



(51) International Patent Classification:
G06T 11/40 (2006.01)

(21) International Application Number:
PCT/US2023/018754

(22) International Filing Date:
16 April 2023 (16.04.2023)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
17/721,653 15 April 2022 (15.04.2022) US

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

(72) Inventors: **KAVILIPATI, Siddartha**; 1601 Willow Road, Menlo Park, CA 94025 (US). **GOODMAN, Christopher, James**; 1601 Willow Road, Menlo Park, CA 94025 (US). **DURFEE, Kyle**; 1601 Willow Road, Menlo Park, CA 94025 (US). **GOSWAMI, Nilanjan**; 1601 Willow Road, Menlo Park, CA 94025 (US).

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

(54) Title: RASTERIZATION OPTIMIZATION FOR ANALYTIC ANTI-ALIASING

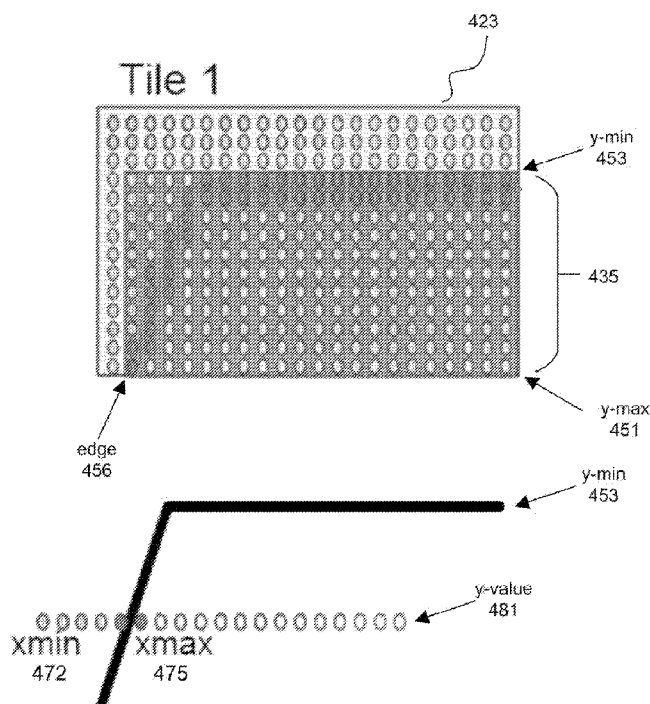


FIG. 4B

(57) Abstract: A method includes receiving instructions to render an image comprising content defined by a two-dimensional (2D) primitive; determining a portion of the 2D primitive covering a tile of multiple tiles of the image; generating an edge definition to represent an edge of the portion of the 2D primitive; and for each row of pixels within at least a portion of the tile containing the portion of the 2D primitive: identifying, based on the edge definition, a left-most pixel and right-most pixel in the row that intersect the edge; identifying, based on the left-most pixel and the right-most pixel, a set of first pixels in the row intersecting the edge; determining, for each first pixel in the set, a coverage weight indicating a proportion of the first pixel covered by the 2D primitive; and determining color information for the set of first pixels based on the associated coverage weights.

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))*

1 RASTERIZATION OPTIMIZATION FOR ANALYTIC ANTI-ALIASING

2 TECHNICAL FIELD

3 [1] This disclosure generally relates to a hardware architecture of a processor unit
4 for 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 SUMMARY OF THE INVENTION

17 [1] In accordance with a first aspect of the present disclosure, there is provided a
18 method comprising, by a computing system: receiving instructions to render an image
19 comprising content defined by at least a two-dimensional (2D) primitive; determining a
20 portion of the 2D primitive covering a tile of a plurality of tiles of the image; generating an
21 edge definition to represent an edge of the portion of the 2D primitive; and for each row of
22 pixels within at least a portion of the tile containing the portion of the 2D primitive:
23 identifying, based on the edge definition, a left-most pixel and a right-most pixel in the row
24 that intersect the edge; identifying, based on the left-most pixel and the right-most pixel, a set
25 of first pixels in the row intersecting the edge; determining, for each first pixel in the set, a
26 coverage weight indicating a proportion of the first pixel covered by the 2D primitive; and
27 determining color information for the set of first pixels based on the associated coverage
28 weights.

29 [2] In some embodiments, the 2D primitive is a horizontally-aligned trapezoid or
30 a quadratic curve.

31 [3] In some embodiments, the method further comprises: determining a bounding
32 box that encompasses the portion of the 2D primitive covering the tile; wherein the at least
33 the portion of the tile containing the portion of the 2D primitive is the bounding box.

1 **[4]** In some embodiments, generating the edge definition to represent the edge of
2 the portion of the 2D primitive comprises: determining a function equation representing the
3 edge.

4 **[5]** In some embodiments, the function equation is a linear equation.

5 **[6]** In some embodiments, the function equation is a quadratic equation.

6 **[7]** In some embodiments, the method further comprises, for each row of the
7 pixels within the at least the portion of the tile containing the portion of the 2D primitive:
8 identifying, based on the left-most pixel and the right-most pixel, a set of second pixels in the
9 row and a set of third pixels in the row, wherein the set of second pixels correspond to pixels
10 that are fully uncovered by the 2D primitive and the set of third pixels correspond to pixels
11 that are fully covered by the 2D primitive.

12 **[8]** In some embodiments, the method further comprises, for each row of the
13 pixels within the at least the portion of the tile containing the portion of the 2D primitive:
14 determining, for each of the second pixels, a coverage weight that indicates that the second
15 pixel is fully uncovered by the 2D primitive; determining, for each of the third pixels, a
16 coverage weight that indicates that the third pixel is fully uncovered by the 2D primitive; and
17 determining color information for the set of second pixels and the set of third pixels based on
18 the associated coverage weights.

19 **[9]** In accordance with a second aspect of the present disclosure, there is provided
20 one or more computer-readable non-transitory storage media including instructions that,
21 when executed by one or more processors, are configured to cause the one or more processors
22 to: receive instructions to render an image comprising content defined by at least a two-
23 dimensional (2D) primitive; determine a portion of the 2D primitive covering a tile of a
24 plurality of tiles of the image; generate an edge definition to represent an edge of the portion
25 of the 2D primitive; and for each row of pixels within at least a portion of the tile containing
26 the portion of the 2D primitive: identify, based on the edge definition, a left-most pixel and a
27 right-most pixel in the row that intersect the edge; identify, based on the left-most pixel and
28 the right-most pixel, a set of first pixels in the row intersecting the edge; determine, for each
29 first pixel in the set, a coverage weight indicating a proportion of the first pixel covered by
30 the 2D primitive; and determine color information for the set of first pixels based on the
31 associated coverage weights.

32 **[10]** In some embodiments, the 2D primitive is a horizontally-aligned trapezoid or
33 a quadratic curve.

34 **[11]** In some embodiments, the instructions are configured to further cause the one

1 or more processors to: determine a bounding box that encompasses the portion of the 2D
2 primitive covering the tile; wherein the at least the portion of the tile containing the portion of
3 the 2D primitive is the bounding box.

4 [12] In some embodiments, generating the edge definition to represent the edge of
5 the portion of the 2D primitive comprises: determining a function equation representing the
6 edge, wherein the function equation is a linear equation or a quadratic equation.

7 [13] In some embodiments, the instructions are configured to further cause the one
8 or more processors to: for each row of the pixels within the at least the portion of the tile
9 containing the portion of the 2D primitive: identify, based on the left-most pixel and the
10 right-most pixel, a set of second pixels in the row and a set of third pixels in the row, wherein
11 the set of second pixels correspond to pixels that are fully uncovered by the 2D primitive and
12 the set of third pixels correspond to pixels that are fully covered by the 2D primitive.

13 [14] In some embodiments, the instructions are configured to further cause the one
14 or more processors to: for each row of the pixels within the at least the portion of the tile
15 containing the portion of the 2D primitive: determine, for each of the second pixels, a
16 coverage weight that indicates that the second pixel is fully uncovered by the 2D primitive;
17 determine, for each of the third pixels, a coverage weight that indicates that the third pixel is
18 fully uncovered by the 2D primitive; and determine color information for the set of second
19 pixels and the set of third pixels based on the associated coverage weights.

20 [15] In accordance with a third aspect of the present disclosure, there is provided a
21 system comprising: one or more processors; and one or more computer-readable non-
22 transitory storage media in communication with the one or more processors, the one or more
23 computer-readable non-transitory storage media comprising instructions that when executed
24 by the one or more processors, cause the system to: receive instructions to render an image
25 comprising content defined by at least a two-dimensional (2D) primitive; determine a portion
26 of the 2D primitive covering a tile of a plurality of tiles of the image; generate an edge
27 definition to represent an edge of the portion of the 2D primitive; and for each row of pixels
28 within at least a portion of the tile containing the portion of the 2D primitive: identify, based
29 on the edge definition, a left-most pixel and a right-most pixel in the row that intersect the
30 edge; identify, based on the left-most pixel and the right-most pixel, a set of first pixels in the
31 row intersecting the edge; determine, for each first pixel in the set, a coverage weight
32 indicating a proportion of the first pixel covered by the 2D primitive; and determine color
33 information for the set of first pixels based on the associated coverage weights.

34 [16] In some embodiments, the 2D primitive is a horizontally-aligned trapezoid or

1 a quadratic curve.

2 [17] In some embodiments, the instructions are configured to further cause the one
3 or more processors to: determine a bounding box that encompasses the portion of the 2D
4 primitive covering the tile; wherein the at least the portion of the tile containing the portion of
5 the 2D primitive is the bounding box.

6 [18] In some embodiments, generating the edge definition to represent the edge of
7 the portion of the 2D primitive comprises: determining a function equation representing the
8 edge, wherein the function equation is a linear equation or a quadratic equation.

9 [19] In some embodiments, the instructions are configured to further cause the one
10 or more processors to: for each row of the pixels within the at least the portion of the tile
11 containing the portion of the 2D primitive: identify, based on the left-most pixel and the
12 right-most pixel, a set of second pixels in the row and a set of third pixels in the row, wherein
13 the set of second pixels correspond to pixels that are fully uncovered by the 2D primitive and
14 the set of third pixels correspond to pixels that are fully covered by the 2D primitive.

15 [20] In some embodiments, the instructions, when executed by the one or more
16 processors, further cause the system to: for each row of the pixels within the at least the
17 portion of the tile containing the portion of the 2D primitive: determine, for each of the
18 second pixels, a coverage weight that indicates that the second pixel is fully uncovered by the
19 2D primitive; determine, for each of the third pixels, a coverage weight that indicates that the
20 third pixel is fully uncovered by the 2D primitive; and determine color information for the set
21 of second pixels and the set of third pixels based on the associated coverage weights.

22 [21] It will be appreciated that any features described herein as being suitable for
23 incorporation into one or more aspects or embodiments of the present disclosure are intended
24 to be generalizable across any and all aspects and embodiments of the present disclosure.
25 Other aspects of the present disclosure can be understood by those skilled in the art in light of
26 the description, the claims, and the drawings of the present disclosure. The foregoing
27 general description and the following detailed description are exemplary and explanatory
28 only and are not restrictive of the claims.

29 BRIEF DESCRIPTION OF THE DRAWINGS

30 [22] FIG. 1A illustrates an example diagram of a 2D graphics system.

31 [23] FIG. 1B illustrates an example of a graphics engine

32 [24] FIG. 2 illustrates an example of sampling and anti-aliasing techniques.

33 [25] FIG. 3A illustrates an example 2D scene.

34 [26] FIG. 3B illustrates an example of 2D content broken down into individual

1 primitives.

2 [27] FIGS. 4A-4B illustrate an example technique of determining whether a pixel
3 intersects with an edge of a trapezoid.

4 [28] FIG. 5A illustrates an example quadratic curve primitive.

5 [29] FIG. 5B illustrates an example tile comprising a curved edge of a quadratic
6 curve.

7 [30] FIG. 6 illustrates an example frame with two primitives, a destination
8 primitive and a source primitive.

9 [31] FIG. 7 illustrates an example encoding pipeline.

10 [32] FIG. 8 illustrates a tile that is segmented into multiple blocks.

11 [33] FIG. 9 illustrates an example encoding pipeline within a block encoder.

12 [34] FIG. 10 illustrates an example of the techniques of a spatial predictor.

13 [35] FIG. 11 illustrates an example a compressed channel of texel values.

14 [36] FIG. 12 illustrates an example diagram for encoding a 4x4 texel block using a
15 variable-length technique.

16 [37] FIG. 13 illustrates an example technique of decoding a 4x4 texel block that
17 has been encoded by a block encoder.

18 [38] FIG. 14 illustrates an example method for determining the color information
19 of primitives in an image base in part by determining the coverage weight of each pixel in the
20 image.

21 [39] FIG. 15 illustrates an example method for determining the color information
22 of a primitive base in part by determining the coverage weight of each pixel of primitive
23 based on function equations representing the edges of the primitives.

24 [40] FIG. 16 illustrates an example method for blending source shape with a
25 destination shape using a blending mode that requires updates to pixels in the color buffer
26 uncovered by the source shape.

27 [41] FIG. 17 illustrates an example method for encoding blocks of pixels based on
28 a tag that is used to temporary represent block headers.

29 [42] FIG. 18 illustrates an example method for determining whether a block of
30 pixels is different from previously-compressed blocks and compressing the block using a
31 variable-length technique.

32 [43] FIG. 19 illustrates an example method for encoding a plurality of pixels based
33 on delta encoding that utilizes a base value, symbol mask, symbol width, and sequence of
34 symbols.

1 [44] **FIG. 20** illustrates an example network environment.

2 [45] **FIG. 21** illustrates an example computer system.

3 **DETAILED DESCRIPTION**

4 [46] This invention is directed to an architecture of a 2D graphics engine (e.g.,
5 graphics processing unit, GPU) that is configured to render high-quality graphics while
6 operating on an ultra-low power budget. Particular embodiments disclosed herein provide an
7 improved technique for anti-aliasing. Anti-aliasing could be done in a variety of ways.
8 Traditionally, anti-aliasing is achieved using Multi-Sample Anti-Aliasing (MSAA), which
9 samples multiple points within a pixel area to determine what color the pixel should display.
10 A more accurate anti-aliasing could be achieved with more sampling points, but sampling is
11 computationally expensive. Instead of sampling, this invention converts 2D content
12 definitions into primitive shapes (e.g., 2D horizontally-aligned trapezoids and quadratic
13 curves) and leverages the known geometric properties of the primitives to perform analytic
14 anti-aliasing (e.g., instead of sampling a pixel at multiple points, embodiments disclosed
15 herein use geometry to compute how pixels/tiles are covered by the primitives). For example,
16 the technique involves calculating the amount of pixel that is covered by a primitive (e.g.,
17 11% of the pixel is covered by a trapezoid), then rendering the pixel shading based on
18 thereof. This technique allows the rendering of high-quality images at low power.

19 [47] In particular embodiments, a graphics engine performs anti-aliasing tile-by-
20 tile. A scene may be broken down into individual tiles, each tile comprising a fixed number
21 of pixels such as 32x32 pixels. For each tile, a “shape walker” component of the graphics
22 engine determines evaluates the pixels within the tile and determines whether the pixels are
23 completely inside, completely outside, or partially inside and partially outside a primitive that
24 is covered in the tile. Pixels that are completely inside or completely outside the primitive do
25 not need anti-aliasing, whereas pixels that intersect or overlap with an edge of the primitive
26 (e.g., outer frame of the primitive) would need to be sent to the “integrator,” where more fine-
27 grained pixel-level analytic anti-aliasing is needed. Particular embodiments disclosed herein
28 provide a novel technique for achieving such tasks.

29 [48] In particular embodiments, 2D scene that is to be rendered is divided into tiles,
30 each tile having a pre-determined number of pixels (e.g., 16x16). Text and 2D content within
31 the scene is defined as paths or contours, which is then converted into shapes of axis-aligned
32 trapezoid or piecewise-biquadratic (simply quadratics) curves. These shapes are referenced as
33 primitives. Then, for each tile, XRU-2D identifies the smallest bounding box within a tile that
34 encompasses a portion of a primitive covered by the tile. Each row of pixels within the

1 bounding box is then traversed row-by-row to determine pixels that overlap with the outer
2 shape of the primitive. Once the overlapping pixels are identified, pixels that do not need to
3 be anti-aliased are identified. The overlapping pixels are then sent to the integrator, while
4 other pixels (pixels falling outside the primitive or fully inside the primitive) are assigned 0
5 and 1 weight values, respectively, and sent to a different process (not to an integrator).
6 Subsequent steps involve, the integrator figuring out the coverage weight of the overlapping
7 pixels against the primitive, which may be used to determine the pixel shading for anti-
8 aliasing.

9 [49] In particular embodiments, the technique of identifying the overlapping pixels
10 discussed above involves one of two variations depending on whether the primitive is an
11 axis-aligned trapezoid or a piecewise-biquadratic (simply quadratics) curve. If the primitive is
12 a trapezoid, the method involves identifying the maximum and minimum Y values of the
13 trapezoid (e.g., top and bottom size of the trapezoid) and Y-intercepts and slope of an edge
14 (or both edges if two sides of the trapezoid fits into one tile). Then, the method continues by
15 traversing row-by-row to identify, based on the slopes and Y-intercepts identified in the
16 previous step, pixels that are overlapping with the shape of the trapezoid. Then, the
17 overlapping pixels are sent to the integrator, and pixels that fall outside of the trapezoid are
18 assigned weight 0 and pixels that fall inside the trapezoid are assigned weight 1. If the
19 primitive is a curve, the same high-level steps of identifying overlapping pixels and applying
20 weights to non-overlapping pixels are used, but in contrast to if the primitive is a trapezoid,
21 quadratic formula is used to represent the curve rather than using Y-intercepts and slope

22 [50] Particular embodiments of a graphics system focus on techniques for blending
23 multiple overlapping primitives in the color buffer. To remove as much processing work as
24 possible, the XRU-2D architecture utilizes a “culling” technique where tiles (e.g., 16x16
25 pixels) of a frame are culled using a smallest bounding box that encompasses a primitive
26 being processed by the hardware. Then, only the tiles covered by the bounding box are sent
27 down the rendering pipeline such that the empty tiles outside the bounding box are ignored
28 by the system. This reduces the overall computing required for rendering makes the system
29 more power-efficient. Incorporating this culling technique, however, presents an issue
30 because it conflicts with some of the special blending modes used for overlapping primitives.
31 Specifically, such special blending modes require access to both (1) the tiles covering the
32 destination primitive (tiles already in the color buffer) but not the source primitive and (2) the
33 tiles covering the source primitive (tiles that are to be written into the color buffer). However,
34 given the culling technique, when processing the source primitive, the system only has access

1 to the tiles of the source primitive. Embodiments disclosed herein provide a solution for these
2 issues.

3 [51] In particular embodiments, the task of clearing a destination primitive involves
4 defining the tiles in a frame as “non-empty tiles” when the tiles cover a source primitive and
5 as “empty tiles” when the tiles do not cover the source primitive. For empty tiles, the task of
6 clearing a destination primitive may involve a graphics system instructing a color buffer to
7 bypass the primitive cull to allow the color buffer to gain access to the empty tiles. For non-
8 empty tiles, the clearing task is a bit more complicated since only the destination primitive
9 must be cleared from the non-empty tiles, thus the embodiments disclose a technique that
10 involves a pixel-by-pixel analysis for each of the non-empty tiles that tracks the blending
11 modes that are used to update each pixel. This allows the system to identify pixels in the non-
12 empty tiles that are covering only the destination primitive and not the source primitive.

13 [52] Particular embodiments herein focus on encoding static assets (e.g., a blit such
14 as an emoji). For optimal compression, an encoder system disclosed herein uses a spatial
15 predictor to find similar pixel blocks within a frame and assign them the same block header.
16 A naïve implementation of the spatial predictor is to (1) compare the current block to a
17 previously processed block pixel-by-pixel and (2) copy the header of the matching block into
18 the current block, such that when the current block is decoded, the decoder can retrieve the
19 data from the matching block. However, there is a problem with the naïve implementation
20 because pixel-by-pixel comparison requires a lot of power and memory. As a solution,
21 embodiments herein disclose a technique of generating a hash code to represent each pixel
22 block and uses the hash codes to compare blocks. There exists another problem with the
23 naïve implementation noted above. The encoding process involves several steps in a pipeline.
24 The step of comparing the blocks (e.g., comparing hash codes) occurs at the first step and the
25 step of assigning a header occurs at the last step, with several steps in between the first and
26 last steps. This means that when a particular block is compared to the one of the previous
27 blocks, the previous blocks may still be going through the encoding pipeline and headings for
28 these blocks may not have been generated yet. In circumstances where a header of a previous
29 block is not generated yet but the current block’s hash code matches that of the previous
30 block, the current block is assigned a tag instead of a header. At the end of each cycle in the
31 pipeline (e.g., when a block is handed off to the next step in the encoding pipeline), the
32 system checks whether a previously unavailable header is available, and if available, replaces
33 the corresponding tag with the header. The system is configured in a way that a block’s tag
34 may be updated with a header wherever the block is in the encoding pipeline. This solution

1 prevents the encoding pipeline from being stalled due to certain headers not being available at
2 the time a matching block is found.

3 [53] In particular embodiments, an encoder and the decoder reside in the graphics
4 engine, but the encoded pixel blocks may be stored in a memory external to the graphics
5 engine and accessed by the decoder from the external memory as necessary. Tiles that are
6 encoded by the graphics engine may be piped through a double buffer such that the encoder
7 can compress the current tile while the next tile is streamed in. For each tile to be encoded, a
8 block scheduler may separate the tile into blocks for the encoder, and the block schedule may
9 schedule the blocks in an arrangement called the “Morton Order” that is optimized for delta
10 coding (e.g., minimizing the distance between the blocks in a sequence). For each block that
11 is encoded, the pixels may be also be encoded based on the Morton Order arrangement. For
12 each block, the different pixel channels may be encoded separately then subsequently
13 collated into a single bitstream.

14 [54] Particular embodiments herein provide a technique of encoding pixel blocks of
15 an image that allows the system to selectively retrieve and decode any particular pixel block
16 of the encoded blocks, e.g., independently from other encoded pixel blocks. More
17 specifically, each block may be encoded in a way that it is self-contained, meaning that the
18 system can selectively retrieve and decompress a particular block simply based on the data
19 contained within the block. For example, with a PNG image that encoded using the disclosed
20 technique, a system may be able to retrieve and decompress specific portions of the PNG
21 image (pixel blocks) independently from other portions of the image.

22 [55] In particular embodiments, the encoding system may analyze each block of
23 texels that are to be encoded and categorizes them into one of two block variants: Flatblock
24 or Codeblock. A block may be categorized as a Flatblock if all texels in the block have the
25 same value. Otherwise, a block may be categorized as a Codeblock, for example, if some of
26 the texels in the block have different values. Later in the encoding process, after a Codeblock
27 is encoded, the encoding system may evaluate the encoded block data of the Codeblock to see
28 whether the encoded data is greater in size than the uncompressed size of the block, that is,
29 whether the encoded data requires more bits than uncompressed data. If so, the encoding
30 system may (1) disregard the encoded block data of the Codeblock, (2) recategorize the
31 Codeblock as a third block variant, i.e., Rawblock, and (3) store the uncompressed block data
32 of the block in lieu of the disregarded encoded data. In an embodiment, the categorization
33 steps described above are implemented for an entire block, that is, without regard to the
34 various channels of texels that the block may have. For example, a block comprising multiple

1 channels of texels may be categorized as a Flatblock only if all texels within the block have
2 the same value, including texels of different channels. Alternatively, if any texel values differ
3 in a block, even across different channels, the block may be categorized as a Codeblock.

4 [56] In particular embodiments, the encoding system may compress each of the
5 three block variants with different techniques. The encoding system compresses a Flatblock
6 using a single texel value such that the encoded data of the Flatblock is the size of a single
7 texel value. The encoding system may compress a Codeblock a single channel at a time, and
8 for each channel, a different encoding technique may be utilized. A particular channel of a
9 Codeblock may be encoded using a “flat” technique if all of the values of the texels in the
10 channel are the same. The flat technique involves using a single value to represent the entire
11 channel. If, for a particular channel in a block, the values of the texels differ, then a “variable-
12 length” technique may be used. The variable-length technique is a novel compression
13 technique that produces different sizes of encoded data depending on the differences in the
14 texel values within the block. After encoding a channel using the variable-length technique,
15 the system may evaluate whether the encoded channel data is computationally more
16 expensive than the uncompressed channel data, that is, whether the encoded data requires
17 more bits than the uncompressed data. If so, the encoding system may disregard the encoded
18 channel data and store the channel data as uncompressed. In particular embodiments, each
19 texel channel within a Codeblock may be independently encoded using any of the techniques
20 described above. For example, for a Codeblock having three channels of texel values, a first
21 channel of the three may be encoded using a flat technique, a second channel of the three may
22 be encoded using the variable-length technique, and a third channel of the three may be
23 stored as uncompressed. Each of the encoded (or uncompressed) channels in a Codeblock
24 may be collated together to form the compressed block data for the Codeblock. If, however,
25 the compressed block data for a Codeblock turns out to be computationally more expensive
26 than the uncompressed data of the Codeblock, then the compressed block data may be
27 disregarded and the Codeblock may be recategorized as a Rawblock. The encoding system
28 stores a Rawblock without any compression. The size of a Rawblock represents the
29 maximum size of a stored block.

30 [57] In particular embodiments, an encoding system disclosed herein may use a
31 technique referred to as the variable-length technique to encode a texel block and produces
32 different sizes of encoded data depending on the texel values within the block. This technique
33 encodes the texels channel-by-channel, meaning each texel component is separately encoded,
34 for example, each of R, G, B components may be separately encoded. The variable-length

1 technique involves using three groups of data to represent the uncompressed texel values:
2 “symbolmask”; “rbits”; “rsymbols.” Data group rsymbols is used to represent the delta values
3 of the texel values of a block, as arranged in a particular way, for example, in an arrangement
4 called the “Morton Order,” which is optimized for delta coding (e.g., minimizing the distance
5 between the blocks in a sequence). Data group symbolmask is used to provide a 1 to 1
6 mapping of the delta values and indicates whether each delta value is a zero value or non-zero
7 value. Data group rbits is used to indicate the maximum number of bits required to represent
8 each of the delta values, the maximum number including an additional bit to indicate whether
9 the delta values are positive or negative values.

10 [58] As an example of the three group of data noted above, consider texel values of
11 a block that are arranged as [0, 4, 1, 1], in which case the delta values for that sequence of
12 values would be [4, -3, 0]. The symbolmask for delta values [4, -3, 0] would be [1, 1, 0],
13 which simply represents which of the delta values are zeros and non-zeros. The number of
14 bits of the encoded data that are assigned to symbolmask equals the number of delta values,
15 or the number of uncompressed texel values minus one. The rbits for delta values [4, -3, 0]
16 would be four bits—a first bit to indicate the positive or negative sign of the delta values and
17 three additional bits to represent each of the delta values which has a maximum value of four.
18 In an embodiment, the number of bits represented by rbits may be stored in the encoded data
19 in a binary representation, such that rbits of 4 may be stored as [100]. In some embodiments,
20 rbits may be stored with an offset, for example, with an offset of 2 such that rbits of 4 bits
21 may be stored as 2, or [010]. In some embodiments, the number of bits in the encoded data
22 that are assigned for rbits may be fixed, for example, to three bits. In embodiments where
23 three bits of the encoded data are assigned to rbits (e.g., [###]) and rbits are stored with an
24 offset of 2, the range of binary values that rbits can represent varies from 2 to 9. The
25 rsymbols for delta values [4, -3, 0] may be generated as [0100, 1011], the first of which
26 represents a positive indicator (“0”) followed by a binary representation of 4 (“100”) and the
27 second of which represents a negative indicator (“1”) followed by a binary representation of 3
28 (“011”). Notably, the data group rsymbols only needs to represent non-zero delta values since
29 any zero delta values are already indicated by symbolmask. Once all three groups of data are
30 generated to represent the texel values of a block, they may be collated together into a stream
31 of bits. Continuing the example above, the three groups of data may be collated as [rsymbols,
32 rbits, symbolmask], or [0100 1011, 010, 1 1 0]. In addition to the three groups of data, the
33 first value of the uncompressed texel values may be encoded as the “base value” of the
34 encoded data, either encoded together with the three groups of data or separately as metadata.

