US 20240265561A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2024/0265561 A1**

Gallo et al. (43) **Pub. Date: Aug. 8, 2024**

(54) **MESH RECONSTRUCTION USING DATA-DRIVEN PRIORS**

(71) Applicant: **NVIDIA Corporation**, Santa Clara, CA (US)

(72) Inventors: **Orazio Gallo**, Santa Cruz, CA (US); **Abhishek Badki**, Goleta, CA (US)

(21) Appl. No.: **18/638,346**

(22) Filed: **Apr. 17, 2024**

Related U.S. Application Data

(63) Continuation of application No. 16/226,329, filed on Dec. 19, 2018, now Pat. No. 11,995,854.

Publication Classification

(51) **Int. Cl.**

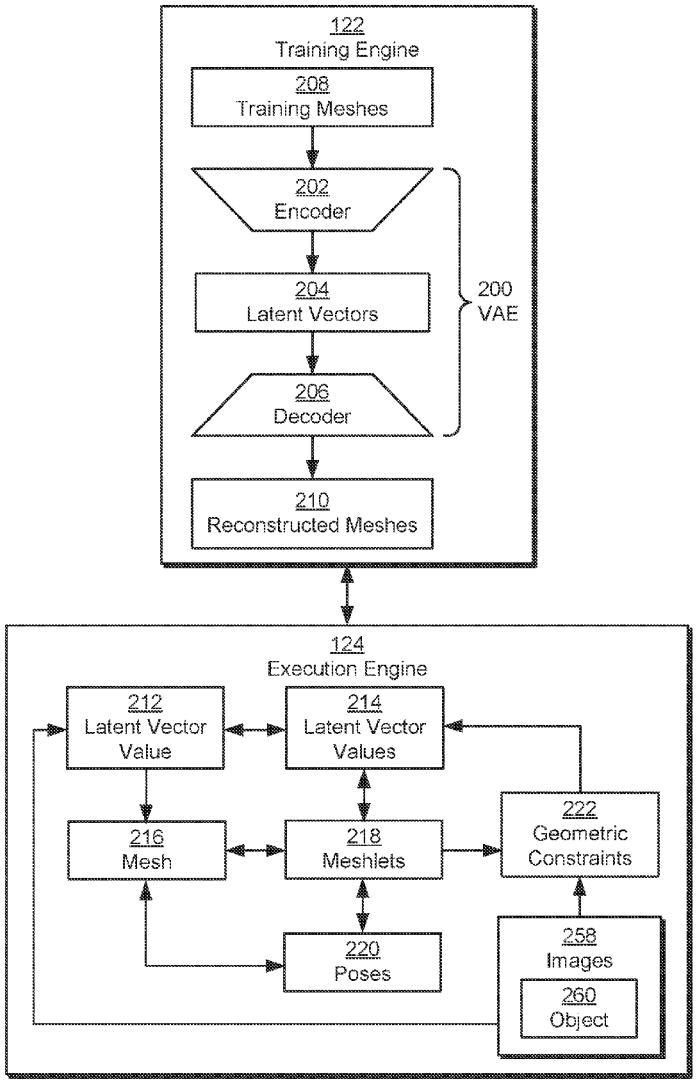
G06T 7/55	(2006.01)
G06F 17/16	(2006.01)
G06F 18/214	(2006.01)
G06N 20/00	(2006.01)
G06T 15/10	(2006.01)
G06T 17/20	(2006.01)

(52) **U.S. Cl.**

CPC ***G06T 7/55*** (2017.01); ***G06F 17/16*** (2013.01); ***G06F 18/214*** (2023.01); ***G06N 20/00*** (2019.01); ***G06T 15/10*** (2013.01); ***G06T 17/205*** (2013.01)

(57) **ABSTRACT**

One embodiment of a method includes predicting one or more three-dimensional (3D) mesh representations based on a plurality of digital images, wherein the one or more 3D mesh representations are refined by minimizing at least one difference between the one or more 3D mesh representations and the plurality of digital images.



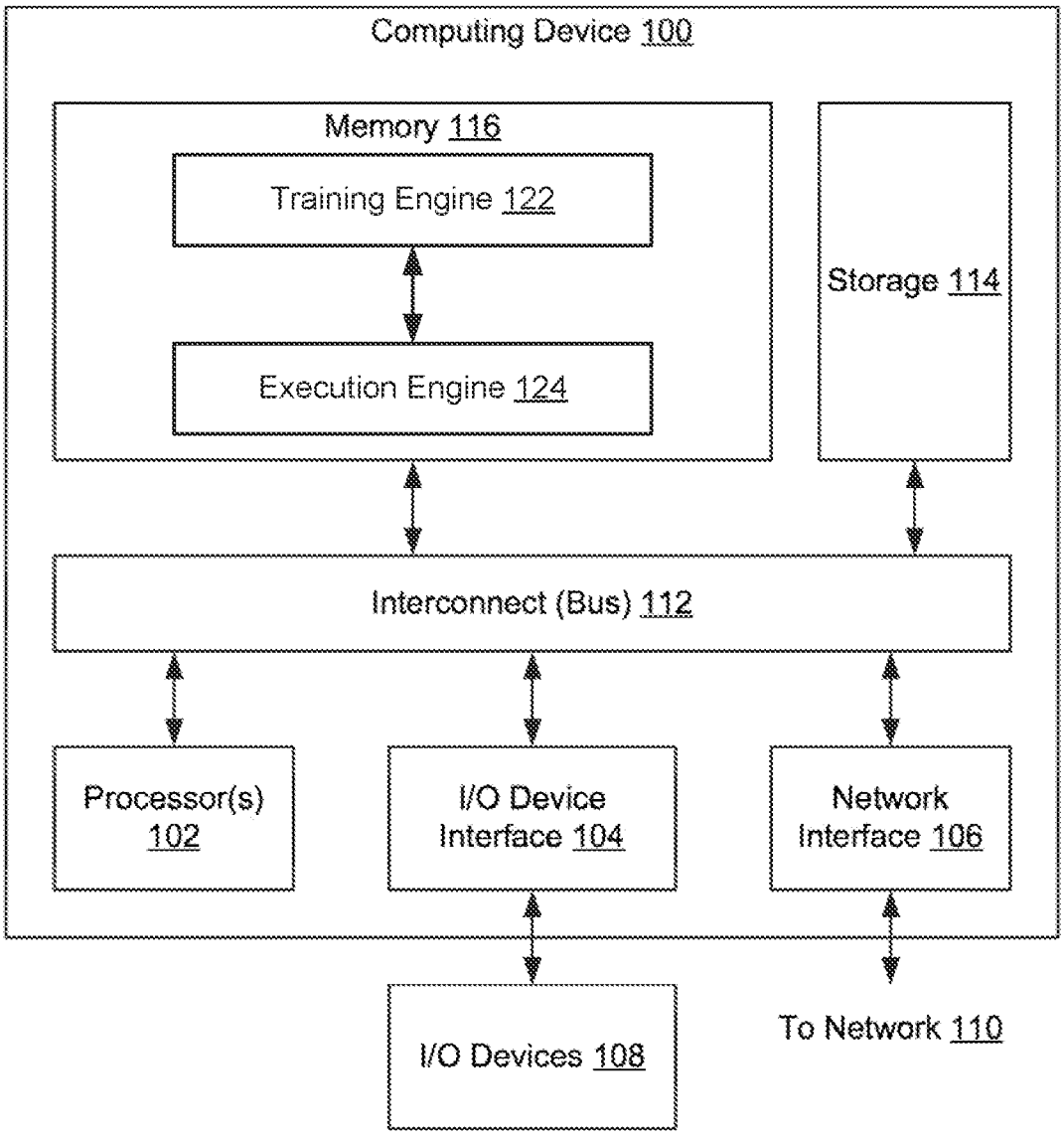

Computing Device 100
Memory 116
Training Engine 122
Execution Engine 124
Storage 114
Interconnect (Bus) 112
Processor(s) 102
I/O Device Interface 104
Network Interface 106
I/O Devices 108
To Network 110

FIG. 1

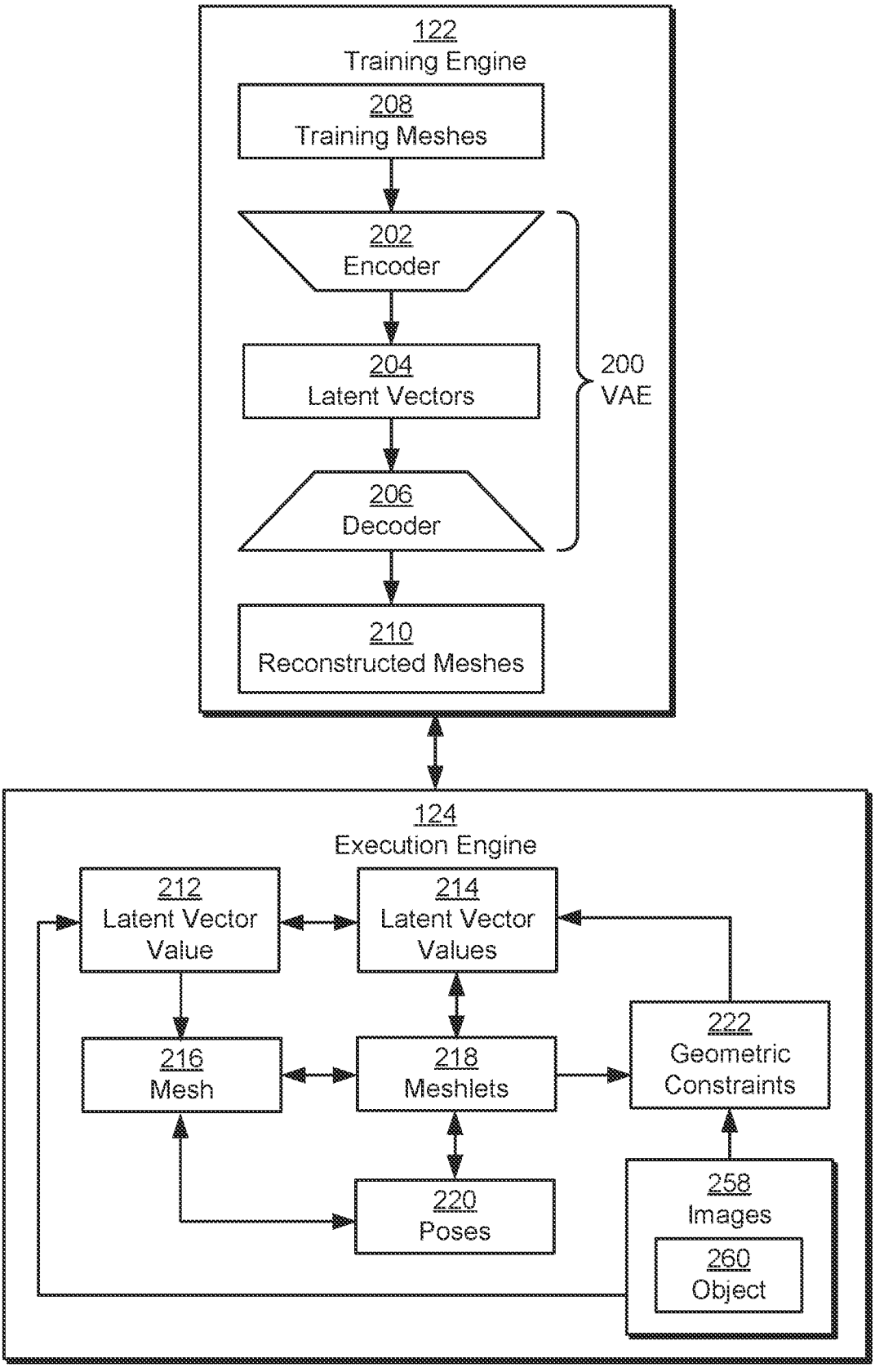

122
Training Engine
208
Training Meshes
202
Encoder
204
Latent Vectors
206
Decoder
200
VAE
210
Reconstructed Meshes
124
Execution Engine
212
Latent Vector
Value
214
Latent Vector
Values
216
Mesh
218
Meshlets
222
Geometric
Constraints
220
Poses
258
Images
260
Object

