(19) DANMARK

(10) **DK/EP 3399756 T3**

Patent- og
Varemærkestyrelsen

(12) Oversættelse af
europæisk patentskrift

(51) Int.Cl.: ***H 04 N 19/597 (2014.01)*** ***H 04 N 13/00 (2018.01)*** ***H 04 N 19/46 (2014.01)***
***H 04 N 19/61 (2014.01)*** ***H 04 N 19/70 (2014.01)***

(45) Oversættelsen bekendtgjort den: **2020-10-12**

(80) Dato for Den Europæiske Patentmyndigheds
bekendtgørelse om meddelelse af patentet: **2020-09-09**

(86) Europæisk ansøgning nr.: **18172263.8**

(86) Europæisk indleveringsdag: **2008-04-11**

(87) Den europæiske ansøgnings publiceringsdag: **2018-11-07**

(30) Prioritet: **2007-04-12 US 923014 P** **2007-04-20 US 925400 P**

(62) Stamansøgningsnr: **08742812.4**

(84) Designerede stater: **AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MT NL NO PL PT RO SE SI SK TR**

(73) Patenthaver: **Dolby International AB, Apollo Building, 3E , Herikerbergweg 1-35, 1101 CN Amsterdam Zuidoost, Holland**

(72) Opfinder: **PANDIT, Purvin Bibhas, 23 Pear Tree Lane, Franklin Park, New Jersey 08823, USA**
**TIAN, Dong, 20214 Heather Drive, West Windsor, New Jersey 08550, USA**
**YIN, Peng, c/o Dolby Laboratories, Inc., 432 Lakeside Drive, 94085-4703, Sunnyvale, California, USA**

(74) Fuldmægtig i Danmark: **Plougmann Vingtoft A/S, Strandvejen 70, 2900 Hellerup, Danmark**

(54) Benævnelse: **TILING I VIDEOKODNING OG -AFKODNING**

(56) Fremdragne publikationer:
**EP-A- 1 581 003**
**US-A- 5 915 091**
**PANDIT P ET AL: "H.264/AVC extension for MVC using SEI message", 23. JVT MEETING; 80. MPEG MEETING; 21-04-2007 - 27-04-2007; SAN JOSÃ¿ CR , US; (JOINT VIDEO TEAM OF ISO/IEC JTC1/SC29/WG11 AND ITU-T SG.16 ),, no. JVT-W067, 23 April 2007 (2007-04-23), XP030007027, ISSN: 0000-0154**
**LAI P ET AL: "MVC using SEI", 24. JVT MEETING; 81. MPEG MEETING; 29-6-2007 - 5-7-2007; GENEVA, CH;(JOINT VIDEO TEAM OF ISO/IEC JTC1/SC29/WG11 AND ITU-T SG.16 ),, no. JVT-X061, 30 June 2007 (2007-06-30), XP030007168, ISSN: 0000-0082**
**MEESSEN J ET AL: "Content browsing and semantic context viewing through JPEG 2000-based scalable video summary", IEE PROCEEDINGS: VISION, IMAGE AND SIGNAL PROCESSING, INSTITUTION OF ELECTRICAL ENGINEERS, GB, vol. 153, no. 3, 8 June 2006 (2006-06-08), pages 274-283, XP006026523, ISSN: 1350-245X**
**EMIN MARTINIAN ET AL: "Extensions of H.264/AVC for Multiview Video Compression", IMAGE PROCESSING, 2006 IEEE INTERNATIONAL CONFERENCE ON, IEEE, PI, 1 October 2006 (2006-10-01), pages 2981-2984, XP031049303, ISBN: 978-1-4244-0480-3**

# DESCRIPTION

## TECHNICAL FIELD

**[0001]** The present principles relate generally to video encoding and/or decoding.

## BACKGROUND

**[0002]** Video display manufacturers may use a framework of arranging or tiling different views on a single frame. The views may then be extracted from their respective locations and rendered.
EP 1 581 003 A1 describes a system for monitoring a plurality of video signals in an internal video network, such as a broadcast recording environment, or security camera network. The system comprises a picture monitor having a screen on which a mosaic video image containing the video signals from a plurality of cameras or other video devices is displayed. Selection of a video signal from the mosaic image can be performed to display a full screen version of that signal.

## SUMMARY

**[0003]** The invention is set out in the appended claims.

**[0004]** According to a general example, a video picture is accessed that includes multiple pictures combined into a single picture. Information is accessed indicating how the multiple pictures in the accessed video picture are combined. The video picture is decoded to provide a decoded representation of the combined multiple pictures. The accessed information and the decoded video picture are provided as output.

**[0005]** According to another general example, information is generated indicating how multiple pictures included in a video picture are combined into a single picture. The video picture is encoded to provide an encoded representation of the combined multiple pictures. The generated information and encoded video picture are provided as output.

**[0006]** According to another general example, a signal or signal structure includes information indicating how multiple pictures included in a single video picture are combined into the single video picture. The signal or signal structure also includes an encoded representation of the combined multiple pictures.

**[0007]** According to another general example, a video picture is accessed that includes multiple pictures combined into a single picture. Information is accessed that indicates how the

multiple pictures in the accessed video picture are combined. The video picture is decoded to provide a decoded representation of at least one of the multiple pictures. The accessed information and the decoded representation are provided as output.

**[0008]** According to another general example, a video picture is accessed that includes multiple pictures combined into a single picture. Information is accessed that indicates how the multiple pictures in the accessed video picture are combined. The video picture is decoded to provide a decoded representation of the combined multiple pictures. User input is received that selects at least one of the multiple pictures for display. A decoded output of the at least one selected picture is provided, the decoded output being provided based on the accessed information, the decoded representation, and the user input.

**[0009]** The details of one or more implementations are set forth in the accompanying drawings and the description below. Even if described in one particular manner, it should be clear that implementations may be configured or embodied in various manners. For example, an implementation may be performed as a method, or embodied as an apparatus configured to perform a set of operations, or embodied as an apparatus storing instructions for performing a set of operations, or embodied in a signal. Other aspects and features will become apparent from the following detailed description considered in conjunction with the accompanying drawings and the claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0010]**

FIG. 1 is a diagram showing an example of four views tiled on a single frame;

FIG. 2 is a diagram showing an example of four views flipped and tiled on a single frame;

FIG. 3 shows a block diagram for a video encoder to which the present principles may be applied, in accordance with an embodiment of the present principles;

FIG. 4 shows a block diagram for a video decoder to which the present principles may be applied, in accordance with an embodiment of the present principles;

FIG. 5 is a flow diagram for a method for encoding pictures for a plurality of views using the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 6 is a flow diagram for a method for decoding pictures for a plurality of views using the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 7 is a flow diagram for a method for encoding pictures for a plurality of views and depths using the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 8 is a flow diagram for a method for decoding pictures for a plurality of views and depths

using the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 9 is a diagram showing an example of a depth signal, in accordance with an embodiment of the present principles;

FIG. 10 is a diagram showing an example of a depth signal added as a tile, in accordance with an embodiment of the present principles;

FIG. 11 is a diagram showing an example of 5 views tiled on a single frame, in accordance with an embodiment of the present principles.

FIG. 12 is a block diagram for an exemplary Multi-view Video Coding (MVC) encoder to which the present principles may be applied, in accordance with an embodiment of the present principles;

FIG. 13 is a block diagram for an exemplary Multi-view Video Coding (MVC) decoder to which the present principles may be applied, in accordance with an embodiment of the present principles;

FIG. 14 is a flow diagram for a method for processing pictures for a plurality of views in preparation for encoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 15 is a flow diagram for a method for encoding pictures for a plurality of views using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 16 is a flow diagram for a method for processing pictures for a plurality of views in preparation for decoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 17 is a flow diagram for a method for decoding pictures for a plurality of views using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 18 is a flow diagram for a method for processing pictures for a plurality of views and depths in preparation for encoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 19 is a flow diagram for a method for encoding pictures for a plurality of views and depths using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 20 is a flow diagram for a method for processing pictures for a plurality of views and depths in preparation for decoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 21 is a flow diagram for a method for decoding pictures for a plurality of views and depths using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard, in accordance with an embodiment of the present principles;

FIG. 22 is a diagram showing tiling examples at the pixel level, in accordance with an embodiment of the present principles; and

FIG. 23 shows a block diagram for a video processing device to which the present principles may be applied, in accordance with an embodiment of the present principles.

## DETAILED DESCRIPTION

**[0011]** Various implementations are directed to methods and apparatus for view tiling in video encoding and decoding. It will thus be appreciated that those skilled in the art will be able to devise various arrangements that, although not explicitly described or shown herein, embody the present principles.

**[0012]** All examples and conditional language recited herein are intended for pedagogical purposes to aid the reader in understanding the present principles and the concepts contributed by the inventor(s) to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions.

**[0013]** Moreover, all statements herein reciting principles, aspects, and embodiments of the present principles, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

**[0014]** Thus, for example, it will be appreciated by those skilled in the art that the block diagrams presented herein represent conceptual views of illustrative circuitry embodying the present principles. Similarly, it will be appreciated that any flow charts, flow diagrams, state transition diagrams, pseudocode, and the like represent various processes which may be substantially represented in computer readable media and so executed by a computer or processor, whether or not such computer or processor is explicitly shown.

**[0015]** The functions of the various elements shown in the figures may be provided through the use of dedicated hardware as well as hardware capable of executing software in association with appropriate software. When provided by a processor, the functions may be provided by a single dedicated processor, by a single shared processor, or by a plurality of individual processors, some of which may be shared. Moreover, explicit use of the term "processor" or "controller" should not be construed to refer exclusively to hardware capable of

executing software, and may implicitly include, without limitation, digital signal processor ("DSP") hardware, read-only memory ("ROM") for storing software, random access memory ("RAM"), and non-volatile storage.

**[0016]** Other hardware, conventional and/or custom, may also be included. Similarly, any switches shown in the figures are conceptual only. Their function may be carried out through the operation of program logic, through dedicated logic, through the interaction of program control and dedicated logic, or even manually, the particular technique being selectable by the implementer as more specifically understood from the context.

**[0017]** In the claims hereof, any element expressed as a means for performing a specified function is intended to encompass any way of performing that function including, for example, a) a combination of circuit elements that performs that function or b) software in any form, including, therefore, firmware, microcode or the like, combined with appropriate circuitry for executing that software to perform the function. The present principles as defined by such claims reside in the fact that the functionalities provided by the various recited means are combined and brought together in the manner which the claims call for. It is thus regarded that any means that can provide those functionalities are equivalent to those shown herein.

**[0018]** Reference in the specification to "one embodiment" (or "one implementation") or "an embodiment" (or "an implementation") of the present principles means that a particular feature, structure, characteristic, and so forth described in connection with the embodiment is included in at least one embodiment of the present principles. Thus, the appearances of the phrase "in one embodiment" or "in an embodiment" appearing in various places throughout the specification are not necessarily all referring to the same embodiment.

**[0019]** It is to be appreciated that the use of the terms "and/or" and "at least one of', for example, in the cases of "A and/or B" and "at least one of A and B", is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of both options (A and B). As a further example, in the cases of "A, B, and/or C" and "at least one of A, B, and C", such phrasing is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of the third listed option (C) only, or the selection of the first and the second listed options (A and B) only, or the selection of the first and third listed options (A and C) only, or the selection of the second and third listed options (B and C) only, or the selection of all three options (A and B and C). This may be extended, as readily apparent by one of ordinary skill in this and related arts, for as many items listed.

**[0020]** Moreover, it is to be appreciated that while one or more embodiments of the present principles are described herein with respect to the MPEG-4 AVC standard, the present principles are not limited to solely this standard and, thus, may be utilized with respect to other standards, recommendations, and extensions thereof, particularly video coding standards, recommendations, and extensions thereof, including extensions of the MPEG-4 AVC standard.

**[0021]** Further, it is to be appreciated that while one or more other embodiments of the present principles are described herein with respect to the multi-view video coding extension of the MPEG-4 AVC standard, the present principles are not limited to solely this extension and/or this standard and, thus, may be utilized with respect to other video coding standards, recommendations, and extensions thereof relating to multi-view video coding. Multi-view video coding (MVC) is the compression framework for the encoding of multi-view sequences. A Multi-view Video Coding (MVC) sequence is a set of two or more video sequences that capture the same scene from a different view point.

**[0022]** Also, it is to be appreciated that while one or more other embodiments of the present principles are described herein that use depth information with respect to video content, the present principles are not limited to such embodiments and, thus, other embodiments may be implemented that do not use depth information.

**[0023]** Additionally, as used herein, "high level syntax" refers to syntax present in the bitstream that resides hierarchically above the macroblock layer. For example, high level syntax, as used herein, may refer to, but is not limited to, syntax at the slice header level, Supplemental Enhancement Information (SEI) level, Picture Parameter Set (PPS) level, Sequence Parameter Set (SPS) level, View Parameter Set (VPS), and Network Abstraction Layer (NAL) unit header level.

**[0024]** In the current implementation of multi-video coding (MVC) based on the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Moving Picture Experts Group-4 (MPEG-4) Part 10 Advanced Video Coding (AVC) standard/International Telecommunication Union, Telecommunication Sector (ITU-T) H.264 Recommendation (hereinafter the "MPEG-4 AVC Standard"), the reference software achieves multi-view prediction by encoding each view with a single encoder and taking into consideration the cross-view references. Each view is coded as a separate bitstream by the encoder in its original resolution and later all the bitstreams are combined to form a single bitstream which is then decoded. Each view produces a separate YUV decoded output.

**[0025]** Another approach for multi-view prediction involves grouping a set of views into pseudo views. In one example of this approach, we can tile the pictures from every N views out of the total M views (sampled at the same time) on a larger frame or a super frame with possible downsampling or other operations. Turning to FIG. 1, an example of four views tiled on a single frame is indicated generally by the reference numeral 100. All four views are in their normal orientation.

**[0026]** Turning to FIG. 2, an example of four views flipped and tiled on a single frame is indicated generally by the reference numeral 200. The top-left view is in its normal orientation. The top-right view is flipped horizontally. The bottom-left view is flipped vertically. The bottom-right view is flipped both horizontally and vertically. Thus, if there are four views, then a picture from each view is arranged in a super-frame like a tile. This results in a single un-coded input sequence with a large resolution.

**[0027]** Alternatively, we can downsample the image to produce a smaller resolution. Thus, we create multiple sequences which each include different views that are tiled together. Each such sequence then forms a pseudo view, where each pseudo view includes N different tiled views. FIG. 1 shows one pseudo-view, and FIG. 2 shows another pseudo-view. These pseudo views can then be encoded using existing video coding standards such as the ISO/IEC MPEG-2 Standard and the MPEG-4 AVC Standard.

**[0028]** Yet another approach for multi-view prediction simply involves encoding the different views independently using a new standard and, after decoding, tiling the views as required by the player.

**[0029]** Further, in another approach, the views can also be tiled in a pixel wise way. For example, in a super view that is composed of four views, pixel (x, y) may be from view 0, while pixel (x+1, y) may be from view 1, pixel (x, y+1) may be from view 2, and pixel (x+1, y+1) may be from view 3.

**[0030]** Many displays manufacturers use such a frame work of arranging or tiling different views on a single frame and then extracting the views from their respective locations and rendering them. In such cases, there is no standard way to determine if the bitstream has such a property. Thus, if a system uses the method of tiling pictures of different views in a large frame, then the method of extracting the different views is proprietary.

**[0031]** However, there is no standard way to determine if the bitstream has such a property. We propose high level syntax in order to facilitate the renderer or player to extract such information in order to assist in display or other post-processing. It is also possible the sub-pictures have different resolutions and some upsampling may be needed to eventually render the view. The user may want to have the method of upsample indicated in the high level syntax as well. Additionally, parameters to change the depth focus can also be transmitted.

**[0032]** In an embodiment, we propose a new Supplemental Enhancement Information (SEI) message for signaling multi-view information in a MPEG-4 AVC Standard compatible bitstream where each picture includes sub-pictures which belong to a different view. The embodiment is intended, for example, for the easy and convenient display of multi-view video streams on three-dimensional (3D) monitors which may use such a framework. The concept can be extended to other video coding standards and recommendations signaling such information using high level syntax.

**[0033]** Moreover, in an embodiment, we propose a signaling method of how to arrange views before they are sent to the multi-view video encoder and/or decoder. Advantageously, the embodiment may lead to a simplified implementation of the multi-view coding, and may benefit the coding efficiency. Certain views can be put together and form a pseudo view or super view and then the tiled super view is treated as a normal view by a common multi-view video encoder and/or decoder, for example, as per the current MPEG-4 AVC Standard based

implementation of multi-view video coding. A new flag is proposed in the Sequence Parameter Set (SPS) extension of multi-view video coding to signal the use of the technique of pseudo views. The embodiment is intended for the easy and convenient display of multi-view video streams on 3D monitors which may use such a framework.

**Encoding/decoding using a single-view video encoding/decoding standard/recommendation**

**[0034]** In the current implementation of multi-video coding (MVC) based on the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Moving Picture Experts Group-4 (MPEG-4) Part 10 Advanced Video Coding (AVC) standard/International Telecommunication Union, Telecommunication Sector (ITU-T) H.264 Recommendation (hereinafter the "MPEG-4 AVC Standard"), the reference software achieves multi-view prediction by encoding each view with a single encoder and taking into consideration the cross-view references. Each view is coded as a separate bitstream by the encoder in its original resolution and later all the bitstreams are combined to form a single bitstream which is then decoded. Each view produces a separate YUV decoded output.

**[0035]** Another approach for multi-view prediction involves tiling the pictures from each view (sampled at the same time) on a larger frame or a super frame with a possible downsampling operation. Turning to FIG. 1, an example of four views tiled on a single frame is indicated generally by the reference numeral 100. Turning to FIG. 2, an example of four views flipped and tiled on a single frame is indicated generally by the reference numeral 200. Thus, if there are four views, then a picture from each view is arranged in a super-frame like a tile. This results in a single un-coded input sequence with a large resolution. This signal can then be encoded using existing video coding standards such as the ISO/IEC MPEG-2 Standard and the MPEG-4 AVC Standard.

**[0036]** Yet another approach for multi-view prediction simply involves encoding the different views independently using a new standard and, after decoding, tiling the views as required by the player.

**[0037]** Many displays manufacturers use such a frame work of arranging or tiling different views on a single frame and then extracting the views from their respective locations and rendering them. In such cases, there is no standard way to determine if the bitstream has such a property. Thus, if a system uses the method of tiling pictures of different views in a large frame, then the method of extracting the different views is proprietary.

**[0038]** Turning to FIG. 3, a video encoder capable of performing video encoding in accordance with the MPEG-4 AVC standard is indicated generally by the reference numeral 300.

**[0039]** The video encoder 300 includes a frame ordering buffer 310 having an output in signal communication with a non-inverting input of a combiner 385. An output of the combiner 385 is

connected in signal communication with a first input of a transformer and quantizer 325. An output of the transformer and quantizer 325 is connected in signal communication with a first input of an entropy coder 345 and a first input of an inverse transformer and inverse quantizer 350. An output of the entropy coder 345 is connected in signal communication with a first non-inverting input of a combiner 390. An output of the combiner 390 is connected in signal communication with a first input of an output buffer 335.

**[0040]** A first output of an encoder controller 305 is connected in signal communication with a second input of the frame ordering buffer 310, a second input of the inverse transformer and inverse quantizer 350, an input of a picture-type decision module 315, an input of a macroblock-type (MB-type) decision module 320, a second input of an intra prediction module 360, a second input of a deblocking filter 365, a first input of a motion compensator 370, a first input of a motion estimator 375, and a second input of a reference picture buffer 380.

**[0041]** A second output of the encoder controller 305 is connected in signal communication with a first input of a Supplemental Enhancement Information (SEI) inserter 330, a second input of the transformer and quantizer 325, a second input of the entropy coder 345, a second input of the output buffer 335, and an input of the Sequence Parameter Set (SPS) and Picture Parameter Set (PPS) inserter 340.

**[0042]** A first output of the picture-type decision module 315 is connected in signal communication with a third input of a frame ordering buffer 310. A second output of the picture-type decision module 315 is connected in signal communication with a second input of a macroblock-type decision module 320.

**[0043]** An output of the Sequence Parameter Set (SPS) and Picture Parameter Set (PPS) inserter 340 is connected in signal communication with a third non-inverting input of the combiner 390. An output of the SEI Inserter 330 is connected in signal communication with a second non-inverting input of the combiner 390.

**[0044]** An output of the inverse quantizer and inverse transformer 350 is connected in signal communication with a first non-inverting input of a combiner 319. An output of the combiner 319 is connected in signal communication with a first input of the intra prediction module 360 and a first input of the deblocking filter 365. An output of the deblocking filter 365 is connected in signal communication with a first input of a reference picture buffer 380. An output of the reference picture buffer 380 is connected in signal communication with a second input of the motion estimator 375 and with a first input of a motion compensator 370. A first output of the motion estimator 375 is connected in signal communication with a second input of the motion compensator 370. A second output of the motion estimator 375 is connected in signal communication with a third input of the entropy coder 345.

**[0045]** An output of the motion compensator 370 is connected in signal communication with a first input of a switch 397. An output of the intra prediction module 360 is connected in signal communication with a second input of the switch 397. An output of the macroblock-type

decision module 320 is connected in signal communication with a third input of the switch 397 in order to provide a control input to the switch 397. The third input of the switch 397 determines whether or not the "data" input of the switch (as compared to the control input, i.e., the third input) is to be provided by the motion compensator 370 or the intra prediction module 360. The output of the switch 397 is connected in signal communication with a second non-inverting input of the combiner 319 and with an inverting input of the combiner 385.

**[0046]** Inputs of the frame ordering buffer 310 and the encoder controller 105 are available as input of the encoder 300, for receiving an input picture 301. Moreover, an input of the Supplemental Enhancement Information (SEI) inserter 330 is available as an input of the encoder 300, for receiving metadata. An output of the output buffer 335 is available as an output of the encoder 300, for outputting a bitstream.

**[0047]** Turning to FIG. 4, a video decoder capable of performing video decoding in accordance with the MPEG-4 AVC standard is indicated generally by the reference numeral 400.

**[0048]** The video decoder 400 includes an input buffer 410 having an output connected in signal communication with a first input of the entropy decoder 445. A first output of the entropy decoder 445 is connected in signal communication with a first input of an inverse transformer and inverse quantizer 450. An output of the inverse transformer and inverse quantizer 450 is connected in signal communication with a second non-inverting input of a combiner 425. An output of the combiner 425 is connected in signal communication with a second input of a deblocking filter 465 and a first input of an intra prediction module 460. A second output of the deblocking filter 465 is connected in signal communication with a first input of a reference picture buffer 480. An output of the reference picture buffer 480 is connected in signal communication with a second input of a motion compensator 470.

**[0049]** A second output of the entropy decoder 445 is connected in signal communication with a third input of the motion compensator 470 and a first input of the deblocking filter 465. A third output of the entropy decoder 445 is connected in signal communication with an input of a decoder controller 405. A first output of the decoder controller 405 is connected in signal communication with a second input of the entropy decoder 445. A second output of the decoder controller 405 is connected in signal communication with a second input of the inverse transformer and inverse quantizer 450. A third output of the decoder controller 405 is connected in signal communication with a third input of the deblocking filter 465. A fourth output of the decoder controller 405 is connected in signal communication with a second input of the intra prediction module 460, with a first input of the motion compensator 470, and with a second input of the reference picture buffer 480.

**[0050]** An output of the motion compensator 470 is connected in signal communication with a first input of a switch 497. An output of the intra prediction module 460 is connected in signal communication with a second input of the switch 497. An output of the switch 497 is connected in signal communication with a first non-inverting input of the combiner 425.

**[0051]** An input of the input buffer 410 is available as an input of the decoder 400, for receiving an input bitstream. A first output of the deblocking filter 465 is available as an output of the decoder 400, for outputting an output picture.

**[0052]** Turning to FIG. 5, an exemplary method for encoding pictures for a plurality of views using the MPEG-4 AVC Standard is indicated generally by the reference numeral 500.

**[0053]** The method 500 includes a start block 502 that passes control to a function block 504. The function block 504 arranges each view at a particular time instance as a sub-picture in tile format, and passes control to a function block 506. The function block 506 sets a syntax element num_coded_views_minus1, and passes control to a function block 508. The function block 508 sets syntax elements org_pic_width_in_mbs_minus1 and org_pic_height_in_mbs_minus1, and passes control to a function block 510. The function block 510 sets a variable i equal to zero, and passes control to a decision block 512. The decision block 512 determines whether or not the variable i is less than the number of views. If so, then control is passed to a function block 514. Otherwise, control is passed to a function block 524.

**[0054]** The function block 514 sets a syntax element view_id[i], and passes control to a function block 516. The function block 516 sets a syntax element num_parts[view_id[i]], and passes control to a function block 518. The function block 518 sets a variable j equal to zero, and passes control to a decision block 520. The decision block 520 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[view_id[i]]. If so, then control is passed to a function block 522. Otherwise, control is passed to a function block 528.

**[0055]** The function block 522 sets the following syntax elements, increments the variable j, and then returns control to the decision block 520: depth_flag[view_id[i]][j]; flip_dir[view_id[i]][j]; loc_left_offset[view_id[i]][j]; loc_top_offset[view_id[i]][j]; frame_crop_left_offset[view_id[i]][j]; frame_crop_right_offset[view_id[i]][j]; frame_crop_top_offset[view_id[i]][j]; and frame_crop_bottom_offset[view_id[i]][j].

**[0056]** The function block 528 sets a syntax element upsample_view_flag[view_id[i]], and passes control to a decision block 530. The decision block 530 determines whether or not the current value of the syntax element upsample_view_flag[view_id[i]] is equal to one. If so, then control is passed to a function block 532. Otherwise, control is passed to a decision block 534.

**[0057]** The function block 532 sets a syntax element upsample_filter[view_id[i]], and passes control to the decision block 534.

**[0058]** The decision block 534 determines whether or not the current value of the syntax element upsample_filter[view_id[i]] is equal to three. If so, then control is passed to a function block 536. Otherwise, control is passed to a function block 540.

**[0059]** The function block 536 sets the following syntax elements and passes control to a

function block 538: vert_dim[view_id[i]]; hor_dim[view_id[i]]; and quantizer[view_id[i]].

**[0060]** The function block 538 sets the filter coefficients for each YUV component, and passes control to the function block 540.

**[0061]** The function block 540 increments the variable i, and returns control to the decision block 512.

**[0062]** The function block 524 writes these syntax elements to at least one of the Sequence Parameter Set (SPS), Picture Parameter Set (PPS), Supplemental Enhancement Information (SEI) message, Network Abstraction Layer (NAL) unit header, and slice header, and passes control to a function block 526. The function block 526 encodes each picture using the MPEG-4 AVC Standard or other single view codec, and passes control to an end block 599.

**[0063]** Turning to FIG. 6, an exemplary method for decoding pictures for a plurality of views using the MPEG-4 AVC Standard is indicated generally by the reference numeral 600.

