



(51) International Patent Classification:

G06T 9/20 (2006.01) H04N 19/593 (2014.01)
H04N 19/436 (2014.01)

(21) International Application Number:

PCT/US2023/018747

(22) International Filing Date:

15 April 2023 (15.04.2023)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

17/721,687 15 April 2022 (15.04.2022) US

(71) Applicant: **META PLATFORMS TECHNOLOGIES, LLC** [US/US]; 1601 Willow Rd., Menlo Park, California 94025 (US).

(72) Inventors: **PINTO, Sonal**; 1601 Willow Road, Menlo Park, California 94025 (US). **GOSWAMI, Nilanjan**; 1601 Willow Road, Menlo Park, California 94025 (US).

(74) Agent: **COLBY, Steven et al.**; Rimon Law, 420 West Main Street, Suite 101B, Boise, Idaho 83702 (US).

(81) Designated States (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM,

DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— with international search report (Art. 21(3))

(54) Title: LOW-POWER HIGH THROUGHPUT HARDWARE DECODER WITH RANDOM BLOCK ACCESS

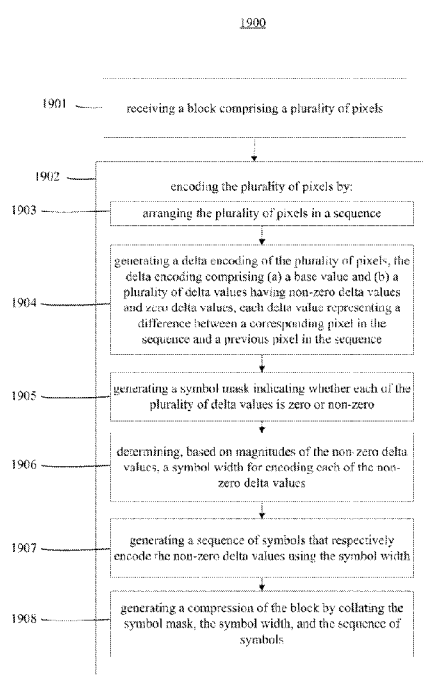


FIG. 19

(57) Abstract: A method includes receiving a block comprising pixels; encoding the pixels by: arranging the pixels in a sequence; generating a delta encoding of the pixels, the delta encoding comprising (a) a base value and (b) delta values having non-zero delta values and zero delta values, each delta value representing a difference between a corresponding pixel in the sequence and a previous pixel in the sequence; generating a symbol mask indicating whether each of the delta values is zero or non-zero; determining, based on magnitudes of the non-zero delta values, a symbol width for encoding each of the non-zero delta values; generating a sequence of symbols that respectively encode the non-zero delta values using the symbol width; generating a compression of the block by collating the symbol mask, the symbol width, and the sequence of symbols.

1 Low-power High Throughput Hardware Decoder with Random Block Access

2 **TECHNICAL FIELD**

3 [1] This disclosure generally relates to a hardware architecture of a processor unit for
4 rendering 2D content.

5 **BACKGROUND**

6 [2] Text is a crucial component of 3-D environments and virtual worlds for user
7 interfaces and wayfinding. Implementing text using standard antialiased texture mapping
8 leads to blurry and illegible writing which hinders usability and navigation. While
9 supersampling removes some of these artifacts, distracting artifacts can still impede legibility,
10 especially for recent high-resolution head-mounted displays. There is a need for an analytic
11 antialiasing technique that efficiently computes the coverage of text glyphs, over pixel
12 footprints, designed to run at real-time rates and an ability to decomposes glyphs into
13 piecewise-biquadratics and trapezoids that can be quickly area-integrated over a pixel
14 footprint to provide crisp legible antialiased text, even when mapped onto an arbitrary surface
15 in a 3-D virtual environment.

16 [3] The present disclosure seeks to address, at least in part, any or all of the
17 drawbacks and disadvantages described above.

18 **SUMMARY**

19 [4] According to a first aspect of the present disclosure there is provided a method
20 comprising, by a computing system: receiving a block comprising a plurality of pixels;
21 encoding the plurality of pixels by: arranging the plurality of pixels in a sequence; generating
22 a delta encoding of the plurality of pixels, the delta encoding comprising (a) a base value and
23 (b) a plurality of delta values having non-zero delta values and zero delta values, each delta
24 value representing a difference between a corresponding pixel in the sequence and a previous
25 pixel in the sequence; generating a symbol mask indicating whether each of the plurality of
26 delta values is zero or non-zero; determining, based on magnitudes of the non-zero delta
27 values, a symbol width for encoding each of the non-zero delta values; generating a sequence
28 of symbols that respectively encode the non-zero delta values using the symbol width;
29 generating a compression of the block by collating the symbol mask, the symbol width, and
30 the sequence of symbols.

31 [5] In some embodiments, the plurality of pixels of the block may correspond to a
32 single channel of pixel values amongst a plurality channels of pixel values.

33 [6] In some embodiments, arranging the plurality of pixels in a sequence may be

1 based on Morton Order.

2 [7] In some embodiments, the method may further comprise: determining that a bit
3 length associated with the compression of the block is more than a bit length associated with
4 the plurality of the pixels of the block as uncompressed; and storing the plurality of pixels of
5 the block as uncompressed.

6 [8] In some embodiments, the symbol width for encoding each of the non-zero delta
7 values may include an offset.

8 [9] In some embodiments, the method may further comprise modifying the
9 compression of the block to be byte-aligned by adding additional bits to the compression of
10 the block.

11 [10] In some embodiments, the method may further comprise decoding the
12 compression of the block by processing two or more delta values of the plurality of delta
13 values in parallel.

14 [11] According to a second aspect of the present disclosure there is provided one or
15 more computer-readable non-transitory storage media including instructions that, when
16 executed by one or more processors, are configured to cause the one or more processors to:
17 receive a block comprising a plurality of pixels; encode the plurality of pixels by: arranging
18 the plurality of pixels in a sequence; generating a delta encoding of the plurality of pixels, the
19 delta encoding comprising (a) a base value and (b) a plurality of delta values having non-zero
20 delta values and zero delta values, each delta value representing a difference between a
21 corresponding pixel in the sequence and a previous pixel in the sequence; generating a
22 symbol mask indicating whether each of the plurality of delta values is zero or non-zero;
23 determining, based on magnitudes of the non-zero delta values, a symbol width for encoding
24 each of the non-zero delta values; generating a sequence of symbols that respectively encode
25 the non-zero delta values using the symbol width; generating a compression of the block by
26 collating the symbol mask, the symbol width, and the sequence of symbols.

27 [12] In some embodiments, the plurality of pixels of the block may correspond to a
28 single channel of pixel values amongst a plurality channels of pixel values.

29 [13] In some embodiments, arranging the plurality of pixels in a sequence may be
30 based on Morton Order.

31 [14] In some embodiments, the instructions may be configured to further cause the
32 one or more processors to: determine that a bit length associated with the compression of the
33 block may be more than a bit length associated with the plurality of the pixels of the block as

1 uncompressed; and store the plurality of pixels of the block as uncompressed.

2 **[15]** In some embodiments, the symbol width for encoding each of the non-zero
3 delta values may include an offset.

4 **[16]** In some embodiments, the instructions may be configured to further cause the
5 one or more processors to: modify the compression of the block to be byte-aligned by adding
6 additional bits to the compression of the block.

7 **[17]** In some embodiments, the instructions may be configured to further cause the
8 one or more processors to: decode the compression of the block by processing two or more
9 delta values of the plurality of delta values in parallel.

10 **[18]** According to a third aspect of the present disclosure there is provided a system
11 comprising: one or more processors; and one or more computer-readable non-transitory
12 storage media in communication with the one or more processors, the one or more computer-
13 readable non-transitory storage media comprising instructions that when executed by the one
14 or more processors, cause the system to: receive a block comprising a plurality of pixels;
15 encode the plurality of pixels by: arranging the plurality of pixels in a sequence;
16 generating a delta encoding of the plurality of pixels, the delta encoding comprising (a) a base
17 value and (b) a plurality of delta values having non-zero delta values and zero delta values,
18 each delta value representing a difference between a corresponding pixel in the sequence and
19 a previous pixel in the sequence; generating a symbol mask indicating whether each of the
20 plurality of delta values is zero or non-zero; determining, based on magnitudes of the non-
21 zero delta values, a symbol width for encoding each of the non-zero delta values; generating a
22 sequence of symbols that respectively encode the non-zero delta values using the symbol
23 width; generating a compression of the block by collating the symbol mask, the symbol
24 width, and the sequence of symbols.

25 **[19]** In some embodiments, the plurality of pixels of the block may correspond to a
26 single channel of pixel values amongst a plurality channels of pixel values.

27 **[20]** In some embodiments, the instructions, when executed by the one or more
28 processors, may further cause the system to: determine that a bit length associated with the
29 compression of the block is more than a bit length associated with the plurality of the pixels
30 of the block as uncompressed; and store the plurality of pixels of the block as uncompressed.

31 **[21]** In some embodiments, the symbol width for encoding each of the non-zero
32 delta values may include an offset.

33 **[22]** In some embodiments, the instructions, when executed by the one or more

1 processors, further cause the system to: modify the compression of the block to be byte-
2 aligned by adding additional bits to the compression of the block.

3 [23] In some embodiments, the instructions, when executed by the one or more
4 processors, may further cause the system to: decode the compression of the block by
5 processing two or more delta values of the plurality of delta values in parallel. It will be
6 appreciated that any features described herein as being suitable for incorporation into one or
7 more aspects or embodiments of the present disclosure are intended to be generalizable
8 across any and all aspects and embodiments of the present disclosure. Other aspects of the
9 present disclosure can be understood by those skilled in the art in light of the description, the
10 claims, and the drawings of the present disclosure. The foregoing general description and the
11 following detailed description are exemplary and explanatory only and are not restrictive of
12 the claims.

13 BRIEF DESCRIPTION OF THE DRAWINGS

14 [24] FIG. 1A illustrates an example diagram of a 2D graphics system according to
15 one or more embodiments of the present disclosure.

16 [25] FIG. 1B illustrates an example of a graphics engine according to one or more
17 embodiments of the present disclosure.

18 [26] FIG. 2 illustrates an example of sampling and anti-aliasing techniques
19 according to one or more embodiments of the present disclosure.

20 [27] FIG. 3A illustrates an example 2D scene according to one or more
21 embodiments of the present disclosure.

22 [28] FIG. 3B illustrates an example of 2D content broken down into individual
23 primitives according to one or more embodiments of the present disclosure.

24 [29] FIGS. 4A-4B illustrate an example technique of determining whether a pixel
25 intersects with an edge of a trapezoid according to one or more embodiments of the present
26 disclosure.

27 [30] FIG. 5A illustrates an example quadratic curve primitive according to one or
28 more embodiments of the present disclosure.

29 [31] FIG. 5B illustrates an example tile comprising a curved edge of a quadratic
30 curve according to one or more embodiments of the present disclosure.

31 [32] FIG. 6 illustrates an example frame with two primitives, a destination
32 primitive and a source primitive according to one or more embodiments of the present
33 disclosure.

1 [33] **FIG. 7** illustrates an example encoding pipeline according to one or more
2 embodiments of the present disclosure.

3 [34] **FIG. 8** illustrates a tile that is segmented into multiple blocks according to one
4 or more embodiments of the present disclosure.

5 [35] **FIG. 9** illustrates an example encoding pipeline within a block encoder
6 according to one or more embodiments of the present disclosure.

7 [36] **FIG. 10** illustrates an example of the techniques of a spatial predictor
8 according to one or more embodiments of the present disclosure.

9 [37] **FIG. 11** illustrates an example a compressed channel of texel values according
10 to one or more embodiments of the present disclosure.

11 [38] **FIG. 12** illustrates an example diagram for encoding a 4x4 texel block using a
12 variable-length technique according to one or more embodiments of the present disclosure.

13 [39] **FIG. 13** illustrates an example technique of decoding a 4x4 texel block that
14 has been encoded by a block encoder according to one or more embodiments of the present
15 disclosure.

16 [40] **FIG. 14** illustrates an example method for determining the color information
17 of primitives in an image base in part by determining the coverage weight of each pixel in the
18 image according to one or more embodiments of the present disclosure.

19 [41] **FIG. 15** illustrates an example method for determining the color information
20 of a primitive base in part by determining the coverage weight of each pixel of primitive
21 based on function equations representing the edges of the primitives according to one or more
22 embodiments of the present disclosure.

23 [42] **FIG. 16** illustrates an example method for blending source shape with a
24 destination shape using a blending mode that requires updates to pixels in the color buffer
25 uncovered by the source shape according to one or more embodiments of the present
26 disclosure.

27 [43] **FIG. 17** illustrates an example method for encoding blocks of pixels based on
28 a tag that is used to temporary represent block headers according to one or more
29 embodiments of the present disclosure.

30 [44] **FIG. 18** illustrates an example method for determining whether a block of
31 pixels is different from previously-compressed blocks and compressing the block using a
32 variable-length technique according to one or more embodiments of the present disclosure.

33 [45] **FIG. 19** illustrates an example method for encoding a plurality of pixels based

1 on delta encoding that utilizes a base value, symbol mask, symbol width, and sequence of
2 symbols according to one or more embodiments of the present disclosure.

3 [46] FIG. 20 illustrates an example network environment according to one or more
4 embodiments of the present disclosure.

5 [47] FIG. 21 illustrates an example computer system according to one or more
6 embodiments of the present disclosure.

7 SUMMARY OF PARTICULAR EMBODIMENTS

8 [48] This invention is directed to an architecture of a 2D graphics engine (e.g.,
9 graphics processing unit, GPU) that is configured to render high-quality graphics while
10 operating on an ultra-low power budget. Particular embodiments disclosed herein provide an
11 improved technique for anti-aliasing. Anti-aliasing could be done in a variety of ways.
12 Traditionally, anti-aliasing is achieved using Multi-Sample Anti-Aliasing (MSAA), which
13 samples multiple points within a pixel area to determine what color the pixel should display.
14 A more accurate anti-aliasing could be achieved with more sampling points, but sampling is
15 computationally expensive. Instead of sampling, this invention converts 2D content
16 definitions into primitive shapes (e.g., 2D horizontally-aligned trapezoids and quadratic
17 curves) and leverages the known geometric properties of the primitives to perform analytic
18 anti-aliasing (e.g., instead of sampling a pixel at multiple points, embodiments disclosed
19 herein use geometry to compute how pixels/tiles are covered by the primitives). For example,
20 the technique involves calculating the amount of pixel that is covered by a primitive (e.g.,
21 11% of the pixel is covered by a trapezoid), then rendering the pixel shading based on
22 thereof. This technique allows the rendering of high-quality images at low power.

23 [49] In particular embodiments, a graphics engine performs anti-aliasing tile-by-
24 tile. A scene may be broken down into individual tiles, each tile comprising a fixed number
25 of pixels such as 32x32 pixels. For each tile, a “shape walker” component of the graphics
26 engine determines evaluates the pixels within the tile and determines whether the pixels are
27 completely inside, completely outside, or partially inside and partially outside a primitive that
28 is covered in the tile. Pixels that are completely inside or completely outside the primitive do
29 not need anti-aliasing, whereas pixels that intersect or overlap with an edge of the primitive
30 (e.g., outer frame of the primitive) would need to be sent to the “integrator,” where more fine-
31 grained pixel-level analytic anti-aliasing is needed. Particular embodiments disclosed herein
32 provide a novel technique for achieving such tasks.

33 [50] In particular embodiments, 2D scene that is to be rendered is divided into tiles,

1 each tile having a pre-determined number of pixels (e.g., 16x16). Text and 2D content within
2 the scene is defined as paths or contours, which is then converted into shapes of axis-aligned
3 trapezoid or piecewise-biquadratic (simply quadratics) curves. These shapes are referenced as
4 primitives. Then, for each tile, XRU-2D identifies the smallest bounding box within a tile that
5 encompasses a portion of a primitive covered by the tile. Each row of pixels within the
6 bounding box is then traversed row-by-row to determine pixels that overlap with the outer
7 shape of the primitive. Once the overlapping pixels are identified, pixels that do not need to
8 be anti-aliased are identified. The overlapping pixels are then sent to the integrator, while
9 other pixels (pixels falling outside the primitive or fully inside the primitive) are assigned 0
10 and 1 weight values, respectively, and sent to a different process (not to an integrator).
11 Subsequent steps involve, the integrator figuring out the coverage weight of the overlapping
12 pixels against the primitive, which may be used to determine the pixel shading for anti-
13 aliasing.

14 [51] In particular embodiments, the technique of identifying the overlapping pixels
15 discussed above involves one of two variations depending on whether the primitive is an
16 axis-aligned trapezoid or a piecewise-biquadratic (simply quadratics) curve. If the primitive is
17 a trapezoid, the method involves identifying the maximum and minimum Y values of the
18 trapezoid (e.g., top and bottom size of the trapezoid) and Y-intercepts and slope of an edge
19 (or both edges if two sides of the trapezoid fits into one tile). Then, the method continues by
20 traversing row-by-row to identify, based on the slopes and Y-intercepts identified in the
21 previous step, pixels that are overlapping with the shape of the trapezoid. Then, the
22 overlapping pixels are sent to the integrator, and pixels that fall outside of the trapezoid are
23 assigned weight 0 and pixels that fall inside the trapezoid are assigned weight 1. If the
24 primitive is a curve, the same high-level steps of identifying overlapping pixels and applying
25 weights to non-overlapping pixels are used, but in contrast to if the primitive is a trapezoid,
26 quadratic formula is used to represent the curve rather than using Y-intercepts and slope

27 [52] Particular embodiments of a graphics system focus on techniques for blending
28 multiple overlapping primitives in the color buffer. To remove as much processing work as
29 possible, the XRU-2D architecture utilizes a “culling” technique where tiles (e.g., 16x16
30 pixels) of a frame are culled using a smallest bounding box that encompasses a primitive
31 being processed by the hardware. Then, only the tiles covered by the bounding box are sent
32 down the rendering pipeline such that the empty tiles outside the bounding box are ignored
33 by the system. This reduces the overall computing required for rendering makes the system

1 more power-efficient. Incorporating this culling technique, however, presents an issue
2 because it conflicts with some of the special blending modes used for overlapping primitives.
3 Specifically, such special blending modes require access to both (1) the tiles covering the
4 destination primitive (tiles already in the color buffer) but not the source primitive and (2) the
5 tiles covering the source primitive (tiles that are to be written into the color buffer). However,
6 given the culling technique, when processing the source primitive, the system only has access
7 to the tiles of the source primitive. Embodiments disclosed herein provide a solution for these
8 issues.

9 [53] In particular embodiments, the task of clearing a destination primitive involves
10 defining the tiles in a frame as “non-empty tiles” when the tiles cover a source primitive and
11 as “empty tiles” when the tiles do not cover the source primitive. For empty tiles, the task of
12 clearing a destination primitive may involve a graphics system instructing a color buffer to
13 bypass the primitive cull to allow the color buffer to gain access to the empty tiles. For non-
14 empty tiles, the clearing task is a bit more complicated since only the destination primitive
15 must be cleared from the non-empty tiles, thus the embodiments disclose a technique that
16 involves a pixel-by-pixel analysis for each of the non-empty tiles that tracks the blending
17 modes that are used to update each pixel. This allows the system to identify pixels in the non-
18 empty tiles that are covering only the destination primitive and not the source primitive.

19 [54] Particular embodiments herein focus on encoding static assets (e.g., a blit such
20 as an emoji). For optimal compression, an encoder system disclosed herein uses a spatial
21 predictor to find similar pixel blocks within a frame and assign them the same block header.
22 A naïve implementation of the spatial predictor is to (1) compare the current block to a
23 previously processed block pixel-by-pixel and (2) copy the header of the matching block into
24 the current block, such that when the current block is decoded, the decoder can retrieve the
25 data from the matching block. However, there is a problem with the naïve implementation
26 because pixel-by-pixel comparison requires a lot of power and memory. As a solution,
27 embodiments herein disclose a technique of generating a hash code to represent each pixel
28 block and uses the hash codes to compare blocks. There exists another problem with the
29 naïve implementation noted above. The encoding process involves several steps in a pipeline.
30 The step of comparing the blocks (e.g., comparing hash codes) occurs at the first step and the
31 step of assigning a header occurs at the last step, with several steps in between the first and
32 last steps. This means that when a particular block is compared to the one of the previous
33 blocks, the previous blocks may still be going through the encoding pipeline and headings for

1 these blocks may not have been generated yet. In circumstances where a header of a previous
2 block is not generated yet but the current block's hash code matches that of the previous
3 block, the current block is assigned a tag instead of a header. At the end of each cycle in the
4 pipeline (e.g., when a block is handed off to the next step in the encoding pipeline), the
5 system checks whether a previously unavailable header is available, and if available, replaces
6 the corresponding tag with the header. The system is configured in a way that a block's tag
7 may be updated with a header wherever the block is in the encoding pipeline. This solution
8 prevents the encoding pipeline from being stalled due to certain headers not being available at
9 the time a matching block is found.

10 [55] In particular embodiments, an encoder and the decoder reside in the graphics
11 engine, but the encoded pixel blocks may be stored in a memory external to the graphics
12 engine and accessed by the decoder from the external memory as necessary. Tiles that are
13 encoded by the graphics engine may be piped through a double buffer such that the encoder
14 can compress the current tile while the next tile is streamed in. For each tile to be encoded, a
15 block scheduler may separate the tile into blocks for the encoder, and the block schedule may
16 schedule the blocks in an arrangement called the "Morton Order" that is optimized for delta
17 coding (e.g., minimizing the distance between the blocks in a sequence). For each block that
18 is encoded, the pixels may be also be encoded based on the Morton Order arrangement. For
19 each block, the different pixel channels may be encoded separately then subsequently
20 collated into a single bitstream.

21 [56] Particular embodiments herein provide a technique of encoding pixel blocks of
22 an image that allows the system to selectively retrieve and decode any particular pixel block
23 of the encoded blocks, e.g., independently from other encoded pixel blocks. More
24 specifically, each block may be encoded in a way that it is self-contained, meaning that the
25 system can selectively retrieve and decompress a particular block simply based on the data
26 contained within the block. For example, with a PNG image that encoded using the disclosed
27 technique, a system may be able to retrieve and decompress specific portions of the PNG
28 image (pixel blocks) independently from other portions of the image.

29 [57] In particular embodiments, the encoding system may analyze each block of
30 texels that are to be encoded and categorizes them into one of two block variants: Flatblock
31 or Codeblock. A block may be categorized as a Flatblock if all texels in the block have the
32 same value. Otherwise, a block may be categorized as a Codeblock, for example, if some of
33 the texels in the block have different values. Later in the encoding process, after a Codeblock

1 is encoded, the encoding system may evaluate the encoded block data of the Codeblock to see
2 whether the encoded data is greater in size than the uncompressed size of the block, that is,
3 whether the encoded data requires more bits than uncompressed data. If so, the encoding
4 system may (1) disregard the encoded block data of the Codeblock, (2) recategorize the
5 Codeblock as a third block variant, i.e., Rawblock, and (3) store the uncompressed block data
6 of the block in lieu of the disregarded encoded data. In an embodiment, the categorization
7 steps described above are implemented for an entire block, that is, without regard to the
8 various channels of texels that the block may have. For example, a block comprising multiple
9 channels of texels may be categorized as a Flatblock only if all texels within the block have
10 the same value, including texels of different channels. Alternatively, if any texel values differ
11 in a block, even across different channels, the block may be categorized as a Codeblock.

12 [58] In particular embodiments, the encoding system may compress each of the
13 three block variants with different techniques. The encoding system compresses a Flatblock
14 using a single texel value such that the encoded data of the Flatblock is the size of a single
15 texel value. The encoding system may compress a Codeblock a single channel at a time, and
16 for each channel, a different encoding technique may be utilized. A particular channel of a
17 Codeblock may be encoded using a “flat” technique if all of the values of the texels in the
18 channel are the same. The flat technique involves using a single value to represent the entire
19 channel. If, for a particular channel in a block, the values of the texels differ, then a “variable-
20 length” technique may be used. The variable-length technique is a novel compression
21 technique that produces different sizes of encoded data depending on the differences in the
22 texel values within the block. After encoding a channel using the variable-length technique,
23 the system may evaluate whether the encoded channel data is computationally more
24 expensive than the uncompressed channel data, that is, whether the encoded data requires
25 more bits than the uncompressed data. If so, the encoding system may disregard the encoded
26 channel data and store the channel data as uncompressed. In particular embodiments, each
27 texel channel within a Codeblock may be independently encoded using any of the techniques
28 described above. For example, for a Codeblock having three channels of texel values, a first
29 channel of the three may be encoded using a flat technique, a second channel of the three may
30 be encoded using the variable-length technique, and a third channel of the three may be
31 stored as uncompressed. Each of the encoded (or uncompressed) channels in a Codeblock
32 may be collated together to form the compressed block data for the Codeblock. If, however,
33 the compressed block data for a Codeblock turns out to be computationally more expensive

1 than the uncompressed data of the Codeblock, then the compressed block data may be
2 disregarded and the Codeblock may be recategorized as a Rawblock. The encoding system
3 stores a Rawblock without any compression. The size of a Rawblock represents the
4 maximum size of a stored block.

5 [59] In particular embodiments, an encoding system disclosed herein may use a
6 technique referred to as the variable-length technique to encode a texel block and produces
7 different sizes of encoded data depending on the texel values within the block. This technique
8 encodes the texels channel-by-channel, meaning each texel component is separately encoded,
9 for example, each of R, G, B components may be separately encoded. The variable-length
10 technique involves using three groups of data to represent the uncompressed texel values:
11 “symbolmask”; “rbits”; “rsymbols.” Data group rsymbols is used to represent the delta values
12 of the texel values of a block, as arranged in a particular way, for example, in an arrangement
13 called the “Morton Order,” which is optimized for delta coding (e.g., minimizing the distance
14 between the blocks in a sequence). Data group symbolmask is used to provide a 1 to 1
15 mapping of the delta values and indicates whether each delta value is a zero value or non-zero
16 value. Data group rbits is used to indicate the maximum number of bits required to represent
17 each of the delta values, the maximum number including an additional bit to indicate whether
18 the delta values are positive or negative values.

19 [60] As an example of the three group of data noted above, consider texel values of
20 a block that are arranged as [0, 4, 1, 1], in which case the delta values for that sequence of
21 values would be [4, -3, 0]. The symbolmask for delta values [4, -3, 0] would be [1, 1, 0],
22 which simply represents which of the delta values are zeros and non-zeros. The number of
23 bits of the encoded data that are assigned to symbolmask equals the number of delta values,
24 or the number of uncompressed texel values minus one. The rbits for delta values [4, -3, 0]
25 would be four bits—a first bit to indicate the positive or negative sign of the delta values and
26 three additional bits to represent each of the delta values which has a maximum value of four.
27 In an embodiment, the number of bits represented by rbits may be stored in the encoded data
28 in a binary representation, such that rbits of 4 may be stored as [100]. In some embodiments,
29 rbits may be stored with an offset, for example, with an offset of 2 such that rbits of 4 bits
30 may be stored as 2, or [010]. In some embodiments, the number of bits in the encoded data
31 that are assigned for rbits may be fixed, for example, to three bits. In embodiments where
32 three bits of the encoded data are assigned to rbits (e.g., [###]) and rbits are stored with an
33 offset of 2, the range of binary values that rbits can represent varies from 2 to 9. The

1 rsymbols for delta values [4, -3, 0] may be generated as [0100, 1011], the first of which
2 represents a positive indicator ("0") followed by a binary representation of 4 ("100") and the
3 second of which represents a negative indicator ("1") followed by a binary representation of 3
4 ("011"). Notably, the data group rsymbols only needs to represent non-zero delta values since
5 any zero delta values are already indicated by symbolmask. Once all three groups of data are
6 generated to represent the texel values of a block, they may be collated together into a stream
7 of bits. Continuing the example above, the three groups of data may be collated as [rsymbols,
8 rbits, symbolmask], or [0100 1011, 010, 1 1 0]. In addition to the three groups of data, the
9 first value of the uncompressed texel values may be encoded as the "base value" of the
10 encoded data, either encoded together with the three groups of data or separately as metadata.
11 In the example above, the base value would be 0 since that is the first value of the
12 uncompressed texel values [0, 4, 1, 1].

