(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 June 2024 (20.06.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/129727 A1

(51) International Patent Classification:
H04N 13/117 (2018.01) *H04N 13/00* (2018.01)
H04N 13/194 (2018.01) *H04N 13/139* (2018.01)

(21) International Application Number:
PCT/US2023/083627

(22) International Filing Date:
12 December 2023 (12.12.2023)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/387,003 12 December 2022 (12.12.2022) US
63/387,004 12 December 2022 (12.12.2022) US
PCT/US2023/079864
15 November 2023 (15.11.2023) US

(71) Applicant: **GOOGLE LLC** [US/US]; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(72) Inventors: **OVERBECK, Ryan Styles**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **MARTINDELL, Jaye Lynn**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **FORTES, Tommy Johan**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **ERICKSON, Daniel William**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **GODARD, Clément Louis Jean-Claude**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **HEAL, Kathryn**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **BROXTON, Michael Joseph**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **FLYNN, John Patrick**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **MURMANN, Lukas**; GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(54) **Title:** REAL-TIME VIEW SYNTHESIS

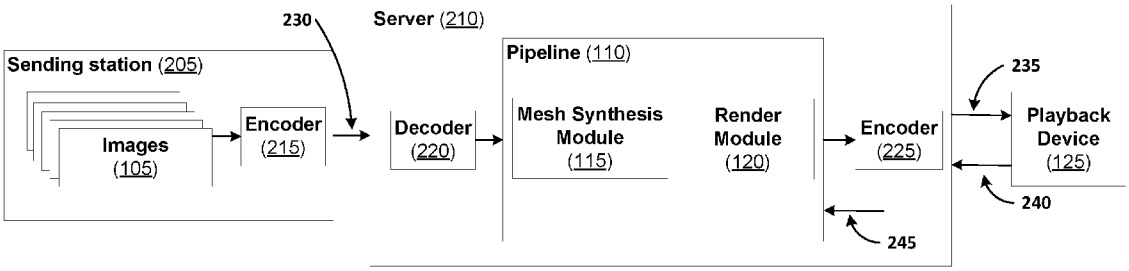

FIG. 2

(57) **Abstract:** A method including receiving, from a source client device, a plurality of two-dimensional (2D) images representing a frame of a streaming three-dimensional (3D) video, receiving, from a playback device, a viewpoint perspective and/or head pose of a user of the playback device, generating a plurality of meshes corresponding to the plurality of 2D images based on the viewpoint perspective and/or head pose, generating a synthesized mesh based on the plurality of meshes, generating a left-eye 3D image and depth based on the synthesized mesh and the viewpoint perspective and/or head pose, generating a right-eye 3D image and depth map based on the synthesized mesh and the viewpoint perspective and/or head pose, and streaming the left-eye 3D image and depth map and the right-eye 3D image and depth map are streamed as the streaming 3D video to the playback device.

WO 2024/129727 A1

[Continued on next page]

(74) **Agent: SMITH, Edward P.** et al.; BRAKE HUGHES BELLERMANN LLP, P.O. Box 1077, Middletown, Maryland 21769 (US).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

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

Published:

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

REAL-TIME VIEW SYNTHESIS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 63/387,003, filed on December 12, 2022, and U.S. Provisional Application No. 63/387,004, filed on December 12, 2022, and also claims priority to International Application No. PCT/US2023/079864, filed on November 15, 2023, the disclosures of which are incorporated herein by reference in their entireties.

FIELD

[0002] Embodiments relate to rendering three-dimensional left-eye and right-eye images.

BACKGROUND

[0003] Generating a three-dimensional (3D) image from a plurality of two-dimensional (2D) images can involve stitching of the 2D images. A stitching operation can include computing all possible translations (x, y, z) between two 2D images with relation to a 3D perspective simultaneously. Computing the translations can determine the best overlap with regard to a cross-correlation measure. If more than two input images are used, the correct placement of portions of the images (sometimes called tiles) can be globally optimized (e.g., the resultant 3D image is modified to remove gaps and overlaps).

SUMMARY

[0004] Example implementations describe a cloud-based neural rendering and view synthesis system and architecture configured to synthesize two viewpoints (e.g., left-eye and right-eye) based on the eye positions of a receiver-side viewer of a streaming sequence of 3D images. The 3D images can be synthesized in the cloud (e.g., by a processor on a network server) based on a plurality of 2D images and rendered prior to streaming the sequence of 3D images.

[0005] In a general aspect, a device, a system, a non-transitory computer-readable medium (having stored thereon computer executable program code which can

be executed on a computer system), and/or a method can perform a process with a method including receiving, from a source client device, a plurality of two-dimensional (2D) images representing a frame of a streaming three-dimensional (3D) video, receiving, from a playback device, a viewpoint perspective and/or head pose of a user of the playback device, generating a plurality of meshes corresponding to the plurality of 2D images based on the viewpoint perspective and/or head pose, generating a synthesized mesh based on the plurality of meshes, generating a left-eye 3D image and depth based on the synthesized mesh and the viewpoint perspective and/or head pose, generating a right-eye 3D image and depth map based on the synthesized mesh and the viewpoint perspective and/or head pose, and streaming the left-eye 3D image and depth map and the right-eye 3D image and depth map are streamed as the streaming 3D video to the playback device.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Example embodiments will become more fully understood from the detailed description given herein below and the accompanying drawings, wherein like elements are represented by like reference numerals, which are given by way of illustration only and thus are not limiting of the example embodiments and wherein:

[0007] FIG. 1 illustrates a block diagram of a client-based streaming pipeline according to an example implementation.

[0008] FIG. 2 illustrates a block diagram of a cloud-based streaming pipeline according to an example implementation.

[0009] FIG. 3 illustrates a block diagram of a view synthesis system according to an example implementation.

[0010] FIG. 4 illustrates a block diagram of a neural rendering system according to an example implementation.

[0011] FIG. 5 is a block diagram illustrating the compute 3D representation stage for the RGB only system according to an example implementation.

[0012] FIG. 6 illustrates a block diagram of a service architecture according to an example implementation.

[0013] FIG. 7 is a block diagram illustrating the cloud pipeline architecture inside the service architecture according to an example implementation.

[0014] FIG. 8 illustrates a block diagram of a cloud pipeline according to an example implementation.

[0015] FIG. 9A illustrates a block diagram of an example machine learning downsampling network according to an example implementation.

[0016] FIG. 9B illustrates a block diagram of an example machine learning view synthesis network according to an example implementation.

[0017] FIG. 9C illustrates a block diagram of an example machine learning upsampling network according to an example implementation.

[0018] FIG. 10 illustrates another block diagram of a view synthesis system according to an example implementation.

[0019] FIG. 11 illustrates a block diagram of a method of cloud based synthesizing and streaming of 3D video according to an example implementation.

[0020] It should be noted that these Figures are intended to illustrate the general characteristics of methods, and/or structures utilized in certain example embodiments and to supplement the written description provided below. These drawings are not, however, to scale and may not precisely reflect the precise structural or performance characteristics of any given embodiment and should not be interpreted as defining or limiting the range of values or properties encompassed by example embodiments. For example, the positioning of modules and/or structural elements may be reduced or exaggerated for clarity. The use of similar or identical reference numbers in the various drawings is intended to indicate the presence of a similar or identical element or feature.

DETAILED DESCRIPTION

[0021] Generating a 3D image by stitching a plurality of 2D images may not be sufficiently fast for real-time streaming of 3D images. In other words, capturing and stitching 2D images in a real-time 3D streaming application can be too slow to provide the desired user experience. Existing solutions may reduce the resolution (e.g., number of pixels) to a very low resolution, and still may not achieve the framerate or frames per second (fps) desired for real-time streaming of 3D images.

[0022] Client-based architectures or client stations can be expensive with the bulk of the expense being the compute operations and/or hardware (GPUs + CPU + motherboard). Client-based architectures or client stations can be expensive and inflexible due to maintenance and/or upgrades. For example, a problem can be that an algorithm can be tightly coupled with the hardware at the client station(s) limiting the ability to upgrade the algorithm. In other words, upgrading the algorithm may require upgrading the client station(s), which can result in undesirable upgrade delays and

costs. Another problem is that client-based architectures or client stations can generate excessive amounts of heat. Without a cooling system in place in the client station room, it becomes uncomfortable for the user resulting in an undesirable user experience.

[0023] These problems can be solved by generating 3D image by using a trained machine learning model with a plurality of 2D images as input. In addition, the trained machine learning model can be implemented as a cloud-based (e.g., on a network device) computing process. Therefore, the client may capture 2D images and communicate the 2D images to the cloud-based architecture to process the 2D images by the trained machine learning model. This cloud-based implementation can reduce the expense and inflexibility of the client-based architecture by minimizing the processing performed by the client-based architecture. In addition, this cloud-based implementation can process the 2D images sufficiently fast for real-time streaming of 3D images.

[0024] Example implementations described herein can use a trained machine learning model for synthesizing a 3D mesh from a plurality of 2D images. Example implementations can further use a trained machine learning model to render the synthesized 3D mesh to generate two 3D images each with a viewpoint perspective (e.g., left-eye and right-eye). The 3D images can then be streamed to a 3D playback device. Example implementations can stream the 3D images at a sufficiently high resolution (e.g., 4K) and framerate (e.g., 30 fps) in a real-time 3D streaming application to provide the desired user experience. Example implementations can be implemented as a cloud-based architecture. In other words, example implementations can be performed by a processor and memory associated with a network server. Including the computing in the cloud can enable upgrade hardware and/or software as upgrades get implemented.

[0025] FIG. 1 illustrates a block diagram of a client-based streaming pipeline according to an example implementation. As shown in FIG. 1, a streaming pipeline 110 (e.g., 3D streaming pipeline) can include a mesh synthesis module 115 and a render module 120. The 3D streaming pipeline 110 can be configured to receive a plurality of 2D images 105 and generate two 3D images that can be streamed to a playback device 125.

[0026] A plurality of cameras (e.g., a camera rig) can be configured to capture the plurality of 2D images 105. In an example implementation, the plurality of cameras (e.g., six cameras) can be rolling shutter RGB cameras that are time synchronized and

share exposure and white balance settings. The plurality of 2D images 105 can represent an image frame. In an example implementation, the plurality of cameras may not capture depth. Therefore, the plurality of 2D images 105 may not include depth information. In an example implementation the 3D streaming pipeline 110 can be on the same device (e.g., a sending station) as the plurality of cameras. In this example implementation, as a frame(s) is captured, the plurality of 2D images 105 can be processed in-line by the 3D streaming pipeline 110.

[0027] In an example implementation the 3D streaming pipeline 110 can be on a different device (e.g., a server) than the plurality of cameras. In this example implementation, as a frame(s) is captured, the plurality of 2D images 105 can be compressed using, for example, the HEVC (h.265) standard at 25 Mbps per camera and then communicated to the server to be processed by the 3D streaming pipeline 110. In this implementation, the plurality of 2D images 105 can be decompressed and processed by the 3D streaming pipeline 110.

[0028] The mesh synthesis module 115 can be configured to synthesize or fuse the plurality of 2D images 105 into a 3D representation of a scene sometimes called a layered mesh (e.g., layered mesh 20 described in more detail below) and the render module 120 can be configured to render the layered mesh as a 3D image representation of a scene. In an example implementation, the layered mesh can represent the complete 3D scene based on the plurality of 2D images 105. In other words, the layered mesh has not been configured to render almost any particular viewpoint perspective (e.g., left-eye and right-eye) of a user viewing the playback device 125. Further, the layered mesh can be used to render any potential viewpoint perspective of a user viewing the playback device 125. Further, the layered mesh can be used to render any potential viewpoint perspective of a user viewing the playback device 125. In other words, the synthesized layered mesh can represent any view perspective corresponding to any head position such that, when displayed on the playback device 125, left eye and right eye images that are rendered based on the synthesized layered mesh can have a view perspective that can be modified with six degrees of freedom (DoF) based on the users view perspective and/or head position.

[0029] The render module 120 can be configured to render two images and generate two depth maps (e.g., one for each of the user's eyes viewing the playback device 125). The two images can be RGB and depth map views. In an example implementation, the playback device 125 can communicate a current or last viewpoint

perspective and/or head pose of the user viewing the playback device 125. Therefore, the render module 120 can be configured to render the two images and generate the two depth maps (e.g., RGB and depth map views) based on the current or last viewpoint perspective and/or head pose of the user. The rendered images and generated depth maps can be streamed (e.g., communicated) to the playback device 125 at, for example, a 4K resolution and 30 fps.

[0030] In an example implementation, the playback device 125 can be configured to perform a last-second reprojection of the rendered images and generated depth maps using the latest viewpoint perspective and/or head pose of the user estimate before rendering to the display of the playback device 125. The reprojection can adjust for user movement (e.g., a change in viewpoint perspective and/or head pose) during the streaming process (e.g., due to system and/or streaming latency).