**[0064]** The method 600 includes a start block 602 that passes control to a function block 604. The function block 604 parses the following syntax elements from at least one of the Sequence Parameter Set (SPS), Picture Parameter Set (PPS), Supplemental Enhancement Information (SEI) message, Network Abstraction Layer (NAL) unit header, and slice header, and passes control to a function block 606. The function block 606 parses a syntax element num_coded_views_minus1, and passes control to a function block 608. The function block 608 parses syntax elements org_pic_width_in_mbs_minus1 and org_pic_height_in_mbs_minus1, and passes control to a function block 610. The function block 610 sets a variable i equal to zero, and passes control to a decision block 612. The decision block 612 determines whether or not the variable i is less than the number of views. If so, then control is passed to a function block 614. Otherwise, control is passed to a function block 624.

**[0065]** The function block 614 parses a syntax element view_id[i], and passes control to a function block 616. The function block 616 parses a syntax element num_parts_minus1[view_id[i]], and passes control to a function block 618. The function block 618 sets a variable j equal to zero, and passes control to a decision block 620. The decision block 620 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[view_id[i]]. If so, then control is passed to a function block 622. Otherwise, control is passed to a function block 628.

**[0066]** The function block 622 parses the following syntax elements, increments the variable j, and then returns control to the decision block 620: depth_flag[view_id[i]][j]; flip_dir[view_id[i]][j]; loc_left_offset[view_id[i]][j]; loc_top_offset[view_id[i]][j]; frame_crop_left_offset[view_id[i]][j]; frame_crop_right_offset[view_id[i]][j]; frame_crop_top_offset[view_id[i]][j]; and frame_crop_bottom_offset[view_id[i]][j].

**[0067]** The function block 628 parses a syntax element upsample_view_flag[view_id[i]], and

passes control to a decision block 630. The decision block 630 determines whether or not the current value of the syntax element upsample_view_flag[view_id[i]] is equal to one. If so, then control is passed to a function block 632. Otherwise, control is passed to a decision block 634.

**[0068]** The function block 632 parses a syntax element upsample_filter[view_id[i]], and passes control to the decision block 634.

**[0069]** The decision block 634 determines whether or not the current value of the syntax element upsample_filter[view_id[i]] is equal to three. If so, then control is passed to a function block 636. Otherwise, control is passed to a function block 640.

**[0070]** The function block 636 parses the following syntax elements and passes control to a function block 638: vert_dim[view_id[i]]; hor_dim[view_id[i]]; and quantizer[view_id[i]].

**[0071]** The function block 638 parses the filter coefficients for each YUV component, and passes control to the function block 640.

**[0072]** The function block 640 increments the variable i, and returns control to the decision block 612.

**[0073]** The function block 624 decodes each picture using the MPEG-4 AVC Standard or other single view codec, and passes control to a function block 626. The function block 626 separates each view from the picture using the high level syntax, and passes control to an end block 699.

**[0074]** Turning to FIG. 7, an exemplary method for encoding pictures for a plurality of views and depths using the MPEG-4 AVC Standard is indicated generally by the reference numeral 700.

**[0075]** The method 700 includes a start block 702 that passes control to a function block 704. The function block 704 arranges each view and corresponding depth at a particular time instance as a sub-picture in tile format, and passes control to a function block 706. The function block 706 sets a syntax element num_coded_views_minus1, and passes control to a function block 708. The function block 708 sets syntax elements org_pic_width_in_mbs_minus1 and org_pic_height_in_mbs_minus1, and passes control to a function block 710. The function block 710 sets a variable i equal to zero, and passes control to a decision block 712. The decision block 712 determines whether or not the variable i is less than the number of views. If so, then control is passed to a function block 714. Otherwise, control is passed to a function block 724.

**[0076]** The function block 714 sets a syntax element view_id[i], and passes control to a function block 716. The function block 716 sets a syntax element num_parts[view_id[i]], and passes control to a function block 718. The function block 718 sets a variable j equal to zero, and passes control to a decision block 720. The decision block 720 determines whether or not

the current value of the variable j is less than the current value of the syntax element num_parts[view_id[i]]. If so, then control is passed to a function block 722. Otherwise, control is passed to a function block 728.

**[0077]** The function block 722 sets the following syntax elements, increments the variable j, and then returns control to the decision block 720: depth_flag[view_id[i]][j]; flip_dir[view_id[i]][j]; loc_left_offset[view_id[i]][j]; loc_top_offset[view_id[i]][j]; frame_crop_left_offset[view_id[i]][j]; frame_crop_right_offset[view_id[i]][j]; frame_crop_top_offset[view_id[i]][j]; and frame_crop_bottom_offset[view_id[i]][j].

**[0078]** The function block 728 sets a syntax element upsample_view_flag[view_id[i]], and passes control to a decision block 730. The decision block 730 determines whether or not the current value of the syntax element upsample_view_flag[view_id[i]] is equal to one. If so, then control is passed to a function block 732. Otherwise, control is passed to a decision block 734.

**[0079]** The function block 732 sets a syntax element upsample_filter[view_id[i]], and passes control to the decision block 734.

**[0080]** The decision block 734 determines whether or not the current value of the syntax element upsample_filter[view_id[i]] is equal to three. If so, then control is passed to a function block 736. Otherwise, control is passed to a function block 740.

**[0081]** The function block 736 sets the following syntax elements and passes control to a function block 738: vert_dim[view_id[i]]; hor_dim[view_id[i]]; and quantizer[view_id[i]].

**[0082]** The function block 738 sets the filter coefficients for each YUV component, and passes control to the function block 740.

**[0083]** The function block 740 increments the variable i, and returns control to the decision block 712.

**[0084]** The function block 724 writes these syntax elements to at least one of the Sequence Parameter Set (SPS), Picture Parameter Set (PPS), Supplemental Enhancement Information (SEI) message, Network Abstraction Layer (NAL) unit header, and slice header, and passes control to a function block 726. The function block 726 encodes each picture using the MPEG-4 AVC Standard or other single view codec, and passes control to an end block 799.

**[0085]** Turning to FIG. 8, an exemplary method for decoding pictures for a plurality of views and depths using the MPEG-4 AVC Standard is indicated generally by the reference numeral 800.

**[0086]** The method 800 includes a start block 802 that passes control to a function block 804. The function block 804 parses the following syntax elements from at least one of the Sequence Parameter Set (SPS), Picture Parameter Set (PPS), Supplemental Enhancement Information

(SEI) message, Network Abstraction Layer (NAL) unit header, and slice header, and passes control to a function block 806. The function block 806 parses a syntax element num_coded_views_minus1, and passes control to a function block 808. The function block 808 parses syntax elements org_pic_width_in_mbs_minus1 and org_pic_height_in_mbs_minus1, and passes control to a function block 810. The function block 810 sets a variable i equal to zero, and passes control to a decision block 812. The decision block 812 determines whether or not the variable i is less than the number of views. If so, then control is passed to a function block 814. Otherwise, control is passed to a function block 824.

**[0087]** The function block 814 parses a syntax element view id[i], and passes control to a function block 816. The function block 816 parses a syntax element num_parts_minus1[view_id[i]], and passes control to a function block 818. The function block 818 sets a variable j equal to zero, and passes control to a decision block 820. The decision block 820 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[view_id[i]]. If so, then control is passed to a function block 822. Otherwise, control is passed to a function block 828.

**[0088]** The function block 822 parses the following syntax elements, increments the variable j, and then returns control to the decision block 820: depth_flag[view _id[i]][j]; flip_dir[view_id[i]][j]; loc_left_offset[view_id[i]][j]; loc_top_offset[view_id[i]][j]; frame_crop_left_offset[view_id[i]][j]; frame_crop_right_offset[view_id[i]][j]; frame_crop_top_offset[view_id[i]][j]; and frame_crop_bottom_offset[view_id[i]][j].

**[0089]** The function block 828 parses a syntax element upsample_view_flag[view_id[i]], and passes control to a decision block 830. The decision block 830 determines whether or not the current value of the syntax element upsample_view_flag[view_id[i]] is equal to one. If so, then control is passed to a function block 832. Otherwise, control is passed to a decision block 834.

**[0090]** The function block 832 parses a syntax element upsample_filter[view_id[i]], and passes control to the decision block 834.

**[0091]** The decision block 834 determines whether or not the current value of the syntax element upsample_filter[view_id[i]] is equal to three. If so, then control is passed to a function block 836. Otherwise, control is passed to a function block 840.

**[0092]** The function block 836 parses the following syntax elements and passes control to a function block 838: vert_dim[view_id[i]]; hor_dim[view_id[i]]; and quantizer[view_id[i]].

**[0093]** The function block 838 parses the filter coefficients for each YUV component, and passes control to the function block 840.

**[0094]** The function block 840 increments the variable i, and returns control to the decision block 812.

**[0095]** The function block 824 decodes each picture using the MPEG-4 AVC Standard or other single view codec, and passes control to a function block 826. The function block 826 separates each view and corresponding depth from the picture using the high level syntax, and passes control to a function block 827. The function block 827 potentially performs view synthesis using the extracted view and depth signals, and passes control to an end block 899.

**[0096]** With respect to the depth used in FIGs. 7 and 8, FIG. 9 shows an example of a depth signal 900, where depth is provided as a pixel value for each corresponding location of an image (not shown). Further, FIG. 10 shows an example of two depth signals included in a tile 1000. The top-right portion of tile 1000 is a depth signal having depth values corresponding to the image on the top-left of tile 1000. The bottom-right portion of tile 1000 is a depth signal having depth values corresponding to the image on the bottom-left of tile 1000.

**[0097]** Turning to FIG. 11, an example of 5 views tiled on a single frame is indicated generally by the reference numeral 1100. The top four views are in a normal orientation. The fifth view is also in a normal orientation, but is split into two portions along the bottom of tile 1100. A left-portion of the fifth view shows the "top" of the fifth view, and a right-portion of the fifth view shows the "bottom" of the fifth view.

**Encoding/decoding using a multi-view video encoding/decoding standard/recommendation**

**[0098]** Turning to FIG. 12, an exemplary Multi-view Video Coding (MVC) encoder is indicated generally by the reference numeral 1200. The encoder 1200 includes a combiner 1205 having an output connected in signal communication with an input of a transformer 1210. An output of the transformer 1210 is connected in signal communication with an input of quantizer 1215. An output of the quantizer 1215 is connected in signal communication with an input of an entropy coder 1220 and an input of an inverse quantizer 1225. An output of the inverse quantizer 1225 is connected in signal communication with an input of an inverse transformer 1230. An output of the inverse transformer 1230 is connected in signal communication with a first non-inverting input of a combiner 1235. An output of the combiner 1235 is connected in signal communication with an input of an intra predictor 1245 and an input of a deblocking filter 1250. An output of the deblocking filter 1250 is connected in signal communication with an input of a reference picture store 1255 (for view i). An output of the reference picture store 1255 is connected in signal communication with a first input of a motion compensator 1275 and a first input of a motion estimator 1280. An output of the motion estimator 1280 is connected in signal communication with a second input of the motion compensator 1275

**[0099]** An output of a reference picture store 1260 (for other views) is connected in signal communication with a first input of a disparity estimator 1270 and a first input of a disparity compensator 1265. An output of the disparity estimator 1270 is connected in signal communication with a second input of the disparity compensator 1265.

**[0100]** An output of the entropy decoder 1220 is available as an output of the encoder 1200. A non-inverting input of the combiner 1205 is available as an input of the encoder 1200, and is connected in signal communication with a second input of the disparity estimator 1270, and a second input of the motion estimator 1280. An output of a switch 1285 is connected in signal communication with a second non-inverting input of the combiner 1235 and with an inverting input of the combiner 1205. The switch 1285 includes a first input connected in signal communication with an output of the motion compensator 1275, a second input connected in signal communication with an output of the disparity compensator 1265, and a third input connected in signal communication with an output of the intra predictor 1245.

**[0101]** A mode decision module 1240 has an output connected to the switch 1285 for controlling which input is selected by the switch 1285.

**[0102]** Turning to FIG. 13, an exemplary Multi-view Video Coding (MVC) decoder is indicated generally by the reference numeral 1300. The decoder 1300 includes an entropy decoder 1305 having an output connected in signal communication with an input of an inverse quantizer 1310. An output of the inverse quantizer is connected in signal communication with an input of an inverse transformer 1315. An output of the inverse transformer 1315 is connected in signal communication with a first non-inverting input of a combiner 1320. An output of the combiner 1320 is connected in signal communication with an input of a deblocking filter 1325 and an input of an intra predictor 1330. An output of the deblocking filter 1325 is connected in signal communication with an input of a reference picture store 1340 (for view i). An output of the reference picture store 1340 is connected in signal communication with a first input of a motion compensator 1335.

**[0103]** An output of a reference picture store 1345 (for other views) is connected in signal communication with a first input of a disparity compensator 1350.

**[0104]** An input of the entropy coder 1305 is available as an input to the decoder 1300, for receiving a residue bitstream. Moreover, an input of a mode module 1360 is also available as an input to the decoder 1300, for receiving control syntax to control which input is selected by the switch 1355. Further, a second input of the motion compensator 1335 is available as an input of the decoder 1300, for receiving motion vectors. Also, a second input of the disparity compensator 1350 is available as an input to the decoder 1300, for receiving disparity vectors.

**[0105]** An output of a switch 1355 is connected in signal communication with a second non-inverting input of the combiner 1320. A first input of the switch 1355 is connected in signal communication with an output of the disparity compensator 1350. A second input of the switch 1355 is connected in signal communication with an output of the motion compensator 1335. A third input of the switch 1355 is connected in signal communication with an output of the intra predictor 1330. An output of the mode module 1360 is connected in signal communication with the switch 1355 for controlling which input is selected by the switch 1355. An output of the deblocking filter 1325 is available as an output of the decoder 1300.

**[0106]** Turning to FIG. 14, an exemplary method for processing pictures for a plurality of views in preparation for encoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1400.

**[0107]** The method 1400 includes a start block 1405 that passes control to a function block 1410. The function block 1410 arranges every N views, among a total of M views, at a particular time instance as a super-picture in tile format, and passes control to a function block 1415. The function block 1415 sets a syntax element num_coded_views_minus1, and passes control to a function block 1420. The function block 1420 sets a syntax element view_id[i] for all (num_coded_views_minus1 + 1) views, and passes control to a function block 1425. The function block 1425 sets the inter-view reference dependency information for anchor pictures, and passes control to a function block 1430. The function block 1430 sets the inter-view reference dependency information for non-anchor pictures, and passes control to a function block 1435. The function block 1435 sets a syntax element pseudo_view_present_flag, and passes control to a decision block 1440. The decision block 1440 determines whether or not the current value of the syntax element pseudo _view_present_flag is equal to true. If so, then control is passed to a function block 1445. Otherwise, control is passed to an end block 1499.

**[0108]** The function block 1445 sets the following syntax elements, and passes control to a function block 1450: tiling_mode; org_pic_width_in_mbs_minus1; and org_pic_height_in_mbs_minus1. The function block 1450 calls a syntax element pseudo_view_info(view_id) for each coded view, and passes control to the end block 1499.

**[0109]** Turning to FIG. 15, an exemplary method for encoding pictures for a plurality of views using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1500.

**[0110]** The method 1500 includes a start block 1502 that has an input parameter pseudo_view_id and passes control to a function block 1504. The function block 1504 sets a syntax element num_sub_views_minus1, and passes control to a function block 1506. The function block 1506 sets a variable i equal to zero, and passes control to a decision block 1508. The decision block 1508 determines whether or not the variable i is less than the number of sub_views. If so, then control is passed to a function block 1510. Otherwise, control is passed to a function block 1520.

**[0111]** The function block 1510 sets a syntax element sub_viewid[i], and passes control to a function block 1512. The function block 1512 sets a syntax element num_parts_minus1[sub_view_id[i]], and passes control to a function block 1514. The function block 1514 sets a variable j equal to zero, and passes control to a decision block 1516. The decision block 1516 determines whether or not the variable j is less than the syntax element num_parts_minus1[sub_view_id[i]]. If so, then control is passed to a function block 1518. Otherwise, control is passed to a decision block 1522.

**[0112]** The function block 1518 sets the following syntax elements, increments the variable j,

and returns control to the decision block 1516: loc_left_offset[sub_view_id[i]][j]; loc_top_offset[sub_view_id[i]][j]; frame_crop_left_offset[sub_view_id[i]][j]; frame_crop_right_offset[sub_view_id[i]][j]; frame_crop_top_offset[sub_view_id[i]][j]; and frame_crop_bottom_offset[sub_view_id[i][j].

**[0113]** The function block 1520 encodes the current picture for the current view using multi-view video coding (MVC), and passes control to an end block 1599.

**[0114]** The decision block 1522 determines whether or not a syntax element tiling_mode is equal to zero. If so, then control is passed to a function block 1524. Otherwise, control is passed to a function block 1538.

**[0115]** The function block 1524 sets a syntax element flip_dir[sub_view_id[i]] and a syntax element upsample_view_flag[sub_view_id[i]], and passes control to a decision block 1526. The decision block 1526 determines whether or not the current value of the syntax element upsample_view_flag[sub_view_id[i]] is equal to one. If so, then control is passed to a function block 1528. Otherwise, control is passed to a decision block 1530.

**[0116]** The function block 1528 sets a syntax element upsample_filter[sub_view_id[i]], and passes control to the decision block 1530. The decision block 1530 determines whether or not a value of the syntax element upsample_filter[sub_view_id[i]] is equal to three. If so, the control is passed to a function block 1532. Otherwise, control is passed to a function block 1536.

**[0117]** The function block 1532 sets the following syntax elements, and passes control to a function block 1534: vert_dim[sub_view_id[i]]; hor_dim[sub_view_id[i]]; and quantizer[sub_view_id[i]]. The function block 1534 sets the filter coefficients for each YUV component, and passes control to the function block 1536.

**[0118]** The function block 1536 increments the variable i, and returns control to the decision block 1508.

**[0119]** The function block 1538 sets a syntax element pixel_dist_x[sub_view_id[i]] and the syntax element flip_dist_y[sub_view_id[i]], and passes control to a function block 1540. The function block 1540 sets the variable j equal to zero, and passes control to a decision block 1542. The decision block 1542 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[sub_view_id[i]]. If so, then control is passed to a function block 1544. Otherwise, control is passed to the function block 1536.

**[0120]** The function block 1544 sets a syntax element num_pixel_tiling_filter_coeffs_minus1[sub_view_id[i]], and passes control to a function block 1546. The function block 1546 sets the coefficients for all the pixel tiling filters, and passes control to the function block 1536.

**[0121]** Turning to FIG. 16, an exemplary method for processing pictures for a plurality of views

in preparation for decoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1600.

**[0122]** The method 1600 includes a start block 1605 that passes control to a function block 1615. The function block 1615 parses a syntax element num_coded_views_minus1, and passes control to a function block 1620. The function block 1620 parses a syntax element view_id[i] for all (num_coded_views_minus1 + 1) views, and passes control to a function block 1625. The function block 1625 parses the inter-view reference dependency information for anchor pictures, and passes control to a function block 1630. The function block 1630 parses the inter-view reference dependency information for non-anchor pictures, and passes control to a function block 1635. The function block 1635 parses a syntax element pseudo _view_present_flag, and passes control to a decision block 1640. The decision block 1640 determines whether or not the current value of the syntax element pseudo _view_present_flag is equal to true. If so, then control is passed to a function block 1645. Otherwise, control is passed to an end block 1699.

**[0123]** The function block 1645 parses the following syntax elements, and passes control to a function block 1650: tiling_mode; org_pic_width_in_mbs_minus1; and org_pic_height_in_mbs_minus1. The function block 1650 calls a syntax element pseudo_view_info(view_id) for each coded view, and passes control to the end block 1699.

**[0124]** Turning to FIG. 17, an exemplary method for decoding pictures for a plurality of views using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1700.

**[0125]** The method 1700 includes a start block 1702 that starts with input parameter pseudo_view_id and passes control to a function block 1704. The function block 1704 parses a syntax element num_sub_views_minus1, and passes control to a function block 1706. The function block 1706 sets a variable i equal to zero, and passes control to a decision block 1708. The decision block 1708 determines whether or not the variable i is less than the number of sub_views. If so, then control is passed to a function block 1710. Otherwise, control is passed to a function block 1720.

**[0126]** The function block 1710 parses a syntax element sub_view_id[i], and passes control to a function block 1712. The function block 1712 parses a syntax element num_parts_minus1[sub_view_id[i]], and passes control to a function block 1714. The function block 1714 sets a variable j equal to zero, and passes control to a decision block 1716. The decision block 1716 determines whether or not the variable j is less than the syntax element num_parts_minus1[sub_view_id[i]]. If so, then control is passed to a function block 1718. Otherwise, control is passed to a decision block 1722.

**[0127]** The function block 1718 sets the following syntax elements, increments the variable j, and returns control to the decision block 1716: loc_left_offset[sub_view_id[i]][j]; loc_top_offset[sub_view_id[i]][j]; frame_crop_left_offset[sub_view_id[i]][j];

frame_crop_right_offset[sub_view_id[i]][j]; frame_crop_top_offset[sub_view_id[i]][j]; and frame_crop_bottom_offset[sub_view_id[i][j].

**[0128]** The function block 1720 decodes the current picture for the current view using multi-view video coding (MVC), and passes control to a function block 1721. The function block 1721 separates each view from the picture using the high level syntax, and passes control to an end block 1799.

**[0129]** The separation of each view from the decoded picture is done using the high level syntax indicated in the bitstream. This high level syntax may indicate the exact location and possible orientation of the views (and possible corresponding depth) present in the picture.

**[0130]** The decision block 1722 determines whether or not a syntax element tiling_mode is equal to zero. If so, then control is passed to a function block 1724. Otherwise, control is passed to a function block 1738.

**[0131]** The function block 1724 parses a syntax element flip_dir[sub_view_id[i]] and a syntax element upsample_view_flag[sub_view_id[i]], and passes control to a decision block 1726. The decision block 1726 determines whether or not the current value of the syntax element upsample_view_flag[sub_view_id[i]] is equal to one. If so, then control is passed to a function block 1728. Otherwise, control is passed to a decision block 1730.

**[0132]** The function block 1728 parses a syntax element upsample_filter[sub_view_id[i]], and passes control to the decision block 1730. The decision block 1730 determines whether or not a value of the syntax element upsample_filter[sub_view_id[i]] is equal to three. If so, the control is passed to a function block 1732. Otherwise, control is passed to a function block 1736.

**[0133]** The function block 1732 parses the following syntax elements, and passes control to a function block 1734: vert_dim[sub_view_id[i]]; hor_dim[sub_view_id[i]]; and quantizer[sub_view_id[i]]. The function block 1734 parses the filter coefficients for each YUV component, and passes control to the function block 1736.

**[0134]** The function block 1736 increments the variable i, and returns control to the decision block 1708.

**[0135]** The function block 1738 parses a syntax element pixel_dist_x[sub_view_id[i]] and the syntax element flip_dist_y[sub_view_id[i]], and passes control to a function block 1740. The function block 1740 sets the variable j equal to zero, and passes control to a decision block 1742. The decision block 1742 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[sub_view_id[i]]. If so, then control is passed to a function block 1744. Otherwise, control is passed to the function block 1736.

**[0136]** The function block 1744 parses a syntax element num_pixel_tiling_filter_coeffs_minus1[sub_view_id[i]], and passes control to a function block

1746. The function block 1776 parses the coefficients for all the pixel tiling filters, and passes control to the function block 1736.

**[0137]** Turning to FIG. 18, an exemplary method for processing pictures for a plurality of views and depths in preparation for encoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1800.

**[0138]** The method 1800 includes a start block 1805 that passes control to a function block 1810. The function block 1810 arranges every N views and depth maps, among a total of M views and depth maps, at a particular time instance as a super-picture in tile format, and passes control to a function block 1815. The function block 1815 sets a syntax element num_coded_views_minus1, and passes control to a function block 1820. The function block 1820 sets a syntax element view_id[i] for all (num_coded_views_minus1 + 1) depths corresponding to view_id[i], and passes control to a function block 1825. The function block 1825 sets the inter-view reference dependency information for anchor depth pictures, and passes control to a function block 1830. The function block 1830 sets the inter-view reference dependency information for non-anchor depth pictures, and passes control to a function block 1835. The function block 1835 sets a syntax element pseudo _view_present_flag, and passes control to a decision block 1840. The decision block 1840 determines whether or not the current value of the syntax element pseudo_view_present_flag is equal to true. If so, then control is passed to a function block 1845. Otherwise, control is passed to an end block 1899.

**[0139]** The function block 1845 sets the following syntax elements, and passes control to a function block 1850: tiling_mode; org_pic_width_in_mbs_minus1; and org_pic_height_in_mbs_minus1. The function block 1850 calls a syntax element pseudo_view_info(view_id) for each coded view, and passes control to the end block 1899.

**[0140]** Turning to FIG. 19, an exemplary method for encoding pictures for a plurality of views and depths using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 1900.

**[0141]** The method 1900 includes a start block 1902 that passes control to a function block 1904. The function block 1904 sets a syntax element num_sub_views_minus1, and passes control to a function block 1906. The function block 1906 sets a variable i equal to zero, and passes control to a decision block 1908. The decision block 1908 determines whether or not the variable i is less than the number of sub_views. If so, then control is passed to a function block 1910. Otherwise, control is passed to a function block 1920.

**[0142]** The function block 1910 sets a syntax element sub_view_id[i], and passes control to a function block 1912. The function block 1912 sets a syntax element num_parts_minus1[sub_view_id[i]], and passes control to a function block 1914. The function block 1914 sets a variable j equal to zero, and passes control to a decision block 1916. The decision block 1916 determines whether or not the variable j is less than the syntax element num_parts_minus1[sub_view_id[i]]. If so, then control is passed to a function block 1918.

Otherwise, control is passed to a decision block 1922.

**[0143]** The function block 1918 sets the following syntax elements, increments the variable j, and returns control to the decision block 1916: loc_left_offset[sub_view_id[i]][j]; loc_top_offset[sub_view_id[i]][j]; frame_crop_left_offset[sub_view_id[i]][j]; frame_crop_right_offset[sub_view_id[i]][j]; frame_crop_top_offset[sub_view_id[i]][j]; and frame_crop_bottom_offset[sub_view_id[i][j].

**[0144]** The function block 1920 encodes the current depth for the current view using multi-view video coding (MVC), and passes control to an end block 1999. The depth signal may be encoded similar to the way its corresponding video signal is encoded. For example, the depth signal for a view may be included on a tile that includes only other depth signals, or only video signals, or both depth and video signals. The tile (pseudo-view) is then treated as a single view for MVC, and there are also presumably other tiles that are treated as other views for MVC.

**[0145]** The decision block 1922 determines whether or not a syntax element tiling_mode is equal to zero. If so, then control is passed to a function block 1924. Otherwise, control is passed to a function block 1938.

**[0146]** The function block 1924 sets a syntax element flip_dir[sub_view_id[i]] and a syntax element upsample_view_flag[sub_view_id[i]], and passes control to a decision block 1926. The decision block 1926 determines whether or not the current value of the syntax element upsample_view_flag[sub_view_id[i]] is equal to one. If so, then control is passed to a function block 1928. Otherwise, control is passed to a decision block 1930.

**[0147]** The function block 1928 sets a syntax element upsample_filter[sub_view_id[i]], and passes control to the decision block 1930. The decision block 1930 determines whether or not a value of the syntax element upsample_filter[sub_view_id[i]] is equal to three. If so, the control is passed to a function block 1932. Otherwise, control is passed to a function block 1936.

