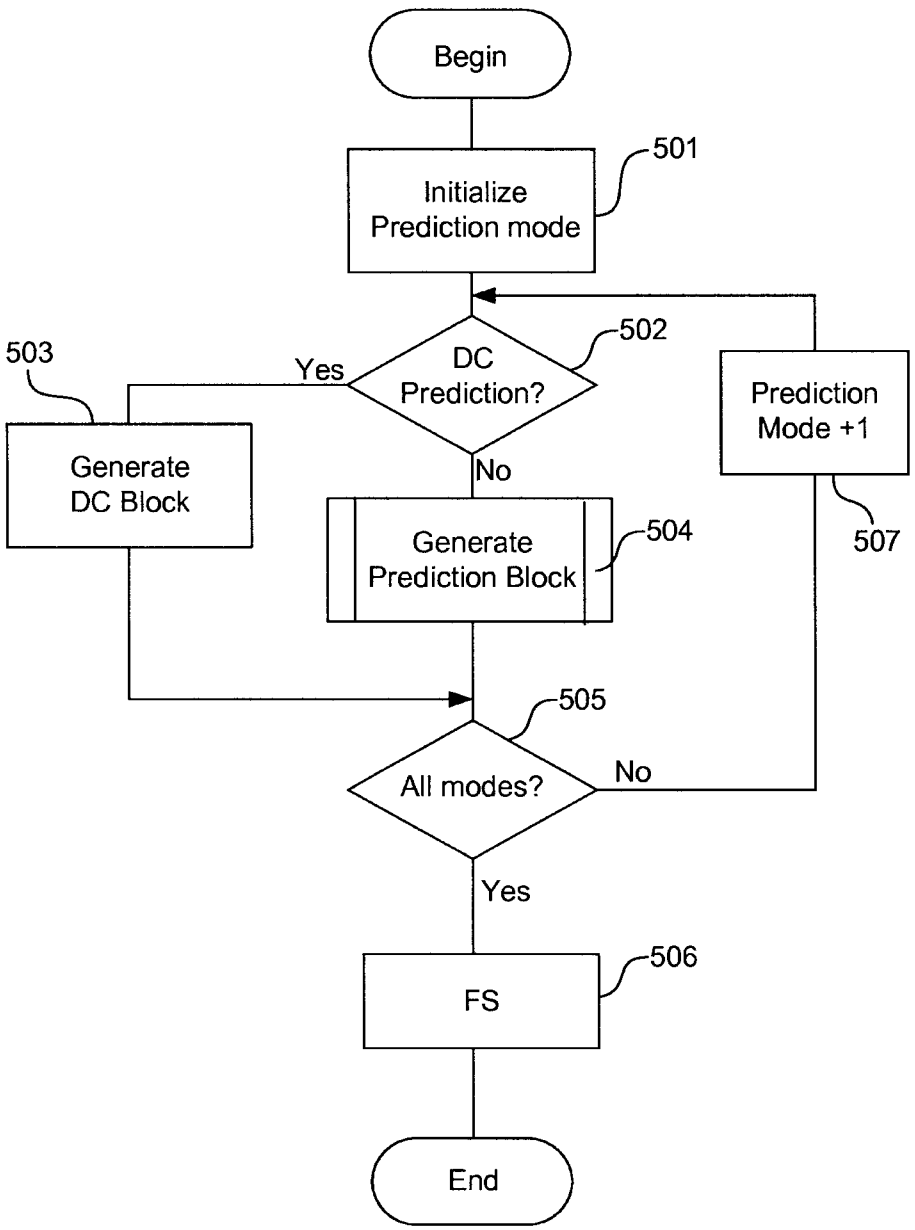

Begin
Initialize Prediction mode
501
DC Prediction?
502
Yes
503
Generate DC Block
No
Generate Prediction Block
504
Prediction Mode +1
507
All modes?
505
No
Yes
FS
506
End