13 [61] In particular embodiments, a decoder disclosed herein may be configured to
14 decode multiple delta values from the uncompressed data per one decoding cycle. For
15 example, if a decoder system is configured to decode three delta values per cycle, each cycle
16 may involve fetching the first three entries from symbolmask to decode the three delta values
17 in parallel. For any delta values that the symbolmask indicates as being a zero value, no
18 additional process may be required to determine that delta value since the value is zero. For
19 any delta values that the symbolmask indicates as a non-zero value, the decoder system may
20 fetch from rsymbols a number of bits indicated by rbits from the appropriate portion of
21 rsymbols. For example, if rbits is 5, the decoding system would fetch the first 5 bits for the
22 first non-zero delta value. The decoding system may be configured to maintain a pointer or
23 other indicators that indicates which portion of the rsymbols should be fetched next to allow
24 the decoder system to determine which portion of the rsymbols should be fetched for the next
25 non-zero delta value. After the decoder system determines the value of each of the first three
26 delta values, the delta values may be further decoded based on the base value of the
27 uncompressed data. For example, the first uncompressed data value can be determined as
28 equating to the base value and the next uncompressed value can be determined by adding the
29 first delta value to the base value, then the combined value can be used to determine the
30 subsequent uncompressed value by adding the subsequent delta value to it.

31 [62] The embodiments disclosed herein are only examples, and the scope of this
32 disclosure is not limited to them. Particular embodiments may include all, some, or none of
33 the components, elements, features, functions, operations, or steps of the embodiments

disclosed above. Embodiments according to the invention are in particular disclosed in the attached claims directed to a method, a storage medium, a system and a computer program product, wherein any feature mentioned in one claim category, e.g. method, can be claimed in another claim category, e.g. system, as well. The dependencies or references back in the attached claims are chosen for formal reasons only. However, any subject matter resulting from a deliberate reference back to any previous claims (in particular multiple dependencies) can be claimed as well, so that any combination of claims and the features thereof are disclosed and can be claimed regardless of the dependencies chosen in the attached claims. The subject-matter which can be claimed comprises not only the combinations of features as set out in the attached claims but also any other combination of features in the claims, wherein each feature mentioned in the claims can be combined with any other feature or combination of other features in the claims. Furthermore, any of the embodiments and features described or depicted herein can be claimed in a separate claim and/or in any combination with any embodiment or feature described or depicted herein or with any of the features of the attached claims.

DESCRIPTION OF EXAMPLE EMBODIMENTS

[63] FIG. 1A illustrates an example diagram of a 2D graphics system according to embodiments disclosed herein. Such embodiments may include an application 101 that provides scene details, a driver 102 that converts paths within a scene into shapes that can be more efficiently processed (referred herein as “primitives”), a 2D graphics engine 103 for rendering a scene, and a display 198 for displaying the rendered scene. A 2D graphics engine 103 may be referred herein as a graphics engine, graphics system, GPU, or simply a “system” for brevity.

[64] In particular embodiments, a driver 102 may be configured to decompose a scene received from an application 101 into individual shapes that can be more efficiently processed by a 2D graphics engine 103, such shapes are referred herein as “primitives.” A scene may consist of a number of 2D content and texts. 2D content and texts contained in a scene may be defined by “paths,” where each path is made up of lines, curves, arcs, or otherwise referred herein as “contours.” In an embodiment, an application 101 defines the paths within a scene. For example without limitation, a typical scene may contain between 2,000-20,000 paths. In an embodiment, each contour may be required to be “closed,” such that the first and last vertices of the contour are identical (e.g., at the same location). In an embodiment, a driver 102 may be configured to process each path in a scene by converting

1 the contours of the path into one of two types of primitives: (1) horizontally aligned
2 trapezoids and (2) piecewise-biquadratic curves. A horizontally aligned trapezoid, referred
3 hereinafter as a “trapezoid” for brevity, comprises two parallel horizontal edges on the top
4 and bottom sides of the trapezoid and two side edges connecting the top and bottom sides. A
5 piecewise-biquadratic curve—referred hereinafter as a “quadratic curve” for brevity—is a 2-
6 D region bounded by a quadratic curve and a line. An example process of converting the
7 contours of a path into primitives is disclosed in the following paper: A. Ellis, W. Hunt, J.
8 Hart, Nerf: *Real-Time Analytic Antialiased Text for 3-D Environments*, Computer Graphics
9 forum, vol. 38, issue 8, November 2019, pp. 23-32.

10 [65] In particular embodiments, a driver 102 may be configured to perform tiling
11 operations by which a scene is segmented into a smaller data structure called a tile, or tile
12 block. Each tile may be composed of a set of pixels. For example, a tile may be comprised of
13 a 16-by-16 pixel block or a 32-by-32 pixel block. In an embodiment, a driver 102 may be
14 configured to determine, for each tile in a scene, every primitive that is covered the tile, then
15 store this information in a memory database 109 that is accessible by a graphics engine 103.

16 [66] While recognizing the differences of terms “pixels” and “texels” as used in the
17 field of art, any references to pixels herein may be interchangeable with references to texels
18 and any references to texels herein may be interchangeable with references to pixels, for the
19 purposes of describing the embodiments herein.

20 [67] FIG. 1B illustrates an example of a graphics engine 103. In particular
21 embodiments, a 2D graphics engine 103 may be configured to perform rendering operations
22 tile by tile or a single tile at a time. In other embodiments, a graphics engine 103 may
23 perform certain rendering operations multiple tiles at a time or in parallel. A command
24 controller 107 may be configured to arrange the tiles within a scene in a specific order and
25 provide instructions to a tile controller 120 to start rendering the tiles according to the tile
26 order. For example, a command controller 107 may be configured to implement a tile
27 walking function that iterates over the tile data structure to determine information about the
28 tile. Such tile information may include which tiles should be processed by the downstream
29 rendering components and in what order the tiles should be processed. The command
30 controller 107 may then provide the tile information to the rendering downstream
31 components, such as a tile controller 120. In an embodiment, a command controller 107 may
32 only identify the tiles that cover a primitive or a background, for example, tiles that are empty
33 may not be sent down the rendering pipeline for efficiency purposes. In an embodiment, a

1 command controller 107 may be configured to determine, for each tile containing at least a
2 portion of a primitive, a tile bounding box that encompasses the at least the portion of the
3 primitive within the tile. The tile bounding box information may then be sent down the
4 rendering pipeline to allow certain operations to focus only on the tile bounding box within a
5 tile rather than the entire tile. In an embodiment, the tile bounding box information may also
6 comprise data indicating which edges of a primitive are contained in the tile bounding box.
7 For example, if a tile contains a top left portion of a trapezoid, the bounding box information
8 may indicate that the tile contains the left and top edges of the trapezoid. In an embodiment, a
9 command controller 107 may be configured to generate a list of primitives that are contained
10 in each of a non-empty tile (a tile that is covering with one or more primitives), and this list
11 may be sent down the rendering pipeline. While memory database 109 is illustrated in FIG.
12 1B, the memory database 109 may be comprised of multiple memory databases, each
13 memory database being responsible for storing data that is unrelated to data stored in other
14 memory databases. In an embodiment, a command controller 107 may be configured to
15 determine, for each primitive, a primitive bounding box that encompasses the primitive
16 across one or more tiles in a frame (image). The primitive bounding box information may
17 then be sent down the rendering pipeline to allow certain operations to focus on the primitive
18 bounding box rather than the entire frame.

19 [68] In particular embodiments, once the command controller 107 provides
20 instructions to a tile controller 120 tiles to render, the tile controller 120 may be configured to
21 gather all the primitive, blit, and/or filter information necessary to render the tiles. For every
22 tile to be rendered, a tile controller 120 may begin the rendering process by fetching the tile
23 data from a tile memory database 109, for example, through the input box 106 shown in FIG.
24 1B. The tile data that is fetched by the tile controller 120 may be passed to downstream
25 components in the rendering pipeline (e.g., shape walker 130). A tile controller 120 may only
26 fetch non-empty tiles from the tile memory database 109. The fetch operation performed by a
27 tile controller 120 may be a single-step process and may involve fetching data associated
28 each primitive within the tiles, including all the vertices of the primitive and a portion of the
29 shader information associated with the primitive. The rest of the shader information may be
30 fetched by a shader 150. A tile controller 120 may also be configured to fetch blit and filter
31 render instructions from memory that is external to the graphics engine. After parsing through
32 the fetched data, a tile controller 120 may be configured to perform a tile bounding box
33 check. Then, a tile controller 120 may be configured to provide the shader information to a

1 shader 150 and the bilt and filter information to a bilt and filtering unit 180. In an
2 embodiment, a tile controller 120 may be configured to provide tile-done and commands-
3 done indicators to the color buffer 191, 193. This information represents the status with
4 respect to what is processed by a tile controller 120 and what is not.

5 [69] In particular embodiments, a shape walker 130 may be configured to
6 determine the coverage weight of each pixel within a tile, the coverage weight representing
7 how much of the pixel is covered by a primitive within the tile. In other words, a shape
8 walker 130 may be configured to examine each of the pixels in the tile (or within the tile
9 bounding box) to determine whether the pixels falls inside, outside, or partially intersects
10 with an edge of a primitive (e.g., trapezoid or a quadratic curve). Pixels that are determined to
11 be fully inside a primitive are given a coverage weight of 1, pixels determined to be that are
12 fully outside a primitive is given a coverage weight of 0, and pixels that are intersecting with
13 an edge of a primitive are sent to an integrator 140 for further processing (e.g., an integration
14 step). Partially interacting, or overlapping, pixels require an integration step to precisely
15 determine how much of the pixel overlaps with an edge of a primitive. This information is
16 used for anti-aliasing at a later step in the rendering pipeline. For pixels that are assigned
17 coverage weights of 0 or 1 by a shape walker 130, their respective coverage weights are
18 provided to coverage buffers 151 or 152.

19 [70] Traditional methods for anti-aliasing typically use sampling or Multi-Sample
20 Anti-Aliasing (MSAA), which involves sampling points within a pixel area to determine the
21 coverage weight for that pixel. For example, as illustrated in **FIG. 2**, to determine whether a
22 triangle 204 overlaps with the pixel area 201, a graphics system may take a sample at point
23 202. In this example, the system may determine that the pixel area 201 is not covered by
24 content 204 because, at sample point 202, the triangle 204 does not cover the pixel 201. The
25 system may then assign a coverage weight of 0 to the pixel 201 to indicate that pixel 201 is
26 not covered by any portion of the triangle. Alternatively, instead of taking a single sample
27 from pixel 201, multiple points of samples 203 may be taken. As shown in the top right
28 example in FIG. 2, taking four samples within pixel 201 allows the system to determine a
29 coverage weight of 0.5 or 50%, which is a more accurate coverage weight than 0. As shown
30 by these examples, traditional methods for anti-aliasing can determine coverage weights of
31 higher accuracy as more samples are taken, however, there is a trade-off for taking more
32 samples since each additional sample point requires additional computing power and/or
33 compute time. Moreover, when coverage weights are determined by a way of taking samples,

1 the resulting coverage weights are typically rough estimates and may only provide fixed
2 coverage weights. For example, if one sample is taken, the coverage weight for a pixel can
3 only be either 0 (not covered) or 1 (fully covered). If four samples are taken, the coverage
4 weight for a pixel can only be 0, 0.25, 0.5, 0.75 or 1. In contrast to such traditional methods,
5 embodiments disclosed herein allow the determination of coverage weights in a more
6 granular fashion (e.g., non-fixed coverage weights) and without any sampling. For example,
7 according to an embodiment illustrated in bottom half of FIG. 2, a graphics engine may be
8 able to utilize techniques disclosed herein to determine that 12% of the pixel 251 is covered
9 by a trapezoid 253, 23% of pixel 252 is covered, 100% of pixel 253 is covered, and 0% of
10 pixel 254 is covered.

11 [71] In particular embodiments, a scene may be broken down into smaller units of
12 pixels referred to as tiles. For example, FIG. 3A illustrates an example 2D scene 305 and also
13 the same scene 307 that is broken down into individual tiles. Each tile may comprise a fixed
14 number of pixels, such as 16-by-16 pixels or 32-by-32 pixels. In particular embodiments,
15 content within a scene may be broken down into smaller units referred herein as primitive
16 (e.g., a quadratic curve, trapezoid). FIG. 3B illustrates an example of 2D content 371 that
17 may be shown in a scene and broken down into individual primitives. Specifically, FIG. 3B
18 illustrates content 371 that is broken down into four quadratic curves 376 representing the
19 corners of content 371, two trapezoids 374 representing top and bottom portions of the
20 content 371, and one “trapezoid” 378 in the center portion of the content 371. While
21 trapezoid 378 may appear to be a rectangle rather than a trapezoid in the literal sense,
22 embodiments may be configured to consider trapezoid 378 as a trapezoid with side edges that
23 are vertically oriented (e.g., perpendicular from the top and bottom edges). In particular
24 embodiments, referring to FIG. 1A, an application 101 may be configured to break down
25 content into primitives and may provide the primitives and information about the primitives
26 to a driver 102.

27 [72] In an embodiment, a shape walker 130 may be configured to utilize an
28 algorithm known as DDA (digital differential analyzer) line generating algorithm to
29 determine whether a pixel intersects with an edge of a primitive. The technique of identifying
30 intersecting pixels may involve first determining a function equation that represents an edge
31 of a primitive (or otherwise referred as an “edge definition”), then utilizing an algorithm to
32 determine whether a pixel overlaps/intersects with the edge represented by the function
33 equation. For example, if a primitive is a trapezoid, a shape walker 130 may first determine

1 the maximum and minimum Y values of the trapezoid (e.g., top and bottom edge of the
2 trapezoid) and y-intercepts and slope of an edge (or both edges if two side edges of the
3 trapezoid fits into one tile). The y-intercepts and slope may be used to determine a function
4 equation (e.g., linear equation) that represents a corresponding edge. Then, the technique may
5 continue by traversing row-by-row of the tile to identify, based on the function equation
6 identified in the previous step, pixels that are intersecting with the edge of the trapezoid (e.g.,
7 the function equation). The intersecting pixels are sent to the integrator to determine the pixel
8 coverage weight, and pixels that fall completely outside of the trapezoid are assigned
9 coverage weight of 0 and pixels that fall completely inside the trapezoid are assigned
10 coverage weight of 1. On the other hand, if the primitive is a quadratic curve, the same high-
11 level technique of identifying the overlapping pixels and applying weights to non-
12 overlapping pixels are used, but in contrast to if the primitive is a trapezoid, a quadratic
13 formula is used to represent the curve rather than a linear equation.

14 [73] FIGS. 4A-4B illustrate an example technique of determining whether a pixel
15 intersects with an edge of a trapezoid. Specifically, FIG. 4A illustrates a trapezoid 402 that is
16 covering pixels of several tiles. In particular embodiments, a shape walker 130 processes a
17 particular primitive tile by tile. For example, a shape walker 130 may be configured to
18 process each of the numbered tiles in box 413 in the sequence of the illustrated numbers (e.g.,
19 tile 1, tile 2, ... tile 11). Such a sequence of the tiles may be determined by a command
20 controller 107 and provided to downstream components in the rendering pipeline such as a
21 shape walker 130. FIG. 4B illustrates the first tile 423 shown in FIG. 4A and further
22 illustrates the step of determining whether pixels within the tile 423 intersects with an edge of
23 the trapezoid 402. In particular embodiments, a shape walker 130 may receive the tile
24 bounding box information that outlines a box 435 that encompasses a primitive or a portion
25 of thereof. A shape walker 130 may be configured to process the pixels only within the
26 bounding box, rather than the entire tile. The tile bounding box information that a shape
27 walker 130 receives may also indicate which edges of primitives are contained within a tile.
28 For example, in the example shown in FIG. 4B, the tile bounding box information may
29 indicate that only the left and top edges of a trapezoid 402 are contained in tile 423. The tile
30 bounding box information may further indicate whether, for an edge within the tile, whether
31 the entirety of the edge is in the tile or only a portion of the edge is within the tile. For
32 example, in the example shown in FIG. 4B, the tile bounding information may indicate that
33 only a portion of the top edge and only a portion the left edge of a trapezoid 402 are

1 contained in tile 412.

2 [74] In particular embodiments, a shape walker 130 may be configured to analyze
3 each pixel position within a tile to determine whether the corresponding pixel
4 overlaps/intersects with an edge of a primitive (e.g., trapezoid or curve). A shape walker 130
5 may be configured to process, for each primitive within a tile, a single edge at a time. For
6 example, in the example shown in FIG. 4B, a shape walker 130 may be configured to process
7 the left edge 456 separately from the top edge. In particular embodiments, when a shape
8 walker 130 is processing one of the side edges of a trapezoid, the shape walker 130 may be
9 configured to determine y-min and y-max values of the primitive. For example, as shown in
10 FIG. 4B, the portion of a trapezoid shown within tile 423 comprises a y-min 453 representing
11 the top edge of the portion of the trapezoid and a y-min 451 representing the bottom portion
12 of the primitive that is within tile 423. Such y-min and y-max values may be determined by a
13 shape walker 130, for example, based on the bounding box information. A shape walker may
14 then determine y-intercepts and a slope of a side edge. For example, as shown in FIG. 4B, a
15 shape walker 130 may be configured to determine the y-intercepts and a slope of the edge
16 456 of the trapezoid. Using this information, a shape walker 130 may be configured to
17 determine a function equation based on a linear equation (e.g., $ax + b$) that defines a side edge
18 of a trapezoid. In a different circumstance where a tile includes only a right side edge of a
19 trapezoid, a function equation for the right edge may similarly be determined based on the y-
20 intercepts and a slope of the right edge. In yet another circumstance where a tile includes both
21 a left side edge and a right side edge, a shape walker 130 may also similarly determine the
22 function equation for both of the edges, but in separate operations.

23 [75] Once the y-min, y-max, and function equation is determined for a particular
24 edge contained in a tile, a shape walker 130 may be configured to traverse the tile row-by-
25 row (e.g., from y-min to y-max) and determine, for each row, pixels that intersect with an
26 edge of a trapezoid based on the function equation. For example, referring to FIG. 4B, a
27 shape walker 130 may be configured to traverse the tile 423 row by row, starting at y-min
28 453 and ending with y-max 451. A shape walker 130 may then determine, for each row, and
29 based on the function equation, the x-min and x-max values for that row (the x-min value
30 representing the leftmost position at which a pixel intersects with a side edge of a trapezoid
31 and the x-max value representing the rightmost position at which a pixel intersects with the
32 left edge of the trapezoid). For example, as shown in FIG. 4B, a shape walker may determine,
33 at the row corresponding to y-value 481, x-min value 472 and x-max value 475 for a left-side

1 edge of a trapezoid. In a different circumstance where a tile includes a right side edge of a
2 trapezoid, the x-min and x-max values may similarly be determined based on a corresponding
3 function equation. In yet another circumstance where a tile includes both a left side edge and
4 a right side edge, a shape walker 130 may also similarly determine the function equation for
5 both of the edges, but in separate operations. In particular embodiments, a shape walker 130
6 may be configured to determine the x-min and x-max values for a top or bottom edge based
7 on the function equation of the side edges and/or the bounding box information. For example,
8 referring to FIG. 4A, a shape walker 130 may determine, based on the bounding box
9 information, that tile 2 contains only a top edge of a trapezoid and that none of the side edges
10 are contained in tile 2. Based on this determination, a shape walker 130 may determine that
11 the top edge spans across the entirety of the length of the bounding box, and thereby
12 determine the x-min values and x-max values based on the position of the bounding box. A
13 shape walker 130 may similarly determine the x-min and x-max values of the bottom edge of
14 a trapezoid contained in tile 10. If a bounding box for a particular tile contains either a top or
15 bottom edge in addition to one of the side edges, such as tile 1 shown in FIG. 4B, a shape
16 walker 130 may be configured to plug in the y-value of the top/bottom edge into the function
17 equation of the side edges to determine the x-min and x-max values of the top/bottom edge.
18 For example, referring to FIG. 4B, a shape walker 130 may determine the x-min value of the
19 top edge by plugging in y-min value 453 into the function equation of edge 456. As for the x-
20 max value, a shape walker 130 may determine that, since the right side edge is not contained
21 in tile 423, the x-max value equals the rightmost x value of the bounding box 435. The
22 techniques discussed above may be used to determine the x-min and x-max values of any of
23 the top or bottom edges of a trapezoid. In an embodiment, for each row in a tile, a shape
24 walker 130 may be configured to determine the individual pixels that are intersecting with an
25 edge of a trapezoid based on the x-min and x-max values determined using the techniques
26 discussed above. For pixels that are intersecting with an edge of a trapezoid, a shape walker
27 130 may identify those pixels to an integrator 140, as explained further below.

28 [76] If the primitive is a quadratic curve, rather than a trapezoid, in accordance to
29 particular embodiments, a shape walker 130 may be configured to analyze each pixel position
30 within a tile to determine whether the corresponding pixel overlaps/intersects with an edge of
31 a quadratic curve. As shown in **FIG. 5A**, a quadratic curve primitive may be comprised of
32 two edges, one flat edge 503 and one curved edge 506. A shape walker 130 may be
33 configured to process, for each quadratic curve within a tile, a single edge at a time. The

1 technique of determining whether a pixel overlaps/intersects with a flat edge of a quadratic
2 curve is substantially similar to the technique described above with reference to a trapezoid,
3 for example, by representing the flat edge with a linear equation. The technique of
4 determining whether a pixel overlaps/intersects with a curved edge of a quadratic curve is
5 also substantially similar to the technique described above with reference to a trapezoid, but
6 in contrast, a quadratic formula is used to represent the curved edge rather than a linear
7 equation. For example, referring to FIG. 5A, a quadratic equation (ax^2+bx+c) may be used to
8 represent the function equation for the curved edge 506. Such quadratic equations may be
9 determined based on the three vertices (P0, P1, P2) of the quadratic curve shown in FIG. 5A.
10 The location of such vertices may be determined by a driver 102 or an application 101 and
11 provided to a graphics engine 103 (e.g., shape walker 130).

12 [77] In particular embodiments, to determine the function equation for a curved
13 edge of a quadratic curve, a shape walker 130 may be configured to determine the y-min and
14 y-max values and y-intercepts of the curved edge. A shape walker 130 may then use this
15 information and the three vertices of a quadratic curve (e.g., such as those shown in FIG. 5A)
16 to determine a quadratic equation that represents the curved edge of the quadratic curve. **FIG.**
17 **5B** illustrates an example tile comprising a curved edge 571 of a quadratic curve. Once the y-
18 min, y-max, and function equation is determined for a curved edge of a quadratic curve, a
19 shape walker 130 may be configured to traverse the tile row-by-row (e.g., from y-min to y-
20 max) and determine, for each row, pixels that intersect with the curved edge based on, for
21 example, the quadratic equation and the DDA line generating algorithm. For example, a
22 shape walker 130 may be configured to traverse the tile 580 row by row, starting at y-min
23 573 and ending with y-max 576. For each row, a shape walker 130 may be configured to
24 determine the x-min and x-max values for that row based on the corresponding quadratic
25 equation. Referring to the bottom half of FIG. 5B, a shape walker 130 may determine, for the
26 row corresponding to y-value 591, that x-min 592 is the leftmost position at which a pixel
27 intersects with a curved edge of a quadratic curve and that x-max 593 is the rightmost
28 position at which a pixel intersects with the curved edge of the trapezoid. In an embodiment,
29 for each row in a tile, a shape walker 130 may be configured to determine the individual
30 pixels that are intersecting with an edge of a quadratic curve based on the x-min and x-max
31 values determined using the techniques discussed above. For pixels that are intersecting with
32 an edge of a quadratic curve, a shape walker 130 may identify those pixels to an integrator
33 140, as explained further below.

1 [78] In particular embodiments, once the x-min and x-max values that are defining
2 the pixels intersecting with an edge of a primitive have been determined for each row within
3 a tile, a shape walker 130 may be configured to assign each pixel within the tile a coverage
4 weight or flagged for the integrator 140. Pixels that are overlapping with an edge of a
5 primitive are flagged and provided to an integrator 140. Pixels that are fully outside a
6 primitive are assigned a coverage weight of 0. Pixels that are fully inside a primitive are
7 assigned a coverage weight of 0. A shape walker 130 may be configured to assign every pixel
8 outside the bounding box a coverage weight of 0. To evaluate pixels that are inside the
9 bounding box, a shape walker 130 may walk through each pixel row-by-row. For example,
10 referring to FIG. 4B, a shape walker 130 may start from y-min 453 and determine that y-min
11 453 corresponds to a top edge of a trapezoid. A shape walker 130 may then assign a coverage
12 weight of 0 to pixels that are located to the left of the previously determined x-min value for
13 this row. A shape walker 130 may also determine that pixels that are located to right of the of
14 the x-min value for that row intersect with the top edge of the trapezoid and flag those pixels
15 to the integrator 140. For the row corresponding to y-min 453 + 1, a shape walker may
16 similarly determine that the row also corresponds to the top edge and assign a coverage
17 weight of 0 to pixels that are located to the left of the x-min value previously determined for
18 that row and flag pixels that are located to right of the x-min value for the integrator 140. For
19 the row corresponding to y-min 453 + 2, a shape walker may determine that that this row
20 corresponds to a left-side edge of a trapezoid. A shape walker may then assign a coverage
21 weight of 0 to pixels that are located to the left of the corresponding x-min value, flag pixels
22 that are between x-min and x-max (including pixels having x-min and x-max values), and
23 assign a coverage weight of 1 to pixels that are located to right of the corresponding x-max
24 value. A shape walker may repeat these steps for each row within the bounding box until all
25 pixels within the bounding box are either assigned a coverage weight or flagged for the
26 integrator 140. This example technique may similarly be applied to tiles containing other
27 edges of a trapezoid. For example, if the tile 423 shown in FIG. 4B included the right side
28 edge of a trapezoid rather than a left side edge, pixels to the left of the edge would be
29 assigned a coverage weight of 1, while pixels to the right of the edge would be assigned a
30 coverage weight of 0. In particular embodiments, pixels that are intersecting with a top edge
31 or a bottom edge of a trapezoid may be flagged for an integrator 140. The above example
32 technique may similarly be applied to tiles containing an edge of a curve based on the x-min
33 and x-max values determined based on a linear equation (for the flat line) or a quadratic

1 equation (for the curved line).

2 [79] In particular embodiments, a shape walker 130 may be configured to examine,
3 prior to determining coverage weights of pixels that are fully outside or inside a primitive and
4 prior to flagging pixels that are intersecting with an edge of a primitive, whether the tile
5 bounding box is bigger than a minimal threshold size. If the bounding box is smaller than a
6 minimal threshold size (such as 1x1 pixel or 2x2 pixels), a shape walker 130 may be
7 configured to send all of the pixels within the bounding box to an integrator 140 to determine
8 their respective coverage weight, rather than going through the steps described in the
9 preceding paragraphs. In an embodiment, determining whether the bounding box is bigger
10 than a threshold size may be implemented for a trapezoid but not for a quadratic curve.

11 [80] An integrator 140 may be configured to determine anti-alias pixel coverage
12 weights for each pixel flagged by a shape walker 130. Pixels that are assigned a coverage
13 weight by an integrator 140 are forwarded to a coverage buffer 151 or 152. An integrator 140
14 may only be responsible for determining coverage weights for pixels that are flagged by a
15 shape walker 130, for example, pixels that intersect an edge of a primitive. As discussed
16 above, coverage weights for pixels that are fully outside or fully inside a primitive are
17 assigned by a shape walker 130. The technique of determining the anti-alias pixel coverage
18 weights for each pixel flagged by a shape walker 130 (e.g., partially covered by a primitive)
19 involves utilizing the well-understood property of a trapezoid or a quadratic curve function.
20 An example of such a technique is disclosed in the following paper: A. Ellis, W. Hunt, J.
21 Hart, Nerf: *Real-Time Analytic Antialiased Text for 3-D Environments*, Computer Graphics
22 forum, vol. 38, issue 8, November 2019, pp. 23-32.