[0031] In the Example of FIG. 1, a majority of the computing can be performed at the client stations and the video streams can be routed through the cloud but no significant computing operations are performed in the cloud. FIG. 1 illustrates a unidirectional dataflow from a source client station to a target client station. In a full system, the data would flow in both directions. However, since this data flow is symmetric, FIG. 1 depicts a one way for simplicity.

[0032] FIG. 2 illustrates a block diagram of a cloud-based streaming pipeline according to an example implementation. In an example implementation, a reduced amount of computing (as compared to the example system of FIG. 1) can be performed on both client stations and an increased amount of computing (as compared to the example system of FIG. 1) can be performed in the cloud. As shown in FIG. 2, in addition to the elements shown in FIG. 1, the cloud-based streaming pipeline can include a sending station 205 and a server 210. Sending station 205 can include an encoder 215. The server 210 can include a decoder 220 and an encoder 225. Sending station 205 can be communicatively coupled with server 210 via channel 230. The server 210 can be communicatively coupled with the playback device 125 via channel 235 and channel 240. Although not shown, playback device 125 can include a decoder.

[0033] In some implementations, the sending station 205 can perform the type of processing that can be achieved by mobile systems on chip (SOCs) and, possibly, inexpensive graphics processing units (GPUs). For example, the encoder 215 can be configured to perform video encoding (e.g., video compression) and multiplexing (MUX'ing). In addition, sending station 205 can be configured to perform infrared (IR)

stereo computation, and light-weight rendering. Further, more complex and resource expensive processing can be performed in the cloud.

[0034] In some implementations, the sending station can be configured to generate images 105 and the encoder 215 can be configured to encode the images 105 to generate an encoded video stream(s). The encoded video stream(s) can be communicated or streamed to the cloud via channel 230. Channel 230 can be a wired and/or wireless communication channel that uses a communication standard. The encoded video stream(s) can be decoded by the decoder 220 in the cloud at the server 210 and then input to a primary compute stage of the pipeline 110. This stage can be configured to compute a lightweight 3D representation (e.g., the RGB+Depth images described below) that can be encoded by the encoder 225 and streamed to the playback device 125 (e.g., as a target client station). The computing stages of server 210 in FIG. 2 may or may not run on the same machine or even in the same program.

[0035] In FIG. 2, the media dataflow is from left to right, from, for example, sending station 205 (e.g., as a source client station) to playback device 125 (e.g., as a target client station). However, there can be at least one stream of data that flows the other direction (see FIG. 7). For example, a stream of data flowing from the playback device 125 to the server 210 via channel 240 can include information associated with a viewpoint perspective and/or head pose 245 of a user of the playback device 125. This information can be the latest viewpoint perspective and/or head pose 245 information as measured at the playback device 125. Thus, pipeline 110 can use, for example, recent viewpoint perspective and/or head pose 245 information to create a light-weight 3D representation based on the viewpoints near that head pose. The light-weight 3D representation somewhat close to the viewpoint can be the preferred 3D representation and may degrade somewhat further away from the viewpoint. A stereo pair (e.g., left-eye and right-eye) of RGB+Depth images can be sufficient to give a desirable quality result from viewpoints near or based on the head pose. In some implementations, the layered mesh can be used to generate several sets of stereo RGB+Depth images where each set of stereo RGB+Depth images is optimized for different viewpoints and/or head poses.

[0036] FIG. 3 illustrates a block diagram of a view synthesis system according to an example implementation. The view synthesis system can be configured to blend image weights and densities and reconstructs depth layers in the form of a layered mesh representation. The view synthesis system can be based on a DeepView algorithm. For

example, the view synthesis system can be configured to generate a layered mesh output through a process of iterative refinement. As shown in FIG. 3, the mesh synthesis module 115 includes a rectification module 305, a downsample module 310, a synthesis module 315, and an upsample module 320.

[0037] The rectification module 305 can be configured to generate rectified images 5. In an example implementation, the rectification module 305 can be configured to reproject each of the images 105 to a layered mesh plane as the rectified images 5. In an example implementation, the layered mesh plane can be a near clipping plane. The rectification module 305 can be configured to decrease resolution of each of the images 105 during the reprojection of each of the images 105. The reprojection of each of the images 105 can ensure or help to ensure that image coordinates are consistent across each of the rectified images 5.

[0038] The downsample module 310 can be configured to downsample each of the rectified images 5 by, for example, eight times (8x) using, for example, a trained machine learning downsampling network. The machine learning downsampling network can be configured to generate low resolution feature maps 10 for each of the rectified images 5. The synthesis module 215 can be configured to generate a feature layered mesh 15. The feature layered mesh 15 can be a low-resolution layered mesh. In an example implementation, the feature layered mesh 15 can include 288 x 184 pixels x 16 layers.

[0039] The upsample module 320 can be configured to generate layered mesh 20. In an example implementation, the upsample module 320 can be configured to increase the resolution of the feature layered mesh 15. In an example implementation, the upsample module 320 can be configured to increase a density 30 in resolution by, for example, eight times (8x). For example, the density 30 of the feature layered mesh 15 can be increased to a 1080p resolution. In an example implementation, the upsample module 320 can be configured to refine blend weights 25 of the feature layered mesh 15. However, the blend weight 25 and mesh vertices 35 of the layered mesh 20 can remain at a low resolution. Leaving the blend weights 25 and mesh vertices 35 at a low resolution can increase efficiency. For example, the final 3D image may not be sensitive to the resolution of blend weights 25 and mesh geometry. By contrast, the final 3D image can be sensitive to alpha and RGB resolution.

[0040] FIG. 4 illustrates a block diagram of a neural rendering system according to an example implementation. As shown in FIG. 4, the render module 120 can be

configured to generate a left eye (LE) image 25 and a right eye (RE) image 30 based on the layered mesh 30. In an example implementation, the LE image 25 and RE image 30 can each be a 4k RGB plus depth image. As shown in FIG. 4, the render module 120 can include a RE project module 405, a LE project module 410, a RE blend module 415, a LE blend module 420, a RE over-composite module 425, and a LE over-composite module 430. In an example implementation, the layered mesh 20 can include 16 mesh layers each including an associated blend weight and density value. Further, the layered mesh 20 can be used to render any potential viewpoint perspective of a user viewing the playback device 125. In other words, the synthesized layered mesh can represent any view perspective corresponding to any head position such that, when displayed on the playback device 125, left eye and right eye images that are rendered based on the synthesized layered mesh can have a view perspective that can be modified with six degrees of freedom (DoF) based on the users view perspective and/or head position.

[0041] In an example implementation each mesh layer of the layered mesh 20 can be rasterized into an image space of the output view to be rendered. In some implementations, the mesh layers of the layered mesh 20 can be rasterized in a back-to-front order (e.g., last background layer to frontmost foreground layer). This process returns the intersection triangle index and barycentric coordinates for each pixel in the output view. In an example implementation, this information can be used to (1) look up a set of blend weights (e.g., using barycentric interpolation of the mesh layer blend weights produced by the model), (2) look up a density value (e.g., using barycentric interpolation of the mesh layer densities produced by the model), and (3) compute 3D coordinates for the point of intersection. In some implementations, the set of blend weights can be based on the number of cameras (e.g., six cameras). These 3D coordinates can be projected back into the image planes of the original high resolution input views to determine (e.g., using bilinear interpolation) an RGB value from each input image.

[0042] In an example implementation, the blend weights can be activated using a softmax non-linearity and then used to compute a simple weighted average of the RGB values from the input views. The density value can be activated via a softplus non-linearity and converted to an alpha value. In an example implementation, rasterization of one layer to an output view can generate an RGB plus alpha value for each pixel in a layer.

[0043] This sequence of rasterization, projection and sampling steps includes the signal flow shown in FIG. 4. For example, the signal flow includes projecting the input images onto the mesh geometry (the RE project module 405 and the LE project module 410), blend the mesh together using the blend weights, combine with alpha values from the layer, and then project the result into the output view (the RE blend module 415 and the LE blend module 420).

[0044] In an example implementation, the non-linear activations can include activating after resampling. In other words, the non-linear activations can include projecting the input images, blend weights, and densities into the output viewpoint. When the network is trained through a differentiable version of the rendering process described above, it can learn to expect these activations to occur in a higher resolution space, and the network can actually learn to take advantage of post-interpolation activations to produce sharper, higher resolution results even when working with relatively low-resolution blend weights and alpha densities. In an example implementation, activating after resampling can be called deferred rendering or deferred sampling of the layered mesh. Deferred rendering or deferred sampling of the layered mesh can help achieve the high quality 4k outputs, even though the layered mesh contains 1080p densities and low-resolution (e.g.,135 x 240 pixel) geometry and blend weights.

[0045] In an example implementation, rasterization can be repeated for each, e.g., 16, of the mesh layers of the layered mesh 20, and then the resulting RGB plus alpha layers can be rendered using alpha compositing to produce a final RGB image for a particular output viewpoint (the RE over-composite module 425 and the LE over-composite module 430). In an example implementation, the RE over-composite module 425 and the LE over-composite module 430 can be configured to generate a depth channel by replacing the RGB in each layer with that layer's disparity, and then compositing disparity plus alpha layers.

[0046] In some implementations, the RE over-composite module 425 and the LE over-composite module 430 can be configured to use the layer mesh's depth. For example, the layer mesh's depth can be used for pixels with high enough density/alpha. In some implementations, (1) the output color image can be cleared and/or converted to black, (2) the output depth image can be cleared/converted to maximum, e.g., one (1), (3) the meshes can be processed back-to-front, (4) an alpha can be calculated from

the density, and (5) the pixels having an alpha below a threshold can be discarded. Otherwise (e.g., for pixels without a high enough density/alpha), (1) the input colors can be blended using the blend weights, (2) the resulting color can be blended using the alpha onto the output image, and (3) the depth can be written to the depth image.

[0047] In an example implementation, the deferred rendering or deferred sampling technique has the advantage of decoupling the resolutions of the input views, the network outputs, and the final RGB plus depth rendered result. This is advantageous because the decoupling can allow tuning the resolution of each component separately, which is useful for trading off quality against performance. For example, the speed of the RGB plus depth renderer can be increased by outputting RGB plus depth images at 1440p while using 4k inputs.

[0048] The layered mesh 20 described above can be used to represent the 3D structure of the scene. In addition, layered mesh provides a mesh representation that can be rasterized to produce the RGB plus depth views used to stream to the playback device 125. Layered meshes can be related to multi-plane images (MPIs). With both MPIs and layered meshes, a viewpoint can be rendered via a combination of perspective projection and alpha compositing. Mesh layers can occupy disparity bands that, like MPI planes, can be equally spaced in disparity (1/z) space. Views can be rendered by first projecting layers into the output viewpoint and then alpha compositing from back to front. However, unlike the flat plane geometry of the MPI, mesh layers can have network-produced geometry that molds itself to the shape of the object being reconstructed. This allows layered meshes to achieve similar quality to an MPI but with far fewer layers. Using layered meshes layers a 30 fps at high resolution can be achieved, as layered meshes enable an efficient (10x faster than with MPIs) way to perform learned upsampling and rendering.

[0049] There can be some latency that is based on (e.g., the sum of) the time it takes for the head pose information to arrive from the target client station and the time it takes to compute the 3D representation, encode the 3D representation, and stream the 3D representation to the target client station. During this time the user may have moved by some amount. In an example implementation, maintaining real-time (e.g., 30-60Hz) framerates can provide stereo RGB+Depth that look good enough from many user viewpoint positions. However, example implementations can also re-render the stereo pair (e.g., left-eye and right-eye) of RGB+Depth images based on a more recent viewpoint perspective and/or head pose of the user. In other words, the viewpoint

perspective and/or head pose 245 can be updated and/or used at various stages of the pipeline and/or the viewpoint perspective and/or head pose 245 can be used in a rerendering process on the playback device 125.

[0050] Some implementations can include use of two different techniques for view synthesis including a stereo IR technique and an RGB only technique. Despite the differences between the stereo IR technique and the RGB only technique, both can be implemented in the cloud-based architecture shown in FIG. 2. The differences between the implementations of the pipeline data streams and stages can be as outlined in Table 1. One of the differences is in the compute 3D rendering representation stage of render module 120.

[0051] For the stereo IR system, in some implementations the depth computation can be performed on the sending station 205 and the resulting depth maps can be communicated as video streams to the server 210 via channel 230. In some implementations, the render module 120 can be configured to perform ray casting and volumetric fusion processing along with most of the rendering and compositing. These workloads can map to (e.g., execute on) standard graphics cards. Other compute loads can use more machine learning (ML) compute resources that may be added, such as neural blending (e.g., a ML model used to blend two or more images together by image cropping and insertion), glasses rendering, and relighting. These may use tensor processors such as the ones on tensor GPUs or TPUs.

