US012394144B2

(12) **United States Patent**
Gautron et al.

(10) **Patent No.: US 12,394,144 B2**
(45) **Date of Patent: Aug. 19, 2025**

(54) **CONSISTENT SAMPLING FOR SPATIAL HASHING**

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

(72) Inventors: **Pascal Albert Gautron**, Speracedes (FR); **Carsten Alexander Waechter**, Berlin (DE)

(73) Assignee: **Nvidia Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 9 days.

(21) Appl. No.: **17/536,531**

(22) Filed: **Nov. 29, 2021**

(65) **Prior Publication Data**

US 2023/0169721 A1 Jun. 1, 2023

(51) **Int. Cl.**
G06T 15/50 (2011.01)
G06T 15/06 (2011.01)

(52) **U.S. Cl.**
CPC ***G06T 15/506*** (2013.01); ***G06T 15/06*** (2013.01)

(58) **Field of Classification Search**
CPC G06T 15/06
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2017/0017645 A1* 1/2017 Neumeier G06F 16/41
2021/0166464 A1* 6/2021 Moloney B64C 39/024
2022/0406002 A1* 12/2022 Boisse G06T 15/06

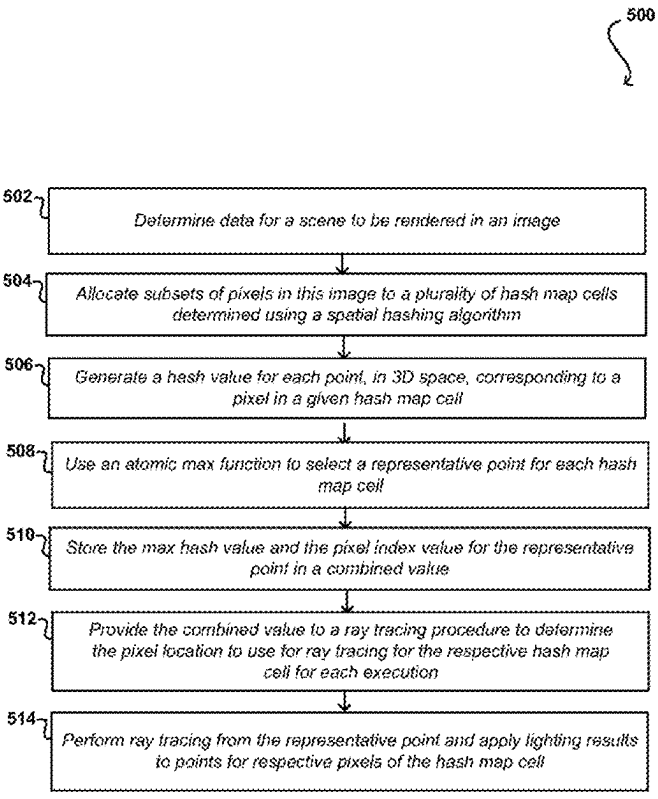
* cited by examiner

Primary Examiner — Ryan McCulley
(74) *Attorney, Agent, or Firm* — Hogan Lovells US LLP

(57) **ABSTRACT**

Approaches presented herein reduce the presence of temporary artifacts such, as flickering, when using spatial hashing with simulation. Flickering can be avoided, at least in part, by ensuring that the same simulation points are utilized in separate executions of an algorithm, even where the execution order can vary. This can involve selecting a single representative point for each hash cell of a spatial hash map, where simulation for that hash cell will be performed for that representative point location, regardless of execution order. Both a location index and a selector value are stored for this hash map entry in a single value, where lower bits store the location index and higher bits represent the selector value. Storing the selector value in the higher-weight bits ensures an atomic maximum operation will primarily consider the selector value, and resort to the location index only in the event of equal selector values.