23 [81] In particular embodiments, coverage buffers 151 and 153 may be configured
24 to store and maintain coverage weights for pixels, as determined by either a shape walker 130
25 or an integrator 140. In particular embodiments, two coverage buffers may be configured in a
26 double buffer configuration such that one coverage buffer is assigned to the rasterization
27 process while the other is assigned to the shading process, then alternating the roles as
28 necessary. For example, referring to FIG. 1A, the double buffer configuration allows a first
29 coverage buffer (e.g., 151) to be updated by a shape walker 130 and integrator 140, while a
30 second coverage buffer (e.g., 153) can be accessed by other components of the system, for
31 example, a shader 150.

32 [82] In particular embodiments, each coverage buffer may be configured to store a
33 coverage weight for each pixel within a tile. A coverage weight of zero represents full

1 transparency, and a value of 1 (or in some embodiments 2^{10-1} (i.e., 1023)) represents a fully
2 opaque. Intermediate values between full transparency and fully opaque represent partially
3 transparent pixels that can be combined with a background image to yield a composite image.
4 As discussed previously, in accordance to embodiments, instructions to update the coverage
5 buffer for pixels that are fully transparent or fully opaque are received from a shape walker
6 130 and instructions to update the coverage buffer for pixels that are partially transparent are
7 received from an integrator 140.

8 [83] In particular embodiments, a shader 150 may be configured to perform fixed
9 function shading of the pixels of a primitive. In particular embodiments, a shader 150 may be
10 configured to perform any of the following types of shading operations: solid fill, gradient
11 fill, and texturing. Texturing involves invoking a texture unit 170. In particular embodiments,
12 a shader 150 performs shading operations tile by tile, and for each tile, pixel by pixel based
13 on the coverage weight associated with each pixel. A shader 150 may be configured to
14 determine the source color information and the determined information may be passed on to a
15 color buffer 191 or 193 for blending operations. In an embodiment, a shader 150 generates
16 the texture space coordinates by transforming the conversion matrix from the shader
17 information into texel space coordinates. A shader 150 may then be configured to adjust for
18 the shear and then clamps the output to send it to the texture block.

19 [84] In particular embodiments, color buffers 191 and 193 may be configured to
20 perform blending operations. In particular embodiments, two color buffers may be configured
21 in a double buffer configuration to allow one color buffer to be updated while the other is
22 being accessed. Color buffers 191 and 193 may receive the source color information and
23 pixel coverage weights from a shader 150 or a blit and filtering unit 180. Based on a gamma
24 correction mode, color buffers 191 and 193 may be configured to convert the input source
25 color into gamma space before performing a blending operation. Once converted, the output
26 may be converted back to linear space using the degamma unit. Such gamma conversion
27 steps are optional. After color buffers 191 or 193 finishes the blending operations, the
28 blended color data may be streamed out to the tile compress and store 195. The blended color
29 data may be streamed out in a block by block fashion (e.g., 4x4 pixel arrays).

30 [85] As discussed above, in an embodiment, a command controller 107 may
31 determine, for each tile containing at least a portion of a primitive, a tile bounding box that
32 encompasses the at least the portion of the primitive. This technique may be referred as a
33 “culling” technique where tiles of a frame (e.g., 16x16 pixels) are culled using a smallest

1 bounding box that encompasses a primitive being processed by the graphics processing unit
2 (GPU), or a graphics system. Only the tiles covered by the primitive bounding box may be
3 identified to the downstream GPU components in the rendering pipeline to allow the
4 downstream GPU components to effectively ignore the empty tiles (i.e., tiles that are
5 completely outside any primitive bounding box). This reduces the overall computing required
6 and makes the system more efficient. Incorporating this culling technique, however, presents
7 a challenge because the technique conflicts with some of the blending modes that are used to
8 blend overlapping primitives. Examples of such blending modes may include blending modes
9 that are referred to as src, srcIn, srcOut, dstAtop, and dstIn (hereinafter referred to as “special
10 blending modes”). These special blending modes require access to both the tiles covering the
11 destination primitive (tiles already in the color buffer) and the tiles covering the source
12 primitive (tiles that are to be written into the color buffer). For example, the special blending
13 modes may require access to the tiles covering the source primitive to update the color
14 information of the pixels in those tiles while also requiring access to the tiles covering the
15 destination primitive to clear/update/remove the color information of the pixels in the tiles
16 covering the destination primitive. However, due to the culling technique, when a graphics
17 system is processing a source primitive, the graphics system only has access to tiles covering
18 the source primitive and do not have access to the tiles of the destination primitive.
19 Embodiments disclosed herein provide a technique for addressing this challenge. Blending
20 modes that do not require updating the pixels in the tiles covering the destination primitive
21 are referred herein as “normal blending modes.” Operations that involve special blending
22 modes may herein be referred to as “special blending operations.” Operations that involve
23 normal blending modes may herein be referred to as “normal blending operations.”

24 [86] References to a destination primitive herein may refer to a “shape” that is
25 stored in a color buffer, which may be a primitive or a blend of multiple primitives that have
26 been blended into the color buffer. References to a source primitive herein may similarly
27 refer to a “shape” that is to be stored/blended into a color buffer, which may be a primitive.

28 [87] In particular embodiment, a graphic system may be configured to implement
29 the blending operations sequentially, primitive by primitive. This means that, when the
30 system is processing a particular primitive, only the tiles covered by the primitive are
31 processed by the system while other tiles are ignored. If, for example, a particular frame
32 comprises multiple primitives, each of the primitive may be processed one at a time, in a
33 sequence, which may require processing the same tiles multiple times if multiple primitives

are covered by the tiles. **FIG. 6** illustrates an example frame with two primitives, a destination primitive 610 and a source primitive 630. The destination primitive 610 represents a primitive that is already stored in a color buffer, while the source primitive 630 represents a primitive that is to be written into the color buffer. Tiles that are covering the destination primitive 610 may be referred herein as destination tiles and tiles that are covering the source primitive 630 may be referred herein as source tiles. When blending two overlapping primitives, such as those illustrated in FIG. 6, special blending modes require an operation where the primitive in the destination tiles are cleared of the pixel values, but as discussed above, a graphics system may not have access to the destination tiles.

[88] In particular embodiments, the task of clearing a destination primitive may first involve categorizing the tiles in a frame as “non-empty tiles” when the tiles cover a source primitive and as “empty tiles” when the tiles do not cover the source primitive. For example, in FIG. 6, the tiles within the dotted outline 643 may be categorized as non-empty tiles since a source primitive 630 touches each of those tiles. Tiles that are outside the dotted outline 643 may be categorized as empty tiles since none of them touch the source primitive 630.

[89] In particular embodiments, when executing a special blending mode, a graphics system may clear a destination primitive from empty tiles by instructing the color buffer to bypass the primitive cull associated with a source primitive (e.g., bounding box of the source primitive) to allow the color buffer to gain access to previously inaccessible tiles (e.g., tiles that are beyond the source primitive’s bounding box). The color buffer may then be configured to clear the empty tiles by updating the pixel values associated each pixel within the empty tiles (e.g., tiles that are beyond the source primitive’s bounding box and associated with a destination primitive/shape). Alternatively, a graphics system may be configured to instruct the color buffer to process a dummy primitive (e.g., a primitive associated with clear color values) that overlaps the destination primitive, effectively “clearing” the color information of the destination primitive by replacing it with clear color information.

[90] For non-empty tiles, the clearing task is a bit more complicated since only the destination primitive must be cleared from the non-empty tiles are covering both the destination primitive and the source primitive. For example, in FIG. 6, the clearing task would require clearing only a portion of tile 645 covering a destination primitive 610 without also clearing the portion of the tile 645 covering a source primitive 630. The techniques

disclosed above with reference to clearing the empty tiles—as opposed to non-empty tiles here—will not be appropriate for tile 645 since, for example, implementing the above techniques may also clear the source primitive within tile 645. Embodiments disclosed herein, therefore provide a solution to this problem by utilizing a pixel-by-pixel analysis to identify particular pixels within a tile that is only associated with a destination primitive then selectively clearing the pixel values associated with the identified pixels.

[91] For clearing a destination primitive from non-empty tiles, in accordance to particular embodiments, a graphics system may maintain status bits for each of the pixels in the non-empty tiles that track the recent blending mode(s) that has been used for that pixel or whether the most recent blending mode used for that pixel is a normal blending mode or a special blending mode. The graphics system may use the status bits to identify pixels that have been touched by the most recent normal blending operation, i.e., pixels covering a destination primitive. In particular embodiments, a graphics system assigns a primitive a blending mode (normal blending mode or special blending mode) before the primitive is blended into a color buffer. For example, referring to FIG. 6, a graphics system may have assigned the destination primitive 610 a normal blending mode before it was blended into the color buffer and the source primitive 630 with a special blending mode before it is blended into the color buffer. Pixel values associated with a primitive are similarly associated with data indicating whether it is associated with a normal blending mode or a special blending mode.

[92] In particular embodiments, a graphics system may be configured to utilize status bit W0 to indicate whether a pixel has been touched by a normal blending mode and status bit W1 to indicate whether a pixel has been touched by a special blending mode. For example, status bits “00” (equivalent to side-by-side status bits W1 and W0) is used to indicate that a pixel has not been touched by any blending operations, and thus pixel values associated with the pixel should correspond to the background color of a frame. Status bits “01” is used to indicate that a pixel has been touched by a normal blending mode. Status bits “10” is used to indicate that a pixel has been touched by a special blending mode. Status bits “11” is used to indicate that a pixel has been touched by both normal and special blending modes. For example, in FIG. 6, when the source primitive 630 is blended into the color buffer, pixels that are covering only the destination primitive 610 may be associated with status bits 01, pixels that are covering only the source primitive 630 may be associated with status bits 10, pixels that are covering both the destination primitive 610 and the source

1 primitive 630 (the overlapping region) may be associated with status bits 11, and pixels that
2 are not associated either primitives may be associated with status bits 00. In an embodiment,
3 when overlapping pixels for which status bits will be 10, appropriate blending operation may
4 be performed by using background color information as destination color. Whereas when
5 status bit 11 is encountered, appropriate blending operation is performed by reading the color
6 from color memory as destination color

7 [93] In particular embodiments, at the end of each special blending operation, a
8 graphics system may be configured to implement a “flag treatment step” by which status bits
9 are reset such that status bits 00 remains as 00, status bits 01 are changed to 00, and status
10 bits 10 and 11 are changed to 01. When the graphics system finishes blending a frame, the
11 graphics system may be configured to export the color information of the pixels based on the
12 current status bits: for pixels with status bits 00, the graphics system may export the
13 background color information rather than retrieving the color information from the color
14 memory; for pixels with status bits 01, the graphics system may export the color information
15 from the color memory. Once the flag treatment step is executed at the end of a special
16 blending operation, the graphics system may be able to identify pixels that have been touched
17 by that special blending operations by searching for pixels that are associated with status bits
18 01. Other pixels in the frame should be associated with status bits 00 due to the resetting
19 process discussed above with reference to the flag treatment step. And, as discussed above,
20 when exporting the color information for pixels associated with status bits 00, the graphics
21 system may not retrieve the color information from the color buffer, rather the system may
22 simply retrieve/use the background color information. The use of the background color
23 information when exporting the pixel color information is effectively equivalent to clearing
24 the pixel values associated the pixels with status bits 00 since pixels without any value
25 correspond to the background color. In other words, the flag treatment step effectively clears
26 out the destination primitive since the pixels that has been touched only by a normal blending
27 mode (status bits 00 and 01 before the flag treatment step) are changed to 00 and background
28 color information is exported for those pixels. In some embodiments, the flag treatment step
29 is executed not at the end of special blending operation but prior to the beginning of a
30 subsequent blending mode that follows a special blending operation. In an embodiment, a
31 graphics system also maintains additional status data indicating the previous blending mode
32 that has touched a pixel, if any, to determine the transition between the blending operations.

33 [94] References to pixel values or pixel color information as used herein may refer

1 to any of the red, green, or blue color channels, and/or opaqueness channel.

2 [95] In particular embodiments, a texture unit 170 may be configured to provide
3 texture information for pixel covered by a primitive and shades the color of the pixel. If the
4 covered pixel has texture fill, then corresponding texture image may be fetched and filtered to
5 obtain the color information for the covered pixel. The covered pixel may then be shaded
6 with the derived color.

7 [96] In particular embodiments, a tile compress store 195 may be configured to
8 receive the rendered tile data from color buffers 191 and 193. A tile compress store 195 may
9 comprise a block encoder (e.g., hardware encoder) that is configured to encode the rendered
10 tile data before being transmitted to a display driver 198. In particular embodiments, a tile
11 compress store 195 may be responsible for encoding static assets (e.g., a blit such as an
12 emoji, a company logo, or a watch face for a smart watch), which may be stored a memory
13 external to the graphic engine 103 to be accessed at a later time point. Static images need to
14 be encoded at low power but with high throughput. To achieve such a feat, when encoding an
15 image (asset), a tile compress store 195 may use a “spatial prediction” technique that
16 leverages the fact that some groups of pixels in an image comprises the same pixel values as
17 other groups. Additional details for this technique are described below.

18 [97] FIG. 7 illustrates an example encoding pipeline. Tiles that are encoded by the
19 encoding system are piped through a double buffer 751 such that the current tile can be
20 compressed while the next tile is streamed in. For each tile to be encoded, a block scheduler
21 753 may separate the tile into blocks for the encoder. A block scheduler 753 may schedule
22 the blocks in an arrangement that is optimized for delta coding, for example, in an
23 arrangement that minimizes the spatial distance between the blocks in a sequence. An
24 example of such an arrangement is called the “Morton Order.” FIG. 8 illustrates a tile 782
25 that is segmented into multiple blocks, e.g., block 874, each block comprising multiple pixels
26 or texel, e.g., 4x4 pixels/texels. A block encoder 760 may be configured to encode blocks in a
27 tile in an arrangement specified by a block scheduler. For a tile comprising pixels of multiple
28 channels, or components, a block encoder 760 may be configured to encode each pixel
29 channel separately. Examples of pixel channels or components are color components (e.g., R,
30 G, B) or an opaque component (e.g., transparency). The encoded channels may then be
31 collated into a single bitstream. The encoded data may be provided to a memory write
32 controller 760. A memory write controller 760 may then send the encoded data to a memory
33 to be stored and made accessible for later retrieval by a graphics engine.

1 [98] FIG. 9 illustrates an example encoding pipeline executed by a block encoder
2 760. In particular embodiments, a tile compress store 195 may be configured to encode an
3 image based on groups of texels, each of which may be referred to as a “block.” A block may
4 be comprised of, e.g., 4x4 texels. A block encoder 760 may comprise a block analyzer 905, a
5 spatial predictor 901, a texel scheduler 901, texel scheduler 910, delta coder 920, channel
6 entropy coder 930, and channel data collator 940. In an embodiment, the encoding pipeline
7 illustrated in FIG. 9 represents an encoding pipeline of a hardware encoder, but substantially
8 similar pipeline may be implemented as a software encoder. Each of the system components
9 illustrated in FIG. 9 may be configured to operate based on an encoding cycle where each
10 system component processes one block per one encoding cycle.

11 [99] In an embodiment, a spatial predictor 901 may be configured to compare the
12 texel values of the current block to previously processed blocks, if any, to determine whether
13 the texel values of the current block matches the texel values of any of the previously
14 processed blocks. For example, a spatial predictor 901 may compare the texel values of the
15 current block with the texel values of up to four of the previously processed blocks. If a
16 matching block is found, the spatial predictor 901 may forgo encoding the texel block of the
17 current block and instead assign a block header to the current block that matches a block
18 header of the matching block. Such a technique allows a block encoder 760 to skip the
19 encoding process for the current block since the duplication of the block header allows the
20 matching block’s compressed block data to be utilized for both the current block and the
21 matching block. However, there is a power-efficiency concern with the above described
22 technique because texel-by-texel comparison of blocks requires a significant amount of
23 compute power and memory storage. As a solution, embodiments disclose a technique of
24 generating a hash code, or hash representation, to represent the texel values of each block and
25 using the hash codes to make the comparison rather than comparing the actual texel values of
26 the blocks. In an embodiment, a block encoder 760 may be configured to generate hash
27 representations that are 32-bits or 64-bits. Notably, a 32-bit or 64-bit block hash comparison
28 is significantly cheaper, computationally, than comparing the 4x4 block data.

29 [100] There exists yet another problem with the technique discussed above with
30 reference to comparing the hash codes. As shown in FIG. 9, the encoding process involves
31 several steps in a pipeline. The step of comparing the blocks (e.g., comparing hash codes)
32 occurs at the first step, by a spatial predictor 901, but the encoding pipeline may be
33 configured such that the block header for each block is generated at the end of the encoding

1 process (e.g., by a channel data collator 940). This means that when a spatial predictor 901
2 compares a particular block to one of the previously processed blocks, the previously blocks
3 may still be going through the encoding pipeline and their block header may not have been
4 generated yet. In circumstances where a spatial predictor 901 finds a matching block but the
5 block header has not been generated yet, a spatial predictor 901 may assign the current block
6 a placeholder tag in place of a header, and a copy of the tag may be passed along the pipeline.
7 Then, at the end of each encoding cycle (e.g., when a block is handed off to the next step in
8 the encoding process), the block encoder 760 may check whether a previously-unavailable
9 header is available, and if so, replaces the corresponding tag with the header. This solution
10 prevents the encoding pipeline from being stalled due to certain headers not being available at
11 the time a matching block is found.

12 [101] FIG. 10 illustrates an example of the techniques described above with
13 reference to a spatial predictor 901. When a spatial predictor 901 processes a block, it may be
14 configured to first analyze the texel values associated with the block to validate whether the
15 block comprises valid texel values, as opposed to having no value or null value. If the block
16 includes valid texel values, a spatial predictor 901 may be configured to generate a hash
17 representation of the texel values associated with the block, via hash function 1020. Then, the
18 spatial predictor 901 may be configured to compare the hash representation of the current
19 block with the hash representation of blocks that were previously processed by the spatial
20 predictor 901. If a match is found for the current block, a spatial predictor 901 may duplicate
21 a block header for the current block that matches the block header of the matching block. For
22 example, as illustrated in FIG. 8, a spatial predictor 901 may maintain a table 1010
23 comprising data associated with up to four previously processed blocks with respect to the
24 current block. Such a table 1010 may be used to store data indicating whether a block is
25 associated with valid texel values (e.g., in column 1031), hash representation of the texel
26 values of the block (e.g., in column 1032), and block header or placeholder tag for the block
27 (e.g., in column 1033). As noted above, if a matching block is found but the block header of
28 the matching block has not been generated yet, a spatial predictor 901 may be configured to
29 generate a placeholder tag, for example, “tag_blockHeader 3” in FIG. 8. A copy of such a tag
30 may be sent along the encoding pipeline illustrated in FIG. 9. At the end of each encoding
31 cycle, a spatial predictor 901 may be configured to determine whether the block header of the
32 matching block is available, and if so, replace the tag with the appropriate block header. In an
33 embodiment, if the current block being processed by a spatial predictor 901 matches one of

1 the previously processed blocks, the current block may still be sent down the encoding
2 pipeline due to the hardware configuration of the encoding pipeline. For example, the current
3 block may still be sent to a texel scheduler 910, delta coder 920, channel entropy coder 930,
4 and channel data collator 940, but text values of the block may not be processed by such
5 system components. If the current block being processed by a spatial predictor 901 does not
6 match one of the previously processed blocks, the current block may be sent down the
7 encoding pipeline (in either hardware or software configurations) and the texel value of the
8 block may be encoded according to embodiments disclosed herein. In an embodiment, a
9 spatial predictor 901 may be configured to maintain a table 1010 based on a first-in-first-out
10 (FIFO) protocol such that when the table is filled, the oldest entry is overwritten upon new
11 incoming data. In a software implementation of the encoding pipeline, if the current block
12 being processed by a spatial predictor 901 matches one of the previously processed blocks,
13 the current block may not need to be passed through the encoding pipeline as is done with a
14 hardware configuration, rather, rest of the encoding pipeline may be skipped. In particular
15 embodiments, a block header specifies a memory region where the encoded block is stored.
16 As such, when multiple blocks are encoded using the same header, a single encoded block
17 data can be used for those multiple blocks.

18 [102] Referring back to FIG. 9, after a spatial predictor 901 completes processing
19 the current block, the current block may be passed onto the subsequent downstream
20 components of the encoding pipeline. Examples of such downstream components of the
21 encoding pipeline include a block analyzer 905, texel scheduler 910, delta coder 920, channel
22 entropy coder 930, and channel data collator 940. Described below are techniques used by the
23 downstream components to analyze and encode texel values of blocks.

24 [103] In particular embodiments, a block analyzer 905 may be configured to analyze
25 texel blocks and categorize them into one of two block variants: Flatblock or Codeblock. A
26 block may be categorized as a Flatblock if all texels in the block have the same value. A
27 block may be categorized as a Codeblock if some of the texels in the block have different
28 values. Once a block is categorized as a Flatblock or a Codeblock, a block analyzer 905 may
29 be configured to pass the block to a texel scheduler 910.

30 [104] In particular embodiments, a texel scheduler 910 may be configured to
31 schedule the texels in a block (e.g., Codeblock) in a sequence optimized for delta encoding.
32 For example, the texels in a block may be scheduled in a Morton Order shown in FIG. 8. The
33 arranged texels may then be provided to a delta coder 920. A texel scheduler 910 may be

1 configured to schedule the texels of a Codeblock and, but not for a Flatblock since delta
2 coding is not necessary for a Flatblock.

3 [105] In particular embodiments, a delta coder 920 may be configured to encode a
4 texel block using various techniques. For a Flatblock, a delta coder 920 may be configured
5 encode the block using a single texel value since a Flatblock contains only a single texel
6 value. For a Codeblock having multiple texel channels (e.g., R, G, B, opacity), a delta coder
7 920 may be configured to encode each texel channel separately from each other, and different
8 encoding techniques may be used to encode each channel. The different encoding techniques
9 used by a delta coder 92 may include a “flat” technique, “variable-length” technique, and an
10 uncompressed technique which essentially involves “encoding” (e.g., storing) texel values as
11 uncompressed. These encoding techniques may also be referred to as compression modes, for
12 example, “variable length” mode, “flat” mode, or uncompressed mode. A particular channel
13 of a Codeblock may be encoded using a “flat” technique if all of the values of the texels in
14 the channel are the same. The flat technique involves using a single value to represent the
15 entire channel. A particular channel of a Codeblock may be encoded using a “variable-
16 length” technique if values of the texels within the channel differ from each other. The
17 variable-length technique is a novel compression technique that produces different sizes of
18 encoded data depending on the differences in the texel values within the block. As for the
19 uncompressed technique, while it may involve storing the corresponding pixel values as
20 uncompressed (e.g., without any compression), for the purposes of describing the
21 embodiments herein, the uncompressed technique/mode may still be referred to as one of the
22 “compression” techniques/modes used to “encode” texel values of a texel block, and its
23 operations may be described as the process of “compressing” the texel values.

24 [106] In particular embodiments, the variable-length technique may involve
25 generating three groups of data to represent the encoded texel values: “symbolmask”; “rbits”;
26 “rsymbols.” Data group rsymbols is used to represent the non-zero delta values of the texel
27 values as arranged by a texel scheduler 910 (e.g., in a Morton Order). For example, if there
28 are 16 texel values in a sequence, there would be 15 delta values, each delta value
29 representing the difference of one texel value to the next in that sequence, or the difference of
30 one texel value to the previous texel value in that sequence if considering how the sequence
31 of texel values may be read by a block encoder 760. Data group rsymbols is used to represent
32 only the non-zero delta of those 15 delta values. Data group symbolmask is used to provide a
33 1 to 1 mapping of the delta values that indicates whether each delta value is a zero value or

1 non-zero value. Data group rbits is used to indicate the maximum number of bits required to
2 represent each of the delta values, along with an additional bit to indicate whether the delta
3 values are positive or negative values. In other words, rbits may be used to indicate the width
4 of a symbol (e.g., a symbol being a delta value), and rbits may be referred to as a “symbol
5 width.” **FIG. 11** illustrates an example of what a compressed channel of texel values would
6 look like when symbolmask, rbits, and rsymbols are continuously packed. As indicated in
7 **FIG. 11**, in particular embodiments, the variable-length technique may be configured to
8 produce variable length of bits for rsymbols while symbolmask and rbits may each be
9 configured with a fixed number of bit lengths that are determined prior to the encoding
10 process. For example, if a block comprises 4x4 texels (16 texel values), a block encoder 760
11 may be configured to assign symbolmask a bit length of 15 bits since there would be 15 delta
12 values. As for rbits, a block encoder 760 may be configured to assign rbits a bit length that is
13 required to represent the magnitude of the delta values along with one additional bit to
14 represent whether a particular delta value is a positive or negative value.

15 **[107] FIG. 12** illustrates an example diagram for encoding a 4x4 texel block using a
16 variable-length technique. Specifically, **FIG. 12** illustrates a 4x4 texel block 872 comprising
17 16 texel values, which when arranged in a Morton Order are as follows: [0, 0, 0, 0, 8, 8, 8, 8,
18 0, 0, 0, 0, 8, 8, 0, 0]. The delta values, or delta coded stream, of the texel values arranged in
19 the Morton Order would have 15 values and are as follows: [0, 0, 0, 8, 0, 0, 0, -8, 0, 0, 0, 8, 0,
20 -8, 0]. The rbits for these delta values would be 5 since a bit length of 5 (i.e., 5 bits) would be
21 required to represent each of the delta values, a first bit to indicate a positive or negative sign
22 of the delta values and four additional bits to represent the delta values’ maximum value of 8.
23 In an embodiment, the rbits may be encoded in a binary representation, such that rbits of 5
24 may be stored as [101]. In some embodiments, rbits may be stored with an offset, for
25 example, with an offset of 2 such that rbits of 5 may be stored as 3, or [011]. Storing rbits
26 with an offset increases the range of the values that rbits can represent and leverages the fact
27 that rbits of 0 or 1 would not be needed because, for example, the maximum number of bits
28 required to represent each of the delta values, which is what rbits represents, would always
29 require a value greater than 0 or 1. Continuing the example illustrated in **FIG. 12**, the
30 rsymbols for the delta values would be [-8, 8, -8, 8] or in binary representation [11000,
31 01000, 11000, 01000], where each value of rsymbols has a bit length of rbits (5 bits) with a
32 first bit used to indicate whether the delta value is a positive or negative value and four
33 additional bits to represent the magnitude of the delta values. The symbolmask for the delta

1 values would be [0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] with the most significant bit (“MSB”) placed to the left side and the least significant bit (“LSB”) placed on the right side. In FIG. 2 12, this sequence of values of symbolmask is presented in a reverse order with respect to how 3 the delta values were presented in the previous steps. As discussed above, symbolmask are 4 used to indicate whether a delta value is zero or non-zero. Notably, the data group rsymbols 5 only needs to represent non-zero delta values since any zero delta values are already indicated 6 by symbolmask. In addition to the three groups of data, the first value of the uncompressed 7 texel values may be encoded as the “base value” of the encoded data, either encoded together 8 with the three groups of data or separately as metadata. In the example above, the base value 9 would be 0 since that is the first value of the uncompressed texel values. Once all three 10 groups of data are generated for a texel block, they may be collated together into a stream of 11 bits, for example, in the configuration shown in FIG. 11. The collated stream of bits 12 represents the encoded data for a particular channel of a texel block. In an embodiment, a 13 block metadata that is encoded with the encoded data may comprise data indicating the 14 number of texel channels included in a block and the type of technique used to each of the 15 channels. In an embodiment, extra bits may be encoded into the encoded data to make it byte- 16 aligned. For example, as shown in FIG. 12, if the encoded data results in a bit length of 38 17 bits, two extra bits may be added to make it byte-aligned (e.g., multiples of 8 bits). 18

19 [108] As discussed above, each texel channel within a Codeblock may be 20 independently encoded using any of the techniques described above (e.g., flat technique, 21 variable-length technique, or uncompressed). For example, for a Codeblock having three 22 channels of texel values, a first channel of the three may be encoded using a flat technique, a 23 second channel of the three may be encoded using the variable-length technique, and a third 24 channel of the three may be stored as uncompressed.