FIG. 2

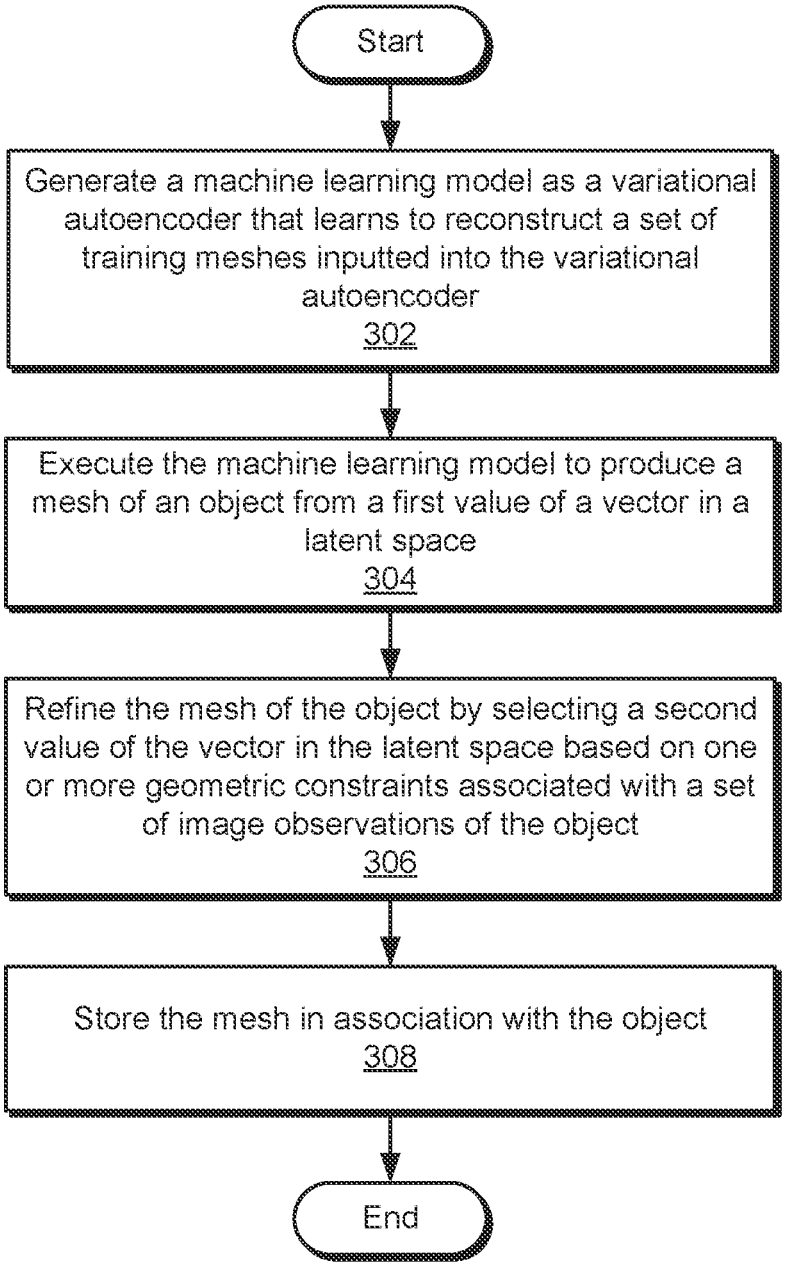

Start
Generate a machine learning model as a variational autoencoder that learns to reconstruct a set of training meshes inputted into the variational autoencoder
302
Execute the machine learning model to produce a mesh of an object from a first value of a vector in a latent space
304
Refine the mesh of the object by selecting a second value of the vector in the latent space based on one or more geometric constraints associated with a set of image observations of the object
306
Store the mesh in association with the object
308
End

FIG. 3

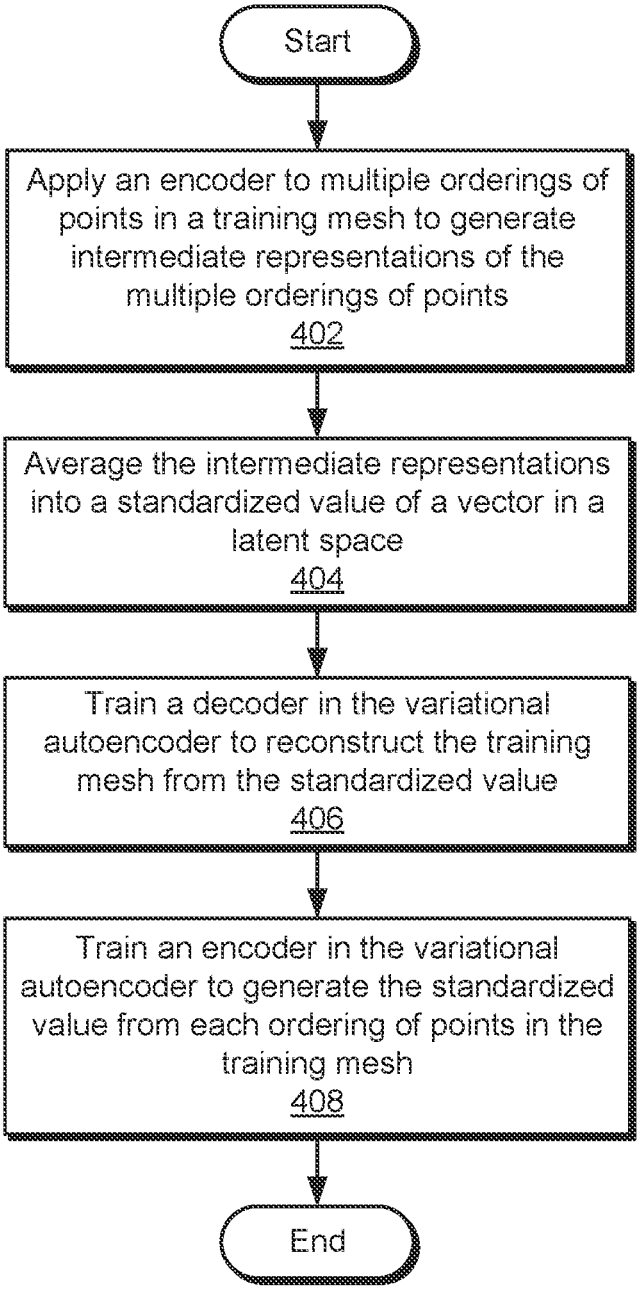

Start
Apply an encoder to multiple orderings of points in a training mesh to generate intermediate representations of the multiple orderings of points
402
Average the intermediate representations into a standardized value of a vector in a latent space
404
Train a decoder in the variational autoencoder to reconstruct the training mesh from the standardized value
406
Train an encoder in the variational autoencoder to generate the standardized value from each ordering of points in the training mesh
408
End

FIG. 4

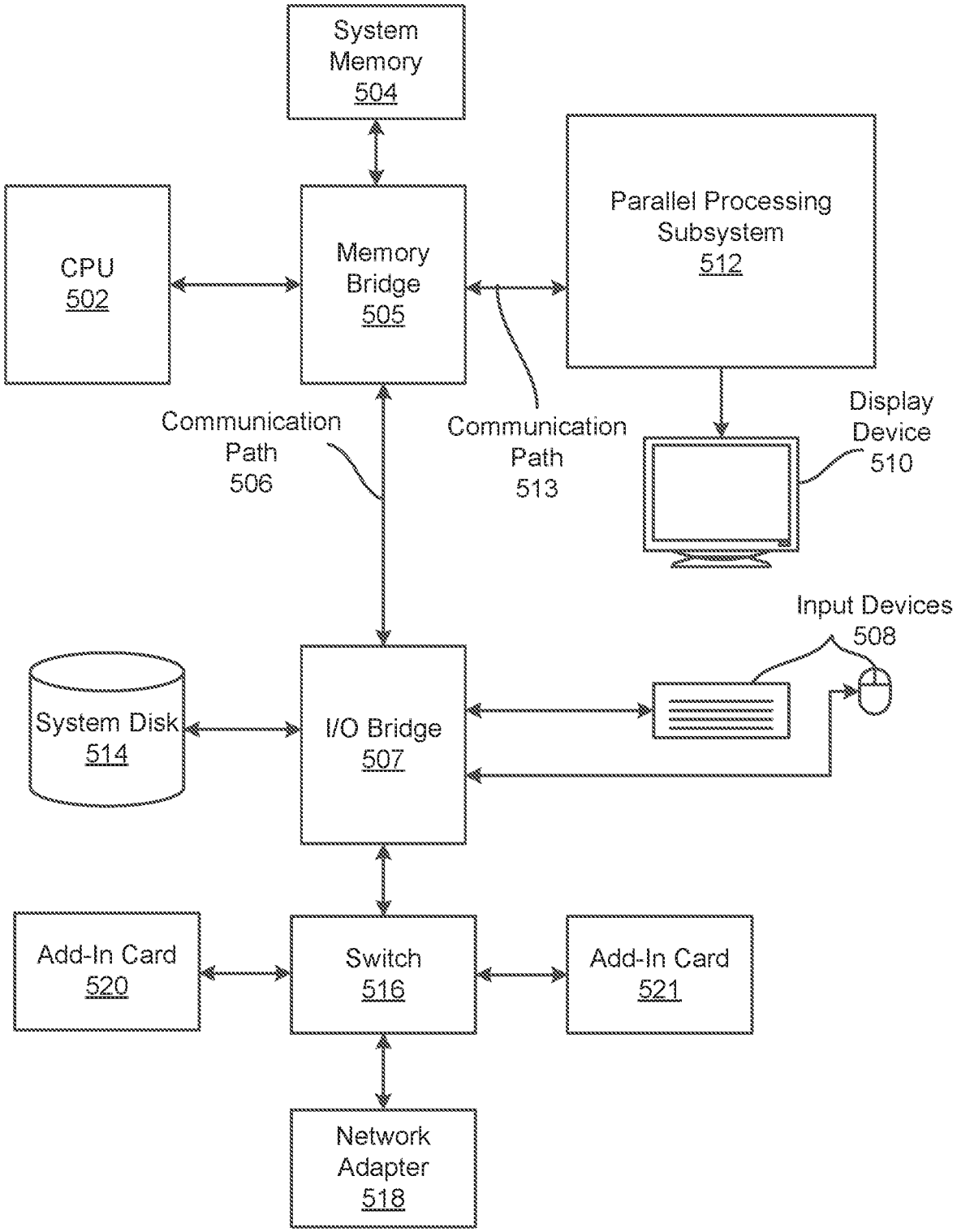

System Memory 504
CPU 502
Memory Bridge 505
Parallel Processing Subsystem 512
Communication Path 506
Communication Path 513
Display Device 510
Input Devices 508
System Disk 514
I/O Bridge 507
Add-In Card 520
Switch 516
Add-In Card 521
Network Adapter 518