**[0148]** The function block 1932 sets the following syntax elements, and passes control to a function block 1934: vert_dim[sub_view_id[i]]; hor_dim[sub_view_id[i]]; and quantizer[sub_view_id[i]]. The function block 1934 sets the filter coefficients for each YUV component, and passes control to the function block 1936.

**[0149]** The function block 1936 increments the variable i, and returns control to the decision block 1908.

**[0150]** The function block 1938 sets a syntax element pixel_dist_x[sub_view_id[i]] and the syntax element flip_dist_y[sub_view_id[i]], and passes control to a function block 1940. The function block 1940 sets the variable j equal to zero, and passes control to a decision block 1942. The decision block 1942 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[sub_view_id[i]]. If so, then control is passed to a function block 1944. Otherwise, control is passed to the function block 1936.

**[0151]** The function block 1944 sets a syntax element num_pixel_tiling_filter_coeffs_minus1[sub_view_id[i]], and passes control to a function block 1946. The function block 1946 sets the coefficients for all the pixel tiling filters, and passes control to the function block 1936.

**[0152]** Turning to FIG. 20, an exemplary method for processing pictures for a plurality of views and depths in preparation for decoding the pictures using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 2000.

**[0153]** The method 2000 includes a start block 2005 that passes control to a function block 2015. The function block 2015 parses a syntax element num_coded_views_minus1, and passes control to a function block 2020. The function block 2020 parses a syntax element view_id[i] for all (num_coded_views_minus1 + 1) depths corresponding to view_id[i], and passes control to a function block 2025. The function block 2025 parses the inter-view reference dependency information for anchor depth pictures, and passes control to a function block 2030. The function block 2030 parses the inter-view reference dependency information for non-anchor depth pictures, and passes control to a function block 2035. The function block 2035 parses a syntax element pseudo _view_present_flag, and passes control to a decision block 2040. The decision block 2040 determines whether or not the current value of the syntax element pseudo _view_present_flag is equal to true. If so, then control is passed to a function block 2045. Otherwise, control is passed to an end block 2099.

**[0154]** The function block 2045 parses the following syntax elements, and passes control to a function block 2050: tiling_mode; org_pic_width_in_mbs_minus1; and org_pic_height_in_mbs_minus1. The function block 2050 calls a syntax element pseudo_view_info(view_id) for each coded view, and passes control to the end block 2099.

**[0155]** Turning to FIG. 21, an exemplary method for decoding pictures for a plurality of views and depths using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard is indicated generally by the reference numeral 2100.

**[0156]** The method 2100 includes a start block 2102 that starts with input parameter pseudo_view_id, and passes control to a function block 2104. The function block 2104 parses a syntax element num_sub_views_minus1, and passes control to a function block 2106. The function block 2106 sets a variable i equal to zero, and passes control to a decision block 2108. The decision block 2108 determines whether or not the variable i is less than the number of sub_views. If so, then control is passed to a function block 2110. Otherwise, control is passed to a function block 2120.

**[0157]** The function block 2110 parses a syntax element sub_view_id[i], and passes control to a function block 2112. The function block 2112 parses a syntax element num_parts_minus1[sub_view_id[i]], and passes control to a function block 2114. The function block 2114 sets a variable j equal to zero, and passes control to a decision block 2116. The

decision block 2116 determines whether or not the variable j is less than the syntax element num_parts_minus1[sub_view_id[i]]. If so, then control is passed to a function block 2118. Otherwise, control is passed to a decision block 2122.

**[0158]** The function block 2118 sets the following syntax elements, increments the variable j, and returns control to the decision block 2116: loc_left_offset[sub_view_id[i]][j]; loc_top_offset[sub_view_id[i]][j]; frame_crop_left_offset[sub_view_id[i]][j]; frame_cropright_offset[sub_view_id[i]][j]; frame_crop_top_offset[sub_view_id[i]][j]; and frame_crop_bottom_offset[sub_view_id[i][j].

**[0159]** The function block 2120 decodes the current picture using multi-view video coding (MVC), and passes control to a function block 2121. The function block 2121 separates each view from the picture using the high level syntax, and passes control to an end block 2199. The separation of each view using high level syntax is as previously described.

**[0160]** The decision block 2122 determines whether or not a syntax element tiling_mode is equal to zero. If so, then control is passed to a function block 2124. Otherwise, control is passed to a function block 2138.

**[0161]** The function block 2124 parses a syntax element flip_dir[sub_view_id[i]] and a syntax element upsample_view_flag[sub_view_id[i]], and passes control to a decision block 2126. The decision block 2126 determines whether or not the current value of the syntax element upsample_view_flag[sub_view_id[i]] is equal to one. If so, then control is passed to a function block 2128. Otherwise, control is passed to a decision block 2130.

**[0162]** The function block 2128 parses a syntax element upsample_filter[sub_view_id[i]], and passes control to the decision block 2130. The decision block 2130 determines whether or not a value of the syntax element upsample_filter[sub_view_id[i]] is equal to three. If so, the control is passed to a function block 2132. Otherwise, control is passed to a function block 2136.

**[0163]** The function block 2132 parses the following syntax elements, and passes control to a function block 2134: vert_dim[sub_view_id[i]]; hor _dim[sub_view_id[i]]; and quantizer[sub_view_id[i]]. The function block 2134 parses the filter coefficients for each YUV component, and passes control to the function block 2136.

**[0164]** The function block 2136 increments the variable i, and returns control to the decision block 2108.

**[0165]** The function block 2138 parses a syntax element pixel_dist_x[sub_view_id[i]] and the syntax element flip_dist_y[sub_view_id[i]], and passes control to a function block 2140. The function block 2140 sets the variable j equal to zero, and passes control to a decision block 2142. The decision block 2142 determines whether or not the current value of the variable j is less than the current value of the syntax element num_parts[sub_view_id[i]]. If so, then control is passed to a function block 2144. Otherwise, control is passed to the function block 2136.

**[0166]** The function block 2144 parses a syntax element num_pixel_tiling_filter_coeffs_minus1[sub_view _id[i]], and passes control to a function block 2146. The function block 2146 parses the coefficients for all the pixel tiling filters, and passes control to the function block 2136.

**[0167]** Turning to FIG. 22, tiling examples at the pixel level are indicated generally by the reference numeral 2200. FIG. 22 is described further below.

## VIEW TILING USING MPEG-4 AVC OR MVC

**[0168]** An application of multi-view video coding is free view point TV (or FTV). This application requires that the user can freely move between two or more views. In order to accomplish this, the "virtual" views in between two views need to be interpolated or synthesized. There are several methods to perform view interpolation. One of the methods uses depth for view interpolation/synthesis.

**[0169]** Each view can have an associated depth signal. Thus, the depth can be considered to be another form of video signal. FIG. 9 shows an example of a depth signal 900. In order to enable applications such as FTV, the depth signal is transmitted along with the video signal. In the proposed framework of tiling, the depth signal can also be added as one of the tiles. FIG. 10 shows an example of depth signals added as tiles. The depth signals/tiles are shown on the right side of FIG. 10.

**[0170]** Once the depth is encoded as a tile of the whole frame, the high level syntax should indicate which tile is the depth signal so that the renderer can use the depth signal appropriately.

**[0171]** In the case when the input sequence (such as that shown in FIG. 1) is encoded using a MPEG-4 AVC Standard encoder (or an encoder corresponding to a different video coding standard and/or recommendation), the proposed high level syntax may be present in, for example, the Sequence Parameter Set (SPS), the Picture Parameter Set (PPS), a slice header, and/or a Supplemental Enhancement Information (SEI) message. An embodiment of the proposed method is shown in TABLE 1 where the syntax is present in a Supplemental Enhancement Information (SEI) message.

**[0172]** In the case when the input sequences of the pseudo views (such as that shown in FIG. 1) is encoded using the multi-view video coding (MVC) extension of the MPEG-4 AVC Standard encoder (or an encoder corresponding to multi-view video coding standard with respect to a different video coding standard and/or recommendation), the proposed high level syntax may be present in the SPS, the PPS, slice header, an SEI message, or a specified profile. An embodiment of the proposed method is shown in TABLE 1. TABLE 1 shows syntax elements present in the Sequence Parameter Set (SPS) structure, including syntax elements proposed

in accordance with an embodiment of the present principles.

TABLE 1

| seq_parameter_set_mvc_extension() { | C | Descriptor |
|---|---|---|
| num_views_minus_ 1 | | ue(v) |
| for(i = 0; i <= num_views_minus_1; i++) | | |
| view_id[i] | | ue(v) |
| for(i = 0; i <= num_views_minus_1; i++) { | | |
| num_anchor_refs_l0[i] | | ue(v) |
| for(j = 0; j < num_anchor_refs_10[i]; j++ ) | | |
| anchor_ref_l0[i][j] | | ue(v) |
| num_anchor_refs_l1 [i] | | ue(v) |
| for(j = 0; j < num_anchor_refs_l1 [i]; j++ ) | | |
| anchor_ref_l1[i][j] | | ue(v) |
| } | | |
| for(i = 0; i <= num_views_minus_1; i++) { | | |
| num_non_anchor_refs_10[i] | | ue(v) |
| for(j = 0; j < num_non_anchor_refs_l0[i]; j++ ) | | |
| non_anchor _re f_l0[i][j] | | ue(v) |
| num_non_anchor_refs_l1[i] | | ue(v) |
| for(j = 0; j < num_non_anchor_refs_l1[i]; j++ ) | | |
| non_anchor_ref_l1[i][j] | | ue(v) |
| } | | |
| pseudo_view_present_ flag | | u(1) |
| if (pseudo _view_present_ flag) { | | |
| tiling_mode | | |
| org_pic_width_in_mbs_minus1 | | |
| org_pic _height_in_mbs_minus1 | | |
| for(i = 0; i < num_views_minus_1; i++) | | |
| pseudo_view_info(i); | | |
| } | | |
| } | | |

**[0173]** TABLE 2 shows syntax elements for the pseudo_view_info syntax element of TABLE 1, in accordance with an embodiment of the present principles.

TABLE 2

| pseudo_view_info (pseudo_view_id) { | C | Descriptor |
|---|---|---|
| num_sub_views_minus_1[pseudo_view_id] | 5 | ue(v) |
| if (num_sub_views_minus_1 != 0) { | | |
| for (i = 0; i < num_sub_views_minus_1[pseudo_view_id]; i++) { | | |
| sub_view_id[i] | 5 | ue(v) |
| num_parts_minus1[sub_view_id[i]] | 5 | ue(v) |
| for(j = 0; j <= num_parts_minus1[sub_view_id[i]]; j++) { | | |
| loc_left_offset[sub_view_id[i]] [j] | 5 | ue(v) |
| loc_top_offset[sub_view_id[i ]] [j ] | 5 | ue(v) |
| frame_crop_left_offset[sub_view_id[ i]][ j ] | 5 | ue(v) |
| frame_crop_right_offset[sub_view_id[i]][j] | 5 | ue(v) |
| frame_crop_top_offset[sub_view_id[i]][j] | 5 | ue(v) |
| frame_crop_bottom_offset[sub_view_id[i]][j] | 5 | ue(v) |
| } | | |
| if (tiling_mode == 0) { | | |
| flip_dir[sub_view_id[i ][j ] | 5 | u(2) |
| upsample_view_flag[sub_view_id[ i]] | 5 | u(1) |
| if(upsample_view_flag[sub_view_id[i]]) | | |
| upsample_filter[sub _view_id[i ]] | 5 | u(2) |
| if(upsample_fiter[sub_view_id[i]] == 3) { | | |
| vert_dim[sub_view_id[i]] | 5 | ue(v) |
| hor_dim[sub_view_id[i]] | 5 | ue(v) |
| quantizer[sub_view_id[i]] | 5 | ue(v) |
| for (yuv= 0; yuv< 3; yuv++) { | | |
| for (y = 0; y < vert_dim[sub_view_id[i]] - 1; y ++) { | | |
| for (x = 0; x < hor_dim[sub_view_id[i]] - 1; x ++) | | |
| filter_coeffs[sub_view_id[i]] [yuv][y][x] | 5 | se(v) |
| } | | |
| } | | |
| } | | |
| } // if(tiling_mode == 0) | | |
| else if (tiling_mode == 1) { | | |

| pseudo_view_info (pseudo_view_id) { | C | Descriptor |
|---|---|---|
| pixel_dist_x[sub_view_id[i ]] | | |
| pixel_dist_y[sub_view_id[i ]] | | |
| for(j = 0; j <= num_parts[sub_view_id[i]]; j++) { | | |
| num_pixel_tiling_filter_coeffs _minus1 [sub_view _id[i ]][j] | | |
| for (coeff_idx = 0; coeff_idx <= num_pixel_ tiling_filter_coeffs_ minus1[sub_view_id[i ]][j];j++) | | |
| pixel_tiling_filter_coeffs[sub_view_id[i]] [j] | | |
| } // for (j = 0; j <= num_parts[sub_view_id[i]]; j++) | | |
| } // else if (tiling_mode == 1) | | |
| } // for (i = 0; i < num_sub_views_minus_1; i++) | | |
| } // if (num_sub_views_minus_1 != 0) | | |
| } | | |

**Semantics of the syntax elements presented in TABLE 1 and TABLE 2**

**[0174]** pseudo_view_present_flag equal to true indicates that some view is a super view of multiple sub-views.

**[0175]** tiling_mode equal to 0 indicates that the sub-views are tiled at the picture level. A value of 1 indicates that the tiling is done at the pixel level.

**[0176]** The new SEI message could use a value for the SEI payload type that has not been used in the MPEG-4 AVC Standard or an extension of the MPEG-4 AVC Standard. The new SEI message includes several syntax elements with the following semantics.

**[0177]** num_coded_views_minus1 plus 1 indicates the number of coded views supported by the bitstream. The value of num_coded_views_minus1 is in the scope of 0 to 1023, inclusive.

**[0178]** org_pic_width_in_mbs_minus1 plus 1 specifies the width of a picture in each view in units of macroblocks.

**[0179]** The variable for the picture width in units of macroblocks is derived as follows:

$$PicWidthInMbs = org\_pic\_width\_in\_mbs\_minus1 + 1$$

**[0180]** The variable for picture width for the luma component is derived as follows:

$$PicWidthInSamplesL = PicWidthInMbs * 16$$

**[0181]** The variable for picture width for the chroma components is derived as follows:

PicWidthInSamplesC = PicWidthInMbs * MbWidthC

**[0182]** org_pic_height_in_mbs_minus1 plus 1 specifies the height of a picture in each view in units of macroblocks.

**[0183]** The variable for the picture height in units of macroblocks is derived as follows:

PicHeightInMbs = org_pic_height_in_mbs_minus1 + 1

**[0184]** The variable for picture height for the luma component is derived as follows:

PicHeightInSamplesL = PicHeightInMbs * 16

**[0185]** The variable for picture height for the chroma components is derived as follows:

PicHeightInSamplesC = PicHeightInMbs * MbHeightC

**[0186]** num_sub_views_minus1 plus 1 indicates the number of coded sub-views included in the current view. The value of num_coded_views_minus1 is in the scope of 0 to 1023, inclusive.

**[0187]** sub_view_id[i] specifies the sub_view_id of the sub-view with decoding order indicated by i.

**[0188]** num_parts[sub_view_id[i]] specifies the number of parts that the picture of sub_view_id[i] is split up into.

**[0189]** loc_left_offset[sub_view_id[i]][j] and loc_top_offset[sub_view_id[i]][j] specify the locations in left and top pixels offsets, respectively, where the current part j is located in the final reconstructed picture of the view with sub_view_id equal to sub_view_id[i].

**[0190]** view_id[i] specifies the view_id of the view with coding order indicate by i.

**[0191]** frame_crop_left_offset[view_id[i]][j], frame_crop_right_offset[view_id[i]][j], frame_crop_top_offset[view_id[i]][j], and frame_crop_bottom_offset[view_id[i]][j] specify the samples of the pictures in the coded video sequence that are part of num_part j and view_id i, in terms of a rectangular region specified in frame coordinates for output.

**[0192]** The variables CropUnitX and CropUnitY are derived as follows:

- If chroma_format_idc is equal to 0, CropUnitX and CropUnitY are derived as follows:

CropUnitX = 1

CropUnitY = 2 - frame_mbs_only_flag

- Otherwise (chroma_format_idc is equal to 1, 2, or 3), CropUnitX and CropUnitY are derived as follows:

CropUnitX = SubWidthC
CropUnitY = SubHeightC * (2 – frame_mbs_only_flag)

**[0193]** The frame cropping rectangle includes luma samples with horizontal frame coordinates from the following:

CropUnitX * frame_crop_left_offset to PicWidthInSamplesL - (CropUnitX * frame_crop_right_offset + 1) and vertical frame coordinates from CropUnitY * frame_crop_top_offset to (16 * FrameHeightInMbs) - (CropUnitY * frame_crop_bottom_offset + 1), inclusive. The value of frame_crop_left_offset shall be in the range of 0 to (PicWidthInSamplesL / CropUnitX) - (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to (16 * FrameHeightInMbs / CropUnitY) - (frame_crop_bottom_offset + 1), inclusive.

**[0194]** When chroma_format_idc is not equal to 0, the corresponding specified samples of the two chroma arrays are the samples having frame coordinates (x / SubWidthC, y / SubHeightC), where (x, y) are the frame coordinates of the specified luma samples.
For decoded fields, the specified samples of the decoded field are the samples that fall within the rectangle specified in frame coordinates.

**[0195]** num_parts[view_id[i]] specifies the number of parts that the picture of view_id[i] is split up into.

**[0196]** depth_flag[view_id[i]] specifies whether or not the current part is a depth signal. If depth_flag is equal to 0, then the current part is not a depth signal. If depth_flag is equal to 1, then the current part is a depth signal associated with the view identified by view_id[i].

**[0197]** flip_dir[sub_view_id[i]][j] specifies the flipping direction for the current part. flip_dir equal to 0 indicates no flipping, flip_dir equal to 1 indicates flipping in a horizontal direction, flip_dir equal to 2 indicates flipping in a vertical direction, and flip_dir equal to 3 indicates flipping in horizontal and vertical directions.

**[0198]** flip_dir[view_id[i]][j] specifies the flipping direction for the current part. flip_dir equal to 0 indicates no flipping, flip_dir equal to 1 indicates flipping in a horizontal direction, flip_dir equal to 2 indicates flipping in vertical direction, and flip_dir equal to 3 indicates flipping in horizontal and vertical directions.

**[0199]** loc_left_offset[view_id[i]][j], loc_top_offset[view_id[i]][j] specifies the location in pixels offsets, where the current part j is located in the final reconstructed picture of the view with view_id equals to view_id[i]

**[0200]** upsample_view_flag[view_id[i]] indicates whether the picture belonging to the view specified by view_id[i] needs to be upsampled. upsample_view_flag[view_id[i]] equal to 0 specifies that the picture with view_id equal to view_id[i] will not be upsampled. upsample_view_flag[view_id[i]] equal to 1 specifies that the picture with view_id equal to view_id[i] will be upsampled.

**[0201]** upsample_filter[view_id[i]] indicates the type of filter that is to be used for upsampling. upsample_filter[view_id[i]] equals to 0 indicates that the 6-tap AVC filter should be used, upsample_filter[view_id[i]] equals to 1 indicates that the 4-tap SVC filter should be used, upsample_filter[view_id[i]] 2 indicates that the bilinear filter should be used, upsample_filter[view_id[i]] equals to 3 indicates that custom filter coefficients are transmitted. When upsample_fiter[view_id[i]] is not present it is set to 0. In this embodiment, we use 2D customized filter. It can be easily extended to 1D filter, and some other nonlinear filter.

**[0202]** vert_dim[view_id[i]] specifies the vertical dimension of the custom 2D filter.

**[0203]** hor_dim[view_id[i]] specifies the horizontal dimension of the custom 2D filter.

**[0204]** quantizer[view_id[i]] specifies the quantization factor for each filter coefficient.

**[0205]** filter_coeffs[view_id[i]] [yuv][y][x] specifies the quantized filter coefficients. yuv signals the component for which the filter coefficients apply. yuv equal to 0 specifies the Y component, yuv equal to 1 specifies the U component, and yuv equal to 2 specifies the V component.

**[0206]** pixel_dist_x[sub_view_id[i]] and pixel_dist_y[sub_view_id[i]] respectively specify the distance in the horizontal direction and the vertical direction in the final reconstructed pseudo view between neighboring pixels in the view with sub_view_id equal to sub_view_id[i].

**[0207]** num_pixel_tiling_filter_coeffs_minus1[sub_view_id[i][j] plus one indicates the number of the filter coefficients when the tiling mode is set equal to 1.

**[0208]** pixel_tiling_filter_coeffs[sub_view_id[i][j] signals the filter coefficients that are required to represent a filter that may be used to filter the tiled picture.

**Tiling examples at pixel level**

**[0209]** Turning to FIG. 22, two examples showing the composing of a pseudo view by tiling pixels from four views are respectively indicated by the reference numerals 2210 and 2220, respectively. The four views are collectively indicated by the reference numeral 2250. The

syntax values for the first example in FIG. 22 are provided in TABLE 3 below.

TABLE 3

| pseudo_view_info (pseudo_view_id) { | Value |
|---|---|
| num_sub_views_minus_1[pseudo_view_id] | 3 |
| sub_view_id[0] | 0 |
| num_parts_minus1[0] | 0 |
| loc_left_offset[0][0] | 0 |
| loc_top_offset[0][0] | 0 |
| pixel_dist_x[0][0] | 0 |
| pixel_dist_y[0][0] | 0 |
| sub_view_id[1] | 0 |
| num_parts_minus1[1] | 0 |
| loc_left_offset[1][0] | 1 |
| loc_top_offset[1][0] | 0 |
| pixel_dist_x[1][0] | 0 |
| pixel_dist_y[1][0] | 0 |
| sub_view_id[2] | 0 |
| num_parts_minus1[2] | 0 |
| loc_left _offset[2][0] | 0 |
| loc_top_offset[2][0] | 1 |
| pixel_dist_x[2][0] | 0 |
| pixel_dist_y[2][0] | 0 |
| sub_view_id[3] | 0 |
| num_parts_minus1[3] | 0 |
| loc_left_offset[3][0] | 1 |
| loc_top_offset[3][0] | 1 |
| pixel_dist_x[3][0] | 0 |
| pixel_dist_y[3][0] | 0 |

**[0210]** The syntax values for the second example in FIG. 22 are all the same except the following two syntax elements: loc_left_offset[3][0] equal to 5 and loc_top_offset[3][0] equal to 3.

**[0211]** The offset indicates that the pixels corresponding to a view should begin at a certain offset location. This is shown in FIG. 22 (2220). This may be done, for example, when two views produce images in which common objects appear shifted from one view to the other. For example, if first and second cameras (representing first and second views) take pictures of an

object, the object may appear to be shifted five pixels to the right in the second view as compared to the first view. This means that pixel(i-5, j) in the first view corresponds to pixel(i, j) in the second view. If the pixels of the two views are simply tiled pixel-by-pixel, then there may not be much correlation between neighboring pixels in the tile, and spatial coding gains may be small. Conversely, by shifting the tiling so that pixel(i-5, j) from view one is placed next to pixel(i, j) from view two, spatial correlation may be increased and spatial coding gain may also be increased. This follows because, for example, the corresponding pixels for the object in the first and second views are being tiled next to each other.

**[0212]** Thus, the presence of loc_left_offset and loc_top_offset may benefit the coding efficiency. The offset information may be obtained by external means. For example, the position information of the cameras or the global disparity vectors between the views may be used to determine such offset information.

**[0213]** As a result of offsetting, some pixels in the pseudo view are not assigned pixel values from any view. Continuing the example above, when tiling pixel(i-5, j) from view one alongside pixel(i, j) from view two, for values of i=0 ... 4 there is no pixel(i-5, j) from view one to tile, so those pixels are empty in the tile. For those pixels in the pseudo-view (tile) that are not assigned pixel values from any view, at least one implementation uses an interpolation procedure similar to the sub-pixel interpolation procedure in motion compensation in AVC. That is, the empty tile pixels may be interpolated from neighboring pixels. Such interpolation may result in greater spatial correlation in the tile and greater coding gain for the tile.

**[0214]** In video coding, we can choose a different coding type for each picture, such as I, P, and B pictures. For multi-view video coding, in addition, we define anchor and non-anchor pictures. In an embodiment, we propose that the decision of grouping can be made based on picture type. This information of grouping is signaled in high level syntax.

**[0215]** Turning to FIG. 11, an example of 5 views tiled on a single frame is indicated generally by the reference numeral 1100. In particular, the ballroom sequence is shown with 5 views tiled on a single frame. Additionally, it can be seen that the fifth view is split into two parts so that it can be arranged on a rectangular frame. Here, each view is of QVGA size so the total frame dimension is 640x600. Since 600 is not a multiple of 16 it should be extended to 608.