20 Claims, 17 Drawing Sheets



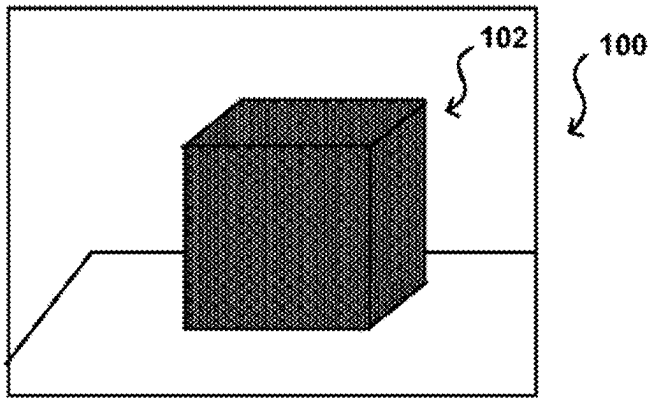


FIG. 1A

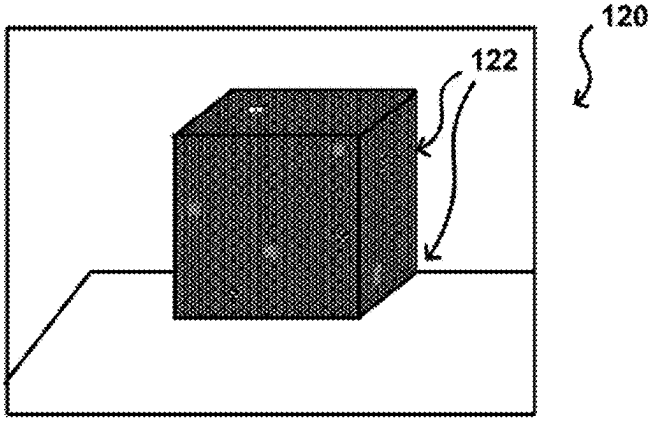


FIG. 1B

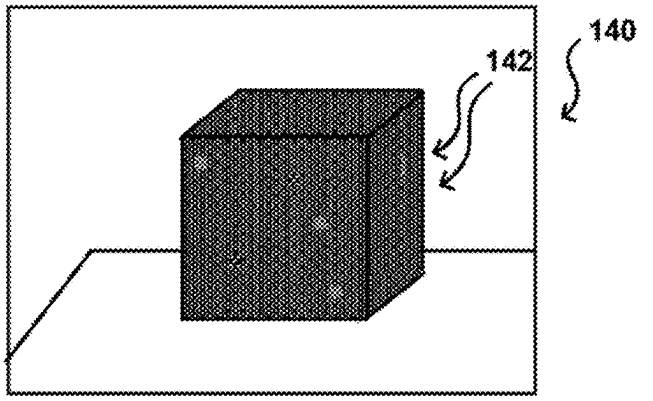


FIG. 1C

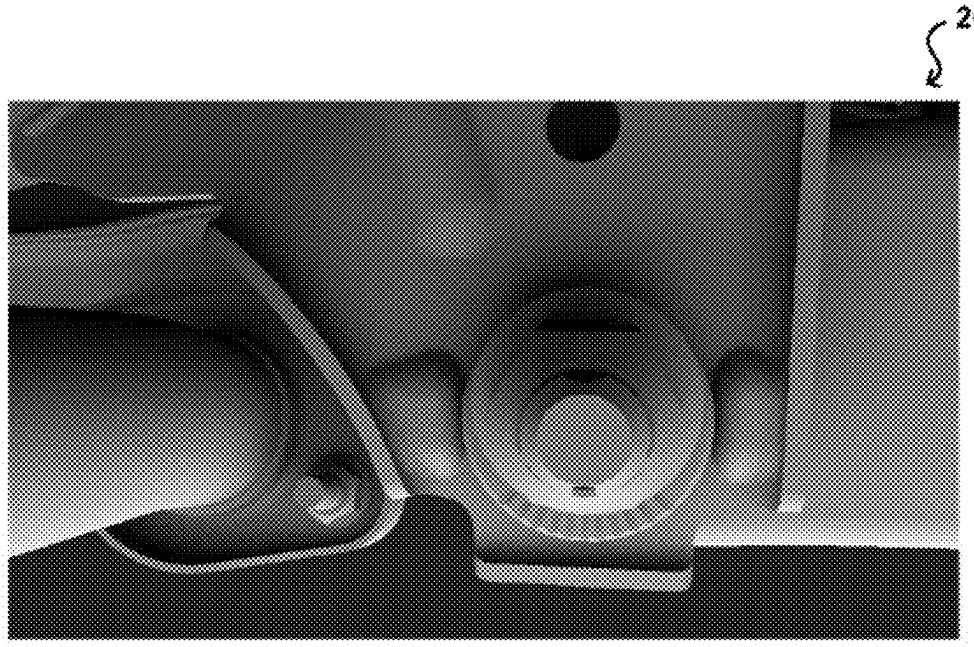


FIG. 2A

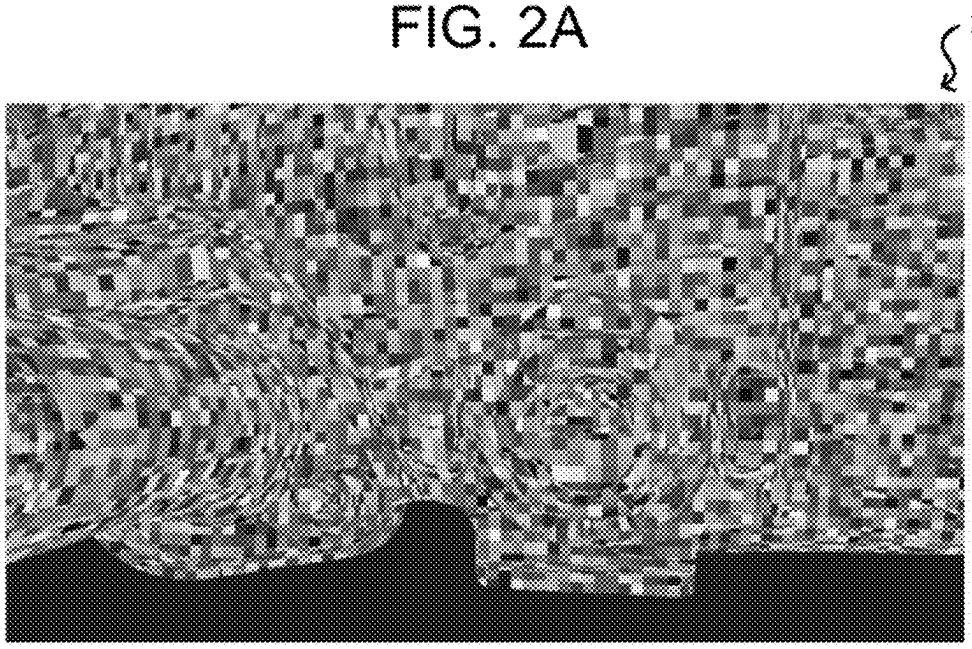