	Stereo IR	**RGB Only**
Encoded Video Streams	Encoded 3-4 RGB streams + 3 depth streams	Encoded 4-8 RGB streams
Decode Video Streams	Decode video streams	Decode video streams
Uncompressed Video Streams	3-4 RGB streams + 3 depth streams	4-8 RGB streams
Compute 3D Rendering Representation	Raycasting + Volumetric Fusion Heavy Rendering + Compositing Neural Blending Glasses Rendering Relighting	Inference Compute 3D delivery format
Lightweight 3D Rendering Representation as Video Streams	2 RGB streams + depth streams	2 RGB streams + depth streams
Encode Video Streams	Encode Video Streams	Encode Video Streams
Encoded Video Streams	Encoded 2 RGB + depth streams	Encoded 2 RGB + depth streams

Table 1

[0052] FIG. 5 is a block diagram illustrating the compute 3D representation stage for the RGB only system according to an example implementation. Computing the 3D representation stage can be an element of pipeline 110. Computing the 3D representation stage can be an alternative implementation of pipeline 110. Computing the 3D representation stage can be an additional process within the implementation of pipeline 110. The compute 3D representation stage can include the decoder 220, an inference module 505, and a format module 510. The decoder 220 can generate an uncompressed video stream(s) 510. The inference module 505 can be configured to generate a MPI (e.g., layered mesh 20). The format module 510 can be configured to generate a 3D delivery format video stream(s) as a lightweight 3D 520 representation (e.g., render module 120).

[0053] In some implementations, the MPI 515 can be generated as a heavy-weight 3D representation of the scene. An MPI can be a stack (e.g., an image stack) of semi-transparent, colored layers arranged at various depths. The light-weight 3D 520 delivery format can be computed based on the MPI 515 (e.g., the stereo RGB+Depth as described above). In some implementations, the inference module 505 may require tensor processors and format module 510 can be configured to convert the MPI 515 to the 3D delivery format efficiently either on regular GPUs or tensor processors. Note that viewpoint perspective and/or head pose 245 can be injected into both stages. By processing viewpoint perspective and/or head pose 245, the inference module 505 can generate an MPI 515 that can be suitable for the user's latest head pose. A more recent viewpoint perspective and/or head pose 245 can be used when the format module 510 is computing the 3D delivery format. For example, the viewpoint perspective and/or head pose 245 can be continually updated. Therefore, the viewpoint perspective and/or head pose 245 can change after the viewpoint perspective and/or head pose 245 can be input to the inference module 505 resulting in a new or updated head pose being input to the format module 510. In an example implementation, for the stereo IR pipeline, injecting the viewpoint perspective and/or head pose 245 into multiple stages can be useful.

[0054] In order to run the above-described computing pipeline(s) in the cloud and have the pipeline interface with a client application, the pipeline may be embedded into an existing service architecture. The service architectures can become complicated

as various data packets are routed to different services served on different machines, compile the results, and deliver the results to clients.

[0055] FIG. 6 illustrates a block diagram of a service architecture according to an example implementation. As shown in FIG. 6 the service architecture can include a plurality of devices 605, a plurality of lens 610, a plurality of media routers 615 including nodes, a media controller 620, and a multiple address system 625. The plurality of lens 610, the plurality of media routers 615, and the media controller 620 can be operating in a media routing layer 630. At least a portion of the service architecture can be operating within cloud 635.

[0056] FIG. 6 illustrates a set of devices 605 that can be joined in one session (e.g., a meeting). A device 605 can be any computer (e.g., phone, tablet, laptop) running a client that is connected to the session (e.g., a meeting). Media packets can enter cloud 635 by way of servers, which can accept media packets from the outside world and route the media packets to the correct cloud service. For example, packets can be routed to media routers 615.

[0057] In an example implementation, the media routers 615 can be the core of the infrastructure for managing media packets. The media routers 615 can form a mesh network that transfers media packets between each other to be processed and delivered to other devices in the meeting. There can be multiple benefits for using this mesh network topology, a primary one being that the mesh topology can make the session (e.g., a meeting) more robust. For example, if one media router 615 goes down, it doesn't bring down the whole meeting. The devices 605 connected to that media router 615 that went down may reconnect to the session via a different media router 615.

[0058] Example implementations may use other support services. The other support services can provide an interface to manage, for example, the non-media metadata such as who is in the meeting, the name of the meeting, and an overall mapping of how media packets can flow through the media routers 615. For implementing new compute services within the architecture or session, the media routers 615 can use an extension interface.

[0059] In order to implement the cloud pipeline architecture described above (Cloud Pipeline Architecture) inside the service architecture, a service that implements a system interaction application programming interface (API) can be used. This service would manage the processing pipeline according to an example implementation.

[0060] FIG. 7 is a block diagram illustrating the cloud pipeline architecture inside the service architecture according to an example implementation. As shown in FIG. the cloud pipeline architecture can include a client 705, 750, a server 715, 720, 740, 745, a media router 720, 735, and a pipeline 725, 730.

[0061] FIG. 7 shows media packets can flow through the service architecture. Given two clients 705, 750, media packets can enter the cloud pipeline architecture by way of server 710 and 745 as load servers. The server 710 and 745 can be a general load balancing service and may route the packets to the nearest servers 715, 740 that also have the least load. From there, the packets can be delivered to the nearest media router 720, 735. Media routers 720, 735 can be stateful, and the same media router 720, 735 can be used for the duration of the session (e.g., meeting) unless the media router 720, 735 goes down for some reason. From there, the media packets can be delivered to the server implementing the media router extension that manages the cloud pipeline 725, 730. Like the media router 720, 735 themselves, the extensions can be stateful. In an example implementation, two pipelines 720, 735 can be required, one for each direction of traffic.

[0062] In an example implementation, operating the cloud pipeline 720, 735 as close to the target client (e.g., client 705, 750) as possible so that the cloud pipeline 720, 735 can have access to recent head poses can be beneficial. This would mean running the cloud pipeline 720, 735 at the egress media router (e.g., the media router closest to the target client). However, the media router 720, 735 extensions may exist only at the ingress media router 720, 735 (e.g., the media router closest to the source client). The ingress media router 720, 735 can be the media router 720, 735 physically closest to the source device (e.g., client 705, 750). Example implementations can make a new extension API for egress media router 720, 735. Although not shown, pipeline 725 can be communicatively coupled with pipeline 730 and the viewpoint perspective and/or head pose 245 can be communicated between pipeline 725 and pipeline 730.

[0063] FIG. 8 illustrates a block diagram of a cloud pipeline according to an example implementation. As shown in FIG. 8, the cloud pipeline includes a decoder 805, a graphics module 810, a tensor module 815, a graphics module 820, and an encoder 825. The cloud pipeline receives an encoded (e.g., compressed) video stream(s) 830. The decoder 805 decodes (e.g., decompresses) the encoded video stream(s) 830 as uncompressed video stream(s) 835. The graphics module 810 generates data 840 based on the uncompressed video stream(s) 835. The tensor module 815 generates data 845

based on data 840. The graphics module 820 generates uncompressed video stream(s) 850 based on data 845. The encoder 825 generates (e.g., encoder and/or compresses) encoded (e.g., compressed) video stream(s) 855 based on the uncompressed video stream(s) 850.

[0064] The simplified cloud pipeline model in FIG. 8 includes at least five (5) basic compute elements: 1.) decode video, 2.) graphics compute, 3.) tensor compute, 4.) graphics compute, and 5.) encode video. These elements can be important to distinguish because different types of hardware are specialized for each of these types of compute loads. Note that the tensor compute stage can be (but may be optionally) sandwiched between two graphics compute nodes.

[0065] GPUs may handle video encode and decode efficiently and can have solid tensor processing capabilities. The high bandwidth connection between video decoders, tensor cores, and video encoders should easily handle the highest bandwidth data streams, including the uncompressed video streams. Example implementations may use the T4s instead of V100s in the stereo IR pipeline because there may be value in using a GPU that can host standard rendering packages. For example, new backgrounds and transition effects can be easier to design and implement standard rendering packages. These rendering tasks could be implemented efficiently on a pure tensor GPU like the V100 but may require significantly more engineering involvement. The cloud pipeline implementations can be based on TPUs at least for the RGB only pipeline. TPUs can be used for tensor processing workloads. TPU host machines could expand hardware support for video encoding or decoding.

[0066] In some implementations the video encoding and decoding can be performed by host CPU cores. For example, some implementations may use eight (8) TPU chips and 108 CPU cores (for ~13 CPU cores per TPU core). Processing and Data Flow can include (1) the video feeds may arrive from the NIC (network interface card), transferred over PCIe, and the video will be queued in host RAM; (2)the video feeds can be divided evenly by the number of cores (8 per machine); (3) CPU decoder threads decode the video feeds and feed the decoded results to the cores via peripheral component interconnect (PCIe); CPU cores can be divided evenly over the video threads; (4) each video feed can be processed by the cores and the results stored in the HBM (High bandwidth memory); (5) the data can be transferred between cores by way of the super-fast ICI bridge; (6) when finished, the data must travel back across PCIe to be encoded to video streams on the CPU cores.

[0067] TPU performance vs. GPU. TPU performance testing suggest that TPUs can be faster than tensor GPUs of the same generation. TPUs can require padding data and weights to specific amounts in order to get optimal performance. It is possible that the work to pad optimally for TPU may get closer performance results.

[0068] The TPUs for the stereo IR pipeline architecture can be a useful approach for the RGB only pipeline. TPUs can be designed mostly for matrix multiplies. However, most of the stereo IR rendering pipeline may not be matrix multiplies and may not provide a standard graphics pipeline. Example implementations can include use of a machine with eight (8) TPUs and other processors.

[0069] Example implementations can include use of core and/or edge computing. The main reason to prefer edge over core would be to reduce latency between the compute 3D rendering representation stage (as shown in FIG. 2) and the target client. This should work well as long as a session is one-on-1. If, instead, there are N people in the session, then the compute requirements are O(N^2), a compute 3D rendering representation compute block for all N-1 incoming streams close to all N target clients can be used. If instead, the compute 3D rendering representation is placed close to the source, better than O(N^2) may be possible. For example, sending N^2 streams of light-weight 3D rendering representations could be communicated with each of the N source clients making N-1 streams. However, there is likely significant compute that can be shared in creating the N-1 streams. For example, in the RGB Only path (e.g., shown in FIG. 3), it is likely that a majority of the compute will be required for making an intermediate 3D representation (e.g., an MPI) that could work for all N-1 target clients, and then uses much less compute to generate the light-weight representation that is optimized for each of the N-1 target clients. In this configuration the compute can be closer to O(N). Also, in this configuration, using edge compute may not help.

[0070] Note that this configuration, where the compute is placed closer to the source, can be more compatible with the media router design, in particular the media router extension mechanism which supports extension services at the ingress media router (e.g., the one closer to the source). This can be useful for extensions like the audio processor, so that each audio stream may be processed once before getting multiplexed to N-1 targets. Similarly, the diffractor extension may decode each stream once and then encode a maximum of N-1 times but probably less since multiple targets probably share the same optimal video codec and resolution.

[0071] FIG. 9A illustrates a block diagram of an example machine learning downsampling network according to an example implementation as an example element (or implementation) of the downsample module 210. The machine learning downsampling network 905 (e.g., of downsampling module 310) can include a plurality of convolution layers 910-1, 910-2, 910-3, 910-4, 910-5, 910-6, 910-7. For example, the machine learning downsampling network 905 can include a series of strided convolutional layers. Therefore, the convolution layers 910-1, 910-2, 910-3, 910-4, 910-5, 910-6, 910-7 can be strided convolutional layers. Striding in a convolution layer indicates a number of pixels the filter matrix of the convolution layer moves across the input image. The stride length of a convolution layer indicates how many steps are taken when sliding the filter matrix across the image. In some implementations, the stride length of the convolution layers 910-1, 910-2, 910-3, 910-4, 910-5, 910-6, 910-7 can be one or two. For example, convolution layers 910-1, 910-3, 910-5, and 910-7 can have a stride length of one and convolution layers 910-2, 910-4, and 910-6 can have a stride length of two.

[0072] The plurality of convolution layers 910-1, 910-2, 910-3, 910-4, 910-5, 910-6, 910-7 can be configured to reduce the resolution of the rectified images 5. For example, the resolution of the rectified images 5 can be reduced by eight times (8x). For example, convolution layers 910-3, 910-4 can be configured to reduce the resolution of the rectified images 5 by two times (2x), convolution layers 910-5, 910-6 can be configured to reduce the resolution of the rectified images 5 by two times (2x), and convolution layers 910-7 can be configured to reduce the resolution of the rectified images 5 by two times (2x) for a total resolution reduction of eight times (8x). In some implementations, the convolution layers 910-1, 910-2, 910-3, 910-4, 910-5, 910-6, 910-7 can be configured to increase each of the rectified images 5 channel count from, for example, 4 to 32 channels.