**[0216]** For this example, the possible SEI message could be as shown in TABLE 4.

TABLE 4

| multiview_display_info(payloadSize ) { | Value |
|---|---|
| num_coded_views_minus1 | 5 |
| org_pic_width_in_mbs_minus1 | 40 |
| org_pic_height_in_mbs_minus1 | 30 |
| | |
| view_id[ 0] | 0 |

| multiview_display_info(payloadSize ) { | Value |
|---|---|
| num_parts[view_id[ 0]] | 1 |
| | |
| depth_flag[view_id[ 0]][0] | 0 |
| flip_dir[view_id[ 0]][0] | 0 |
| loc_left_offset[view_id[ 0]][0] | 0 |
| loc_top_offset[view_id[ 0]][0] | 0 |
| frame_crop_left_offset[view_id[ 0]][0] | 0 |
| frame_crop_right_offset[view_id[ 0]][0] | 320 |
| frame_crop_top_offset[view_id[ 0]][0] | 0 |
| frame_crop_bottom_offset[view_id[ 0]][0] | 240 |
| | |
| upsample_view_flag[view_id[ 0]] | 1 |
| if(upsample_view_flag[view_id[0]]) { | |
| vert_dimview_id[0]] | 6 |
| hor dim[view_ id[0]] | 6 |
| quantizer[view_id[0]] | 32 |
| for (yuv= 0; yuv< 3; yuv++) { | |
| for (y = 0; y < vert_dim[view_id[i]] - 1; y ++) { for (x = 0; x < hor dim[view_ id[i]] - 1; x ++) | |
| filter_coeffs[view_id[j]] [yuv] [y] [x] | XX |
| | |
| view_id[ 1] | 1 |
| num_parts[view_id[ 1]] | 1 |
| | |
| depth_flag[view_id[ 0]][0] | 0 |
| flip_dir[view_id[ 1]][0] | 0 |
| loc_left_offset[view_id[ 1]][0] | 0 |
| loc_top_offset[view_id[ 1]][0] | 0 |
| frame_crop_left_offset[view_id[ 1]][0] | 320 |
| frame_crop_right_offset[view_id[ 1]][0] | 640 |
| frame_crop_top_offset[view_id[ 1]][0] | 0 |
| frame_crop_bottom_offset[view_id[1]][0] | 320 |
| | |
| upsample_view_flag[view_id[1]] | 1 |
| if(upsample _view_flag[view_ id[ 1]]){ | |

| multiview_display_info(payloadSize ) { | Value |
|---|---|
| vert_ dim[view_ id[1]] | 6 |
| hor_ dim[view_ id[1]] | 6 |
| quantizer[view_id[1]] | 32 |
| for (yuv= 0; yuv< 3; yuv++) { | |
| for (v = 0; y < vert dim[view_id[i]] - 1; y ++) { | |
| for (x = 0; x < hor dim[view_id[i]] - 1; x ++) | |
| filter_coeffs[view_id[i]] [yuv] [y] [x] | XX |
| | |
| | |
| | |
| ...(similarly for view 2,3) | |
| | |
| view_id[ 4] | 4 |
| num_parts[view_id[ 4]] | 2 |
| | |
| depth_flag[view_id[ 0]][0] | 0 |
| flip_dir[view_id[ 4]][0] | 0 |
| loc_left_offset[view_id[ 4]][0] | 0 |
| loc_top_offset[view_id[ 4]][0] | 0 |
| frame_crop_left_offset[view_id[ 4]][0] | 0 |
| frame_crop_right_offset[view_id[ 4]][0] | 320 |
| frame_crop_top_offset[view_id[ 4]][0] | 480 |
| frame_crop_bottom_offset[view_id[ 4]][0] | 600 |
| | |
| flip_dir[view_id[ 4]][1] | 0 |
| loc_left_offset[view_id[ 4]][1] | 0 |
| loc_top_offset[view_id[ 4]][1] | 120 |
| frame_crop_left_offset[view_id[ 4]][1] | 320 |
| frame_crop_right_offset[view_id[ 4]][1] | 640 |
| frame_crop_top_offset[view_id[ 4]][1] | 480 |
| frame_crop_bottom_offset[view_id[ 4]][1] | 600 |
| | |
| | |
| upsample_view_ flag[view _ id[ 4]] if(upsample _view_flag[view_ id[ 4]]) { | 1 |

| multiview_display_info(payloadSize ) { | Value |
|---|---|
| vert dim[view_ id[4]] | 6 |
| hor dim[view_id[4]] | 6 |
| quantizer[view_id[4]] | 32 |
| for (yuv= 0; yuv< 3; yuv++) { | |
| for (v = 0; y < vert dim[view_id[i]] - 1; y ++) { | |
| for (x = 0; x < hor dim[view_ id[i]] - 1; x ++) | |
| filter_coeffs[view_id[i]] [yuv] [y] [x] | XX |

**[0217]** TABLE 5 shows the general syntax structure for transmitting multi-view information for the example shown in TABLE 4.

TABLE 5

| multiview_display_info(payloadSize) { | C | Descriptor |
|---|---|---|
| num_coded_views_minus1 | 5 | ue(v) |
| org_pic_width_in_mbs_minus1 | 5 | ue(v) |
| org_pic_height_in_mbs_minus1 | 5 | ue(v) |
| for(i = 0; i <= num_coded_views_minus1; i++) { | | |
| view_id[i ] | 5 | ue(v) |
| num_parts[view_id[i ]] | 5 | ue(v) |
| for( j = 0; j <= num_parts[i]; j++ ) { | | |
| depth_flag[view_id[i]][j] | | |
| flip_dir[view_id[i ]][j ] | 5 | u(2) |
| loc_left_offset[view_id[i ]] [j ] | 5 | ue(v) |
| loc_top_offset[view_id[ i]][ j ] | 5 | ue(v) |
| frame_crop_left_offset[view_id[ i]][ j ] | 5 | ue(v) |
| frame_crop_right_offset[view_id[ i]][ j ] | 5 | ue(v) |
| frame_crop_top_offset[view_id[ i]][ j ] | 5 | ue(v) |
| frame_crop_bottom_offset[view_ id[i ]] [j ] | 5 | ue(v) |
| } | | |
| upsample _view_flag[view _id[ i ]] | 5 | u(1) |
| if(upsample _view_flag[view _id[i ]]) | | |
| upsample_filter[view_id[i ]] | 5 | u(2) |
| if(upsample fiter[view id[i]] == 3) { | | |
| vert_ dim[view_ id[i]] | 5 | ue(v) |
| hor_ dim[view id[i]] | 5 | ue(v) |

| multiview_display_info(payloadSize) { | C | Descriptor |
|---|---|---|
| quantizer[view id[i]] | 5 | ue(v) |
| for (yuv= 0; yuv< 3; yuv++) { | | |
| for (y = 0; y < vert dim[view id[i]] - 1; y ++) { | | |
| for (x = 0; x < hor dim[view id[i]] - 1; x ++) | | |
| filter_coeffs[view_id[i]] [yuv][y][x] | 5 | se(v) |
| } | | |
| } | | |
| } | | |
| } | | |
| } | | |

**[0218]** Referring to FIG. 23, a video processing device 2300 is shown. The video processing device 2300 may be, for example, a set top box or other device that receives encoded video and provides, for example, decoded video for display to a user or for storage. Thus, the device 2300 may provide its output to a television, computer monitor, or a computer or other processing device.

**[0219]** The device 2300 includes a decoder 2310 that receive a data signal 2320. The data signal 2320 may include, for example, an AVC or an MVC compatible stream. The decoder 2310 decodes all or part of the received signal 2320 and provides as output a decoded video signal 2330 and tiling information 2340. The decoded video 2330 and the tiling information 2340 are provided to a selector 2350. The device 2300 also includes a user interface 2360 that receives a user input 2370. The user interface 2360 provides a picture selection signal 2380, based on the user input 2370, to the selector 2350. The picture selection signal 2380 and the user input 2370 indicate which of multiple pictures a user desires to have displayed. The selector 2350 provides the selected picture(s) as an output 2390. The selector 2350 uses the picture selection information 2380 to select which of the pictures in the decoded video 2330 to provide as the output 2390. The selector 2350 uses the tiling information 2340 to locate the selected picture(s) in the decoded video 2330.

**[0220]** In various implementations, the selector 2350 includes the user interface 2360, and in other implementations no user interface 2360 is needed because the selector 2350 receives the user input 2370 directly without a separate interface function being performed. The selector 2350 may be implemented in software or as an integrated circuit, for example. The selector 2350 may also incorporate the decoder 2310.

**[0221]** More generally, the decoders of various implementations described in this application may provide a decoded output that includes an entire tile. Additionally or alternatively, the decoders may provide a decoded output that includes only one or more selected pictures

(images or depth signals, for example) from the tile.

**[0222]** As noted above, high level syntax may be used to perform signaling in accordance with one or more embodiments of the present principles. The high level syntax may be used, for example, but is not limited to, signaling any of the following: the number of coded views present in the larger frame; the original width and height of all the views; for each coded view, the view identifier corresponding to the view; for each coded view ,the number of parts the frame of a view is split into; for each part of the view, the flipping direction (which can be, for example, no flipping, horizontal flipping only, vertical flipping only or horizontal and vertical flipping); for each part of the view, the left position in pixels or number of macroblocks where the current part belongs in the final frame for the view; for each part of the view, the top position of the part in pixels or number of macroblocks where the current part belongs in the final frame for the view; for each part of the view, the left position, in the current large decoded/encoded frame, of the cropping window in pixels or number of macroblocks; for each part of the view, the right position, in the current large decoded/encoded frame, of the cropping window in pixels or number of macroblocks; for each part of the view, the top position, in the current large decoded/encoded frame, of the cropping window in pixels or number of macroblocks; and, for each part of the view, the bottom position, in the current large decoded/encoded frame, of the cropping window in pixels or number of macroblocks; for each coded view whether the view needs to be upsampled before output (where if the upsampling needs to be performed, a high level syntax may be used to indicate the method for upsampling (including, but not limited to, AVC 6-tap filter, SVC 4-tap filter, bilinear filter or a custom ID, 2D linear or non-linear filter).

**[0223]** It is to be noted that the terms "encoder" and "decoder" connote general structures and are not limited to any particular functions or features. For example, a decoder may receive a modulated carrier that carries an encoded bitstream, and demodulate the encoded bitstream, as well as decode the bitstream.

**[0224]** Various methods have been described. Many of these methods are detailed to provide ample disclosure. It is noted, however, that variations are contemplated that may vary one or many of the specific features described for these methods. Further, many of the features that are recited are known in the art and are, accordingly, not described in great detail.

**[0225]** Further, reference has been made to the use of high level syntax for sending certain information in several implementations. It is to be understood, however, that other implementations use lower level syntax, or indeed other mechanisms altogether (such as, for example, sending information as part of encoded data) to provide the same information (or variations of that information).

**[0226]** Various implementations provide tiling and appropriate signaling to allow multiple views (pictures, more generally) to be tiled into a single picture, encoded as a single picture, and sent as a single picture. The signaling information may allow a post-processor to pull the views/pictures apart. Also, the multiple pictures that are tiled could be views, but at least one of

the pictures could be depth information. These implementations may provide one or more advantages. For example, users may want to display multiple views in a tiled manner, and these various implementations provide an efficient way to encode and transmit or store such views by tiling them prior to encoding and transmitting/storing them in a tiled manner.

**[0227]** Implementations that tile multiple views in the context of AVC and/or MVC also provide additional advantages. AVC is ostensibly only used for a single view, so no additional view is expected. However, such AVC-based implementations can provide multiple views in an AVC environment because the tiled views can be arranged so that, for example, a decoder knows that that the tiled pictures belong to different views (for example, top left picture in the pseudo-view is view 1, top right picture is view 2, etc).

**[0228]** Additionally, MVC already includes multiple views, so multiple views are not expected to be included in a single pseudo-view. Further, MVC has a limit on the number of views that can be supported, and such MVC-based implementations effectively increase the number of views that can be supported by allowing (as in the AVC-based implementations) additional views to be tiled. For example, each pseudo-view may correspond to one of the supported views of MVC, and the decoder may know that each "supported view" actually includes four views in a pre-arranged tiled order. Thus, in such an implementation, the number of possible views is four times the number of "supported views".

**[0229]** The implementations described herein may be implemented in, for example, a method or process, an apparatus, or a software program. Even if only discussed in the context of a single form of implementation (for example, discussed only as a method), the implementation of features discussed may also be implemented in other forms (for example, an apparatus or program). An apparatus may be implemented in, for example, appropriate hardware, software, and firmware. The methods may be implemented in, for example, an apparatus such as, for example, a processor, which refers to processing devices in general, including, for example, a computer, a microprocessor, an integrated circuit, or a programmable logic device. Processing devices also include communication devices, such as, for example, computers, cell phones, portable/personal digital assistants ("PDAs"), and other devices that facilitate communication of information between end-users.

**[0230]** Implementations of the various processes and features described herein may be embodied in a variety of different equipment or applications, particularly, for example, equipment or applications associated with data encoding and decoding. Examples of equipment include video coders, video decoders, video codecs, web servers, set-top boxes, laptops, personal computers, cell phones, PDAs, and other communication devices. As should be clear, the equipment may be mobile and even installed in a mobile vehicle.

**[0231]** Additionally, the methods may be implemented by instructions being performed by a processor, and such instructions may be stored on a processor-readable medium such as, for example, an integrated circuit, a software carrier or other storage device such as, for example, a hard disk, a compact diskette, a random access memory ("RAM"), or a read-only memory

("ROM"). The instructions may form an application program tangibly embodied on a processor-readable medium. As should be clear, a processor may include a processor-readable medium having, for example, instructions for carrying out a process. Such application programs may be uploaded to, and executed by, a machine comprising any suitable architecture. Preferably, the machine is implemented on a computer platform having hardware such as one or more central processing units ("CPU"), a random access memory ("RAM"), and input/output ("I/O") interfaces. The computer platform may also include an operating system and microinstruction code. The various processes and functions described herein may be either part of the microinstruction code or part of the application program, or any combination thereof, which may be executed by a CPU. In addition, various other peripheral units may be connected to the computer platform such as an additional data storage unit and a printing unit.

**[0232]** As should be evident to one of skill in the art, implementations may also produce a signal formatted to carry information that may be, for example, stored or transmitted. The information may include, for example, instructions for performing a method, or data produced by one of the described implementations. Such a signal may be formatted, for example, as an electromagnetic wave (for example, using a radio frequency portion of spectrum) or as a baseband signal. The formatting may include, for example, encoding a data stream, producing syntax, and modulating a carrier with the encoded data stream and the syntax. The information that the signal carries may be, for example, analog or digital information. The signal may be transmitted over a variety of different wired or wireless links, as is known.

**[0233]** It is to be further understood that, because some of the constituent system components and methods depicted in the accompanying drawings are preferably implemented in software, the actual connections between the system components or the process function blocks may differ depending upon the manner in which the present principles are programmed. Given the teachings herein, one of ordinary skill in the pertinent art will be able to contemplate these and similar implementations or configurations of the present principles.

**[0234]** A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made. For example, elements of different implementations may be combined, supplemented, modified, or removed to produce other implementations. Additionally, one of ordinary skill will understand that other structures and processes may be substituted for those disclosed and the resulting implementations will perform at least substantially the same function(s), in at least substantially the same way(s), to achieve at least substantially the same result(s) as the implementations disclosed. In particular, although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present principles is not limited to those precise embodiments, and that various changes and modifications may be effected therein by one of ordinary skill in the pertinent art without departing from the scope of the present principles as defined by the claims. Accordingly, these and other implementations are contemplated by this application and are within the scope of the appended claims.

# REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

**Patent documents cited in the description**


- EP1581003A1 **[0002]**

**Patentkrav**

**1.** Fremgangsmåde, som omfatter:

- modtagelse af en videostrøm;

- tilgåelse af et videobillede der inkluderer flere billeder kombineret til et enkelt billede (826), hvor de flere billeder inkluderer et første billede af et første view af en multiview-video og et andet billede af et andet view af multiview-videoen, hvor indhold af det første billede overlapper indhold af det andet billede, hvor videobilledet er en del af den modtagne videostrøm;

- tilgåelse af information der indikerer hvordan de flere billeder i det tilgåede videobillede kombineres, hvor den tilgåede information indikerer at mindst et af de flere billeder individuelt er vendt om i en eller flere af en horisontal retning eller en vertikal retning,
og hvor den tilgåede information er omfattet i mindst en af en slice-header, et sekvensparametersæt, et billedparametersæt, en netværksabstraktionslagsenhed-header, og en supplerende videreudviklingsinformationsbesked,
hvor det første billede af det første view ikke vendes om og det andet billede af det andet view vendes om, hvor det første billede af det første view og det andet billede af det andet view er anbragt side om side eller over hinanden; og

- afkodning af videobilledet for at tilvejebringe en afkodet repræsentation af de kombinerede flere billeder (824, 826),
hvor den tilgåede information (824, 826) kan anvendes til efterbehandling af den afkodede repræsentation for individuelt at vende det mindst ene af de flere billeder om.

**2.** Fremgangsmåden ifølge krav 1, hvor tilgåelsen af videobilledet, tilgåelsen af informationen, og afkodningen af videobilledet udføres på en afkoder (1650).

**3.** Fremgangsmåde, som omfatter:

- dannelse af et videobillede der inkluderer flere billeder kombineret til et enkelt billede, hvor de flere billeder inkluderer et første billede af et første

view af en multiview-video og et andet billede af et andet view af multiview-videoen, hvor indhold af det første billede overlapper indhold af det andet billede;

- dannelse af information der indikerer hvordan de flere billeder i det dannede videobillede kombineres, hvor den dannede information indikerer at mindst et af de flere billeder individuelt er vendt om i en eller flere af en horisontal retning eller en vertikal retning,
og hvor den dannede information er omfattet i mindst en af en slice-header, et sekvensparametersæt, et billedparametersæt, en netværksabstraktionslagsenhed-header, og en supplerende videreudviklingsinformationsbesked,
hvor det første billede af det første view ikke er vendt om og det andet billede af det andet view er vendt om, hvor det første billede af det første view og det andet billede af det andet view er anbragt side om side eller over hinanden;

- kodning af det dannede videobillede og den dannede information; og

- tilvejebringelse af en videostrøm der inkluderer det kodede videobillede og den kodede information.

**4.** Fremgangsmåden ifølge krav 3, hvor dannelsen af videobilledet, dannelsen af informationen, kodningen, og tilvejebringelsen af videostrømmen udføres på en koder.

**5.** Computerlæsbart medium med computerlæsbar programkode, som er computereksekverbare instruktioner til udførelse af en proces udformet derpå, hvilken computerlæsbare programkode omfatter:

- programkode til dannelse af et videobillede der inkluderer flere billeder kombineret til et enkelt billede, de flere billeder inkluderer et første billede af et første view af en multiview-video og et andet billede af et andet view af multiview-videoen, hvor indhold af det første billede overlapper indhold af det andet billede;

- programkode til dannelse af information der indikerer hvordan de flere billeder i det dannede videobillede kombineres, hvor den dannede

information indikerer at mindst et af de flere billeder individuelt er vendt om i en eller flere af en horisontal retning eller en vertikal retning,
og hvor den dannede information er omfattet i mindst en af en slice-header, et sekvensparametersæt, et billedparametersæt, en netværksabstraktionslagsenhed-header, og en supplerende videreudviklingsinformationsbesked,
hvor det første billede af det første view ikke er vendt om og det andet billede af det andet view er vendt om, hvor det første billede af det første view og det andet billede af det andet view er anbragt side om side eller over hinanden;

- programkode til kodning af det dannede videobillede og den dannede information; og

- programkode til tilvejebringelse af en videostrøm der inkluderer det kodede videobillede og den kodede information.

**6.** Det computerlæsbare medium ifølge krav 5, hvor dannelsen af videobilledet, dannelsen af informationen, kodningen, og tilvejebringelsen af videostrømmen udføres på en koder.

# DRAWINGS

100

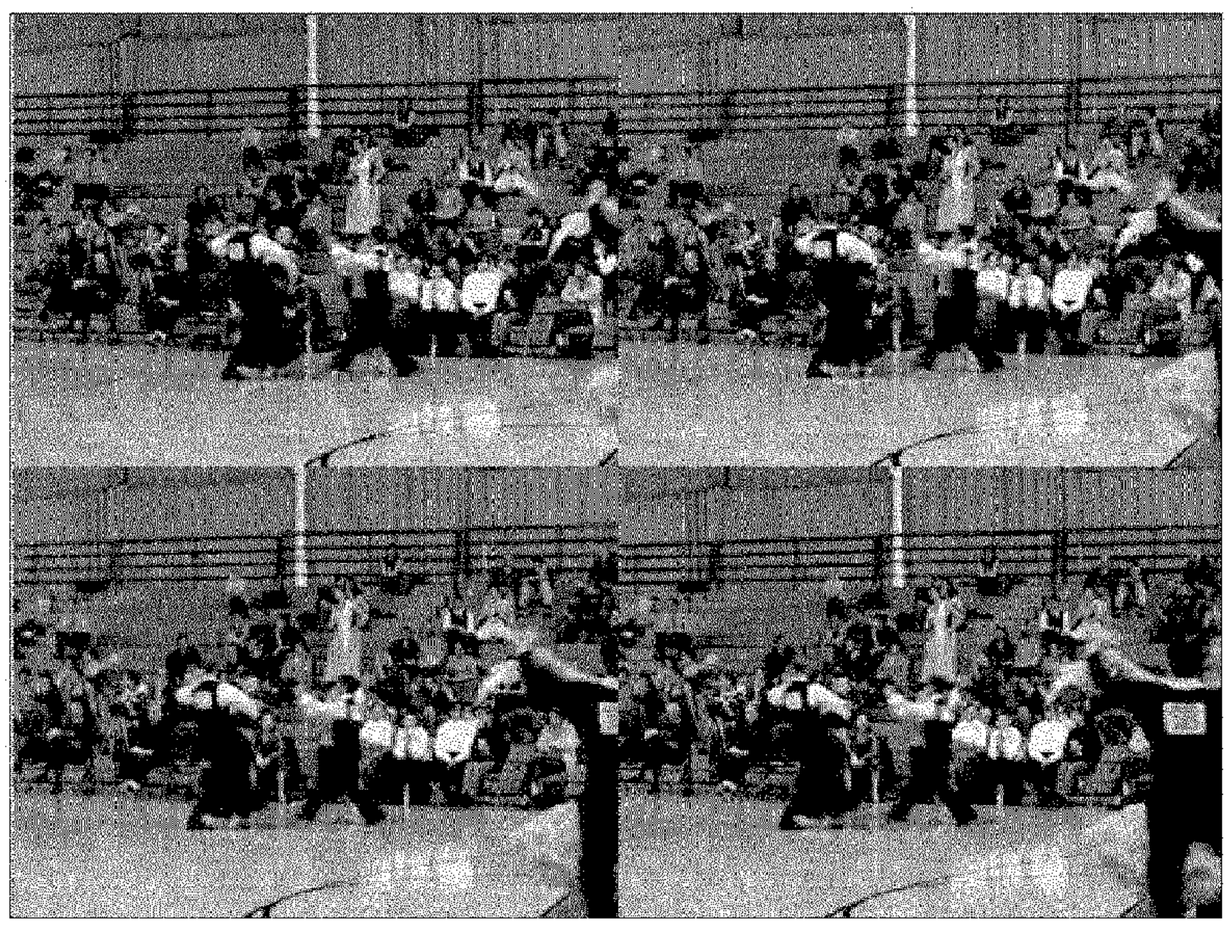


*FIG. 1*

200

*FIG. 2*

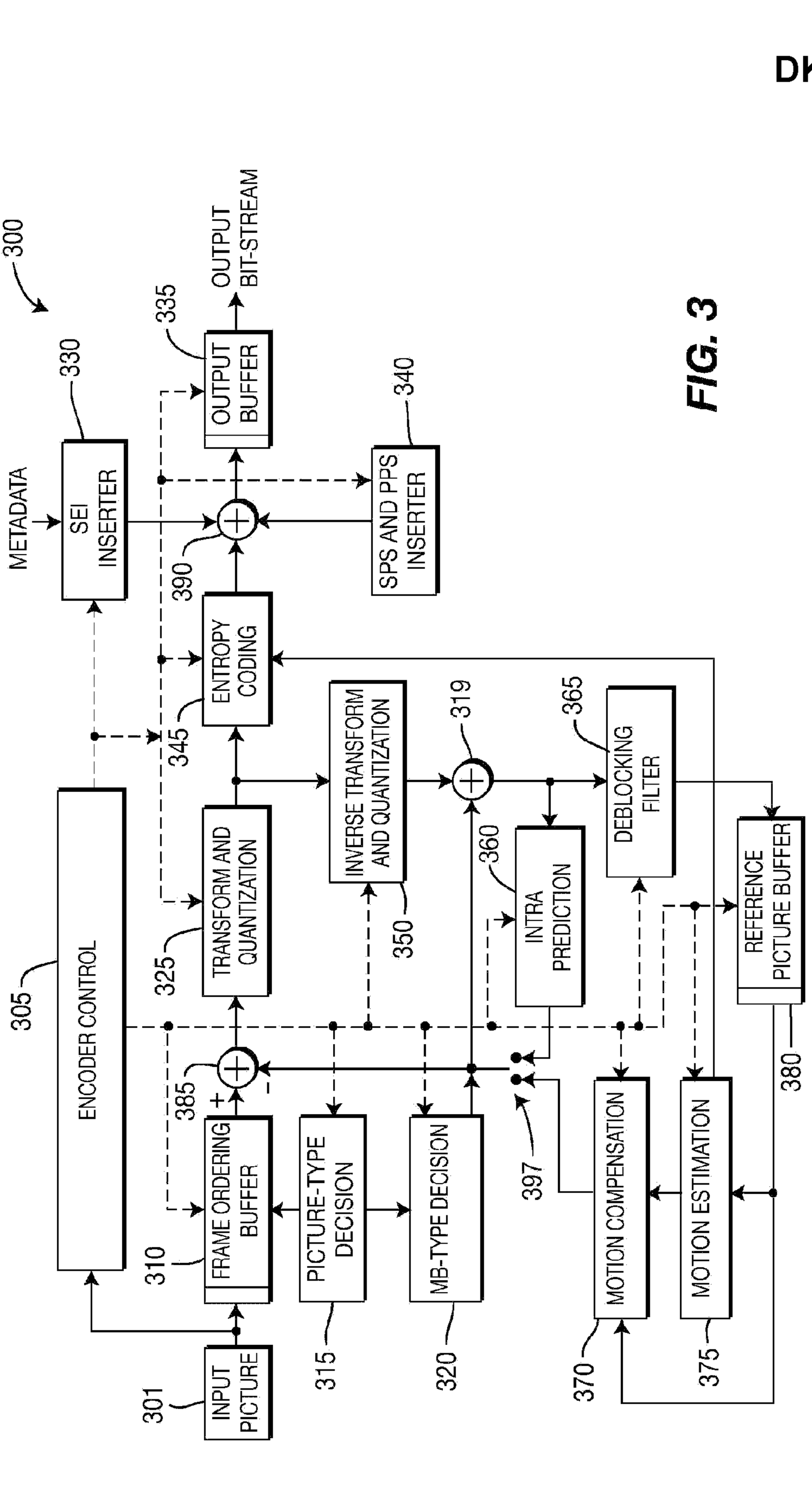

300
330
335
OUTPUT BIT-STREAM
OUTPUT BUFFER
340
METADATA
SEI INSERTER
SPS AND PPS INSERTER
390
ENTROPY CODING
345
305
ENCODER CONTROL
325
TRANSFORM AND QUANTIZATION
INVERSE TRANSFORM AND QUANTIZATION
350
319
365
DEBLOCKING FILTER
360
INTRA PREDICTION
REFERENCE PICTURE BUFFER
380
385
+
-
310
FRAME ORDERING BUFFER
PICTURE-TYPE DECISION
MB-TYPE DECISION
397
MOTION COMPENSATION
MOTION ESTIMATION
370
375
315
320
301
INPUT PICTURE