FIG. 5

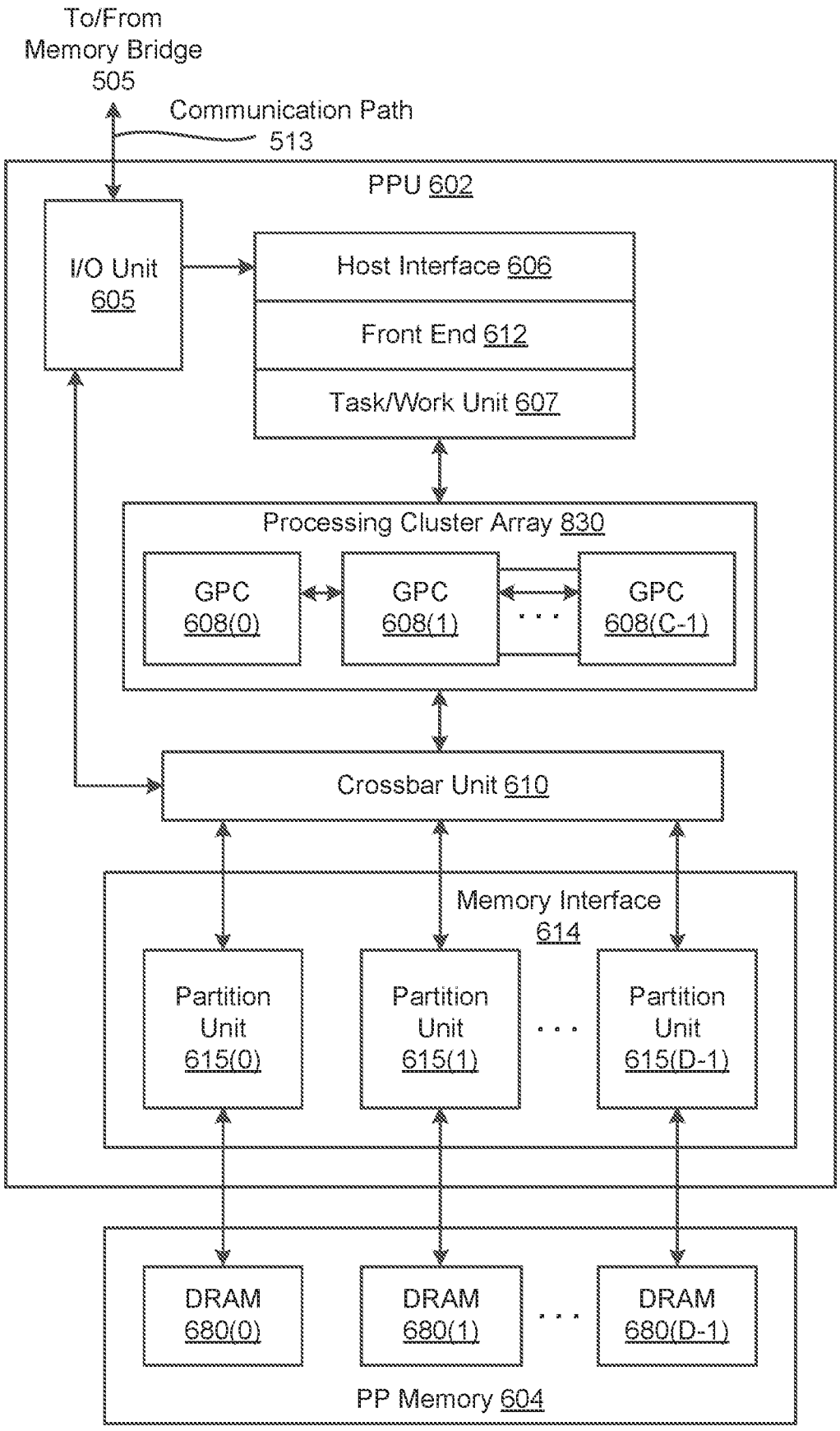

To/From
Memory Bridge
505
Communication Path
513
PPU 602
I/O Unit
605
Host Interface 606
Front End 612
Task/Work Unit 607
Processing Cluster Array 830
GPC
608(0)
GPC
608(1)
GPC
608(C-1)
Crossbar Unit 610
Memory Interface
614
Partition
Unit
615(0)
Partition
Unit
615(1)
Partition
Unit
615(D-1)
DRAM
680(0)
DRAM
680(1)
DRAM
680(D-1)
PP Memory 604

FIG. 6

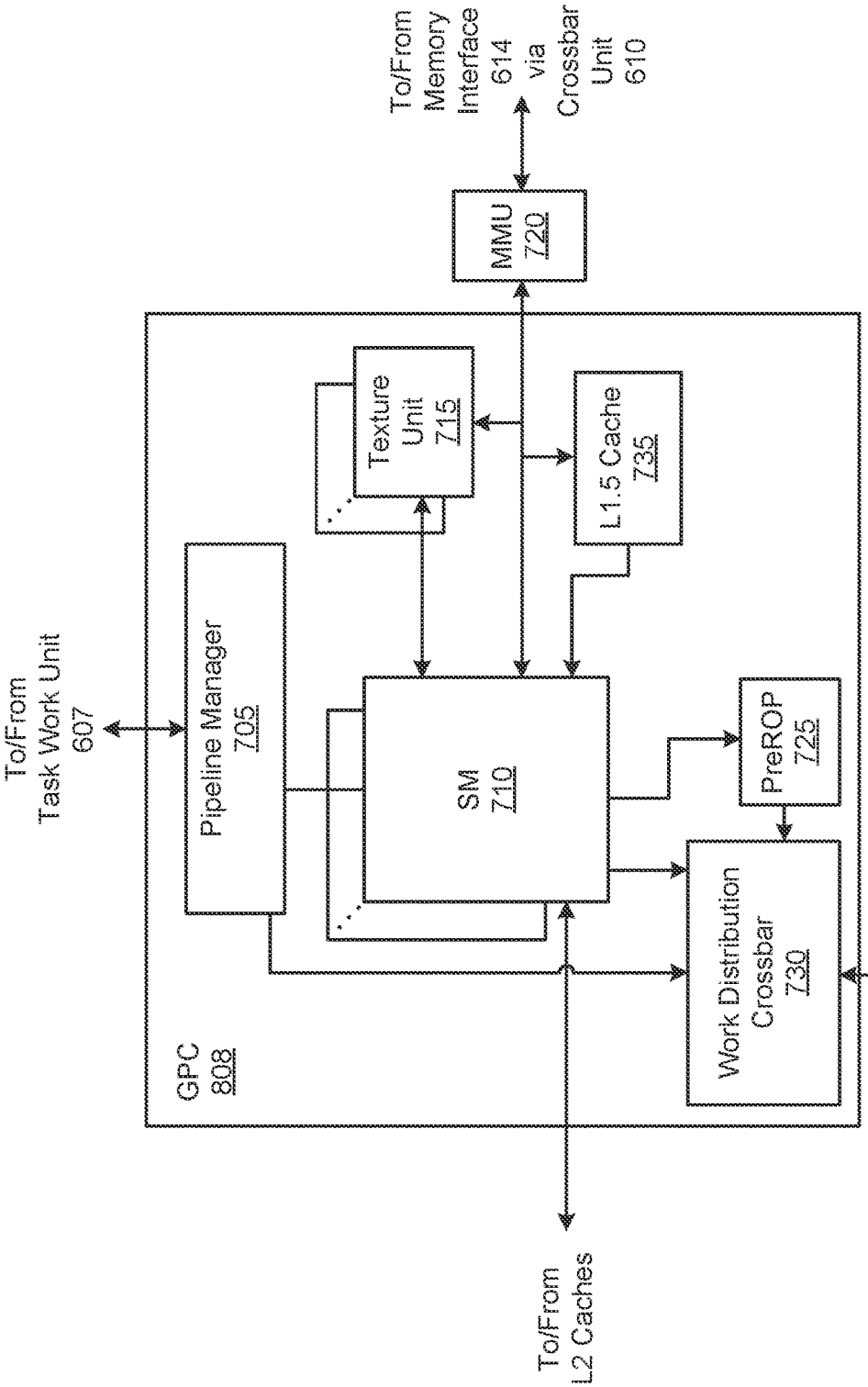

To/From Memory Interface 614 via Crossbar Unit 610
MMU 720
Texture Unit 715
L1.5 Cache 735
To/From Task Work Unit 607
Pipeline Manager 705
SM 710
PreROP 725
Work Distribution Crossbar 730
To Crossbar Unit 610 and GPCs 608
GPC 808
To/From L2 Caches

FIG. 7

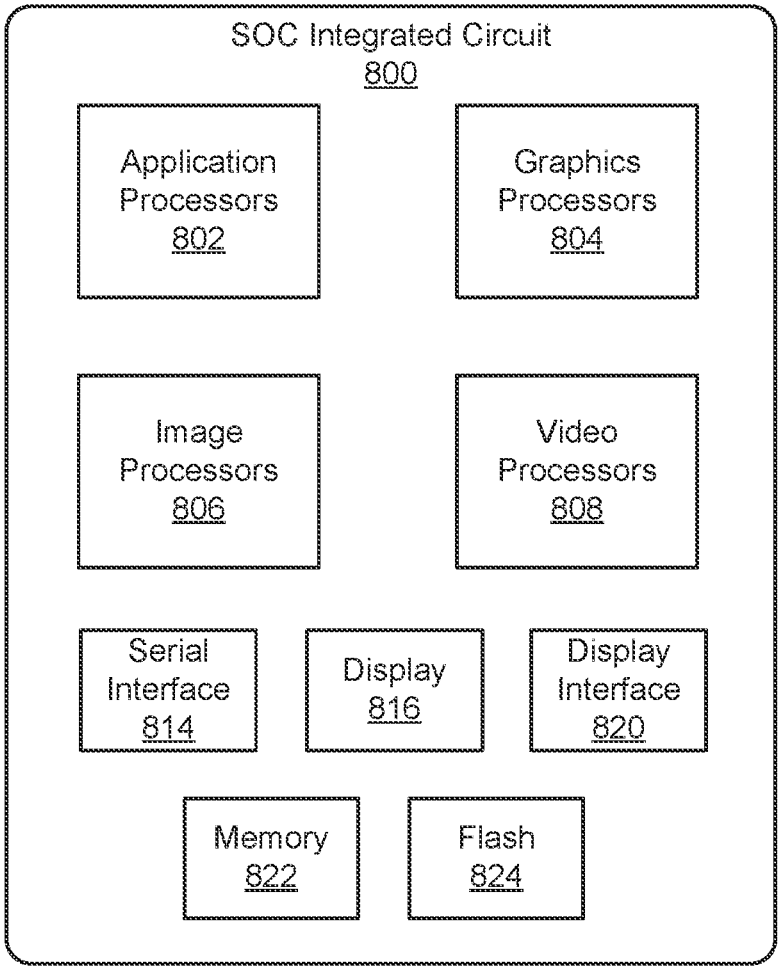

SOC Integrated Circuit
800
Application Processors
802
Graphics Processors
804
Image Processors
806
Video Processors
808
Serial Interface
814
Display
816
Display Interface
820
Memory
822
Flash
824

FIG. 8

MESH RECONSTRUCTION USING DATA-DRIVEN PRIORS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of U.S. application Ser. No. 16/226,329, filed on Dec. 19, 2018, entitled "MESH RECONSTRUCTION USING DATA-DRIVEN PRIORS." The subject matter of this related application is hereby incorporated herein by reference.

BACKGROUND

[0002] Multiple view stereovision (MVS) techniques involve constructing three-dimensional (3D) surfaces from a number of overlapping two-dimensional (2D) images of an object. Such techniques may estimate the most likely 3D shape from the 2D images based on assumptions related to textures, viewpoints, lighting, and/or other conditions under which the images were taken. Given a set of images of an object and corresponding assumptions, MVS uses stereo correspondence among the images to reconstruct the 3D geometry of the scene captured by the images.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] So that the manner in which the above recited features of the various embodiments can be understood in detail, a more particular description of the inventive concepts, briefly summarized above, may be had by reference to various embodiments, some of which are illustrated in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical embodiments of the inventive concepts and are therefore not to be considered limiting of scope in any way, and that there are other equally effective embodiments.

[0004] FIG. **1** is a block diagram illustrating a system configured to implement one or more aspects of various embodiments.