[0073] FIG. 9B illustrates a block diagram of an example machine learning view synthesis network according to an example implementation as an example element (or implementation) of the synthesis module 315. As shown in FIG. 9B, in an example implementation, the feature layered mesh 15 layers can be initialized to have a flat geometry and project the feature map 10 onto these layers to generate a plane sweep volume (PSV). Then, according to an example implementation, an initialization network (e.g., images to layers transition 915-1 and neural network 920-1 (e.g., a convolutional neural network (CNN))) can be configured to compute an initial estimate

of the feature layered mesh 15 (e.g., then output of neural network 920-1 or LayeredMesh) based on the PSV. At this point, the feature layered mesh 15 layers can include network features (with 32 channels). In an example implementation, the first 30 channels of the machine learning view synthesis network can include abstract network features (e.g., the features do not have any particular physical interpretation, therefore the network can be free to use features of the feature layered mesh 15 in any useful way). However, in an example implementation, the final two channels of the feature layered mesh 15 can be used by the machine learning view synthesis network to derive depth and density information. Accordingly, the machine learning view synthesis network can be configured to generate (or learn to generate) features for learning depth and density information.

[0074] The feature layered mesh 15 generated by the initialization network can be refined via two successive update steps. A first update step (Update 1) can include layers to images transition 925-1, images to layers transition 915-2, neural network 920-2 (e.g., a CNN), visibility components 930-1, and an activation block 935-1. A second update step (Update 2) can include layers to images transition 925-2, images to layers transition 915-3, neural network 920-2 (e.g., a CNN), visibility components 930-2, and an activation block 935-2. During each update step, the current feature layered mesh 15 can be projected back into the input feature map 10. In some implementations, the current feature layered mesh 15 can be compared to determine how well the current feature layered mesh 15 can approximate the real imagery captured by the input cameras. However, the feature layered mesh 15 that has been projected back into the input feature map 10 can be used in any useful process. In an example implementation, when projecting the features of the feature layered mesh 15 into the viewpoint of the input feature map 10, a geometry derived from the second to last depth channel in the layered mesh can be used. This channel can be activated (e.g., activation block 935-1, 935-2, 935-3) with a tanh nonlinearity, scaled by the layer width, and added to a set of depth anchors (e.g., the output of activation block 935-1, 935-2, 935-3) that are equally spaced in disparity. Constructing geometry using this technique can prevent feature layered mesh 15 layers from overlapping, because each layer can inhabit the layers own unique disparity band.

[0075] The layer geometry can then be used to warp from a layer space to a view space (and back again). While in view space the last channel (e.g., the density feature) of the feature layered mesh 15 can be used to perform compositing operations

that can help communicate visibility information across the layers. These visibility components 930-1, 930-2, 930-3 can be used or help the update network because the visibility components 930-1, 930-2, 930-3 can reason about occlusions and understand across-layer dependencies. The visibility components 930-1, 930-2, 930-3 can be accumulated over and include a net transmittance. Accumulated over can include the reconstruction of the scene from behind the plane, and the net transmittance can be the soft occlusion mask for the plane.

[0076] The computation of visibility components 930-1, 930-2, 930-3 can improve the functioning of machine learning view synthesis network, because the computation of visibility components can be the time when information is communicated between layers. To complete the update step, the visibility components 930-1, 930-2, 930-3 can be warped from each of the input view spaces back to a central layered mesh representation and then these features can be input into the update network. The update network can be configured to generate a delta that can be added via a residual connection to the layered mesh computed in the previous iteration. This can iteratively (e.g., a plurality of update steps) generate the feature layered mesh 15 based on the feature map 10. In Addition, activation block 935-3, layer to images 925-3, and visibility components 930-3 together can generate gradient computations 945. Alternatively (or in addition), the layer to images 925-3 can calculate visibility components 930-3 (as the gradient computations 945) based on the feature layered mesh 15 and the depth calculated by the activation block 935-3. This reconstruct, check, and then refine strategy implemented in the update steps can be repeated several times, and strategy can function like an iterative optimization algorithm. Convergence to a high-quality solution can occur in a few (e.g., three) iterations.

[0077] FIG. 9C illustrates a block diagram of an example machine learning upsampling network according to an example implementation as an example element (or implementation) of the upsample module 320. As shown in FIG. 9C, in an example implementation, the machine learning upsampling network 980 can use the feature layered mesh 15 computed by the view synthesis network 940 and a final set of visibility components (gradient computations 945), and then processes these using a series of convolutions 950-1, 950-2, 950-3, 950-4, 950-5, 950-6, 950-7, 950-8, 950-9, 950-10, and 950-11, concatenations 955-1, 955-2, and a squeeze and excitation network (including the features and weights, a softmax 965 with the multiplication and addition elements) to estimate a low-resolution blend weight 25, mesh vertex 35 positions, and

higher resolution density 30 layers of the layered mesh 20. In some implementations, the convolutions 950-7, 950-8 can be referred to as a blend model and the convolutions 950-9, 950-10, 950-11 can be referred to as a density model. In an example implementation, the above-mentioned density 30 layers can be upsampled via a depth2space transform 970. In some implementations, the feature layered mesh 15 (e.g., a second to last channel of the feature layered mesh 15) can be activated and converted 975 to produce a tensor containing 3D mesh vertex 35 locations.

[0078] Image matting is the process of estimating the foreground object in images and videos. For example, image matting can be the process of extracting an alpha matte that separates foreground and background objects in an image. The alpha matte can be a binary mask defining the image elements that are visible. The alpha matte can be used as the transparency map of a top image. The matting problem can be defined for an image I as I =AF + (1-A)B, where F is the foreground, B is the background, and A is the alpha matte. Image matting can be based on the assumption that an image is a composite of foreground and background images, and hence, the intensity of each pixel is a linear combination of the foreground and the background.

[0079] The image can be segmented in a binary manner where a pixel either belongs to the foreground or background. This type of segmentation, however, may have difficulty processing natural scenes that contain fine details (e.g., hair and fur) which can be based on estimating a transparency value for each pixel of the foreground object. Alpha mattes can be precise, preserving strand-level hair details and accurate foreground boundaries.

[0080] In some implementations, matting can be implemented using a machine learned model. The machine learned model can be a convolutional neural network including a sequence of encoder-decoder blocks. Training the machine learned model can include progressively estimating a high-quality alpha matte.

[0081] An example implementation can perform background removal tools based on a streaming media machine learning model combined with a virtual machine service. Steps can include using the machine learned model to segment the person, prepare the mask with a low resolution, and/or improving this mask and its alignment considering image borders. When all or some of these tasks are completed, the video output can be rendered, and the background can be blurred and/or changed using a mask.

[0082] Example implementations can also use a trained machine learning model for synthesizing a 3D mesh from a plurality of 2D images. Example implementations can further use a trained machine learning model to render the synthesized 3D mesh to generate two 3D images each with a viewpoint perspective (e.g., left-eye and right-eye). The 3D images can then be streamed to a 3D playback device. Example implementations can be used to stream the 3D images at a sufficiently high resolution (e.g., 4K) and framerate (e.g., 30 fps) in a real-time 3D streaming application to provide the desired user experience.

[0083] FIG. 10 illustrates another block diagram of a view synthesis system according to an example implementation. Example implementations can combine the machine learning model used for background removal with the machine learning model used for synthesizing a 3D mesh from a plurality of 2D images and/or the machine learning model used to render the synthesized 3D mesh. For example, the background subtraction model can be used to generate a background subtraction matte for ground truth training data. Then, training the synthesizing model on this ground truth data to create a background subtraction mask in its alpha channels in order to get the rendered results to match this ground truth result. Accordingly, the mesh synthesis module 115 can be modified to include a matting module 1005. The matting module 1005 can be configured to generate a foreground layered mesh 1010 based on feature layered mesh 15. The matting module 1005 can include a matting model.

[0084] During runtime, in some implementations, the matting model can be executed (e.g., run) on a target image and the rendering loss can be masked for background pixels using the matte. A loss between the total alpha and the matte can be added. A goal of this implementation can be to minimize haloing in the image with the background removed.

[0085] An example implementation can increase a weight in the machine learned model. The increased weight can increase the loss on the alpha matte and reduce the haloing. However, the haloing may not be completely removed, and foreground edges may be blurred.

[0086] Example implementations can include generating and/or compositing a random background on the target images. The same background can be used when rendering the image which can force the machine learning model to produce and alpha equal to 0 for the background.

[0087] An example implementation can simplify the random background by using one color. Example implementations can use a single-color random background than can minimize haloing and the blurring of foreground edges. Example implementations can be performed when removing a background using a combined machine learning model used for synthesizing a 3D image and removing a background of the 3D image.

[0088] In an alternative (or additional) example implementation a random noise background can be used. The image synthesis model can be configured to generate a semi-transparent layer(s) in 3D. These layers can be rendered from a different viewpoint to generate the final image. The machine learning model can assume the background to be black. Instead, in an example implementation, a random noise image can be used as the background. This random noise image can force the machine learning model to produce layers which are opaque such that the random noise may not be visible in the final image and can prevent the machine learning model from relying on a black background.

[0089] In an example implementation, in a machine learning training operation the background of the target image can be replaced with the same random noise. For example, a high-quality and slow matting model can be used to provide the matte used to replace the background. This can force the model to produce a transparent image for the background. The generated image may not suffer from haloing and can reproduce thin details. The increased quality can be explained by the power of the reconstruction loss used to train the model. In an example implementation, a Learned Perceptual Image Patch Similarity (LPIPS) metric can be used to compare image features at multiple scales. The added benefit of this implementation can be the simplicity as the matting objective is folded into the rendering loss.

[0090] Example 1. FIG. 11 illustrates a block diagram of a method according to an example implementation. As shown in FIG. 11, in step S1105 a plurality of two-dimensional (2D) images representing a frame of a streaming three-dimensional (3D) video is received from a source device or source client device. In step S1110 a viewpoint perspective and/or head pose of a user of a playback device is received. In step S1115 a plurality of meshes corresponding to the plurality of 2D images is generated based on the viewpoint perspective and/or head pose. In step S1120 a synthesized mesh is generated based on the plurality of meshes. In step S1125 a left-eye 3D image and depth is generated based on the synthesized mesh and the viewpoint

perspective and/or head pose. In step S1130 a right-eye 3D image and depth map is generated based on the synthesized mesh and the viewpoint perspective and/or head pose. In step S1135 the left-eye 3D image and depth map and the right-eye 3D image and depth map as the streaming 3D video are streamed to the playback device. Here, a single 2D image can represent a single frame. The term "frame" can be understood as a single image that, when played in sequence with the other frames of the video, creates motion on the playback surface. A mesh of the plurality of meshes can be generated for each one of the plurality of 2D images. The step of generating S1120 can refer to synthesizing or fusing the plurality of 2D images into a 3D representation of a scene. The viewpoint perspective can be the perspective of a user receiving the streaming of the 3D images on a 3D playback device.

[0091] Example 2. The method of Example 1, wherein the head pose can be based on a viewpoint perspective of the user of the second client device (e.g., the user of the device receiving the frame of the streaming 3D video) and the plurality of 2D images can have a different viewpoint perspective as compared to the viewpoint perspective of the user of the second client device.

[0092] Example 3. The method of Example 1, wherein the generating of the plurality of meshes can includes downsampling the plurality of 2D images to generate a plurality of feature maps corresponding to one of the plurality of 2D images and generating the plurality of meshes based on the plurality of feature maps.

[0093] Example 4. The method of Example 3, wherein the generating of the left-eye 3D image and depth map and the generating of the right-eye 3D image and depth map can include synthesizing the plurality of feature maps as a feature layered mesh, upsampling the feature layered mesh as a layered mesh, and generating the left-eye 3D image and depth map and generating the right-eye 3D image and depth map based on the layered mesh.

[0094] Example 5. The method of Example 4, wherein the synthesizing of the plurality of feature maps can include initializing the plurality of feature maps to have a flat geometry and projecting the plurality of feature maps to generate a plane sweep volume (PSV).

[0095] Example 6. The method of Example 4, wherein the feature layered mesh can include a plurality of channels, a first subset of the plurality of channels can include abstract network features, and a second subset of the plurality of channels can include depth and density information.

[0096] Example 7. The method of Example 4, wherein the synthesizing of the plurality of feature maps can include generating visibility components to identify occlusions and cross-layer dependencies.