Fig. 5

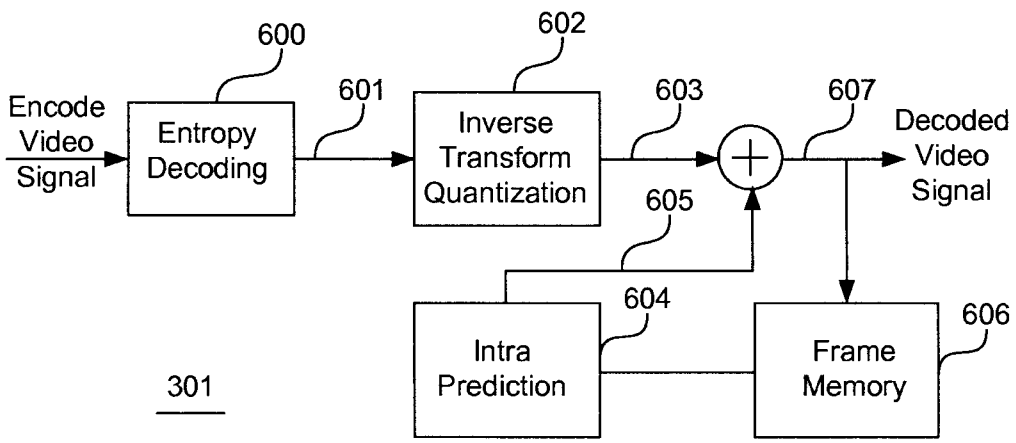

600
602
Encode
Video
Signal
Entropy
Decoding
601
Inverse
Transform
Quantization
603
607
Decoded
Video
Signal
605
604
606
Intra
Prediction
Frame
Memory
301