25 [109] In particular embodiments, after encoding a channel using the variable-length 26 technique, a channel entropy coder 930 illustrated in FIG. 9 may be configured to evaluate 27 whether the encoded channel data is computationally more expensive than the uncompressed 28 channel data, that is, whether the encoded data requires more bits than the uncompressed 29 data. If so, the channel entropy coder may disregard the encoded channel data and instead use 30 the uncompressed channel data. In other words, for each channel/component of a texel block, 31 a channel entropy coder 930 may determine whether to encode the texel values for that 32 channel using one of the compression techniques disclosed above or based on the 33 uncompressed texel values. In particular embodiments, a channel entropy coder 930 may

1 evaluate the encoded block data of the Codeblock to see whether the encoded data is greater
2 in size than the uncompressed size of the block, that is, whether the encoded data requires
3 more bits than uncompressed data. If so, the encoding system may be configured to (1)
4 disregard the encoded block data of the Codeblock, (2) recategorize the Codeblock as a third
5 block variant referred to as a Rawblock, and (3) store the uncompressed block data in lieu of
6 the disregarded encoded data. The encoding system stores a Rawblock without any
7 compression. The size of a Rawblock represents the maximum size of a stored block. In an
8 embodiment, a channel entropy coder 930 may be configured to evaluate the entirety of a
9 block to categorize the block into one of the variants described above, meaning that, if a
10 block includes multiple texel components, all texels within the block are evaluated without
11 separately evaluating texels of different components. For example, a block comprising
12 multiple channels of texels may be categorized as a Flatblock only if all texels within the
13 block have the same value, including texels of different components. Alternatively, if any
14 texel values differ in a block, even across different channels, the block may be categorized as
15 a Codeblock. Examples of texel components, or channels, include color components (e.g., R,
16 G, B) or an opaque component (e.g., transparency).

17 [110] In particular embodiments, a channel data collator 940 may be configured to
18 collate each of the encoded, or uncompressed, channels of texel values into a bit stream that
19 results in the encoded block data. In particular embodiments, a channel data collator 940 may
20 generate a block header for each texel block. A block header may comprise a pointer (e.g., an
21 offset value) that indicates the location of the block data in the memory that is relative to
22 block data associated other blocks of an image. A block header may comprise data that
23 indicates whether the encoded texel block is compressed or uncompressed, the number of
24 texel channels in the block, and if the block is compressed, the size or length of the
25 compressed block data (e.g., measured in bits or bytes). In particular embodiment, as shown
26 in FIG. 9, once a block header is generated for a texel block, the block header may be
27 provided to a spatial predictor 901. The spatial predictor 901 may then evaluate whether any
28 texel block is associated with a placeholder tag that has been generated in place for the block
29 header, and if so, replace the placeholder tag with the block header.

30 [111] The encoding pipeline illustrated in FIG. 9 provides a unique way of encoding
31 the blocks that allows a decoder to selectively retrieve and decode any particular block of the
32 encoded blocks independently from other encoded pixel blocks. More specifically, each
33 block is encoded in a way that it is self-contained, meaning that a decoder can selectively

1 retrieve and decompress a particular block simply based on the data contained within the
2 block. For example, if a PNG image that encoded using the techniques described herein, a
3 decoder may be able to retrieve and decompress specific portions of the PNG image
4 independently from other portions of the PNG image.

5 [112] In particular embodiments, a blit and filtering unit 180 may be configured to
6 retrieve static graphics content from a memory database 109 and, if necessary, perform
7 decoding operations and/or transformation or filtering operations on the graphics content
8 referred to as a “blit” operation. A blit operation refers to a hardware feature that moves a
9 rectangular block of bits from main memory into display memory. In particular embodiments,
10 a graphics system disclosed herein may store static graphic content, such as pre-rendered
11 images (e.g., emoji), in a memory 109 that is external to the graphics engine. A blit and
12 filtering unit 180 may be configured to retrieve content from the memory and perform
13 transformation or filtering operations, then provide the transformed/filtered content to a color
14 buffer. In particular embodiments, a blit and filtering unit 180 may be configured to update
15 the input data based on the command it receives from a tile controller 120. A blit and filtering
16 unit 180 may include a memory structure, for example, a single color buffer. Incoming source
17 image information per tile may be buffered in this memory structure to improve the
18 performance of the blit and filtering unit 180. A blit and filtering unit 180 may perform a set
19 of predefined operations and filters. A blit and filtering unit 180 provides a power-
20 performance-area (PPA) optimized solution to some common data rearrangement/movement
21 (with filter) operations to the hardware. In particular embodiments, a blit and filtering unit
22 180 may comprises a decoder configured to decode static graphics content that has been
23 encoded and stored in a memory database 109. A blit and filtering unit 180 may be
24 configured to provide the decoded graphics content to a color buffer 191, 193.

25 [113] FIG. 13 illustrates an example technique of decoding a 4x4 texel block that
26 has been encoded by a block encoder 760. Specifically, FIG. 13 illustrates an encoded texel
27 data comprising three data groups, rsymbols, rbits, and symbolmask. As discussed above,
28 data group rsymbols is used to represent the non-zero delta values of the sequence of texel
29 values of a block as arranged in, for example, a Morton Order. Data group symbolmask is
30 used to provide a 1 to 1 mapping of the delta values that indicates whether each delta value is
31 a zero value or non-zero value. Data group rbits is used to indicate the maximum number of
32 bits required to represent each of the delta values, along with an additional bit to indicate
33 whether the delta values are positive or negative values. In an embodiment, a decoder may be

1 configured to decode multiple delta values per one decoding cycle. For example, FIG. 13
2 illustrates an embodiment where three multiplexers, i.e., symbolMUX 1312, 1314, 1316, are
3 configured to decode three segments of rsymbols (delta values) in parallel during each
4 decoding cycle. Although FIG. 13 illustrates an embodiment where three segments of
5 rsymbols are decoded per each decoding cycle, any number of segments may be configured
6 to be decoded per each cycle, for example, five segments of rsymbols in parallel. While FIG.
7 13 illustrates one instance of a decoding operation for a particular channel of texel values,
8 multiple of such instances may be configured to be implemented such that all texel channels
9 are decoded in parallel. Once all channels are decoded, the decoded values may be collated,
10 resulting in uncompressed texel values.

11 [114] In the embodiment illustrated in FIG. 13, for example, a decoder may be
12 configured to decode a 4x4 texel block having rbits of 8, which indicates that each segment
13 of rsymbols (e.g., each delta value) is 8 bits long. Given that the decoder is configured to
14 decode three delta values in parallel, rMUX 1301 may be configured to fetch up to three delta
15 values per decoding cycle, that is, up to 27 bits at a time. In an embodiment, a decoder may
16 be configured to implement an initializing operation where symbolmask is parsed to
17 determine the number of delta values rMUX that should be fetched in each decoding cycle.
18 For example, if the first three symbolmask bits are [101], indicating that the first and third
19 values are non-zero values and the second is a zero value, then rMUX 1301 may be
20 configured to fetch two segments of rsymbols for the first decoding cycle (first two delta
21 values). Also during the first decoding cycle, the first three symbolmask bits may be provided
22 to symbolMUX 1312, 1314, or 1316, respectively. For each zero value, a corresponding
23 symbolMUX (e.g., symbolMUX 1312, 1314, or 1316) may be configured to pass a zero value
24 to the next component in the decoder, for example, to a corresponding adder 1341, 1343, or
25 1345. For each non-zero value, a corresponding MUX (e.g., symbolMUX 1312, 1314, or
26 1316) may be configured to fetch a non-zero segment of rsymbols (delta value). For example,
27 if the first three symbolmask bits are [101] such that 1 is provided to symbolMUX 1316, 0 is
28 provided to symbolMUX 1314, and 1 is provided to symbolMUX 1312, then symbolMUX
29 1316 may fetch the first non-zero delta value from rMUX 1301, symbolMUX 1312 may fetch
30 the second non-zero delta value from rMUX 1301, and a zero value may be passed through
31 symbolMUX 1314. Once the first three delta values are retrieved by the respective
32 symbolMUX 1312, 1314, and 1316 (whether non-zero or zero), the delta values may be
33 passed to the respective adders 1341, 1343, and 1345. Then, a decoder may be configured to

1 add the base value of the encoded data to the first delta value to determine the second texel
2 value, add the resulting value to the second delta value to determine the third texel value, then
3 add the resulting value to the third delta value to determine the fourth texel value. In such a
4 fashion, the first four texel values of the encoded data may be determined after the first
5 decoding cycle, the first texel value being the base value. The next three texel values may
6 similarly be decoded during a second decoding cycle, and additional decoding cycles may
7 further be implemented until the block is decoded completely.

8 [115] FIG. 14 illustrates an example method 1400 for determining the color
9 information of primitives in an image base in part by determining the coverage weight of
10 each pixel in the image. The method may begin at step 1401 by receiving a list of primitives
11 covering a tile of an image that is to be rendered, the image comprising content defined by at
12 least the list of primitives. At step 1402, the method may continue by, for each primitive in
13 the list, identifying, in the tile, partially-covered pixels that are partially covered by the
14 primitive, fully-uncovered pixels that are fully uncovered by the primitive, and fully-covered
15 pixels that are fully covered by the primitive. At step 1403, the method may continue by, for
16 each primitive in the list, computing, for each of the partially-covered pixels, a coverage
17 weight indicating a proportion of the partially-covered pixel that is covered by the primitive.
18 At step 1404, the method may continue by, for each primitive in the list, storing coverage
19 data in a coverage buffer corresponding to the tile, the coverage data comprising the coverage
20 weights of the partially-covered pixels, fully-uncovered indicators for the fully-uncovered
21 pixels, and fully-covered indicators for the fully-covered pixels. At step 1405, the method
22 may continue by, for each primitive in the list, determining color information for the
23 primitive in the tile based on the stored coverage data. At step 1406, the method may
24 continue by, for each primitive in the list, aggregating the color information of the list of
25 primitives in a color buffer for output. Particular embodiments may repeat one or more steps
26 of the method of FIG. 14, where appropriate. Although this disclosure describes and
27 illustrates particular steps of the method of FIG. 14 as occurring in a particular order, this
28 disclosure contemplates any suitable steps of the method of FIG. 14 occurring in any suitable
29 order. Moreover, although this disclosure describes and illustrates an example method for
30 determining the color information of primitives in an image, this disclosure contemplates any
31 suitable method for determining the color information of primitives in an image including
32 any suitable steps, which may include all, some, or none of the steps of the method of FIG.
33 14, where appropriate. Furthermore, although this disclosure describes and illustrates

1 particular components, devices, or systems carrying out particular steps of the method of FIG.
2 14, this disclosure contemplates any suitable combination of any suitable components,
3 devices, or systems carrying out any suitable steps of the method of FIG. 14.

4 [116] FIG. 15 illustrates an example method 1500 for determining the color
5 information of a primitive base in part by determining the coverage weight of each pixel of
6 primitive based on function equations representing the edges of the primitives. The method
7 may begin at step 1501 by receiving instructions to render an image comprising content
8 defined by at least a two-dimensional (2D) primitive. At step 1502, the method may continue
9 by determining a portion of the 2D primitive covering a tile of a plurality of tiles of the
10 image. At step 1503, the method may continue by generating an edge definition to represent
11 an edge of the portion of the 2D primitive. At step 1504, the method may continue by, for
12 each row of pixels within at least a portion of the tile containing the portion of the 2D
13 primitive, identifying, based on the edge definition, a left-most pixel and a right-most pixel in
14 the row that intersect the edge. At step 1505, the method may continue by, for each row of
15 pixels within at least a portion of the tile containing the portion of the 2D primitive,
16 identifying, based on the left-most pixel and the right-most pixel, a set of first pixels in the
17 row intersecting the edge. At step 1506, the method may continue by, for each row of pixels
18 within at least a portion of the tile containing the portion of the 2D primitive, determining, for
19 each first pixel in the set, a coverage weight indicating a proportion of the first pixel covered
20 by the 2D primitive. At step 1507, the method may continue by, for each row of pixels within
21 at least a portion of the tile containing the portion of the 2D primitive, determining color
22 information for the set of first pixels based on the associated coverage weights. Particular
23 embodiments may repeat one or more steps of the method of FIG. 15, where appropriate.
24 Although this disclosure describes and illustrates particular steps of the method of FIG. 15 as
25 occurring in a particular order, this disclosure contemplates any suitable steps of the method
26 of FIG. 15 occurring in any suitable order. Moreover, although this disclosure describes and
27 illustrates an example method for determining the color information of a primitive, this
28 disclosure contemplates any suitable method for determining the color information of a
29 primitive including any suitable steps, which may include all, some, or none of the steps of
30 the method of FIG. 15, where appropriate. Furthermore, although this disclosure describes
31 and illustrates particular components, devices, or systems carrying out particular steps of the
32 method of FIG. 15, this disclosure contemplates any suitable combination of any suitable
33 components, devices, or systems carrying out any suitable steps of the method of FIG. 15.

1 **[117] FIG. 16** illustrates an example method 1600 for blending source shape with a
2 destination shape using a blending mode that requires updates to pixels in the color buffer
3 uncovered by the source shape. The method may begin at step 1601 by receiving a source
4 shape that is to be blended with a destination shape stored in a color buffer for an image. The
5 following steps are performed in response to determining that the source shape is associated
6 with a blending mode that requires updates to pixels in the color buffer uncovered by the
7 source shape. At step 1602, the method may continue by identifying one or more empty tiles
8 in the color buffer uncovered by the source shape and one or more non-empty tiles in the
9 color buffer covered by the source shape. At step 1603, the method may continue by, for each
10 of the one or more empty tiles, sending instructions to clear pixel values associated with the
11 empty tile in the color buffer. At step 1604, the method may continue by, for each of the one
12 or more non-empty tiles, identifying one or more pixels of the non-empty tile that are covered
13 by the destination shape but not the source shape and sending instructions to clear pixel
14 values associated with the one or more pixels. Particular embodiments may repeat one or
15 more steps of the method of FIG. 16, where appropriate. Although this disclosure describes
16 and illustrates particular steps of the method of FIG. 16 as occurring in a particular order, this
17 disclosure contemplates any suitable steps of the method of FIG. 16 occurring in any suitable
18 order. Moreover, although this disclosure describes and illustrates an example method for
19 blending source shape with a destination shape using a blending mode that requires updates
20 to pixels in the color buffer uncovered by the source shape, this disclosure contemplates any
21 suitable method for blending source shape with a destination shape using a blending mode
22 that requires updates to pixels in the color buffer uncovered by the source shape including
23 any suitable steps, which may include all, some, or none of the steps of the method of FIG.
24 16, where appropriate. Furthermore, although this disclosure describes and illustrates
25 particular components, devices, or systems carrying out particular steps of the method of FIG.
26 16, this disclosure contemplates any suitable combination of any suitable components,
27 devices, or systems carrying out any suitable steps of the method of FIG. 16.

28 **[118] FIG. 17** illustrates an example method 1700 for encoding blocks of pixels
29 based on a tag that is used to temporary represent block headers. The method may begin at
30 step 1701 by receiving a plurality of blocks of pixels of an image, wherein the blocks are to
31 be sequentially encoded using a hardware-encoding pipeline. At steps 1702-1706, the method
32 may continue by encoding a first block of the plurality of blocks. Specifically, at step 1702,
33 the method may continue by generating a first hash to represent the first block. At step 1703,

1 the method may continue by identifying a second hash stored in memory matching the first
2 hash, the second hash (i) representing a second block of the plurality of blocks previously
3 processed by the hardware-encoding pipeline and (ii) is associated with a tag corresponding
4 to a placeholder for a second header associated with the second block. At step 1704, the
5 method may continue by passing a copy of the tag through the hardware-encoding pipeline as
6 metadata for the first block. At step 1705, the method may continue by determining that the
7 second header is available. At step 1706, the method may continue by replacing the copy of
8 the tag with the second header to generate a first encoding for the first block, wherein the
9 second header specifies a memory region where a second encoding of the second block is
10 stored. Particular embodiments may repeat one or more steps of the method of FIG. 17,
11 where appropriate. Although this disclosure describes and illustrates particular steps of the
12 method of FIG. 17 as occurring in a particular order, this disclosure contemplates any suitable
13 steps of the method of FIG. 17 occurring in any suitable order. Moreover, although this
14 disclosure describes and illustrates an example method for encoding blocks of pixels based
15 on a tag that is used to temporary represent block headers, this disclosure contemplates any
16 suitable method for encoding blocks of pixels based on a tag that is used to temporary
17 represent block headers including any suitable steps, which may include all, some, or none of
18 the steps of the method of FIG. 17, where appropriate. Furthermore, although this disclosure
19 describes and illustrates particular components, devices, or systems carrying out particular
20 steps of the method of FIG. 17, this disclosure contemplates any suitable combination of any
21 suitable components, devices, or systems carrying out any suitable steps of the method of
22 FIG. 17.

23 [119] FIG. 18 illustrates an example method 1800 for determining whether a block
24 of pixels is different from previously-compressed blocks and compressing the block using a
25 variable-length technique. The method may begin at step 1801 by determining a sequence for
26 compressing blocks of pixels in an image. At step 1802, the method may continue by
27 compressing the blocks sequentially according to the sequence, wherein a first component of
28 a first block is compressed, details of which are laid out in steps 1803 and 1807. At step
29 1803, the method may continue by selecting a variable-length mode from a plurality of
30 supported compression modes to compress the first component of the first block, which is
31 based on steps 1804-1806. At step 1804, the method may continue by determining that the
32 first block is different from previously-compressed blocks compressed according to the
33 sequence. At step 1805, the method may continue by determining that pixels within the first

1 component are different. At step 1806, the method may continue by determining that a bit
2 length needed for compressing the first component using the variable-length mode is less
3 than a bit length needed for representing the first component uncompressed. At step 1807, the
4 method may continue by generating a first compression of the first component of the first
5 block using a symbol width selected based on magnitudes of delta values used for encoding
6 the pixels within the first component of the first block. Particular embodiments may repeat
7 one or more steps of the method of FIG. 18, where appropriate. Although this disclosure
8 describes and illustrates particular steps of the method of FIG. 18 as occurring in a particular
9 order, this disclosure contemplates any suitable steps of the method of FIG. 18 occurring in
10 any suitable order. Moreover, although this disclosure describes and illustrates an example
11 method for determining whether a block of pixels is different from previously-compressed
12 blocks and compressing the block using a variable-length technique, this disclosure
13 contemplates any suitable method for determining whether a block of pixels is different from
14 previously-compressed blocks and compressing the block using a variable-length technique
15 including any suitable steps, which may include all, some, or none of the steps of the method
16 of FIG. 18, where appropriate. Furthermore, although this disclosure describes and illustrates
17 particular components, devices, or systems carrying out particular steps of the method of FIG.
18 18, this disclosure contemplates any suitable combination of any suitable components,
19 devices, or systems carrying out any suitable steps of the method of FIG. 18.

20 [120] **FIG. 19** illustrates an example method 1900 for encoding a plurality of pixels
21 based on delta encoding that utilizes a base value, symbol mask, symbol width, and sequence
22 of symbols. The method may begin at step 1901 by receiving a block comprising a plurality
23 of pixels. At step 1902, the method may continue by encoding the plurality of pixels, details
24 of which are laid out in steps 1903-1908. At step 1903, the method may continue by
25 arranging the plurality of pixels in a sequence. At step 1904, the method may continue by
26 generating a delta encoding of the plurality of pixels, the delta encoding comprising (a) a base
27 value and (b) a plurality of delta values having non-zero delta values and zero delta values,
28 each delta value representing a difference between a corresponding pixel in the sequence and
29 a previous pixel in the sequence. At step 1905, the method may continue by generating a
30 symbol mask indicating whether each of the plurality of delta values is zero or non-zero. At
31 step 1906, the method may continue by determining, based on magnitudes of the non-zero
32 delta values, a symbol width for encoding each of the non-zero delta values. At step 1907, the
33 method may continue by generating a sequence of symbols that respectively encode the non-

1 zero delta values using the symbol width. At step 1908, the method may continue by
2 generating a compression of the block by collating the symbol mask, the symbol width, and
3 the sequence of symbols. Particular embodiments may repeat one or more steps of the
4 method of FIG. 19, where appropriate. Although this disclosure describes and illustrates
5 particular steps of the method of FIG. 19 as occurring in a particular order, this disclosure
6 contemplates any suitable steps of the method of FIG. 19 occurring in any suitable order.
7 Moreover, although this disclosure describes and illustrates an example method for encoding
8 a plurality of pixels based on delta encoding that utilizes a base value, symbol mask, symbol
9 width, and sequence of symbols, this disclosure contemplates any suitable method for
10 encoding a plurality of pixels based on delta encoding that utilizes a base value, symbol
11 mask, symbol width, and sequence of symbols including any suitable steps, which may
12 include all, some, or none of the steps of the method of FIG. 19, where appropriate.
13 Furthermore, although this disclosure describes and illustrates particular components,
14 devices, or systems carrying out particular steps of the method of FIG. 19, this disclosure
15 contemplates any suitable combination of any suitable components, devices, or systems
16 carrying out any suitable steps of the method of FIG. 19.

17 [121] FIG. 20 illustrates an example network environment 2000 associated with a
18 social-networking system. Network environment 2000 includes a client system 2030, a
19 social-networking system 2060, and a third-party system 2070 connected to each other by a
20 network 2010. Although FIG. 20 illustrates a particular arrangement of client system 2030,
21 social-networking system 2060, third-party system 2070, and network 2010, this disclosure
22 contemplates any suitable arrangement of client system 2030, social-networking system
23 2060, third-party system 2070, and network 2010. As an example and not by way of
24 limitation, two or more of client system 2030, social-networking system 2060, and third-party
25 system 2070 may be connected to each other directly, bypassing network 2010. As another
26 example, two or more of client system 2030, social-networking system 2060, and third-party
27 system 2070 may be physically or logically co-located with each other in whole or in part.
28 For example, an AR/VR headset 2030 may be connected to a local computer or mobile
29 computing device 2070 via short-range wireless communication (e.g., Bluetooth). Moreover,
30 although FIG. 20 illustrates a particular number of client systems 2030, social-networking
31 systems 2060, third-party systems 2070, and networks 2010, this disclosure contemplates any
32 suitable number of client systems 2030, social-networking systems 2060, third-party systems
33 2070, and networks 2010. As an example and not by way of limitation, network environment

1 2000 may include multiple client system 2030, social-networking systems 2060, third-party
2 systems 2070, and networks 2010.

3 [122] This disclosure contemplates any suitable network 2010. As an example and
4 not by way of limitation, one or more portions of network 2010 may include a short-range
5 wireless network (e.g., Bluetooth, Zigbee, etc.), an ad hoc network, an intranet, an extranet, a
6 virtual private network (VPN), a local area network (LAN), a wireless LAN (WLAN), a wide
7 area network (WAN), a wireless WAN (WWAN), a metropolitan area network (MAN), a
8 portion of the Internet, a portion of the Public Switched Telephone Network (PSTN), a
9 cellular telephone network, or a combination of two or more of these. Network 2010 may
10 include one or more networks 2010.

11 [123] Links 2050 may connect client system 2030, social-networking system 2060,
12 and third-party system 2070 to communication network 2010 or to each other. This disclosure
13 contemplates any suitable links 2050. In particular embodiments, one or more links 2050
14 include one or more wireline (such as for example Digital Subscriber Line (DSL) or Data
15 Over Cable Service Interface Specification (DOCSIS)), wireless (such as for example Wi-Fi,
16 Worldwide Interoperability for Microwave Access (WiMAX), Bluetooth), or optical (such as
17 for example Synchronous Optical Network (SONET) or Synchronous Digital Hierarchy
18 (SDH)) links. In particular embodiments, one or more links 2050 each include an ad hoc
19 network, an intranet, an extranet, a VPN, a LAN, a WLAN, a WAN, a WWAN, a MAN, a
20 portion of the Internet, a portion of the PSTN, a cellular technology-based network, a satellite
21 communications technology-based network, another link 2050, or a combination of two or
22 more such links 2050. Links 2050 need not necessarily be the same throughout network
23 environment 2000. One or more first links 2050 may differ in one or more respects from one
24 or more second links 2050.

25 [124] In particular embodiments, client system 2030 may be an electronic device
26 including hardware, software, or embedded logic components or a combination of two or
27 more such components and capable of carrying out the appropriate functionalities
28 implemented or supported by client system 2030. As an example and not by way of
29 limitation, a client system 2030 may include a computer system such as a VR/AR headset,
30 desktop computer, notebook or laptop computer, netbook, a tablet computer, e-book reader,
31 GPS device, camera, personal digital assistant (PDA), handheld electronic device, cellular
32 telephone, smartphone, augmented/virtual reality device, other suitable electronic device, or
33 any suitable combination thereof. This disclosure contemplates any suitable client systems

1 2030. A client system 2030 may enable a network user at client system 2030 to access
2 network 2010. A client system 2030 may enable its user to communicate with other users at
3 other client systems 2030.

4 [125] In particular embodiments, social-networking system 2060 may be a network-
5 addressable computing system that can host an online social network. Social-networking
6 system 2060 may generate, store, receive, and send social-networking data, such as, for
7 example, user-profile data, concept-profile data, social-graph information, or other suitable
8 data related to the online social network. Social-networking system 2060 may be accessed by
9 the other components of network environment 2000 either directly or via network 2010. As
10 an example and not by way of limitation, client system 2030 may access social-networking
11 system 2060 using a web browser, or a native application associated with social-networking
12 system 2060 (e.g., a mobile social-networking application, a messaging application, another
13 suitable application, or any combination thereof) either directly or via network 2010. In
14 particular embodiments, social-networking system 2060 may include one or more servers
15 2062. Each server 2062 may be a unitary server or a distributed server spanning multiple
16 computers or multiple datacenters. Servers 2062 may be of various types, such as, for
17 example and without limitation, web server, news server, mail server, message server,
18 advertising server, file server, application server, exchange server, database server, proxy
19 server, another server suitable for performing functions or processes described herein, or any
20 combination thereof. In particular embodiments, each server 2062 may include hardware,
21 software, or embedded logic components or a combination of two or more such components
22 for carrying out the appropriate functionalities implemented or supported by server 2062. In
23 particular embodiments, social-networking system 2060 may include one or more data stores
24 2064. Data stores 2064 may be used to store various types of information. In particular
25 embodiments, the information stored in data stores 2064 may be organized according to
26 specific data structures. In particular embodiments, each data store 2064 may be a relational,
27 columnar, correlation, or other suitable database. Although this disclosure describes or
28 illustrates particular types of databases, this disclosure contemplates any suitable types of
29 databases. Particular embodiments may provide interfaces that enable a client system 2030, a
30 social-networking system 2060, or a third-party system 2070 to manage, retrieve, modify,
31 add, or delete, the information stored in data store 2064.

32 [126] In particular embodiments, social-networking system 2060 may store one or
33 more social graphs in one or more data stores 2064. In particular embodiments, a social graph

1 may include multiple nodes—which may include multiple user nodes (each corresponding to
2 a particular user) or multiple concept nodes (each corresponding to a particular concept)—
3 and multiple edges connecting the nodes. Social-networking system 2060 may provide users
4 of the online social network the ability to communicate and interact with other users. In
5 particular embodiments, users may join the online social network via social-networking
6 system 2060 and then add connections (e.g., relationships) to a number of other users of
7 social-networking system 2060 to whom they want to be connected. Herein, the term “friend”
8 may refer to any other user of social-networking system 2060 with whom a user has formed a
9 connection, association, or relationship via social-networking system 2060.