[0097] Example 8. The method of Example 4, wherein the synthesizing of the plurality of feature maps can include projecting the feature layered mesh onto at least one of the plurality of feature maps to determine whether (e.g., a quality, some criteria, how closely, a comparison of the two, a feature match, a quantity of features match, and/or the like) the feature layered mesh approximates at least one of the plurality of 2D images. Alternatively, or additionally, the synthesizing of the plurality of feature maps can include projecting the feature layered mesh onto at least one of the plurality of feature maps and comparing to at least one of the plurality of 2D images to determine an approximation of the feature layered mesh to the at least one of the plurality of 2D images. Alternatively, or additionally, the synthesizing of the plurality of feature maps can include projecting the feature layered mesh onto at least one of the plurality of feature maps and comparing the result to at least one of the plurality of 2D images to determine whether a difference between the feature layered mesh to the at least one of the plurality of 2D images meets a criteria. The criteria can include a per pixel delta threshold, a region of pixels average delta threshold, an object pixel delta threshold, a total loss threshold, a peak signal-to-noise ratio (PSNR), and the like.

[0098] Example 9. The method of Example 1 generating of the plurality of meshes can include downsampling the plurality of 2D images to generate a plurality of feature maps corresponding to one of the plurality of 2D images, generating the plurality of meshes based on the plurality of feature maps, and applying an image matte to the plurality of meshes to remove a background associated with the plurality of 2D images to generate a plurality of foreground meshes.

[0099] Example 10. The method of Example 9, wherein the applying of the image matte can include using a single-color random background.

[00100] Example 11. The method of Example 9, wherein the applying of the image matte can include using a random noise background.

[00101] Example 12. The method of Example 1, wherein the head pose of the user can be a first head pose, the method can further include receiving, from the second client device, a second head pose of the user of the second client device, the second head pose being generated later in time than the first head pose, wherein the plurality of meshes corresponding to the plurality of 2D images can be generated based on the

first head pose, the left-eye 3D image and depth map can be generated based on the second head pose, and the right-eye 3D image and depth map can be generated based on the second head pose.

[00102] Example 13. A method can include any combination of one or more of Example 1 to Example 12.

[00103] Example 14. A non-transitory computer-readable storage medium comprising instructions stored thereon that, when executed by at least one processor, are configured to cause a computing system to perform the method of any of Examples 1-13.

[00104] Example 15. An apparatus comprising means for performing the method of any of Examples 1-13.

[00105] Example 16. An apparatus comprising at least one processor and at least one memory including computer program code, the at least one memory and the computer program code configured to, with the at least one processor, cause the apparatus at least to perform the method of any of Examples 1-13.

[00106] Example implementations can include a non-transitory computer-readable storage medium comprising instructions stored thereon that, when executed by at least one processor, are configured to cause a computing system to perform any of the methods described above. Example implementations can include an apparatus including means for performing any of the methods described above. Example implementations can include an apparatus including at least one processor and at least one memory including computer program code, the at least one memory and the computer program code configured to, with the at least one processor, cause the apparatus at least to perform any of the methods described above.

[00107] Various implementations of the systems and techniques described here can be realized in digital electronic circuitry, integrated circuitry, specially designed ASICs (application specific integrated circuits), computer hardware, firmware, software, and/or combinations thereof. These various implementations can include implementation in one or more computer programs that are executable and/or interpretable on a programmable system including at least one programmable processor, which may be special or general purpose, coupled to receive data and instructions from, and to transmit data and instructions to, a storage system, at least one input device, and at least one output device.

[00108] These computer programs (also known as programs, software, software applications or code) include machine instructions for a programmable processor, and can be implemented in a high-level procedural and/or object-oriented programming language, and/or in assembly/machine language. As used herein, the terms "machine-readable medium" "computer-readable medium" refers to any computer program product, apparatus and/or device (e.g., magnetic discs, optical disks, memory, Programmable Logic Devices (PLDs)) used to provide machine instructions and/or data to a programmable processor, including a machine-readable medium that receives machine instructions as a machine-readable signal. The term "machine-readable signal" refers to any signal used to provide machine instructions and/or data to a programmable processor.

[00109] To provide for interaction with a user, the systems and techniques described here can be implemented on a computer having a display device (a LED (light-emitting diode), or OLED (organic LED), or LCD (liquid crystal display) monitor/screen) for displaying information to the user and a keyboard and a pointing device (e.g., a mouse or a trackball) by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback (e.g., visual feedback, auditory feedback, or tactile feedback); and input from the user can be received in any form, including acoustic, speech, or tactile input.

[00110] The systems and techniques described here can be implemented in a computing system that includes a back end component (e.g., as a data server), or that includes a middleware component (e.g., an application server), or that includes a front end component (e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the systems and techniques described here), or any combination of such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication (e.g., a communication network). Examples of communication networks include a local area network ("LAN"), a wide area network ("WAN"), and the Internet.

[00111] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of

computer programs running on the respective computers and having a client-server relationship to each other.

[00112] A number of embodiments have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the specification.

[00113] In addition, the logic flows depicted in the figures do not require the particular order shown, or sequential order, to achieve desirable results. In addition, other steps may be provided, or steps may be eliminated, from the described flows, and other components may be added to, or removed from, the described systems. Accordingly, other embodiments are within the scope of the following claims.

[00114] While certain features of the described implementations have been illustrated as described herein, many modifications, substitutions, changes and equivalents will now occur to those skilled in the art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes as fall within the scope of the implementations. It should be understood that they have been presented by way of example only, not limitation, and various changes in form and details may be made. Any portion of the apparatus and/or methods described herein may be combined in any combination, except mutually exclusive combinations. The implementations described herein can include various combinations and/or sub-combinations of the functions, components and/or features of the different implementations described.

[00115] While example embodiments may include various modifications and alternative forms, embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that there is no intent to limit example embodiments to the particular forms disclosed, but on the contrary, example embodiments are to cover all modifications, equivalents, and alternatives falling within the scope of the claims. Like numbers refer to like elements throughout the description of the figures.

[00116] Some of the above example embodiments are described as processes or methods depicted as flowcharts. Although the flowcharts describe the operations as sequential processes, many of the operations may be performed in parallel, concurrently or simultaneously. In addition, the order of operations may be re-arranged. The processes may be terminated when their operations are completed, but may also have

additional steps not included in the figure. The processes may correspond to methods, functions, procedures, subroutines, subprograms, etc.

[00117] Methods discussed above, some of which are illustrated by the flow charts, may be implemented by hardware, software, firmware, middleware, microcode, hardware description languages, or any combination thereof. When implemented in software, firmware, middleware or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine or computer readable medium such as a storage medium. A processor(s) may perform the necessary tasks.

[00118] Specific structural and functional details disclosed herein are merely representative for purposes of describing example embodiments. Example embodiments, however, be embodied in many alternate forms and should not be construed as limited to only the embodiments set forth herein.

[00119] It will be understood that, although the terms first, second, etc. may be used herein to describe various elements, these elements should not be limited by these terms. These terms are only used to distinguish one element from another. For example, a first element could be termed a second element, and, similarly, a second element could be termed a first element, without departing from the scope of example embodiments. As used herein, the term and/or includes any and all combinations of one or more of the associated listed items.

[00120] It will be understood that when an element is referred to as being connected or coupled to another element, it can be directly connected or coupled to the other element or intervening elements may be present. In contrast, when an element is referred to as being directly connected or directly coupled to another element, there are no intervening elements present. Other words used to describe the relationship between elements should be interpreted in a like fashion (*e.g.*, between versus directly between, adjacent versus directly adjacent, etc.).

[00121] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting of example embodiments. As used herein, the singular forms a, an and the are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms comprises, comprising, includes and/or including, when used herein, specify the presence of stated features, integers, steps, operations, elements and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components and/or groups thereof.

[00122] It should also be noted that in some alternative implementations, the functions/acts noted may occur out of the order noted in the figures. For example, two figures shown in succession may in fact be executed concurrently or may sometimes be executed in the reverse order, depending upon the functionality/acts involved.

[00123] Unless otherwise defined, all terms (including technical and scientific terms) used herein have the same meaning as commonly understood by one of ordinary skill in the art to which example embodiments belong. It will be further understood that terms, e.g., those defined in commonly used dictionaries, should be interpreted as having a meaning that is consistent with their meaning in the context of the relevant art and will not be interpreted in an idealized or overly formal sense unless expressly so defined herein.

[00124] Portions of the above example embodiments and corresponding detailed description are presented in terms of software, or algorithms and symbolic representations of operation on data bits within a computer memory. These descriptions and representations are the ones by which those of ordinary skill in the art effectively convey the substance of their work to others of ordinary skill in the art. An algorithm, as the term is used here, and as it is used generally, is conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of optical, electrical, or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[00125] In the above illustrative embodiments, reference to acts and symbolic representations of operations (e.g., in the form of flowcharts) that may be implemented as program modules or functional processes include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types and may be described and/or implemented using existing hardware at existing structural elements. Such existing hardware may include one or more Central Processing Units (CPUs), digital signal processors (DSPs), application-specific-integrated-circuits, field programmable gate arrays (FPGAs) computers or the like.

[00126] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient

labels applied to these quantities. Unless specifically stated otherwise, or as is apparent from the discussion, terms such as processing or computing or calculating or determining of displaying or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical, electronic quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[00127] Note also that the software implemented aspects of the example embodiments are typically encoded on some form of non-transitory program storage medium or implemented over some type of transmission medium. The program storage medium may be magnetic (e.g., a floppy disk or a hard drive) or optical (e.g., a compact disk read only memory, or CD ROM), and may be read only or random access. Similarly, the transmission medium may be twisted wire pairs, coaxial cable, optical fiber, or some other suitable transmission medium known to the art. The example embodiments not limited by these aspects of any given implementation.

WHAT IS CLAIMED IS:

1. A method comprising:

receiving, from a first client device, a plurality of two-dimensional (2D) images representing a frame of a streaming three-dimensional (3D) video;

receiving, from a second client device, a head pose of a user of the second client device;

generating a plurality of meshes corresponding to the plurality of 2D images and based on the head pose;

generating a synthesized mesh based on the plurality of meshes;

generating a left-eye 3D image and depth based on the synthesized mesh and the head pose;

generating a right-eye 3D image and depth map based on the synthesized mesh and the head pose; and

streaming, to the second client device, the right-eye 3D image and depth map and the left-eye 3D image and depth map as the frame of the streaming 3D video.

2. The method of claim 1, wherein

the head pose is based on a viewpoint perspective of the user of the second client device, and

the plurality of 2D images has a different viewpoint perspective as compared to the viewpoint perspective of the user of the second client device.

3. The method of claim 1 or claim 2, wherein the generating of the plurality of meshes includes:

downsampling the plurality of 2D images to generate a plurality of feature maps corresponding to one of the plurality of 2D images, and

generating the plurality of meshes based on the plurality of feature maps.

4. The method of claim 3, wherein the generating of the left-eye 3D image and depth map and the generating of the right-eye 3D image and depth map include:

synthesizing the plurality of feature maps as a feature layered mesh,

upsampling the feature layered mesh as a layered mesh, and

generating the left-eye 3D image and depth map and generating the right-eye 3D image and depth map based on the layered mesh.

5. The method of claim 4, wherein the synthesizing of the plurality of feature maps includes initializing the plurality of feature maps to have a flat geometry and projecting the plurality of feature maps to generate a plane sweep volume (PSV).

6. The method of claim 4 or claim 5, wherein

the feature layered mesh includes a plurality of channels,

a first subset of the plurality of channels include abstract network features, and

a second subset of the plurality of channels include depth and density information.

7. The method of any of claims 4 to 6, wherein the synthesizing of the plurality of feature maps includes generating visibility components to identify occlusions and cross-layer dependencies.

8. The method of any of claims 4 to 7, wherein the synthesizing of the plurality of feature maps includes projecting the feature layered mesh onto at least one of the plurality of feature maps to determine whether the feature layered mesh approximates at least one of the plurality of 2D images.

9. The method of any of claims 1 to 8, wherein the generating of the plurality of meshes includes:

downsampling the plurality of 2D images to generate a plurality of feature maps corresponding to one of the plurality of 2D images,

generating the plurality of meshes based on the plurality of feature maps, and

applying an image matte to the plurality of meshes to remove a background associated with the plurality of 2D images to generate a plurality of foreground meshes.

10. The method of claim 9, wherein the applying of the image matte includes using a single-color random background.

11. The method of claim 9 or claim 10, wherein the applying of the image matte includes using a random noise background.

12. The method of any of claims 1 to 11, wherein head pose of the user is a first head pose, the method further comprising:

receiving, from the second client device, a second head pose of the user of the second client device, the second head pose being generated later in time than the first head pose, wherein:

the plurality of meshes corresponding to the plurality of 2D images is generated based on the first head pose,

the left-eye 3D image and depth map is generated based on the second head pose, and

the right-eye 3D image and depth map is generated based on the second head pose.