Fig. 6

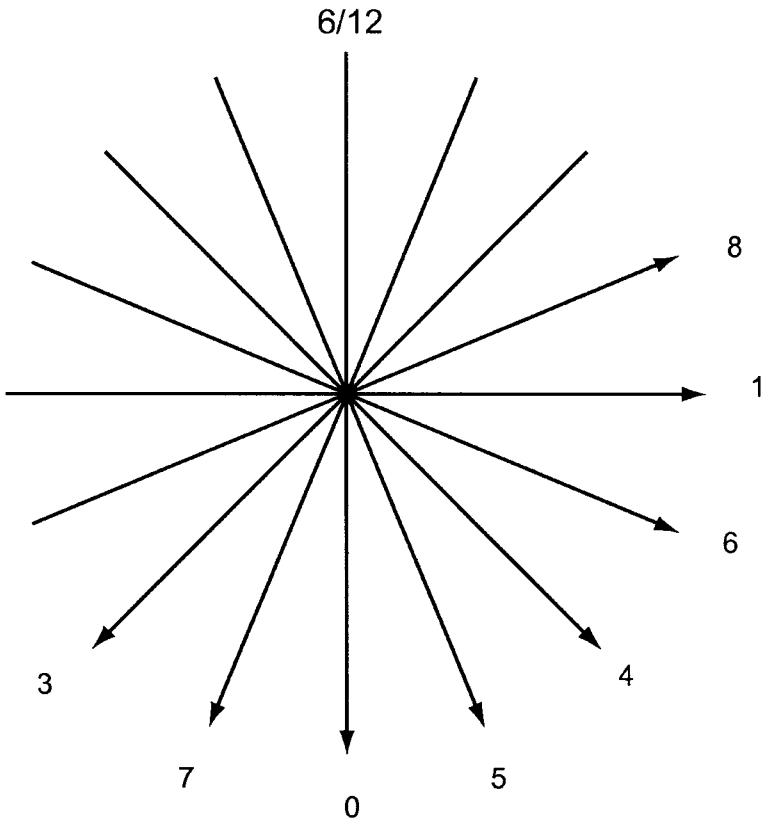


Fig. 7

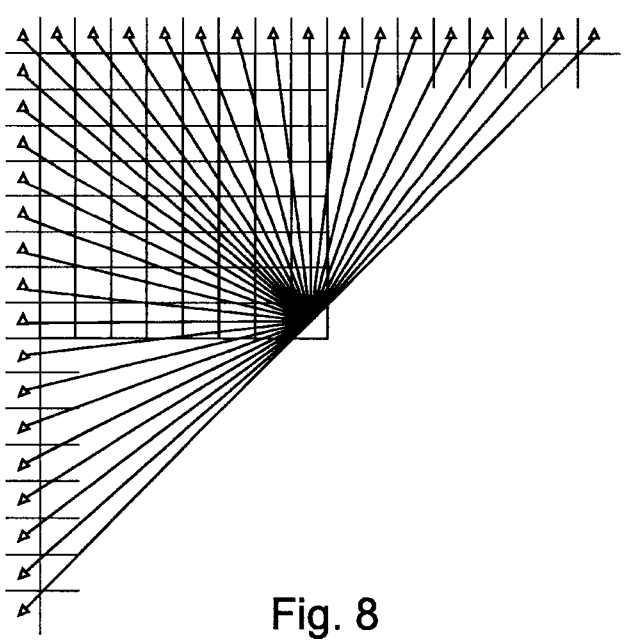


Fig. 8

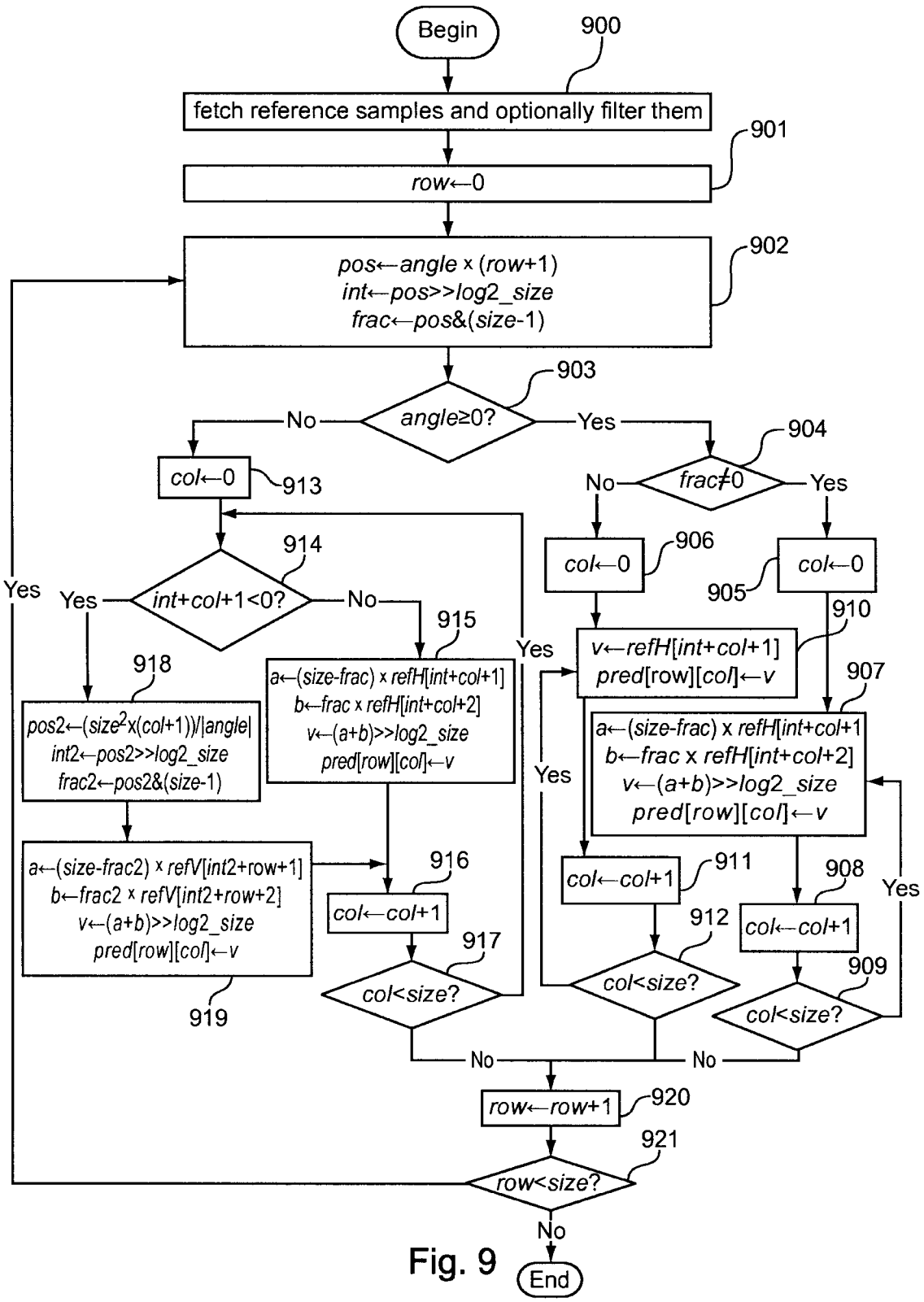

Begin
900
fetch reference samples and optionally filter them
901
row←0
902
pos←angle x (row+1)
int←pos>>log2_size
frac←pos&(size-1)
903
angle≥0?
No
Yes
904
frac≠0
No
Yes
col←0
913
914
int+col+1<0?
Yes
No
915
a←(size-frac) x refH[int+col+1]
b←frac x refH[int+col+2]
v←(a+b)>>log2_size
pred[row][col]←v
916
col←col+1
917
col<size?
Yes
No
918
pos2←(size2x(col+1))/|angle|
int2←pos2>>log2_size
frac2←pos2&(size-1)
a←(size-frac2) x refV[int2+row+1]
b←frac2 x refV[int2+row+2]
v←(a+b)>>log2_size
pred[row][col]←v
919
col←0
906
v←refH[int+col+1]
pred[row][col]←v
col←col+1
911
col<size?
912
Yes
No
col←0
905
910
a←(size-frac) x refH[int+col+1
b←frac x refH[int+col+2]
v←(a+b)>>log2_size
pred[row][col]←v
907
col←col+1
908
col<size?
909
Yes
No
row←row+1
920
row<size?
921
Yes
No
End