FIG. 2B

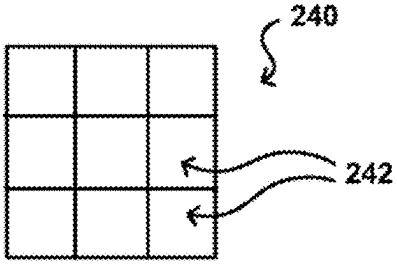


FIG. 2C

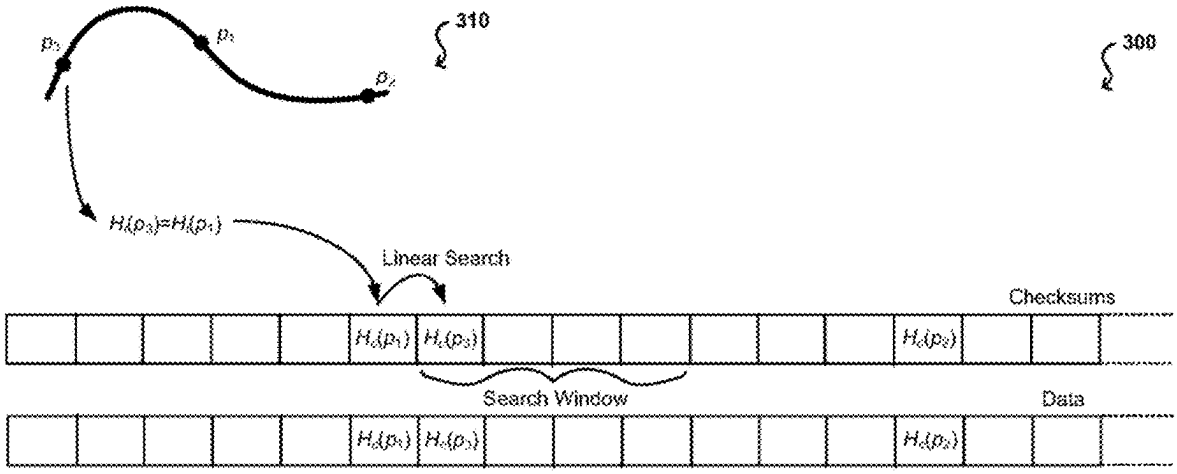

310
300
p_3
p_1
p_2
$H(p_3)=H(p_1)$
Linear Search
Checksums
$H_s(p_1)$
$H_s(p_3)$
$H_s(p_2)$
Search Window
Data
$H_d(p_1)$
$H_d(p_3)$
$H_d(p_2)$