*FIG. 3*

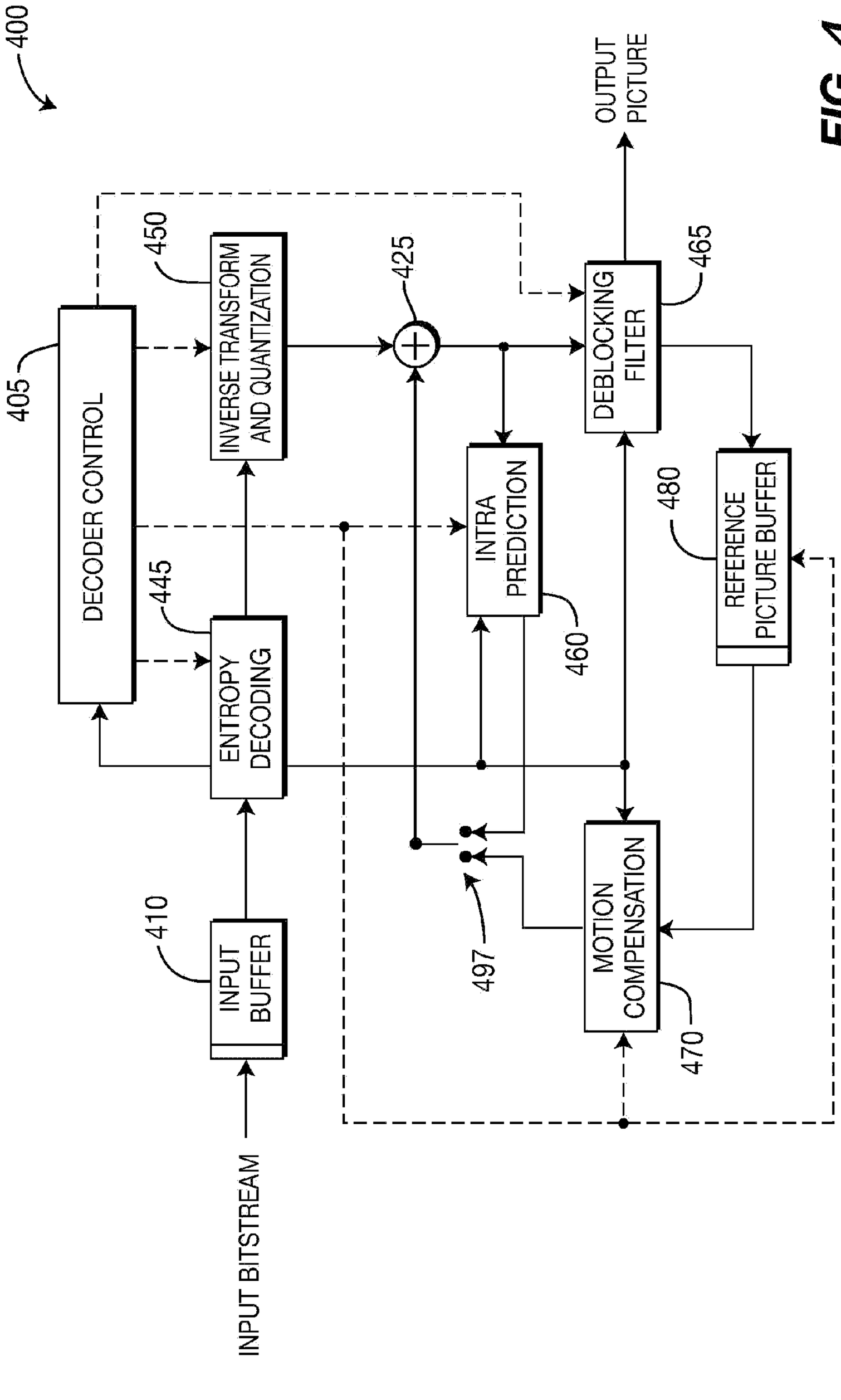

400
405
DECODER CONTROL
410
INPUT BUFFER
INPUT BITSTREAM
445
ENTROPY DECODING
450
INVERSE TRANSFORM AND QUANTIZATION
425
460
INTRA PREDICTION
497
465
DEBLOCKING FILTER
480
REFERENCE PICTURE BUFFER
470
MOTION COMPENSATION
OUTPUT PICTURE

FIG. 4

FIG. 5

FIG. 5A

FIG. 5B

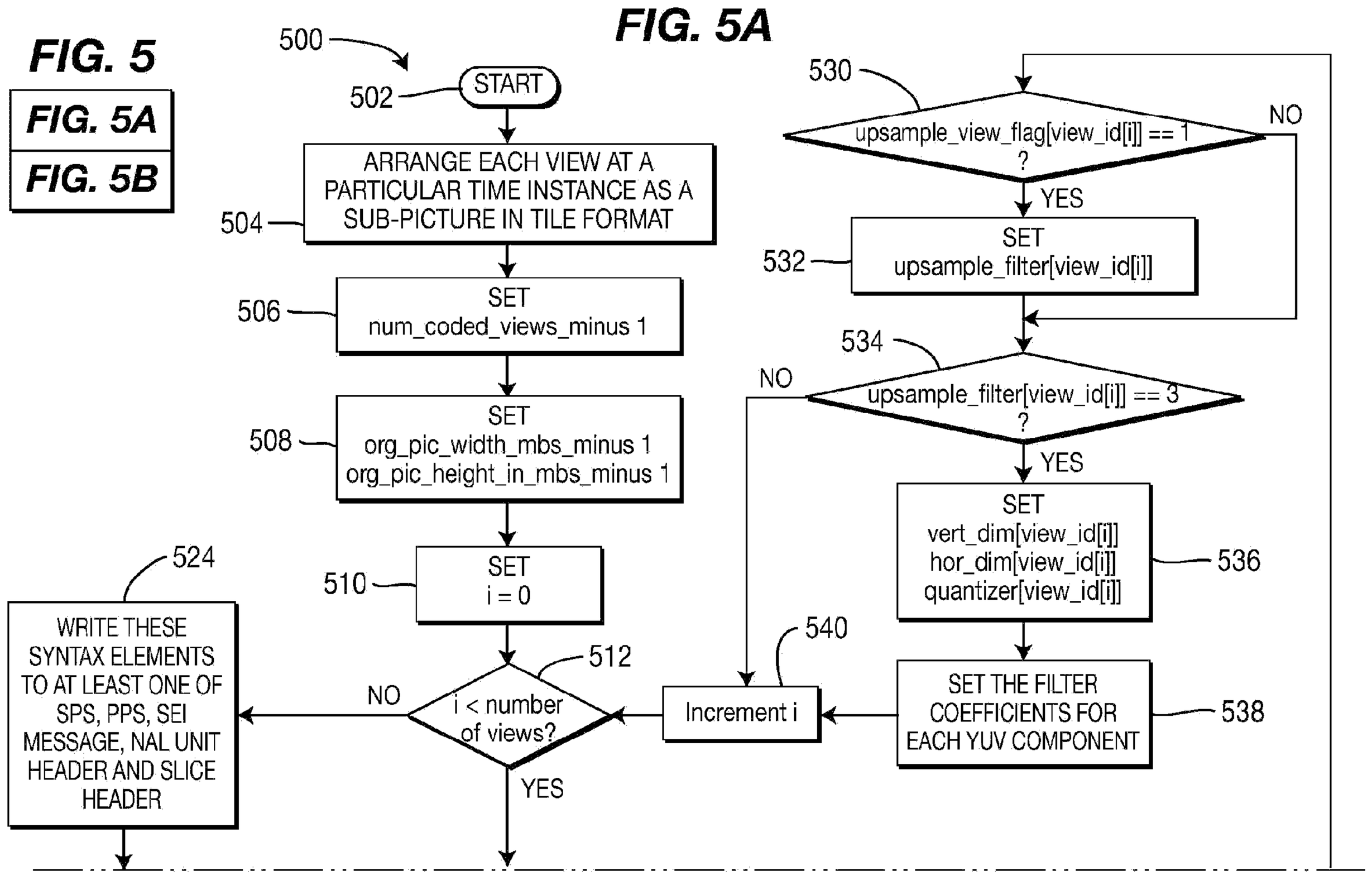



FIG. 5A

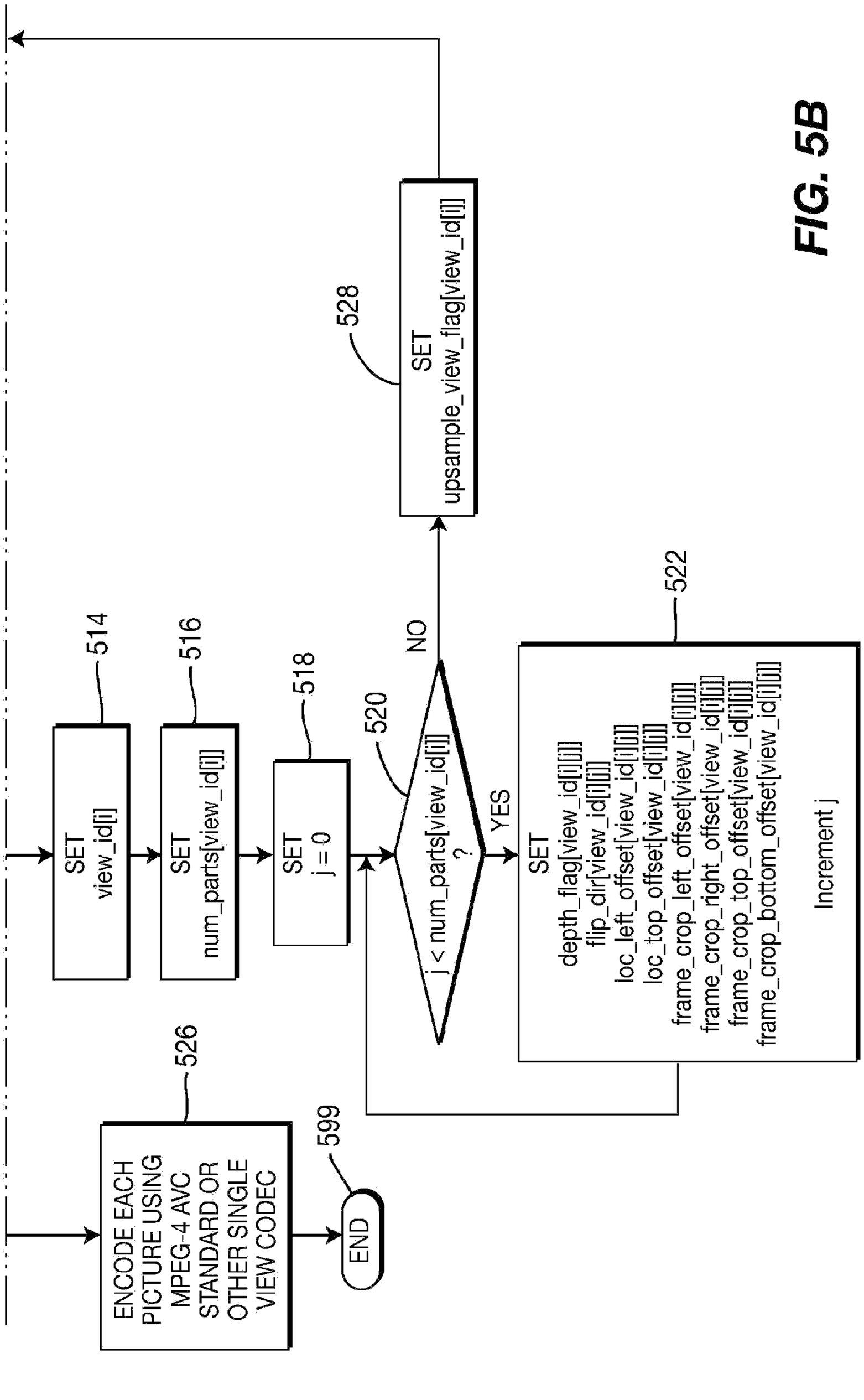

514
SET
view_id[i]
516
SET
num_parts[view_id[i]]
518
SET
j = 0
520
j < num_parts[view_id[i]]
?
NO
YES
522
SET
depth_flag[view_id[i]][j]
flip_dir[view_id[i][j]]
loc_left_offset[view_id[i][j]]
loc_top_offset[view_id[i][j]]
frame_crop_left_offset[view_id[i][j]]
frame_crop_right_offset[view_id[i][j]]
frame_crop_top_offset[view_id[i][j]]
frame_crop_bottom_offset[view_id[i][j]]
Increment j
528
SET
upsample_view_flag[view_id[i]]
526
ENCODE EACH
PICTURE USING
MPEG-4 AVC
STANDARD OR
OTHER SINGLE
VIEW CODEC
599
END

FIG. 5B

FIG. 6

FIG. 6A

FIG. 6B

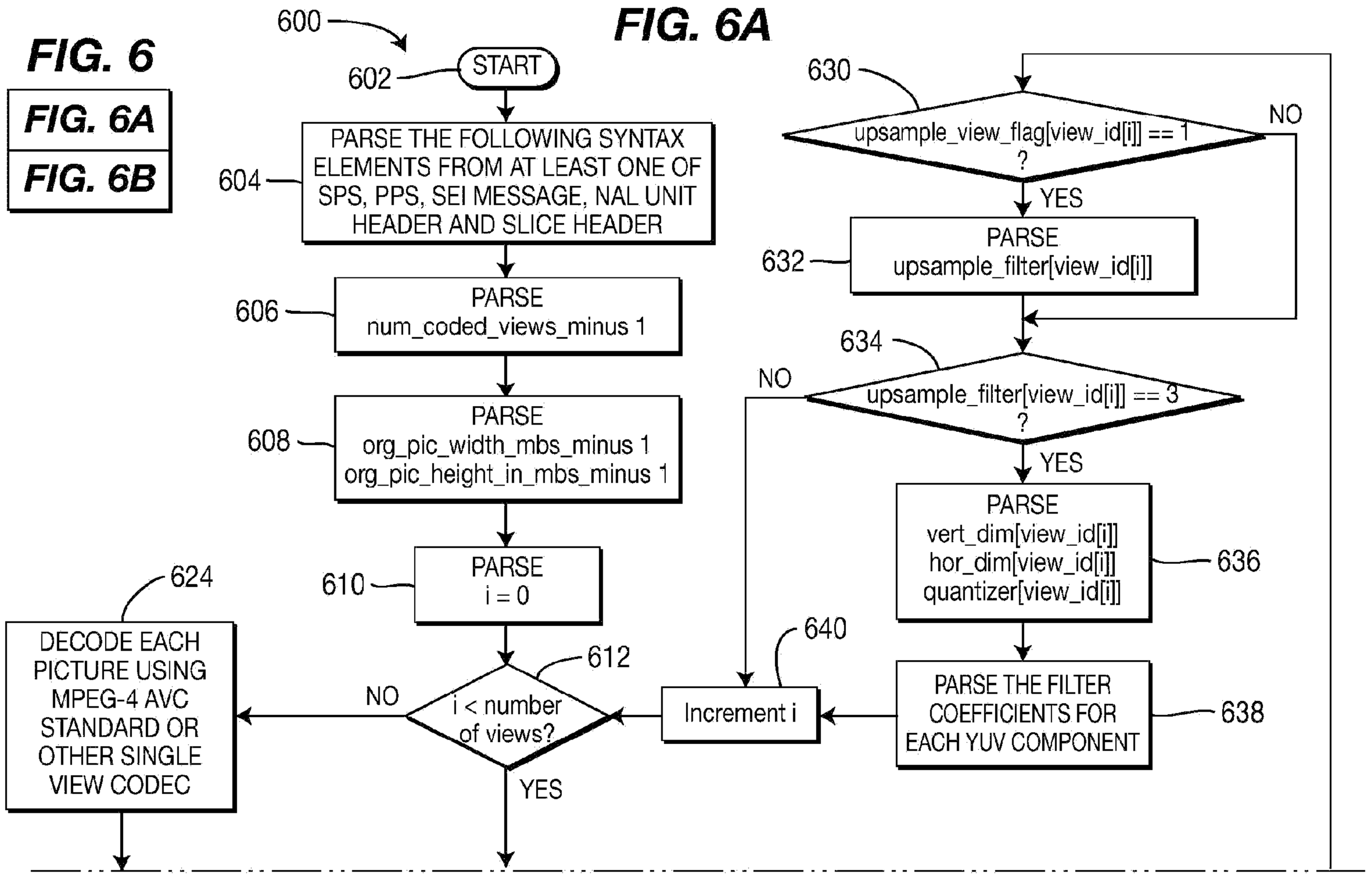



FIG. 6A

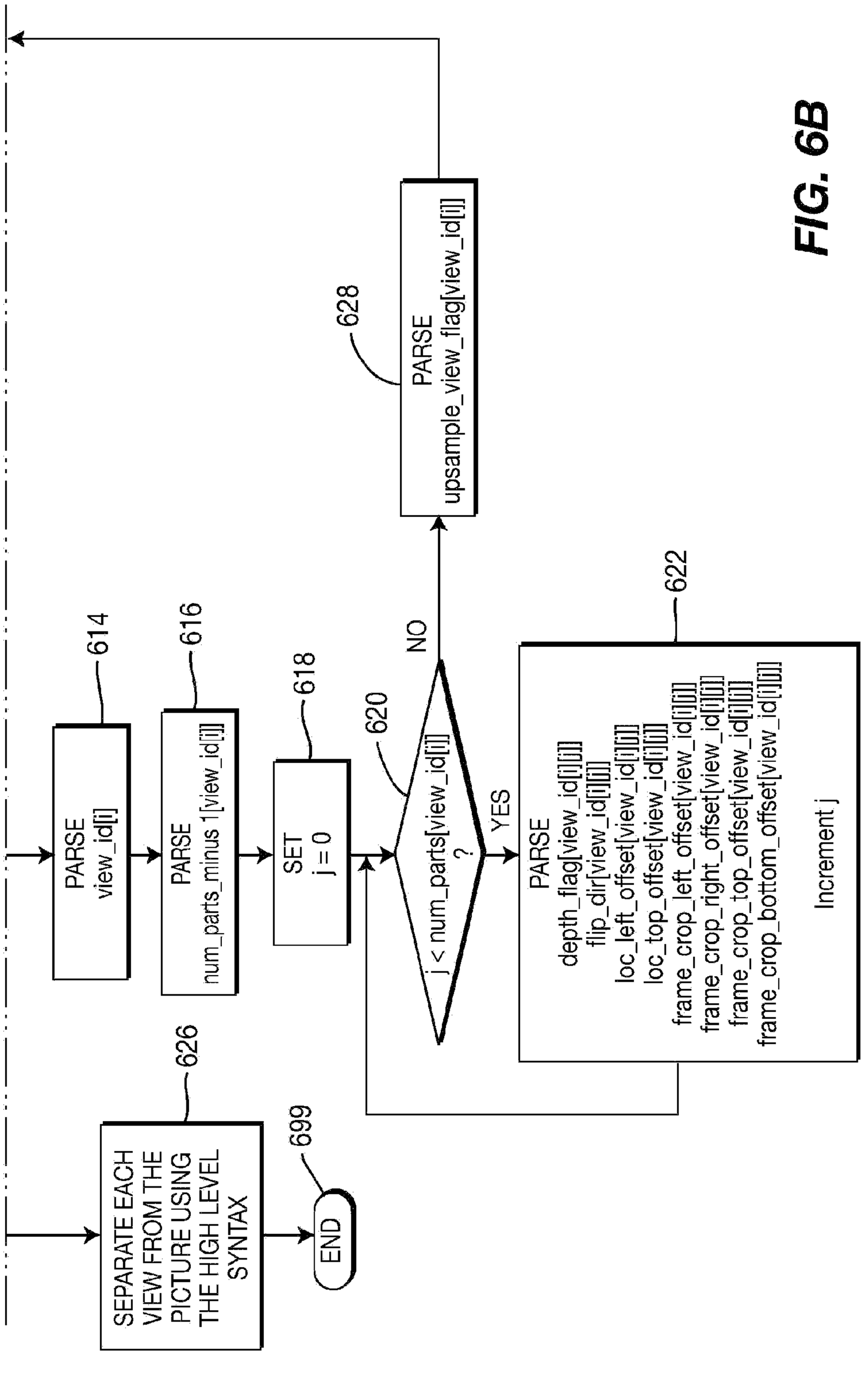

SEPARATE EACH VIEW FROM THE PICTURE USING THE HIGH LEVEL SYNTAX
626
PARSE view_id[i]
614
PARSE num_parts_minus 1[view_id[i]]
616
SET j = 0
618
j < num_parts[view_id[i]] ?
620
NO
YES
PARSE depth_flag[view_id[i]][j]
flip_dir[view_id[i]][j]
loc_left_offset[view_id[i]][j]
loc_top_offset[view_id[i]][j]
frame_crop_left_offset[view_id[i]][j]
frame_crop_right_offset[view_id[i]][j]
frame_crop_top_offset[view_id[i]][j]
frame_crop_bottom_offset[view_id[i]][j]
Increment j
622
PARSE upsample_view_flag[view_id[i]]
628
END
699

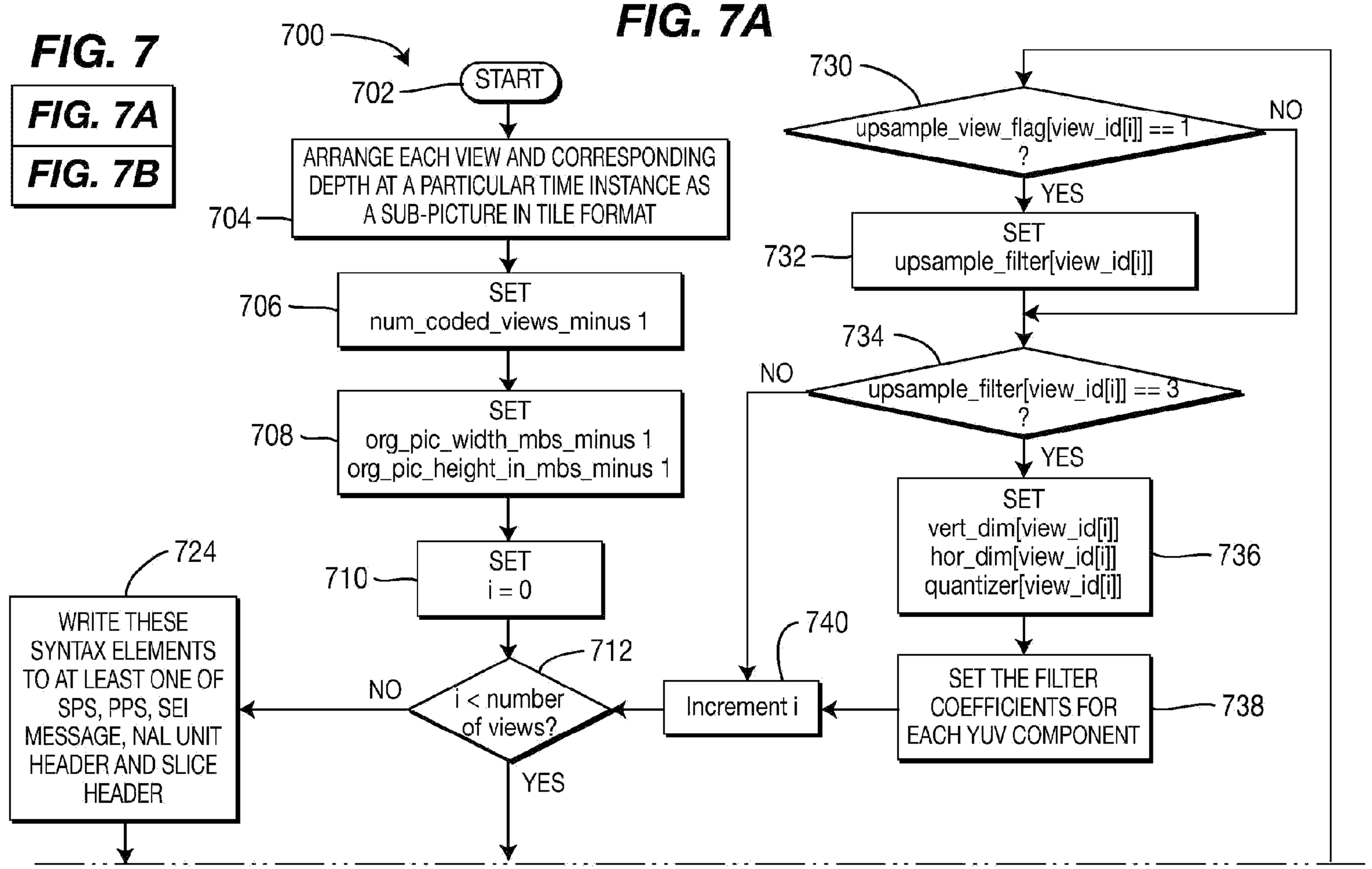
FIG. 6B

FIG. 7

FIG. 7A

FIG. 7B

FIG. 7A



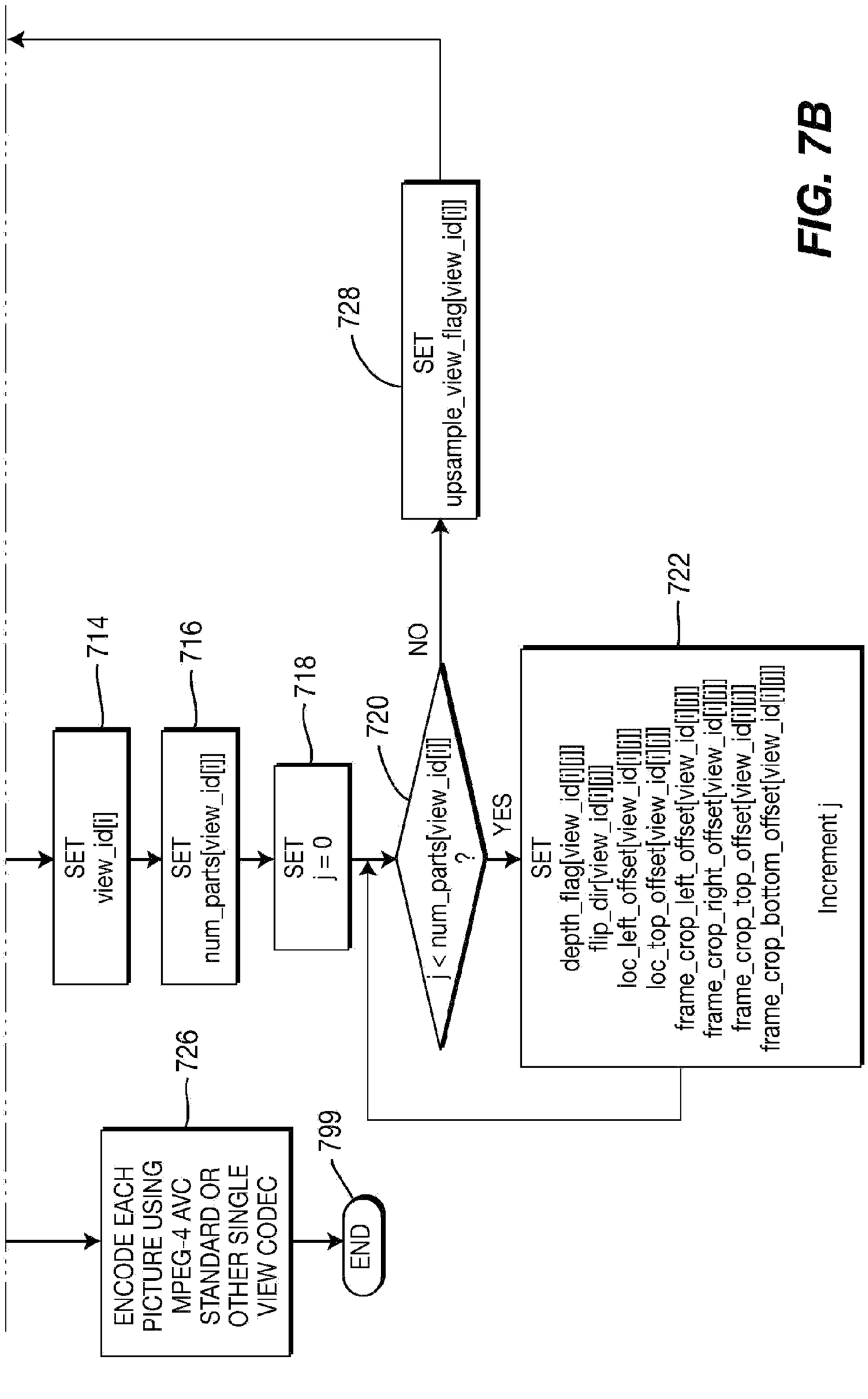

714
SET
view_id[i]
716
SET
num_parts[view_id[i]]
718
SET
j = 0
720
j < num_parts[view_id[i]]
?
NO
YES
722
SET
depth_flag[view_id[i]][j]
flip_dir[view_id[i][j]]
loc_left_offset[view_id[i][j]]
loc_top_offset[view_id[i][j]]
frame_crop_left_offset[view_id[i][j]]
frame_crop_right_offset[view_id[i][j]]
frame_crop_top_offset[view_id[i][j]]
frame_crop_bottom_offset[view_id[i][j]]
Increment j
724
SET
upsample_view_flag[view_id[i]]
726
ENCODE EACH
PICTURE USING
MPEG-4 AVC
STANDARD OR
OTHER SINGLE
VIEW CODEC
799
END