Fig. 9

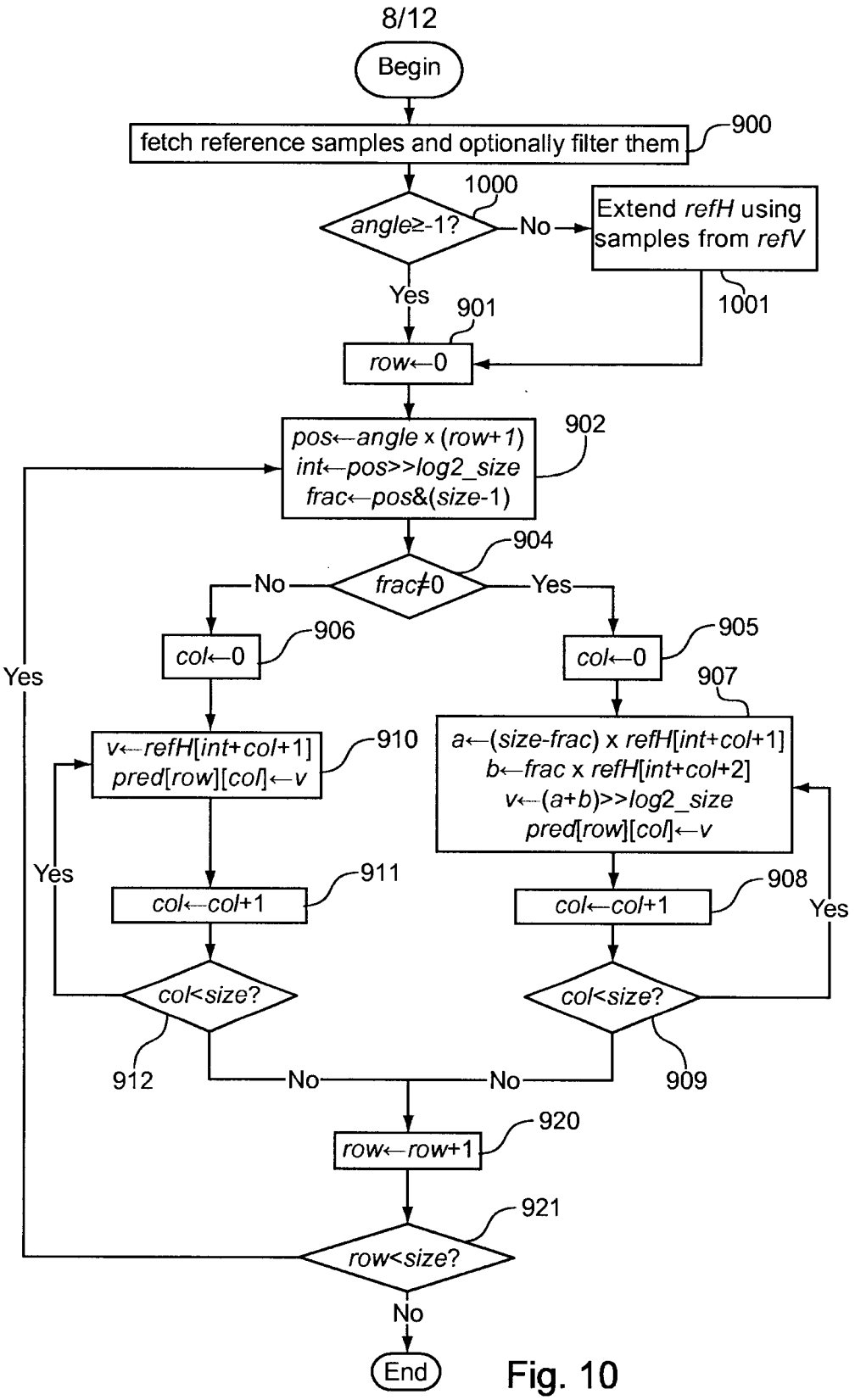

Begin
fetch reference samples and optionally filter them
900
angle≥-1?
1000
No
Extend refH using samples from refV
1001
Yes
901
row←0
pos←angle x (row+1)
int←pos>>log2_size
frac←pos&(size-1)
902
frac≠0
904
No
Yes
col←0
906
col←0
905
907
v←refH[int+col+1]
pred[row][col]←v
910
a←(size-frac) x refH[int+col+1]
b←frac x refH[int+col+2]
v←(a+b)>>log2_size
pred[row][col]←v
Yes
col←col+1
911
col←col+1
908
Yes
col<size?
912
col<size?
909
No
No
row←row+1
920
921
row<size?
Yes
No
End