[0005] FIG. **2** is a more detailed illustration of the training engine and execution engine of FIG. **1**, according to various embodiments.

[0006] FIG. **3** is a flow diagram of method steps for performing mesh reconstruction using data-driven priors, according to various embodiments.

[0007] FIG. **4** is a flow diagram of method steps for training a machine learning model to learn data-driven mesh priors, according to various embodiments.

[0008] FIG. **5** is a block diagram illustrating a computer system configured to implement one or more aspects of various embodiments.

[0009] FIG. **6** is a block diagram of a parallel processing unit (PPU) included in the parallel processing subsystem of FIG. **5**, according to various embodiments.

[0010] FIG. **7** is a block diagram of a general processing cluster (GPC) included in the parallel processing unit (PPU) of FIG. **6**, according to various embodiments.

[0011] FIG. **8** is a block diagram of an exemplary system on a chip (SoC) integrated circuit, according to various embodiments.

DETAILED DESCRIPTION

[0012] In the following description, numerous specific details are set forth to provide a more thorough understanding of the various embodiments. However, it will be apparent to one of skilled in the art that the inventive concepts may be practiced without one or more of these specific details.

System Overview

[0013] FIG. **1** illustrates a computing device **100** configured to implement one or more aspects of various embodiments. In one embodiment, computing device **100** may be a desktop computer, a laptop computer, a smart phone, a personal digital assistant (PDA), tablet computer, or any other type of computing device configured to receive input, process data, and optionally display images, and is suitable for practicing one or more embodiments. Computing device **100** is configured to run a training engine **122** and execution engine **124** that reside in a memory **116**. It is noted that the computing device described herein is illustrative and that any other technically feasible configurations fall within the scope of the present disclosure.

[0014] In one embodiment, computing device **100** includes, without limitation, an interconnect (bus) **112** that connects one or more processing units **102**, an input/output (I/O) device interface **104** coupled to one or more input/output (I/O) devices **108**, memory **116**, a storage **114**, and a network interface **106**. Processing unit(s) **102** may be any suitable processor implemented as a central processing unit (CPU), a graphics processing unit (GPU), an application-specific integrated circuit (ASIC), a field programmable gate array (FPGA), an artificial intelligence (AI) accelerator, any other type of processing unit, or a combination of different processing units, such as a CPU configured to operate in conjunction with a GPU. In general, processing unit(s) **102** may be any technically feasible hardware unit capable of processing data and/or executing software applications. Further, in the context of this disclosure, the computing elements shown in computing device **100** may correspond to a physical computing system (e.g., a system in a data center) or may be a virtual computing instance executing within a computing cloud.

[0015] In one embodiment, I/O devices **108** include devices capable of providing input, such as a keyboard, a mouse, a touch-sensitive screen, and so forth, as well as devices capable of providing output, such as a display device. Additionally, I/O devices **108** may include devices capable of both receiving input and providing output, such as a touchscreen, a universal serial bus (USB) port, and so forth. I/O devices **108** may be configured to receive various types of input from an end-user (e.g., a designer) of computing device **100**, and to also provide various types of output to the end-user of computing device **100**, such as displayed digital images or digital videos or text. In some embodiments, one or more of I/O devices **108** are configured to couple computing device **100** to a network **110**.

[0016] In one embodiment, network **110** is any technically feasible type of communications network that allows data to be exchanged between computing device **100** and external entities or devices, such as a web server or another networked computing device. For example, network **110** may include a wide area network (WAN), a local area network (LAN), a wireless (WiFi) network, and/or the Internet, among others.

[0017] In one embodiment, storage **114** includes non-volatile storage for applications and data, and may include fixed or removable disk drives, flash memory devices, and CD-ROM, DVD-ROM, Blu-Ray, HD-DVD, or other mag-

netic, optical, or solid state storage devices. Training engine **122** and execution engine **124** may be stored in storage **114** and loaded into memory **116** when executed.

[0018] In one embodiment, memory **116** includes a random access memory (RAM) module, a flash memory unit, or any other type of memory unit or combination thereof. Processing unit(s) **102**, I/O device interface **104**, and network interface **106** are configured to read data from and write data to memory **116**. Memory **116** includes various software programs that can be executed by processor(s) **102** and application data associated with said software programs, including training engine **122** and execution engine **124**.

[0019] In one embodiment, training engine **122** generates one or more machine learning models for performing mesh reconstruction using data-driven priors. Each machine learning model may learn priors related to vertices, edges, corners, faces, polygons, surfaces, shapes, and/or other attributes of two-dimensional (2D) and/or three-dimensional (3D) meshes. For example, each machine learning model may include a variational autoencoder (VAE) that learns to convert input meshes into latent vectors and reconstruct the meshes from the latent vectors.

[0020] In one embodiment, execution engine **124** executes the machine learning models to perform mesh reconstruction using the mesh priors learned by the machine learning models. Continuing with the above example, execution engine **124** may input an initial value of the latent vector into the decoder of the VAE to produce an initial estimate of a mesh. Execution engine **124** may refine the mesh by selecting subsequent values of the latent vector based on geometric constraints associated with a set of image observations of an object. In various embodiments, the subsequent values of the latent vector are selected to minimize errors between the mesh and the image observations, thus allowing the mesh to approximate the shape of the object. Training engine **122** and execution engine **124** are described in further detail below with respect to FIG. **2**.

Mesh Reconstruction Using Data-Driven Priors

[0021] FIG. **2** is a more detailed illustration of training engine **122** and execution engine **124** of FIG. **1**, according to various embodiments. In the embodiment shown, training engine **122** creates a machine learning model that learns to reconstruct a set of training meshes **208** by learning and/or encoding priors associated with training meshes **208**. For example, the machine learning model may learn, from training meshes **208**, a number of "basic types" related to vertices, edges, corners, faces, triangles, polygons, surfaces, shapes, and/or other attributes of 2D and/or 3D meshes.

[0022] In one embodiment, execution engine **124** uses the machine learning model to perform inverse rendering of an object **260** captured in a set of images **258** into a corresponding mesh **216**. For example, execution engine **124** may use the machine learning model to estimate a 3D mesh **216** of object from 2D images **258** that capture multiple views and/or multiple illuminations of object **260**. Thus, images **258** may represent ground truth image observations of object **260**.

[0023] Machine learning models created by training engine **122** can include any technically feasible form of machine learning model. For example, the machine learning models may include recurrent neural networks (RNNs), convolutional neural networks (CNNs), deep neural networks (DNNs), deep convolutional networks (DCNs), deep belief networks (DBNs), restricted Boltzmann machines (RBMs), long-short-term memory (LSTM) units, gated recurrent units (GRUs), generative adversarial networks (GANs), self-organizing maps (SOMs), and/or other types of artificial neural networks or components of artificial neural networks. In another example, the machine learning models may include functionality to perform clustering, principal component analysis (PCA), latent semantic analysis (LSA), Word2vec, and/or another unsupervised learning technique. In a third example, the machine learning models may include regression models, support vector machines, decision trees, random forests, gradient boosted trees, naïve Bayes classifiers, Bayesian networks, hierarchical models, and/or ensemble models.

[0024] In some embodiments, the machine learning model created by training engine **122** includes a VAE **200**, which includes an encoder **202** and a decoder **206**. For example, training engine **122** may input 2D and/or 3D training meshes **208** into encoder **202**, and encoder **202** may "encode" training meshes **208** into latent vectors **204** (e.g., scalars, geometric vectors, tensors, and/or other geometric objects) within a latent space of lower dimensionality than training meshes **208**. A decoder **206** in VAE **200** may "decode" latent vectors **204** into higher dimensionality reconstructed meshes **210** that substantially reproduce training meshes **208**.

[0025] In various embodiments, training meshes **208** may include collections of 2D and/or 3D points, representations of edges between pairs of points, and/or representations of triangles, polygons, and/or other shapes formed by the points and edges. Edges and/or shapes in training meshes **208** may be encoded in orderings of the points within training meshes **208**. For example, a 2D mesh may be parameterized as an ordered set of points, with each adjacent pair of points in the ordered set connected by an edge between the points. In another example, a 3D mesh may be parameterized using one or more triangle strips, triangle fans, and/or other representations of triangles within the mesh.

[0026] In one or more embodiments, training engine **122** configures VAE **200** to encode training meshes **208** in a way that is unaffected by the parameterization of training meshes **208** inputted into encoder **202**. In these embodiments, training engine **122** inputs multiple ordered sets of points in each training mesh into encoder **202** and standardizes the output of encoder **202** from the multiple ordered sets of points. Thus, training engine **122** may decouple the parameterizations of training meshes **208** from the latent space of VAE **200** into which training meshes **208** are encoded.

[0027] In one embodiment, ordered sets of points inputted into encoder **202** may include all possible orderings of points in a training mesh, a randomly selected subset of possible orderings of points in the training mesh, and/or any other combination of orderings of points in the training mesh. For example, training engine **122** may generate different ordered sets of points in a 2D mesh of a polygon by selecting different vertices of the polygon and/or various points along edges between the vertices as starting points in the ordered sets. In another example, training engine **122** may generate two different orderings of points in the 2D mesh from the same starting point by traversing the mesh in the clockwise and counterclockwise directions beginning at the starting point.

[0028] In one embodiment, training engine **122** trains a different encoder that is not in VAE **200** to generate inter-

mediate representations of ordered sets of points and/or other parameterizations of training meshes **208**. In one embodiment, training engine **122** averages and/or otherwise aggregates the intermediate representations into a standardized value in the latent space, such that the standardized value defines the mean and standard deviation from which the output of encoder **202** is sampled.

[0029] In one embodiment, training engine **122** inputs the output sampled using the standardized value into decoder **206** to obtain reconstructed meshes **210** as output of decoder **206**. Training engine **122** also computes an error between each training mesh and a corresponding reconstructed mesh as the minimum error between an ordering of points in the training mesh inputted into encoder **202** and the outputted ordering of points from decoder **206**. For example, training engine **122** may compute the minimum error based on the intersection of points in the training mesh and reconstructed mesh, a Chamfer distance between the training mesh and reconstructed mesh, an earth mover's distance, and/or another measure of distance, similarity, and/or dissimilarity between the points, edges, shapes, and/or other attributes of the training mesh and reconstructed mesh.

[0030] In one embodiment, training engine **122** uses the error between each training mesh and the corresponding reconstructed mesh to update the parameters of encoder **202** and decoder **206**. For example, training engine **122** may backpropagate the error across the layers and/or parameters of decoder **206** so that decoder **206** learns to decode the reconstructed mesh from the sampled output of encoder **202**. Training engine **122** may also backpropagate the error across the layers and/or parameters of encoder **202** so that encoder **202** learns to produce a consistent latent vector representation (e.g., mean and standard deviation vectors) for different orderings of points in the corresponding training mesh.

[0031] In one embodiment, training engine **122** trains decoder **206** to generate reconstructed meshes **210** from training meshes **208** at varying resolutions until a desired mesh resolution is reached. For example, decoder **206** may include a number of neural network layers, with each layer producing a reconstructed mesh with a higher number of vertices than the previous layer (e.g., by adding vertices to the centers of edges and/or polygon faces in the mesh from the previous layer). In one embodiment, during training of decoder **206**, training engine **122** may compare a coarse reconstructed mesh produced by the first layer of decoder **206** with the corresponding training mesh and update parameters of the first layer to reduce the error between the vertices of the reconstructed mesh and corresponding vertices of the training mesh. Training engine **122** may repeat the process with subsequent layers of decoder **206**, which upsample the reconstructed mesh from the previous layer, until the output layer of decoder **206** and/or the desired reconstructed mesh resolution is reached.