10 [127] In particular embodiments, social-networking system 2060 may provide users
11 with the ability to take actions on various types of items or objects, supported by social-
12 networking system 2060. As an example and not by way of limitation, the items and objects
13 may include groups or social networks to which users of social-networking system 2060 may
14 belong, events or calendar entries in which a user might be interested, computer-based
15 applications that a user may use, transactions that allow users to buy or sell items via the
16 service, interactions with advertisements that a user may perform, or other suitable items or
17 objects. A user may interact with anything that is capable of being represented in social-
18 networking system 2060 or by an external system of third-party system 2070, which is
19 separate from social-networking system 2060 and coupled to social-networking system 2060
20 via a network 2010.

21 [128] In particular embodiments, social-networking system 2060 may be capable of
22 linking a variety of entities. As an example and not by way of limitation, social-networking
23 system 2060 may enable users to interact with each other as well as receive content from
24 third-party systems 2070 or other entities, or to allow users to interact with these entities
25 through an application programming interfaces (API) or other communication channels.

26 [129] In particular embodiments, a third-party system 2070 may include a local
27 computing device that is communicatively coupled to the client system 2030. For example, if
28 the client system 2030 is an AR/VR headset, the third-party system 2070 may be a local
29 laptop configured to perform the necessary graphics rendering and provide the rendered
30 results to the AR/VR headset 2030 for subsequent processing and/or display. In particular
31 embodiments, the third-party system 2070 may execute software associated with the client
32 system 2030 (e.g., a rendering engine). The third-party system 2070 may generate sample
33 datasets with sparse pixel information of video frames and send the sparse data to the client

1 system 2030. The client system 2030 may then generate frames reconstructed from the
2 sample datasets.

3 [130] In particular embodiments, the third-party system 2070 may also include one
4 or more types of servers, one or more data stores, one or more interfaces, including but not
5 limited to APIs, one or more web services, one or more content sources, one or more
6 networks, or any other suitable components, e.g., that servers may communicate with. A
7 third-party system 2070 may be operated by a different entity from an entity operating social-
8 networking system 2060. In particular embodiments, however, social-networking system
9 2060 and third-party systems 2070 may operate in conjunction with each other to provide
10 social-networking services to users of social-networking system 2060 or third-party systems
11 2070. In this sense, social-networking system 2060 may provide a platform, or backbone,
12 which other systems, such as third-party systems 2070, may use to provide social-networking
13 services and functionality to users across the Internet.

14 [131] In particular embodiments, a third-party system 2070 may include a third-
15 party content object provider (e.g., including sparse sample datasets described herein). A
16 third-party content object provider may include one or more sources of content objects, which
17 may be communicated to a client system 2030. As an example and not by way of limitation,
18 content objects may include information regarding things or activities of interest to the user,
19 such as, for example, movie show times, movie reviews, restaurant reviews, restaurant
20 menus, product information and reviews, or other suitable information. As another example
21 and not by way of limitation, content objects may include incentive content objects, such as
22 coupons, discount tickets, gift certificates, or other suitable incentive objects.

23 [132] In particular embodiments, social-networking system 2060 also includes user-
24 generated content objects, which may enhance a user's interactions with social-networking
25 system 2060. User-generated content may include anything a user can add, upload, send, or
26 "post" to social-networking system 2060. As an example and not by way of limitation, a user
27 communicates posts to social-networking system 2060 from a client system 2030. Posts may
28 include data such as status updates or other textual data, location information, photos, videos,
29 links, music or other similar data or media. Content may also be added to social-networking
30 system 2060 by a third-party through a "communication channel," such as a newsfeed or
31 stream.

32 [133] In particular embodiments, social-networking system 2060 may include a
33 variety of servers, sub-systems, programs, modules, logs, and data stores. In particular

1 embodiments, social-networking system 2060 may include one or more of the following: a
2 web server, action logger, API-request server, relevance-and-ranking engine, content-object
3 classifier, notification controller, action log, third-party-content-object-exposure log,
4 inference module, authorization/privacy server, search module, advertisement-targeting
5 module, user-interface module, user-profile store, connection store, third-party content store,
6 or location store. Social-networking system 2060 may also include suitable components such
7 as network interfaces, security mechanisms, load balancers, failover servers, management-
8 and-network-operations consoles, other suitable components, or any suitable combination
9 thereof. In particular embodiments, social-networking system 2060 may include one or more
10 user-profile stores for storing user profiles. A user profile may include, for example,
11 biographic information, demographic information, behavioral information, social
12 information, or other types of descriptive information, such as work experience, educational
13 history, hobbies or preferences, interests, affinities, or location. Interest information may
14 include interests related to one or more categories. Categories may be general or specific. As
15 an example and not by way of limitation, if a user “likes” an article about a brand of shoes the
16 category may be the brand, or the general category of “shoes” or “clothing.” A connection
17 store may be used for storing connection information about users. The connection
18 information may indicate users who have similar or common work experience, group
19 memberships, hobbies, educational history, or are in any way related or share common
20 attributes. The connection information may also include user-defined connections between
21 different users and content (both internal and external). A web server may be used for linking
22 social-networking system 2060 to one or more client systems 2030 or one or more third-party
23 system 2070 via network 2010. The web server may include a mail server or other messaging
24 functionality for receiving and routing messages between social-networking system 2060 and
25 one or more client systems 2030. An API-request server may allow a third-party system 2070
26 to access information from social-networking system 2060 by calling one or more APIs. An
27 action logger may be used to receive communications from a web server about a user’s
28 actions on or off social-networking system 2060. In conjunction with the action log, a third-
29 party-content-object log may be maintained of user exposures to third-party-content objects.
30 A notification controller may provide information regarding content objects to a client system
31 2030. Information may be pushed to a client system 2030 as notifications, or information may
32 be pulled from client system 2030 responsive to a request received from client system 2030.
33 Authorization servers may be used to enforce one or more privacy settings of the users of

1 social-networking system 2060. A privacy setting of a user determines how particular
2 information associated with a user can be shared. The authorization server may allow users to
3 opt in to or opt out of having their actions logged by social-networking system 2060 or shared
4 with other systems (e.g., third-party system 2070), such as, for example, by setting
5 appropriate privacy settings. Third-party-content-object stores may be used to store content
6 objects received from third parties, such as a third-party system 2070. Location stores may be
7 used for storing location information received from client systems 2030 associated with
8 users. Advertisement-pricing modules may combine social information, the current time,
9 location information, or other suitable information to provide relevant advertisements, in the
10 form of notifications, to a user.

11 [134] FIG. 21 illustrates an example computer system 2100. In particular
12 embodiments, one or more computer systems 2100 perform one or more steps of one or more
13 methods described or illustrated herein. In particular embodiments, one or more computer
14 systems 2100 provide functionality described or illustrated herein. In particular embodiments,
15 software running on one or more computer systems 2100 performs one or more steps of one
16 or more methods described or illustrated herein or provides functionality described or
17 illustrated herein. Particular embodiments include one or more portions of one or more
18 computer systems 2100. Herein, reference to a computer system may encompass a computing
19 device, and vice versa, where appropriate. Moreover, reference to a computer system may
20 encompass one or more computer systems, where appropriate.

21 [135] This disclosure contemplates any suitable number of computer systems 2100.
22 This disclosure contemplates computer system 2100 taking any suitable physical form. As
23 example and not by way of limitation, computer system 2100 may be an embedded computer
24 system, a system-on-chip (SOC), a single-board computer system (SBC) (such as, for
25 example, a computer-on-module (COM) or system-on-module (SOM)), a desktop computer
26 system, a laptop or notebook computer system, an interactive kiosk, a mainframe, a mesh of
27 computer systems, a mobile telephone, a personal digital assistant (PDA), a server, a tablet
28 computer system, an augmented/virtual reality device, or a combination of two or more of
29 these. Where appropriate, computer system 2100 may include one or more computer systems
30 2100; be unitary or distributed; span multiple locations; span multiple machines; span
31 multiple data centers; or reside in a cloud, which may include one or more cloud components
32 in one or more networks. Where appropriate, one or more computer systems 2100 may
33 perform without substantial spatial or temporal limitation one or more steps of one or more

1 methods described or illustrated herein. As an example and not by way of limitation, one or
2 more computer systems 2100 may perform in real time or in batch mode one or more steps of
3 one or more methods described or illustrated herein. One or more computer systems 2100
4 may perform at different times or at different locations one or more steps of one or more
5 methods described or illustrated herein, where appropriate.

6 [136] In particular embodiments, computer system 2100 includes a processor 2102,
7 memory 2104, storage 2106, an input/output (I/O) interface 2108, a communication interface
8 2110, and a bus 2112. Although this disclosure describes and illustrates a particular computer
9 system having a particular number of particular components in a particular arrangement, this
10 disclosure contemplates any suitable computer system having any suitable number of any
11 suitable components in any suitable arrangement.

12 [137] In particular embodiments, processor 2102 includes hardware for executing
13 instructions, such as those making up a computer program. As an example and not by way of
14 limitation, to execute instructions, processor 2102 may retrieve (or fetch) the instructions
15 from an internal register, an internal cache, memory 2104, or storage 2106; decode and
16 execute them; and then write one or more results to an internal register, an internal cache,
17 memory 2104, or storage 2106. In particular embodiments, processor 2102 may include one
18 or more internal caches for data, instructions, or addresses. This disclosure contemplates
19 processor 2102 including any suitable number of any suitable internal caches, where
20 appropriate. As an example and not by way of limitation, processor 2102 may include one or
21 more instruction caches, one or more data caches, and one or more translation lookaside
22 buffers (TLBs). Instructions in the instruction caches may be copies of instructions in
23 memory 2104 or storage 2106, and the instruction caches may speed up retrieval of those
24 instructions by processor 2102. Data in the data caches may be copies of data in memory
25 2104 or storage 2106 for instructions executing at processor 2102 to operate on; the results of
26 previous instructions executed at processor 2102 for access by subsequent instructions
27 executing at processor 2102 or for writing to memory 2104 or storage 2106; or other suitable
28 data. The data caches may speed up read or write operations by processor 2102. The TLBs
29 may speed up virtual-address translation for processor 2102. In particular embodiments,
30 processor 2102 may include one or more internal registers for data, instructions, or addresses.
31 This disclosure contemplates processor 2102 including any suitable number of any suitable
32 internal registers, where appropriate. Where appropriate, processor 2102 may include one or
33 more arithmetic logic units (ALUs); be a multi-core processor; or include one or more

1 processors 2102. Although this disclosure describes and illustrates a particular processor, this
2 disclosure contemplates any suitable processor.

3 [138] In particular embodiments, memory 2104 includes main memory for storing
4 instructions for processor 2102 to execute or data for processor 2102 to operate on. As an
5 example and not by way of limitation, computer system 2100 may load instructions from
6 storage 2106 or another source (such as, for example, another computer system 2100) to
7 memory 2104. Processor 2102 may then load the instructions from memory 2104 to an
8 internal register or internal cache. To execute the instructions, processor 2102 may retrieve
9 the instructions from the internal register or internal cache and decode them. During or after
10 execution of the instructions, processor 2102 may write one or more results (which may be
11 intermediate or final results) to the internal register or internal cache. Processor 2102 may
12 then write one or more of those results to memory 2104. In particular embodiments,
13 processor 2102 executes only instructions in one or more internal registers or internal caches
14 or in memory 2104 (as opposed to storage 2106 or elsewhere) and operates only on data in
15 one or more internal registers or internal caches or in memory 2104 (as opposed to storage
16 2106 or elsewhere). One or more memory buses (which may each include an address bus and
17 a data bus) may couple processor 2102 to memory 2104. Bus 2112 may include one or more
18 memory buses, as described below. In particular embodiments, one or more memory
19 management units (MMUs) reside between processor 2102 and memory 2104 and facilitate
20 accesses to memory 2104 requested by processor 2102. In particular embodiments, memory
21 2104 includes random access memory (RAM). This RAM may be volatile memory, where
22 appropriate. Where appropriate, this RAM may be dynamic RAM (DRAM) or static RAM
23 (SRAM). Moreover, where appropriate, this RAM may be single-ported or multi-ported
24 RAM. This disclosure contemplates any suitable RAM. Memory 2104 may include one or
25 more memories 2104, where appropriate. Although this disclosure describes and illustrates
26 particular memory, this disclosure contemplates any suitable memory.

27 [139] In particular embodiments, storage 2106 includes mass storage for data or
28 instructions. As an example and not by way of limitation, storage 2106 may include a hard
29 disk drive (HDD), a floppy disk drive, flash memory, an optical disc, a magneto-optical disc,
30 magnetic tape, or a Universal Serial Bus (USB) drive or a combination of two or more of
31 these. Storage 2106 may include removable or non-removable (or fixed) media, where
32 appropriate. Storage 2106 may be internal or external to computer system 2100, where
33 appropriate. In particular embodiments, storage 2106 is non-volatile, solid-state memory. In

1 particular embodiments, storage 2106 includes read-only memory (ROM). Where
2 appropriate, this ROM may be mask-programmed ROM, programmable ROM (PROM),
3 erasable PROM (EPROM), electrically erasable PROM (EEPROM), electrically alterable
4 ROM (EAROM), or flash memory or a combination of two or more of these. This disclosure
5 contemplates mass storage 2106 taking any suitable physical form. Storage 2106 may include
6 one or more storage control units facilitating communication between processor 2102 and
7 storage 2106, where appropriate. Where appropriate, storage 2106 may include one or more
8 storages 2106. Although this disclosure describes and illustrates particular storage, this
9 disclosure contemplates any suitable storage.

10 [140] In particular embodiments, I/O interface 2108 includes hardware, software, or
11 both, providing one or more interfaces for communication between computer system 2100
12 and one or more I/O devices. Computer system 2100 may include one or more of these I/O
13 devices, where appropriate. One or more of these I/O devices may enable communication
14 between a person and computer system 2100. As an example and not by way of limitation, an
15 I/O device may include a keyboard, keypad, microphone, monitor, mouse, printer, scanner,
16 speaker, still camera, stylus, tablet, touch screen, trackball, video camera, another suitable I/O
17 device or a combination of two or more of these. An I/O device may include one or more
18 sensors. This disclosure contemplates any suitable I/O devices and any suitable I/O interfaces
19 2108 for them. Where appropriate, I/O interface 2108 may include one or more device or
20 software drivers enabling processor 2102 to drive one or more of these I/O devices. I/O
21 interface 2108 may include one or more I/O interfaces 2108, where appropriate. Although
22 this disclosure describes and illustrates a particular I/O interface, this disclosure contemplates
23 any suitable I/O interface.

24 [141] In particular embodiments, communication interface 2110 includes hardware,
25 software, or both providing one or more interfaces for communication (such as, for example,
26 packet-based communication) between computer system 2100 and one or more other
27 computer systems 2100 or one or more networks. As an example and not by way of
28 limitation, communication interface 2110 may include a network interface controller (NIC) or
29 network adapter for communicating with an Ethernet or other wire-based network or a
30 wireless NIC (WNIC) or wireless adapter for communicating with a wireless network, such
31 as a WI-FI network. This disclosure contemplates any suitable network and any suitable
32 communication interface 2110 for it. As an example and not by way of limitation, computer
33 system 2100 may communicate with an ad hoc network, a personal area network (PAN), a

1 local area network (LAN), a wide area network (WAN), a metropolitan area network (MAN),
2 or one or more portions of the Internet or a combination of two or more of these. One or more
3 portions of one or more of these networks may be wired or wireless. As an example,
4 computer system 2100 may communicate with a wireless PAN (WPAN) (such as, for
5 example, a BLUETOOTH WPAN), a WI-FI network, a WI-MAX network, a cellular
6 telephone network (such as, for example, a Global System for Mobile Communications
7 (GSM) network), or other suitable wireless network or a combination of two or more of
8 these. Computer system 2100 may include any suitable communication interface 2110 for
9 any of these networks, where appropriate. Communication interface 2110 may include one or
10 more communication interfaces 2110, where appropriate. Although this disclosure describes
11 and illustrates a particular communication interface, this disclosure contemplates any suitable
12 communication interface.

13 [142] In particular embodiments, bus 2112 includes hardware, software, or both
14 coupling components of computer system 2100 to each other. As an example and not by way
15 of limitation, bus 2112 may include an Accelerated Graphics Port (AGP) or other graphics
16 bus, an Enhanced Industry Standard Architecture (EISA) bus, a front-side bus (FSB), a
17 HYPERTRANSPORT (HT) interconnect, an Industry Standard Architecture (ISA) bus, an
18 INFINIBAND interconnect, a low-pin-count (LPC) bus, a memory bus, a Micro Channel
19 Architecture (MCA) bus, a Peripheral Component Interconnect (PCI) bus, a PCI-Express
20 (PCIe) bus, a serial advanced technology attachment (SATA) bus, a Video Electronics
21 Standards Association local (VLB) bus, or another suitable bus or a combination of two or
22 more of these. Bus 2112 may include one or more buses 2112, where appropriate. Although
23 this disclosure describes and illustrates a particular bus, this disclosure contemplates any
24 suitable bus or interconnect.

25 [143] Herein, a computer-readable non-transitory storage medium or media may
26 include one or more semiconductor-based or other integrated circuits (ICs) (such, as for
27 example, field-programmable gate arrays (FPGAs) or application-specific ICs (ASICs)), hard
28 disk drives (HDDs), hybrid hard drives (HHDs), optical discs, optical disc drives (ODDs),
29 magneto-optical discs, magneto-optical drives, floppy diskettes, floppy disk drives (FDDs),
30 magnetic tapes, solid-state drives (SSDs), RAM-drives, SECURE DIGITAL cards or drives,
31 any other suitable computer-readable non-transitory storage media, or any suitable
32 combination of two or more of these, where appropriate. A computer-readable non-transitory
33 storage medium may be volatile, non-volatile, or a combination of volatile and non-volatile,

1 where appropriate.

2 **[144]** Herein, “or” is inclusive and not exclusive, unless expressly indicated
3 otherwise or indicated otherwise by context. Therefore, herein, “A or B” means “A, B, or
4 both,” unless expressly indicated otherwise or indicated otherwise by context. Moreover,
5 “and” is both joint and several, unless expressly indicated otherwise or indicated otherwise by
6 context. Therefore, herein, “A and B” means “A and B, jointly or severally,” unless expressly
7 indicated otherwise or indicated otherwise by context.

8 The scope of this disclosure encompasses all changes, substitutions, variations, alterations,
9 and modifications to the example embodiments described or illustrated herein that a person
10 having ordinary skill in the art would comprehend. The scope of this disclosure is not limited
11 to the example embodiments described or illustrated herein. Moreover, although this
12 disclosure describes and illustrates respective embodiments herein as including particular
13 components, elements, feature, functions, operations, or steps, any of these embodiments may
14 include any combination or permutation of any of the components, elements, features,
15 functions, operations, or steps described or illustrated anywhere herein that a person having
16 ordinary skill in the art would comprehend. Furthermore, reference in the appended claims to
17 an apparatus or system or a component of an apparatus or system being adapted to, arranged
18 to, capable of, configured to, enabled to, operable to, or operative to perform a particular
19 function encompasses that apparatus, system, component, whether or not it or that particular
20 function is activated, turned on, or unlocked, as long as that apparatus, system, or component
21 is so adapted, arranged, capable, configured, enabled, operable, or operative. Additionally,
22 although this disclosure describes or illustrates particular embodiments as providing
23 particular advantages, particular embodiments may provide none, some, or all of these
24 advantages.

25

CLAIMS

What is claimed is:

1. A method comprising, by a computing system:
 - receiving a block comprising a plurality of pixels;
 - encoding the plurality of pixels by:
 - arranging the plurality of pixels in a sequence;
 - generating a delta encoding of the plurality of pixels, the delta encoding comprising (a) a base value and (b) a plurality of delta values having non-zero delta values and zero delta values, each delta value representing a difference between a corresponding pixel in the sequence and a previous pixel in the sequence;
 - generating a symbol mask indicating whether each of the plurality of delta values is zero or non-zero;
 - determining, based on magnitudes of the non-zero delta values, a symbol width for encoding each of the non-zero delta values;
 - generating a sequence of symbols that respectively encode the non-zero delta values using the symbol width;
 - generating a compression of the block by collating the symbol mask, the symbol width, and the sequence of symbols.
2. The method of Claim 1, wherein the plurality of pixels of the block correspond to a single channel of pixel values amongst a plurality channels of pixel values.
3. The method of Claim 1 or 2, wherein arranging the plurality of pixels in a sequence is based on Morton Order.
4. The method of any one of the preceding Claims, further comprising:
 - determining that a bit length associated with the compression of the block is more than a bit length associated with the plurality of the pixels of the block as uncompressed; and
 - storing the plurality of pixels of the block as uncompressed.
5. The method of any one of the preceding Claims, wherein the symbol width for encoding each of the non-zero delta values includes an offset.
6. The method of any one of the preceding Claims, further comprising modifying the compression of the block to be byte-aligned by adding additional bits to the compression of the block.
7. The method of any one of the preceding Claims, further comprising decoding the compression of the block by processing two or more delta values of the plurality of delta

1 values in parallel.

2 8. One or more computer-readable non-transitory storage media including instructions
3 that, when executed by one or more processors, are configured to cause the one or more
4 processors to:

5 receive a block comprising a plurality of pixels;

6 encode the plurality of pixels by:

7 arranging the plurality of pixels in a sequence;

8 generating a delta encoding of the plurality of pixels, the delta encoding
9 comprising (a) a base value and (b) a plurality of delta values having non-zero delta
10 values and zero delta values, each delta value representing a difference between a
11 corresponding pixel in the sequence and a previous pixel in the sequence;

12 generating a symbol mask indicating whether each of the plurality of delta
13 values is zero or non-zero;

14 determining, based on magnitudes of the non-zero delta values, a symbol
15 width for encoding each of the non-zero delta values;

16 generating a sequence of symbols that respectively encode the non-zero delta
17 values using the symbol width;

18 generating a compression of the block by collating the symbol mask, the
19 symbol width, and the sequence of symbols.

20 9. The one or more computer-readable non-transitory storage media of Claim 8,
21 wherein the plurality of pixels of the block correspond to a single channel of pixel values
22 amongst a plurality channels of pixel values.

23 10. The one or more computer-readable non-transitory storage media of Claim 8 or 9,
24 wherein arranging the plurality of pixels in a sequence is based on Morton Order.

25 11. The one or more computer-readable non-transitory storage media of any one of
26 Claims 8 to 10, wherein the instructions are configured to further cause the one or more
27 processors to:

28 determine that a bit length associated with the compression of the block is more than a
29 bit length associated with the plurality of the pixels of the block as uncompressed; and

30 store the plurality of pixels of the block as uncompressed.

31 12. The one or more computer-readable non-transitory storage media of any one of
32 Claims 8 to 11, wherein the symbol width for encoding each of the non-zero delta values
33 includes an offset.

1 13. The one or more computer-readable non-transitory storage media of any one of
2 Claims 8 to 12, wherein the instructions are configured to further cause the one or more
3 processors to:

4 modify the compression of the block to be byte-aligned by adding additional bits to
5 the compression of the block.

6 14. The one or more computer-readable non-transitory storage media of any one of
7 Claims 8 to 13, wherein the instructions are configured to further cause the one or more
8 processors to:

9 decode the compression of the block by processing two or more delta values of the
10 plurality of delta values in parallel.

11 15. A system comprising: one or more processors; and one or more computer-
12 readable non-transitory storage media in communication with the one or more processors
13 according to any one of claims 8 to 14.