Fig. 10

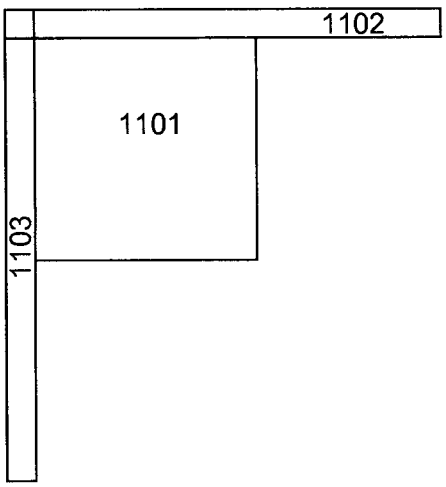


Fig. 11A

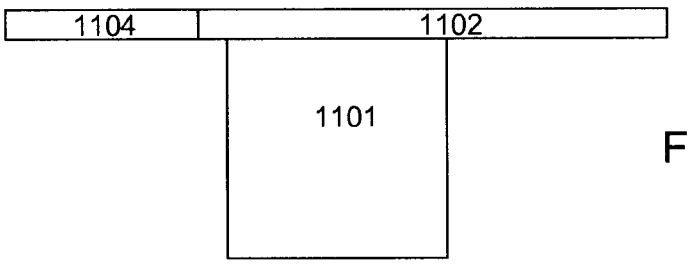


Fig. 11B

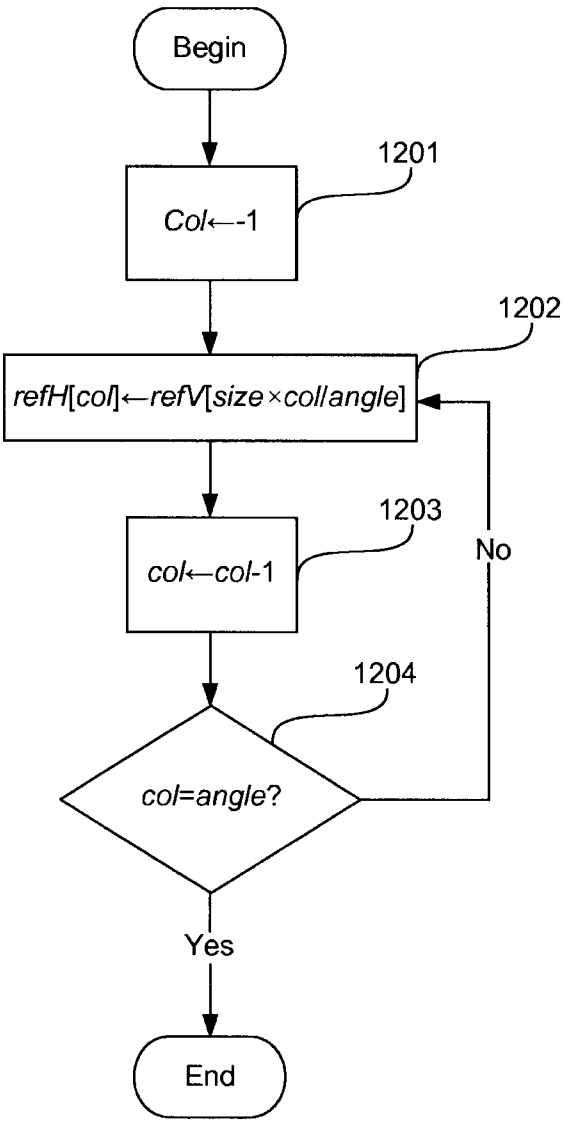

Begin
1201
Col←-1
1202
refH[col]←refV[size×col/angle]
1203
col←col-1
No
1204
col=angle?
Yes
End

Fig. 12

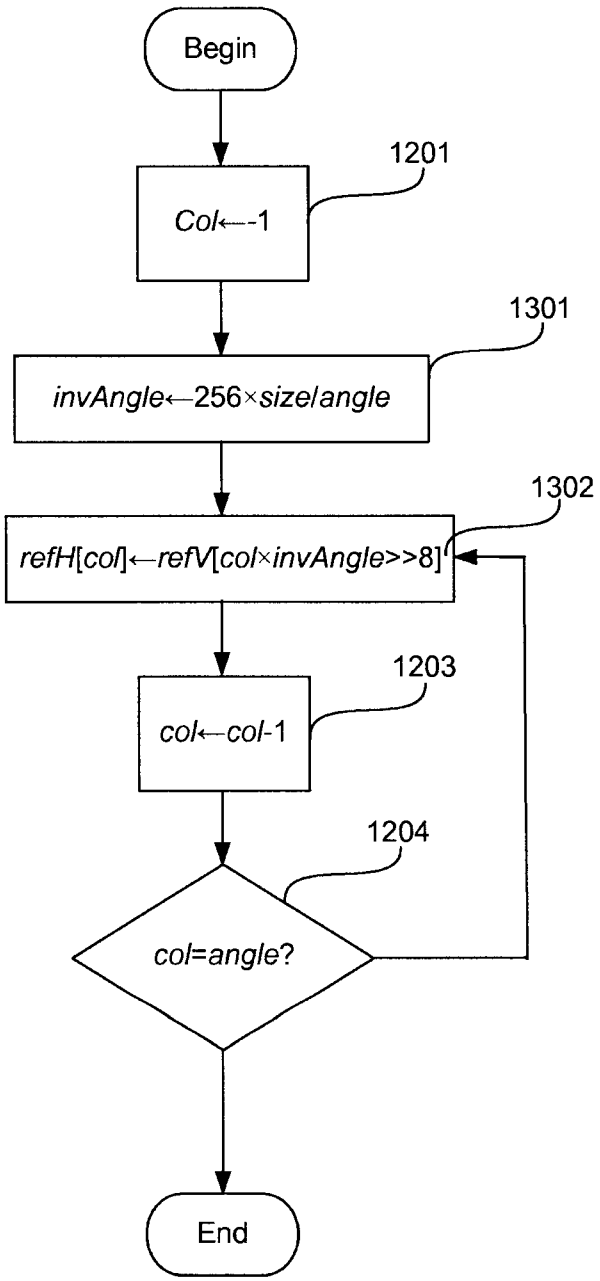

Begin
1201
Col←-1
1301
invAngle←256×sizeIangle
1302
refH[col]←refV[col×invAngle>>8]
1203
col←col-1
1204
col=angle?
End