1 In the example above, the base value would be 0 since that is the first value of the
2 uncompressed texel values [0, 4, 1, 1].

3 [59] In particular embodiments, a decoder disclosed herein may be configured to
4 decode multiple delta values from the uncompressed data per one decoding cycle. For
5 example, if a decoder system is configured to decode three delta values per cycle, each cycle
6 may involve fetching the first three entries from symbolmask to decode the three delta values
7 in parallel. For any delta values that the symbolmask indicates as being a zero value, no
8 additional process may be required to determine that delta value since the value is zero. For
9 any delta values that the symbolmask indicates as a non-zero value, the decoder system may
10 fetch from rsymbols a number of bits indicated by rbits from the appropriate portion of
11 rsymbols. For example, if rbits is 5, the decoding system would fetch the first 5 bits for the
12 first non-zero delta value. The decoding system may be configured to maintain a pointer or
13 other indicators that indicates which portion of the rsymbols should be fetched next to allow
14 the decoder system to determine which portion of the rsymbols should be fetched for the next
15 non-zero delta value. After the decoder system determines the value of each of the first three
16 delta values, the delta values may be further decoded based on the base value of the
17 uncompressed data. For example, the first uncompressed data value can be determined as
18 equating to the base value and the next uncompressed value can be determined by adding the
19 first delta value to the base value, then the combined value can be used to determine the
20 subsequent uncompressed value by adding the subsequent delta value to it.

21 [60] The embodiments disclosed herein are only examples, and the scope of this
22 disclosure is not limited to them. Particular embodiments may include all, some, or none of
23 the components, elements, features, functions, operations, or steps of the embodiments
24 disclosed above. Embodiments according to the invention are in particular disclosed in the
25 attached claims directed to a method, a storage medium, a system and a computer program
26 product, wherein any feature mentioned in one claim category, e.g. method, can be claimed in
27 another claim category, e.g. system, as well. The dependencies or references back in the
28 attached claims are chosen for formal reasons only. However, any subject matter resulting
29 from a deliberate reference back to any previous claims (in particular multiple dependencies)
30 can be claimed as well, so that any combination of claims and the features thereof are
31 disclosed and can be claimed regardless of the dependencies chosen in the attached claims.
32 The subject-matter which can be claimed comprises not only the combinations of features as
33 set out in the attached claims but also any other combination of features in the claims,
34 wherein each feature mentioned in the claims can be combined with any other feature or

1 combination of other features in the claims. Furthermore, any of the embodiments and
2 features described or depicted herein can be claimed in a separate claim and/or in any
3 combination with any embodiment or feature described or depicted herein or with any of the
4 features of the attached claims.

5 [61] FIG. 1A illustrates an example diagram of a 2D graphics system according to
6 embodiments disclosed herein. Such embodiments may include an application 101 that
7 provides scene details, a driver 102 that converts paths within a scene into shapes that can be
8 more efficiently processed (referred herein as “primitives”), a 2D graphics engine 103 for
9 rendering a scene, and a display 198 for displaying the rendered scene. A 2D graphics engine
10 103 may be referred herein as a graphics engine, graphics system, GPU, or simply a “system”
11 for brevity.

12 [62] In particular embodiments, a driver 102 may be configured to decompose a
13 scene received from an application 101 into individual shapes that can be more efficiently
14 processed by a 2D graphics engine 103, such shapes are referred herein as “primitives.” A
15 scene may consist of a number of 2D content and texts. 2D content and texts contained in a
16 scene may be defined by “paths,” where each path is made up of lines, curves, arcs, or
17 otherwise referred herein as “contours.” In an embodiment, an application 101 defines the
18 paths within a scene. For example without limitation, a typical scene may contain between
19 2,000-20,000 paths. In an embodiment, each contour may be required to be “closed,” such
20 that the first and last vertices of the contour are identical (e.g., at the same location). In an
21 embodiment, a driver 102 may be configured to process each path in a scene by converting
22 the contours of the path into one of two types of primitives: (1) horizontally aligned
23 trapezoids and (2) piecewise-biquadratic curves. A horizontally aligned trapezoid, referred
24 hereinafter as a “trapezoid” for brevity, comprises two parallel horizontal edges on the top
25 and bottom sides of the trapezoid and two side edges connecting the top and bottom sides. A
26 piecewise-biquadratic curve—referred hereinafter as a “quadratic curve” for brevity—is a 2-
27 D region bounded by a quadratic curve and a line. An example process of converting the
28 contours of a path into primitives is disclosed in the following paper: A. Ellis, W. Hunt, J.
29 Hart, Nerf: *Real-Time Analytic Antialiased Text for 3-D Environments*, Computer Graphics
30 forum, vol. 38, issue 8, November 2019, pp. 23-32.

31 [63] In particular embodiments, a driver 102 may be configured to perform tiling
32 operations by which a scene is segmented into a smaller data structure called a tile, or tile
33 block. Each tile may be composed of a set of pixels. For example, a tile may be comprised of
34 a 16-by-16 pixel block or a 32-by-32 pixel block. In an embodiment, a driver 102 may be

1 configured to determine, for each tile in a scene, every primitive that is covered the tile, then
2 store this information in a memory database 109 that is accessible by a graphics engine 103.

3 [64] While recognizing the differences of terms “pixels” and “texels” as used in the
4 field of art, any references to pixels herein may be interchangeable with references to texels
5 and any references to texels herein may be interchangeable with references to pixels, for the
6 purposes of describing the embodiments herein.

7 [65] FIG. 1B illustrates an example of a graphics engine 103. In particular
8 embodiments, a 2D graphics engine 103 may be configured to perform rendering operations
9 tile by tile or a single tile at a time. In other embodiments, a graphics engine 103 may
10 perform certain rendering operations multiple tiles at a time or in parallel. A command
11 controller 107 may be configured to arrange the tiles within a scene in a specific order and
12 provide instructions to a tile controller 120 to start rendering the tiles according to the tile
13 order. For example, a command controller 107 may be configured to implement a tile
14 walking function that iterates over the tile data structure to determine information about the
15 tile. Such tile information may include which tiles should be processed by the downstream
16 rendering components and in what order the tiles should be processed. The command
17 controller 107 may then provide the tile information to the rendering downstream
18 components, such as a tile controller 120. In an embodiment, a command controller 107 may
19 only identify the tiles that cover a primitive or a background, for example, tiles that are empty
20 may not be sent down the rendering pipeline for efficiency purposes. In an embodiment, a
21 command controller 107 may be configured to determine, for each tile containing at least a
22 portion of a primitive, a tile bounding box that encompasses the at least the portion of the
23 primitive within the tile. The tile bounding box information may then be sent down the
24 rendering pipeline to allow certain operations to focus only on the tile bounding box within a
25 tile rather than the entire tile. In an embodiment, the tile bounding box information may also
26 comprise data indicating which edges of a primitive are contained in the tile bounding box.
27 For example, if a tile contains a top left portion of a trapezoid, the bounding box information
28 may indicate that the tile contains the left and top edges of the trapezoid. In an embodiment, a
29 command controller 107 may be configured to generate a list of primitives that are contained
30 in each of a non-empty tile (a tile that is covering with one or more primitives), and this list
31 may be sent down the rendering pipeline. While memory database 109 is illustrated in FIG.
32 1B, the memory database 109 may be comprised of multiple memory databases, each
33 memory database being responsible for storing data that is unrelated to data stored in other
34 memory databases. In an embodiment, a command controller 107 may be configured to

1 determine, for each primitive, a primitive bounding box that encompasses the primitive
2 across one or more tiles in a frame (image). The primitive bounding box information may
3 then be sent down the rendering pipeline to allow certain operations to focus on the primitive
4 bounding box rather than the entire frame.

5 [66] In particular embodiments, once the command controller 107 provides
6 instructions to a tile controller 120 tiles to render, the tile controller 120 may be configured to
7 gather all the primitive, blit, and/or filter information necessary to render the tiles. For every
8 tile to be rendered, a tile controller 120 may begin the rendering process by fetching the tile
9 data from a tile memory database 109, for example, through the input box 106 shown in FIG.
10 1B. The tile data that is fetched by the tile controller 120 may be passed to downstream
11 components in the rendering pipeline (e.g., shape walker 130). A tile controller 120 may only
12 fetch non-empty tiles from the tile memory database 109. The fetch operation performed by a
13 tile controller 120 may be a single-step process and may involve fetching data associated
14 each primitive within the tiles, including all the vertices of the primitive and a portion of the
15 shader information associated with the primitive. The rest of the shader information may be
16 fetched by a shader 150. A tile controller 120 may also be configured to fetch blit and filter
17 render instructions from memory that is external to the graphics engine. After parsing through
18 the fetched data, a tile controller 120 may be configured to perform a tile bounding box
19 check. Then, a tile controller 120 may be configured to provide the shader information to a
20 shader 150 and the blit and filter information to a blit and filtering unit 180. In an
21 embodiment, a tile controller 120 may be configured to provide tile-done and commands-
22 done indicators to the color buffer 191, 193. This information represents the status with
23 respect to what is processed by a tile controller 120 and what is not.

24 [67] In particular embodiments, a shape walker 130 may be configured to
25 determine the coverage weight of each pixel within a tile, the coverage weight representing
26 how much of the pixel is covered by a primitive within the tile. In other words, a shape
27 walker 130 may be configured to examine each of the pixels in the tile (or within the tile
28 bounding box) to determine whether the pixels falls inside, outside, or partially intersects
29 with an edge of a primitive (e.g., trapezoid or a quadratic curve). Pixels that are determined to
30 be fully inside a primitive are given a coverage weight of 1, pixels determined to be that are
31 fully outside a primitive is given a coverage weight of 0, and pixels that are intersecting with
32 an edge of a primitive are sent to an integrator 140 for further processing (e.g., an integration
33 step). Partially interacting, or overlapping, pixels require an integration step to precisely
34 determine how much of the pixel overlaps with an edge of a primitive. This information is

1 used for anti-aliasing at a later step in the rendering pipeline. For pixels that are assigned
2 coverage weights of 0 or 1 by a shape walker 130, their respective coverage weights are
3 provided to coverage buffers 151 or 152.

4 [68] Traditional methods for anti-aliasing typically use sampling or Multi-Sample
5 Anti-Aliasing (MSAA), which involves sampling points within a pixel area to determine the
6 coverage weight for that pixel. For example, as illustrated in **FIG. 2**, to determine whether a
7 triangle 204 overlaps with the pixel area 201, a graphics system may take a sample at point
8 202. In this example, the system may determine that the pixel area 201 is not covered by
9 content 204 because, at sample point 202, the triangle 204 does not cover the pixel 201. The
10 system may then assign a coverage weight of 0 to the pixel 201 to indicate that pixel 201 is
11 not covered by any portion of the triangle. Alternatively, instead of taking a single sample
12 from pixel 201, multiple points of samples 203 may be taken. As shown in the top right
13 example in **FIG. 2**, taking four samples within pixel 201 allows the system to determine a
14 coverage weight of 0.5 or 50%, which is a more accurate coverage weight than 0. As shown
15 by these examples, traditional methods for anti-aliasing can determine coverage weights of
16 higher accuracy as more samples are taken, however, there is a trade-off for taking more
17 samples since each additional sample point requires additional computing power and/or
18 compute time. Moreover, when coverage weights are determined by a way of taking samples,
19 the resulting coverage weights are typically rough estimates and may only provide fixed
20 coverage weights. For example, if one sample is taken, the coverage weight for a pixel can
21 only be either 0 (not covered) or 1 (fully covered). If four samples are taken, the coverage
22 weight for a pixel can only be 0, 0.25, 0.5, 0.75 or 1. In contrast to such traditional methods,
23 embodiments disclosed herein allow the determination of coverage weights in a more
24 granular fashion (e.g., non-fixed coverage weights) and without any sampling. For example,
25 according to an embodiment illustrated in bottom half of **FIG. 2**, a graphics engine may be
26 able to utilize techniques disclosed herein to determine that 12% of the pixel 251 is covered
27 by a trapezoid 253, 23% of pixel 252 is covered, 100% of pixel 253 is covered, and 0% of
28 pixel 254 is covered.

29 [69] In particular embodiments, a scene may be broken down into smaller units of
30 pixels referred to as tiles. For example, **FIG. 3A** illustrates an example 2D scene 305 and also
31 the same scene 307 that is broken down into individual tiles. Each tile may comprise a fixed
32 number of pixels, such as 16-by-16 pixels or 32-by-32 pixels. In particular embodiments,
33 content within a scene may be broken down into smaller units referred herein as primitive
34 (e.g., a quadratic curve, trapezoid). **FIG. 3B** illustrates an example of 2D content 371 that

1 may be shown in a scene and broken down into individual primitives. Specifically, FIG. 3B
2 illustrates content 371 that is broken down into four quadratic curves 376 representing the
3 corners of content 371, two trapezoids 374 representing top and bottom portions of the
4 content 371, and one “trapezoid” 378 in the center portion of the content 371. While
5 trapezoid 378 may appear to be a rectangle rather than a trapezoid in the literal sense,
6 embodiments may be configured to consider trapezoid 378 as a trapezoid with side edges that
7 are vertically oriented (e.g., perpendicular from the top and bottom edges). In particular
8 embodiments, referring to FIG. 1A, an application 101 may be configured to break down
9 content into primitives and may provide the primitives and information about the primitives
10 to a driver 102.

11 [70] In an embodiment, a shape walker 130 may be configured to utilize an
12 algorithm known as DDA (digital differential analyzer) line generating algorithm to
13 determine whether a pixel intersects with an edge of a primitive. The technique of identifying
14 intersecting pixels may involve first determining a function equation that represents an edge
15 of a primitive (or otherwise referred as an “edge definition”), then utilizing an algorithm to
16 determine whether a pixel overlaps/intersects with the edge represented by the function
17 equation. For example, if a primitive is a trapezoid, a shape walker 130 may first determine
18 the maximum and minimum Y values of the trapezoid (e.g., top and bottom edge of the
19 trapezoid) and y-intercepts and slope of an edge (or both edges if two side edges of the
20 trapezoid fits into one tile). The y-intercepts and slope may be used to determine a function
21 equation (e.g., linear equation) that represents a corresponding edge. Then, the technique may
22 continue by traversing row-by-row of the tile to identify, based on the function equation
23 identified in the previous step, pixels that are intersecting with the edge of the trapezoid (e.g.,
24 the function equation). The intersecting pixels are sent to the integrator to determine the pixel
25 coverage weight, and pixels that fall completely outside of the trapezoid are assigned
26 coverage weight of 0 and pixels that fall completely inside the trapezoid are assigned
27 coverage weight of 1. On the other hand, if the primitive is a quadratic curve, the same high-
28 level technique of identifying the overlapping pixels and applying weights to non-
29 overlapping pixels are used, but in contrast to if the primitive is a trapezoid, a quadratic
30 formula is used to represent the curve rather than a linear equation.

31 [71] FIGS. 4A-4B illustrate an example technique of determining whether a pixel
32 intersects with an edge of a trapezoid. Specifically, FIG. 4A illustrates a trapezoid 402 that is
33 covering pixels of several tiles. In particular embodiments, a shape walker 130 processes a
34 particular primitive tile by tile. For example, a shape walker 130 may be configured to

1 process each of the numbered tiles in box 413 in the sequence of the illustrated numbers (e.g.,
2 tile 1, tile 2, ... tile 11). Such a sequence of the tiles may be determined by a command
3 controller 107 and provided to downstream components in the rendering pipeline such as a
4 shape walker 130. FIG. 4B illustrates the first tile 423 shown in FIG. 4A and further
5 illustrates the step of determining whether pixels within the tile 423 intersects with an edge of
6 the trapezoid 402. In particular embodiments, a shape walker 130 may receive the tile
7 bounding box information that outlines a box 435 that encompasses a primitive or a portion
8 of thereof. A shape walker 130 may be configured to process the pixels only within the
9 bounding box, rather than the entire tile. The tile bounding box information that a shape
10 walker 130 receives may also indicate which edges of primitives are contained within a tile.
11 For example, in the example shown in FIG. 4B, the tile bounding box information may
12 indicate that only the left and top edges of a trapezoid 402 are contained in tile 423. The tile
13 bounding box information may further indicate whether, for an edge within the tile, whether
14 the entirety of the edge is in the tile or only a portion of the edge is within the tile. For
15 example, in the example shown in FIG. 4B, the tile bounding information may indicate that
16 only a portion of the top edge and only a portion the left edge of a trapezoid 402 are
17 contained in tile 412.

18 [72] In particular embodiments, a shape walker 130 may be configured to analyze
19 each pixel position within a tile to determine whether the corresponding pixel
20 overlaps/intersects with an edge of a primitive (e.g., trapezoid or curve). A shape walker 130
21 may be configured to process, for each primitive within a tile, a single edge at a time. For
22 example, in the example shown in FIG. 4B, a shape walker 130 may be configured to process
23 the left edge 456 separately from the top edge. In particular embodiments, when a shape
24 walker 130 is processing one of the side edges of a trapezoid, the shape walker 130 may be
25 configured to determine y-min and y-max values of the primitive. For example, as shown in
26 FIG. 4B, the portion of a trapezoid shown within tile 423 comprises a y-min 453 representing
27 the top edge of the portion of the trapezoid and a y-min 451 representing the bottom portion
28 of the primitive that is within tile 423. Such y-min and y-max values may be determined by a
29 shape walker 130, for example, based on the bounding box information. A shape walker may
30 then determine y-intercepts and a slope of a side edge. For example, as shown in FIG. 4B, a
31 shape walker 130 may be configured to determine the y-intercepts and a slope of the edge
32 456 of the trapezoid. Using this information, a shape walker 130 may be configured to
33 determine a function equation based on a linear equation (e.g., $ax + b$) that defines a side edge
34 of a trapezoid. In a different circumstance where a tile includes only a right side edge of a

1 trapezoid, a function equation for the right edge may similarly be determined based on the y-
2 intercepts and a slope of the right edge. In yet another circumstance where a tile includes both
3 a left side edge and a right side edge, a shape walker 130 may also similarly determine the
4 function equation for both of the edges, but in separate operations.

5 [73] Once the y-min, y-max, and function equation is determined for a particular
6 edge contained in a tile, a shape walker 130 may be configured to traverse the tile row-by-
7 row (e.g., from y-min to y-max) and determine, for each row, pixels that intersect with an
8 edge of a trapezoid based on the function equation. For example, referring to FIG. 4B, a
9 shape walker 130 may be configured to traverse the tile 423 row by row, starting at y-min
10 453 and ending with y-max 451. A shape walker 130 may then determine, for each row, and
11 based on the function equation, the x-min and x-max values for that row (the x-min value
12 representing the leftmost position at which a pixel intersects with a side edge of a trapezoid
13 and the x-max value representing the rightmost position at which a pixel intersects with the
14 left edge of the trapezoid). For example, as shown in FIG. 4B, a shape walker may determine,
15 at the row corresponding to y-value 481, x-min value 472 and x-max value 475 for a left-side
16 edge of a trapezoid. In a different circumstance where a tile includes a right side edge of a
17 trapezoid, the x-min and x-max values may similarly be determined based on a corresponding
18 function equation. In yet another circumstance where a tile includes both a left side edge and
19 a right side edge, a shape walker 130 may also similarly determine the function equation for
20 both of the edges, but in separate operations. In particular embodiments, a shape walker 130
21 may be configured to determine the x-min and x-max values for a top or bottom edge based
22 on the function equation of the side edges and/or the bounding box information. For example,
23 referring to FIG. 4A, a shape walker 130 may determine, based on the bounding box
24 information, that tile 2 contains only a top edge of a trapezoid and that none of the side edges
25 are contained in tile 2. Based on this determination, a shape walker 130 may determine that
26 the top edge spans across the entirety of the length of the bounding box, and thereby
27 determine the x-min values and x-max values based on the position of the bounding box. A
28 shape walker 130 may similarly determine the x-min and x-max values of the bottom edge of
29 a trapezoid contained in tile 10. If a bounding box for a particular tile contains either a top or
30 bottom edge in addition to one of the side edges, such as tile 1 shown in FIG. 4B, a shape
31 walker 130 may be configured to plug in the y-value of the top/bottom edge into the function
32 equation of the side edges to determine the x-min and x-max values of the top/bottom edge.
33 For example, referring to FIG. 4B, a shape walker 130 may determine the x-min value of the
34 top edge by plugging in y-min value 453 into the function equation of edge 456. As for the x-

1 max value, a shape walker 130 may determine that, since the right side edge is not contained
2 in tile 423, the x-max value equals the rightmost x value of the bounding box 435. The
3 techniques discussed above may be used to determine the x-min and x-max values of any of
4 the top or bottom edges of a trapezoid. In an embodiment, for each row in a tile, a shape
5 walker 130 may be configured to determine the individual pixels that are intersecting with an
6 edge of a trapezoid based on the x-min and x-max values determined using the techniques
7 discussed above. For pixels that are intersecting with an edge of a trapezoid, a shape walker
8 130 may identify those pixels to an integrator 140, as explained further below.

9 [74] If the primitive is a quadratic curve, rather than a trapezoid, in accordance to
10 particular embodiments, a shape walker 130 may be configured to analyze each pixel position
11 within a tile to determine whether the corresponding pixel overlaps/intersects with an edge of
12 a quadratic curve. As shown in **FIG. 5A**, a quadratic curve primitive may be comprised of
13 two edges, one flat edge 503 and one curved edge 506. A shape walker 130 may be
14 configured to process, for each quadratic curve within a tile, a single edge at a time. The
15 technique of determining whether a pixel overlaps/intersects with a flat edge of a quadratic
16 curve is substantially similar to the technique described above with reference to a trapezoid,
17 for example, by representing the flat edge with a linear equation. The technique of
18 determining whether a pixel overlaps/intersects with a curved edge of a quadratic curve is
19 also substantially similar to the technique described above with reference to a trapezoid, but
20 in contrast, a quadratic formula is used to represent the curved edge rather than a linear
21 equation. For example, referring to **FIG. 5A**, a quadratic equation (ax^2+bx+c) may be used to
22 represent the function equation for the curved edge 506. Such quadratic equations may be
23 determined based on the three vertices (P0, P1, P2) of the quadratic curve shown in **FIG. 5A**.
24 The location of such vertices may be determined by a driver 102 or an application 101 and
25 provided to a graphics engine 103 (e.g., shape walker 130).

26 [75] In particular embodiments, to determine the function equation for a curved
27 edge of a quadratic curve, a shape walker 130 may be configured to determine the y-min and
28 y-max values and y-intercepts of the curved edge. A shape walker 130 may then use this
29 information and the three vertices of a quadratic curve (e.g., such as those shown in **FIG. 5A**)
30 to determine a quadratic equation that represents the curved edge of the quadratic curve. **FIG.**
31 **5B** illustrates an example tile comprising a curved edge 571 of a quadratic curve. Once the y-
32 min, y-max, and function equation is determined for a curved edge of a quadratic curve, a
33 shape walker 130 may be configured to traverse the tile row-by-row (e.g., from y-min to y-
34 max) and determine, for each row, pixels that intersect with the curved edge based on, for

1 example, the quadratic equation and the DDA line generating algorithm. For example, a
2 shape walker 130 may be configured to traverse the tile 580 row by row, starting at y-min
3 573 and ending with y-max 576. For each row, a shape walker 130 may be configured to
4 determine the x-min and x-max values for that row based on the corresponding quadratic
5 equation. Referring to the bottom half of FIG. 5B, a shape walker 130 may determine, for the
6 row corresponding to y-value 591, that x-min 592 is the leftmost position at which a pixel
7 intersects with a curved edge of a quadratic curve and that x-max 593 is the rightmost
8 position at which a pixel intersects with the curved edge of the trapezoid. In an embodiment,
9 for each row in a tile, a shape walker 130 may be configured to determine the individual
10 pixels that are intersecting with an edge of a quadratic curve based on the x-min and x-max
11 values determined using the techniques discussed above. For pixels that are intersecting with
12 an edge of a quadratic curve, a shape walker 130 may identify those pixels to an integrator
13 140, as explained further below.

14 [76] In particular embodiments, once the x-min and x-max values that are defining
15 the pixels intersecting with an edge of a primitive have been determined for each row within
16 a tile, a shape walker 130 may be configured to assign each pixel within the tile a coverage
17 weight or flagged for the integrator 140. Pixels that are overlapping with an edge of a
18 primitive are flagged and provided to an integrator 140. Pixels that are fully outside a
19 primitive are assigned a coverage weight of 0. Pixels that are fully inside a primitive are
20 assigned a coverage weight of 1. A shape walker 130 may be configured to assign every pixel
21 outside the bounding box a coverage weight of 0. To evaluate pixels that are inside the
22 bounding box, a shape walker 130 may walk through each pixel row-by-row. For example,
23 referring to FIG. 4B, a shape walker 130 may start from y-min 453 and determine that y-min
24 453 corresponds to a top edge of a trapezoid. A shape walker 130 may then assign a coverage
25 weight of 0 to pixels that are located to the left of the previously determined x-min value for
26 this row. A shape walker 130 may also determine that pixels that are located to right of the of
27 the x-min value for that row intersect with the top edge of the trapezoid and flag those pixels
28 to the integrator 140. For the row corresponding to y-min 453 + 1, a shape walker may
29 similarly determine that the row also corresponds to the top edge and assign a coverage
30 weight of 0 to pixels that are located to the left of the x-min value previously determined for
31 that row and flag pixels that are located to right of the x-min value for the integrator 140. For
32 the row corresponding to y-min 453 + 2, a shape walker may determine that that this row
33 corresponds to a left-side edge of a trapezoid. A shape walker may then assign a coverage
34 weight of 0 to pixels that are located to the left of the corresponding x-min value, flag pixels

1 that are between x-min and x-max (including pixels having x-min and x-max values), and
2 assign a coverage weight of 1 to pixels that are located to right of the corresponding x-max
3 value. A shape walker may repeat these steps for each row within the bounding box until all
4 pixels within the bounding box are either assigned a coverage weight or flagged for the
5 integrator 140. This example technique may similarly be applied to tiles containing other
6 edges of a trapezoid. For example, if the tile 423 shown in FIG. 4B included the right side
7 edge of a trapezoid rather than a left side edge, pixels to the left of the edge would be
8 assigned a coverage weight of 1, while pixels to the right of the edge would be assigned a
9 coverage weight of 0. In particular embodiments, pixels that are intersecting with a top edge
10 or a bottom edge of a trapezoid may be flagged for an integrator 140. The above example
11 technique may similarly be applied to tiles containing an edge of a curve based on the x-min
12 and x-max values determined based on a linear equation (for the flat line) or a quadratic
13 equation (for the curved line).

14 [77] In particular embodiments, a shape walker 130 may be configured to examine,
15 prior to determining coverage weights of pixels that are fully outside or inside a primitive and
16 prior to flagging pixels that are intersecting with an edge of a primitive, whether the tile
17 bounding box is bigger than a minimal threshold size. If the bounding box is smaller than a
18 minimal threshold size (such as 1x1 pixel or 2x2 pixels), a shape walker 130 may be
19 configured to send all of the pixels within the bounding box to an integrator 140 to determine
20 their respective coverage weight, rather than going through the steps described in the
21 preceding paragraphs. In an embodiment, determining whether the bounding box is bigger
22 than a threshold size may be implemented for a trapezoid but not for a quadratic curve.