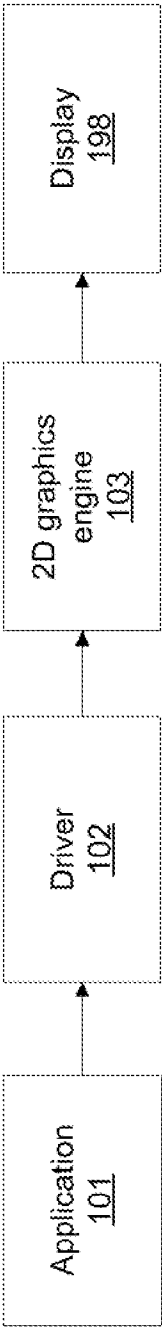


FIG. 1A

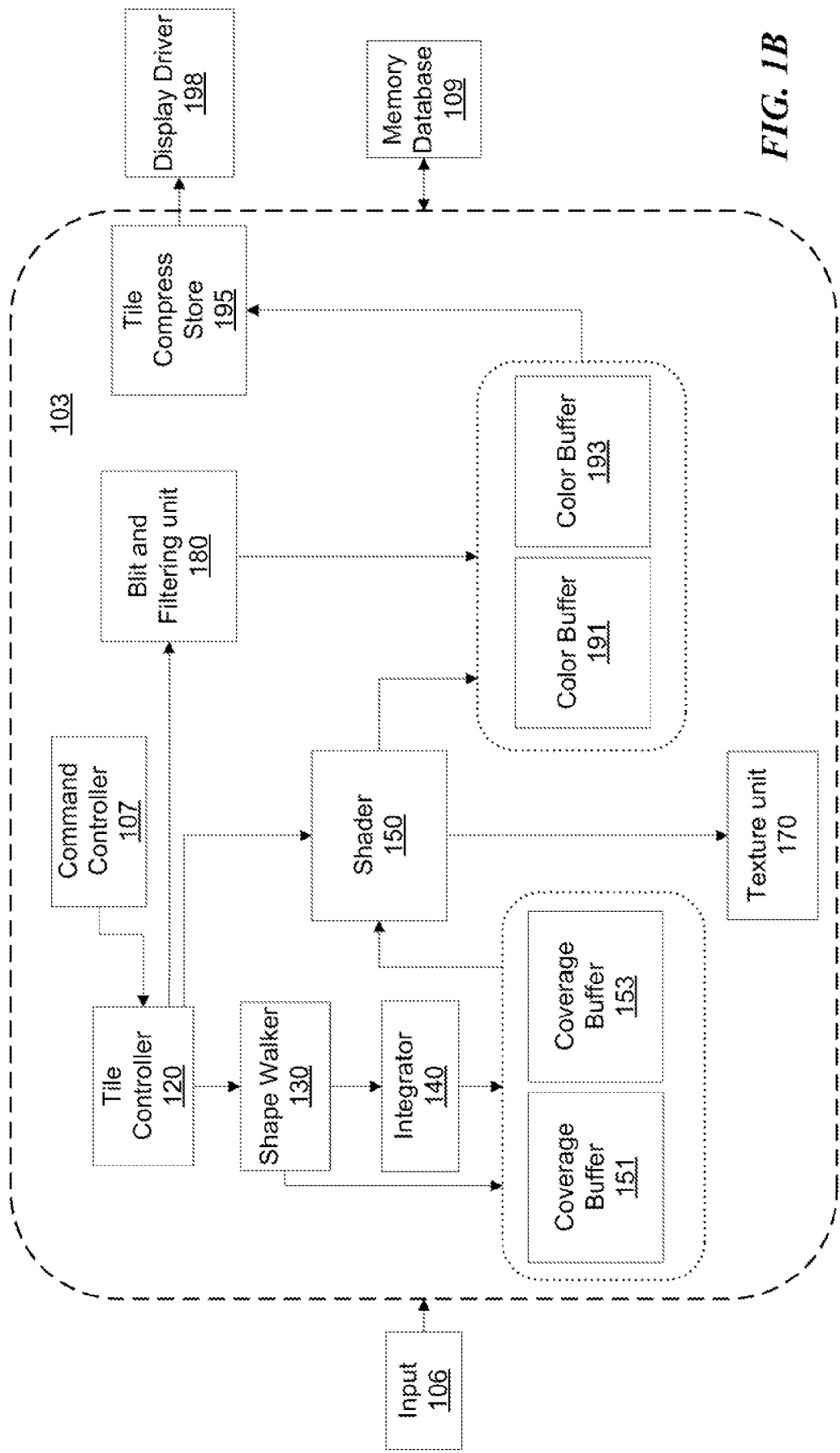


FIG. 1B

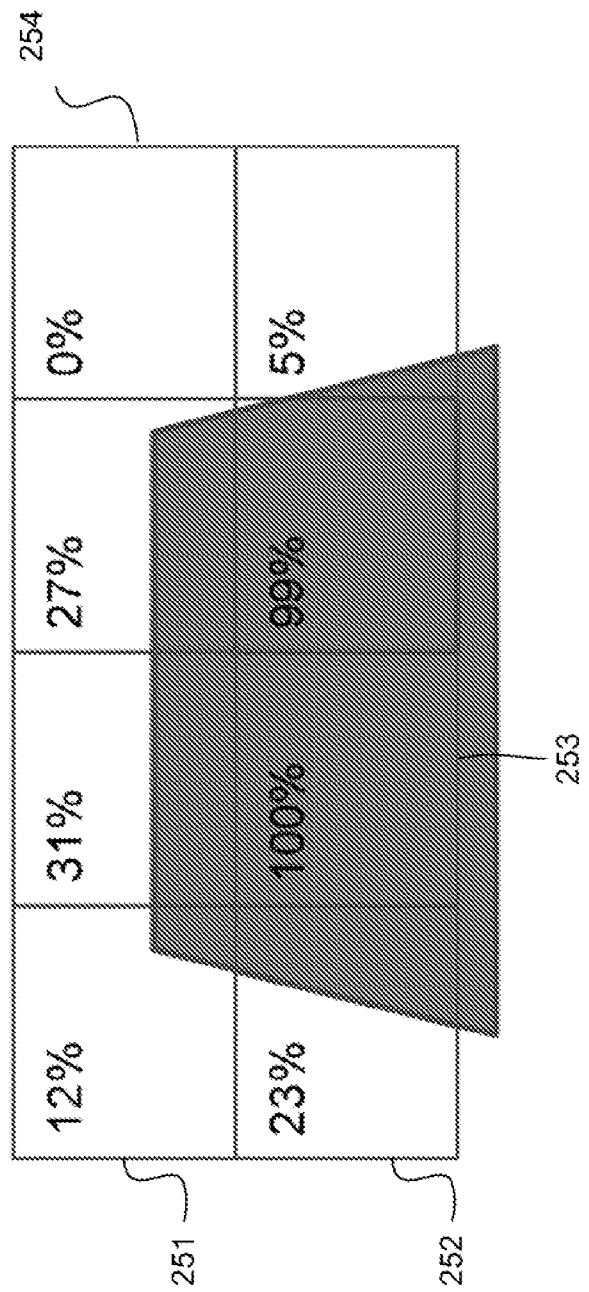
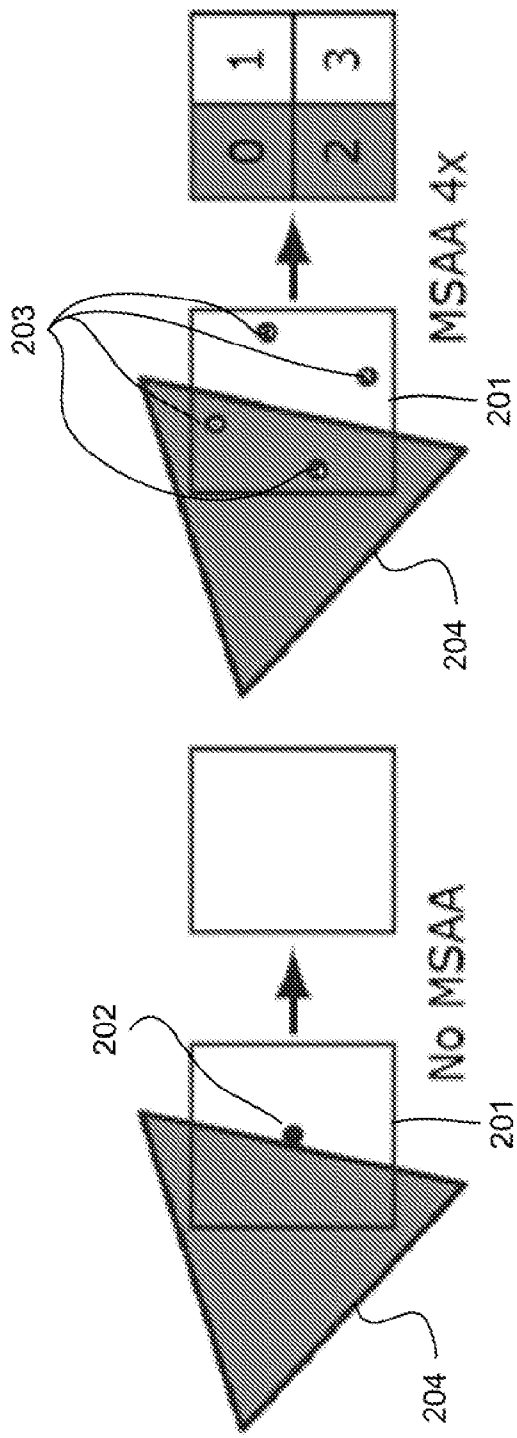


FIG. 2

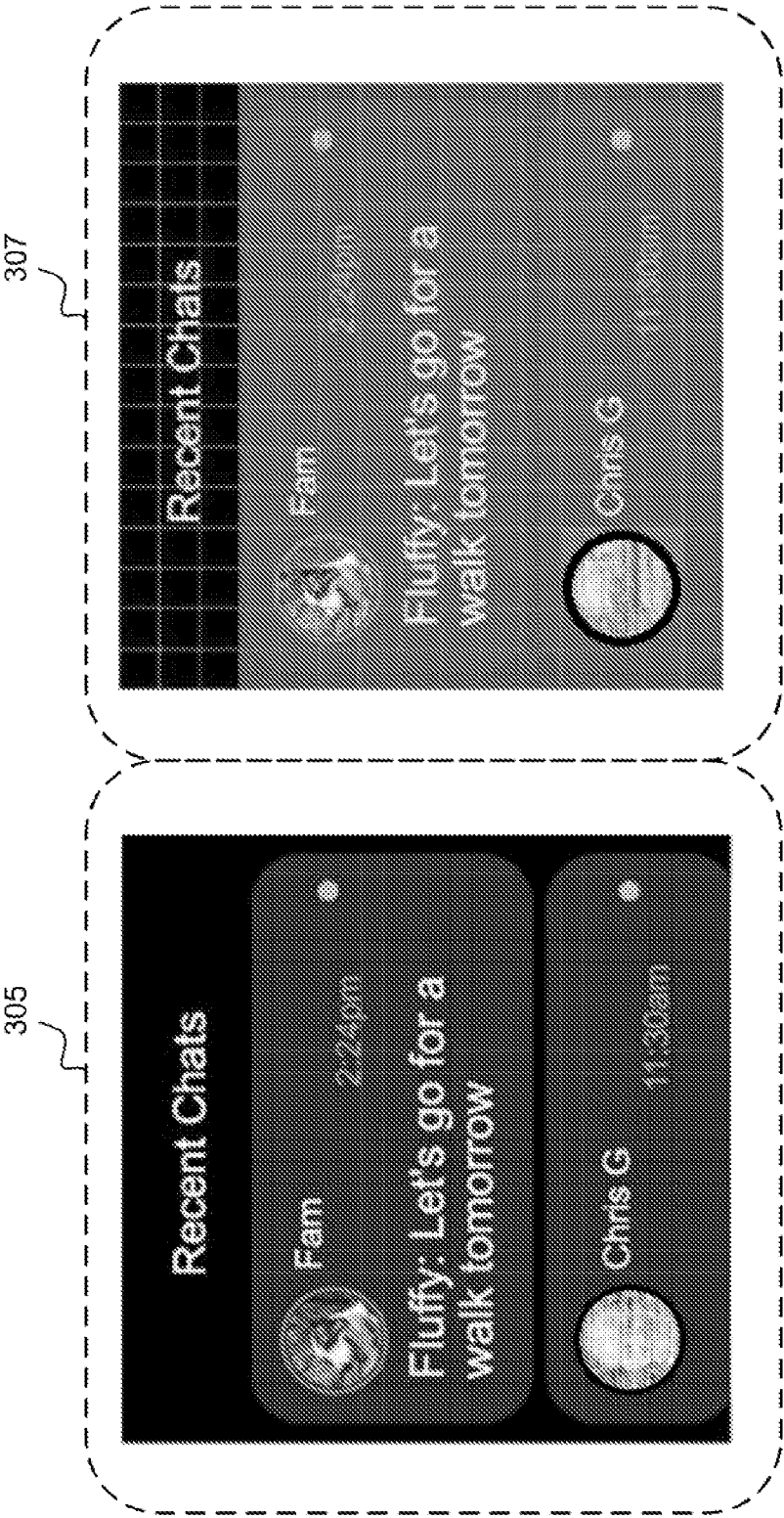


FIG. 3A

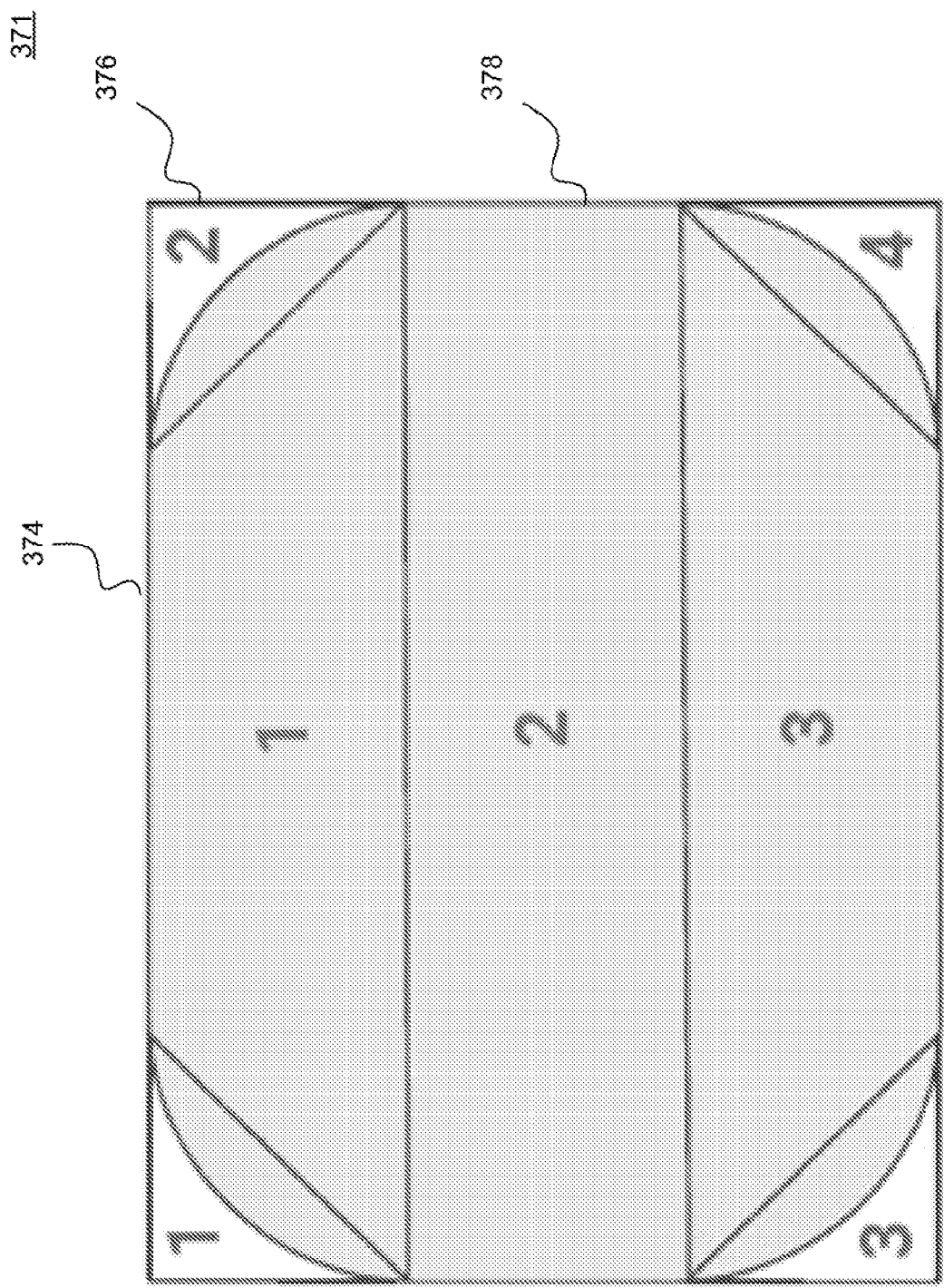
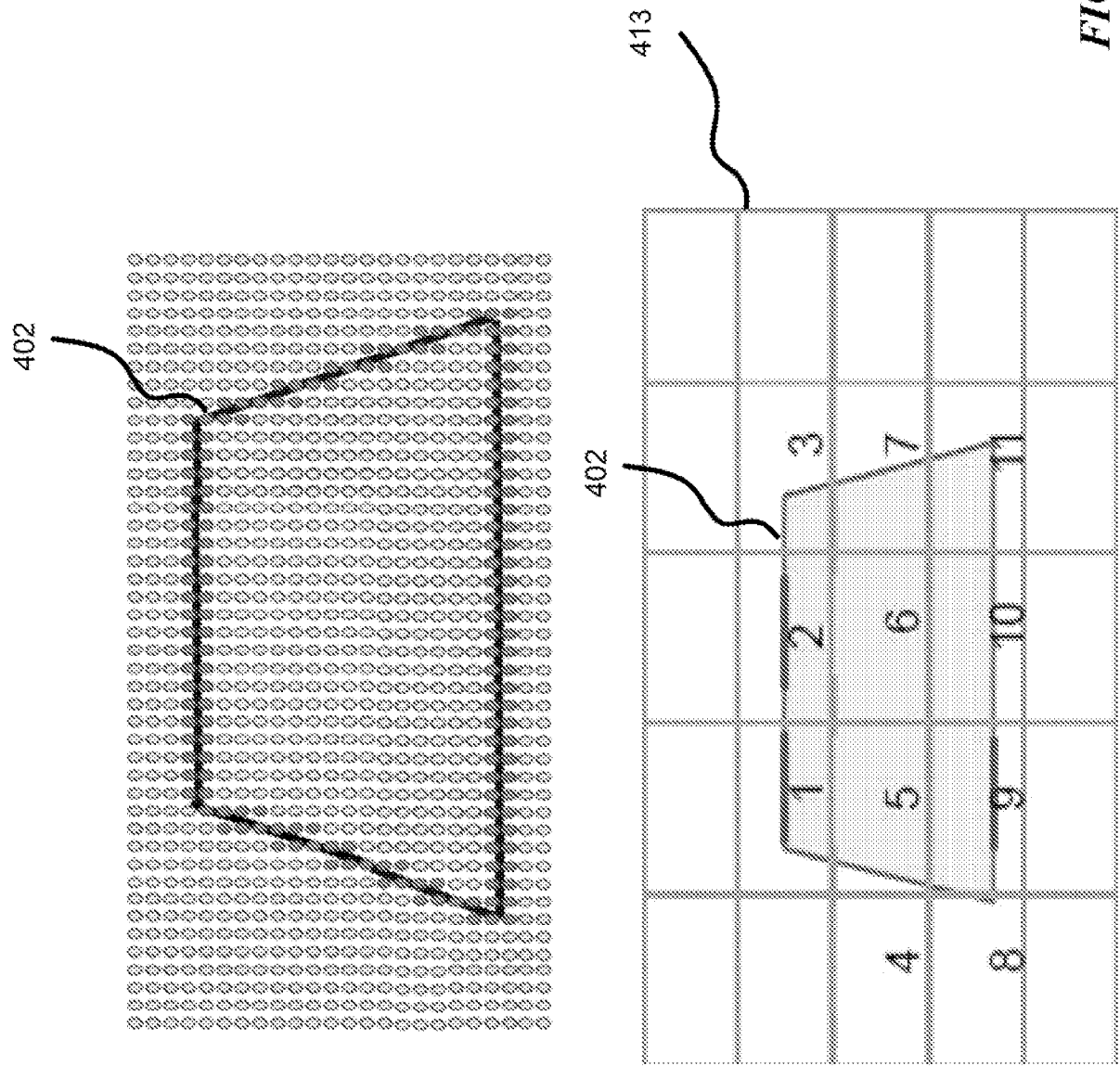