Fig. 13

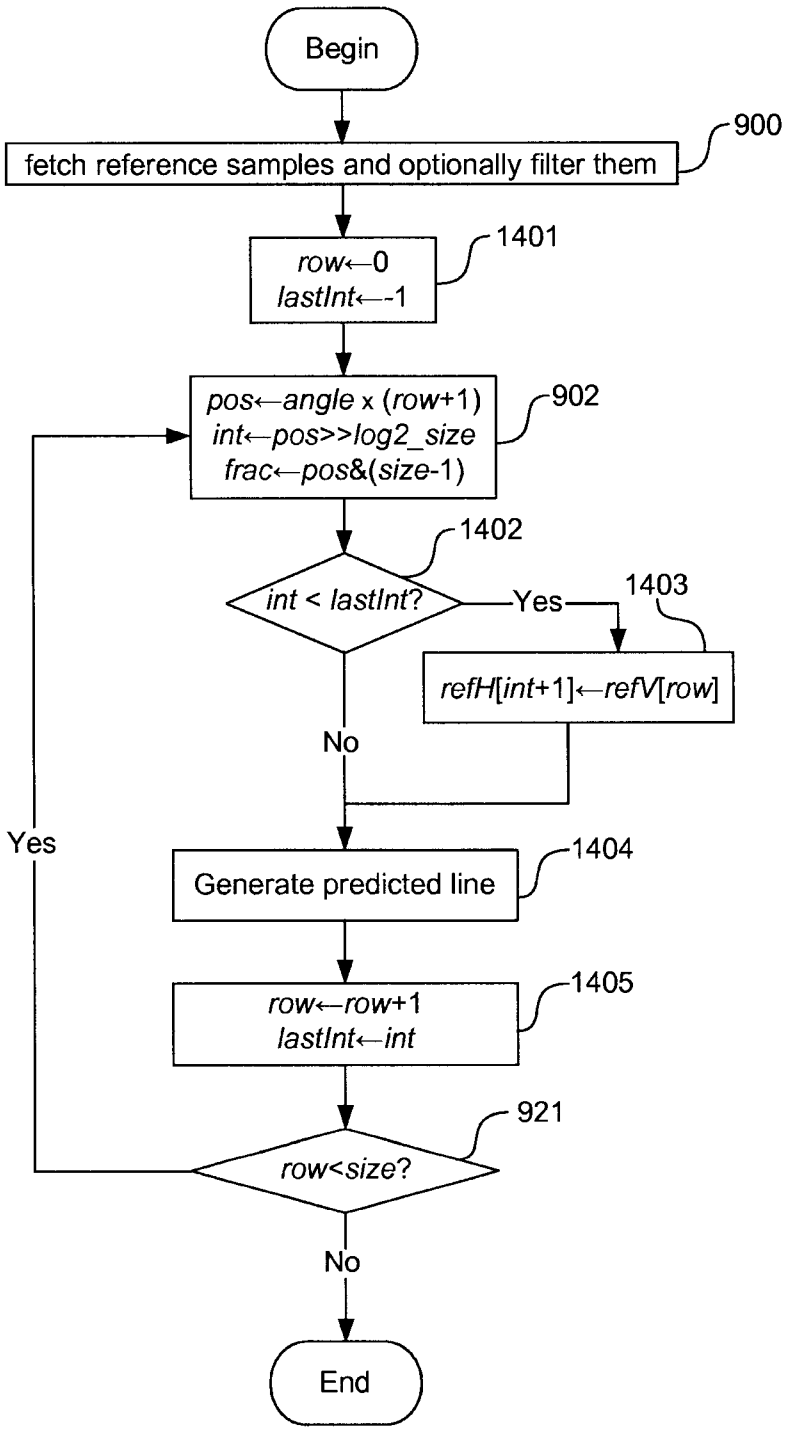

Begin
fetch reference samples and optionally filter them
900
row←0
lastInt←-1
1401
pos←angle x (row+1)
int←pos>>log2_size
frac←pos&(size-1)
902
int < lastInt?
1402
Yes
1403
refH[int+1]←refV[row]
No
Yes
Generate predicted line
1404
row←row+1
lastInt←int
1405
row<size?
921
No
End

Fig. 14