FIG. 7B

FIG. 8

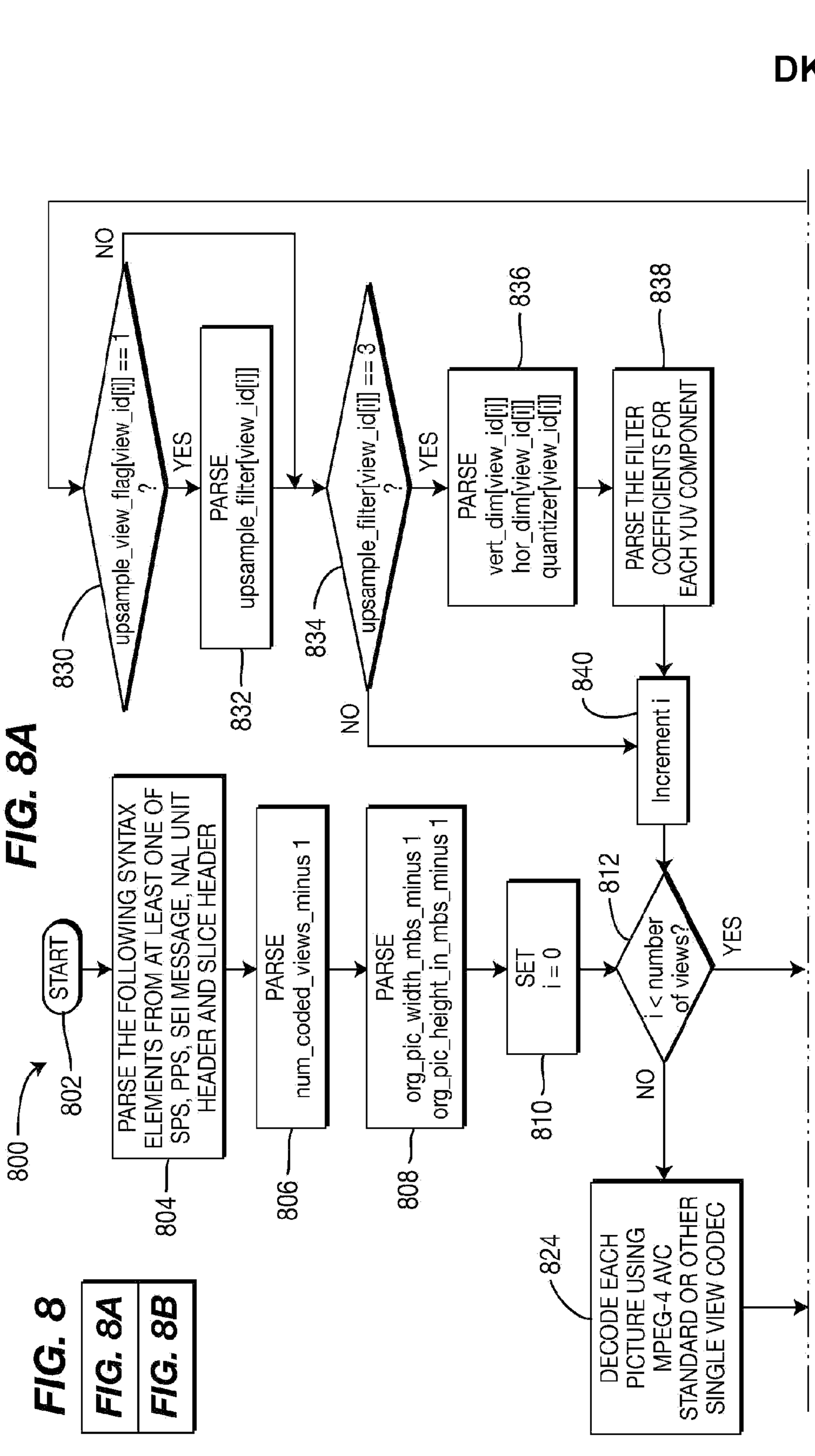



FIG. 8A

FIG. 8B

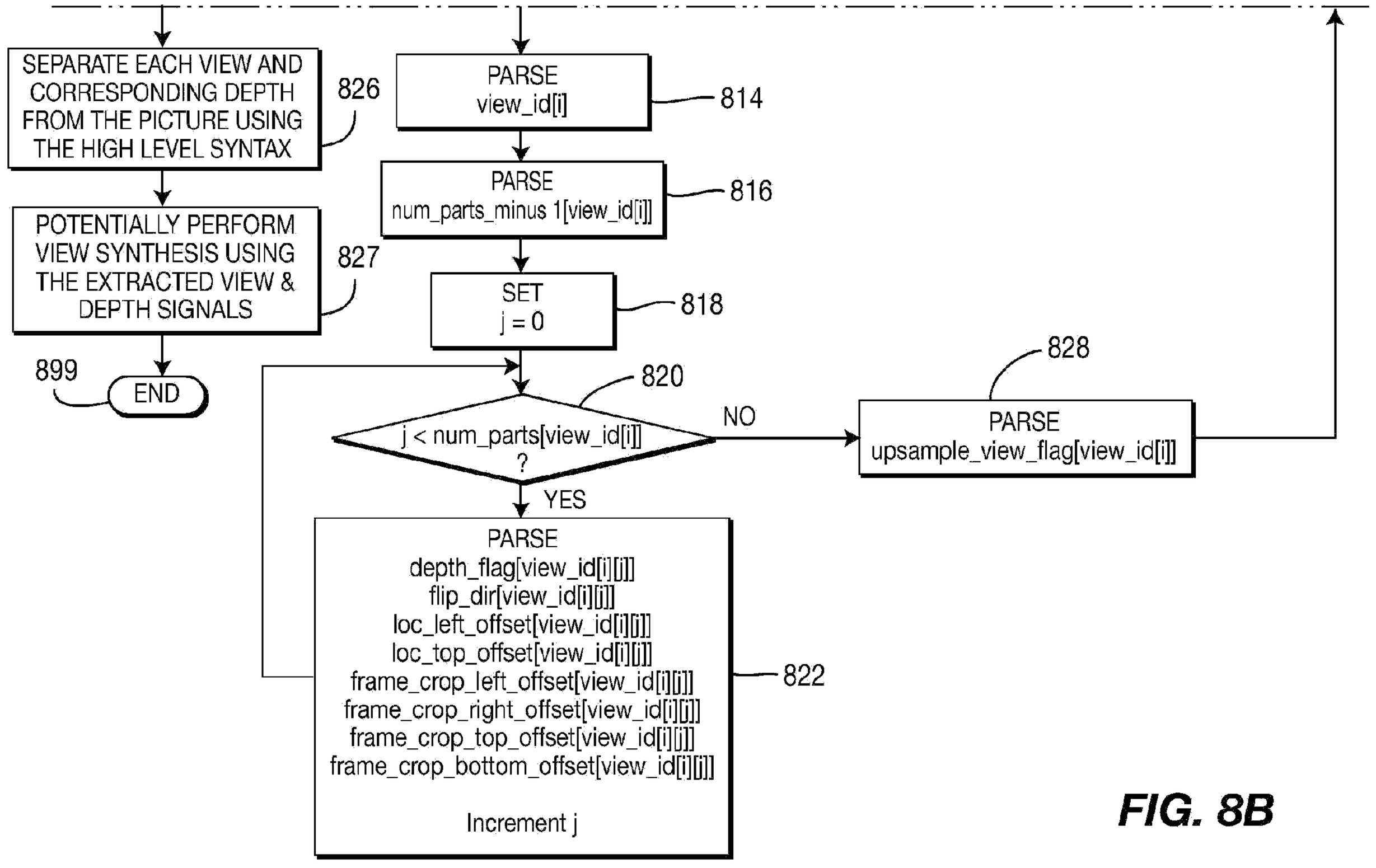



*FIG. 8B*

900

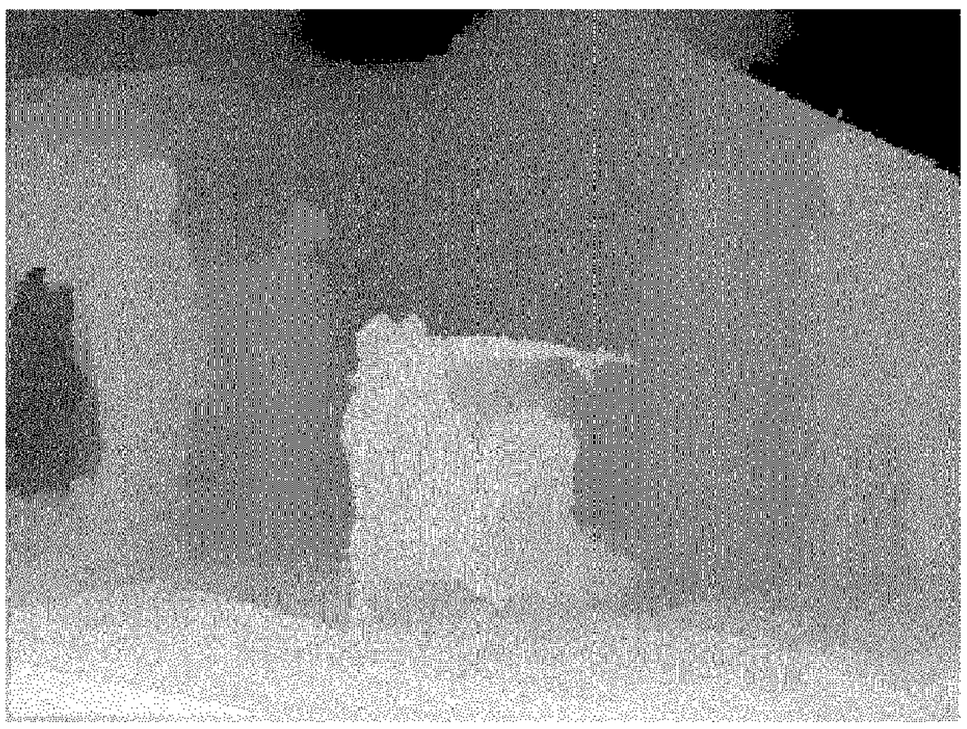


FIG. 9

1000

FIG. 10

1100

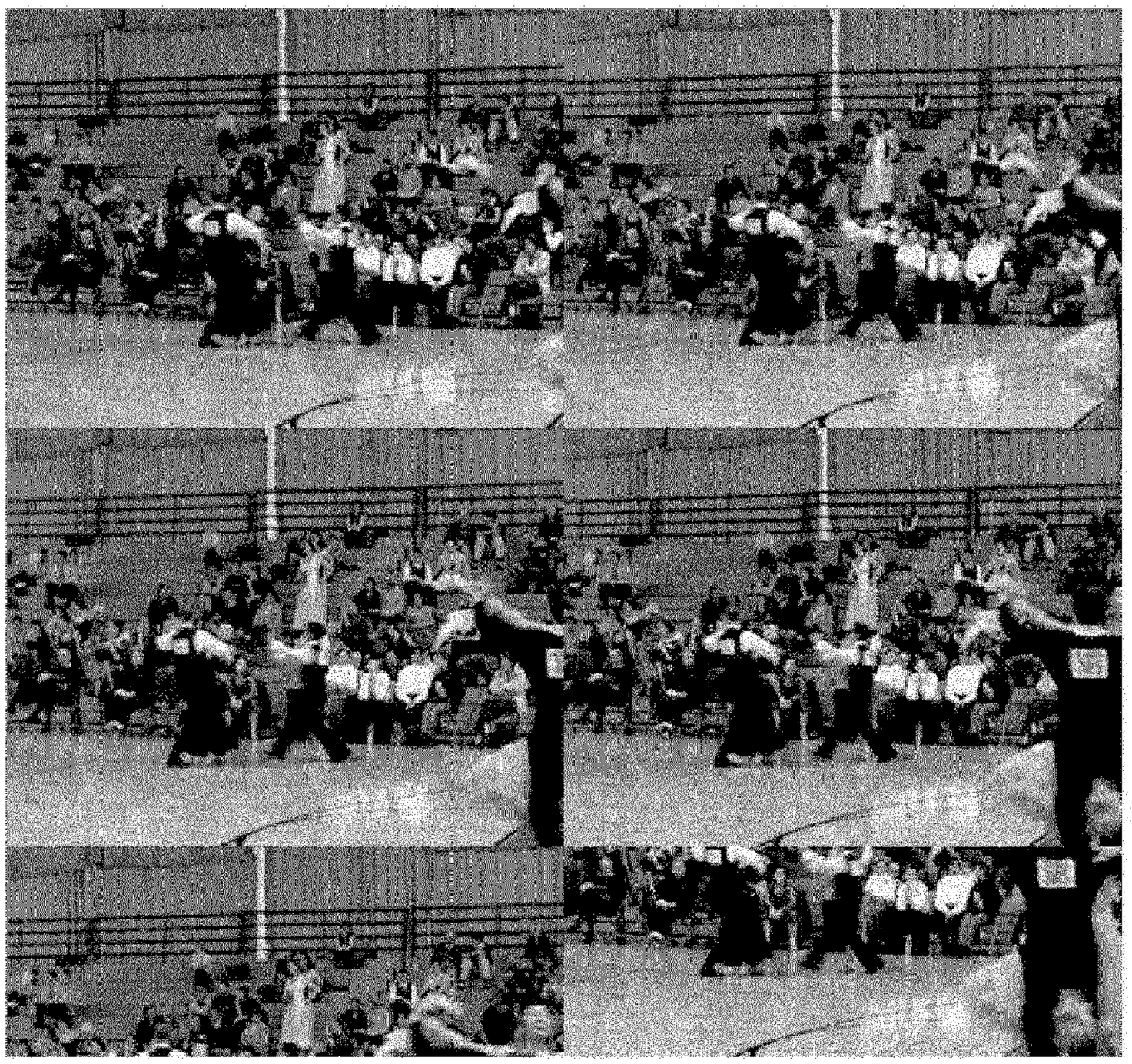


FIG. 11

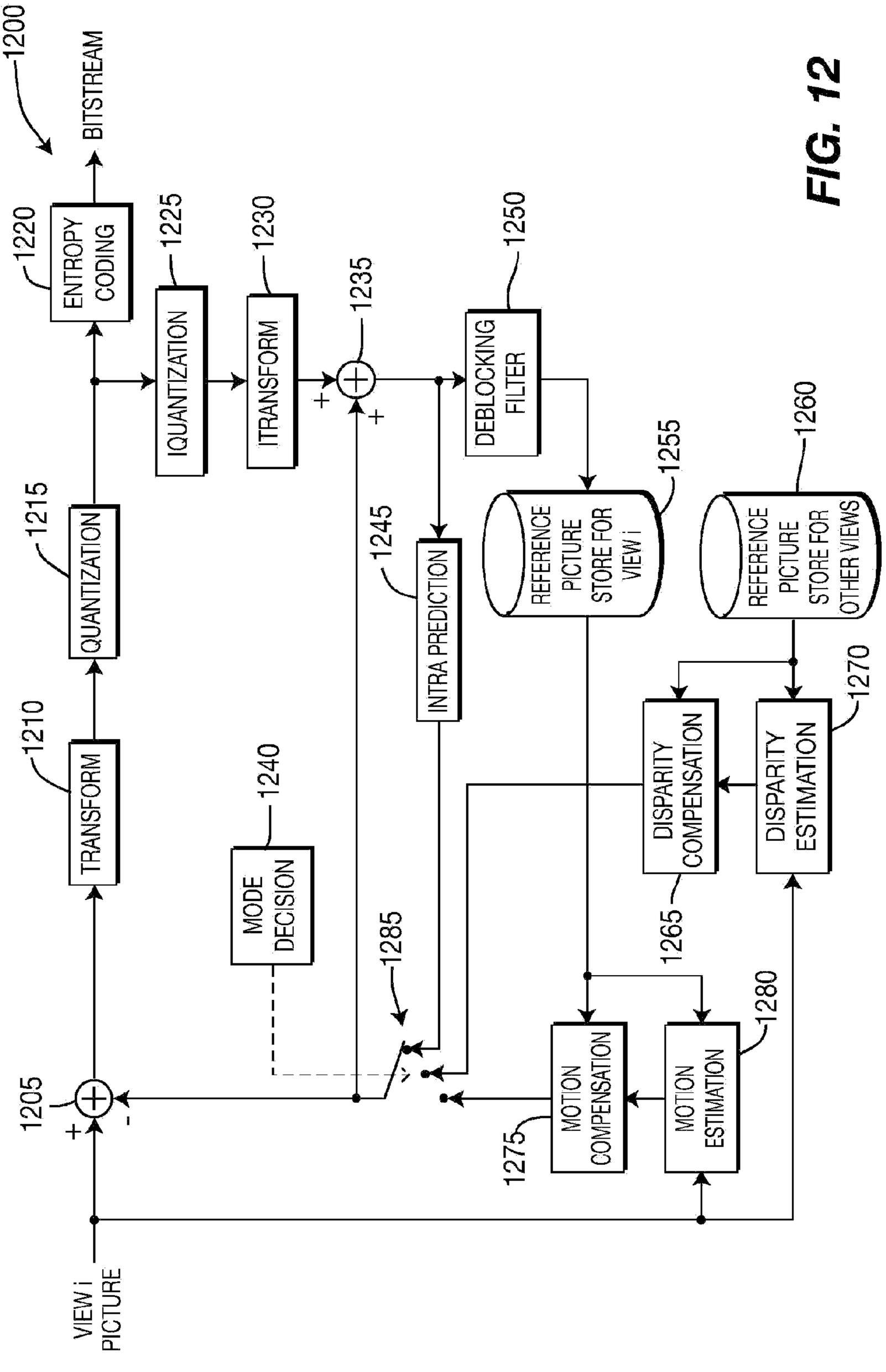

VIEW i PICTURE
1205
+
-
TRANSFORM
1210
QUANTIZATION
1215
ENTROPY CODING
1220
BITSTREAM
IQUANTIZATION
ITRANSFORM
1225
1230
1235
+
+
MODE DECISION
1240
INTRA PREDICTION
1245
DEBLOCKING FILTER
REFERENCE PICTURE STORE FOR VIEW i
1255
REFERENCE PICTURE STORE FOR OTHER VIEWS
1260
DISPARITY ESTIMATION
1270
DISPARITY COMPENSATION
1265
MOTION ESTIMATION
1280
MOTION COMPENSATION
1275
1285

*FIG. 12*

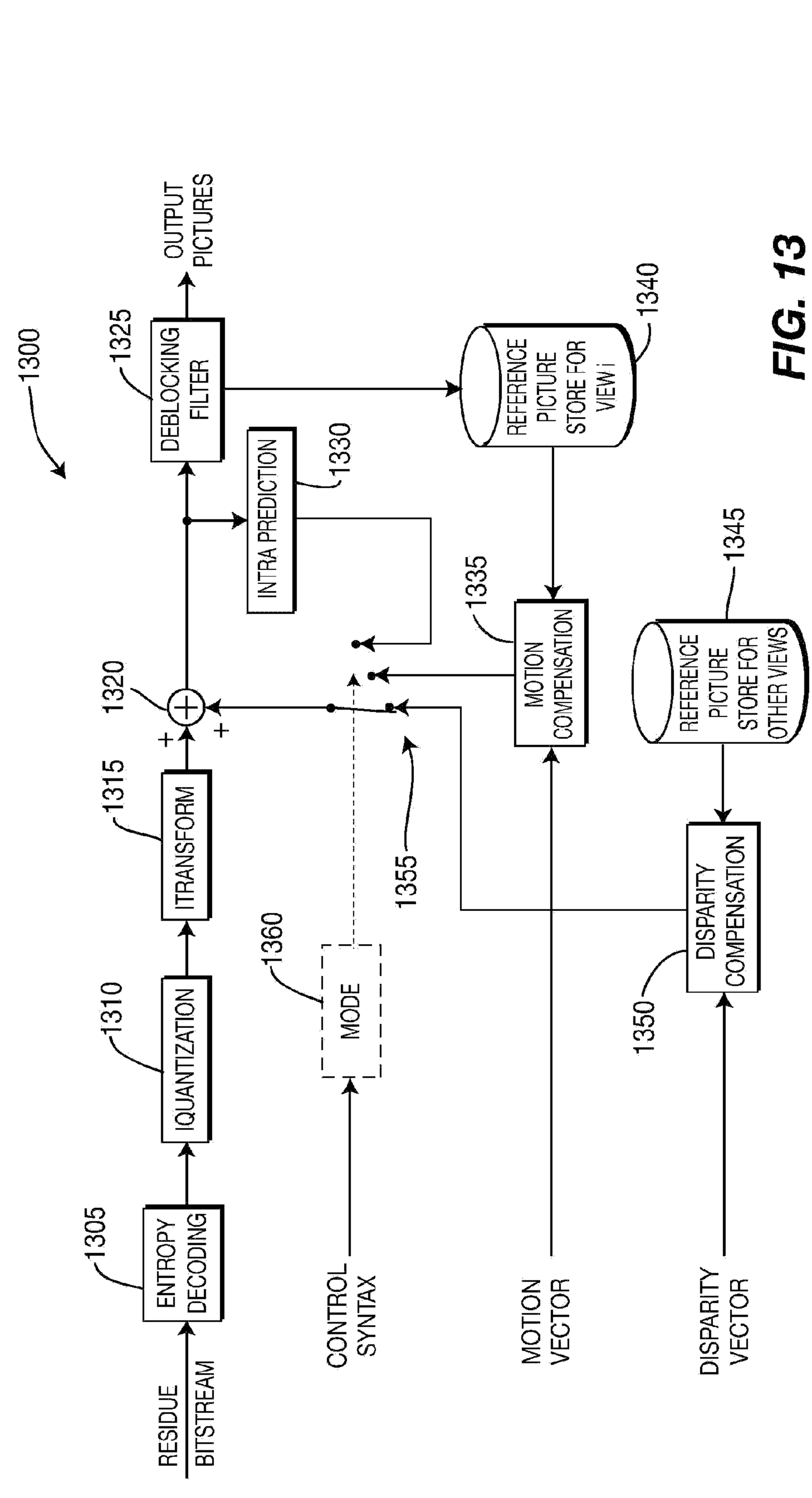

1300
1325
DEBLOCKING FILTER
OUTPUT PICTURES
1340
REFERENCE PICTURE STORE FOR VIEW i
1330
INTRA PREDICTION
1335
MOTION COMPENSATION
1345
REFERENCE PICTURE STORE FOR OTHER VIEWS
1320
+
+
1315
ITRANSFORM
1355
1360
1365
MODE
1310
IQUANTIZATION
1305
ENTROPY DECODING
RESIDUE BITSTREAM
CONTROL SYNTAX
MOTION VECTOR
1350
DISPARITY COMPENSATION
DISPARITY VECTOR

FIG. 13

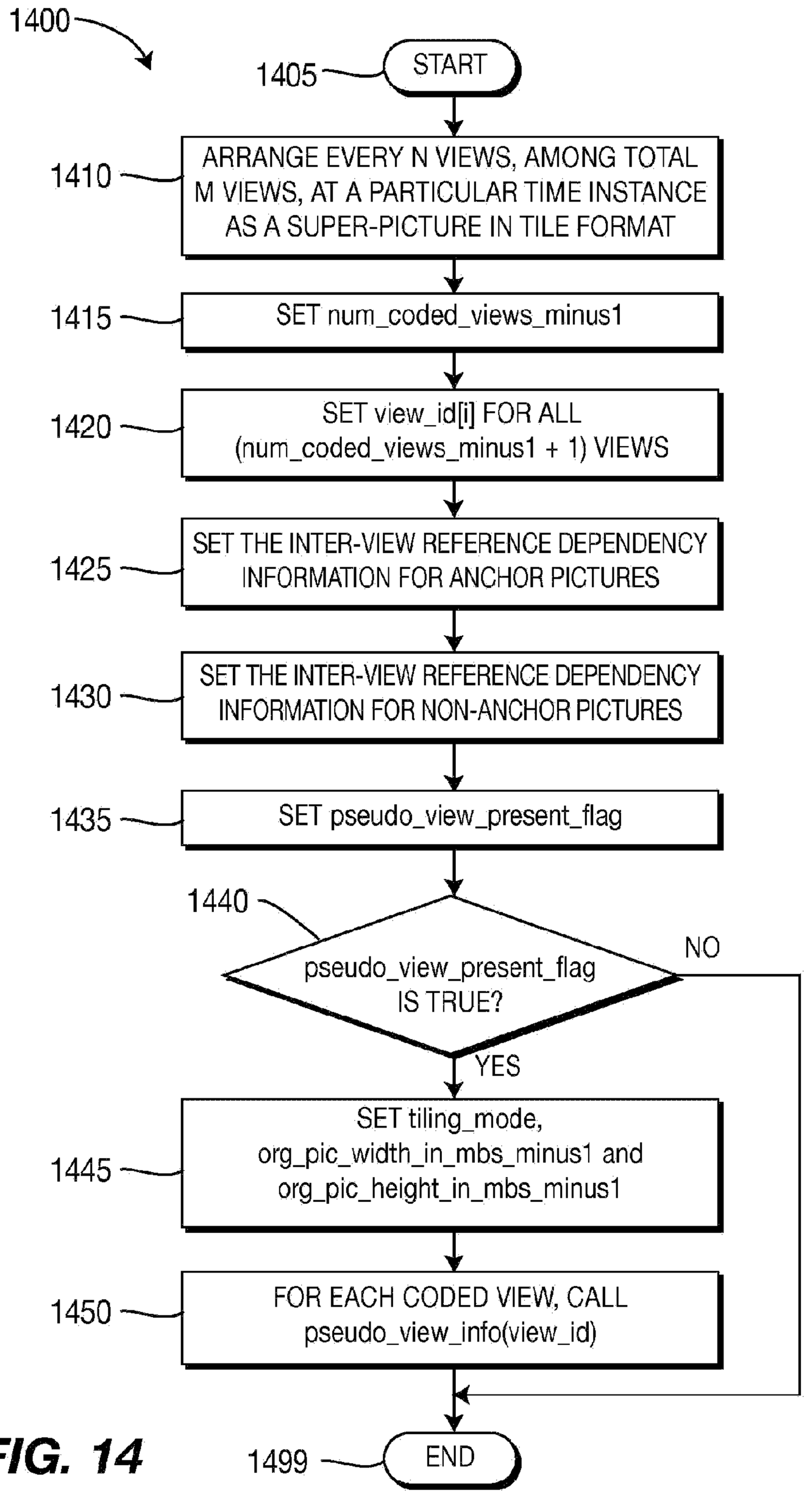

1400
1405
START
1410
ARRANGE EVERY N VIEWS, AMONG TOTAL M VIEWS, AT A PARTICULAR TIME INSTANCE AS A SUPER-PICTURE IN TILE FORMAT
1415
SET num_coded_views_minus1
1420
SET view_id[i] FOR ALL (num_coded_views_minus1 + 1) VIEWS
1425
SET THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR ANCHOR PICTURES
1430
SET THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR NON-ANCHOR PICTURES
1435
SET pseudo_view_present_flag
1440
pseudo_view_present_flag IS TRUE?
NO
YES
1445
SET tiling_mode, org_pic_width_in_mbs_minus1 and org_pic_height_in_mbs_minus1
1450
FOR EACH CODED VIEW, CALL pseudo_view_info(view_id)
1499
END