23 [78] An integrator 140 may be configured to determine anti-alias pixel coverage
24 weights for each pixel flagged by a shape walker 130. Pixels that are assigned a coverage
25 weight by an integrator 140 are forwarded to a coverage buffer 151 or 152. An integrator 140
26 may only be responsible for determining coverage weights for pixels that are flagged by a
27 shape walker 130, for example, pixels that intersect an edge of a primitive. As discussed
28 above, coverage weights for pixels that are fully outside or fully inside a primitive are
29 assigned by a shape walker 130. The technique of determining the anti-alias pixel coverage
30 weights for each pixel flagged by a shape walker 130 (e.g., partially covered by a primitive)
31 involves utilizing the well-understood property of a trapezoid or a quadratic curve function.
32 An example of such a technique is disclosed in the following paper, which is incorporated
33 herein: A. Ellis, W. Hunt, J. Hart, Nerf: *Real-Time Analytic Antialiased Text for 3-D*
34 *Environments*, Computer Graphics forum, vol. 38, issue 8, November 2019, pp. 23-32.

1 [79] In particular embodiments, coverage buffers 151 and 153 may be configured
2 to store and maintain coverage weights for pixels, as determined by either a shape walker 130
3 or an integrator 140. In particular embodiments, two coverage buffers may be configured in a
4 double buffer configuration such that one coverage buffer is assigned to the rasterization
5 process while the other is assigned to the shading process, then alternating the roles as
6 necessary. For example, referring to FIG. 1A, the double buffer configuration allows a first
7 coverage buffer (e.g., 151) to be updated by a shape walker 130 and integrator 140, while a
8 second coverage buffer (e.g., 153) can be accessed by other components of the system, for
9 example, a shader 150.

10 [80] In particular embodiments, each coverage buffer may be configured to store a
11 coverage weight for each pixel within a tile. A coverage weight of zero represents full
12 transparency, and a value of 1 (or in some embodiments $2^{10}-1$ (i.e., 1023)) represents a fully
13 opaque. Intermediate values between full transparency and fully opaque represent partially
14 transparent pixels that can be combined with a background image to yield a composite image.
15 As discussed previously, in accordance to embodiments, instructions to update the coverage
16 buffer for pixels that are fully transparent or fully opaque are received from a shape walker
17 130 and instructions to update the coverage buffer for pixels that are partially transparent are
18 received from an integrator 140.

19 [81] In particular embodiments, a shader 150 may be configured to perform fixed
20 function shading of the pixels of a primitive. In particular embodiments, a shader 150 may be
21 configured to perform any of the following types of shading operations: solid fill, gradient
22 fill, and texturing. Texturing involves invoking a texture unit 170. In particular embodiments,
23 a shader 150 performs shading operations tile by tile, and for each tile, pixel by pixel based
24 on the coverage weight associated with each pixel. A shader 150 may be configured to
25 determine the source color information and the determined information may be passed on to a
26 color buffer 191 or 193 for blending operations. In an embodiment, a shader 150 generates
27 the texture space coordinates by transforming the conversion matrix from the shader
28 information into texel space coordinates. A shader 150 may then be configured to adjust for
29 the shear and then clamps the output to send it to the texture block.

30 [82] In particular embodiments, color buffers 191 and 193 may be configured to
31 perform blending operations. In particular embodiments, two color buffers may be configured
32 in a double buffer configuration to allow one color buffer to be updated while the other is
33 being accessed. Color buffers 191 and 193 may receive the source color information and
34 pixel coverage weights from a shader 150 or a blit and filtering unit 180. Based on a gamma

1 correction mode, color buffers 191 and 193 may be configured to convert the input source
2 color into gamma space before performing a blending operation. Once converted, the output
3 may be converted back to linear space using the degamma unit. Such gamma conversion
4 steps are optional. After color buffers 191 or 193 finishes the blending operations, the
5 blended color data may be streamed out to the tile compress and store 195. The blended color
6 data may be streamed out in a block by block fashion (e.g., 4x4 pixel arrays).

7 [83] As discussed above, in an embodiment, a command controller 107 may
8 determine, for each tile containing at least a portion of a primitive, a tile bounding box that
9 encompasses the at least the portion of the primitive. This technique may be referred as a
10 “culling” technique where tiles of a frame (e.g., 16x16 pixels) are culled using a smallest
11 bounding box that encompasses a primitive being processed by the graphics processing unit
12 (GPU), or a graphics system. Only the tiles covered by the primitive bounding box may be
13 identified to the downstream GPU components in the rendering pipeline to allow the
14 downstream GPU components to effectively ignore the empty tiles (i.e., tiles that are
15 completely outside any primitive bounding box). This reduces the overall computing required
16 and makes the system more efficient. Incorporating this culling technique, however, presents
17 a challenge because the technique conflicts with some of the blending modes that are used to
18 blend overlapping primitives. Examples of such blending modes may include blending modes
19 that are referred to as src, srcIn, srcOut, dstAtop, and dstIn (hereinafter referred to as “special
20 blending modes”). These special blending modes require access to both the tiles covering the
21 destination primitive (tiles already in the color buffer) and the tiles covering the source
22 primitive (tiles that are to be written into the color buffer). For example, the special blending
23 modes may require access to the tiles covering the source primitive to update the color
24 information of the pixels in those tiles while also requiring access to the tiles covering the
25 destination primitive to clear/update/remove the color information of the pixels in the tiles
26 covering the destination primitive. However, due to the culling technique, when a graphics
27 system is processing a source primitive, the graphics system only has access to tiles covering
28 the source primitive and do not have access to the tiles of the destination primitive.
29 Embodiments disclosed herein provide a technique for addressing this challenge. Blending
30 modes that do not require updating the pixels in the tiles covering the destination primitive
31 are referred herein as “normal blending modes.” Operations that involve special blending
32 modes may herein be referred to as “special blending operations.” Operations that involve
33 normal blending modes may herein be referred to as “normal blending operations.”

34 [84] References to a destination primitive herein may refer to a “shape” that is

1 stored in a color buffer, which may be a primitive or a blend of multiple primitives that have
2 been blended into the color buffer. References to a source primitive herein may similarly
3 refer to a “shape” that is to be stored/blended into a color buffer, which may be a primitive.

4 [85] In particular embodiment, a graphic system may be configured to implement
5 the blending operations sequentially, primitive by primitive. This means that, when the
6 system is processing a particular primitive, only the tiles covered by the primitive are
7 processed by the system while other tiles are ignored. If, for example, a particular frame
8 comprises multiple primitives, each of the primitive may be processed one at a time, in a
9 sequence, which may require processing the same tiles multiple times if multiple primitives
10 are covered by the tiles. **FIG. 6** illustrates an example frame with two primitives, a
11 destination primitive 610 and a source primitive 630. The destination primitive 610
12 represents a primitive that is already stored in a color buffer, while the source primitive 630
13 represents a primitive that is to be written into the color buffer. Tiles that are covering the
14 destination primitive 610 may be referred herein as destination tiles and tiles that are covering
15 the source primitive 630 may be referred herein as source tiles. When blending two
16 overlapping primitives, such as those illustrated in FIG. 6, special blending modes require an
17 operation where the primitive in the destination tiles are cleared of the pixel values, but as
18 discussed above, a graphics system may not have access to the destination tiles.

19 [86] In particular embodiments, the task of clearing a destination primitive may
20 first involve categorizing the tiles in a frame as “non-empty tiles” when the tiles cover a
21 source primitive and as “empty tiles” when the tiles do not cover the source primitive. For
22 example, in FIG. 6, the tiles within the dotted outline 643 may be categorized as non-empty
23 tiles since a source primitive 630 touches each of those tiles. Tiles that are outside the dotted
24 outline 643 may be categorized as empty tiles since none of them touch the source primitive
25 630.

26 [87] In particular embodiments, when executing a special blending mode, a
27 graphics system may clear a destination primitive from empty tiles by instructing the color
28 buffer to bypass the primitive cull associated with a source primitive (e.g., bounding box of
29 the source primitive) to allow the color buffer to gain access to previously inaccessible tiles
30 (e.g., tiles that are beyond the source primitive’s bounding box). The color buffer may then
31 be configured to clear the empty tiles by updating the pixel values associated each pixel
32 within the empty tiles (e.g., tiles that are beyond the source primitive’s bounding box and
33 associated with a destination primitive/shape). Alternatively, a graphics system may be
34 configured to instruct the color buffer to process a dummy primitive (e.g., a primitive

1 associated with clear color values) that overlaps the destination primitive, effectively
2 “clearing” the color information of the destination primitive by replacing it with clear color
3 information.

4 [88] For non-empty tiles, the clearing task is a bit more complicated since only the
5 destination primitive must be cleared from the non-empty tiles are covering both the
6 destination primitive and the source primitive. For example, in FIG. 6, the clearing task
7 would require clearing only a portion of tile 645 covering a destination primitive 610 without
8 also clearing the portion of the tile 645 covering a source primitive 630. The techniques
9 disclosed above with reference to clearing the empty tiles—as opposed to non-empty tiles
10 here—will not be appropriate for tile 645 since, for example, implementing the above
11 techniques may also clear the source primitive within tile 645. Embodiments disclosed
12 herein, therefore provide a solution to this problem by utilizing a pixel-by-pixel analysis to
13 identify particular pixels within a tile that is only associated with a destination primitive then
14 selectively clearing the pixel values associated with the identified pixels.

15 [89] For clearing a destination primitive from non-empty tiles, in accordance to
16 particular embodiments, a graphics system may maintain status bits for each of the pixels in
17 the non-empty tiles that track the recent blending mode(s) that has been used for that pixel or
18 whether the most recent blending mode used for that pixel is a normal blending mode or a
19 special blending mode. The graphics system may use the status bits to identify pixels that
20 have been touched by the most recent normal blending operation, i.e., pixels covering a
21 destination primitive. In particular embodiments, a graphics system assigns a primitive a
22 blending mode (normal blending mode or special blending mode) before the primitive is
23 blended into a color buffer. For example, referring to FIG. 6, a graphics system may have
24 assigned the destination primitive 610 a normal blending mode before it was blended into the
25 color buffer and the source primitive 630 with a special blending mode before it is blended
26 into the color buffer. Pixel values associated with a primitive are similarly associated with
27 data indicating whether it is associated with a normal blending mode or a special blending
28 mode.

29 [90] In particular embodiments, a graphics system may be configured to utilize
30 status bit W0 to indicate whether a pixel has been touched by a normal blending mode and
31 status bit W1 to indicate whether a pixel has been touched by a special blending mode. For
32 example, status bits “00” (equivalent to side-by-side status bits W1 and W0) is used to
33 indicate that a pixel has not been touched by any blending operations, and thus pixel values
34 associated with the pixel should correspond to the background color of a frame. Status bits

1 “01” is used to indicate that a pixel has been touched by a normal blending mode. Status bits
2 “10” is used to indicate that a pixel has been touched by a special blending mode. Status bits
3 “11” is used to indicate that a pixel has been touched by both normal and special blending
4 modes. For example, in FIG. 6, when the source primitive 630 is blended into the color
5 buffer, pixels that are covering only the destination primitive 610 may be associated with
6 status bits 01, pixels that are covering only the source primitive 630 may be associated with
7 status bits 10, pixels that are covering both the destination primitive 610 and the source
8 primitive 630 (the overlapping region) may be associated with status bits 11, and pixels that
9 are not associated either primitives may be associated with status bits 00. In an embodiment,
10 when overlapping pixels for which status bits will be 10, appropriate blending operation may
11 be performed by using background color information as destination color. Whereas when
12 status bit 11 is encountered, appropriate blending operation is performed by reading the color
13 from color memory as destination color

14 [91] In particular embodiments, at the end of each special blending operation, a
15 graphics system may be configured to implement a “flag treatment step” by which status bits
16 are reset such that status bits 00 remains as 00, status bits 01 are changed to 00, and status
17 bits 10 and 11 are changed to 01. When the graphics system finishes blending a frame, the
18 graphics system may be configured to export the color information of the pixels based on the
19 current status bits: for pixels with status bits 00, the graphics system may export the
20 background color information rather than retrieving the color information from the color
21 memory; for pixels with status bits 01, the graphics system may export the color information
22 from the color memory. Once the flag treatment step is executed at the end of a special
23 blending operation, the graphics system may be able to identify pixels that have been touched
24 by that special blending operations by searching for pixels that are associated with status bits
25 01. Other pixels in the frame should be associated with status bits 00 due to the resetting
26 process discussed above with reference to the flag treatment step. And, as discussed above,
27 when exporting the color information for pixels associated with status bits 00, the graphics
28 system may not retrieve the color information from the color buffer, rather the system may
29 simply retrieve/use the background color information. The use of the background color
30 information when exporting the pixel color information is effectively equivalent to clearing
31 the pixel values associated the pixels with status bits 00 since pixels without any value
32 correspond to the background color. In other words, the flag treatment step effectively clears
33 out the destination primitive since the pixels that has been touched only by a normal blending
34 mode (status bits 00 and 01 before the flag treatment step) are changed to 00 and background

1 color information is exported for those pixels. In some embodiments, the flag treatment step
2 is executed not at the end of special blending operation but prior to the beginning of a
3 subsequent blending mode that follows a special blending operation. In an embodiment, a
4 graphics system also maintains additional status data indicating the previous blending mode
5 that has touched a pixel, if any, to determine the transition between the blending operations.

6 [92] References to pixel values or pixel color information as used herein may refer
7 to any of the red, green, or blue color channels, and/or opaqueness channel.

8 [93] In particular embodiments, a texture unit 170 may be configured to provide
9 texture information for pixel covered by a primitive and shades the color of the pixel. If the
10 covered pixel has texture fill, then corresponding texture image may be fetched and filtered to
11 obtain the color information for the covered pixel. The covered pixel may then be shaded
12 with the derived color.

13 [94] In particular embodiments, a tile compress store 195 may be configured to
14 receive the rendered tile data from color buffers 191 and 193. A tile compress store 195 may
15 comprise a block encoder (e.g., hardware encoder) that is configured to encode the rendered
16 tile data before being transmitted to a display driver 198. In particular embodiments, a tile
17 compress store 195 may be responsible for encoding static assets (e.g., a blit such as an
18 emoji, a company logo, or a watch face for a smart watch), which may be stored a memory
19 external to the graphic engine 103 to be accessed at a later time point. Static images need to
20 be encoded at low power but with high throughput. To achieve such a feat, when encoding an
21 image (asset), a tile compress store 195 may use a “spatial prediction” technique that
22 leverages the fact that some groups of pixels in an image comprises the same pixel values as
23 other groups. Additional details for this technique are described below.

24 [95] FIG. 7 illustrates an example encoding pipeline. Tiles that are encoded by the
25 encoding system are piped through a double buffer 751 such that the current tile can be
26 compressed while the next tile is streamed in. For each tile to be encoded, a block scheduler
27 753 may separate the tile into blocks for the encoder. A block scheduler 753 may schedule
28 the blocks in an arrangement that is optimized for delta coding, for example, in an
29 arrangement that minimizes the spatial distance between the blocks in a sequence. An
30 example of such an arrangement is called the “Morton Order.” FIG. 8 illustrates a tile 782
31 that is segmented into multiple blocks, e.g., block 874, each block comprising multiple pixels
32 or texel, e.g., 4x4 pixels/texels. A block encoder 760 may be configured to encode blocks in a
33 tile in an arrangement specified by a block scheduler. For a tile comprising pixels of multiple
34 channels, or components, a block encoder 760 may be configured to encode each pixel

1 channel separately. Examples of pixel channels or components are color components (e.g., R,
2 G, B) or an opaque component (e.g., transparency). The encoded channels may then be
3 collated into a single bitstream. The encoded data may be provided to a memory write
4 controller 760. A memory write controller 760 may then send the encoded data to a memory
5 to be stored and made accessible for later retrieval by a graphics engine.

6 [96] FIG. 9 illustrates an example encoding pipeline executed by a block encoder
7 760. In particular embodiments, a tile compress store 195 may be configured to encode an
8 image based on groups of texels, each of which may be referred to as a “block.” A block may
9 be comprised of, e.g., 4x4 texels. A block encoder 760 may comprise a block analyzer 905, a
10 spatial predictor 901, a texel scheduler 901, texel scheduler 910, delta coder 920, channel
11 entropy coder 930, and channel data collator 940. In an embodiment, the encoding pipeline
12 illustrated in FIG. 9 represents an encoding pipeline of a hardware encoder, but substantially
13 similar pipeline may be implemented as a software encoder. Each of the system components
14 illustrated in FIG. 9 may be configured to operate based on an encoding cycle where each
15 system component processes one block per one encoding cycle.

16 [97] In an embodiment, a spatial predictor 901 may be configured to compare the
17 texel values of the current block to previously processed blocks, if any, to determine whether
18 the texel values of the current block matches the texel values of any of the previously
19 processed blocks. For example, a spatial predictor 901 may compare the texel values of the
20 current block with the texel values of up to four of the previously processed blocks. If a
21 matching block is found, the spatial predictor 901 may forgo encoding the texel block of the
22 current block and instead assign a block header to the current block that matches a block
23 header of the matching block. Such a technique allows a block encoder 760 to skip the
24 encoding process for the current block since the duplication of the block header allows the
25 matching block’s compressed block data to be utilized for both the current block and the
26 matching block. However, there is a power-efficiency concern with the above described
27 technique because texel-by-texel comparison of blocks requires a significant amount of
28 compute power and memory storage. As a solution, embodiments disclose a technique of
29 generating a hash code, or hash representation, to represent the texel values of each block and
30 using the hash codes to make the comparison rather than comparing the actual texel values of
31 the blocks. In an embodiment, a block encoder 760 may be configured to generate hash
32 representations that are 32-bits or 64-bits. Notably, a 32-bit or 64-bit block hash comparison
33 is significantly cheaper, computationally, than comparing the 4x4 block data.

34 [98] There exists yet another problem with the technique discussed above with

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

16 [99] FIG. 10 illustrates an example of the techniques described above with
17 reference to a spatial predictor 901. When a spatial predictor 901 processes a block, it may be
18 configured to first analyze the texel values associated with the block to validate whether the
19 block comprises valid texel values, as opposed to having no value or null value. If the block
20 includes valid texel values, a spatial predictor 901 may be configured to generate a hash
21 representation of the texel values associated with the block, via hash function 1020. Then, the
22 spatial predictor 901 may be configured to compare the hash representation of the current
23 block with the hash representation of blocks that were previously processed by the spatial
24 predictor 901. If a match is found for the current block, a spatial predictor 901 may duplicate
25 a block header for the current block that matches the block header of the matching block. For
26 example, as illustrated in FIG. 8, a spatial predictor 901 may maintain a table 1010
27 comprising data associated with up to four previously processed blocks with respect to the
28 current block. Such a table 1010 may be used to store data indicating whether a block is
29 associated with valid texel values (e.g., in column 1031), hash representation of the texel
30 values of the block (e.g., in column 1032), and block header or placeholder tag for the block
31 (e.g., in column 1033). As noted above, if a matching block is found but the block header of
32 the matching block has not been generated yet, a spatial predictor 901 may be configured to
33 generate a placeholder tag, for example, "tag_blockHeader 3" in FIG. 8. A copy of such a tag
34 may be sent along the encoding pipeline illustrated in FIG. 9. At the end of each encoding

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

21 [100] Referring back to FIG. 9, after a spatial predictor 901 completes processing
22 the current block, the current block may be passed onto the subsequent downstream
23 components of the encoding pipeline. Examples of such downstream components of the
24 encoding pipeline include a block analyzer 905, texel scheduler 910, delta coder 920, channel
25 entropy coder 930, and channel data collator 940. Described below are techniques used by the
26 downstream components to analyze and encode texel values of blocks.

27 [101] In particular embodiments, a block analyzer 905 may be configured to analyze
28 texel blocks and categorize them into one of two block variants: Flatblock or Codeblock. A
29 block may be categorized as a Flatblock if all texels in the block have the same value. A
30 block may be categorized as a Codeblock if some of the texels in the block have different
31 values. Once a block is categorized as a Flatblock or a Codeblock, a block analyzer 905 may
32 be configured to pass the block to a texel scheduler 910.

33 [102] In particular embodiments, a texel scheduler 910 may be configured to
34 schedule the texels in a block (e.g., Codeblock) in a sequence optimized for delta encoding.

1 For example, the texels in a block may be scheduled in a Morton Order shown in FIG. 8. The
2 arranged texels may then be provided to a delta coder 920. A texel scheduler 910 may be
3 configured to schedule the texels of a Codeblock and, but not for a Flatblock since delta
4 coding is not necessary for a Flatblock.

5 [103] In particular embodiments, a delta coder 920 may be configured to encode a
6 texel block using various techniques. For a Flatblock, a delta coder 920 may be configured
7 encode the block using a single texel value since a Flatblock contains only a single texel
8 value. For a Codeblock having multiple texel channels (e.g., R, G, B, opacity), a delta coder
9 920 may be configured to encode each texel channel separately from each other, and different
10 encoding techniques may be used to encode each channel. The different encoding techniques
11 used by a delta coder 92 may include a “flat” technique, “variable-length” technique, and an
12 uncompressed technique which essentially involves “encoding” (e.g., storing) texel values as
13 uncompressed. These encoding techniques may also be referred to as compression modes, for
14 example, “variable length” mode, “flat” mode, or uncompressed mode. A particular channel
15 of a Codeblock may be encoded using a “flat” technique if all of the values of the texels in
16 the channel are the same. The flat technique involves using a single value to represent the
17 entire channel. A particular channel of a Codeblock may be encoded using a “variable-
18 length” technique if values of the texels within the channel differ from each other. The
19 variable-length technique is a novel compression technique that produces different sizes of
20 encoded data depending on the differences in the texel values within the block. As for the
21 uncompressed technique, while it may involve storing the corresponding pixel values as
22 uncompressed (e.g., without any compression), for the purposes of describing the
23 embodiments herein, the uncompressed technique/mode may still be referred to as one of the
24 “compression” techniques/modes used to “encode” texel values of a texel block, and its
25 operations may be described as the process of “compressing” the texel values.

26 [104] In particular embodiments, the variable-length technique may involve
27 generating three groups of data to represent the encoded texel values: “symbolmask”; “rbits”;
28 “rsymbols.” Data group rsymbols is used to represent the non-zero delta values of the texel
29 values as arranged by a texel scheduler 910 (e.g., in a Morton Order). For example, if there
30 are 16 texel values in a sequence, there would be 15 delta values, each delta value
31 representing the difference of one texel value to the next in that sequence, or the difference of
32 one texel value to the previous texel value in that sequence if considering how the sequence
33 of texel values may be read by a block encoder 760. Data group rsymbols is used to represent
34 only the non-zero delta of those 15 delta values. Data group symbolmask is used to provide a

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

16 **[105] FIG. 12** illustrates an example diagram for encoding a 4x4 texel block using a
17 variable-length technique. Specifically, **FIG. 12** illustrates a 4x4 texel block 872 comprising
18 16 texel values, which when arranged in a Morton Order are as follows: [0, 0, 0, 0, 8, 8, 8, 8,
19 0, 0, 0, 0, 8, 8, 0, 0]. The delta values, or delta coded stream, of the texel values arranged in
20 the Morton Order would have 15 values and are as follows: [0, 0, 0, 8, 0, 0, 0, -8, 0, 0, 0, 8, 0,
21 -8, 0]. The rbits for these delta values would be 5 since a bit length of 5 (i.e., 5 bits) would be
22 required to represent each of the delta values, a first bit to indicate a positive or negative sign
23 of the delta values and four additional bits to represent the delta values’ maximum value of 8.
24 In an embodiment, the rbits may be encoded in a binary representation, such that rbits of 5
25 may be stored as [101]. In some embodiments, rbits may be stored with an offset, for
26 example, with an offset of 2 such that rbits of 5 may be stored as 3, or [011]. Storing rbits
27 with an offset increases the range of the values that rbits can represent and leverages the fact
28 that rbits of 0 or 1 would not be needed because, for example, the maximum number of bits
29 required to represent each of the delta values, which is what rbits represents, would always
30 require a value greater than 0 or 1. Continuing the example illustrated in **FIG. 12**, the
31 rsymbols for the delta values would be [-8, 8, -8, 8] or in binary representation [11000,
32 01000, 11000, 01000], where each value of rsymbols has a bit length of rbits (5 bits) with a
33 first bit used to indicate whether the delta value is a positive or negative value and four
34 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. 12, this sequence of values of symbolmask is presented in a reverse order with respect to how the delta values were presented in the previous steps. As discussed above, symbolmask are used to indicate whether a delta value is zero or non-zero. Notably, the data group rsymbols only needs to represent non-zero delta values since any zero delta values are already indicated by symbolmask. In addition to the three groups of data, the first value of the uncompressed texel values may be encoded as the “base value” of the encoded data, either encoded together with the three groups of data or separately as metadata. In the example above, the base value would be 0 since that is the first value of the uncompressed texel values. Once all three groups of data are generated for a texel block, they may be collated together into a stream of bits, for example, in the configuration shown in FIG. 11. The collated stream of bits represents the encoded data for a particular channel of a texel block. In an embodiment, a block metadata that is encoded with the encoded data may comprise data indicating the number of texel channels included in a block and the type of technique used to each of the channels. In an embodiment, extra bits may be encoded into the encoded data to make it byte-aligned. For example, as shown in FIG. 12, if the encoded data results in a bit length of 38 bits, two extra bits may be added to make it byte-aligned (e.g., multiples of 8 bits).

19 [106] As discussed above, each texel channel within a Codeblock may be independently encoded using any of the techniques described above (e.g., flat technique, variable-length technique, or uncompressed). For example, for a Codeblock having three channels of texel values, a first channel of the three may be encoded using a flat technique, a second channel of the three may be encoded using the variable-length technique, and a third channel of the three may be stored as uncompressed.

25 [107] In particular embodiments, after encoding a channel using the variable-length technique, a channel entropy coder 930 illustrated in FIG. 9 may be configured to evaluate whether the encoded channel data is computationally more expensive than the uncompressed channel data, that is, whether the encoded data requires more bits than the uncompressed data. If so, the channel entropy coder may disregard the encoded channel data and instead use the uncompressed channel data. In other words, for each channel/component of a texel block, a channel entropy coder 930 may determine whether to encode the texel values for that channel using one of the compression techniques disclosed above or based on the uncompressed texel values. In particular embodiments, a channel entropy coder 930 may evaluate the encoded block data of the Codeblock to see whether the encoded data is greater

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

16 [108] In particular embodiments, a channel data collator 940 may be configured to
17 collate each of the encoded, or uncompressed, channels of texel values into a bit stream that
18 results in the encoded block data. In particular embodiments, a channel data collator 940 may
19 generate a block header for each texel block. A block header may comprise a pointer (e.g., an
20 offset value) that indicates the location of the block data in the memory that is relative to
21 block data associated other blocks of an image. A block header may comprise data that
22 indicates whether the encoded texel block is compressed or uncompressed, the number of
23 texel channels in the block, and if the block is compressed, the size or length of the
24 compressed block data (e.g., measured in bits or bytes). In particular embodiment, as shown
25 in FIG. 9, once a block header is generated for a texel block, the block header may be
26 provided to a spatial predictor 901. The spatial predictor 901 may then evaluate whether any
27 texel block is associated with a placeholder tag that has been generated in place for the block
28 header, and if so, replace the placeholder tag with the block header.

29 [109] The encoding pipeline illustrated in FIG. 9 provides a unique way of encoding
30 the blocks that allows a decoder to selectively retrieve and decode any particular block of the
31 encoded blocks independently from other encoded pixel blocks. More specifically, each
32 block is encoded in a way that it is self-contained, meaning that a decoder can selectively
33 retrieve and decompress a particular block simply based on the data contained within the
34 block. For example, if a PNG image that encoded using the techniques described herein, a

1 decoder may be able to retrieve and decompress specific portions of the PNG image
2 independently from other portions of the PNG image.