13. A non-transitory computer-readable storage medium comprising instructions stored thereon that, when executed by at least one processor, are configured to cause a computing system to perform the method of any of claims 1-12.

14. An apparatus comprising means for performing the method of any of claims 1-12.

15. An apparatus comprising:

at least one processor; and

at least one memory including computer program code;

the at least one memory and the computer program code configured to, with the at least one processor, cause the apparatus at least to perform the method of any of claims 1-12.

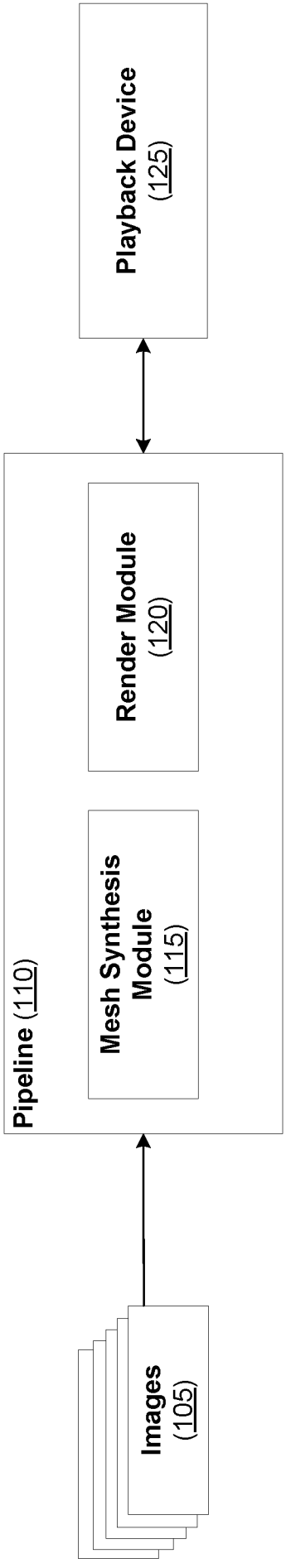
Playback Device
(125)
Pipeline (110)
Render Module
(120)
Mesh Synthesis
Module
(115)
Images
(105)

FIG. 1

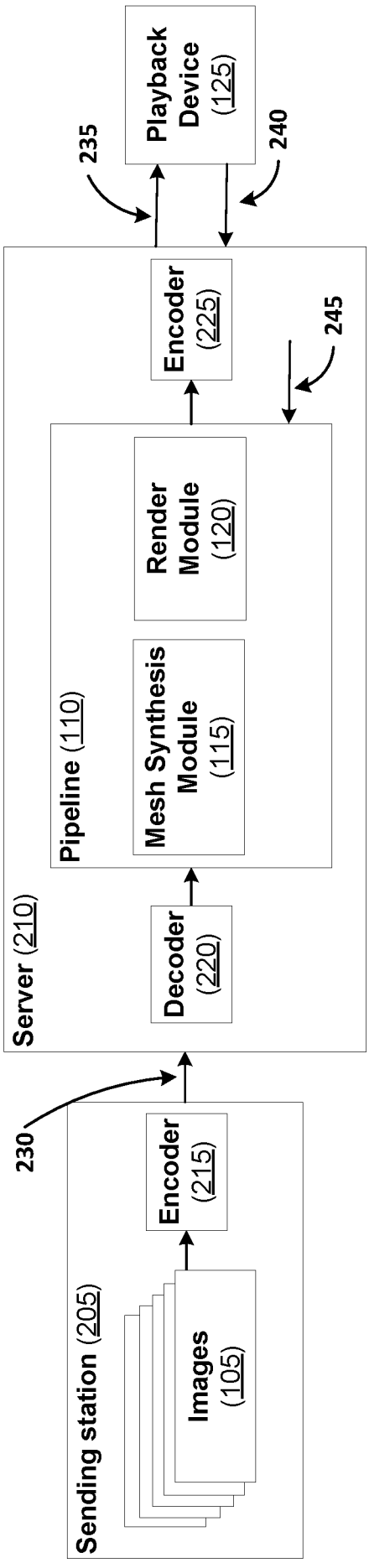
Sending station (205)
Images (105)
Encoder (215)
230
Server (210)
Decoder (220)
Pipeline (110)
Mesh Synthesis Module (115)
Render Module (120)
Encoder (225)
245
235
240
Playback Device (125)

FIG. 2

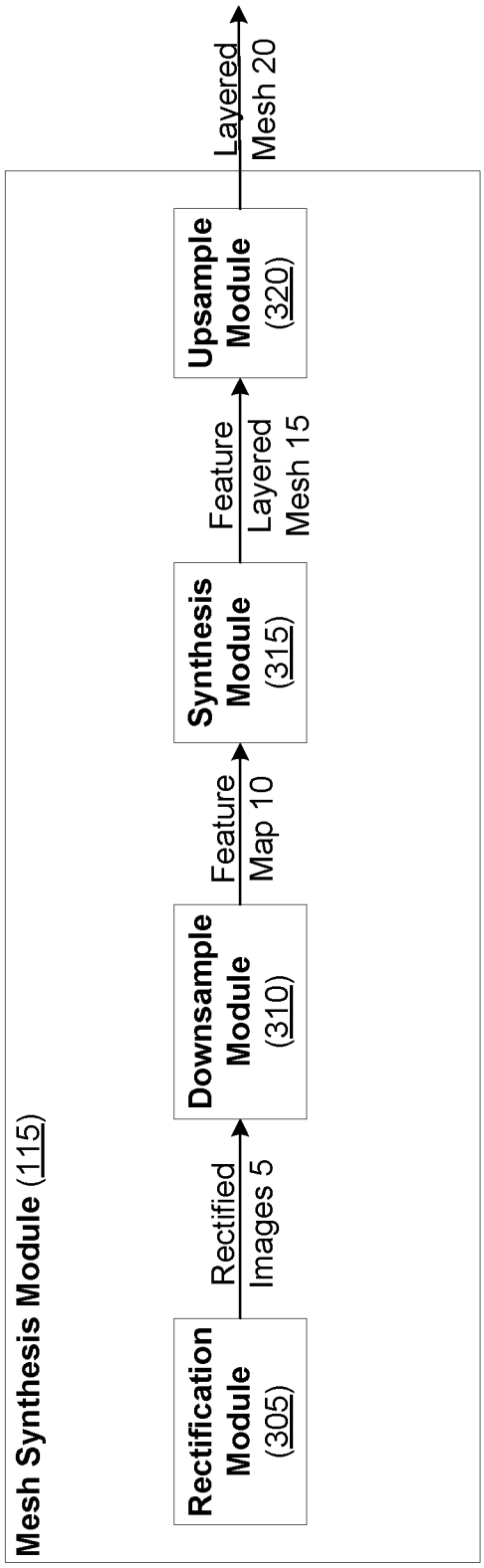
Mesh Synthesis Module (115)
Rectification Module (305)
Rectified Images 5
Downsample Module (310)
Feature Map 10
Synthesis Module (315)
Feature Layered Mesh 15
Upsample Module (320)
Layered Mesh 20

FIG. 3

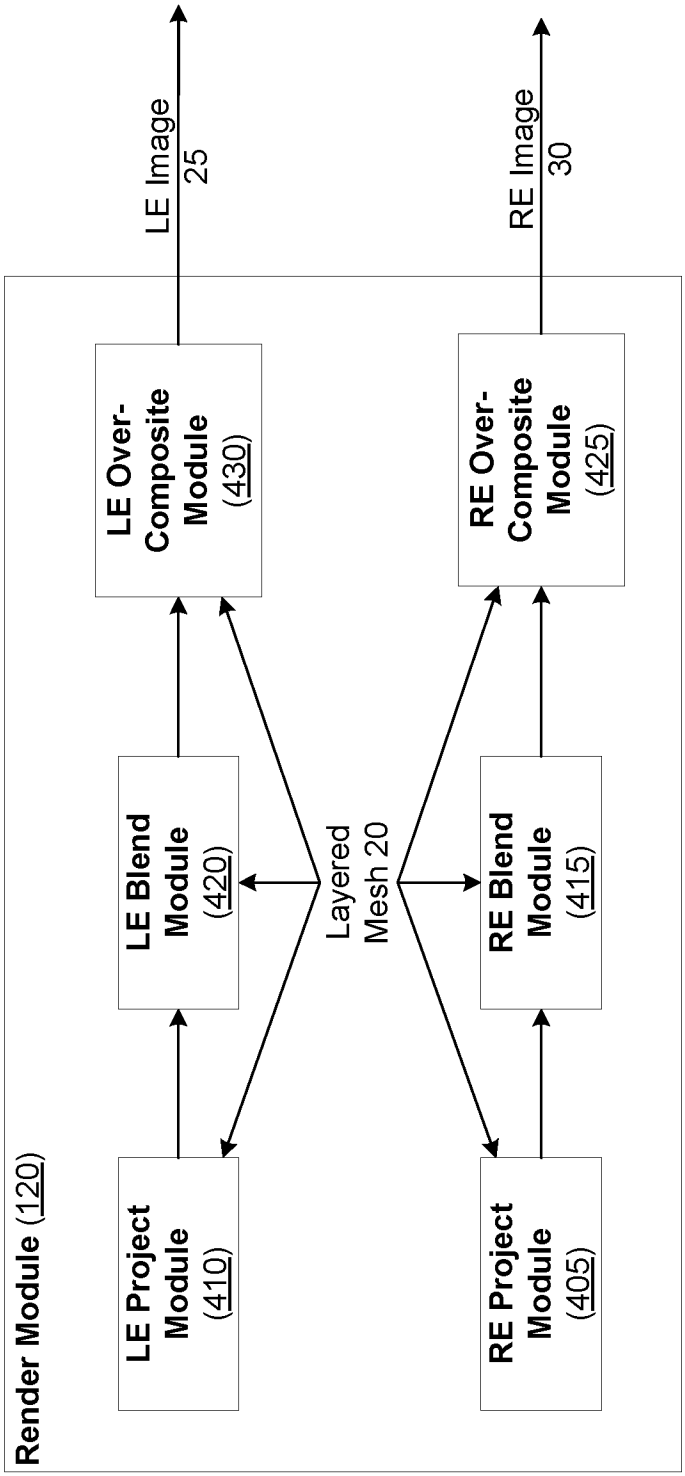
Render Module (120)
LE Project Module (410)
LE Blend Module (420)
LE Over-Composite Module (430)
Layered Mesh 20
RE Project Module (405)
RE Blend Module (415)
RE Over-Composite Module (425)
LE Image 25
RE Image 30