[0032] By performing a "coarse to fine" autoencoding of training meshes **208**, training engine **122** may allow the coarse mesh from the first layer of decoder **206** to be defined in a given reference coordinate frame and subsequent refinements to the coarse mesh from subsequent layers of decoder **206** to be defined in relative terms with respect to the coarse mesh. As a result, the refinements and, in turn, most of the reconstructed mesh may be independent of the reference frame or global pose of the object. This allows decoder **206** to learn variations in reference coordinate system across meshes more easily and/or potentially omit estimation of global pose during generation of reconstructed meshes **210**.

[0033] In the embodiment shown, execution engine **124** executes one or more portions of VAE **200** to produce a mesh **216** of an object **260** from a latent vector value **212** of a vector in the latent space of VAE **200** and images **258** of object **260**. For example, execution engine **124** may input latent vector value **212** into decoder **206** to generate mesh **216** as a decoded representation of the vector.

[0034] More specifically, in one embodiment, execution engine **124** varies latent vector value **212** inputted into decoder **206** with fixed parameters to generate mesh **216** so that mesh **216** reproduces an object **260** captured in a set of images **258**. In one or more embodiments, images **258** are associated with a set of assumptions that allow mesh **216** to be reconstructed base on images **258**. For example, images **258** may be associated with known camera poses that generate multiple views of object **260**, multiple lighting conditions that allow object **260** to be captured under different illuminations, and/or Lambertian surfaces on object **260** that have isotropic luminance (i.e., surfaces with uniform brightness from any direction of view).

[0035] In one embodiment, execution engine **124** initially selects latent vector value **212** based on one or more criteria. For example, execution engine **124** may randomly select the initial latent vector value **212** and/or set the initial latent vector value **212** to a default latent vector value associated with decoder **206**. In another example, execution engine **124** may set the initial latent vector value **212** based on sparse features extracted from 2D images **258** (e.g., features extracted using a scale-invariant feature transform (SIFT) technique, a corner detection technique, and/or another image feature detection technique). Execution engine **124** may use the sparse features to generate the initial latent vector value **212** by estimating 3D locations of the sparse features based on the assumptions associated with images **258**, inputting the locations into encoder **202**, and obtaining the initial latent vector value **212** based on output from encoder **202**.

[0036] In one embodiment, execution engine **124** uses assumptions associated with images **258** to determine geometric constraints **222** associated with object **260**, which may include constraints related to the shape and/or geometry of object **260**. Execution engine **124** also refines mesh **216** by updating latent vector value **212** based on mesh **216** and geometric constraints **222**.

[0037] In one embodiment, execution engine **124** uses geometric constraints **222** to calculate one or more errors associated with mesh **216** and updates latent vector value **212** based on gradients associated with the errors. For example, execution engine **124** may use known camera poses and/or lighting conditions associated with images **258** to back-project an image from a first camera to mesh **216** and project a warped image back to a second camera. Execution engine **124** may calculate a photometric error and/or silhouette error between the warped image and a corresponding image observation of object **260** from images **258** (e.g., an image of object **260** taken from the second camera's location and/or under the same lighting conditions used to produce the warped image). Execution engine **124** may use gradients associated with the photometric and/or silhouette error to modify latent vector value **212** so that mesh **216** better approximates the shape of object **260**. Execution engine **124** may also repeat the process, thus

iteratively updating latent vector value **212** to search the latent space associated with VAE **200** and using decoder **206** to generate a more accurate mesh **216** from latent vector value **212** until mesh **216** is a substantially accurate representation of object **260**.

[0038] In one embodiment, execution engine **124** generates warped images of mesh **216** for comparison to ground truth images **258** of object **260** by casting a ray from the first camera toward mesh **216**, selecting the ray's closest point of intersection on mesh **216**, and casting another ray from the point of intersection to the second camera. During this process, a face of mesh **216** may be represented as a binary indicator function in an image, where the face is "visible" in the image at locations where rays from the corresponding camera intersect with the face and invisible otherwise. Because the indicator function is binary and includes an abrupt change from "seen" pixels of the face to "unseen" pixels elsewhere, the image may be non-differentiable, which may interfere with generating gradients that are used to update latent vector value **212**.

[0039] In one or more embodiments, execution engine **124** generates images of mesh **216** using differentiable indicator functions that smooth the transition between visible and invisible portions of faces in mesh **216**. For example, a face in mesh **216** may have an indicator function that shows the face as "visible" wherever a ray from a camera intersects the face, invisible where a ray from the camera does not intersect the face, and smoothly but quickly transitioning between visible and invisible in the vicinity of the border between the two.

[0040] When mesh **216** is generated from a single latent vector value **212** inputted into decoder, decoder **206** may be required to learn the prior for the entire object **260**. As a result, execution engine **124** may be capable of generating mesh **216** as an accurate reconstruction of object **260** only when object **260** belongs to the distribution of training meshes **208** used to produce VAE **200**.

[0041] In one or more embodiments, execution engine **124** reduces the number and/or complexity of mesh priors learned by decoder **206** by dividing mesh **216** into multiple smaller meshlets **218** that represent different portions of mesh **218**. For example, execution engine **124** may generate each meshlet as one or more polygons, surfaces, and/or shapes in mesh **216**. Each meshlet may be defined using more basic priors than a much larger mesh **216** of object **260**, which may allow the priors to be generalized to a greater variety of objects and/or shapes. Each meshlet may also encompass a different part of mesh **218**, and meshlets **218** may be combined to produce mesh **218**.

[0042] As with generation of mesh **216** from a single latent vector value **212** inputted into decoder **206**, in one embodiment, execution engine **124** inputs multiple latent vector values **214** into decoder **206** to generate multiple corresponding meshlets **218** as output from decoder **206**. Execution engine **124** may similarly update latent vector values **214** to enforce subsets of geometric constraints **222** associated with individual meshlets **218** and/or reduce errors between meshlets **218** and the corresponding portions of object **260**. For example, execution engine **124** may map a meshlet to a portion of object **260** and use images **258** containing the portion to determine geometric constraints **222** associated with the portion. Execution engine **124** may use geometric constraints **222** to generate errors between the meshlet and images **258** and perform gradient descent on the latent vector value used to produce the meshlet based on the errors.

[0043] In the embodiment shown, meshlets **218** are additionally associated with poses **220** that project meshlets **218** back into mesh **216**. For example, the shape and/or geometry of each meshlet may be generated from a point in the latent space of VAE **200**. Each meshlet may also be associated with one or more custom poses **220** (e.g., one or more rotations, translations, and/or other transformations) that align the global pose of images **258** and/or mesh **216** with the canonical pose of the meshlet learned by decoder **206**.

[0044] In one embodiment, execution engine **124** uses gradients and/or errors associated with geometric constraints **222** to update both latent vector values **214** of meshlets **218** and poses **220** that transform meshlets **218** into their corresponding positions and/or orientations in mesh **216**. For example, object **260** may include a rectangular surface with four right-angle corners. During reconstruction of mesh **216** for object **260**, execution engine **124** may use geometric constraints **222** to identify a value in the latent space associated with VAE **200** that produces a meshlet with a right-angle corner and learn four different poses **220** that map the meshlet to the four corners of the rectangular surface.

[0045] In one embodiment, execution engine **124** iteratively increases a resolution of meshlets **218** to meet geometric constraints **222**. For example, execution engine **124** may use a first layer of decoder **206** to generate an initial coarse mesh **216** that matches low-resolution images **258** of object **260**. Execution engine **124** may use subsequent layers of decoder **206** to upsample the coarse mesh **216** into increasingly small and numerous meshlets **218**. At each layer of decoder **206**, execution engine **124** may perform gradient descent on latent vector values **214** used to produce meshlets **218** and poses **220** of meshlets **218** with respect to the coarse mesh until the desired mesh **216** and/or meshlet resolution is reached.

[0046] In one embodiment, execution engine **124** reconstructs mesh **216** from meshlets **218** and the corresponding poses **220** and stores the reconstructed mesh **216** in association with object **260**. For example, execution engine **124** may apply poses **220** to the corresponding meshlets **218** to map the points in the canonical poses of meshlets **218** to their locations in mesh **216**. Execution engine **124** may then render the completed mesh **216** and/or store the points in mesh **216** under an identifier for object **260**, with images **258** of object **260**, and/or with other metadata for object **260**.

[0047] FIG. **3** is a flow diagram of method steps for performing mesh reconstruction using data-driven priors, according to various embodiments. Although the method steps are described in conjunction with the systems of FIGS. **1** and **2**, persons skilled in the art will understand that any system configured to perform the method steps in any order falls within the scope of the present disclosure.

[0048] As shown, training engine **122** generates **302** a machine learning model as a VAE that learns to reconstruct a set of training meshes inputted into the VAE, as described in further detail below with respect to FIG. **4**. In various embodiments, the training meshes may include 2D and/or 3D meshes and/or meshlets.

[0049] Next, execution engine **124** executes **304** the machine learning model to produce a mesh of an object from a first value of a vector in a latent space. For example,

execution engine **124** may select the first value of the vector as a random value, a default value, a value that is based on a set of image observations of the object, and/or a value that is based on sparse features extracted from the image observations. Execution engine **124** may input the first value into a decoder in the VAE and obtain the mesh as output from the decoder. The similarity of the mesh to the object may be affected by the criteria used to select the first value of the vector (e.g., a first value of the vector that reflects sparse features from the image observations is likely to produce a mesh that is more similar to the object than a first value of the vector that is selected randomly).

[0050] Execution engine **124** refines **306** the mesh of the object by selecting a second value of the vector in the latent space based on one or more geometric constraints associated with the image observations of the object. For example, execution engine **124** may use known camera poses and/or lighting conditions under which the image observations were made to project and/or back-project warped images of the mesh. Execution engine **124** may also calculate photometric and/or silhouette errors between the warped images of the mesh and corresponding image observations (e.g., image observations made under the same camera poses and/or lighting conditions). Execution engine **124** may then use gradients associated with the errors and constant parameters of the machine learning model to update the value of the vector.

[0051] In another example, execution engine **124** may project and/or divide the mesh into a set of meshlets that represent different portions of the mesh. For each meshlet, execution engine **124** may select a value of the vector in the latent space to learn a prior for a portion of the mesh represented by the meshlet. Execution engine **124** may also learn a custom pose of the meshlet that maps the meshlet into a position and/or orientation within the mesh and/or aligns a global pose of the image observations with the canonical pose of the meshlet learned by the machine learning model. Execution engine **124** may then reconstruct the mesh from the set of meshlets by using the custom poses of the meshlets to map points in the meshlets into the reference system of the mesh.

[0052] Finally, execution engine **124** stores **308** the mesh in association with the object. For example, execution engine **124** may store the points in mesh **216** under an identifier for the object, with images of the object, and/or with other metadata for the object.

[0053] FIG. **4** is a flow diagram of method steps for training a machine learning model to learn data-driven mesh priors, according to various embodiments. Although the method steps are described in conjunction with the systems of FIGS. **1** and **2**, persons skilled in the art will understand that any system configured to perform the method steps in any order falls within the scope of the present disclosure.

[0054] As shown, training engine **122** applies **402** an encoder to multiple orderings of points in a training mesh to generate intermediate representations of the multiple orderings of points. For example, training engine **122** may select a random subset of all possible orderings of points in the training mesh as input to the encoder. The encoder may "encode" each ordering of points into a vector in a latent space.

[0055] Next, training engine **122** averages **404** the intermediate representations into a standardized value of a vector in the latent space. The standardized value may define the mean and standard deviation vectors outputted by an encoder in a VAE.