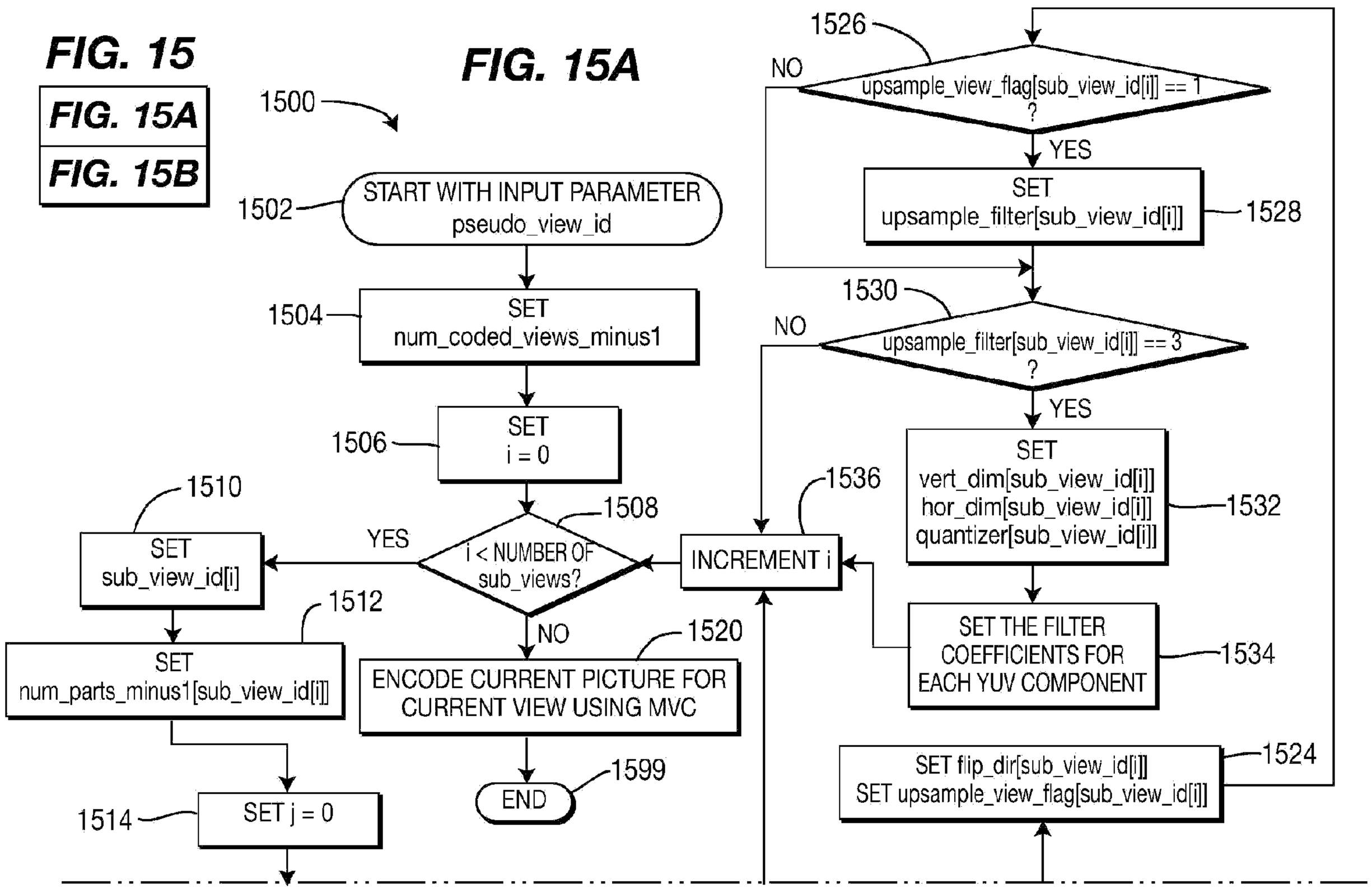
FIG. 14

**FIG. 15**

**FIG. 15A**

**FIG. 15B**

**FIG. 15A**



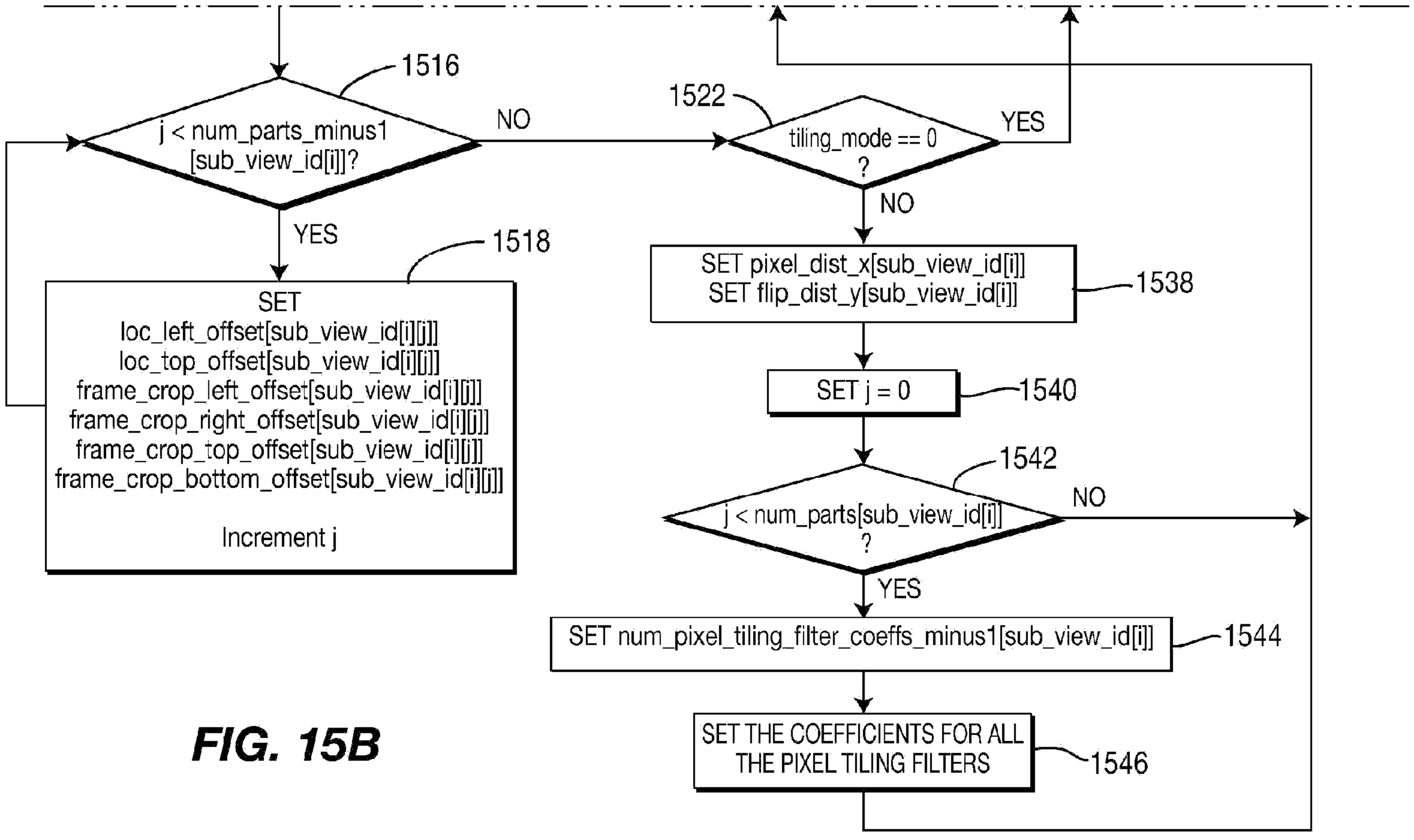



*FIG. 15B*

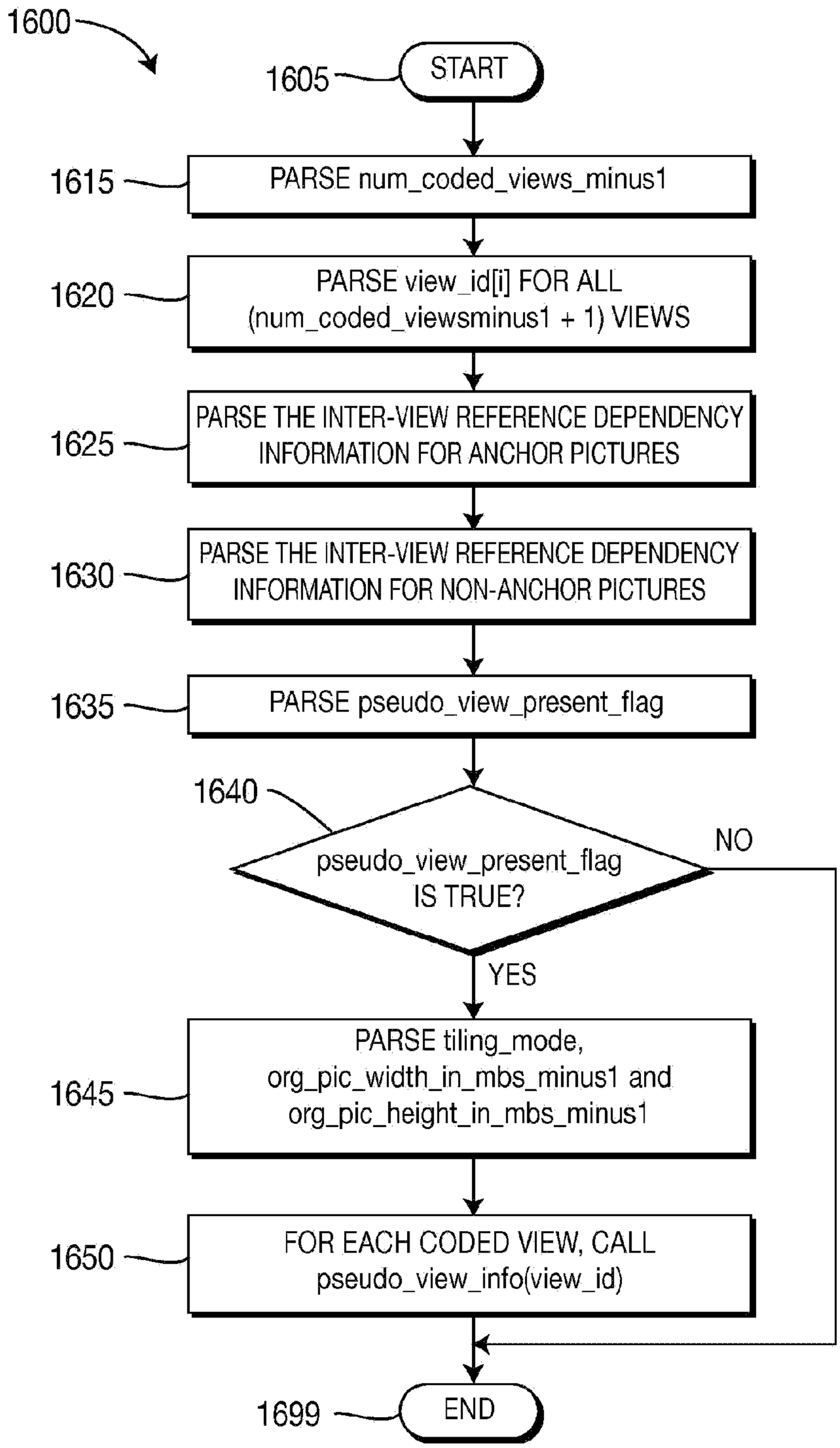
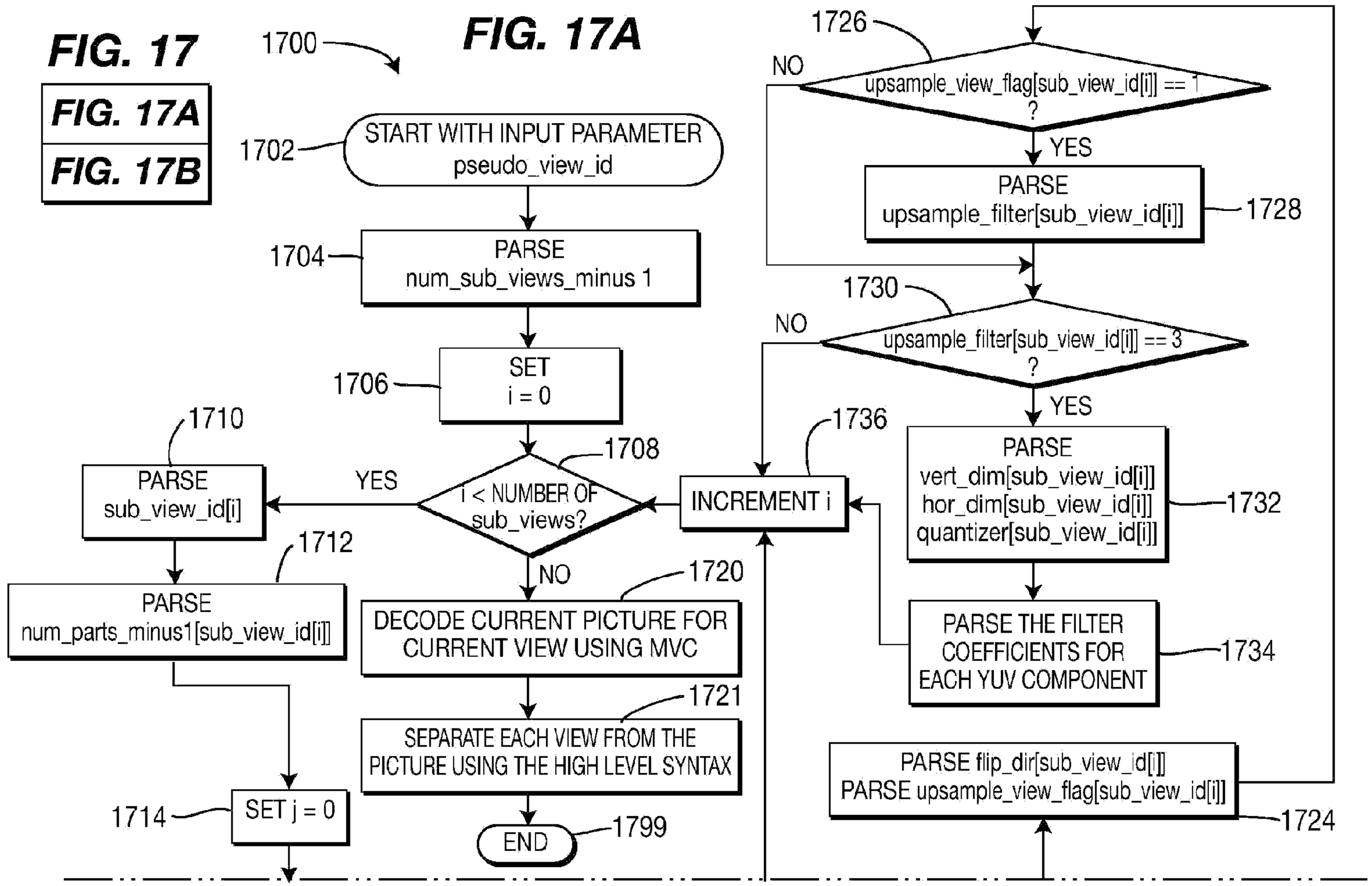

1600
1605
START
1615
PARSE num_coded_views_minus1
1620
PARSE view_id[i] FOR ALL
(num_coded_viewsminus1 + 1) VIEWS
1625
PARSE THE INTER-VIEW REFERENCE DEPENDENCY
INFORMATION FOR ANCHOR PICTURES
1630
PARSE THE INTER-VIEW REFERENCE DEPENDENCY
INFORMATION FOR NON-ANCHOR PICTURES
1635
PARSE pseudo_view_present_flag
1640
pseudo_view_present_flag
IS TRUE?
NO
YES
1645
PARSE tiling_mode,
org_pic_width_in_mbs_minus1 and
org_pic_height_in_mbs_minus1
1650
FOR EACH CODED VIEW, CALL
pseudo_view_info(view_id)
1699
END

*FIG. 16*

FIG. 17

FIG. 17A

FIG. 17B



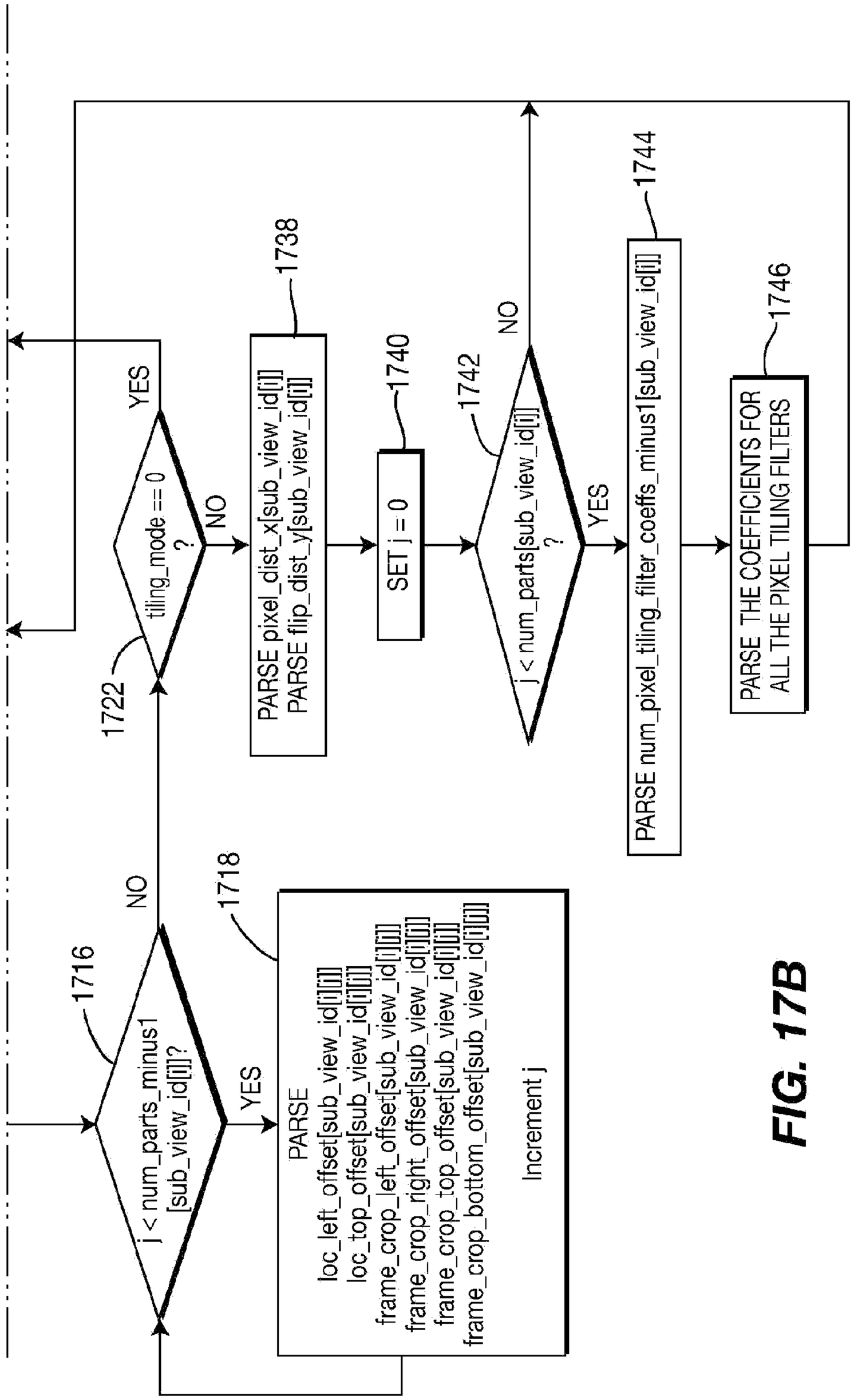

1722
tiling_mode == 0 ?
YES
NO
PARSE pixel_dist_x[sub_view_id[i]]
PARSE flip_dist_y[sub_view_id[i]]
1740
SET j = 0
1742
j < num_parts[sub_view_id[i]] ?
NO
YES
PARSE num_pixel_tiling_filter_coeffs_minus1[sub_view_id[i]]
1746
PARSE THE COEFFICIENTS FOR ALL THE PIXEL TILING FILTERS
1744
1738
1718
1716
j < num_parts_minus1 [sub_view_id[i]]?
NO
YES
PARSE
loc_left_offset[sub_view_id[i][j]]
loc_top_offset[sub_view_id[i][j]]
frame_crop_left_offset[sub_view_id[i][j]]
frame_crop_right_offset[sub_view_id[i][j]]
frame_crop_top_offset[sub_view_id[i][j]]
frame_crop_bottom_offset[sub_view_id[i][j]]
Increment j

*FIG. 17B*

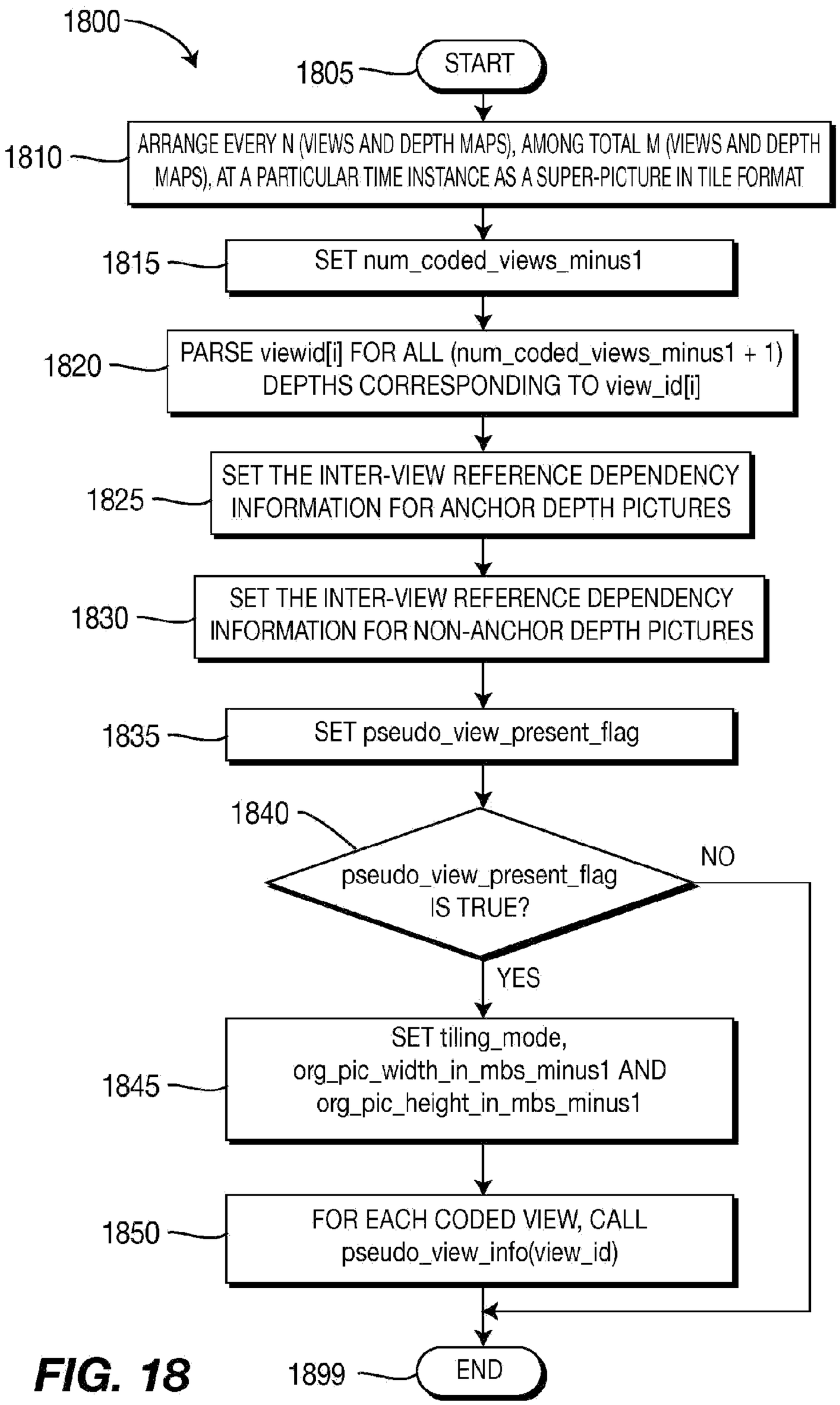

1800
1805
START
1810
ARRANGE EVERY N (VIEWS AND DEPTH MAPS), AMONG TOTAL M (VIEWS AND DEPTH MAPS), AT A PARTICULAR TIME INSTANCE AS A SUPER-PICTURE IN TILE FORMAT
1815
SET num_coded_views_minus1
1820
PARSE viewid[i] FOR ALL (num_coded_views_minus1 + 1) DEPTHS CORRESPONDING TO view_id[i]
1825
SET THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR ANCHOR DEPTH PICTURES
1830
SET THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR NON-ANCHOR DEPTH PICTURES
1835
SET pseudo_view_present_flag
1840
pseudo_view_present_flag IS TRUE?
NO
YES
1845
SET tiling_mode, org_pic_width_in_mbs_minus1 AND org_pic_height_in_mbs_minus1
1850
FOR EACH CODED VIEW, CALL pseudo_view_info(view_id)
1899
END