FIG. 4

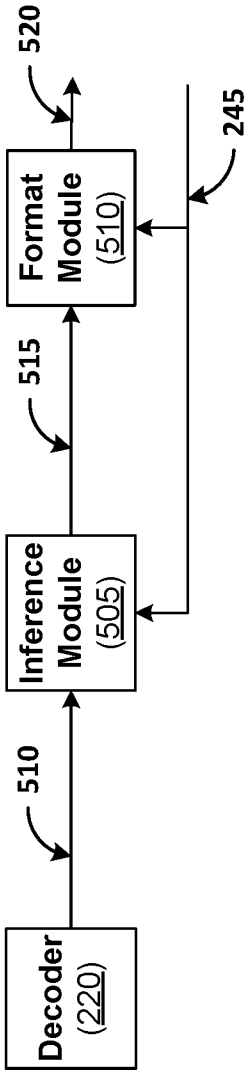

FIG. 5

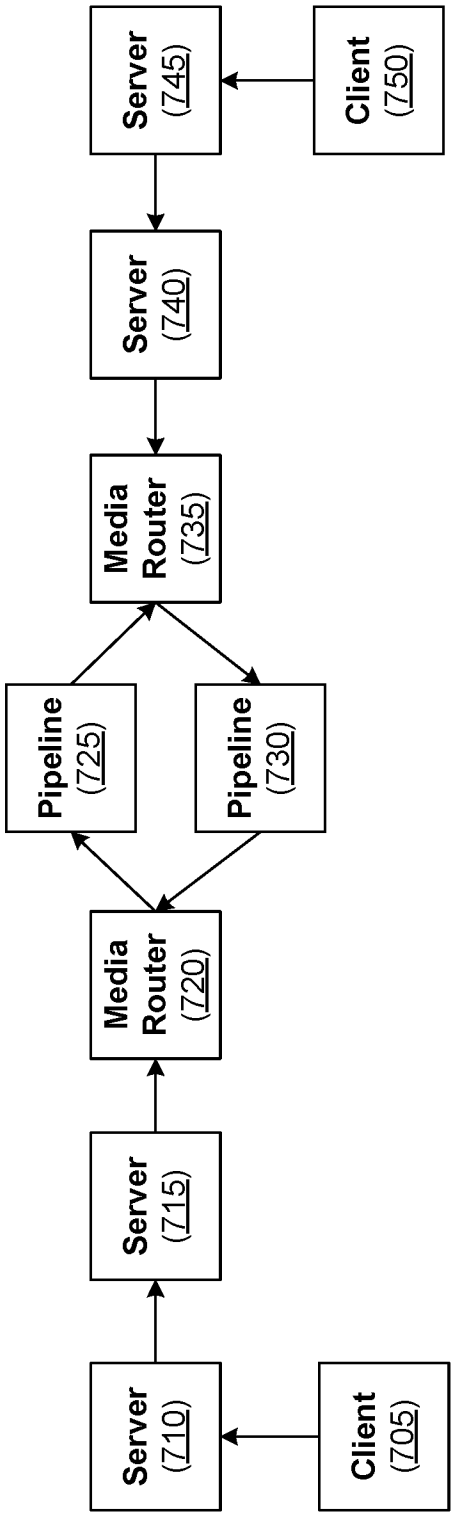

FIG. 7

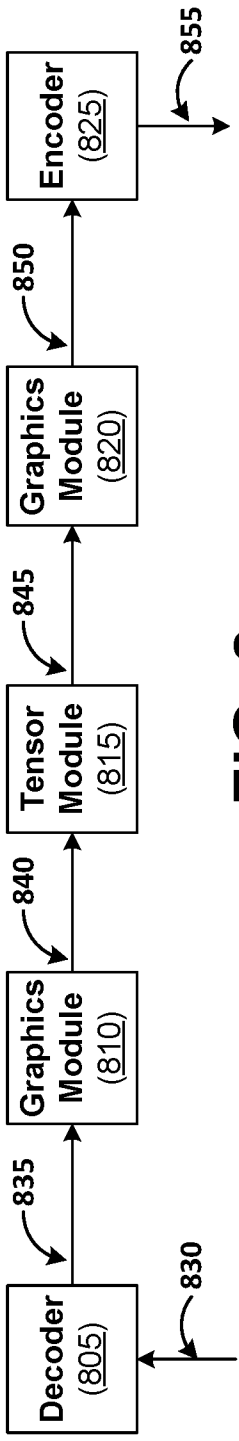

FIG. 8

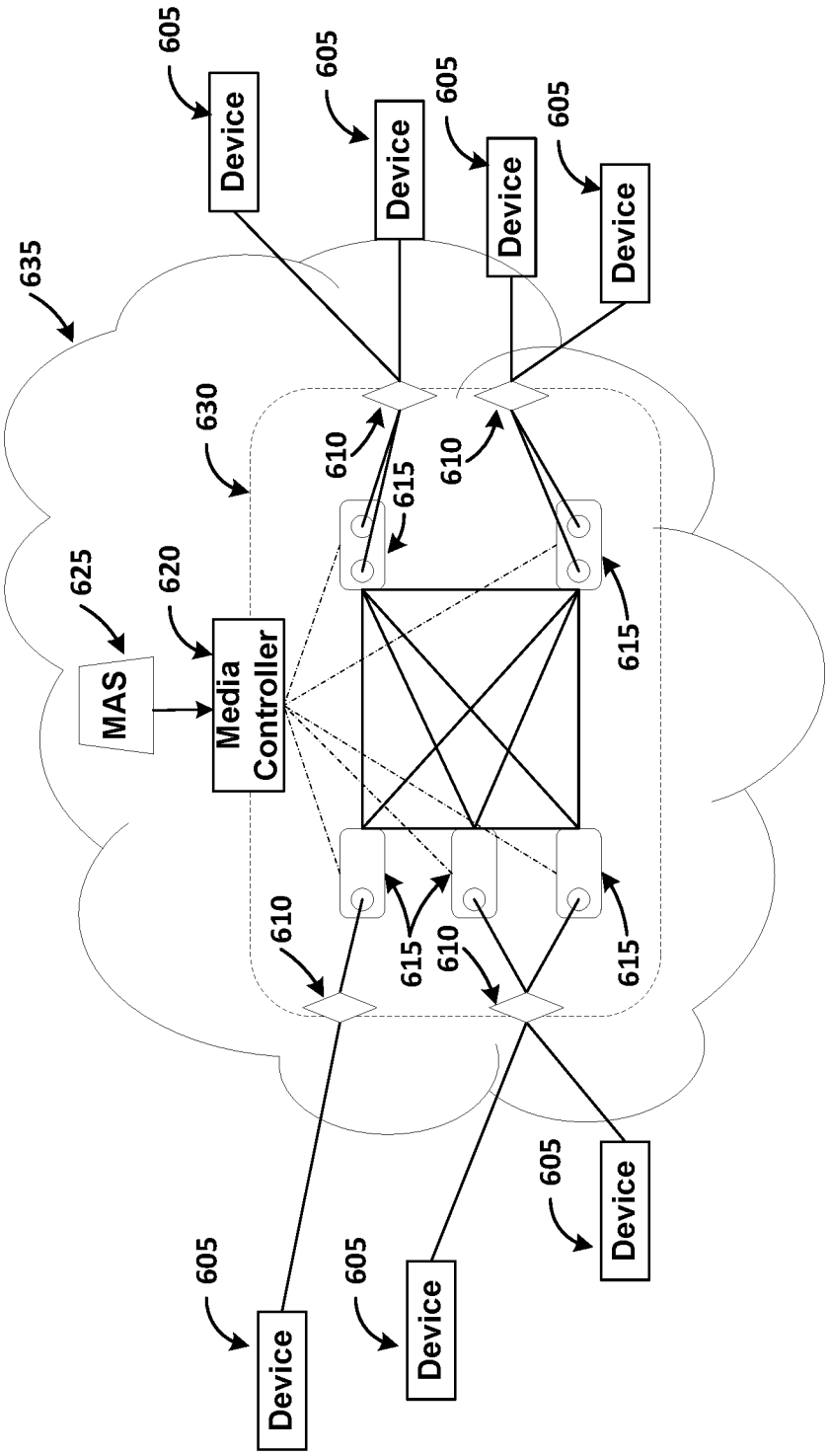
605
Device
605
Device
605
Device
605
Device
635
630
610
615
610
615
625
620
MAS
Media Controller
615
610
615
610
615
605
Device
605
Device
605
Device

FIG. 6

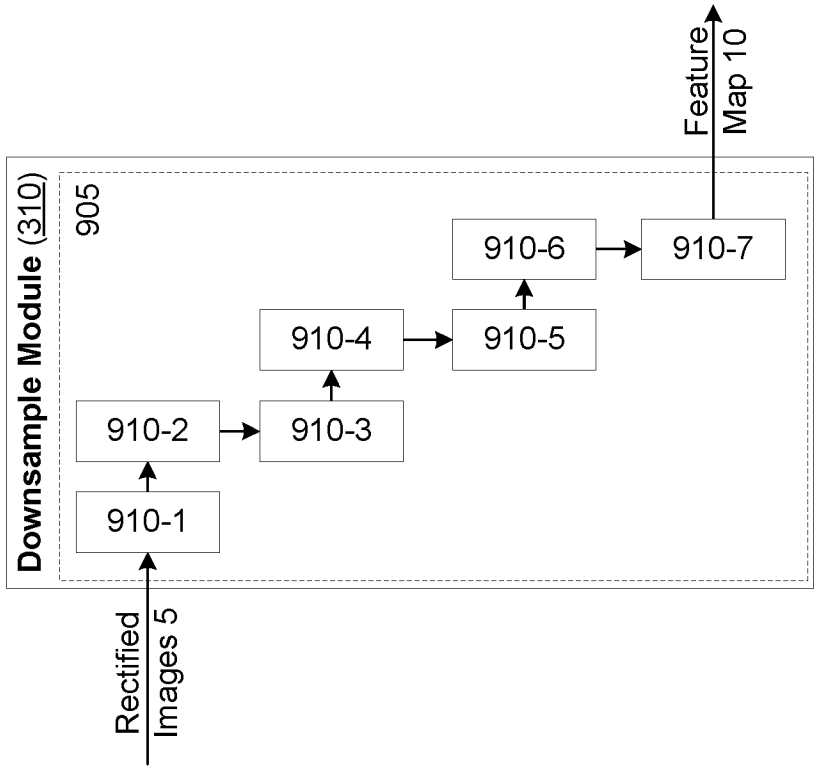
Feature Map 10
Downsample Module (310)
905
910-6
910-7
910-4
910-5
910-2
910-3
910-1
Rectified Images 5

FIG. 9A

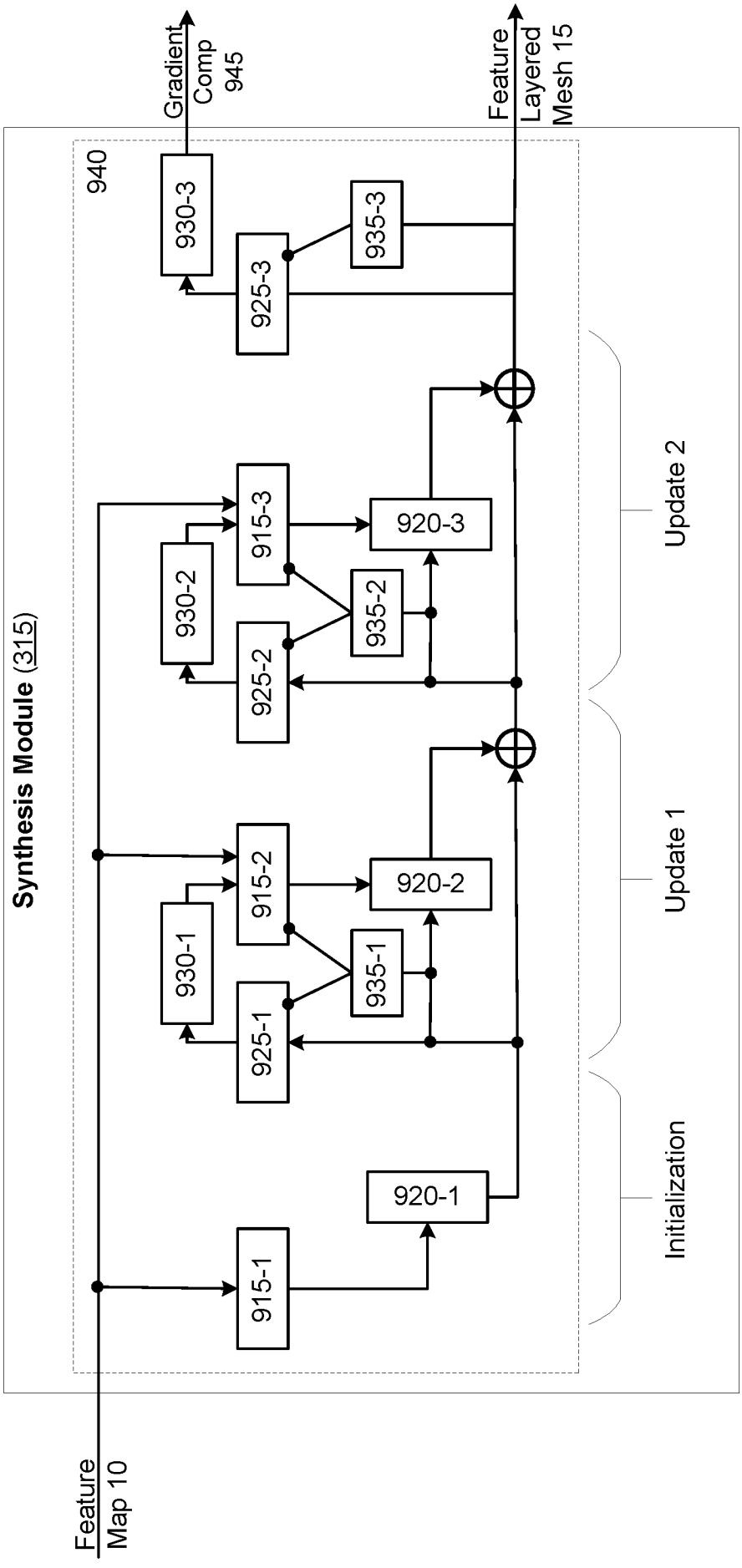
Feature Map 10
Synthesis Module (315)
940
915-1
915-2
915-3
920-1
920-2
920-3
925-1
925-2
925-3
930-1
930-2
930-3
935-1
935-2
935-3
Gradient Comp 945
Feature Layered Mesh 15
Initialization
Update 1
Update 2