3 [110] In particular embodiments, a blit and filtering unit 180 may be configured to
4 retrieve static graphics content from a memory database 109 and, if necessary, perform
5 decoding operations and/or transformation or filtering operations on the graphics content
6 referred to as a “blit” operation. A blit operation refers to a hardware feature that moves a
7 rectangular block of bits from main memory into display memory. In particular embodiments,
8 a graphics system disclosed herein may store static graphic content, such as pre-rendered
9 images (e.g., emoji), in a memory 109 that is external to the graphics engine. A blit and
10 filtering unit 180 may be configured to retrieve content from the memory and perform
11 transformation or filtering operations, then provide the transformed/filtered content to a color
12 buffer. In particular embodiments, a blit and filtering unit 180 may be configured to update
13 the input data based on the command it receives from a tile controller 120. A blit and filtering
14 unit 180 may include a memory structure, for example, a single color buffer. Incoming source
15 image information per tile may be buffered in this memory structure to improve the
16 performance of the blit and filtering unit 180. A blit and filtering unit 180 may perform a set
17 of predefined operations and filters. A blit and filtering unit 180 provides a power-
18 performance-area (PPA) optimized solution to some common data rearrangement/movement
19 (with filter) operations to the hardware. In particular embodiments, a blit and filtering unit
20 180 may comprises a decoder configured to decode static graphics content that has been
21 encoded and stored in a memory database 109. A blit and filtering unit 180 may be
22 configured to provide the decoded graphics content to a color buffer 191, 193.

23 [111] FIG. 13 illustrates an example technique of decoding a 4x4 texel block that
24 has been encoded by a block encoder 760. Specifically, FIG. 13 illustrates an encoded texel
25 data comprising three data groups, rsymbols, rbits, and symbolmask. As discussed above,
26 data group rsymbols is used to represent the non-zero delta values of the sequence of texel
27 values of a block as arranged in, for example, a Morton Order. Data group symbolmask is
28 used to provide a 1 to 1 mapping of the delta values that indicates whether each delta value is
29 a zero value or non-zero value. Data group rbits is used to indicate the maximum number of
30 bits required to represent each of the delta values, along with an additional bit to indicate
31 whether the delta values are positive or negative values. In an embodiment, a decoder may be
32 configured to decode multiple delta values per one decoding cycle. For example, FIG. 13
33 illustrates an embodiment where three multiplexers, i.e., symbolMUX 1312, 1314, 1316, are
34 configured to decode three segments of rsymbols (delta values) in parallel during each

1 decoding cycle. Although FIG. 13 illustrates an embodiment where three segments of
2 rsymbols are decoded per each decoding cycle, any number of segments may be configured
3 to be decoded per each cycle, for example, five segments of rsymbols in parallel. While FIG.
4 13 illustrates one instance of a decoding operation for a particular channel of texel values,
5 multiple of such instances may be configured to be implemented such that all texel channels
6 are decoded in parallel. Once all channels are decoded, the decoded values may be collated,
7 resulting in uncompressed texel values.

8 [112] In the embodiment illustrated in FIG. 13, for example, a decoder may be
9 configured to decode a 4x4 texel block having rbits of 8, which indicates that each segment
10 of rsymbols (e.g., each delta value) is 8 bits long. Given that the decoder is configured to
11 decode three delta values in parallel, rMUX 1301 may be configured to fetch up to three delta
12 values per decoding cycle, that is, up to 27 bits at a time. In an embodiment, a decoder may
13 be configured to implement an initializing operation where symbolmask is parsed to
14 determine the number of delta values rMUX that should be fetched in each decoding cycle.
15 For example, if the first three symbolmask bits are [101], indicating that the first and third
16 values are non-zero values and the second is a zero value, then rMUX 1301 may be
17 configured to fetch two segments of rsymbols for the first decoding cycle (first two delta
18 values). Also during the first decoding cycle, the first three symbolmask bits may be provided
19 to symbolMUX 1312, 1314, or 1316, respectively. For each zero value, a corresponding
20 symbolMUX (e.g., symbolMUX 1312, 1314, or 1316) may be configured to pass a zero value
21 to the next component in the decoder, for example, to a corresponding adder 1341, 1343, or
22 1345. For each non-zero value, a corresponding MUX (e.g., symbolMUX 1312, 1314, or
23 1316) may be configured to fetch a non-zero segment of rsymbols (delta value). For example,
24 if the first three symbolmask bits are [101] such that 1 is provided to symbolMUX 1316, 0 is
25 provided to symbolMUX 1314, and 1 is provided to symbolMUX 1312, then symbolMUX
26 1316 may fetch the first non-zero delta value from rMUX 1301, symbolMUX 1312 may fetch
27 the second non-zero delta value from rMUX 1301, and a zero value may be passed through
28 symbolMUX 1314. Once the first three delta values are retrieved by the respective
29 symbolMUX 1312, 1314, and 1316 (whether non-zero or zero), the delta values may be
30 passed to the respective adders 1341, 1343, and 1345. Then, a decoder may be configured to
31 add the base value of the encoded data to the first delta value to determine the second texel
32 value, add the resulting value to the second delta value to determine the third texel value, then
33 add the resulting value to the third delta value to determine the fourth texel value. In such a
34 fashion, the first four texel values of the encoded data may be determined after the first

1 decoding cycle, the first texel value being the base value. The next three texel values may
2 similarly be decoded during a second decoding cycle, and additional decoding cycles may
3 further be implemented until the block is decoded completely.

4 [113] FIG. 14 illustrates an example method 1400 for determining the color
5 information of primitives in an image base in part by determining the coverage weight of
6 each pixel in the image. The method may begin at step 1401 by receiving a list of primitives
7 covering a tile of an image that is to be rendered, the image comprising content defined by at
8 least the list of primitives. At step 1402, the method may continue by, for each primitive in
9 the list, identifying, in the tile, partially-covered pixels that are partially covered by the
10 primitive, fully-uncovered pixels that are fully uncovered by the primitive, and fully-covered
11 pixels that are fully covered by the primitive. At step 1403, the method may continue by, for
12 each primitive in the list, computing, for each of the partially-covered pixels, a coverage
13 weight indicating a proportion of the partially-covered pixel that is covered by the primitive.
14 At step 1404, the method may continue by, for each primitive in the list, storing coverage
15 data in a coverage buffer corresponding to the tile, the coverage data comprising the coverage
16 weights of the partially-covered pixels, fully-uncovered indicators for the fully-uncovered
17 pixels, and fully-covered indicators for the fully-covered pixels. At step 1405, the method
18 may continue by, for each primitive in the list, determining color information for the
19 primitive in the tile based on the stored coverage data. At step 1406, the method may
20 continue by, for each primitive in the list, aggregating the color information of the list of
21 primitives in a color buffer for output. Particular embodiments may repeat one or more steps
22 of the method of FIG. 14, where appropriate. Although this disclosure describes and
23 illustrates particular steps of the method of FIG. 14 as occurring in a particular order, this
24 disclosure contemplates any suitable steps of the method of FIG. 14 occurring in any suitable
25 order. Moreover, although this disclosure describes and illustrates an example method for
26 determining the color information of primitives in an image, this disclosure contemplates any
27 suitable method for determining the color information of primitives in an image including
28 any suitable steps, which may include all, some, or none of the steps of the method of FIG.
29 14, where appropriate. Furthermore, although this disclosure describes and illustrates
30 particular components, devices, or systems carrying out particular steps of the method of FIG.
31 14, this disclosure contemplates any suitable combination of any suitable components,
32 devices, or systems carrying out any suitable steps of the method of FIG. 14.

33 [114] FIG. 15 illustrates an example method 1500 for determining the color
34 information of a primitive base in part by determining the coverage weight of each pixel of

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

29 **[115] FIG. 16** illustrates an example method 1600 for blending source shape with a
30 destination shape using a blending mode that requires updates to pixels in the color buffer
31 uncovered by the source shape. The method may begin at step 1601 by receiving a source
32 shape that is to be blended with a destination shape stored in a color buffer for an image. The
33 following steps are performed in response to determining that the source shape is associated
34 with a blending mode that requires updates to pixels in the color buffer uncovered by the

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

22 **[116] FIG. 17** illustrates an example method 1700 for encoding blocks of pixels
23 based on a tag that is used to temporary represent block headers. The method may begin at
24 step 1701 by receiving a plurality of blocks of pixels of an image, wherein the blocks are to
25 be sequentially encoded using a hardware-encoding pipeline. At steps 1702-1706, the method
26 may continue by encoding a first block of the plurality of blocks. Specifically, at step 1702,
27 the method may continue by generating a first hash to represent the first block. At step 1703,
28 the method may continue by identifying a second hash stored in memory matching the first
29 hash, the second hash (i) representing a second block of the plurality of blocks previously
30 processed by the hardware-encoding pipeline and (ii) is associated with a tag corresponding
31 to a placeholder for a second header associated with the second block. At step 1704, the
32 method may continue by passing a copy of the tag through the hardware-encoding pipeline as
33 metadata for the first block. At step 1705, the method may continue by determining that the
34 second header is available. At step 1706, the method may continue by replacing the copy of

1 the tag with the second header to generate a first encoding for the first block, wherein the
2 second header specifies a memory region where a second encoding of the second block is
3 stored. Particular embodiments may repeat one or more steps of the method of FIG. 17,
4 where appropriate. Although this disclosure describes and illustrates particular steps of the
5 method of FIG. 17 as occurring in a particular order, this disclosure contemplates any suitable
6 steps of the method of FIG. 17 occurring in any suitable order. Moreover, although this
7 disclosure describes and illustrates an example method for encoding blocks of pixels based
8 on a tag that is used to temporary represent block headers, this disclosure contemplates any
9 suitable method for encoding blocks of pixels based on a tag that is used to temporary
10 represent block headers including any suitable steps, which may include all, some, or none of
11 the steps of the method of FIG. 17, where appropriate. Furthermore, although this disclosure
12 describes and illustrates particular components, devices, or systems carrying out particular
13 steps of the method of FIG. 17, this disclosure contemplates any suitable combination of any
14 suitable components, devices, or systems carrying out any suitable steps of the method of
15 FIG. 17.

16 [117] **FIG. 18** illustrates an example method 1800 for determining whether a block
17 of pixels is different from previously-compressed blocks and compressing the block using a
18 variable-length technique. The method may begin at step 1801 by determining a sequence for
19 compressing blocks of pixels in an image. At step 1802, the method may continue by
20 compressing the blocks sequentially according to the sequence, wherein a first component of
21 a first block is compressed, details of which are laid out in steps 1803 and 1807. At step
22 1803, the method may continue by selecting a variable-length mode from a plurality of
23 supported compression modes to compress the first component of the first block, which is
24 based on steps 1804-1806. At step 1804, the method may continue by determining that the
25 first block is different from previously-compressed blocks compressed according to the
26 sequence. At step 1805, the method may continue by determining that pixels within the first
27 component are different. At step 1806, the method may continue by determining that a bit
28 length needed for compressing the first component using the variable-length mode is less
29 than a bit length needed for representing the first component uncompressed. At step 1807, the
30 method may continue by generating a first compression of the first component of the first
31 block using a symbol width selected based on magnitudes of delta values used for encoding
32 the pixels within the first component of the first block. Particular embodiments may repeat
33 one or more steps of the method of FIG. 18, where appropriate. Although this disclosure
34 describes and illustrates particular steps of the method of FIG. 18 as occurring in a particular

1 order, this disclosure contemplates any suitable steps of the method of FIG. 18 occurring in
2 any suitable order. Moreover, although this disclosure describes and illustrates an example
3 method for determining whether a block of pixels is different from previously-compressed
4 blocks and compressing the block using a variable-length technique, this disclosure
5 contemplates any suitable method for determining whether a block of pixels is different from
6 previously-compressed blocks and compressing the block using a variable-length technique
7 including any suitable steps, which may include all, some, or none of the steps of the method
8 of FIG. 18, where appropriate. Furthermore, although this disclosure describes and illustrates
9 particular components, devices, or systems carrying out particular steps of the method of FIG.
10 18, this disclosure contemplates any suitable combination of any suitable components,
11 devices, or systems carrying out any suitable steps of the method of FIG. 18.

12 [118] FIG. 19 illustrates an example method 1900 for encoding a plurality of pixels
13 based on delta encoding that utilizes a base value, symbol mask, symbol width, and sequence
14 of symbols. The method may begin at step 1901 by receiving a block comprising a plurality
15 of pixels. At step 1902, the method may continue by encoding the plurality of pixels, details
16 of which are laid out in steps 1903-1908. At step 1903, the method may continue by
17 arranging the plurality of pixels in a sequence. At step 1904, the method may continue by
18 generating a delta encoding of the plurality of pixels, the delta encoding comprising (a) a base
19 value and (b) a plurality of delta values having non-zero delta values and zero delta values,
20 each delta value representing a difference between a corresponding pixel in the sequence and
21 a previous pixel in the sequence. At step 1905, the method may continue by generating a
22 symbol mask indicating whether each of the plurality of delta values is zero or non-zero. At
23 step 1906, the method may continue by determining, based on magnitudes of the non-zero
24 delta values, a symbol width for encoding each of the non-zero delta values. At step 1907, the
25 method may continue by generating a sequence of symbols that respectively encode the non-
26 zero delta values using the symbol width. At step 1908, the method may continue by
27 generating a compression of the block by collating the symbol mask, the symbol width, and
28 the sequence of symbols. Particular embodiments may repeat one or more steps of the
29 method of FIG. 19, where appropriate. Although this disclosure describes and illustrates
30 particular steps of the method of FIG. 19 as occurring in a particular order, this disclosure
31 contemplates any suitable steps of the method of FIG. 19 occurring in any suitable order.
32 Moreover, although this disclosure describes and illustrates an example method for encoding
33 a plurality of pixels based on delta encoding that utilizes a base value, symbol mask, symbol
34 width, and sequence of symbols, this disclosure contemplates any suitable method for

1 encoding a plurality of pixels based on delta encoding that utilizes a base value, symbol
2 mask, symbol width, and sequence of symbols including any suitable steps, which may
3 include all, some, or none of the steps of the method of FIG. 19, where appropriate.
4 Furthermore, although this disclosure describes and illustrates particular components,
5 devices, or systems carrying out particular steps of the method of FIG. 19, this disclosure
6 contemplates any suitable combination of any suitable components, devices, or systems
7 carrying out any suitable steps of the method of FIG. 19.

8 [119] FIG. 20 illustrates an example network environment 2000 associated with a
9 social-networking system. Network environment 2000 includes a client system 2030, a
10 social-networking system 2060, and a third-party system 2070 connected to each other by a
11 network 2010. Although FIG. 20 illustrates a particular arrangement of client system 2030,
12 social-networking system 2060, third-party system 2070, and network 2010, this disclosure
13 contemplates any suitable arrangement of client system 2030, social-networking system
14 2060, third-party system 2070, and network 2010. As an example and not by way of
15 limitation, two or more of client system 2030, social-networking system 2060, and third-party
16 system 2070 may be connected to each other directly, bypassing network 2010. As another
17 example, two or more of client system 2030, social-networking system 2060, and third-party
18 system 2070 may be physically or logically co-located with each other in whole or in part.
19 For example, an AR/VR headset 2030 may be connected to a local computer or mobile
20 computing device 2070 via short-range wireless communication (e.g., Bluetooth). Moreover,
21 although FIG. 20 illustrates a particular number of client systems 2030, social-networking
22 systems 2060, third-party systems 2070, and networks 2010, this disclosure contemplates any
23 suitable number of client systems 2030, social-networking systems 2060, third-party systems
24 2070, and networks 2010. As an example and not by way of limitation, network environment
25 2000 may include multiple client system 2030, social-networking systems 2060, third-party
26 systems 2070, and networks 2010.

27 [120] This disclosure contemplates any suitable network 2010. As an example and
28 not by way of limitation, one or more portions of network 2010 may include a short-range
29 wireless network (e.g., Bluetooth, Zigbee, etc.), an ad hoc network, an intranet, an extranet, a
30 virtual private network (VPN), a local area network (LAN), a wireless LAN (WLAN), a wide
31 area network (WAN), a wireless WAN (WWAN), a metropolitan area network (MAN), a
32 portion of the Internet, a portion of the Public Switched Telephone Network (PSTN), a
33 cellular telephone network, or a combination of two or more of these. Network 2010 may
34 include one or more networks 2010.

1 **[121]** Links 2050 may connect client system 2030, social-networking system 2060,
2 and third-party system 2070 to communication network 2010 or to each other. This disclosure
3 contemplates any suitable links 2050. In particular embodiments, one or more links 2050
4 include one or more wireline (such as for example Digital Subscriber Line (DSL) or Data
5 Over Cable Service Interface Specification (DOCSIS)), wireless (such as for example Wi-Fi,
6 Worldwide Interoperability for Microwave Access (WiMAX), Bluetooth), or optical (such as
7 for example Synchronous Optical Network (SONET) or Synchronous Digital Hierarchy
8 (SDH)) links. In particular embodiments, one or more links 2050 each include an ad hoc
9 network, an intranet, an extranet, a VPN, a LAN, a WLAN, a WAN, a WWAN, a MAN, a
10 portion of the Internet, a portion of the PSTN, a cellular technology-based network, a satellite
11 communications technology-based network, another link 2050, or a combination of two or
12 more such links 2050. Links 2050 need not necessarily be the same throughout network
13 environment 2000. One or more first links 2050 may differ in one or more respects from one
14 or more second links 2050.

15 **[122]** In particular embodiments, client system 2030 may be an electronic device
16 including hardware, software, or embedded logic components or a combination of two or
17 more such components and capable of carrying out the appropriate functionalities
18 implemented or supported by client system 2030. As an example and not by way of
19 limitation, a client system 2030 may include a computer system such as a VR/AR headset,
20 desktop computer, notebook or laptop computer, netbook, a tablet computer, e-book reader,
21 GPS device, camera, personal digital assistant (PDA), handheld electronic device, cellular
22 telephone, smartphone, augmented/virtual reality device, other suitable electronic device, or
23 any suitable combination thereof. This disclosure contemplates any suitable client systems
24 2030. A client system 2030 may enable a network user at client system 2030 to access
25 network 2010. A client system 2030 may enable its user to communicate with other users at
26 other client systems 2030.

27 **[123]** In particular embodiments, social-networking system 2060 may be a network-
28 addressable computing system that can host an online social network. Social-networking
29 system 2060 may generate, store, receive, and send social-networking data, such as, for
30 example, user-profile data, concept-profile data, social-graph information, or other suitable
31 data related to the online social network. Social-networking system 2060 may be accessed by
32 the other components of network environment 2000 either directly or via network 2010. As
33 an example and not by way of limitation, client system 2030 may access social-networking
34 system 2060 using a web browser, or a native application associated with social-networking

1 system 2060 (e.g., a mobile social-networking application, a messaging application, another
2 suitable application, or any combination thereof) either directly or via network 2010. In
3 particular embodiments, social-networking system 2060 may include one or more servers
4 2062. Each server 2062 may be a unitary server or a distributed server spanning multiple
5 computers or multiple datacenters. Servers 2062 may be of various types, such as, for
6 example and without limitation, web server, news server, mail server, message server,
7 advertising server, file server, application server, exchange server, database server, proxy
8 server, another server suitable for performing functions or processes described herein, or any
9 combination thereof. In particular embodiments, each server 2062 may include hardware,
10 software, or embedded logic components or a combination of two or more such components
11 for carrying out the appropriate functionalities implemented or supported by server 2062. In
12 particular embodiments, social-networking system 2060 may include one or more data stores
13 2064. Data stores 2064 may be used to store various types of information. In particular
14 embodiments, the information stored in data stores 2064 may be organized according to
15 specific data structures. In particular embodiments, each data store 2064 may be a relational,
16 columnar, correlation, or other suitable database. Although this disclosure describes or
17 illustrates particular types of databases, this disclosure contemplates any suitable types of
18 databases. Particular embodiments may provide interfaces that enable a client system 2030, a
19 social-networking system 2060, or a third-party system 2070 to manage, retrieve, modify,
20 add, or delete, the information stored in data store 2064.

21 **[124]** In particular embodiments, social-networking system 2060 may store one or
22 more social graphs in one or more data stores 2064. In particular embodiments, a social graph
23 may include multiple nodes—which may include multiple user nodes (each corresponding to
24 a particular user) or multiple concept nodes (each corresponding to a particular concept)—
25 and multiple edges connecting the nodes. Social-networking system 2060 may provide users
26 of the online social network the ability to communicate and interact with other users. In
27 particular embodiments, users may join the online social network via social-networking
28 system 2060 and then add connections (e.g., relationships) to a number of other users of
29 social-networking system 2060 to whom they want to be connected. Herein, the term “friend”
30 may refer to any other user of social-networking system 2060 with whom a user has formed a
31 connection, association, or relationship via social-networking system 2060.

32 **[125]** In particular embodiments, social-networking system 2060 may provide users
33 with the ability to take actions on various types of items or objects, supported by social-
34 networking system 2060. As an example and not by way of limitation, the items and objects

1 may include groups or social networks to which users of social-networking system 2060 may
2 belong, events or calendar entries in which a user might be interested, computer-based
3 applications that a user may use, transactions that allow users to buy or sell items via the
4 service, interactions with advertisements that a user may perform, or other suitable items or
5 objects. A user may interact with anything that is capable of being represented in social-
6 networking system 2060 or by an external system of third-party system 2070, which is
7 separate from social-networking system 2060 and coupled to social-networking system 2060
8 via a network 2010.

9 [126] In particular embodiments, social-networking system 2060 may be capable of
10 linking a variety of entities. As an example and not by way of limitation, social-networking
11 system 2060 may enable users to interact with each other as well as receive content from
12 third-party systems 2070 or other entities, or to allow users to interact with these entities
13 through an application programming interfaces (API) or other communication channels.

14 [127] In particular embodiments, a third-party system 2070 may include a local
15 computing device that is communicatively coupled to the client system 2030. For example, if
16 the client system 2030 is an AR/VR headset, the third-party system 2070 may be a local
17 laptop configured to perform the necessary graphics rendering and provide the rendered
18 results to the AR/VR headset 2030 for subsequent processing and/or display. In particular
19 embodiments, the third-party system 2070 may execute software associated with the client
20 system 2030 (e.g., a rendering engine). The third-party system 2070 may generate sample
21 datasets with sparse pixel information of video frames and send the sparse data to the client
22 system 2030. The client system 2030 may then generate frames reconstructed from the
23 sample datasets.

24 [128] In particular embodiments, the third-party system 2070 may also include one
25 or more types of servers, one or more data stores, one or more interfaces, including but not
26 limited to APIs, one or more web services, one or more content sources, one or more
27 networks, or any other suitable components, e.g., that servers may communicate with. A
28 third-party system 2070 may be operated by a different entity from an entity operating social-
29 networking system 2060. In particular embodiments, however, social-networking system
30 2060 and third-party systems 2070 may operate in conjunction with each other to provide
31 social-networking services to users of social-networking system 2060 or third-party systems
32 2070. In this sense, social-networking system 2060 may provide a platform, or backbone,
33 which other systems, such as third-party systems 2070, may use to provide social-networking
34 services and functionality to users across the Internet.

1 **[129]** In particular embodiments, a third-party system 2070 may include a third-
2 party content object provider (e.g., including sparse sample datasets described herein). A
3 third-party content object provider may include one or more sources of content objects, which
4 may be communicated to a client system 2030. As an example and not by way of limitation,
5 content objects may include information regarding things or activities of interest to the user,
6 such as, for example, movie show times, movie reviews, restaurant reviews, restaurant
7 menus, product information and reviews, or other suitable information. As another example
8 and not by way of limitation, content objects may include incentive content objects, such as
9 coupons, discount tickets, gift certificates, or other suitable incentive objects.

10 **[130]** In particular embodiments, social-networking system 2060 also includes user-
11 generated content objects, which may enhance a user's interactions with social-networking
12 system 2060. User-generated content may include anything a user can add, upload, send, or
13 "post" to social-networking system 2060. As an example and not by way of limitation, a user
14 communicates posts to social-networking system 2060 from a client system 2030. Posts may
15 include data such as status updates or other textual data, location information, photos, videos,
16 links, music or other similar data or media. Content may also be added to social-networking
17 system 2060 by a third-party through a "communication channel," such as a newsfeed or
18 stream.

19 **[131]** In particular embodiments, social-networking system 2060 may include a
20 variety of servers, sub-systems, programs, modules, logs, and data stores. In particular
21 embodiments, social-networking system 2060 may include one or more of the following: a
22 web server, action logger, API-request server, relevance-and-ranking engine, content-object
23 classifier, notification controller, action log, third-party-content-object-exposure log,
24 inference module, authorization/privacy server, search module, advertisement-targeting
25 module, user-interface module, user-profile store, connection store, third-party content store,
26 or location store. Social-networking system 2060 may also include suitable components such
27 as network interfaces, security mechanisms, load balancers, failover servers, management-
28 and-network-operations consoles, other suitable components, or any suitable combination
29 thereof. In particular embodiments, social-networking system 2060 may include one or more
30 user-profile stores for storing user profiles. A user profile may include, for example,
31 biographic information, demographic information, behavioral information, social
32 information, or other types of descriptive information, such as work experience, educational
33 history, hobbies or preferences, interests, affinities, or location. Interest information may
34 include interests related to one or more categories. Categories may be general or specific. As

1 an example and not by way of limitation, if a user “likes” an article about a brand of shoes the
2 category may be the brand, or the general category of “shoes” or “clothing.” A connection
3 store may be used for storing connection information about users. The connection
4 information may indicate users who have similar or common work experience, group
5 memberships, hobbies, educational history, or are in any way related or share common
6 attributes. The connection information may also include user-defined connections between
7 different users and content (both internal and external). A web server may be used for linking
8 social-networking system 2060 to one or more client systems 2030 or one or more third-party
9 system 2070 via network 2010. The web server may include a mail server or other messaging
10 functionality for receiving and routing messages between social-networking system 2060 and
11 one or more client systems 2030. An API-request server may allow a third-party system 2070
12 to access information from social-networking system 2060 by calling one or more APIs. An
13 action logger may be used to receive communications from a web server about a user’s
14 actions on or off social-networking system 2060. In conjunction with the action log, a third-
15 party-content-object log may be maintained of user exposures to third-party-content objects.
16 A notification controller may provide information regarding content objects to a client system
17 2030. Information may be pushed to a client system 2030 as notifications, or information may
18 be pulled from client system 2030 responsive to a request received from client system 2030.
19 Authorization servers may be used to enforce one or more privacy settings of the users of
20 social-networking system 2060. A privacy setting of a user determines how particular
21 information associated with a user can be shared. The authorization server may allow users to
22 opt in to or opt out of having their actions logged by social-networking system 2060 or shared
23 with other systems (e.g., third-party system 2070), such as, for example, by setting
24 appropriate privacy settings. Third-party-content-object stores may be used to store content
25 objects received from third parties, such as a third-party system 2070. Location stores may be
26 used for storing location information received from client systems 2030 associated with
27 users. Advertisement-pricing modules may combine social information, the current time,
28 location information, or other suitable information to provide relevant advertisements, in the
29 form of notifications, to a user.

30 [132] FIG. 21 illustrates an example computer system 2100. In particular
31 embodiments, one or more computer systems 2100 perform one or more steps of one or more
32 methods described or illustrated herein. In particular embodiments, one or more computer
33 systems 2100 provide functionality described or illustrated herein. In particular embodiments,
34 software running on one or more computer systems 2100 performs one or more steps of one

1 or more methods described or illustrated herein or provides functionality described or
2 illustrated herein. Particular embodiments include one or more portions of one or more
3 computer systems 2100. Herein, reference to a computer system may encompass a computing
4 device, and vice versa, where appropriate. Moreover, reference to a computer system may
5 encompass one or more computer systems, where appropriate.