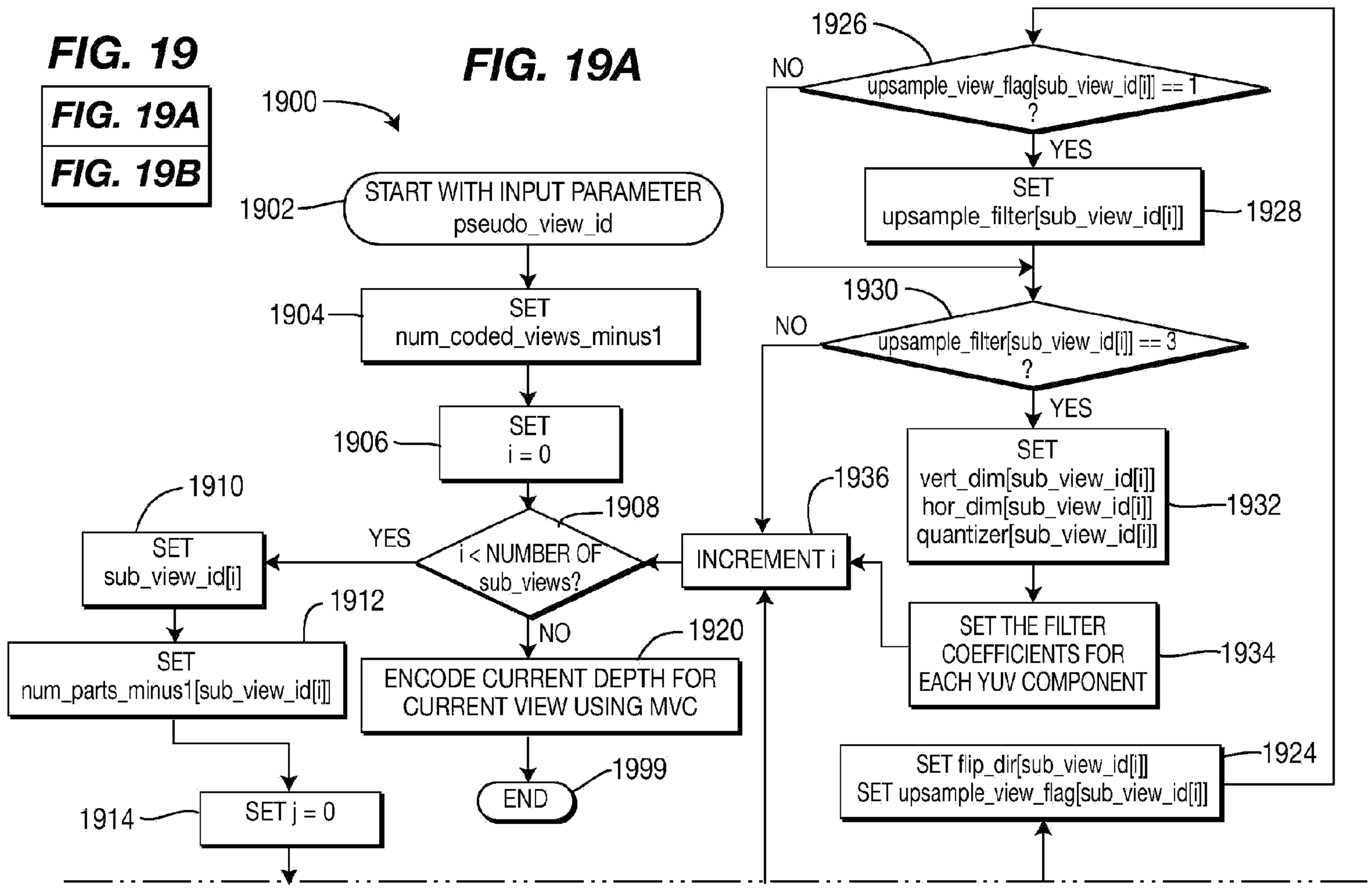
*FIG. 18*

**FIG. 19**

**FIG. 19A**

**FIG. 19B**

**FIG. 19A**



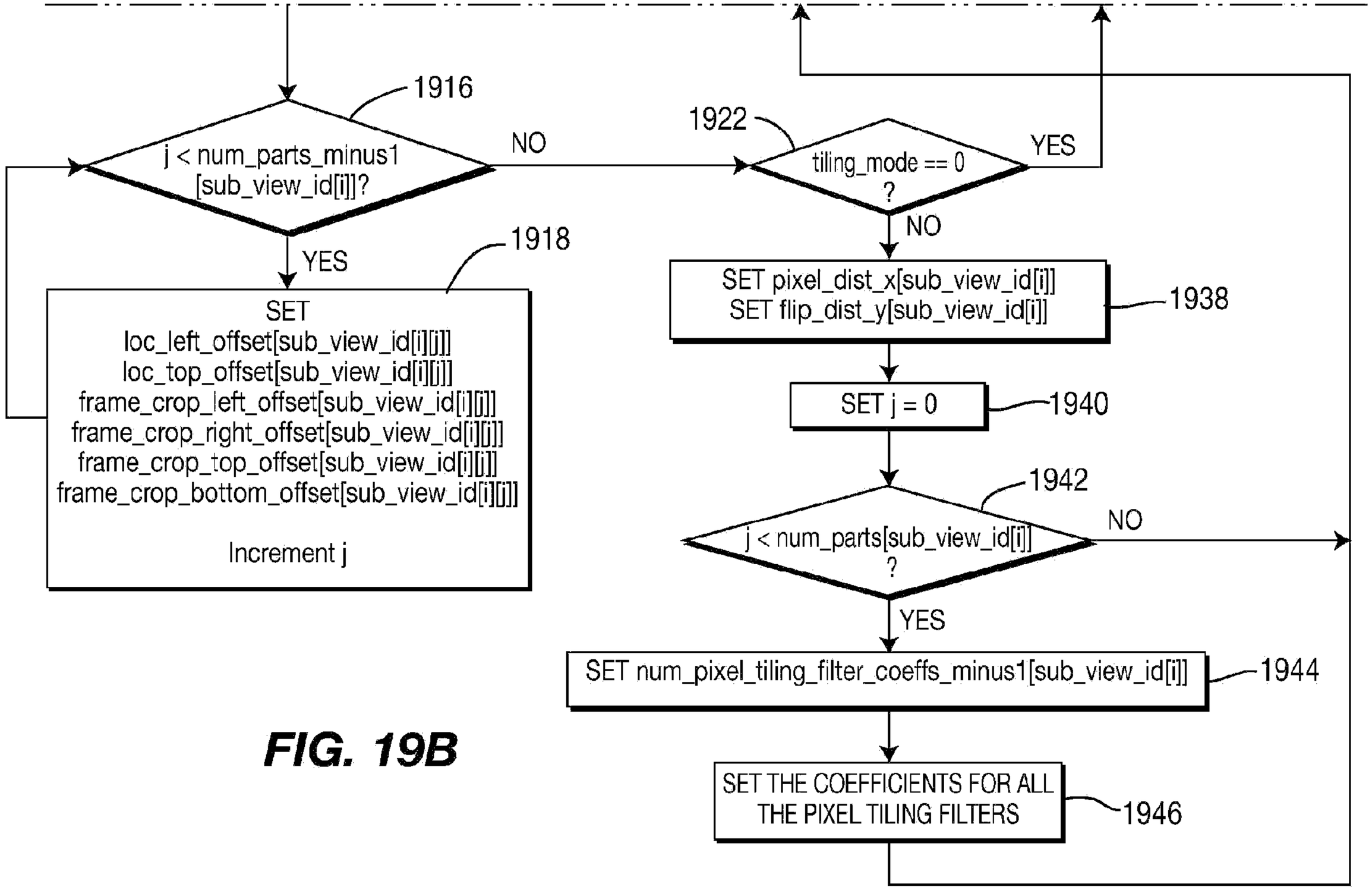



*FIG. 19B*

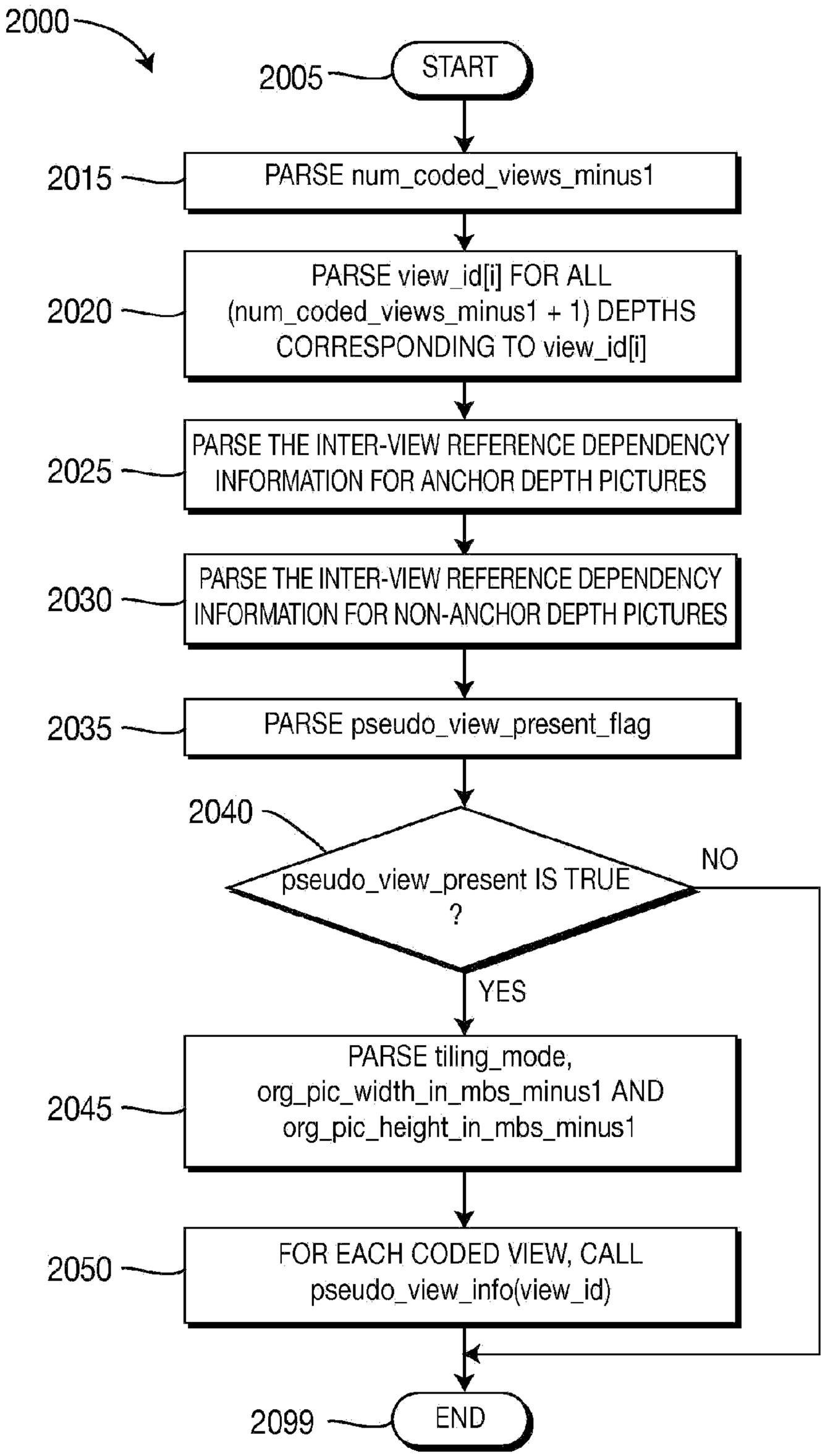
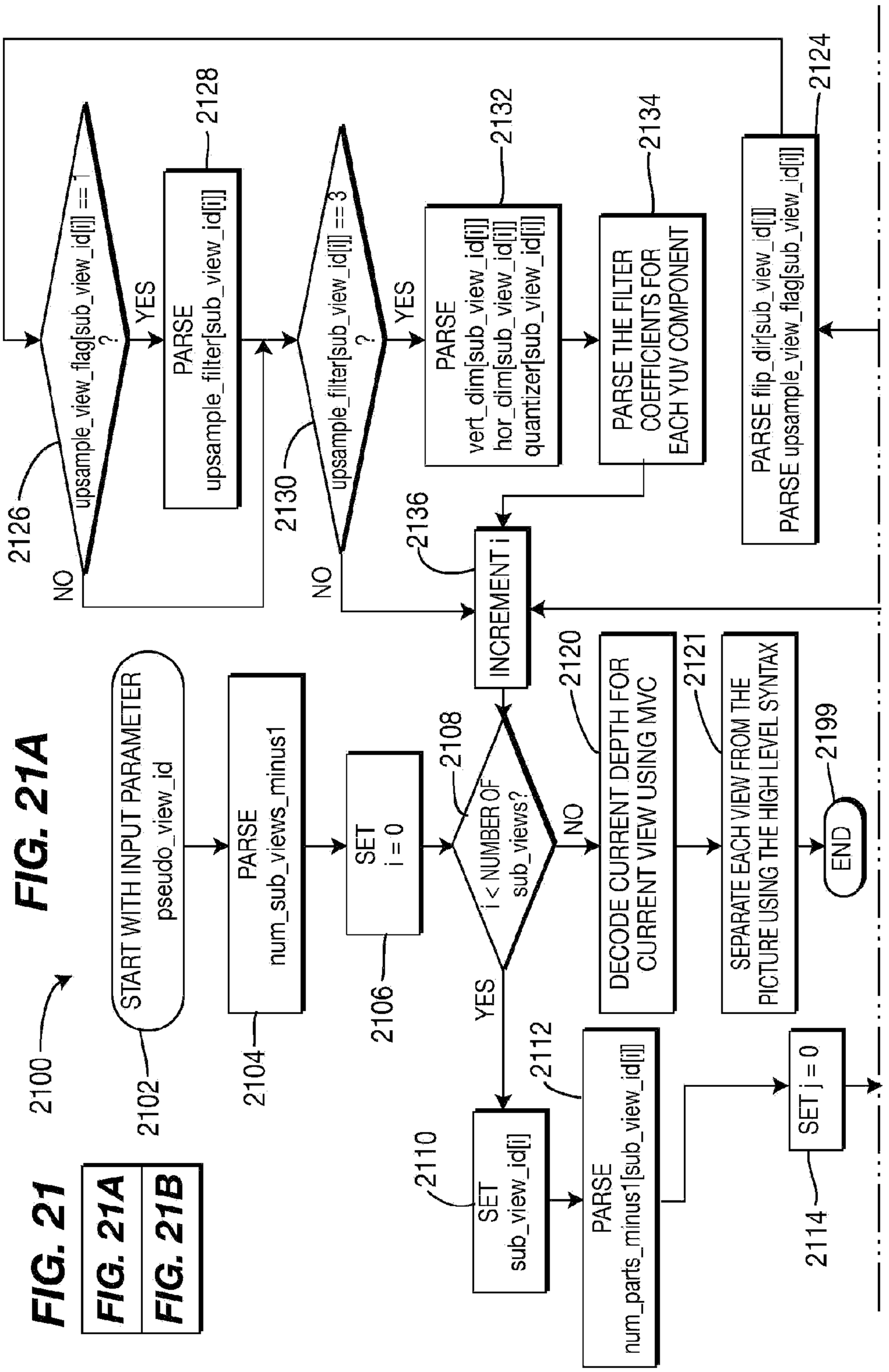

2000
2005
START
2015
PARSE num_coded_views_minus1
2020
PARSE view_id[i] FOR ALL (num_coded_views_minus1 + 1) DEPTHS CORRESPONDING TO view_id[i]
2025
PARSE THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR ANCHOR DEPTH PICTURES
2030
PARSE THE INTER-VIEW REFERENCE DEPENDENCY INFORMATION FOR NON-ANCHOR DEPTH PICTURES
2035
PARSE pseudo_view_present_flag
2040
pseudo_view_present IS TRUE ?
NO
YES
2045
PARSE tiling_mode, org_pic_width_in_mbs_minus1 AND org_pic_height_in_mbs_minus1
2050
FOR EACH CODED VIEW, CALL pseudo_view_info(view_id)
2099
END

*FIG. 20*

FIG. 21

| FIG. 21A |
| --- |
| FIG. 21B |



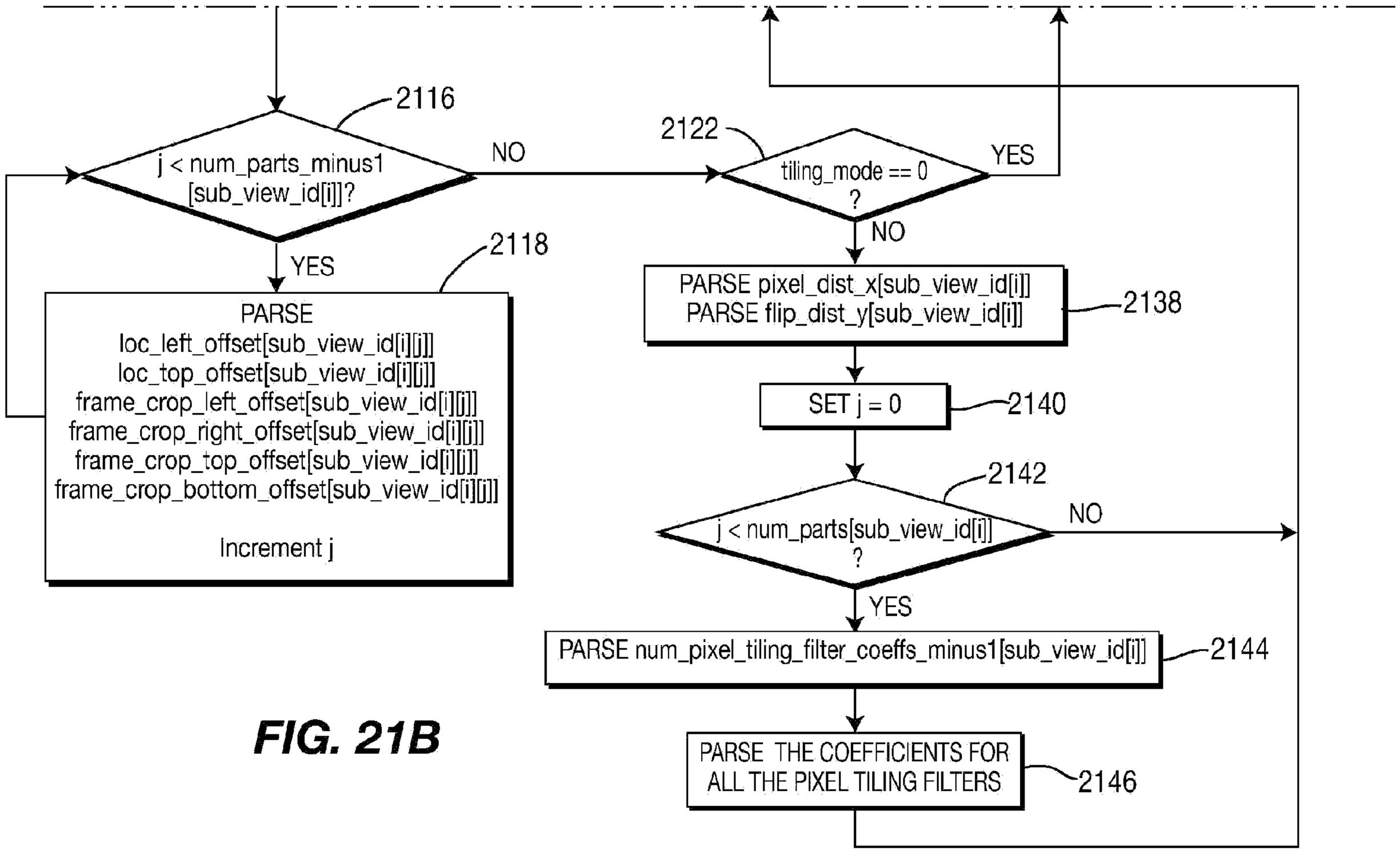



*FIG. 21B*

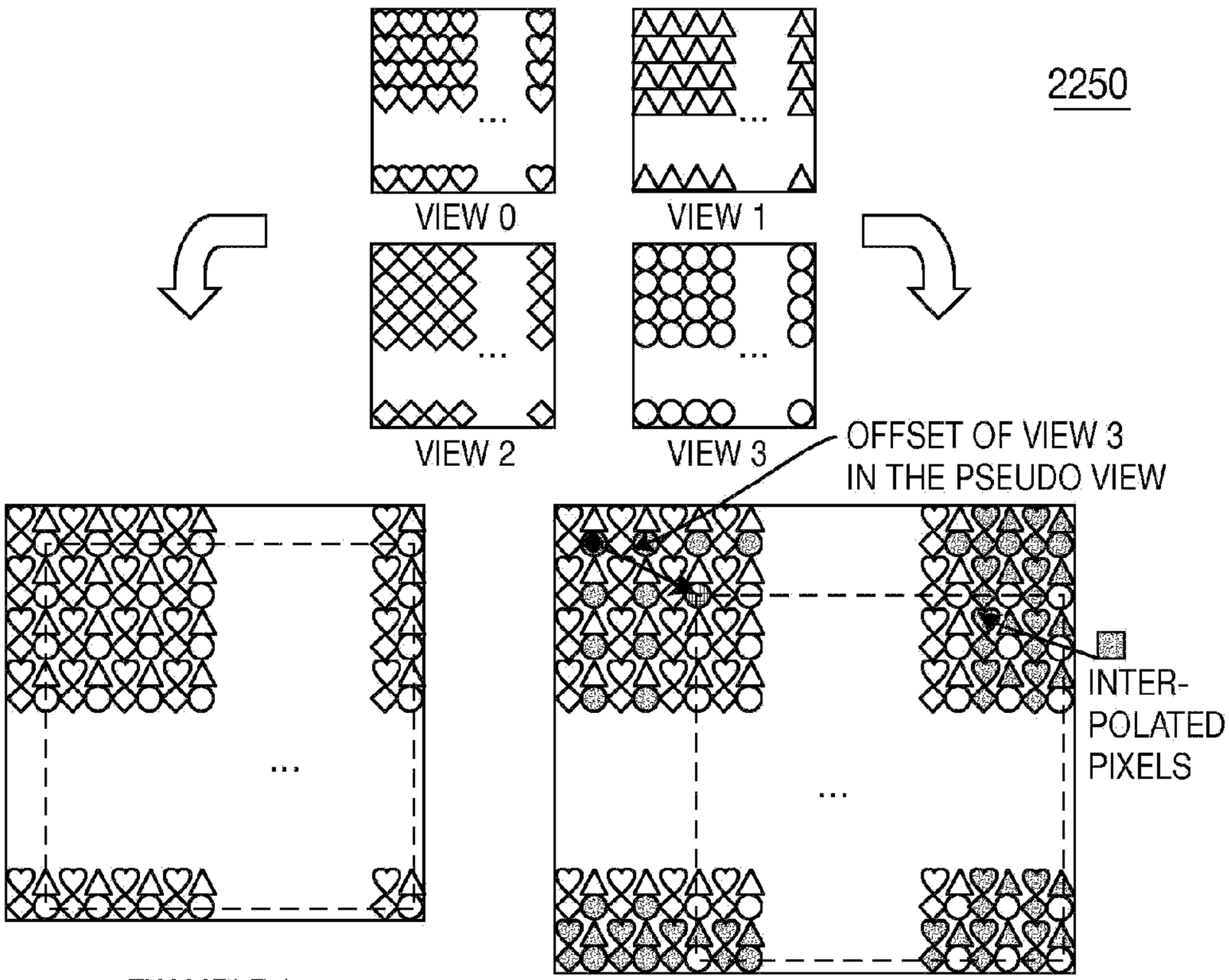

2250
VIEW 0
VIEW 1
VIEW 2
VIEW 3
OFFSET OF VIEW 3
IN THE PSEUDO VIEW
INTER-
POLATED
PIXELS
EXAMPLE 1:
TILING AT PIXEL LEVEL
2210
EXAMPLE 2:
TILING AT PIXEL LEVEL
2220

FIG. 22

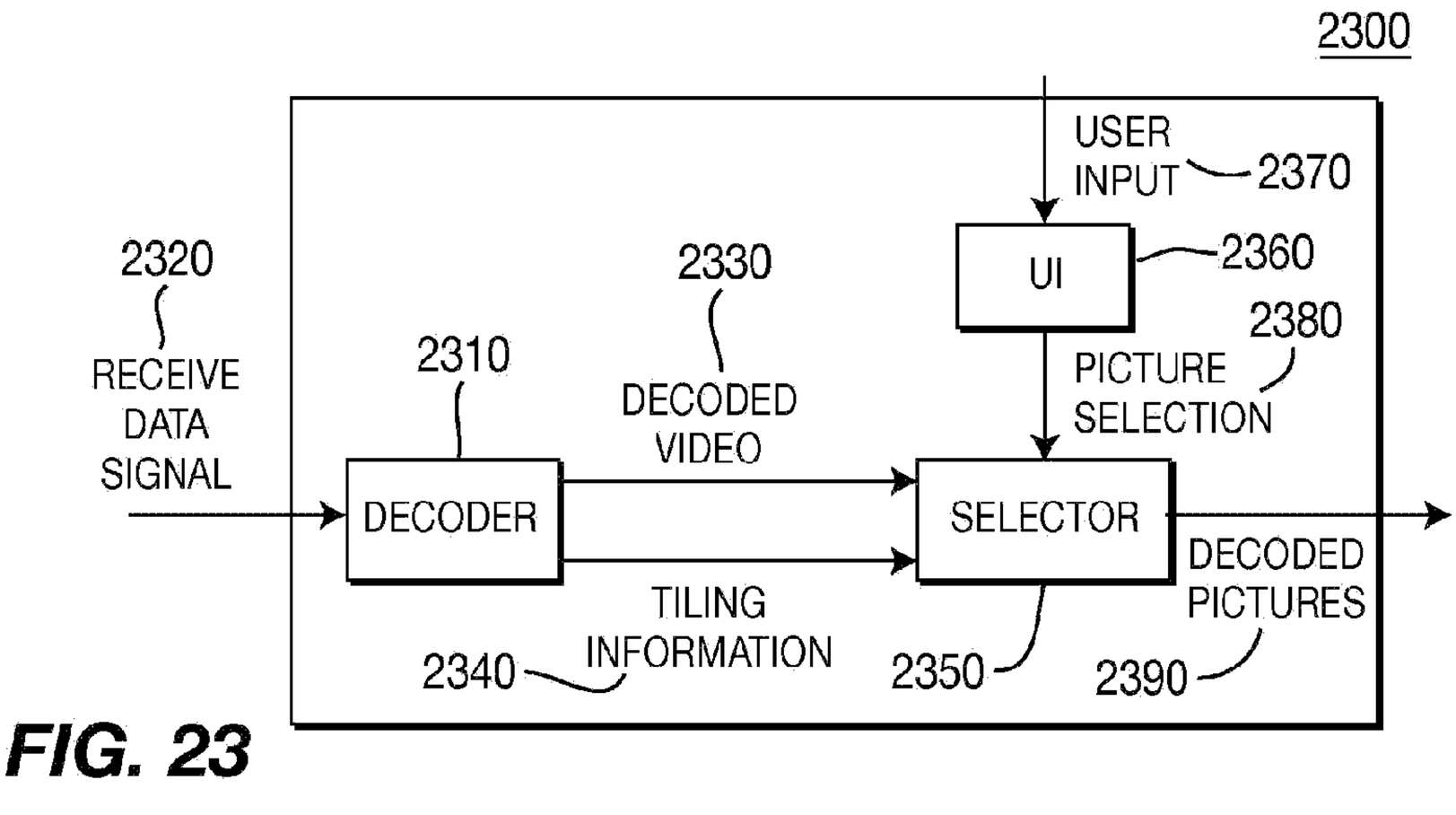

2300
USER
INPUT
2370
2320
RECEIVE
DATA
SIGNAL
2330
DECODED
VIDEO
2310
DECODER
UI
2360
2380
PICTURE
SELECTION
SELECTOR
DECODED
PICTURES
TILING
INFORMATION
2340
2350
2390

FIG. 23