[0056] Training engine **122** then trains **406** a decoder in the VAE to reconstruct the training mesh from the standardized value. Training engine **122** also optionally trains **408** an encoder in the VAE to generate the standardized value from each ordering of points in the training mesh. For example, training engine **122** may calculate an error between the reconstructed mesh and the training mesh as the minimum error between an ordering of points in the training mesh inputted into the encoder and the outputted ordering of points from the decoder. The error may be based on an intersection of points between the training mesh and reconstructed mesh, a distance metric between the training mesh and reconstructed mesh, and/or other measures of similarity or dissimilarity between the training mesh and reconstructed mesh. Training engine **122** may backpropagate the error across the layers and/or parameters of the decoder so that the decoder learns to reconstruct the training mesh based on the standardized value. Training engine **122** may also backpropagate the error across the layers and/or parameters of the encoder so that the encoder learns to output the standardized value for different orderings of points in the corresponding training mesh.

Example Hardware Architecture

[0057] FIG. **5** is a block diagram illustrating a computer system **500** configured to implement one or more aspects of various embodiments. In some embodiments, computer system **500** is a server machine operating in a data center or a cloud computing environment that provides scalable computing resources as a service over a network. In some embodiments, computer system **500** implements the functionality of computing device **100** of FIG. **1**.

[0058] In various embodiments, computer system **500** includes, without limitation, a central processing unit (CPU) **502** and a system memory **504** coupled to a parallel processing subsystem **512** via a memory bridge **505** and a communication path **513**. Memory bridge **505** is further coupled to an I/O (input/output) bridge **507** via a communication path **506**, and I/O bridge **507** is, in turn, coupled to a switch **516**.

[0059] In one embodiment, I/O bridge **507** is configured to receive user input information from optional input devices **508**, such as a keyboard or a mouse, and forward the input information to CPU **502** for processing via communication path **506** and memory bridge **505**. In some embodiments, computer system **500** may be a server machine in a cloud computing environment. In such embodiments, computer system **500** may not have input devices **508**. Instead, computer system **500** may receive equivalent input information by receiving commands in the form of messages transmitted over a network and received via the network adapter **518**. In one embodiment, switch **516** is configured to provide connections between I/O bridge **507** and other components of the computer system **500**, such as a network adapter **518** and various add-in cards **520** and **521**.

[0060] In one embodiment, I/O bridge **507** is coupled to a system disk **514** that may be configured to store content and applications and data for use by CPU **502** and parallel processing subsystem **512**. In one embodiment, system disk **514** provides non-volatile storage for applications and data and may include fixed or removable hard disk drives, flash

memory devices, and CD-ROM (compact disc read-only-memory), DVD-ROM (digital versatile disc-ROM), Blu-ray, HD-DVD (high definition DVD), or other magnetic, optical, or solid state storage devices. In various embodiments, other components, such as universal serial bus or other port connections, compact disc drives, digital versatile disc drives, film recording devices, and the like, may be connected to I/O bridge **507** as well.

[0061] In various embodiments, memory bridge **505** may be a Northbridge chip, and I/O bridge **507** may be a Southbridge chip. In addition, communication paths **506** and **513**, as well as other communication paths within computer system **500**, may be implemented using any technically suitable protocols, including, without limitation, AGP (Accelerated Graphics Port), HyperTransport, or any other bus or point-to-point communication protocol known in the art.

[0062] In some embodiments, parallel processing subsystem **512** comprises a graphics subsystem that delivers pixels to an optional display device **510** that may be any conventional cathode ray tube, liquid crystal display, light-emitting diode display, or the like. In such embodiments, the parallel processing subsystem **512** incorporates circuitry optimized for graphics and video processing, including, for example, video output circuitry. As described in greater detail below in conjunction with FIGS. **6** and **7**, such circuitry may be incorporated across one or more parallel processing units (PPUs), also referred to herein as parallel processors, included within parallel processing subsystem **512**.

[0063] In other embodiments, the parallel processing subsystem **512** incorporates circuitry optimized for general purpose and/or compute processing. Again, such circuitry may be incorporated across one or more PPUs included within parallel processing subsystem **512** that are configured to perform such general purpose and/or compute operations. In yet other embodiments, the one or more PPUs included within parallel processing subsystem **512** may be configured to perform graphics processing, general purpose processing, and compute processing operations. System memory **504** includes at least one device driver configured to manage the processing operations of the one or more PPUs within parallel processing subsystem **512**.

[0064] In various embodiments, parallel processing subsystem **512** may be integrated with one or more of the other elements of FIG. **5** to form a single system. For example, parallel processing subsystem **512** may be integrated with CPU **502** and other connection circuitry on a single chip to form a system on chip (SoC).

[0065] In one embodiment, CPU **502** is the master processor of computer system **500**, controlling and coordinating operations of other system components. In one embodiment, CPU **502** issues commands that control the operation of PPUs. In some embodiments, communication path **513** is a PCI Express link, in which dedicated lanes are allocated to each PPU, as is known in the art. Other communication paths may also be used. PPU advantageously implements a highly parallel processing architecture. A PPU may be provided with any amount of local parallel processing memory (PP memory).

[0066] It will be appreciated that the system shown herein is illustrative and that variations and modifications are possible. The connection topology, including the number and arrangement of bridges, the number of CPUs **502**, and the number of parallel processing subsystems **512**, may be modified as desired. For example, in some embodiments, system memory **504** could be connected to CPU **502** directly rather than through memory bridge **505**, and other devices would communicate with system memory **504** via memory bridge **505** and CPU **502**. In other embodiments, parallel processing subsystem **512** may be connected to I/O bridge **507** or directly to CPU **502**, rather than to memory bridge **505**. In still other embodiments, I/O bridge **507** and memory bridge **505** may be integrated into a single chip instead of existing as one or more discrete devices. Lastly, in certain embodiments, one or more components shown in FIG. **5** may not be present. For example, switch **516** could be eliminated, and network adapter **518** and add-in cards **520**, **521** would connect directly to I/O bridge **507**.

[0067] FIG. **6** is a block diagram of a parallel processing unit (PPU) **602** included in the parallel processing subsystem **512** of FIG. **5**, according to various embodiments. Although FIG. **6** depicts one PPU **602**, as indicated above, parallel processing subsystem **512** may include any number of PPUs **602**. As shown, PPU **602** is coupled to a local parallel processing (PP) memory **604**. PPU **602** and PP memory **604** may be implemented using one or more integrated circuit devices, such as programmable processors, application specific integrated circuits (ASICs), or memory devices, or in any other technically feasible fashion.

[0068] In some embodiments, PPU **602** comprises a graphics processing unit (GPU) that may be configured to implement a graphics rendering pipeline to perform various operations related to generating pixel data based on graphics data supplied by CPU **502** and/or system memory **504**. When processing graphics data, PP memory **604** can be used as graphics memory that stores one or more conventional frame buffers and, if needed, one or more other render targets as well. Among other things, PP memory **604** may be used to store and update pixel data and deliver final pixel data or display frames to an optional display device **510** for display. In some embodiments, PPU **602** also may be configured for general-purpose processing and compute operations. In some embodiments, computer system **500** may be a server machine in a cloud computing environment. In such embodiments, computer system **500** may not have a display device **510**. Instead, computer system **500** may generate equivalent output information by transmitting commands in the form of messages over a network via the network adapter **518**.

[0069] In some embodiments, CPU **502** is the master processor of computer system **500**, controlling and coordinating operations of other system components. In one embodiment, CPU **502** issues commands that control the operation of PPU **602**. In some embodiments, CPU **502** writes a stream of commands for PPU **602** to a data structure (not explicitly shown in either FIG. **5** or FIG. **6**) that may be located in system memory **504**, PP memory **604**, or another storage location accessible to both CPU **502** and PPU **602**. A pointer to the data structure is written to a command queue, also referred to herein as a pushbuffer, to initiate processing of the stream of commands in the data structure. In one embodiment, the PPU **602** reads command streams from the command queue and then executes commands asynchronously relative to the operation of CPU **502**. In embodiments where multiple pushbuffers are generated, execution priorities may be specified for each pushbuffer by an application program via device driver to control scheduling of the different pushbuffers.

[0070] In one embodiment, PPU **602** includes an I/O (input/output) unit **605** that communicates with the rest of computer system **500** via the communication path **513** and memory bridge **505**. In one embodiment, I/O unit **605** generates packets (or other signals) for transmission on communication path **513** and also receives all incoming packets (or other signals) from communication path **513**, directing the incoming packets to appropriate components of PPU **602**. For example, commands related to processing tasks may be directed to a host interface **606**, while commands related to memory operations (e.g., reading from or writing to PP memory **604**) may be directed to a crossbar unit **610**. In one embodiment, host interface **606** reads each command queue and transmits the command stream stored in the command queue to a front end **612**.

[0071] As mentioned above in conjunction with FIG. **5**, the connection of PPU **602** to the rest of computer system **500** may be varied. In some embodiments, parallel processing subsystem **512**, which includes at least one PPU **602**, is implemented as an add-in card that can be inserted into an expansion slot of computer system **500**. In other embodiments, PPU **602** can be integrated on a single chip with a bus bridge, such as memory bridge **505** or I/O bridge **507**. Again, in still other embodiments, some or all of the elements of PPU **602** may be included along with CPU **502** in a single integrated circuit or system of chip (SoC).

[0072] In one embodiment, front end **612** transmits processing tasks received from host interface **606** to a work distribution unit (not shown) within task/work unit **607**. In one embodiment, the work distribution unit receives pointers to processing tasks that are encoded as task metadata (TMD) and stored in memory. The pointers to TMDs are included in a command stream that is stored as a command queue and received by the front end unit **612** from the host interface **606**. Processing tasks that may be encoded as TMDs include indices associated with the data to be processed as well as state parameters and commands that define how the data is to be processed. For example, the state parameters and commands could define the program to be executed on the data. Also for example, the TMD could specify the number and configuration of the set of CTAs. Generally, each TMD corresponds to one task. The task/work unit **607** receives tasks from the front end **612** and ensures that GPCs **608** are configured to a valid state before the processing task specified by each one of the TMDs is initiated. A priority may be specified for each TMD that is used to schedule the execution of the processing task. Processing tasks also may be received from the processing cluster array **630**. Optionally, the TMD may include a parameter that controls whether the TMD is added to the head or the tail of a list of processing tasks (or to a list of pointers to the processing tasks), thereby providing another level of control over execution priority.

[0073] In one embodiment, PPU **602** implements a highly parallel processing architecture based on a processing cluster array **630** that includes a set of C general processing clusters (GPCs) **608**, where $C \geq 1$. Each GPC **608** is capable of executing a large number (e.g., hundreds or thousands) of threads concurrently, where each thread is an instance of a program. In various applications, different GPCs **608** may be allocated for processing different types of programs or for performing different types of computations. The allocation of GPCs **608** may vary depending on the workload arising for each type of program or computation.

[0074] In one embodiment, memory interface **614** includes a set of D of partition units **615**, where $D \geq 1$. Each partition unit **615** is coupled to one or more dynamic random access memories (DRAMs) **620** residing within PPM memory **604**. In some embodiments, the number of partition units **615** equals the number of DRAMs **620**, and each partition unit **615** is coupled to a different DRAM **620**. In other embodiments, the number of partition units **615** may be different than the number of DRAMs **620**. Persons of ordinary skill in the art will appreciate that a DRAM **620** may be replaced with any other technically suitable storage device. In operation, various render targets, such as texture maps and frame buffers, may be stored across DRAMs **620**, allowing partition units **615** to write portions of each render target in parallel to efficiently use the available bandwidth of PP memory **604**.

[0075] In one embodiment, a given GPC **608** may process data to be written to any of the DRAMs **620** within PP memory **604**. In one embodiment, crossbar unit **610** is configured to route the output of each GPC **608** to the input of any partition unit **615** or to any other GPC **608** for further processing. GPCs **608** communicate with memory interface **614** via crossbar unit **610** to read from or write to various DRAMs **620**. In some embodiments, crossbar unit **610** has a connection to I/O unit **605**, in addition to a connection to PP memory **604** via memory interface **614**, thereby enabling the processing cores within the different GPCs **608** to communicate with system memory **504** or other memory not local to PPU **602**. In the embodiment of FIG. **6**, crossbar unit **610** is directly connected with I/O unit **605**. In various embodiments, crossbar unit **610** may use virtual channels to separate traffic streams between the GPCs **608** and partition units **615**.