FIG. 3B



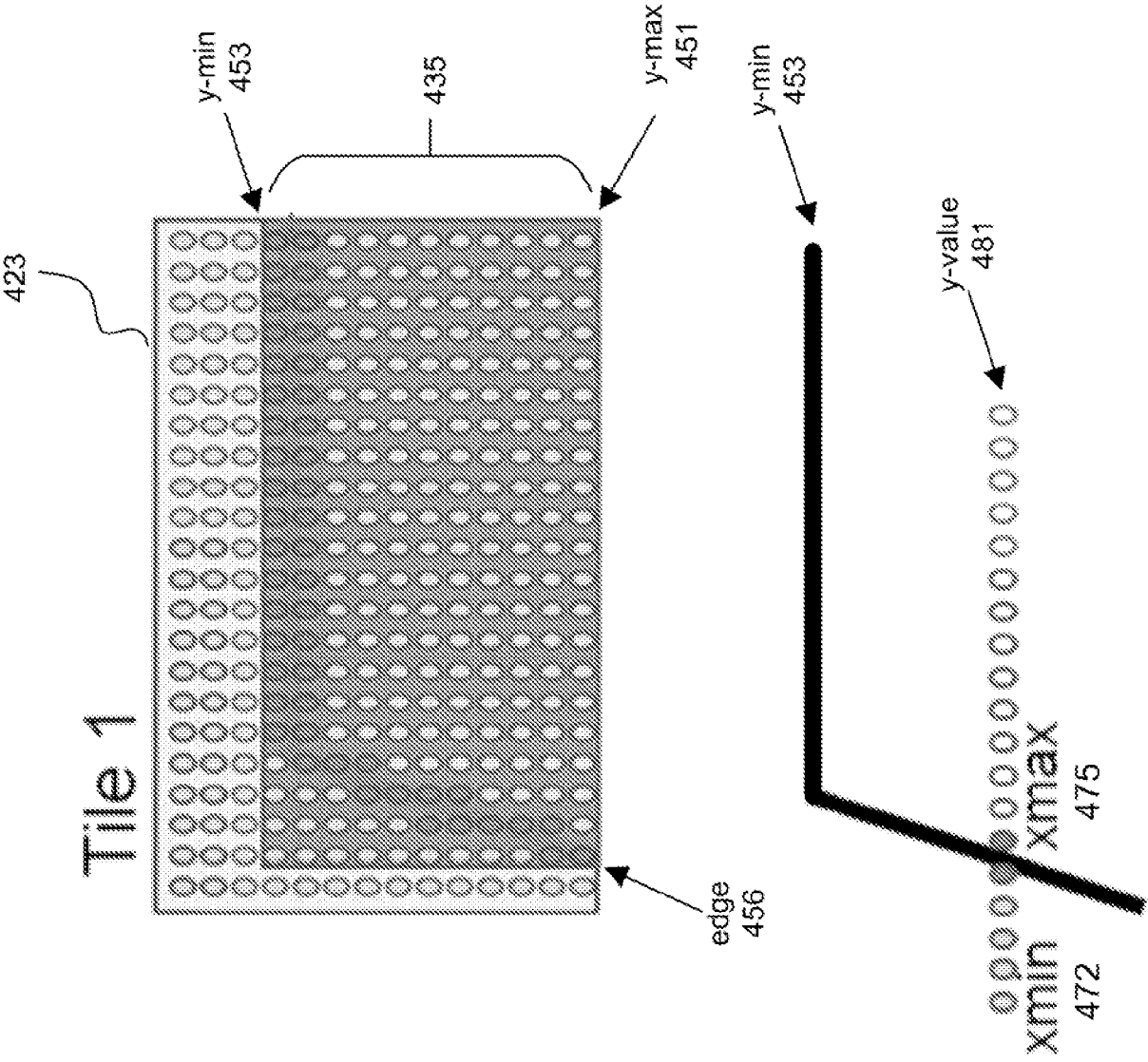


FIG. 4B

Frame

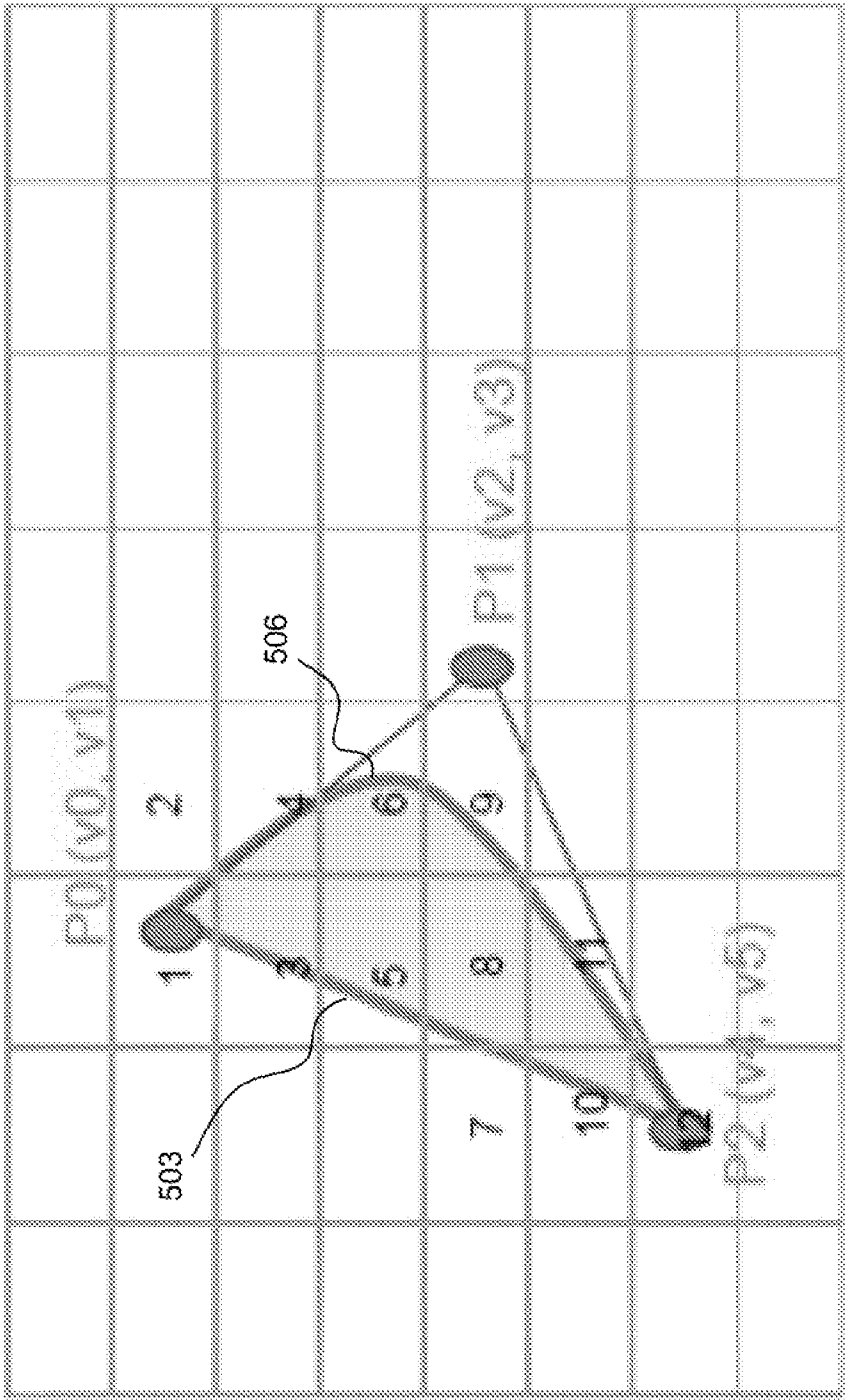


FIG. 5A

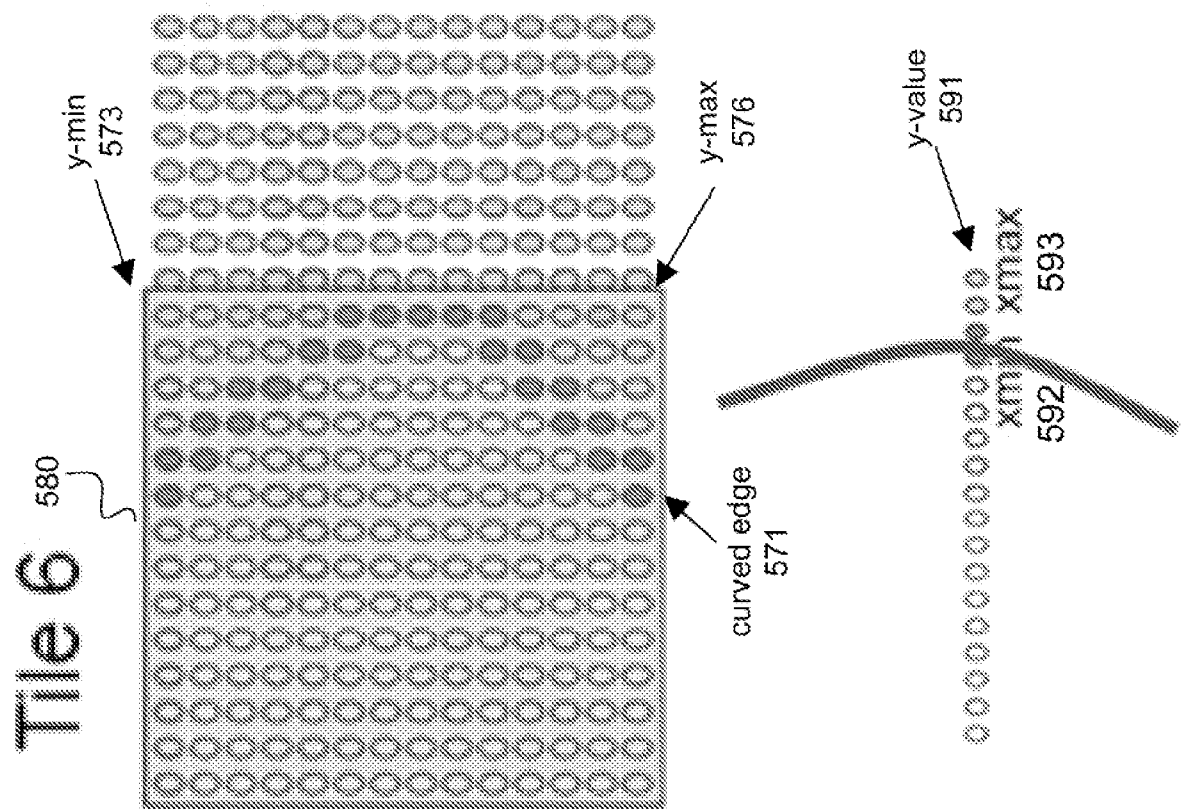


FIG. 5B

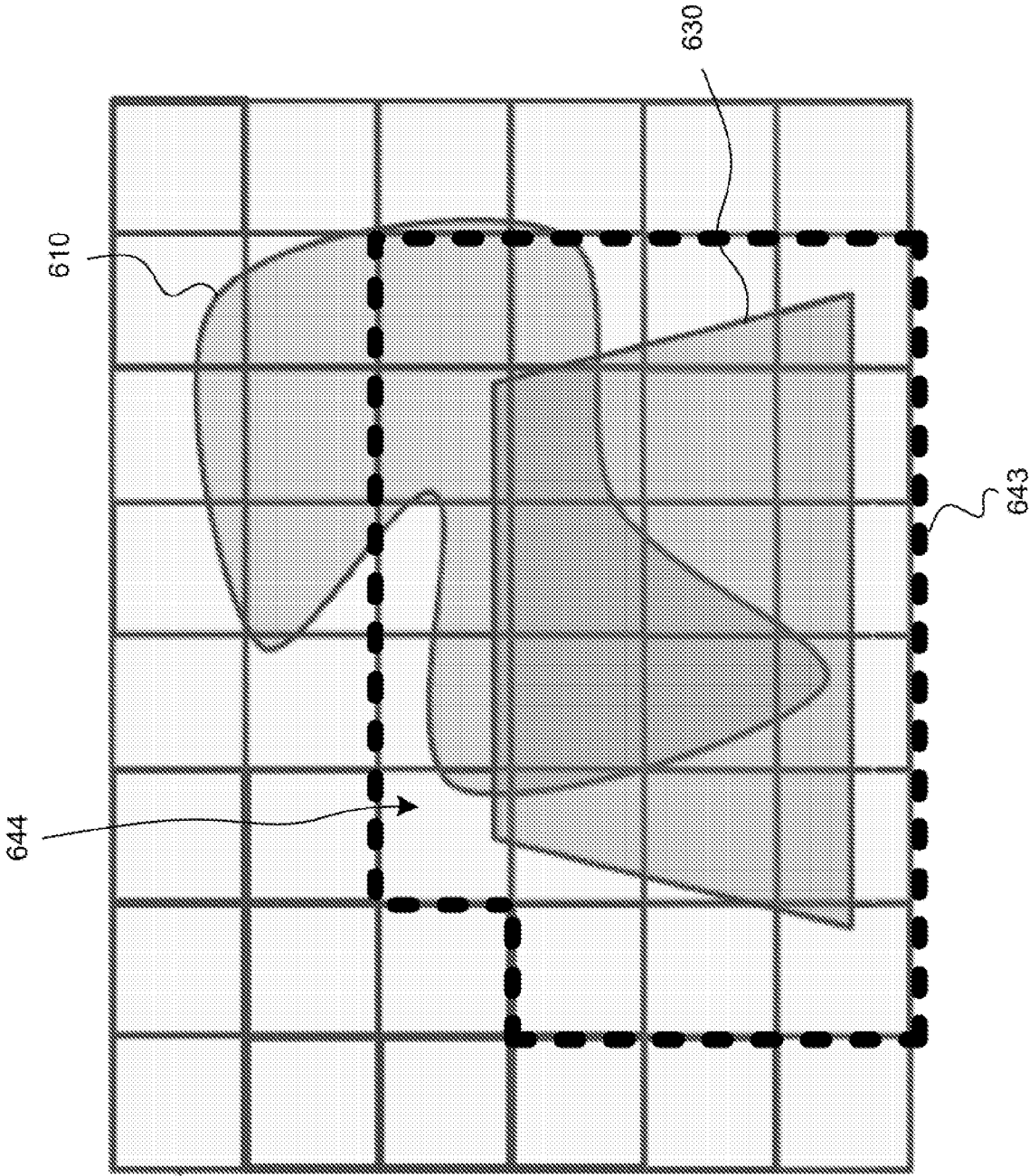


FIG. 6

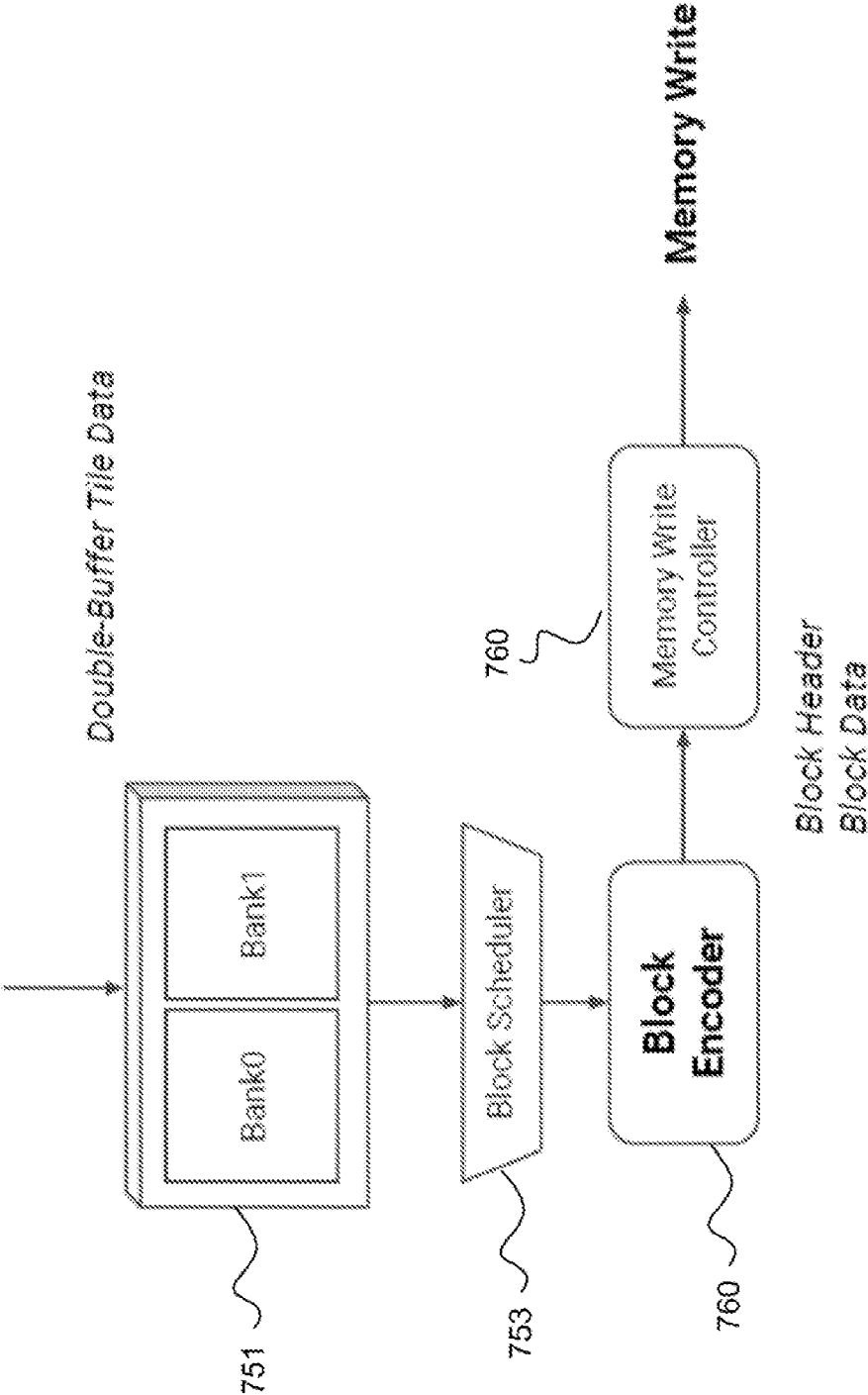


FIG. 7

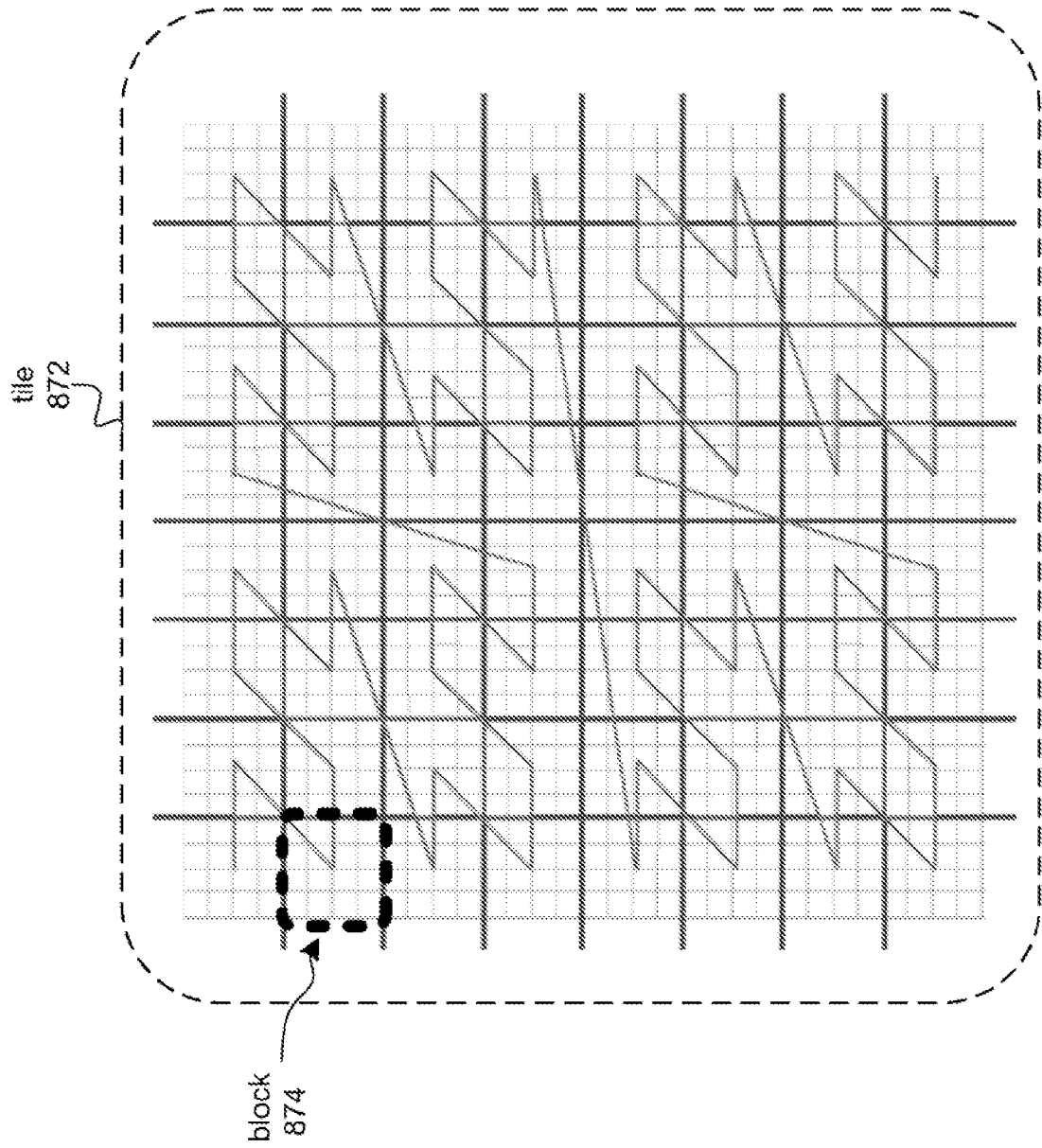


FIG. 8

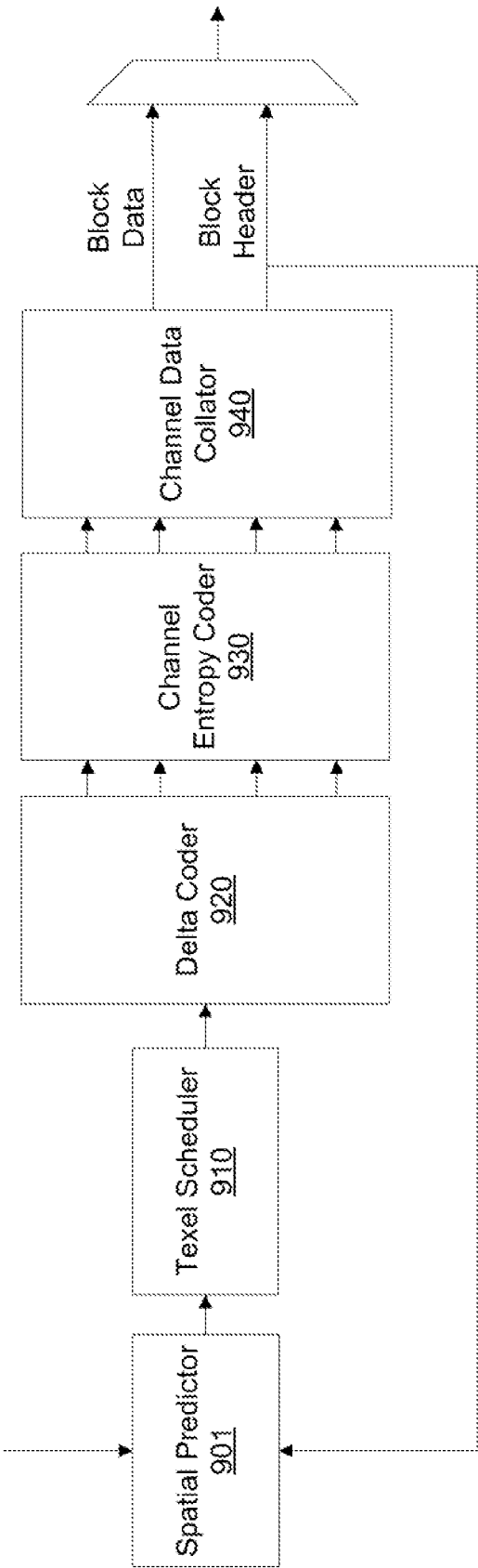


FIG. 9

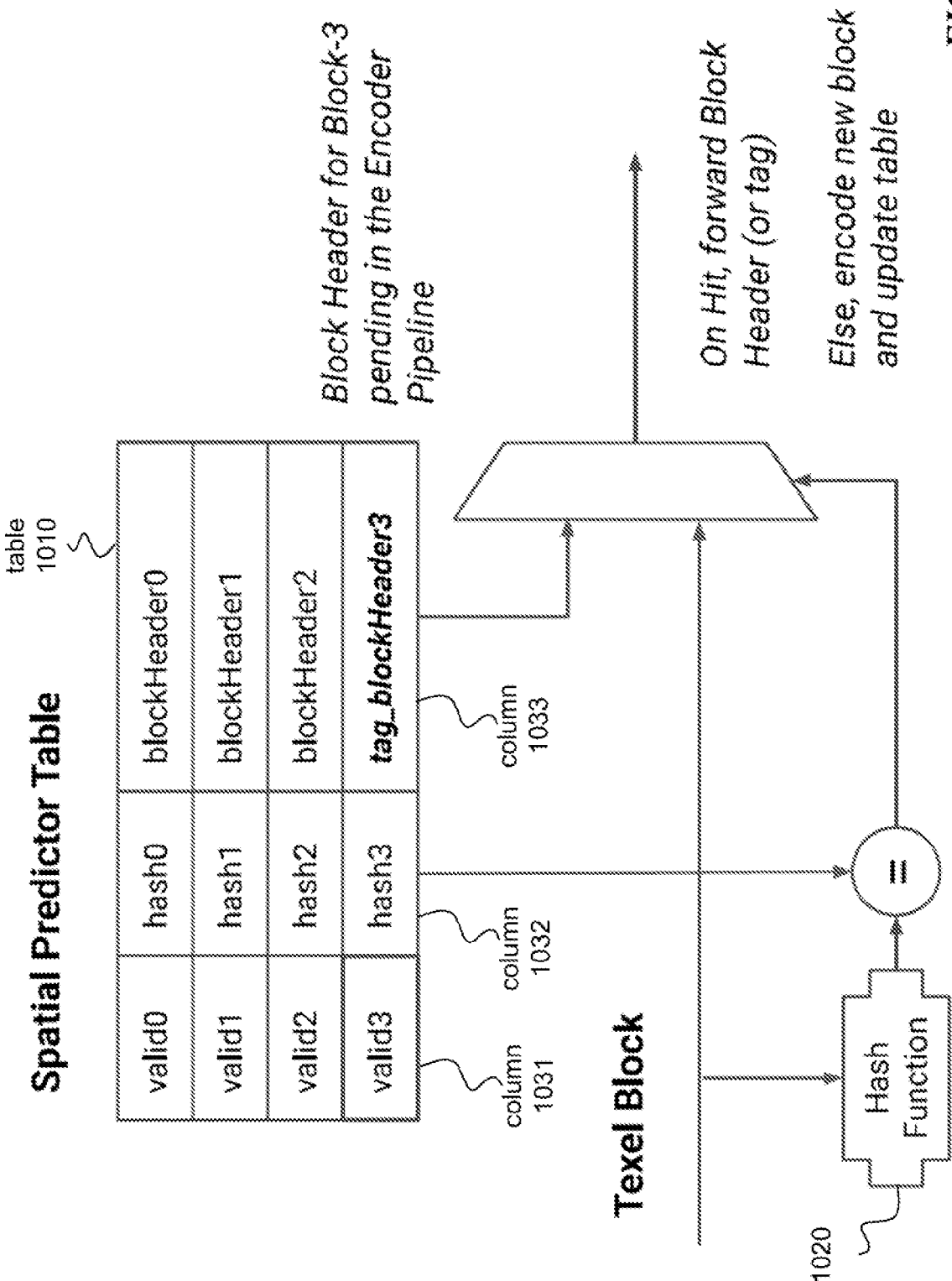


FIG. 10

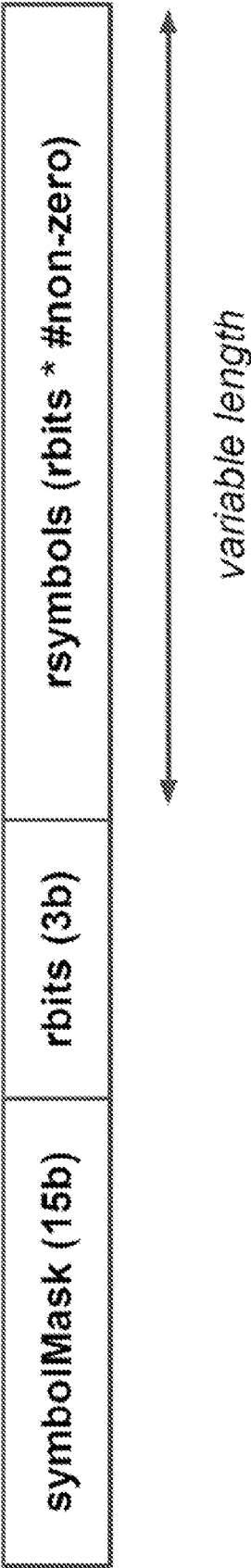


FIG. 11

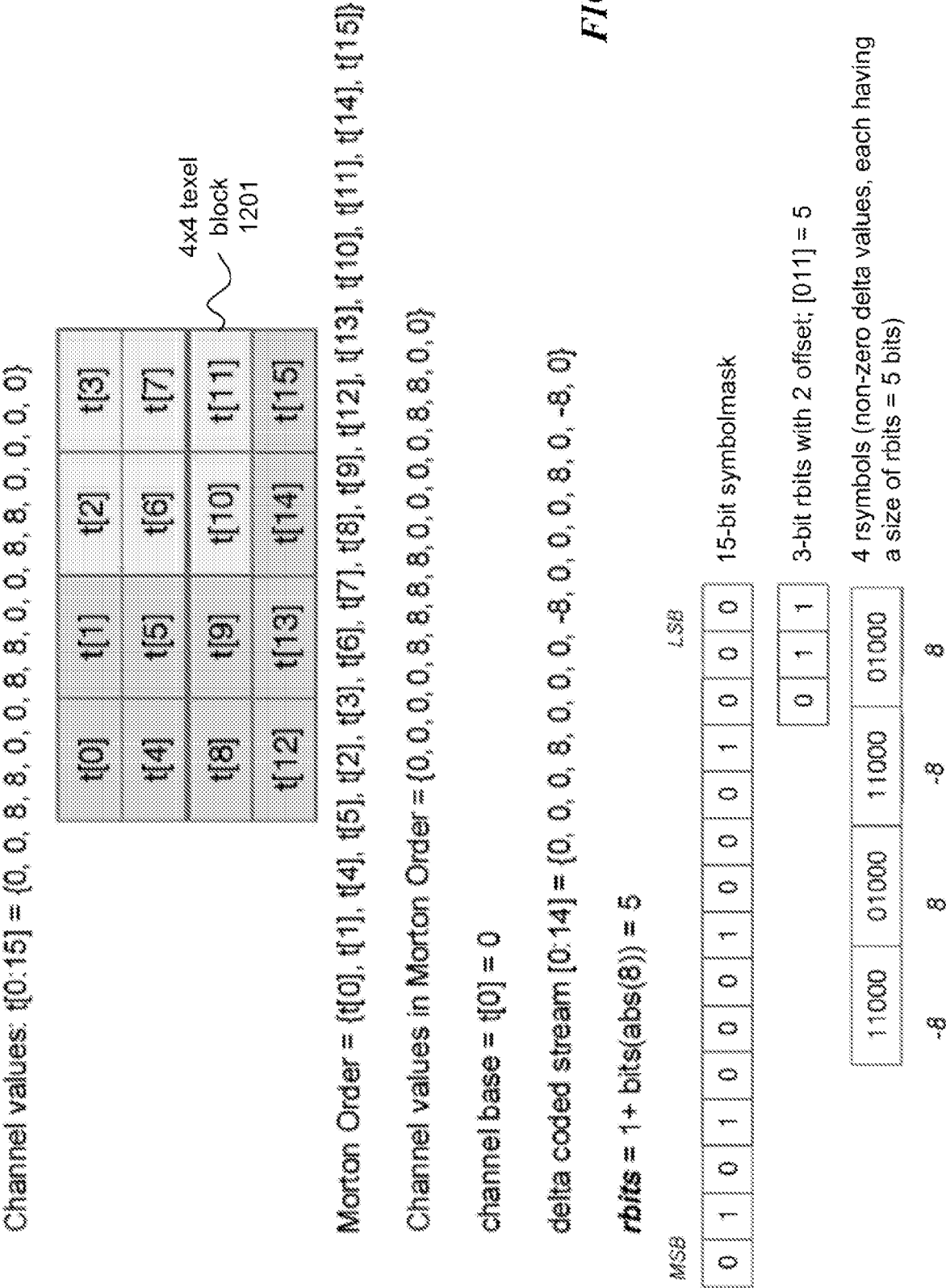


FIG. 12

2 extra bits of padding may be added to the 38 bits of encoded data to make it byte-aligned, e.g., to 40 bits.