6 [133] This disclosure contemplates any suitable number of computer systems 2100.
7 This disclosure contemplates computer system 2100 taking any suitable physical form. As
8 example and not by way of limitation, computer system 2100 may be an embedded computer
9 system, a system-on-chip (SOC), a single-board computer system (SBC) (such as, for
10 example, a computer-on-module (COM) or system-on-module (SOM)), a desktop computer
11 system, a laptop or notebook computer system, an interactive kiosk, a mainframe, a mesh of
12 computer systems, a mobile telephone, a personal digital assistant (PDA), a server, a tablet
13 computer system, an augmented/virtual reality device, or a combination of two or more of
14 these. Where appropriate, computer system 2100 may include one or more computer systems
15 2100; be unitary or distributed; span multiple locations; span multiple machines; span
16 multiple data centers; or reside in a cloud, which may include one or more cloud components
17 in one or more networks. Where appropriate, one or more computer systems 2100 may
18 perform without substantial spatial or temporal limitation one or more steps of one or more
19 methods described or illustrated herein. As an example and not by way of limitation, one or
20 more computer systems 2100 may perform in real time or in batch mode one or more steps of
21 one or more methods described or illustrated herein. One or more computer systems 2100
22 may perform at different times or at different locations one or more steps of one or more
23 methods described or illustrated herein, where appropriate.

24 [134] In particular embodiments, computer system 2100 includes a processor 2102,
25 memory 2104, storage 2106, an input/output (I/O) interface 2108, a communication interface
26 2110, and a bus 2112. Although this disclosure describes and illustrates a particular computer
27 system having a particular number of particular components in a particular arrangement, this
28 disclosure contemplates any suitable computer system having any suitable number of any
29 suitable components in any suitable arrangement.

30 [135] In particular embodiments, processor 2102 includes hardware for executing
31 instructions, such as those making up a computer program. As an example and not by way of
32 limitation, to execute instructions, processor 2102 may retrieve (or fetch) the instructions
33 from an internal register, an internal cache, memory 2104, or storage 2106; decode and
34 execute them; and then write one or more results to an internal register, an internal cache,

1 memory 2104, or storage 2106. In particular embodiments, processor 2102 may include one
2 or more internal caches for data, instructions, or addresses. This disclosure contemplates
3 processor 2102 including any suitable number of any suitable internal caches, where
4 appropriate. As an example and not by way of limitation, processor 2102 may include one or
5 more instruction caches, one or more data caches, and one or more translation lookaside
6 buffers (TLBs). Instructions in the instruction caches may be copies of instructions in
7 memory 2104 or storage 2106, and the instruction caches may speed up retrieval of those
8 instructions by processor 2102. Data in the data caches may be copies of data in memory
9 2104 or storage 2106 for instructions executing at processor 2102 to operate on; the results of
10 previous instructions executed at processor 2102 for access by subsequent instructions
11 executing at processor 2102 or for writing to memory 2104 or storage 2106; or other suitable
12 data. The data caches may speed up read or write operations by processor 2102. The TLBs
13 may speed up virtual-address translation for processor 2102. In particular embodiments,
14 processor 2102 may include one or more internal registers for data, instructions, or addresses.
15 This disclosure contemplates processor 2102 including any suitable number of any suitable
16 internal registers, where appropriate. Where appropriate, processor 2102 may include one or
17 more arithmetic logic units (ALUs); be a multi-core processor; or include one or more
18 processors 2102. Although this disclosure describes and illustrates a particular processor, this
19 disclosure contemplates any suitable processor.

20 [136] In particular embodiments, memory 2104 includes main memory for storing
21 instructions for processor 2102 to execute or data for processor 2102 to operate on. As an
22 example and not by way of limitation, computer system 2100 may load instructions from
23 storage 2106 or another source (such as, for example, another computer system 2100) to
24 memory 2104. Processor 2102 may then load the instructions from memory 2104 to an
25 internal register or internal cache. To execute the instructions, processor 2102 may retrieve
26 the instructions from the internal register or internal cache and decode them. During or after
27 execution of the instructions, processor 2102 may write one or more results (which may be
28 intermediate or final results) to the internal register or internal cache. Processor 2102 may
29 then write one or more of those results to memory 2104. In particular embodiments,
30 processor 2102 executes only instructions in one or more internal registers or internal caches
31 or in memory 2104 (as opposed to storage 2106 or elsewhere) and operates only on data in
32 one or more internal registers or internal caches or in memory 2104 (as opposed to storage
33 2106 or elsewhere). One or more memory buses (which may each include an address bus and
34 a data bus) may couple processor 2102 to memory 2104. Bus 2112 may include one or more

1 memory buses, as described below. In particular embodiments, one or more memory
2 management units (MMUs) reside between processor 2102 and memory 2104 and facilitate
3 accesses to memory 2104 requested by processor 2102. In particular embodiments, memory
4 2104 includes random access memory (RAM). This RAM may be volatile memory, where
5 appropriate. Where appropriate, this RAM may be dynamic RAM (DRAM) or static RAM
6 (SRAM). Moreover, where appropriate, this RAM may be single-ported or multi-ported
7 RAM. This disclosure contemplates any suitable RAM. Memory 2104 may include one or
8 more memories 2104, where appropriate. Although this disclosure describes and illustrates
9 particular memory, this disclosure contemplates any suitable memory.

10 [137] In particular embodiments, storage 2106 includes mass storage for data or
11 instructions. As an example and not by way of limitation, storage 2106 may include a hard
12 disk drive (HDD), a floppy disk drive, flash memory, an optical disc, a magneto-optical disc,
13 magnetic tape, or a Universal Serial Bus (USB) drive or a combination of two or more of
14 these. Storage 2106 may include removable or non-removable (or fixed) media, where
15 appropriate. Storage 2106 may be internal or external to computer system 2100, where
16 appropriate. In particular embodiments, storage 2106 is non-volatile, solid-state memory. In
17 particular embodiments, storage 2106 includes read-only memory (ROM). Where
18 appropriate, this ROM may be mask-programmed ROM, programmable ROM (PROM),
19 erasable PROM (EPROM), electrically erasable PROM (EEPROM), electrically alterable
20 ROM (EAROM), or flash memory or a combination of two or more of these. This disclosure
21 contemplates mass storage 2106 taking any suitable physical form. Storage 2106 may include
22 one or more storage control units facilitating communication between processor 2102 and
23 storage 2106, where appropriate. Where appropriate, storage 2106 may include one or more
24 storages 2106. Although this disclosure describes and illustrates particular storage, this
25 disclosure contemplates any suitable storage.

26 [138] In particular embodiments, I/O interface 2108 includes hardware, software, or
27 both, providing one or more interfaces for communication between computer system 2100
28 and one or more I/O devices. Computer system 2100 may include one or more of these I/O
29 devices, where appropriate. One or more of these I/O devices may enable communication
30 between a person and computer system 2100. As an example and not by way of limitation, an
31 I/O device may include a keyboard, keypad, microphone, monitor, mouse, printer, scanner,
32 speaker, still camera, stylus, tablet, touch screen, trackball, video camera, another suitable I/O
33 device or a combination of two or more of these. An I/O device may include one or more
34 sensors. This disclosure contemplates any suitable I/O devices and any suitable I/O interfaces

1 2108 for them. Where appropriate, I/O interface 2108 may include one or more device or
2 software drivers enabling processor 2102 to drive one or more of these I/O devices. I/O
3 interface 2108 may include one or more I/O interfaces 2108, where appropriate. Although
4 this disclosure describes and illustrates a particular I/O interface, this disclosure contemplates
5 any suitable I/O interface.

6 [139] In particular embodiments, communication interface 2110 includes hardware,
7 software, or both providing one or more interfaces for communication (such as, for example,
8 packet-based communication) between computer system 2100 and one or more other
9 computer systems 2100 or one or more networks. As an example and not by way of
10 limitation, communication interface 2110 may include a network interface controller (NIC) or
11 network adapter for communicating with an Ethernet or other wire-based network or a
12 wireless NIC (WNIC) or wireless adapter for communicating with a wireless network, such
13 as a WI-FI network. This disclosure contemplates any suitable network and any suitable
14 communication interface 2110 for it. As an example and not by way of limitation, computer
15 system 2100 may communicate with an ad hoc network, a personal area network (PAN), a
16 local area network (LAN), a wide area network (WAN), a metropolitan area network (MAN),
17 or one or more portions of the Internet or a combination of two or more of these. One or more
18 portions of one or more of these networks may be wired or wireless. As an example,
19 computer system 2100 may communicate with a wireless PAN (WPAN) (such as, for
20 example, a BLUETOOTH WPAN), a WI-FI network, a WI-MAX network, a cellular
21 telephone network (such as, for example, a Global System for Mobile Communications
22 (GSM) network), or other suitable wireless network or a combination of two or more of
23 these. Computer system 2100 may include any suitable communication interface 2110 for
24 any of these networks, where appropriate. Communication interface 2110 may include one or
25 more communication interfaces 2110, where appropriate. Although this disclosure describes
26 and illustrates a particular communication interface, this disclosure contemplates any suitable
27 communication interface.

28 [140] In particular embodiments, bus 2112 includes hardware, software, or both
29 coupling components of computer system 2100 to each other. As an example and not by way
30 of limitation, bus 2112 may include an Accelerated Graphics Port (AGP) or other graphics
31 bus, an Enhanced Industry Standard Architecture (EISA) bus, a front-side bus (FSB), a
32 HYPERTRANSPORT (HT) interconnect, an Industry Standard Architecture (ISA) bus, an
33 INFINIBAND interconnect, a low-pin-count (LPC) bus, a memory bus, a Micro Channel
34 Architecture (MCA) bus, a Peripheral Component Interconnect (PCI) bus, a PCI-Express

1 (PCIe) bus, a serial advanced technology attachment (SATA) bus, a Video Electronics
2 Standards Association local (VLB) bus, or another suitable bus or a combination of two or
3 more of these. Bus 2112 may include one or more buses 2112, where appropriate. Although
4 this disclosure describes and illustrates a particular bus, this disclosure contemplates any
5 suitable bus or interconnect.

6 [141] Herein, a computer-readable non-transitory storage medium or media may
7 include one or more semiconductor-based or other integrated circuits (ICs) (such, as for
8 example, field-programmable gate arrays (FPGAs) or application-specific ICs (ASICs)), hard
9 disk drives (HDDs), hybrid hard drives (HHDs), optical discs, optical disc drives (ODDs),
10 magneto-optical discs, magneto-optical drives, floppy diskettes, floppy disk drives (FDDs),
11 magnetic tapes, solid-state drives (SSDs), RAM-drives, SECURE DIGITAL cards or drives,
12 any other suitable computer-readable non-transitory storage media, or any suitable
13 combination of two or more of these, where appropriate. A computer-readable non-transitory
14 storage medium may be volatile, non-volatile, or a combination of volatile and non-volatile,
15 where appropriate.

16 [142] Herein, “or” is inclusive and not exclusive, unless expressly indicated
17 otherwise or indicated otherwise by context. Therefore, herein, “A or B” means “A, B, or
18 both,” unless expressly indicated otherwise or indicated otherwise by context. Moreover,
19 “and” is both joint and several, unless expressly indicated otherwise or indicated otherwise by
20 context. Therefore, herein, “A and B” means “A and B, jointly or severally,” unless expressly
21 indicated otherwise or indicated otherwise by context.

22 [143] The scope of this disclosure encompasses all changes, substitutions, variations,
23 alterations, and modifications to the example embodiments described or illustrated herein that
24 a person having ordinary skill in the art would comprehend. The scope of this disclosure is
25 not limited to the example embodiments described or illustrated herein. Moreover, although
26 this disclosure describes and illustrates respective embodiments herein as including particular
27 components, elements, feature, functions, operations, or steps, any of these embodiments may
28 include any combination or permutation of any of the components, elements, features,
29 functions, operations, or steps described or illustrated anywhere herein that a person having
30 ordinary skill in the art would comprehend. Furthermore, reference in the appended claims to
31 an apparatus or system or a component of an apparatus or system being adapted to, arranged
32 to, capable of, configured to, enabled to, operable to, or operative to perform a particular
33 function encompasses that apparatus, system, component, whether or not it or that particular
34 function is activated, turned on, or unlocked, as long as that apparatus, system, or component

1 is so adapted, arranged, capable, configured, enabled, operable, or operative. Additionally,
2 although this disclosure describes or illustrates particular embodiments as providing
3 particular advantages, particular embodiments may provide none, some, or all of these
4 advantages.

5

CLAIMS

1. A method comprising, by a computing system:
 - receiving instructions to render an image comprising content defined by at least a two-dimensional (2D) primitive;
 - determining a portion of the 2D primitive covering a tile of a plurality of tiles of the image;
 - generating an edge definition to represent an edge of the portion of the 2D primitive;
 - and
 - for each row of pixels within at least a portion of the tile containing the portion of the 2D primitive:
 - identifying, based on the edge definition, a left-most pixel and a right-most pixel in the row that intersect the edge;
 - identifying, based on the left-most pixel and the right-most pixel, a set of first pixels in the row intersecting the edge;
 - determining, for each first pixel in the set, a coverage weight indicating a proportion of the first pixel covered by the 2D primitive; and
 - determining color information for the set of first pixels based on the associated coverage weights.
2. The method of Claim 1, wherein the 2D primitive is a horizontally-aligned trapezoid or a quadratic curve.
3. The method of Claim 1 or Claim 2, further comprising:
 - determining a bounding box that encompasses the portion of the 2D primitive covering the tile;
 - wherein the at least the portion of the tile containing the portion of the 2D primitive is the bounding box.
4. The method of any preceding Claim, wherein generating the edge definition to represent the edge of the portion of the 2D primitive comprises:
 - determining a function equation representing the edge;
 - optionally, wherein the function equation is a linear equation.
 - or
 - optionally, wherein the function equation is a quadratic equation.
5. The method of any preceding Claim, further comprising, for each row of the pixels within the at least the portion of the tile containing the portion of the 2D primitive:
 - identifying, based on the left-most pixel and the right-most pixel, a set of second

1 pixels in the row and a set of third pixels in the row, wherein the set of second pixels
2 correspond to pixels that are fully uncovered by the 2D primitive and the set of third pixels
3 correspond to pixels that are fully covered by the 2D primitive;

4 optionally, further comprising, for each row of the pixels within the at least the
5 portion of the tile containing the portion of the 2D primitive:

6 determining, for each of the second pixels, a coverage weight that indicates that the
7 second pixel is fully uncovered by the 2D primitive;

8 determining, for each of the third pixels, a coverage weight that indicates that the third
9 pixel is fully uncovered by the 2D primitive; and

10 determining color information for the set of second pixels and the set of third pixels
11 based on the associated coverage weights.

12 6. One or more computer-readable non-transitory storage media including instructions
13 that, when executed by one or more processors, are configured to cause the one or more
14 processors to:

15 receive instructions to render an image comprising content defined by at least a two-
16 dimensional (2D) primitive;

17 determine a portion of the 2D primitive covering a tile of a plurality of tiles of the
18 image;

19 generate an edge definition to represent an edge of the portion of the 2D primitive;

20 and

21 for each row of pixels within at least a portion of the tile containing the portion of the
22 2D primitive:

23 identify, based on the edge definition, a left-most pixel and a right-most pixel
24 in the row that intersect the edge;

25 identify, based on the left-most pixel and the right-most pixel, a set of first
26 pixels in the row intersecting the edge;

27 determine, for each first pixel in the set, a coverage weight indicating a
28 proportion of the first pixel covered by the 2D primitive; and

29 determine color information for the set of first pixels based on the associated
30 coverage weights.

31 7. The one or more computer-readable non-transitory storage media of Claim 6,
32 wherein the 2D primitive is a horizontally-aligned trapezoid or a quadratic curve.

33 8. The one or more computer-readable non-transitory storage media of Claim 6 or
34 Claim 7, wherein the instructions are configured to further cause the one or more processors

1 to:

2 determine a bounding box that encompasses the portion of the 2D primitive covering
3 the tile;

4 wherein the at least the portion of the tile containing the portion of the 2D primitive is
5 the bounding box.

6 9. The one or more computer-readable non-transitory storage media of any one of
7 Claims 6 to 8, wherein generating the edge definition to represent the edge of the portion of
8 the 2D primitive comprises:

9 determining a function equation representing the edge, wherein the function equation
10 is a linear equation or a quadratic equation.

11 10. The one or more computer-readable non-transitory storage media of any one of
12 Claims 6 to 9, wherein the instructions are configured to further cause the one or more
13 processors to:

14 for each row of the pixels within the at least the portion of the tile containing the
15 portion of the 2D primitive:

16 identify, based on the left-most pixel and the right-most pixel, a set of second
17 pixels in the row and a set of third pixels in the row, wherein the set of second pixels
18 correspond to pixels that are fully uncovered by the 2D primitive and the set of third
19 pixels correspond to pixels that are fully covered by the 2D primitive;

20 optionally, wherein the instructions are configured to further cause the one or more
21 processors to:

22 for each row of the pixels within the at least the portion of the tile containing the
23 portion of the 2D primitive:

24 determine, for each of the second pixels, a coverage weight that indicates that
25 the second pixel is fully uncovered by the 2D primitive;

26 determine, for each of the third pixels, a coverage weight that indicates that
27 the third pixel is fully uncovered by the 2D primitive; and

28 determine color information for the set of second pixels and the set of third
29 pixels based on the associated coverage weights.

30 11. A system comprising: one or more processors; and one or more computer-
31 readable non-transitory storage media in communication with the one or more processors, the
32 one or more computer-readable non-transitory storage media comprising instructions that
33 when executed by the one or more processors, cause the system to:

34 receive instructions to render an image comprising content defined by at least a two-

1 dimensional (2D) primitive;

2 determine a portion of the 2D primitive covering a tile of a plurality of tiles of the
3 image;

4 generate an edge definition to represent an edge of the portion of the 2D primitive;
5 and

6 for each row of pixels within at least a portion of the tile containing the portion of the
7 2D primitive:

8 identify, based on the edge definition, a left-most pixel and a right-most pixel
9 in the row that intersect the edge;

10 identify, based on the left-most pixel and the right-most pixel, a set of first
11 pixels in the row intersecting the edge;

12 determine, for each first pixel in the set, a coverage weight indicating a
13 proportion of the first pixel covered by the 2D primitive; and

14 determine color information for the set of first pixels based on the associated
15 coverage weights.

16 12. The system of Claim 11, wherein the 2D primitive is a horizontally-aligned
17 trapezoid or a quadratic curve.

18 13. The system of Claim 11 or Claim 12, wherein the instructions are configured to
19 further cause the one or more processors to:

20 determine a bounding box that encompasses the portion of the 2D primitive covering
21 the tile;

22 wherein the at least the portion of the tile containing the portion of the 2D primitive is
23 the bounding box.

24 14. The system of any one of Claims 11 to 13, wherein generating the edge definition
25 to represent the edge of the portion of the 2D primitive comprises:

26 determining a function equation representing the edge, wherein the function equation
27 is a linear equation or a quadratic equation.

28 15. The system of any one of Claims 11 to 14, wherein the instructions are configured
29 to further cause the one or more processors to:

30 for each row of the pixels within the at least the portion of the tile containing the
31 portion of the 2D primitive:

32 identify, based on the left-most pixel and the right-most pixel, a set of second
33 pixels in the row and a set of third pixels in the row, wherein the set of second pixels
34 correspond to pixels that are fully uncovered by the 2D primitive and the set of third

1 pixels correspond to pixels that are fully covered by the 2D primitive;
2 optionally, wherein the instructions, when executed by the one or more processors,
3 further cause the system to:
4 for each row of the pixels within the at least the portion of the tile containing the
5 portion of the 2D primitive:
6 determine, for each of the second pixels, a coverage weight that indicates that
7 the second pixel is fully uncovered by the 2D primitive;
8 determine, for each of the third pixels, a coverage weight that indicates that
9 the third pixel is fully uncovered by the 2D primitive; and
10 determine color information for the set of second pixels and the set of third
11 pixels based on the associated coverage weights.

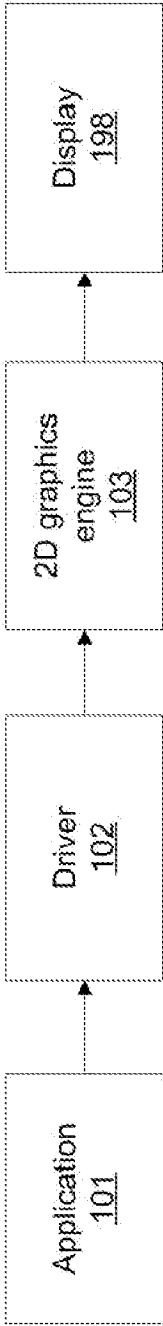


FIG. 1A

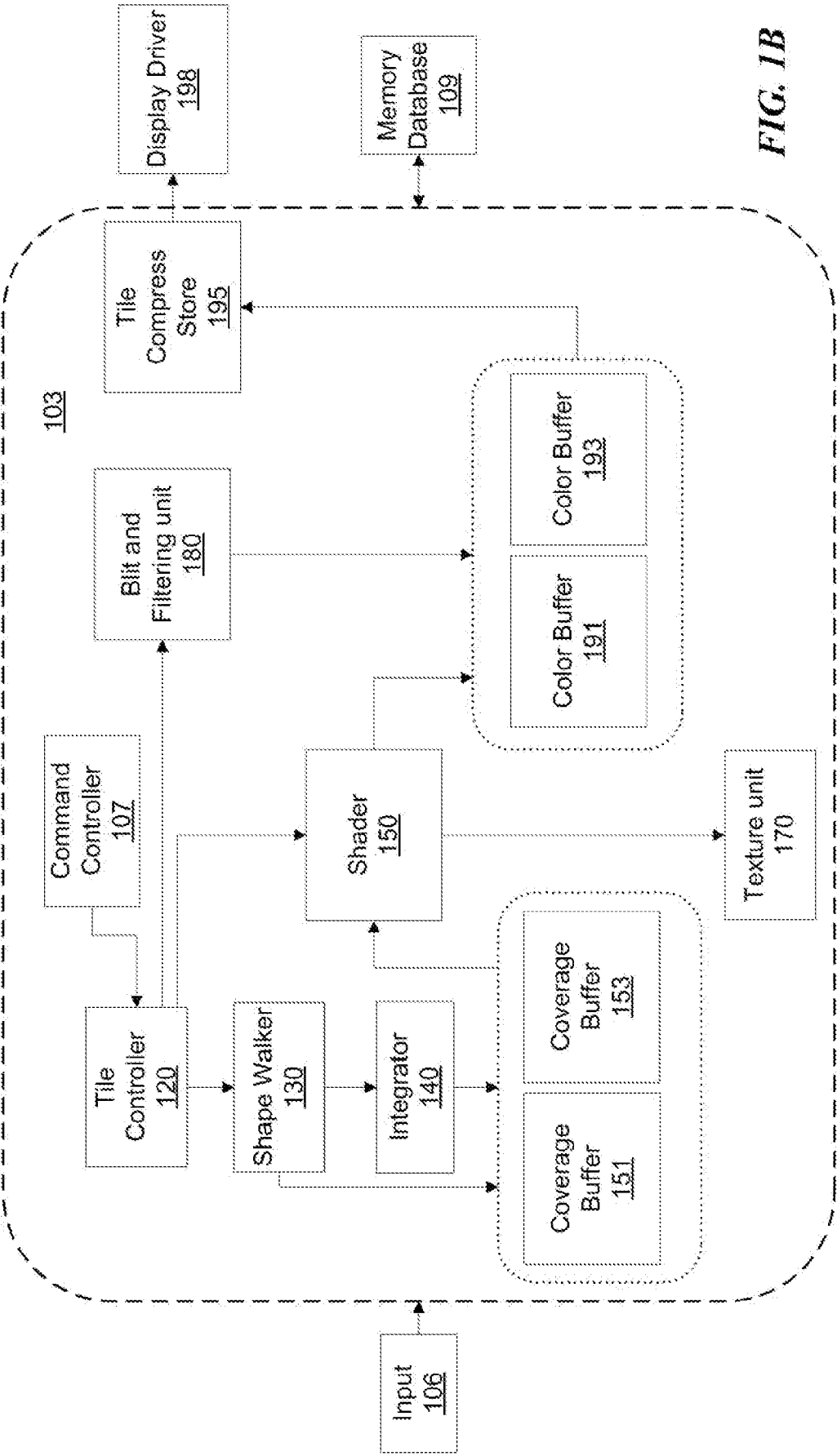


FIG. 1B

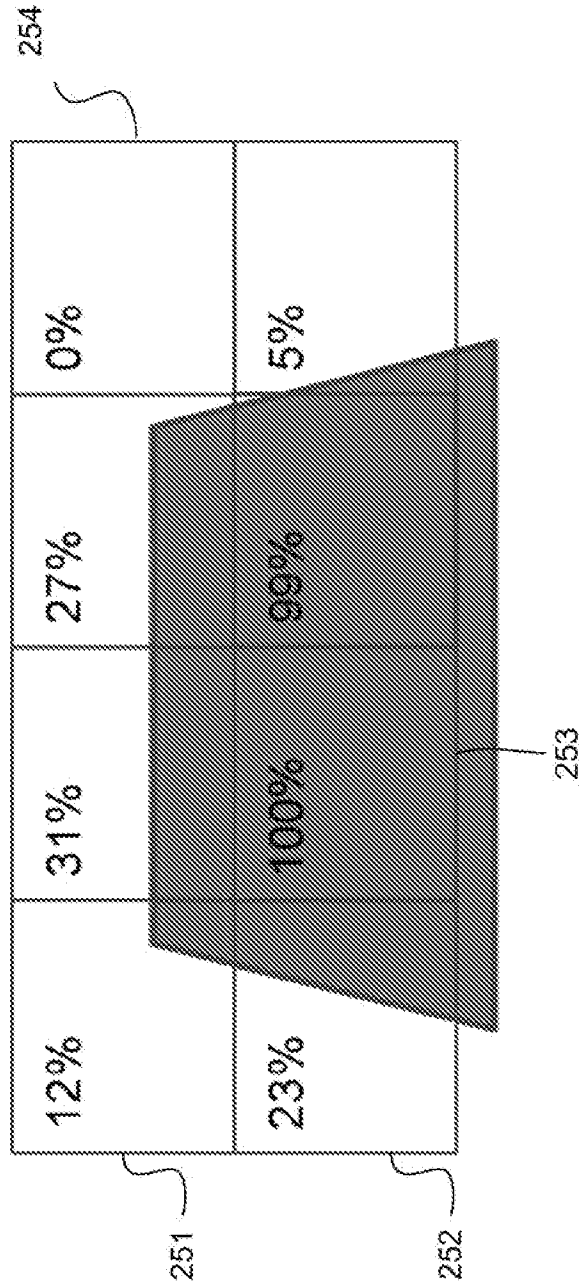
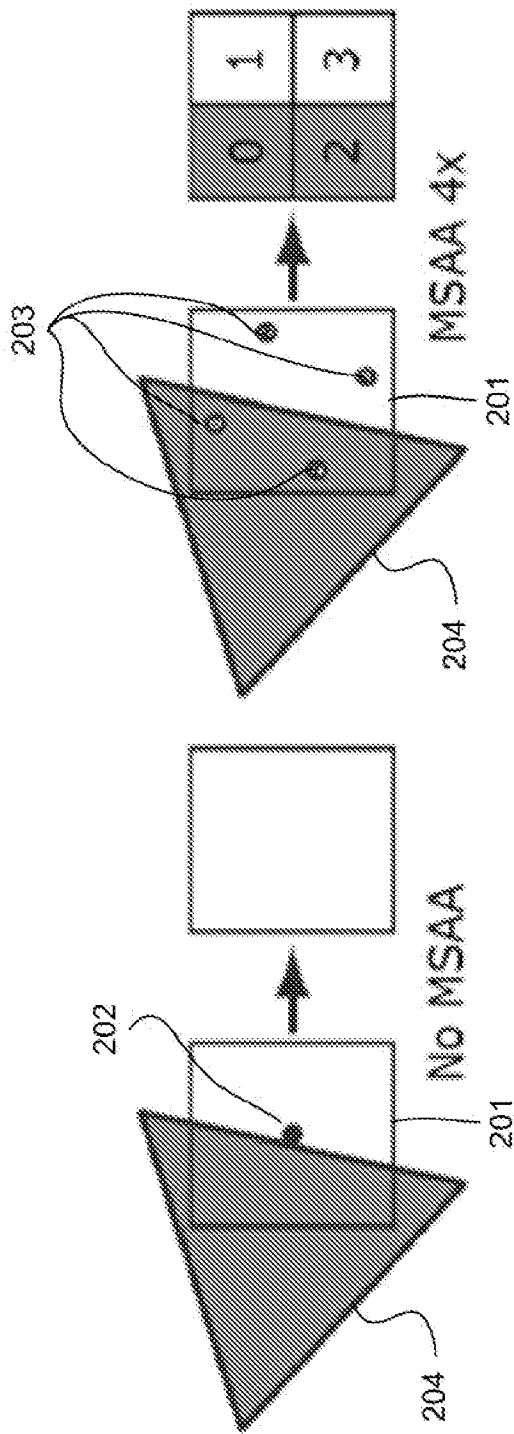