[0076] In one embodiment, GPCs **608** can be programmed to execute processing tasks relating to a wide variety of applications, including, without limitation, linear and non-linear data transforms, filtering of video and/or audio data, modeling operations (e.g., applying laws of physics to determine position, velocity and other attributes of objects), image rendering operations (e.g., tessellation shader, vertex shader, geometry shader, and/or pixel/fragment shader programs), general compute operations, etc. In operation, PPU **602** is configured to transfer data from system memory **504** and/or PP memory **604** to one or more on-chip memory units, process the data, and write result data back to system memory **504** and/or PP memory **604**. The result data may then be accessed by other system components, including CPU **502**, another PPU **602** within parallel processing subsystem **512**, or another parallel processing subsystem **512** within computer system **500**.

[0077] In one embodiment, any number of PPUs **602** may be included in a parallel processing subsystem **512**. For example, multiple PPUs **602** may be provided on a single add-in card, or multiple add-in cards may be connected to communication path **513**, or one or more of PPUs **602** may be integrated into a bridge chip. PPUs **602** in a multi-PPU system may be identical to or different from one another. For example, different PPUs **602** might have different numbers of processing cores and/or different amounts of PP memory **604**. In implementations where multiple PPUs **602** are present, those PPUs may be operated in parallel to process data at a higher throughput than is possible with a single PPU **602**. Systems incorporating one or more PPUs **602** may be implemented in a variety of configurations and form

factors, including, without limitation, desktops, laptops, handheld personal computers or other handheld devices, servers, workstations, game consoles, embedded systems, and the like.

[0078] FIG. **7** is a block diagram of a general processing cluster (GPC) **608** included in the parallel processing unit (PPU) **602** of FIG. **6**, according to various embodiments. As shown, the GPC **608** includes, without limitation, a pipeline manager **705**, one or more texture units **715**, a preROP unit **725**, a work distribution crossbar **730**, and an L1.5 cache **735**.

[0079] In one embodiment, GPC **608** may be configured to execute a large number of threads in parallel to perform graphics, general processing and/or compute operations. As used herein, a "thread" refers to an instance of a particular program executing on a particular set of input data. In some embodiments, single-instruction, multiple-data (SIMD) instruction issue techniques are used to support parallel execution of a large number of threads without providing multiple independent instruction units. In other embodiments, single-instruction, multiple-thread (SIMT) techniques are used to support parallel execution of a large number of generally synchronized threads, using a common instruction unit configured to issue instructions to a set of processing engines within GPC **608**. Unlike a SIMD execution regime, where all processing engines typically execute identical instructions, SIMT execution allows different threads to more readily follow divergent execution paths through a given program. Persons of ordinary skill in the art will understand that a SIMD processing regime represents a functional subset of a SIMT processing regime.

[0080] In one embodiment, operation of GPC **608** is controlled via a pipeline manager **705** that distributes processing tasks received from a work distribution unit (not shown) within task/work unit **607** to one or more streaming multiprocessors (SMs) **710**. Pipeline manager **705** may also be configured to control a work distribution crossbar **730** by specifying destinations for processed data output by SMs **710**.

[0081] In various embodiments, GPC **608** includes a set of M of SMs **710**, where M≥1. Also, each SM **710** includes a set of functional execution units (not shown), such as execution units and load-store units. Processing operations specific to any of the functional execution units may be pipelined, which enables a new instruction to be issued for execution before a previous instruction has completed execution. Any combination of functional execution units within a given SM **710** may be provided. In various embodiments, the functional execution units may be configured to support a variety of different operations including integer and floating point arithmetic (e.g., addition and multiplication), comparison operations, Boolean operations (AND, OR, 5OR), bit-shifting, and computation of various algebraic functions (e.g., planar interpolation and trigonometric, exponential, and logarithmic functions, etc.). Advantageously, the same functional execution unit can be configured to perform different operations.

[0082] In various embodiments, each SM **710** includes multiple processing cores. In one embodiment, the SM **710** includes a large number (e.g., 128, etc.) of distinct processing cores. Each core may include a fully-pipelined, single-precision, double-precision, and/or mixed precision processing unit that includes a floating point arithmetic logic unit and an integer arithmetic logic unit. In one embodiment, the floating point arithmetic logic units implement the IEEE 754-2008 standard for floating point arithmetic. In one embodiment, the cores include 64 single-precision (32-bit) floating point cores, 64 integer cores, 32 double-precision (64-bit) floating point cores, and 8 tensor cores.

[0083] In one embodiment, tensor cores configured to perform matrix operations, and, in one embodiment, one or more tensor cores are included in the cores. In particular, the tensor cores are configured to perform deep learning matrix arithmetic, such as convolution operations for neural network training and inferencing. In one embodiment, each tensor core operates on a 4×4 matrix and performs a matrix multiply and accumulate operation D=A×B+C, where A, B, C, and D are 4×4 matrices.

[0084] In one embodiment, the matrix multiply inputs A and B are 16-bit floating point matrices, while the accumulation matrices C and D may be 16-bit floating point or 32-bit floating point matrices. Tensor Cores operate on 16-bit floating point input data with 32-bit floating point accumulation. The 16-bit floating point multiply requires 64 operations and results in a full precision product that is then accumulated using 32-bit floating point addition with the other intermediate products for a 4×4×4 matrix multiply. In practice, Tensor Cores are used to perform much larger two-dimensional or higher dimensional matrix operations, built up from these smaller elements. An API, such as CUDA 9 C++ API, exposes specialized matrix load, matrix multiply and accumulate, and matrix store operations to efficiently use tensor cores from a CUDA-C++ program. At the CUDA level, the warp-level interface assumes 16×16 size matrices spanning all 32 threads of the warp.

[0085] Neural networks rely heavily on matrix math operations, and complex multi-layered networks require tremendous amounts of floating-point performance and bandwidth for both efficiency and speed. In various embodiments, with thousands of processing cores, optimized for matrix math operations, and delivering tens to hundreds of TFLOPS of performance, the SMs **710** provide a computing platform capable of delivering performance required for deep neural network-based artificial intelligence and machine learning applications.

[0086] In various embodiments, each SM **710** may also comprise multiple special function units (SFUs) that perform special functions (e.g., attribute evaluation, reciprocal square root, and the like). In one embodiment, the SFUs may include a tree traversal unit configured to traverse a hierarchical tree data structure. In one embodiment, the SFUs may include texture unit configured to perform texture map filtering operations. In one embodiment, the texture units are configured to load texture maps (e.g., a 2D array of texels) from memory and sample the texture maps to produce sampled texture values for use in shader programs executed by the SM. In various embodiments, each SM **710** also comprises multiple load/store units (LSUs) that implement load and store operations between the shared memory/L1 cache and register files internal to the SM **710**.

[0087] In one embodiment, each SM **710** is configured to process one or more thread groups. As used herein, a "thread group" or "warp" refers to a group of threads concurrently executing the same program on different input data, with one thread of the group being assigned to a different execution unit within an SM **710**. A thread group may include fewer threads than the number of execution units within the SM **710**, in which case some of the execution may be idle during

cycles when that thread group is being processed. A thread group may also include more threads than the number of execution units within the SM **710**, in which case processing may occur over consecutive clock cycles. Since each SM **710** can support up to G thread groups concurrently, it follows that up to G*M thread groups can be executing in GPC **608** at any given time.

[0088] Additionally, in one embodiment, a plurality of related thread groups may be active (in different phases of execution) at the same time within an SM **710**. This collection of thread groups is referred to herein as a "cooperative thread array" ("CTA") or "thread array." The size of a particular CTA is equal to m*k, where k is the number of concurrently executing threads in a thread group, which is typically an integer multiple of the number of execution units within the SM **710**, and m is the number of thread groups simultaneously active within the SM **710**. In some embodiments, a single SM **710** may simultaneously support multiple CTAs, where such CTAs are at the granularity at which work is distributed to the SMs **710**.

[0089] In one embodiment, each SM **710** contains a level one (L1) cache or uses space in a corresponding L1 cache outside of the SM **710** to support, among other things, load and store operations performed by the execution units. Each SM **710** also has access to level two (L2) caches (not shown) that are shared among all GPCs **608** in PPU **602**. The L2 caches may be used to transfer data between threads. Finally, SMs **710** also have access to off-chip "global" memory, which may include PP memory **604** and/or system memory **504**. It is to be understood that any memory external to PPU **602** may be used as global memory. Additionally, as shown in FIG. **7**, a level one-point-five (L1.5) cache **735** may be included within GPC **608** and configured to receive and hold data requested from memory via memory interface **614** by SM **710**. Such data may include, without limitation, instructions, uniform data, and constant data. In embodiments having multiple SMs **710** within GPC **608**, the SMs **710** may beneficially share common instructions and data cached in L1.5 cache **735**.

[0090] In one embodiment, each GPC **608** may have an associated memory management unit (MMU) **720** that is configured to map virtual addresses into physical addresses. In various embodiments, MMU **720** may reside either within GPC **608** or within the memory interface **614**. The MMU **720** includes a set of page table entries (PTEs) used to map a virtual address to a physical address of a tile or memory page and optionally a cache line index. The MMU **720** may include address translation lookaside buffers (TLB) or caches that may reside within SMs **710**, within one or more L1 caches, or within GPC **608**.

[0091] In one embodiment, in graphics and compute applications, GPC **608** may be configured such that each SM **710** is coupled to a texture unit **715** for performing texture mapping operations, such as determining texture sample positions, reading texture data, and filtering texture data.

[0092] In one embodiment, each SM **710** transmits a processed task to work distribution crossbar **730** in order to provide the processed task to another GPC **608** for further processing or to store the processed task in an L2 cache (not shown), parallel processing memory **604**, or system memory **504** via crossbar unit **610**. In addition, a pre-raster operations (preROP) unit **725** is configured to receive data from SM **710**, direct data to one or more raster operations (ROP) units within partition units **615**, perform optimizations for color blending, organize pixel color data, and perform address translations.

[0093] It will be appreciated that the architecture described herein is illustrative and that variations and modifications are possible. Among other things, any number of processing units, such as SMs **710**, texture units **715**, or preROP units **725**, may be included within GPC **608**. Further, as described above in conjunction with FIG. **6**, PPU **602** may include any number of GPCs **608** that are configured to be functionally similar to one another so that execution behavior does not depend on which GPC **608** receives a particular processing task. Further, each GPC **608** operates independently of the other GPCs **608** in PPU **602** to execute tasks for one or more application programs.

[0094] FIG. **8** is a block diagram of an exemplary system on a chip (SoC) integrated circuit **800**, according to various embodiments. SoC integrated circuit **800** includes one or more application processors **802** (e.g., CPUs), one or more graphics processors **804** (e.g., GPUs), one or more image processors **806**, and/or one or more video processors **808**. SoC integrated circuit **800** also includes peripheral or bus components such as a serial interface controller **814** that implements Universal Serial Bus (USB), Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), Secure Digital Input Output (SDIO), inter-IC sound (I^2S), and/or Inter-Integrated Circuit (I^2C). SoC integrated circuit **800** additionally includes a display device **818** coupled to a display interface **820** such as high-definition multimedia interface (HDMI) and/or a mobile industry processor interface (MIPI). SoC integrated circuit **800** further includes a Flash memory subsystem **824** that provides storage on the integrated circuit, as well as a memory controller **822** that provides a memory interface for access to memory devices.