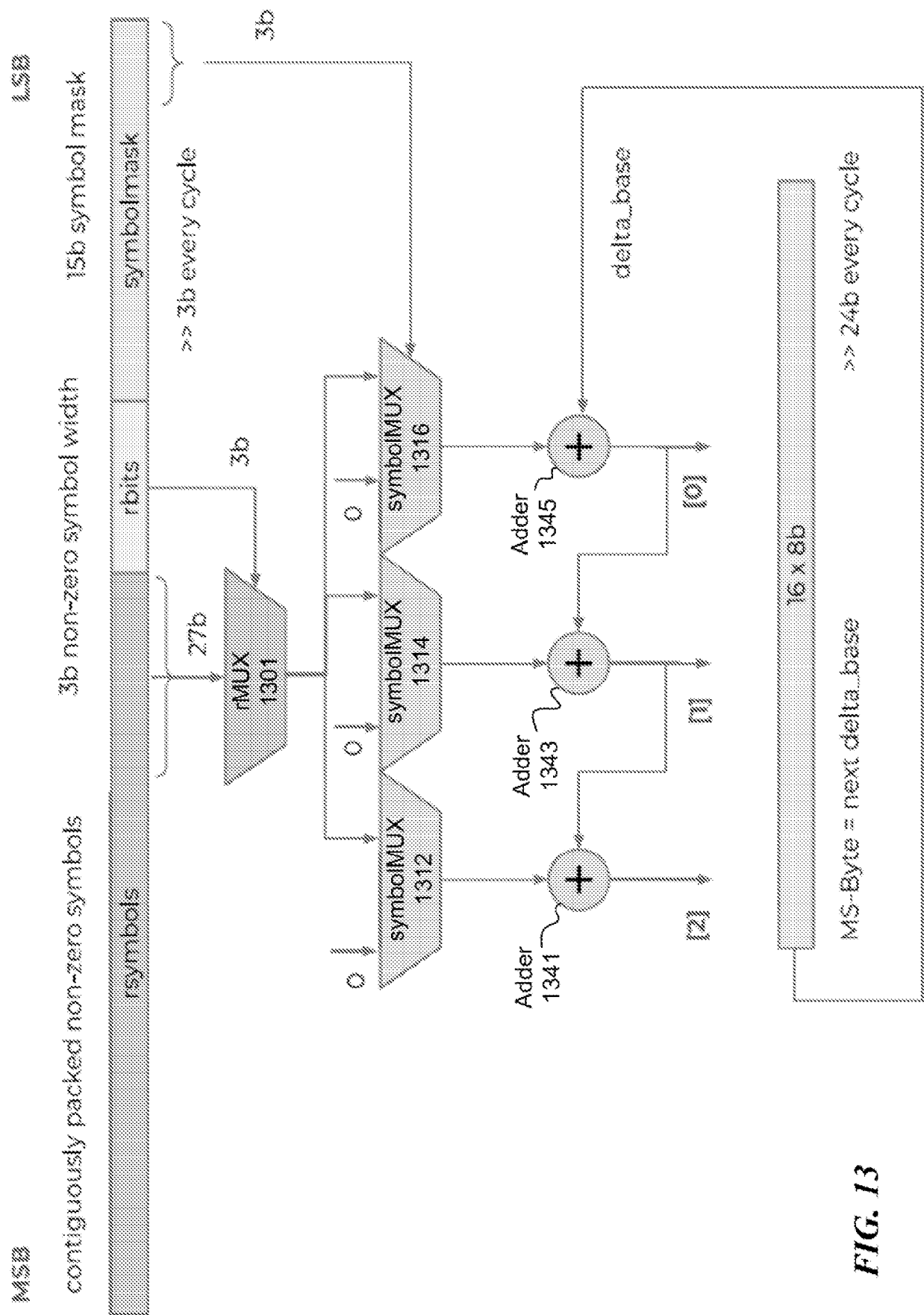
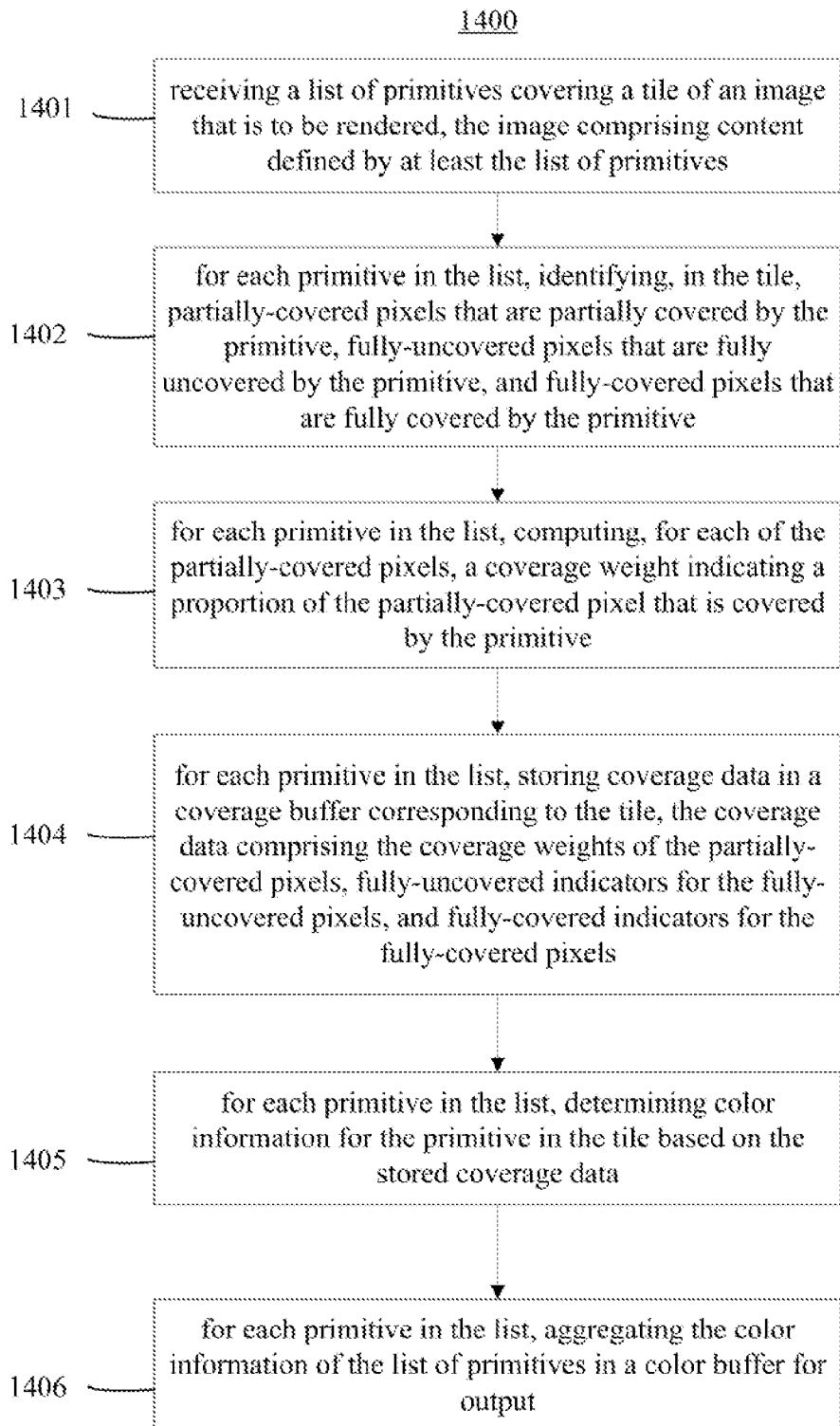
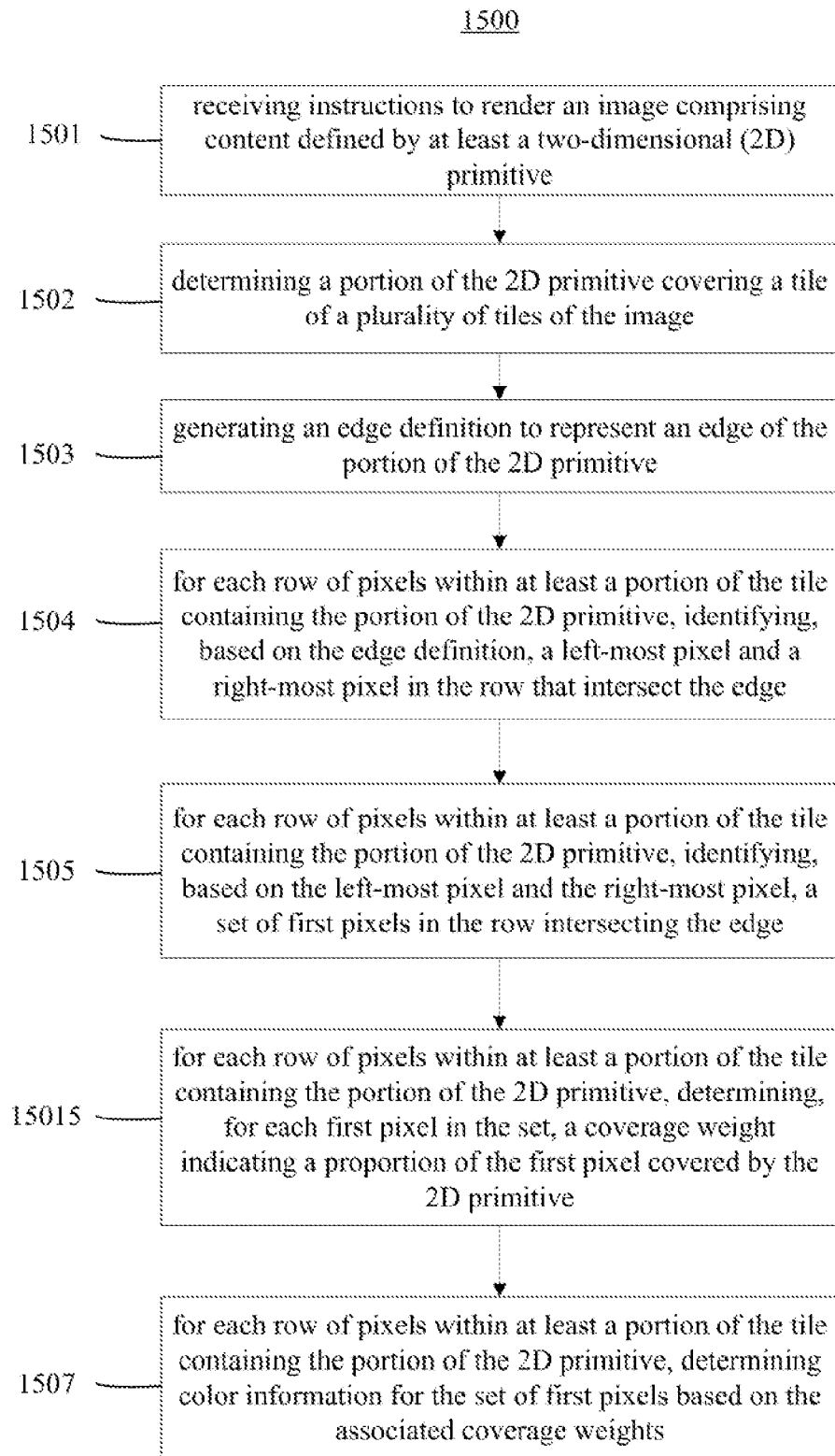
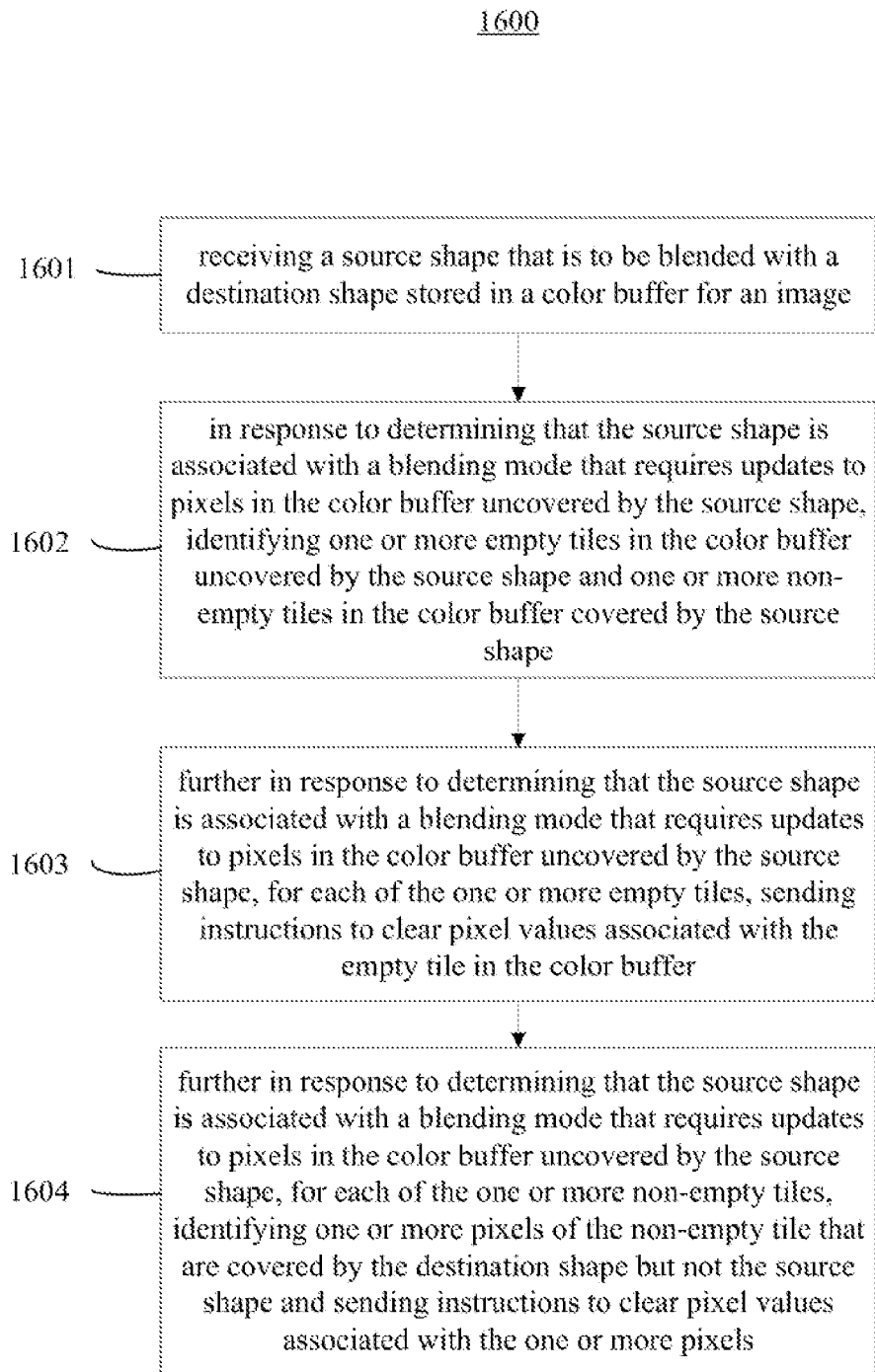
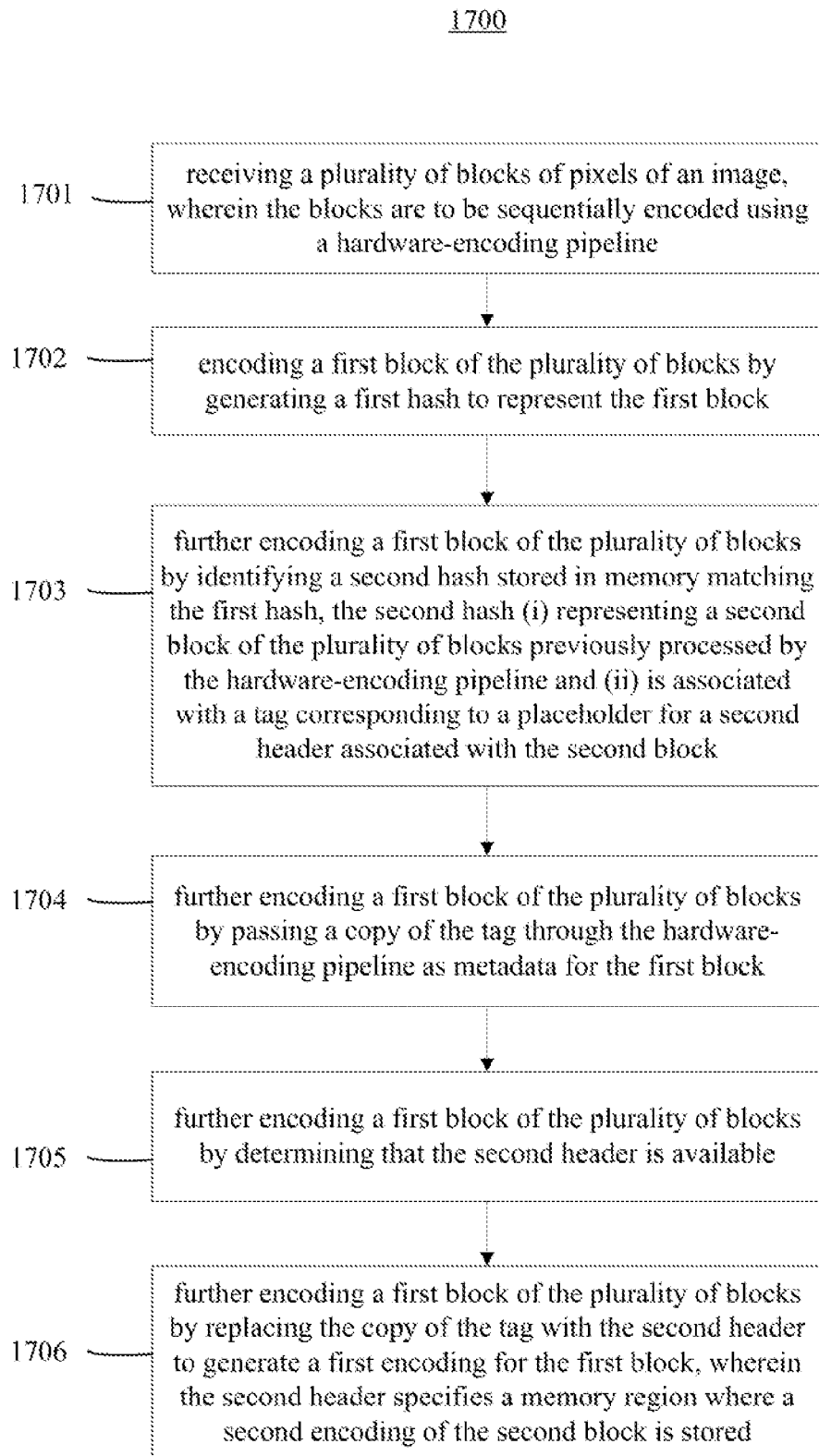


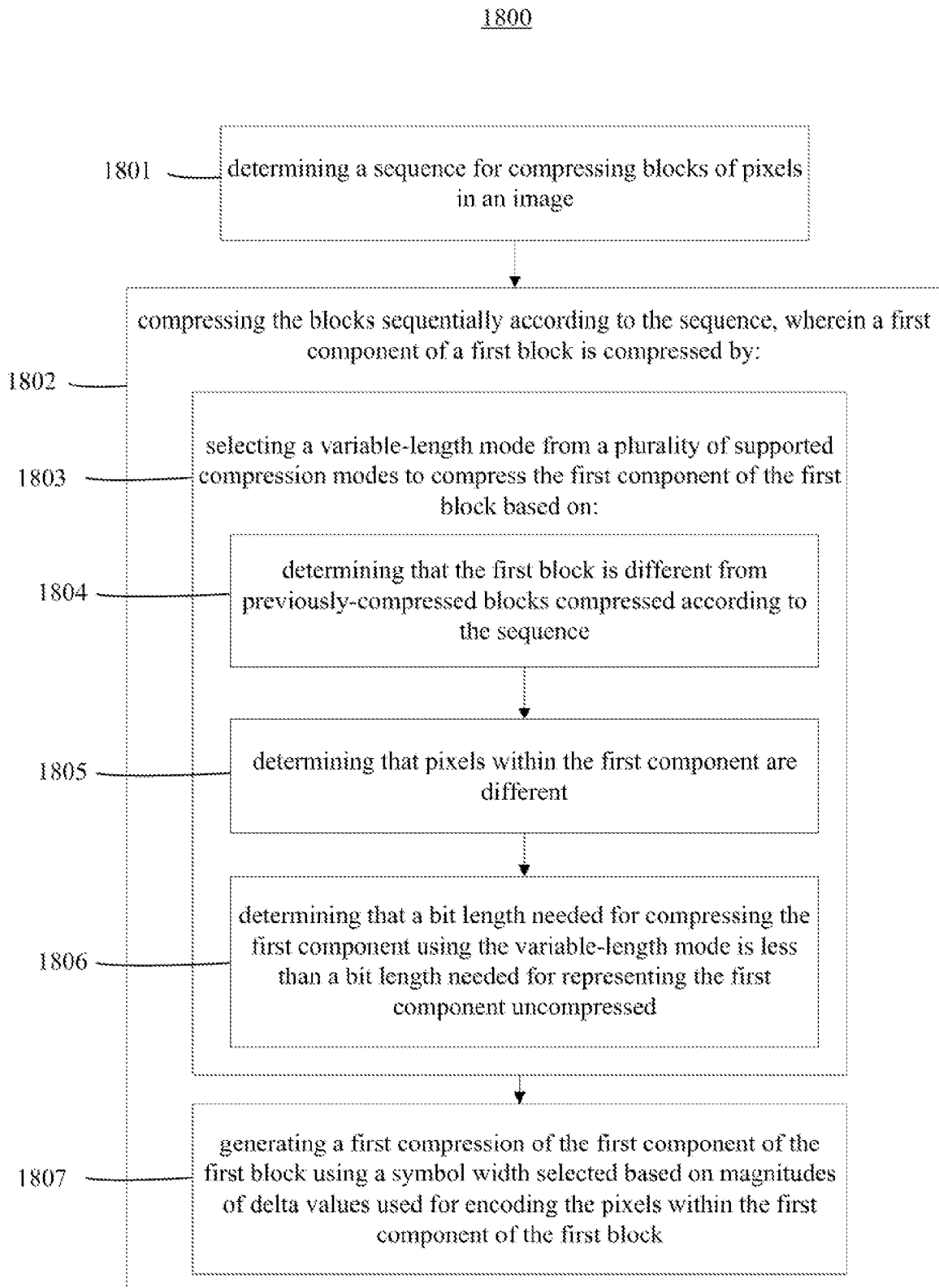
FIG. 13

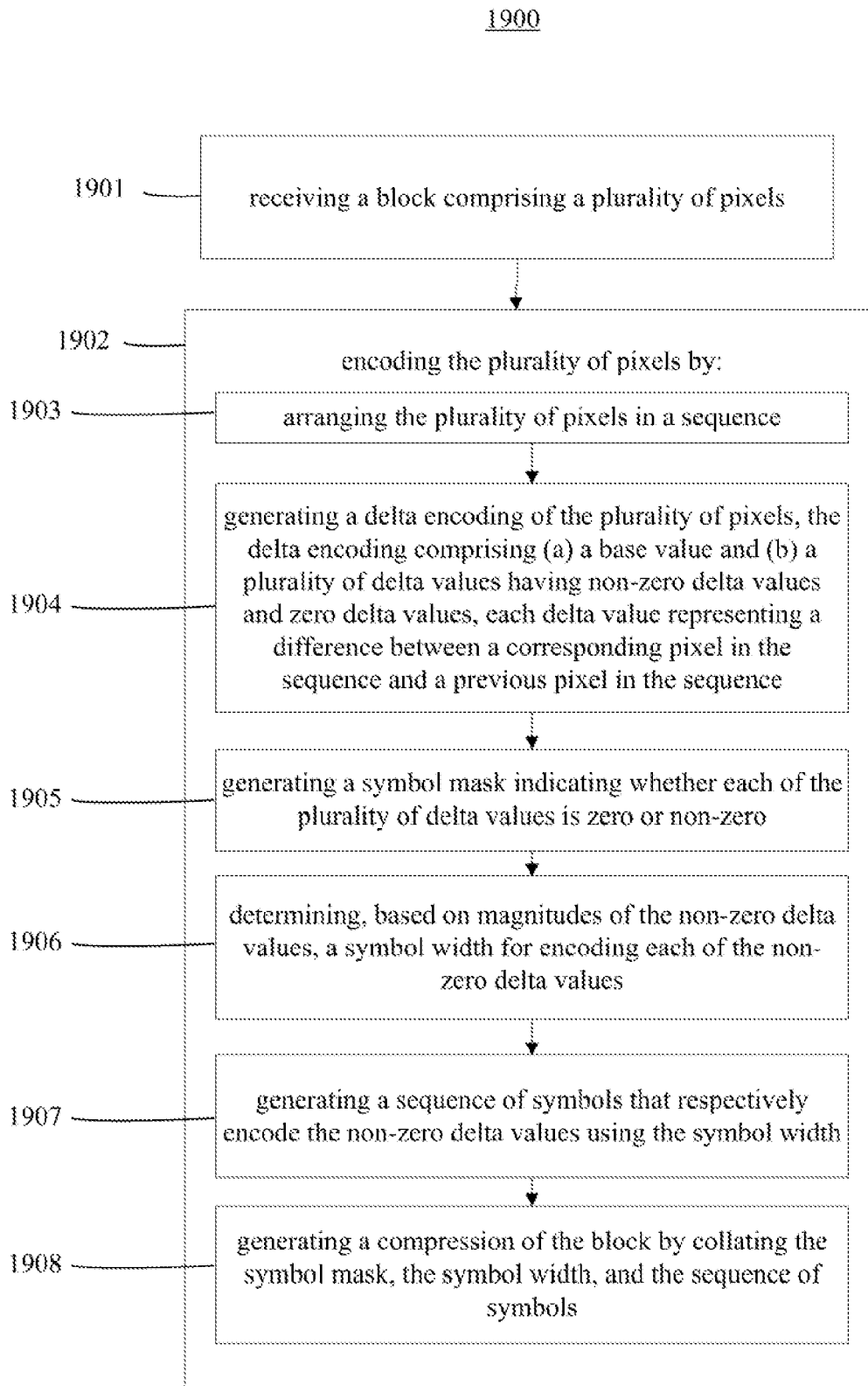
**FIG. 14**

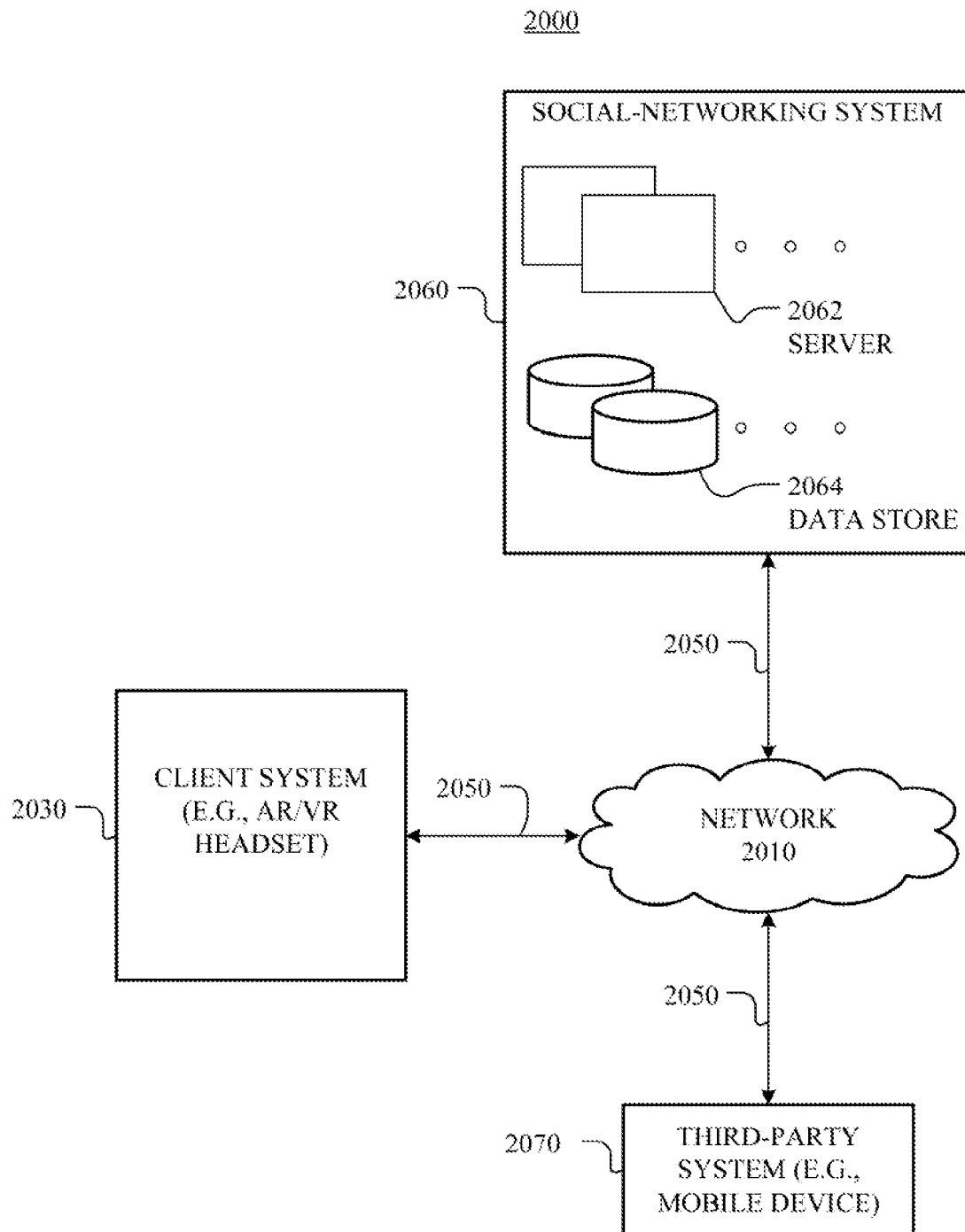
**FIG. 15**

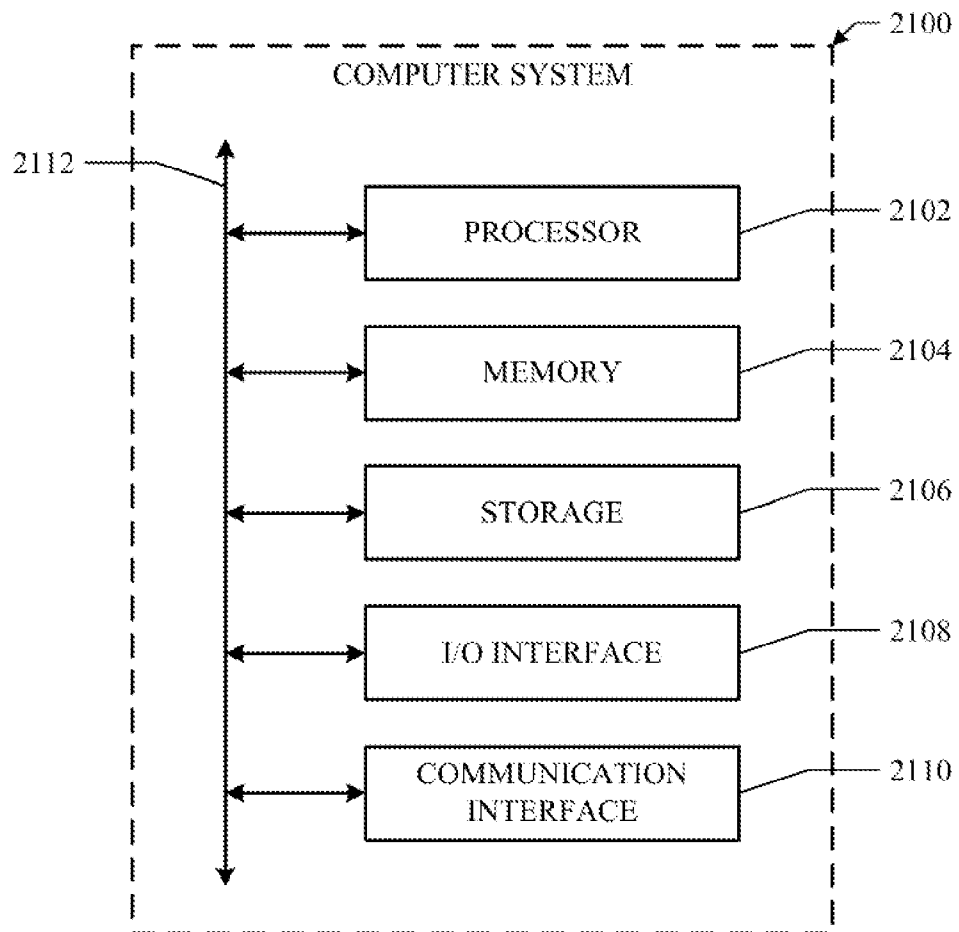
**FIG. 16**

**FIG. 17**

**FIG. 18**

**FIG. 19**

**FIG. 20**

**FIG. 21**

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/018747

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06T9/20 H04N19/436 H04N19/593
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06T H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	PAJAROLA R ET AL: "AN IMAGE COMPRESSION METHOD FOR SPATIAL SEARCH", IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE, USA, vol. 9, no. 3, 1 March 2000 (2000-03-01), pages 357-365, XP000908668, ISSN: 1057-7149, DOI: 10.1109/83.826774 the whole document -----	1-15

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

5 July 2023

Date of mailing of the international search report

14/07/2023

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Oelbaum, Tobias