FIG. 2

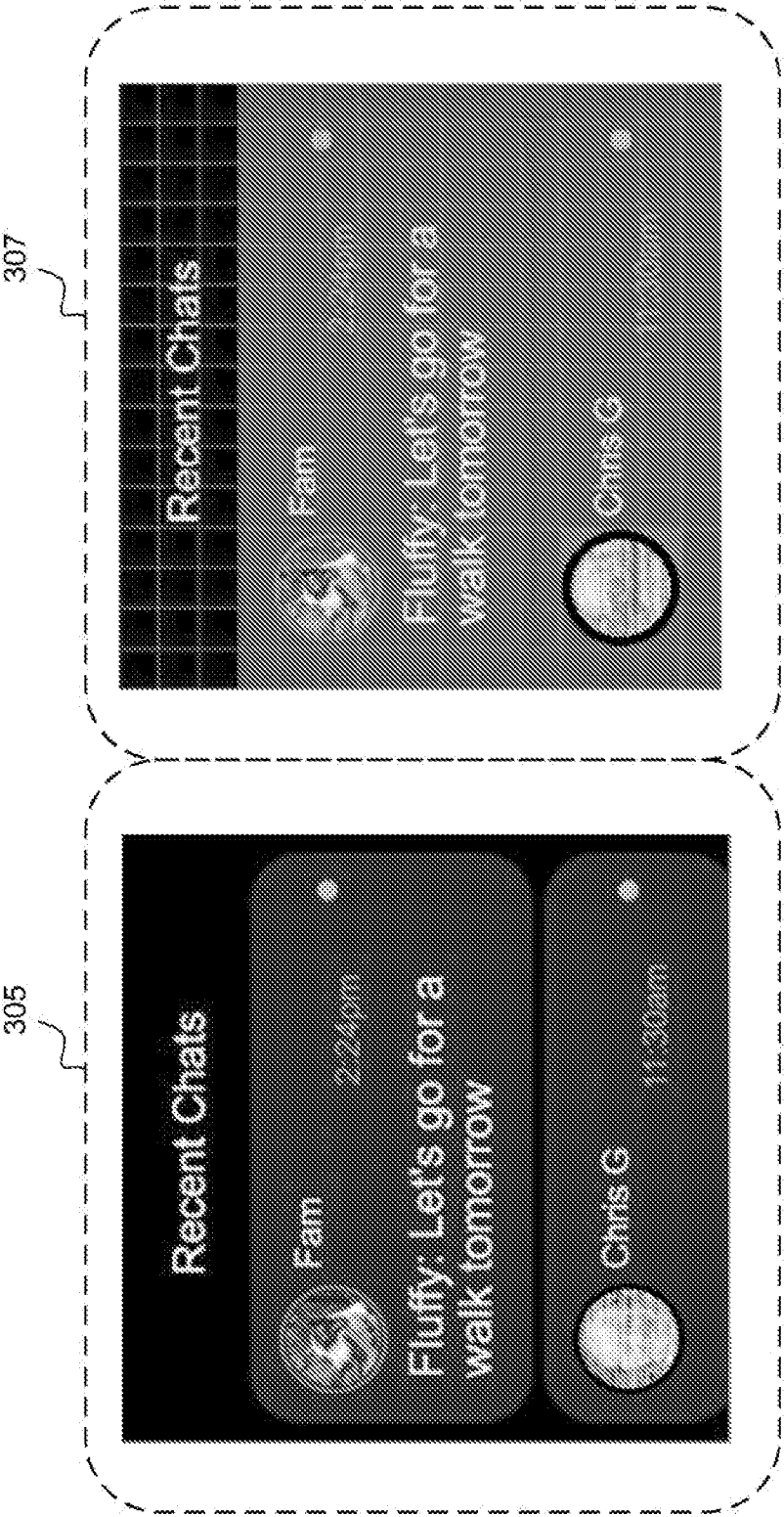


FIG. 3A

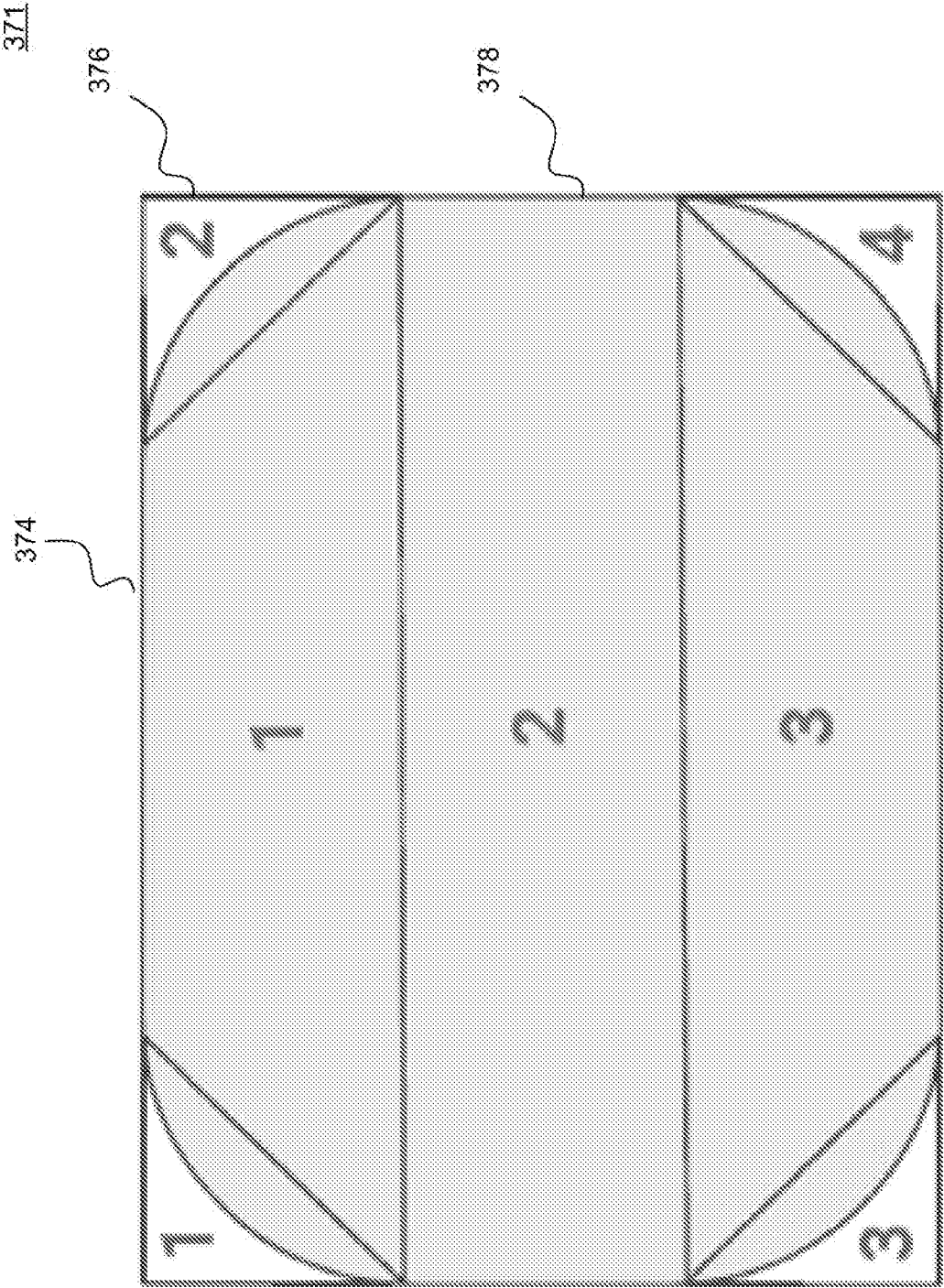
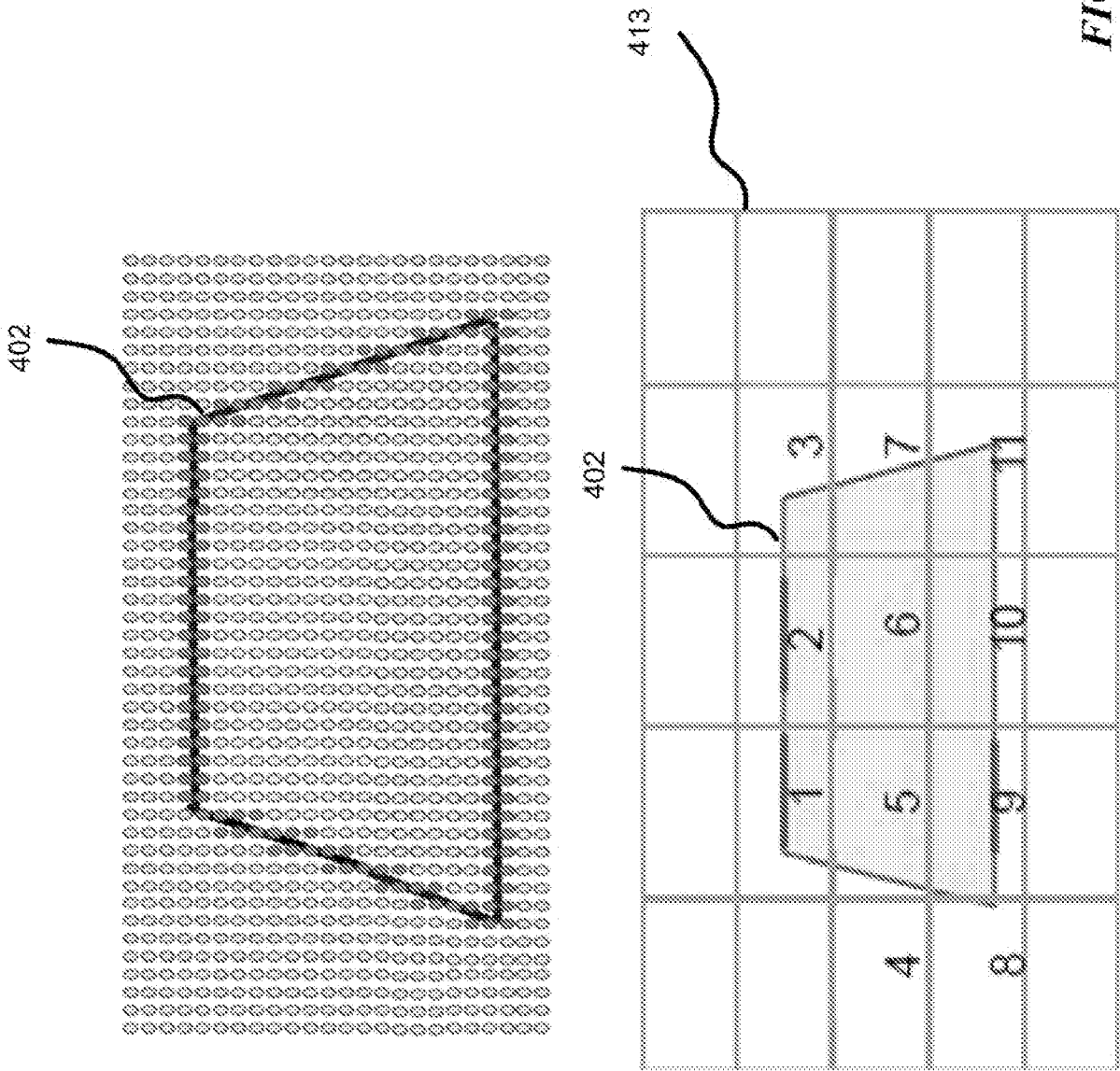