FIG. 9B

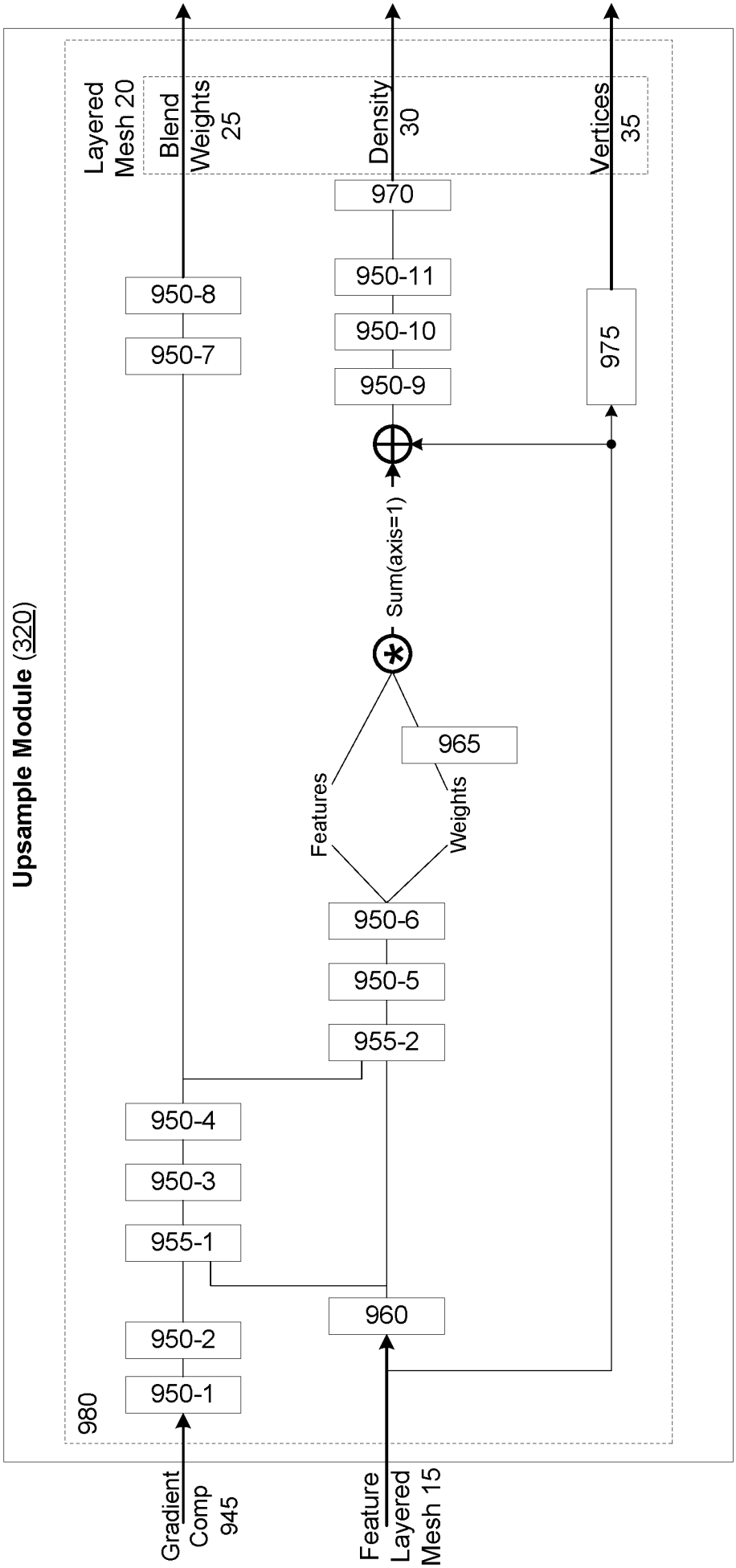
Upsample Module (320)
980
Layered Mesh 20
Blend Weights 25
Density 30
Vertices 35
Gradient Comp 945
Feature Layered Mesh 15
950-1
950-2
950-3
950-4
950-5
950-6
950-7
950-8
950-9
950-10
950-11
955-1
955-2
960
965
970
975
Features
Weights
Sum(axis=1)

FIG. 9C

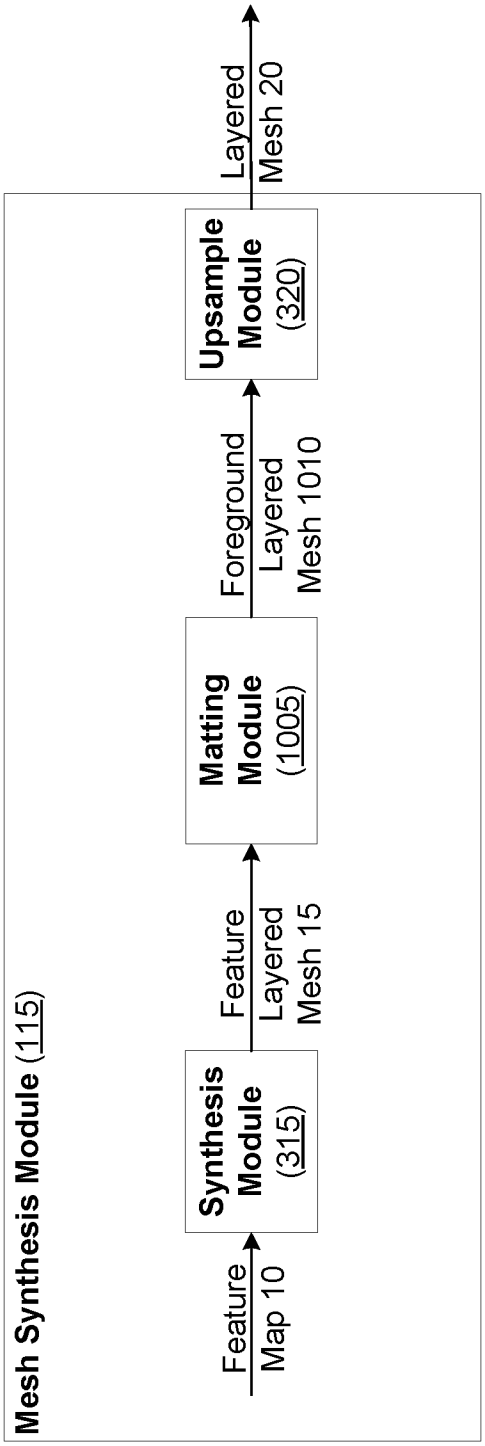
Mesh Synthesis Module (115)
Feature Map 10
Synthesis Module (315)
Feature Layered Mesh 15
Matting Module (1005)
Foreground Layered Mesh 1010
Upsample Module (320)
Layered Mesh 20

FIG. 10

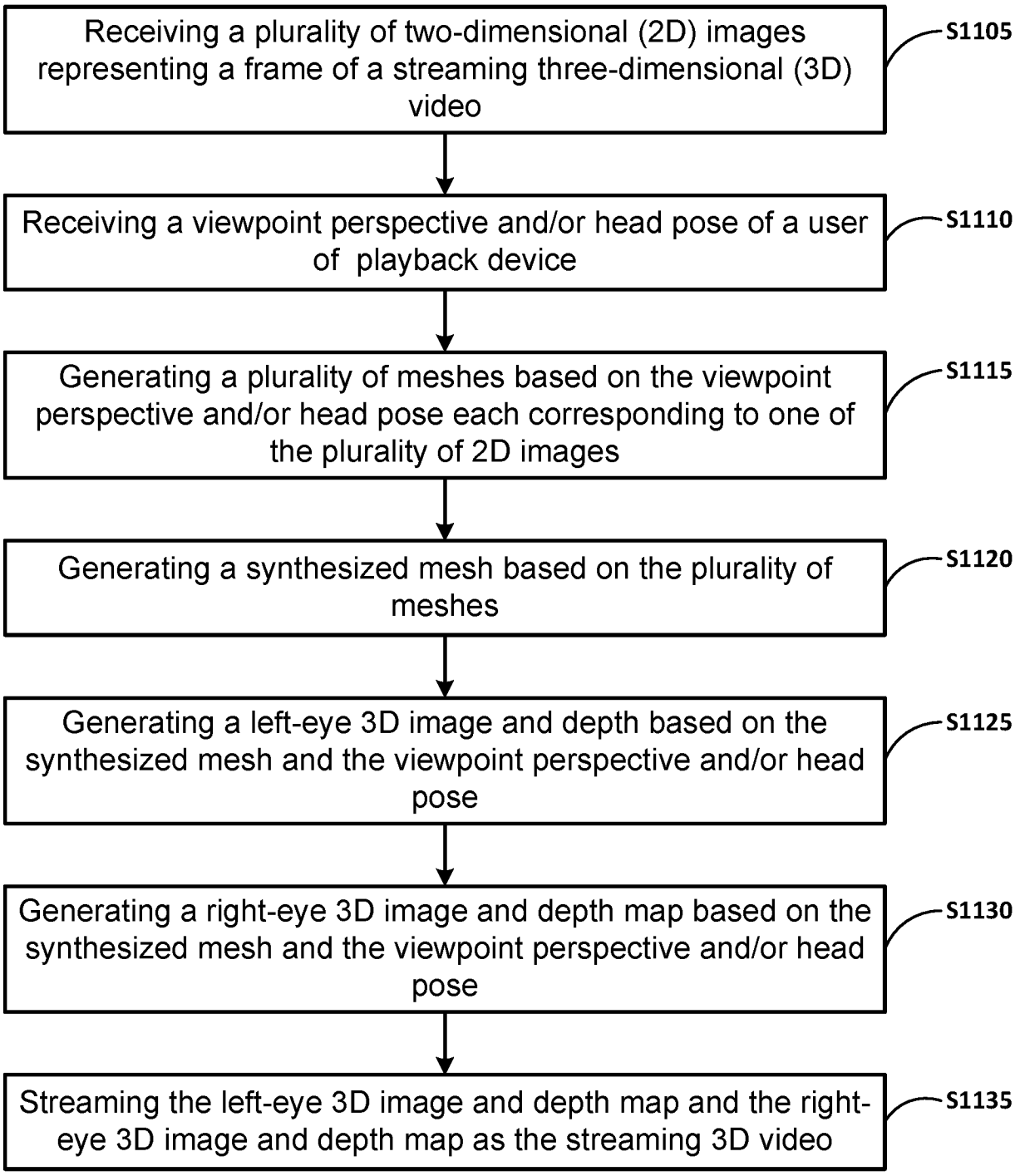
Receiving a plurality of two-dimensional (2D) images representing a frame of a streaming three-dimensional (3D) video
S1105
Receiving a viewpoint perspective and/or head pose of a user of playback device
S1110
Generating a plurality of meshes based on the viewpoint perspective and/or head pose each corresponding to one of the plurality of 2D images
S1115
Generating a synthesized mesh based on the plurality of meshes
S1120
Generating a left-eye 3D image and depth based on the synthesized mesh and the viewpoint perspective and/or head pose
S1125
Generating a right-eye 3D image and depth map based on the synthesized mesh and the viewpoint perspective and/or head pose
S1130
Streaming the left-eye 3D image and depth map and the right-eye 3D image and depth map as the streaming 3D video
S1135

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/083627

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04N13/117 H04N13/194
ADD. H04N13/00 H04N13/139

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)
H04N G06T

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.
Y	US 2021/150802 A1 (VAREKAMP CHRISTIAAN [NL] ET AL) 20 May 2021 (2021-05-20)	1-3,9-15
A	the whole document	4-8
Y	US 2022/130111 A1 (MARTIN BRUALLA RICARDO [US] ET AL) 28 April 2022 (2022-04-28) paragraphs [0039], [0040], [0070], [0111]; figures 1-2	1-3,9-15
Y	US 2017/257609 A1 (CHU DAVID CHIYUAN [US] ET AL) 7 September 2017 (2017-09-07) paragraphs [0003], [0037], [0044], [0050], [0053], [0099], [0101]; figures 1,14,20	12

☐ 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	Date of mailing of the international search report
21 February 2024	11/03/2024

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Mao, Pauline

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2023/083627

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2021150802 A1	20-05-2021	BR 112019027116 A2	07-07-2020
		CN 110999285 A	10-04-2020
		EP 3419286 A1	26-12-2018
		EP 3643059 A1	29-04-2020
		JP 7181233 B2	30-11-2022
		JP 2020524851 A	20-08-2020
		KR 20200020879 A	26-02-2020
		RU 2020102462 A	26-07-2021
		TW 201921921 A	01-06-2019
		US 2021150802 A1	20-05-2021
		WO 2018234258 A1	27-12-2018
US 2022130111 A1	28-04-2022	CN 114631127 A	14-06-2022
		EP 4007992 A1	08-06-2022
		JP 7386888 B2	27-11-2023
		JP 2023513980 A	05-04-2023
		KR 20220047719 A	19-04-2022
		US 2022130111 A1	28-04-2022
		WO 2022076020 A1	14-04-2022
US 2017257609 A1	07-09-2017	US 2016217760 A1	28-07-2016
		US 2017257609 A1	07-09-2017