[0095] In one or more embodiments, SoC integrated circuit **800** is implemented using one or more types of integrated circuit components. For example, SoC integrated circuit **800** may include one or more processor cores for application processors **802** and/or graphics processors **804**. Additional functionality associated with serial interface controller **814**, display device **818**, display interface **820**, image processors **806**, video processors **808**, AI acceleration, machine vision, and/or other specialized tasks may be provided by application-specific integrated circuits (ASICs), application-specific standard parts (ASSPs), field-programmable gate arrays (FPGAs), and/or other types of customized components.

[0096] In sum, the disclosed techniques use a VAE to learn priors associated with meshes and/or portions of meshes. A decoder in the VAE is used to generate additional meshes of objects by searching the latent space of the VAE for latent vector values that reduce errors between the meshes and image observations of the objects. Each mesh may also be subdivided into meshlets, and shapes and poses of the meshlets may be individually refined to meet geometric constraints associated with the image observations before the meshlets are combined back into the mesh.

[0097] One technological advantage of the disclosed techniques is that mesh priors learned by the VAE are used to perform inverse rendering of objects in a way that enforces geometric constraints associated with image observations of the objects. Another technological advantage of the disclosed techniques includes reduced complexity associated

with estimating the global pose of the objects by performing coarse-to-fine rendering of the meshes. A third technological advantage of the disclosed techniques is increased generalizability and reduced complexity in reconstructing the meshes through the division of the meshes into meshlets. Consequently, the disclosed techniques provide technological improvements in machine learning models, computer systems, applications, and/or techniques for performing mesh reconstruction.

[0098] 1. In some embodiments, a processor comprises logic to predict one or more three-dimensional (3D) mesh representations based on a plurality of digital images, wherein the one or more 3D mesh representations are refined by minimizing a difference between the one or more 3D mesh representations and the plurality of digital images.

[0099] 2. The processor of clause 1, wherein predicting the one or more 3D mesh representations based on the plurality of digital images comprises executing a machine learning model to produce a mesh of an object from a first value in a latent space; and refining the mesh of the object by selecting a second value in the latent space based on one or more geometric constraints associated with the plurality of digital images of the object.

[0100] 3. The processor of clauses 1-2, wherein refining the mesh of the object comprises selecting the first value; calculating an error between the mesh and the plurality of digital images of the object; and performing gradient descent on the first value with parameters of the machine learning model to reduce the error.

[0101] 4. The processor of clauses 1-3, wherein selecting the first value comprises at least one of randomizing the first value, selecting the first value based on the set of image observations, and initializing the first value based on sparse features extracted from the set of image observations.

[0102] 5. The processor of clauses 1-4, wherein refining the mesh of the object comprises dividing the mesh into a set of meshlets; for each meshlet in the set of meshlets, selecting the second value in the latent space to learn a prior for a portion of the mesh represented by the meshlet; and reconstructing the mesh from the set of meshlets.

[0103] 6. The processor of clauses 1-5, wherein refining the mesh of the object further comprises, for each meshlet in the set of meshlets, learning a custom pose of the meshlet that aligns a global pose of the images with a canonical pose of the meshlet.

[0104] 7. The processor of clauses 1-6, wherein refining the mesh of the object further comprises iteratively increasing a resolution of the set of meshlets to meet the one or more geometric constraints.

[0105] 8. The processor of clauses 1-7, wherein the logic further generates the machine learning model as a variational autoencoder to reconstruct a set of training meshes inputted into the variational autoencoder.

[0106] 9. The processor of clauses 1-8, wherein generating the machine learning model comprises for each training mesh in the set of training meshes, aggregating multiple orderings of points in the training mesh into a standardized value in the latent space; and training a decoder in the variational autoencoder to reconstruct the training mesh from the standardized value.

[0107] 10. The processor of clauses 1-9, wherein aggregating the multiple orderings of points in the training mesh into the standardized value comprises applying an encoder to the multiple orderings of points to generate intermediate representations of the multiple orderings of points; and averaging the intermediate representations into the standardized value.

[0108] 11. In some embodiments, a method comprises predicting one or more three-dimensional (3D) mesh representations based on a plurality of digital images, wherein the one or more 3D mesh representations are refined by minimizing at least one difference between the one or more 3D mesh representations and the plurality of digital images.

[0109] 12. The method of clause 11, wherein predicting the one or more 3D mesh representations based on the plurality of digital images comprises executing a machine learning model to produce a mesh of an object from a first value in a latent space; and refining the mesh of the object by selecting a second value in the latent space based on one or more geometric constraints associated with the plurality of digital images of the object.

[0110] 13. The method of clauses 11-12, further comprising generating the machine learning model as a variational autoencoder to reconstruct a set of training meshes inputted into the variational autoencoder.

[0111] 14. The method of clauses 11-13, wherein generating the machine learning model comprises for each training mesh in the set of training meshes, aggregating multiple orderings of points in the training mesh into a standardized value in the latent space; and training a decoder in the variational autoencoder to reconstruct the training mesh from the standardized value.

[0112] 15. The method of clauses 11-14, wherein aggregating the multiple orderings of points in the training mesh into the standardized value comprises applying an encoder to the multiple orderings of points to generate intermediate representations of the multiple orderings of points; and averaging the intermediate representations into the standardized value.

[0113] 16. In some embodiments, a non-transitory computer readable medium store instructions that, when executed by a processor, cause the processor to at least execute a machine learning model to produce a mesh of an object from a first value in a latent space; refine the mesh of the object by selecting a second value in the latent space based on one or more geometric constraints associated with a set of image observations of the object; and store the mesh in association with the object.

[0114] 17. The non-transitory computer readable medium of clause 16, wherein refining the mesh of the object comprises selecting the first value; and performing gradient descent on the first value with parameters of the machine learning model to reduce an error between the mesh and the set of image observations.

[0115] 18. The non-transitory computer readable medium of clauses 16-17, wherein the error comprises at least one of a photometric error and a silhouette error.

[0116] 19. The non-transitory computer readable medium of clauses 16-18, wherein refining the mesh of the object comprises dividing the mesh into a set of meshlets; for each meshlet in the set of meshlets, selecting the second value in the latent space to learn a prior for a portion of the mesh represented by the meshlet; and reconstructing the mesh from the set of meshlets.

[0117] 20. The non-transitory computer readable medium of clauses 16-19, wherein refining the mesh of the object

further comprises iteratively increasing a resolution of the set of meshlets to meet the one or more geometric constraints.

[0118] Any and all combinations of any of the claim elements recited in any of the claims and/or any elements described in this application, in any fashion, fall within the contemplated scope of the present embodiments and protection.

[0119] The descriptions of the various embodiments have been presented for purposes of illustration, but are not intended to be exhaustive or limited to the embodiments disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the described embodiments.

[0120] Aspects of the present embodiments may be embodied as a system, method or computer program product. Accordingly, aspects of the present disclosure may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "module" or "system." In addition, any hardware and/or software technique, process, function, component, engine, module, or system described in the present disclosure may be implemented as a circuit or set of circuits. Furthermore, aspects of the present disclosure may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

[0121] Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

[0122] Aspects of the present disclosure are described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine. The instructions, when executed via the processor of the computer or other programmable data processing apparatus, enable the implementation of the functions/acts specified in the flowchart and/or block diagram block or blocks. Such processors may be, without limitation, general purpose processors, special-purpose processors, application-specific processors, or field-programmable gate arrays.

[0123] The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function (s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0124] While the preceding is directed to embodiments of the present disclosure, other and further embodiments of the disclosure may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

What is claimed is:

1. A processor, comprising:

one or more circuits to use one or more neural networks to modify one or more three-dimensional (3D) mesh representations of one or more images of one or more objects based, at least in part, on one or more differences between the one or more 3D mesh representations and the one or more images of the one or more objects.

2. The processor of claim **1**, wherein the one or more circuits are to use the one or more neural networks to estimate the one or more 3D mesh representations of an object using one or more 2-dimensional (2D) images comprising two or more views of the object.

3. The processor of claim **1**, wherein the one or more circuits are to use the one or more neural networks to generate the one or more 3D mesh representations of an object using one or more latent vector values in latent space and the one or more images of the one or more objects.

4. The processor of claim **1**, wherein the one or more circuits are to:

identify one or more geometric constraints corresponding to an object of the one or more objects; and

use the one or more neural networks to modify the one or more 3D representations by updating latent vector values based on the one or more 3D mesh representations and the one or more geometric constraints.

5. The processor of claim **1**, wherein the one or more circuits are to use the one or more neural networks to modify one or more portions of the one or more 3D mesh representations corresponding to the one or more portions of the one or more objects.

6. The processor of claim 1, wherein the one or more 3D mesh representations are modified by using an error between the one or more 3D mesh representations and one or more corresponding reconstructed meshes to update one or more parameters of the one or more neural networks.

7. The processor of claim 1, wherein the one or more 3D mesh representations are modified by using one or more reconstructed meshes at varying resolutions until a desired resolution of the one or more 3D mesh representations is reached.

8. The processor of claim 1, wherein the one or more 3D mesh representations are modified by generating set of reconstructed meshes with increasing number of vertices and comparing the set of reconstructed meshes to the one or more 3D mesh representations to reduce an error between the set of reconstructed meshes and the one or more 3D mesh representations.

9. A method, comprising:

using one or more NNs to modify one or more three-dimensional (3D) mesh representations of one or more images of one or more objects based, at least in part, on one or more differences between the one or more 3D mesh representations and the one or more images of the one or more objects.

10. The method of claim 9, further comprising extracting a value representing one or more features of the one or more objects and using the value to cause the one or more neural networks to generate the one or more 3D mesh representations.

11. The method of claim 9, wherein the one or more differences comprises an error between warped images of the one or more 3D mesh representations and the one or more images.

12. The method of claim 9, further comprising generating a set of meshlets from the one or more 3D mesh representations, wherein each meshlet of the set of meshlets representing a different portion of the one or more 3D mesh representations.

13. The method of claim 9, wherein using the one or more neural networks to modify the one or more 3D mesh representations comprises reconstructing the one or more 3D mesh representations from a set of meshlets representing different portions of the one or more objects.

14. The method of claim 9, further comprising using the one or more neural networks to estimate the one or more 3D mesh representations of an object using one or more two-dimensional (2D) images comprising two or more illuminations of the object.

15. A non-transitory computer readable medium storing instructions that, when executed by a processor, cause the processor to at least:

use one or more neural networks to modify one or more three-dimensional (3D) mesh representations of one or more images of one or more objects based, at least in part, on one or more differences between the one or more 3D mesh representations and the one or more images of the one or more objects.

16. The non-transitory computer readable medium of claim 15, wherein the instructions, when executed by the processor, further cause the processor to at least:

use a decoder to generate the one or more 3D mesh representations of an object, wherein the decoder receives one or more latent vector values as input and generates a decoded representation of the one or more latent vector values.

17. The non-transitory computer readable medium of claim 15, wherein the one or more 3D mesh representations are modified by increasing a resolution of the one or more 3D mesh representations using geometric constraints imposed on the one or more objects, and wherein the geometric constraints correspond to one or more conditions of a set of conditions under which the one or more images were captured.

18. The non-transitory computer readable medium of claim 15, wherein the one or more 3D mesh representations are modified by increasing a resolution of the one or more 3D mesh representations by iteratively increasing a number of meshlets of a set of meshlets used to reconstruct the one or more 3D mesh representations, wherein each meshlet of the set of meshlets representing a portion of the one or more 3D mesh representations.

19. The non-transitory computer readable medium of claim 15, wherein the neural network is trained to modify the one or more 3D mesh representations by generating one or more 3D mesh representations and one or more reconstructed meshes generated based on the one or more 3D mesh representations, and minimizing an error between each of the one or more 3D mesh representations and a corresponding reconstructed mesh of the one or more reconstructed meshes.

20. The non-transitory computer readable medium of claim 15, wherein the one or more differences comprises reducing an error between the one or more 3D mesh representations and the one or more images of the one or more objects.

* * * * *