FIG. 3B



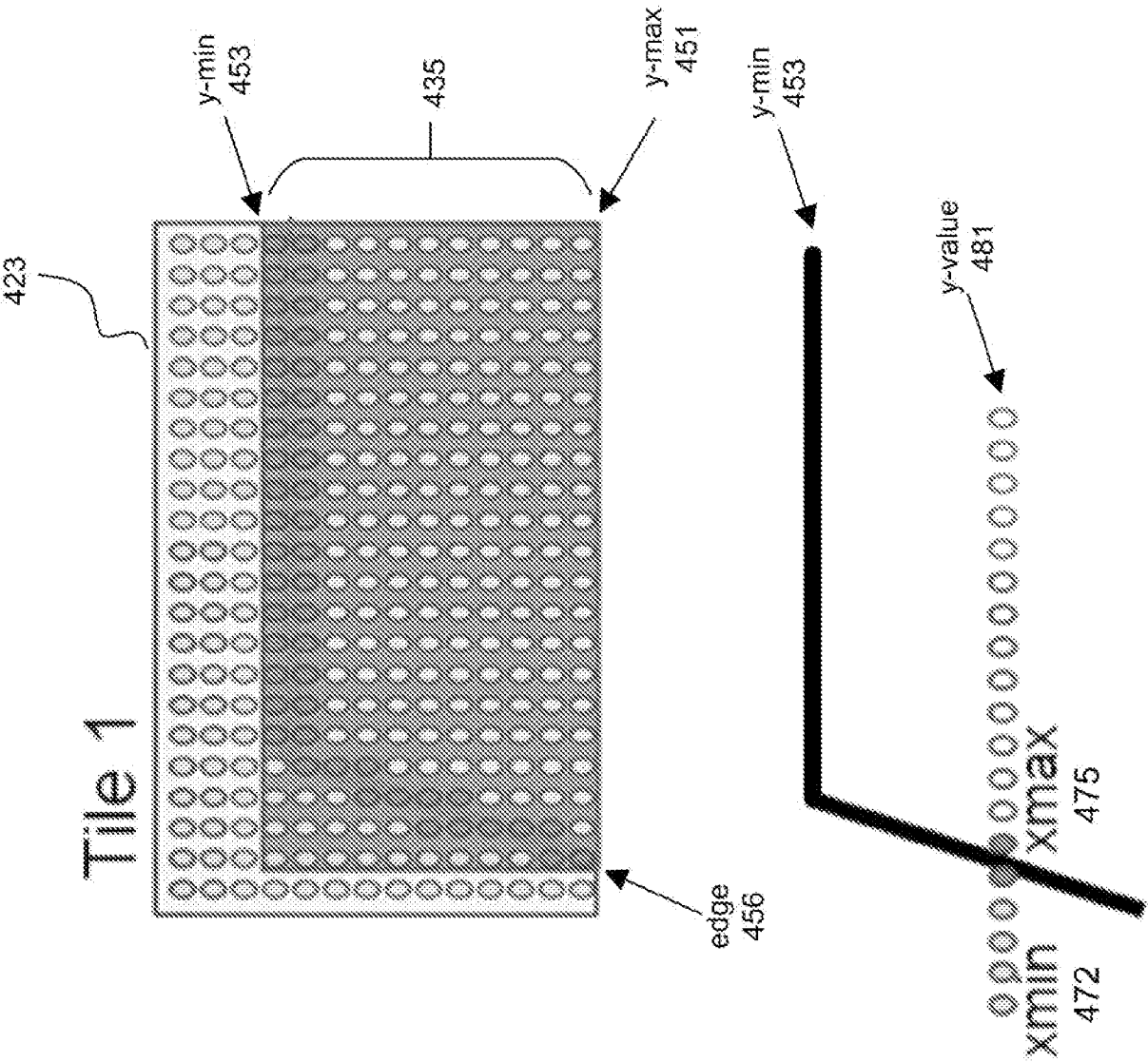


FIG. 4B

Frame

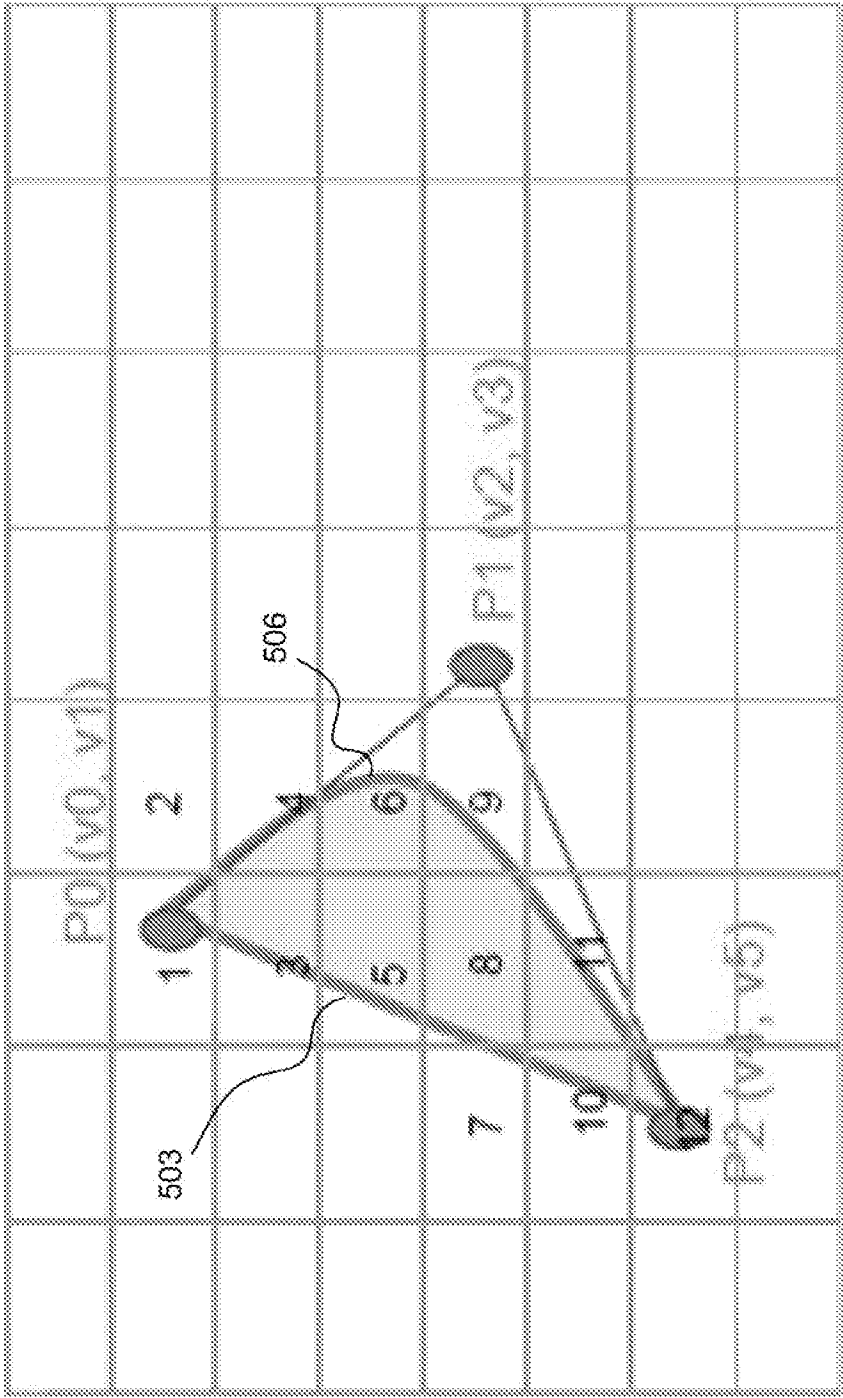


FIG. 5A

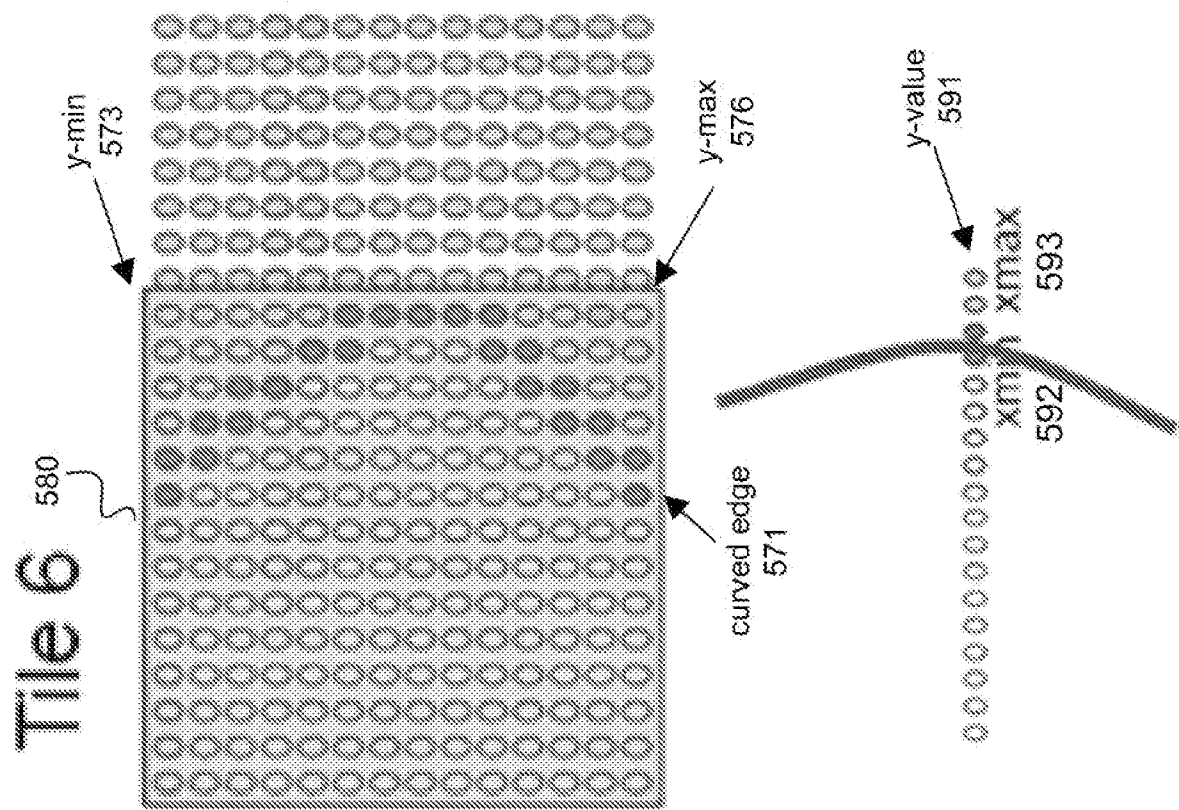


FIG. 5B

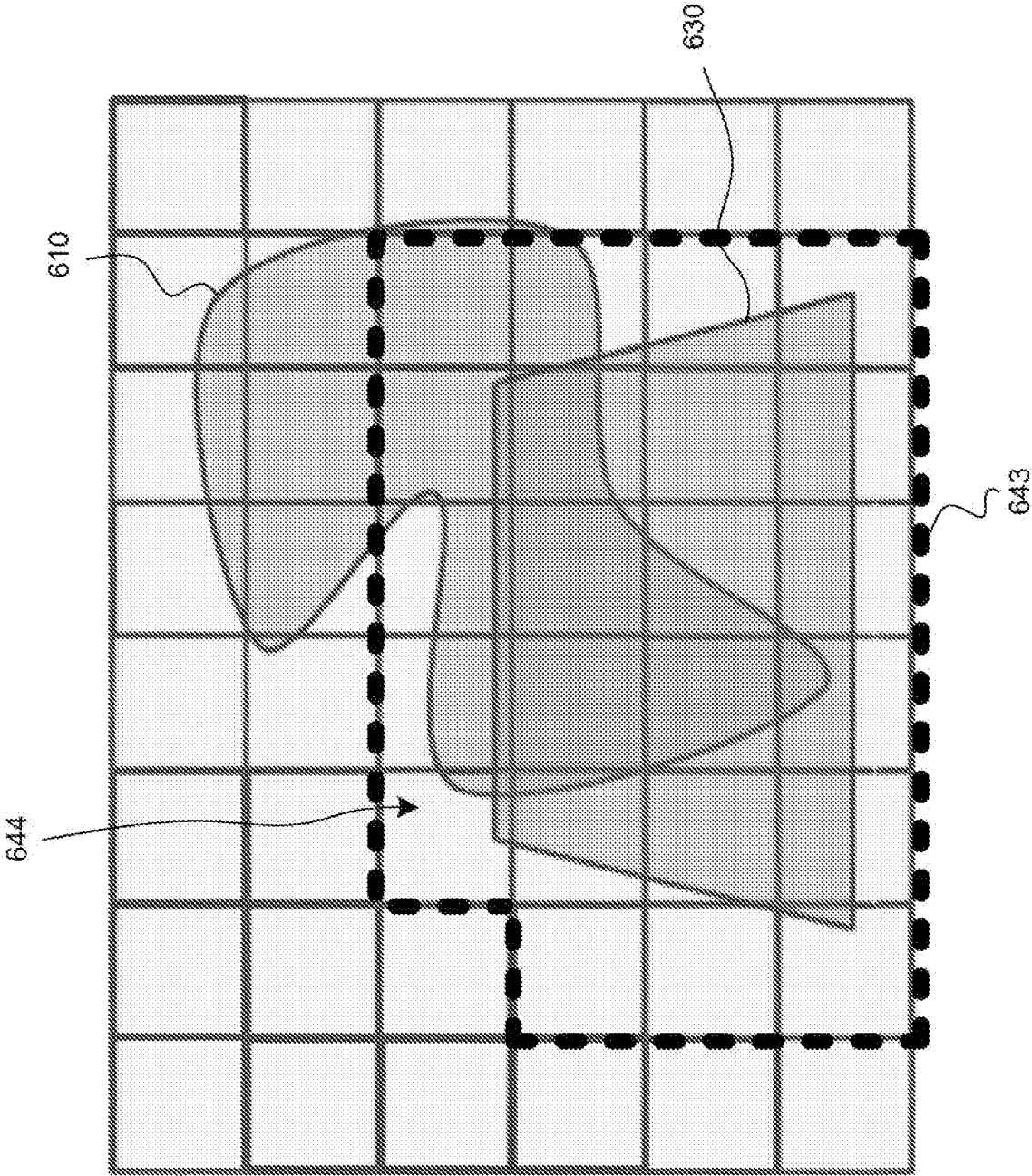


FIG. 6

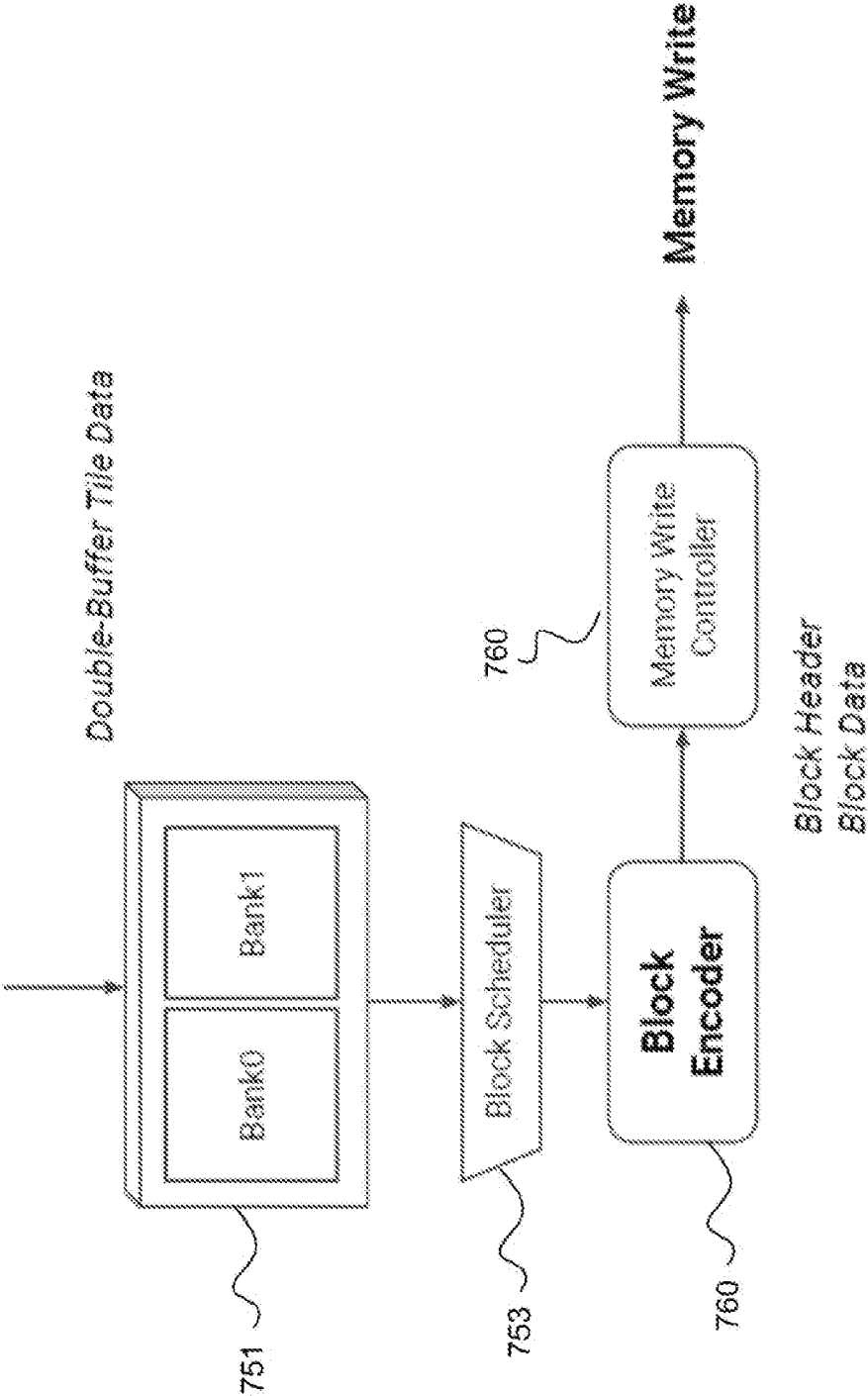


FIG. 7

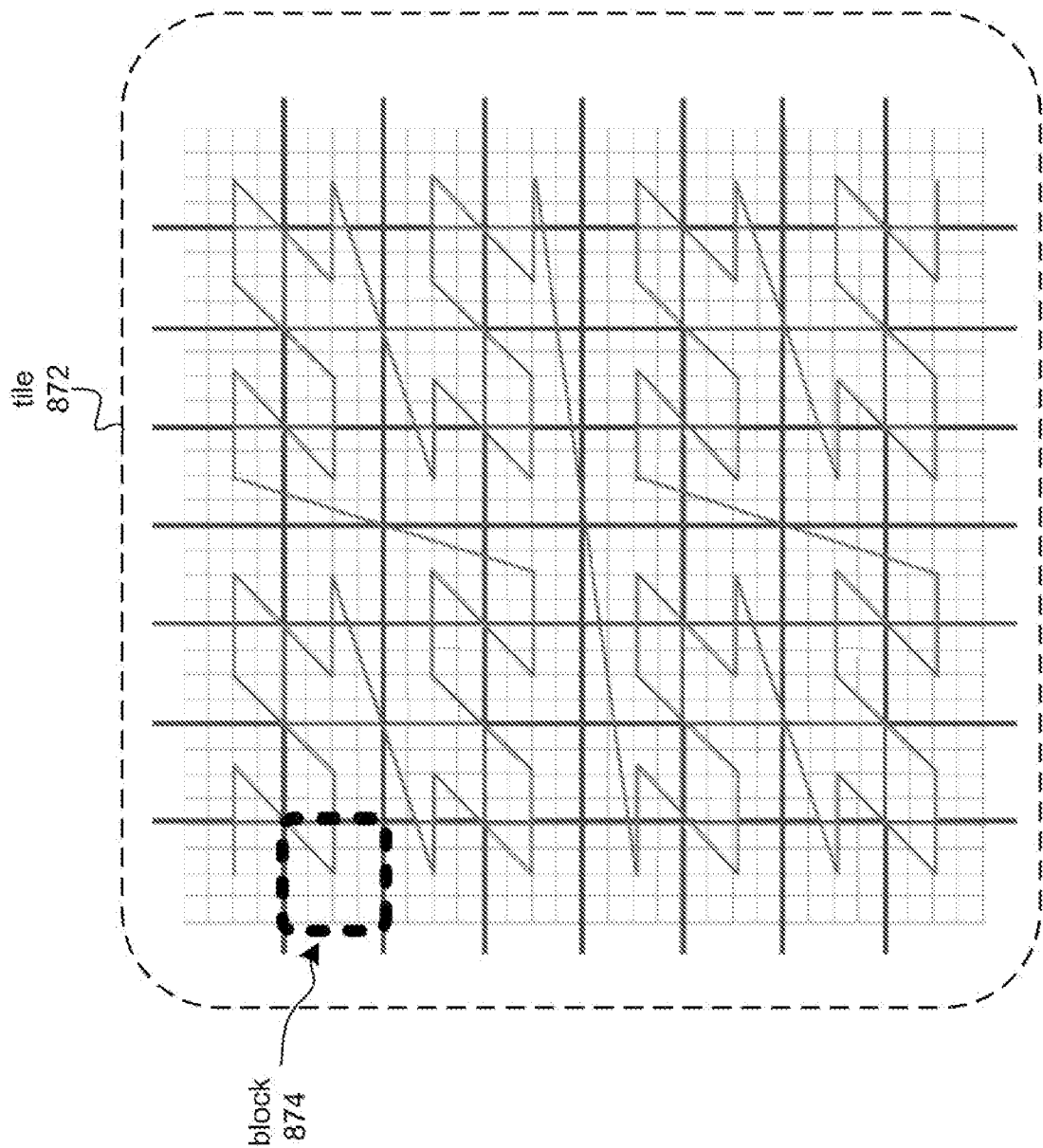


FIG. 8

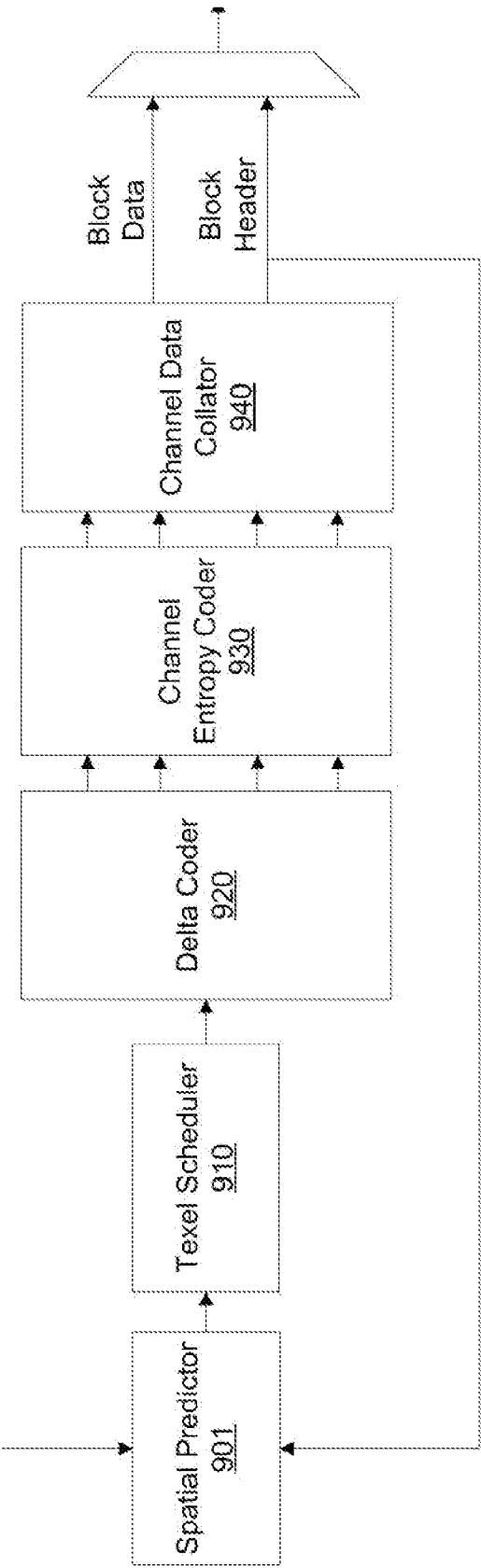


FIG. 9

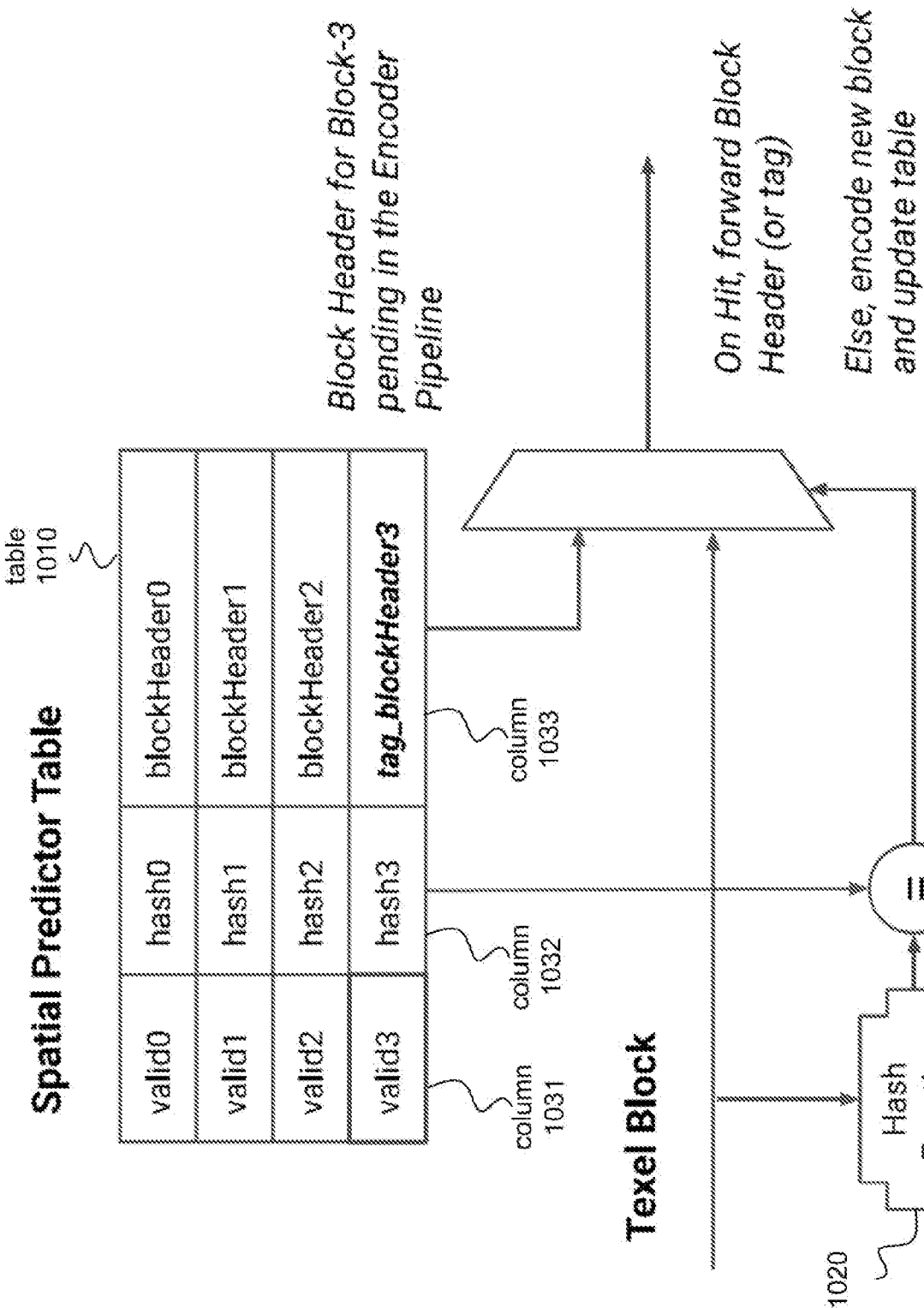


FIG. 10

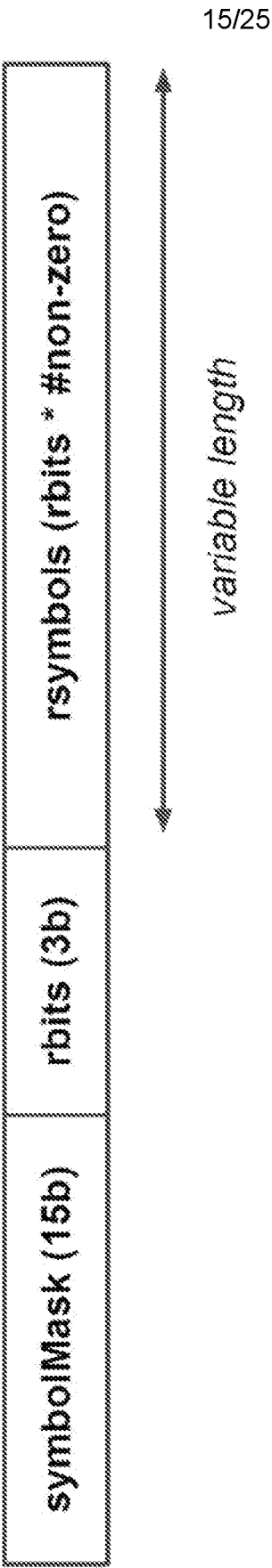


FIG. 11

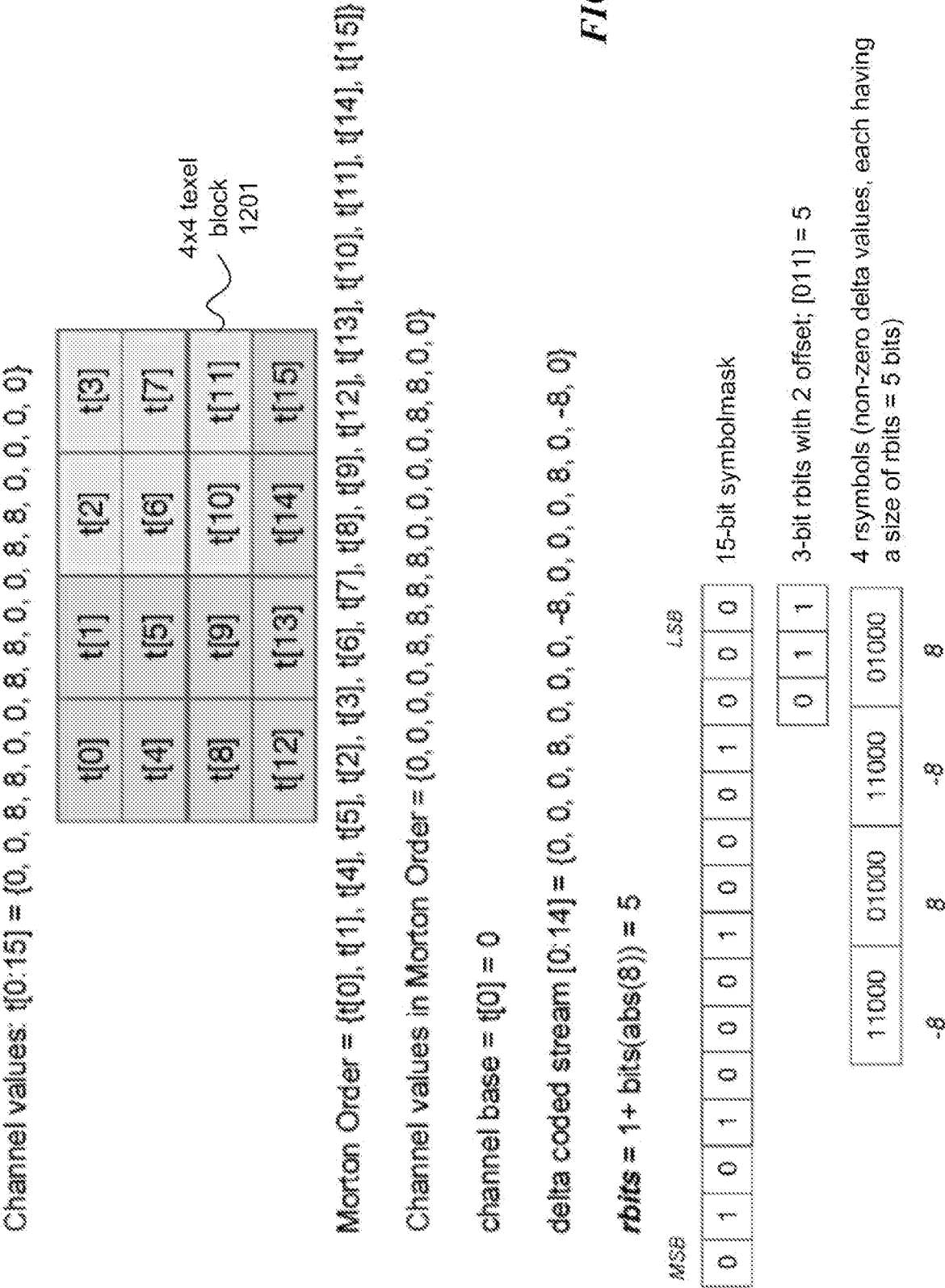


FIG. 12

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.

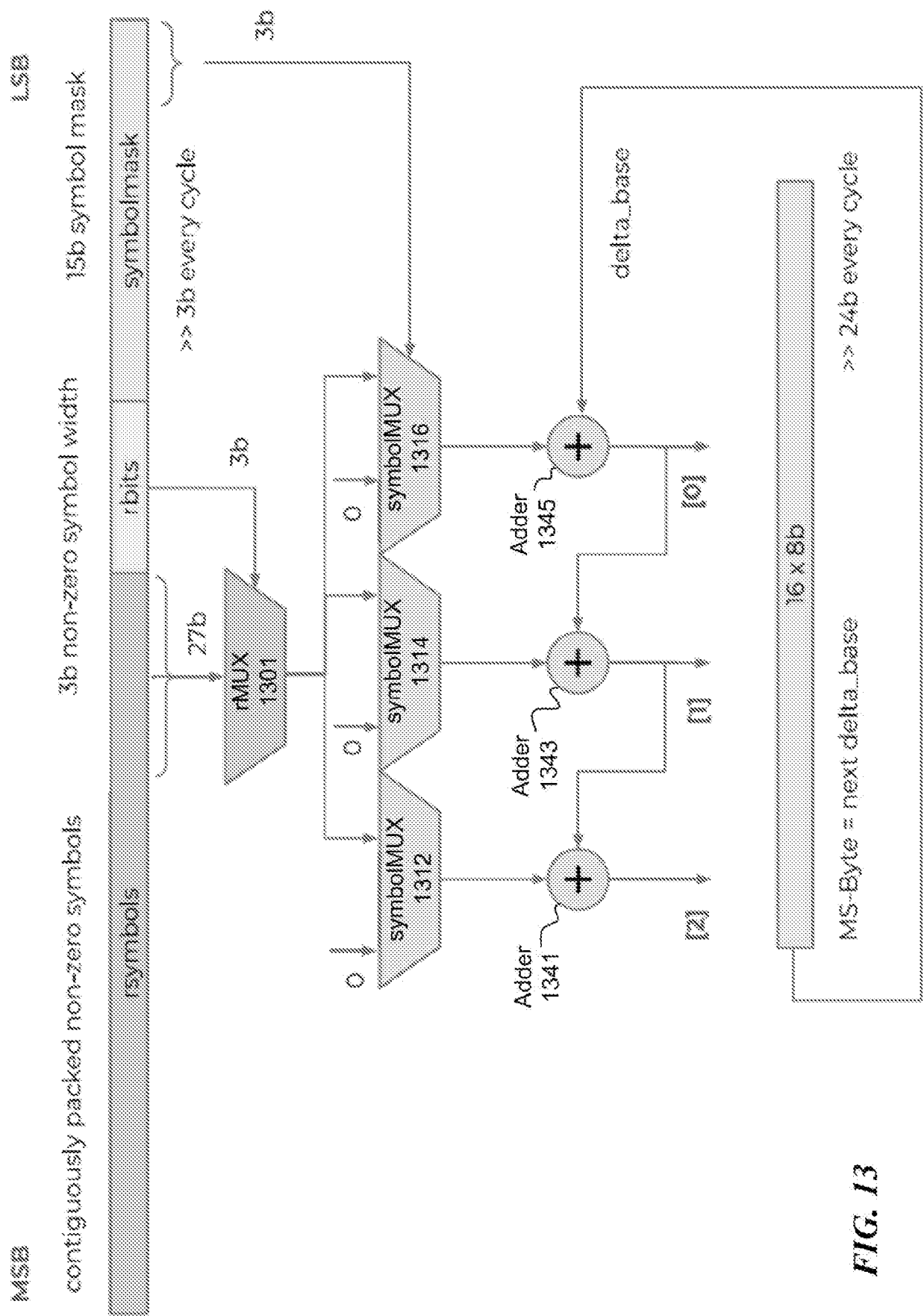
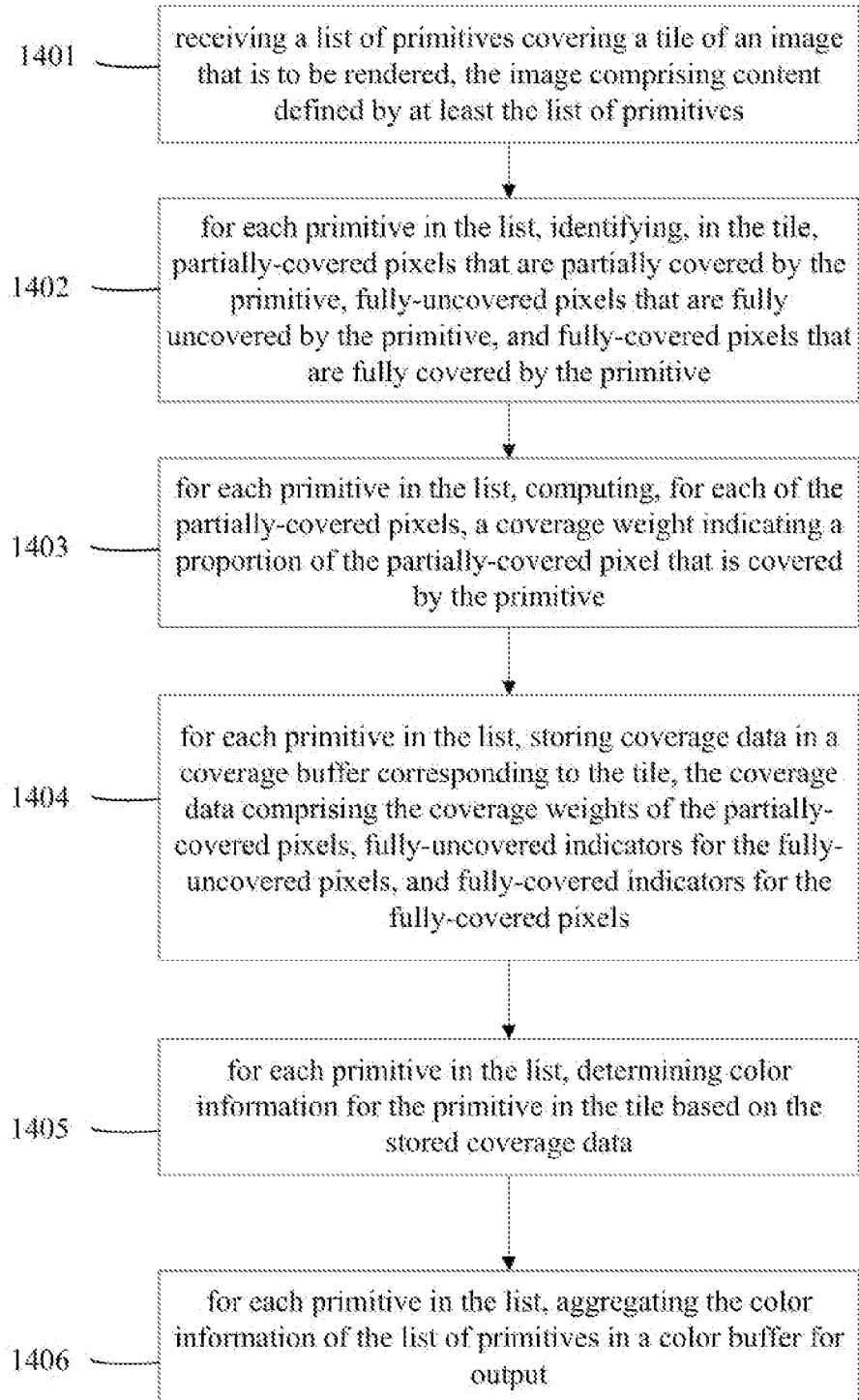
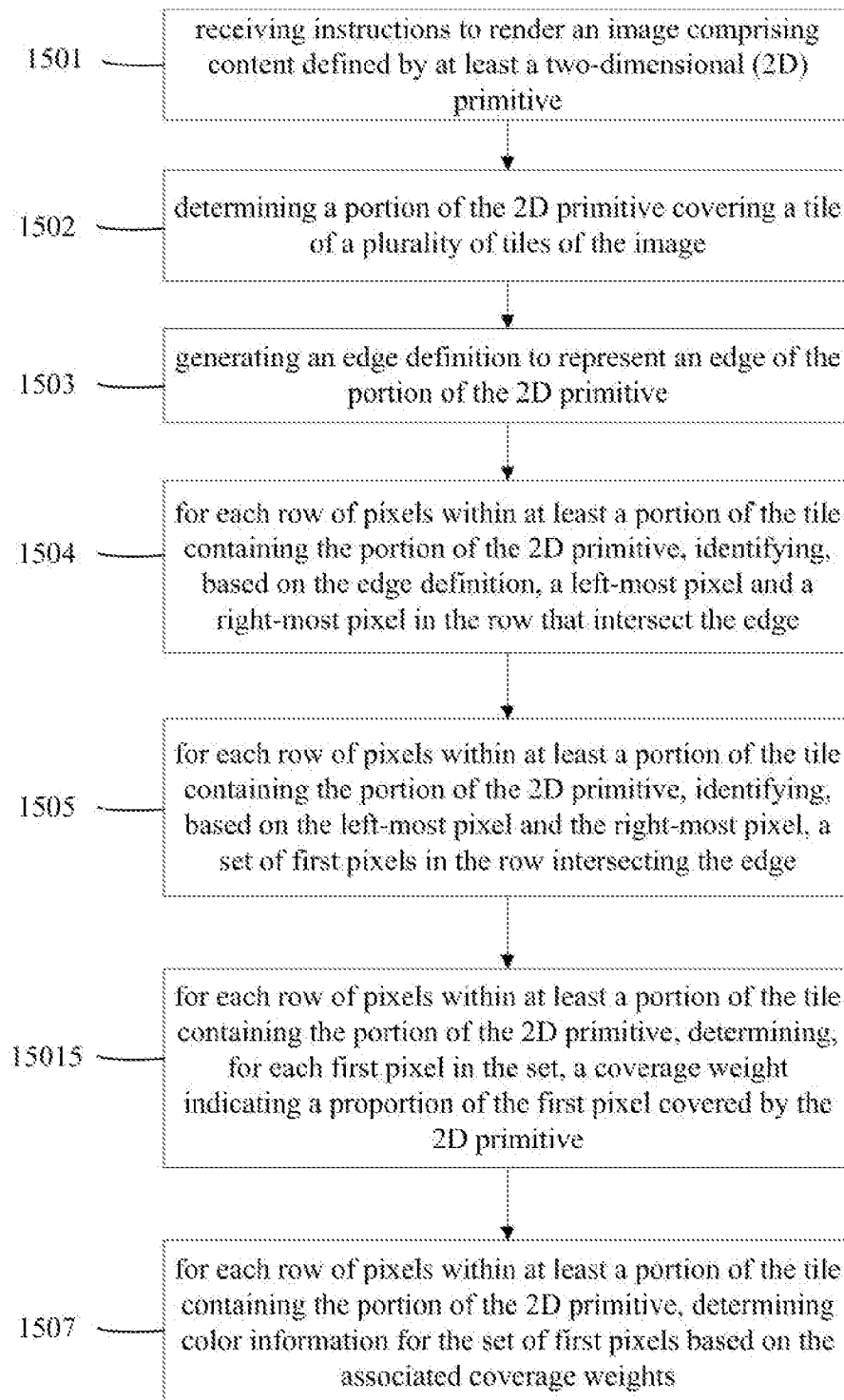


FIG. 13

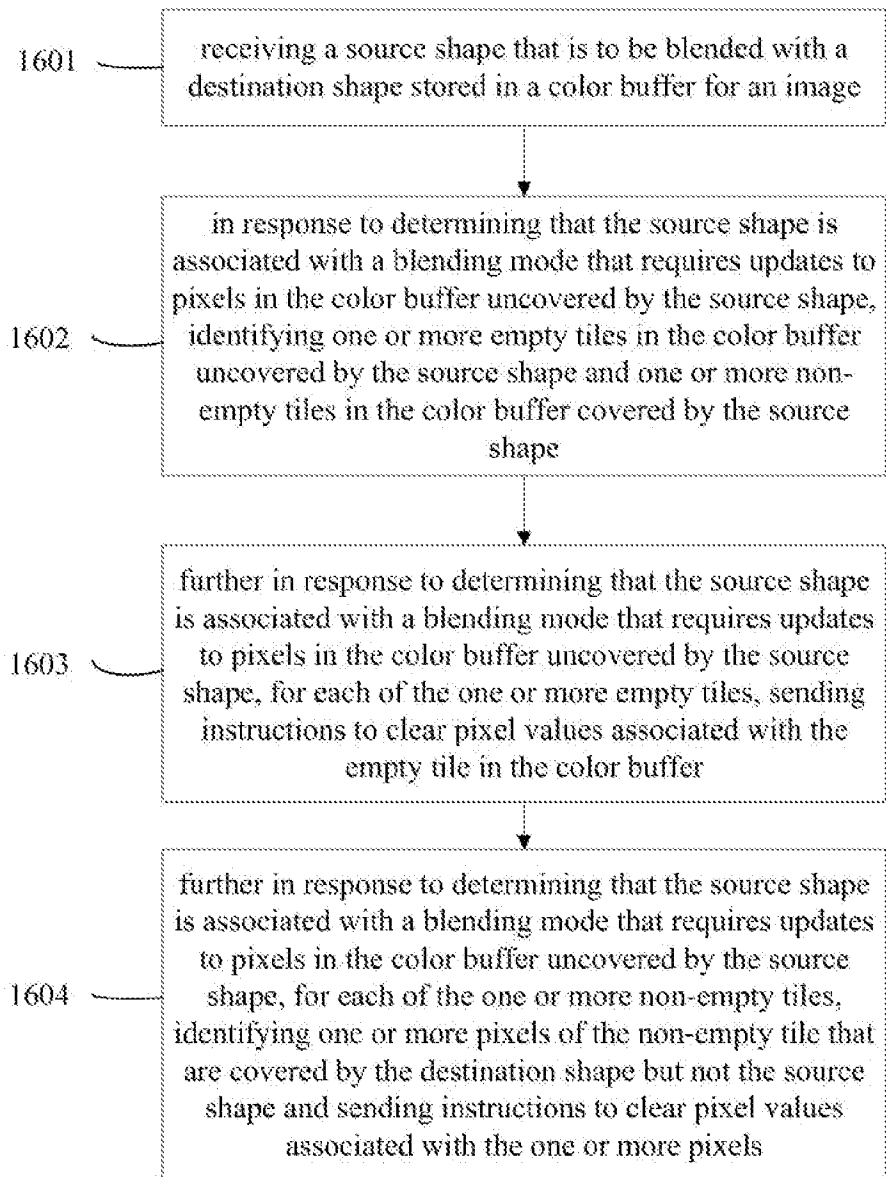
18/25

1400**FIG. 14**

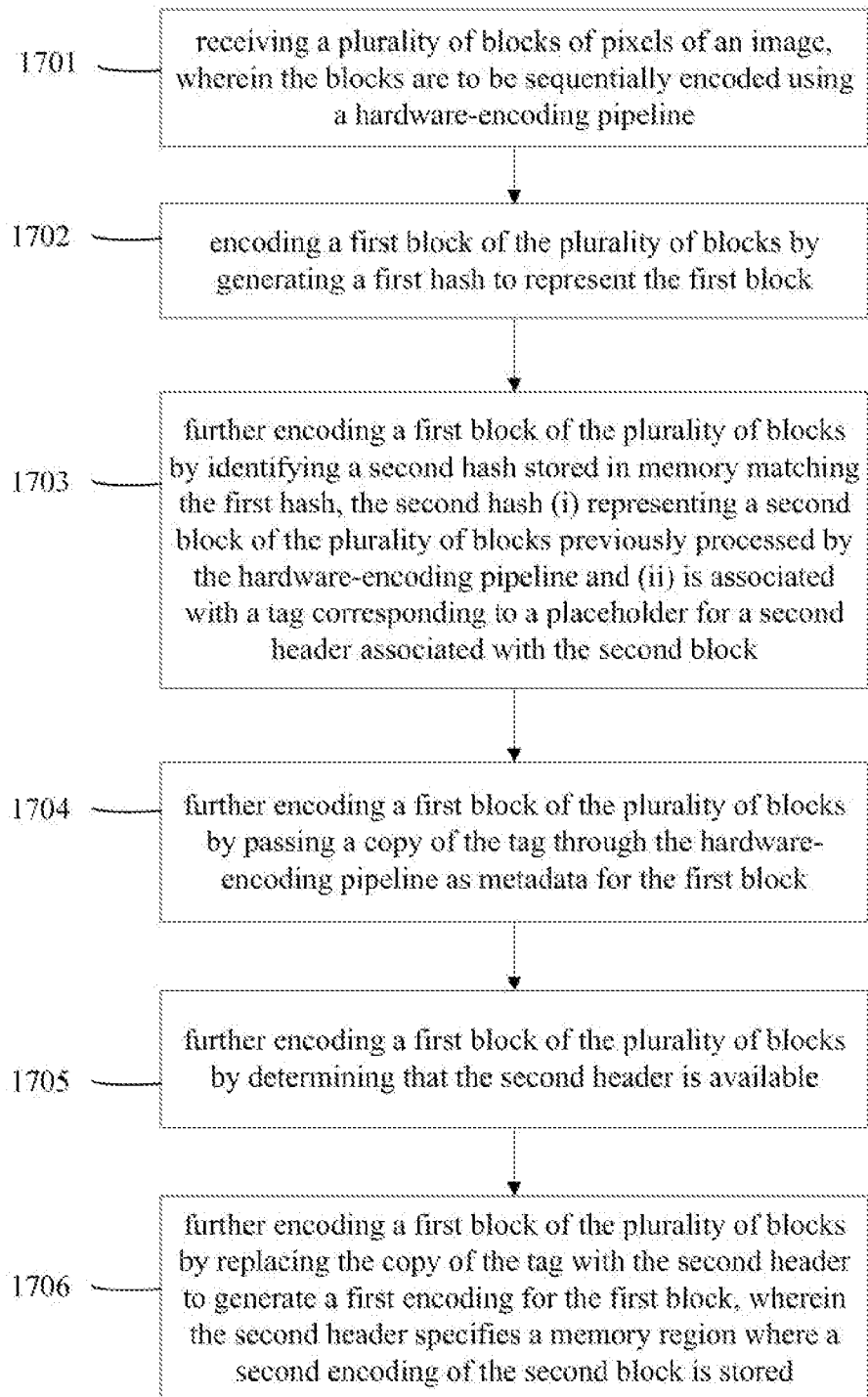
19/25

1500**FIG. 15**

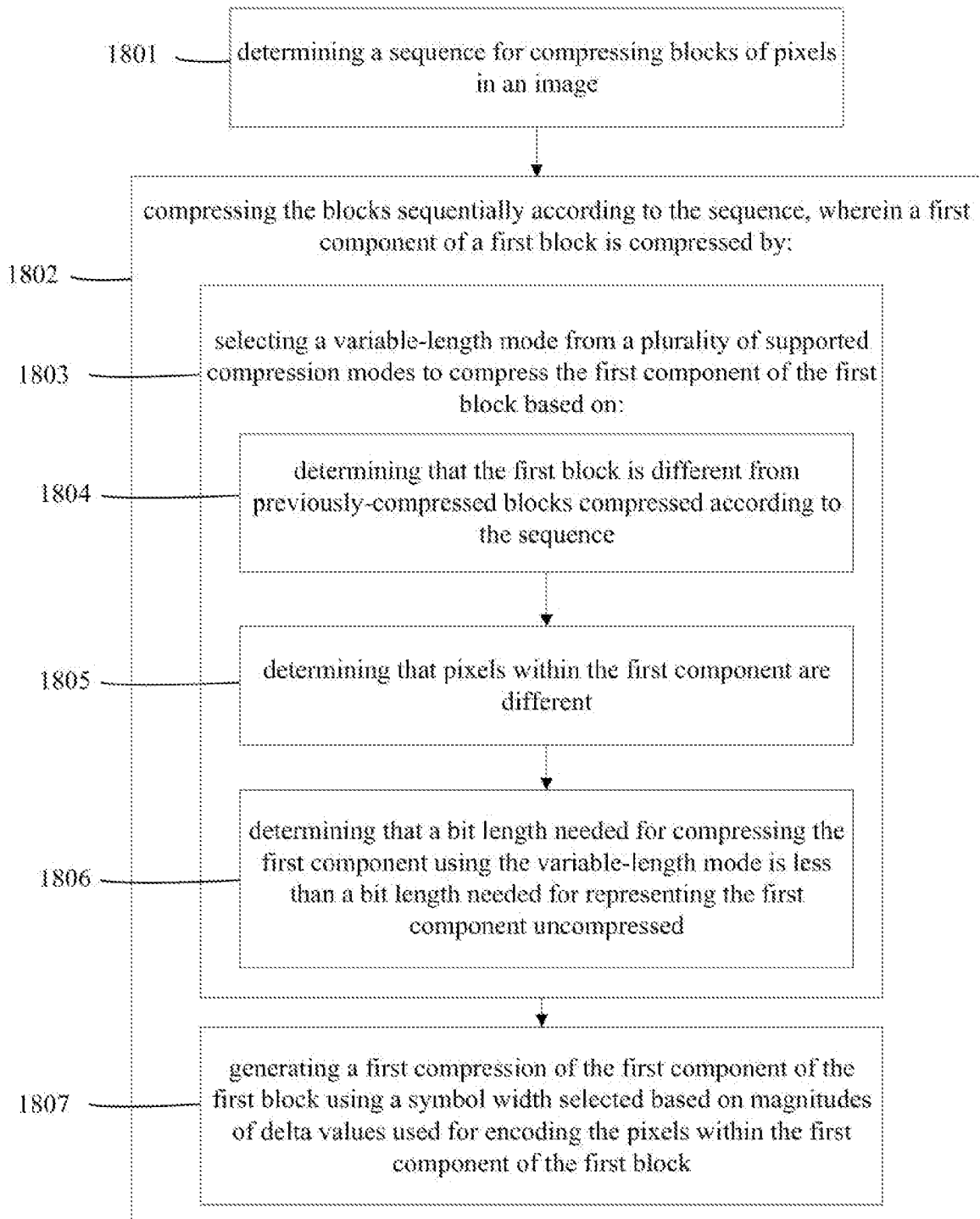
20/25

1600**FIG. 16**

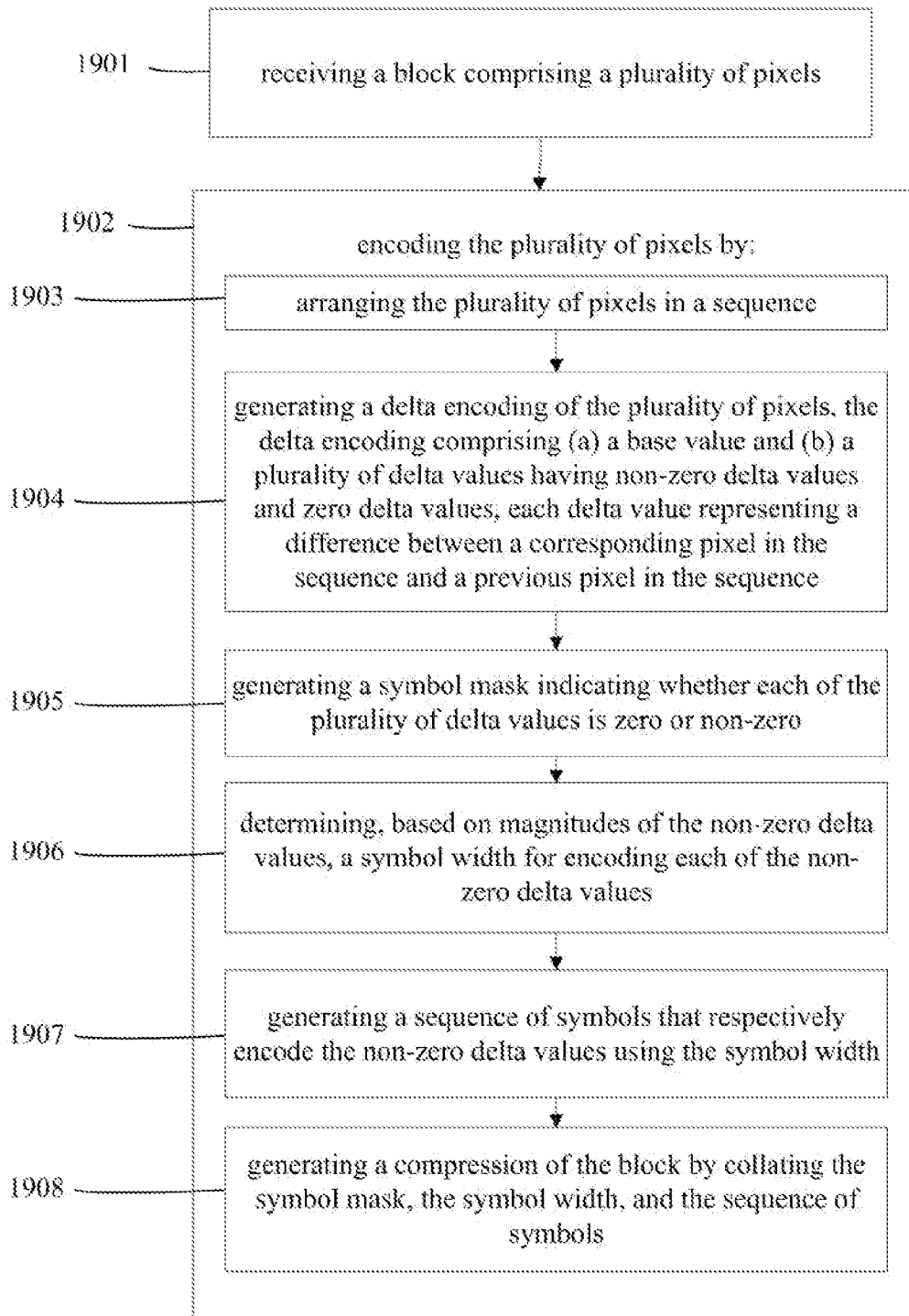
21/25

1700**FIG. 17**

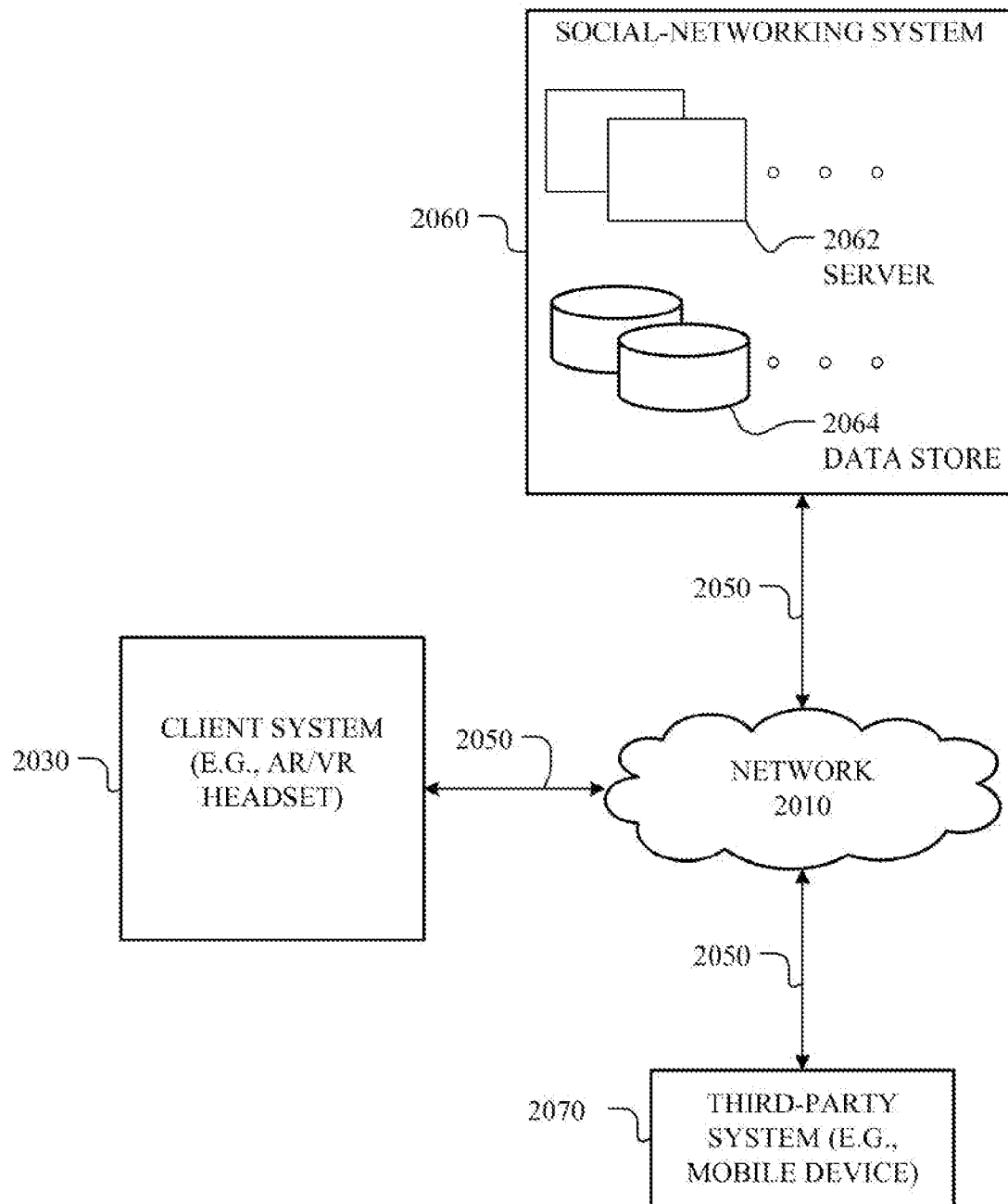
22/25

1800**FIG. 18**

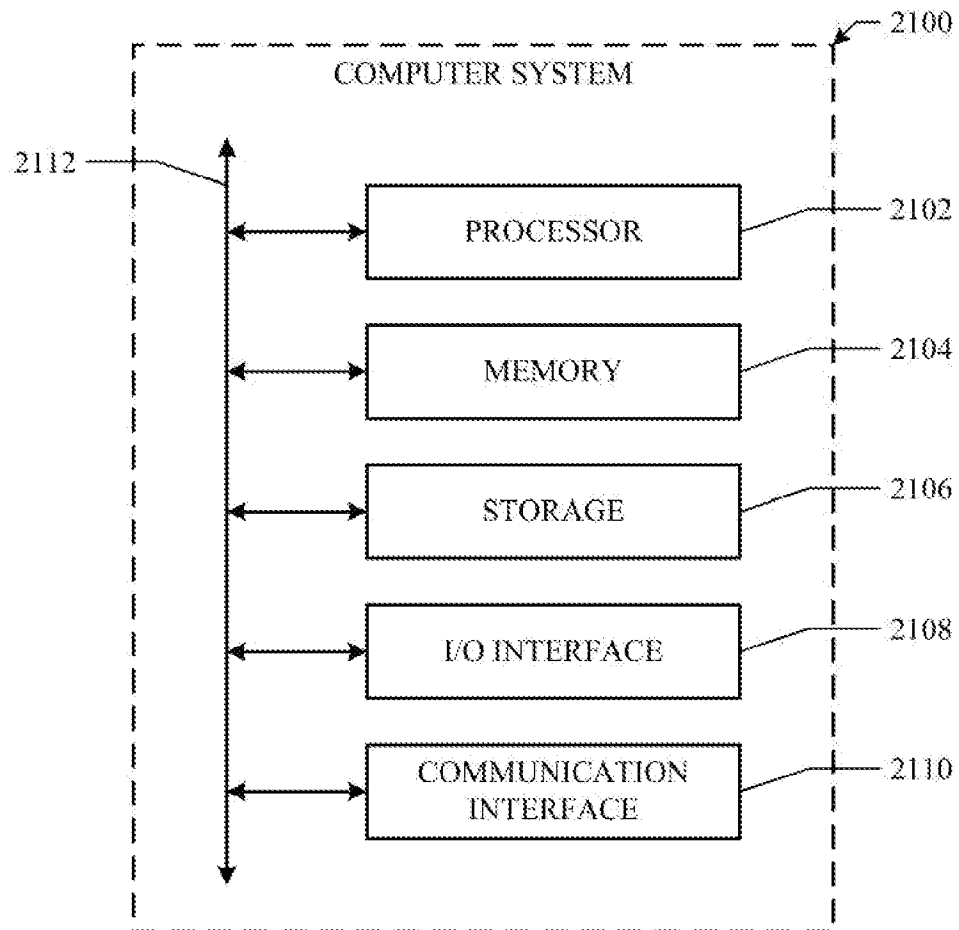
23/25

1900**FIG. 19**

24/25

2000**FIG. 20**

25/25

**FIG. 21**

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/018754

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06T11/40

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 A63F

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	US 6 717 576 B1 (DULUK JR JEROME F [US] ET AL) 6 April 2004 (2004-04-06) paragraph [0030] – paragraph [0031] paragraph [0323] paragraph [0472] paragraph [0527] paragraph [0608] – paragraph [0612] paragraph [0930] – paragraph [0940] paragraph [1094] paragraph [1106] paragraph [1138] paragraph [1309] paragraph [1377] paragraph [1400] – paragraph [1401] paragraph [1409] – paragraph [1419] paragraph [1504] paragraph [1513] paragraph [1548] paragraph [1636] –/–	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

28 June 2023

Date of mailing of the international search report

10/07/2023

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

Garton, Paul

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2023/018754

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	----- US 2020/134913 A1 (NEVRAEV IVAN [US] ET AL) 30 April 2020 (2020-04-30) paragraph [0007] paragraph [0034] paragraph [0045] - paragraph [0048] paragraph [0050] paragraph [0065] paragraph [0074] paragraph [0089] - paragraph [0090] paragraph [0093] - paragraph [0094] paragraph [0096] - paragraph [0100] paragraph [0103] paragraph [0136]	1-15
X	----- US 8 508 544 B1 (MOLNAR STEVEN E [US] ET AL) 13 August 2013 (2013-08-13) paragraph [0003] paragraph [0007] paragraph [0010] paragraph [0029] paragraph [0034] paragraph [0037] - paragraph [0038] paragraph [0041] - paragraph [0042] paragraph [0045] paragraph [0051] - paragraph [0056]	1-15
A	----- KR 2005 0121263 A (NOKIA CORP [FI]) 26 December 2005 (2005-12-26) paragraph [0009] paragraph [0018] - paragraph [0019] paragraph [0028] paragraph [0042] paragraph [0098]	1-15
A	----- JP 6 361267 B2 (YAMAHA CORP) 25 July 2018 (2018-07-25) paragraph [0012] paragraph [0015] paragraph [0097] - paragraph [0099]	1-15
A	----- CN 105 898 273 B (SONY COMPUTER ENTERTAINMENT AMERICA LLC) 10 April 2018 (2018-04-10) paragraph [0041] - paragraph [0042] paragraph [0045] - paragraph [0047] paragraph [0055] - paragraph [0059] paragraph [0084] - paragraph [0085] paragraph [0088] -----	1-15

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2023/018754

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6717576	B1	06-04-2004	AU 5576599 A 14-03-2000
			AU 5687599 A 14-03-2000
			AU 5687899 A 17-04-2000
			AU 5690499 A 14-03-2000
			AU 5779799 A 05-06-2000
			AU 5782599 A 14-03-2000
			EP 1105844 A1 13-06-2001
			EP 1138023 A1 04-10-2001
			JP 3657518 B2 08-06-2005
			JP 3657519 B2 08-06-2005
			JP 4516350 B2 04-08-2010
			JP 2002526842 A 20-08-2002
			JP 2003515798 A 07-05-2003
			JP 2004272928 A 30-09-2004
			KR 20010085424 A 07-09-2001
			KR 20010085426 A 07-09-2001
			US 6597363 B1 22-07-2003
			US 6614444 B1 02-09-2003
			US 6717576 B1 06-04-2004
			US 6771264 B1 03-08-2004
			US 2004130552 A1 08-07-2004
			WO 0011604 A2 02-03-2000
			WO 0011605 A2 02-03-2000
			WO 0011613 A2 02-03-2000
			WO 0011614 A2 02-03-2000
			WO 0019377 A1 06-04-2000
			WO 0030040 A1 25-05-2000
US 2020134913	A1	30-04-2020	CN 110383337 A 25-10-2019
			EP 3563349 A1 06-11-2019
			US 2018240268 A1 23-08-2018
			US 2019172257 A1 06-06-2019
			US 2020134913 A1 30-04-2020
			WO 2018152017 A1 23-08-2018
US 8508544	B1	13-08-2013	NONE
KR 20050121263	A	26-12-2005	CN 1774727 A 17-05-2006
			EP 1614071 A2 11-01-2006
			KR 20050121263 A 26-12-2005
			US 2004207642 A1 21-10-2004
			WO 2004093012 A2 28-10-2004
JP 6361267	B2	25-07-2018	JP 6361267 B2 25-07-2018
			JP 2015210730 A 24-11-2015
CN 105898273	B	10-04-2018	BR 112013016887 A2 30-06-2020
			BR 112013017321 A2 24-09-2019
			CN 103283241 A 04-09-2013
			CN 103329165 A 25-09-2013
			CN 103348360 A 09-10-2013
			CN 103947198 A 23-07-2014
			CN 105894567 A 24-08-2016
			CN 105898273 A 24-08-2016
			CN 105959664 A 21-09-2016
			KR 20130132922 A 05-12-2013
			KR 20140004115 A 10-01-2014
			RU 2013129687 A 20-02-2015

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2023/018754

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
		RU 2013136687 A	20-02-2015
		WO 2012094074 A2	12-07-2012
		WO 2012094075 A1	12-07-2012
		WO 2012094076 A1	12-07-2012
		WO 2012094077 A1	12-07-2012