FIG. 3A

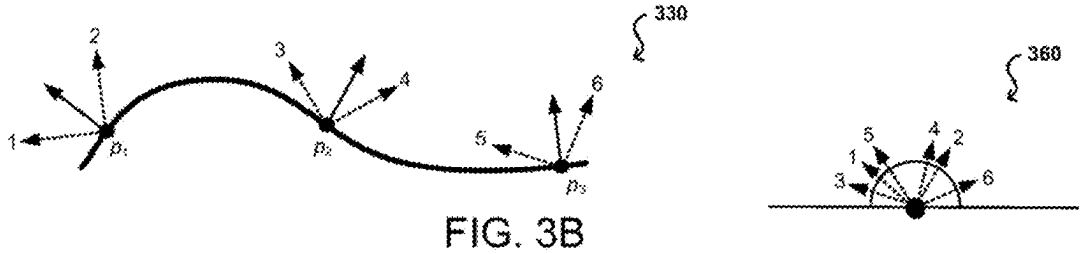

330
360
1
2
3
4
5
6
p_1
p_2
p_3

FIG. 3B

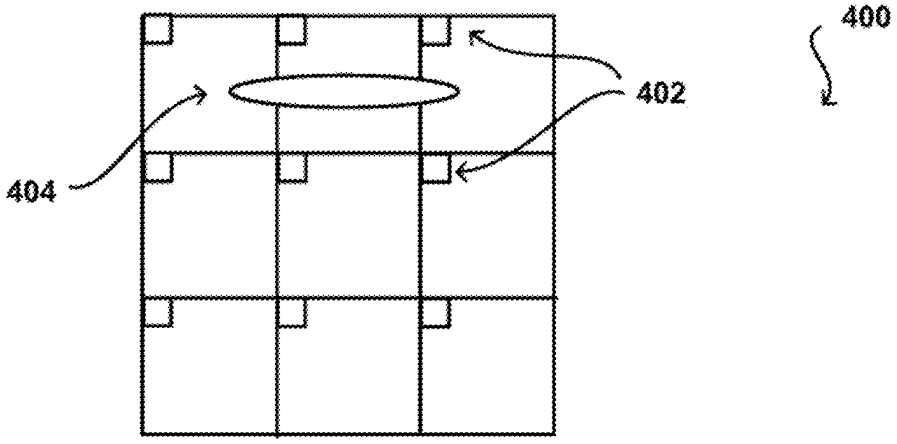

400
402
404

FIG. 4A

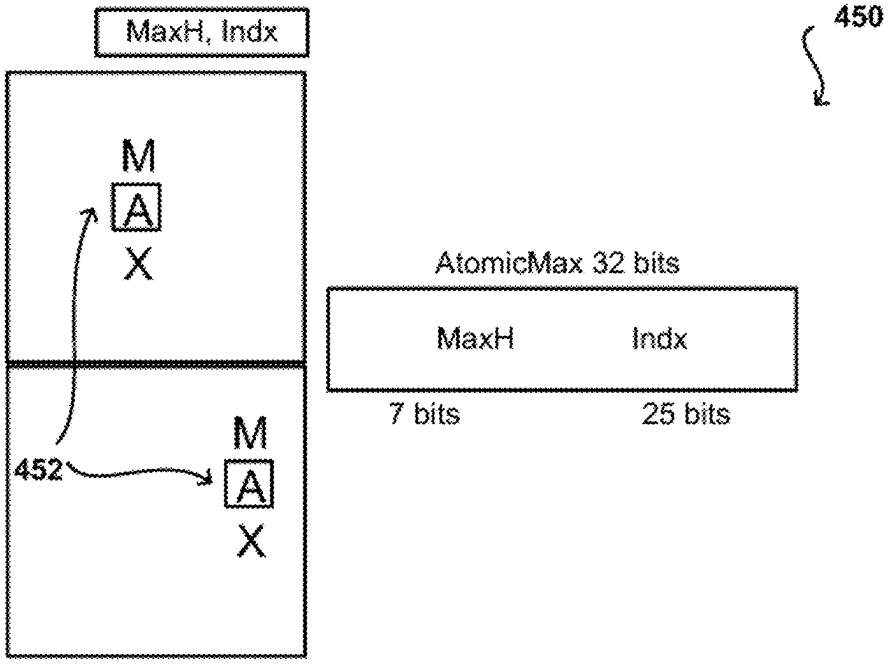

450
MaxH, Indx
M
A
X
AtomicMax 32 bits
MaxH
Indx
7 bits
25 bits
452
M
A
X

FIG. 4B

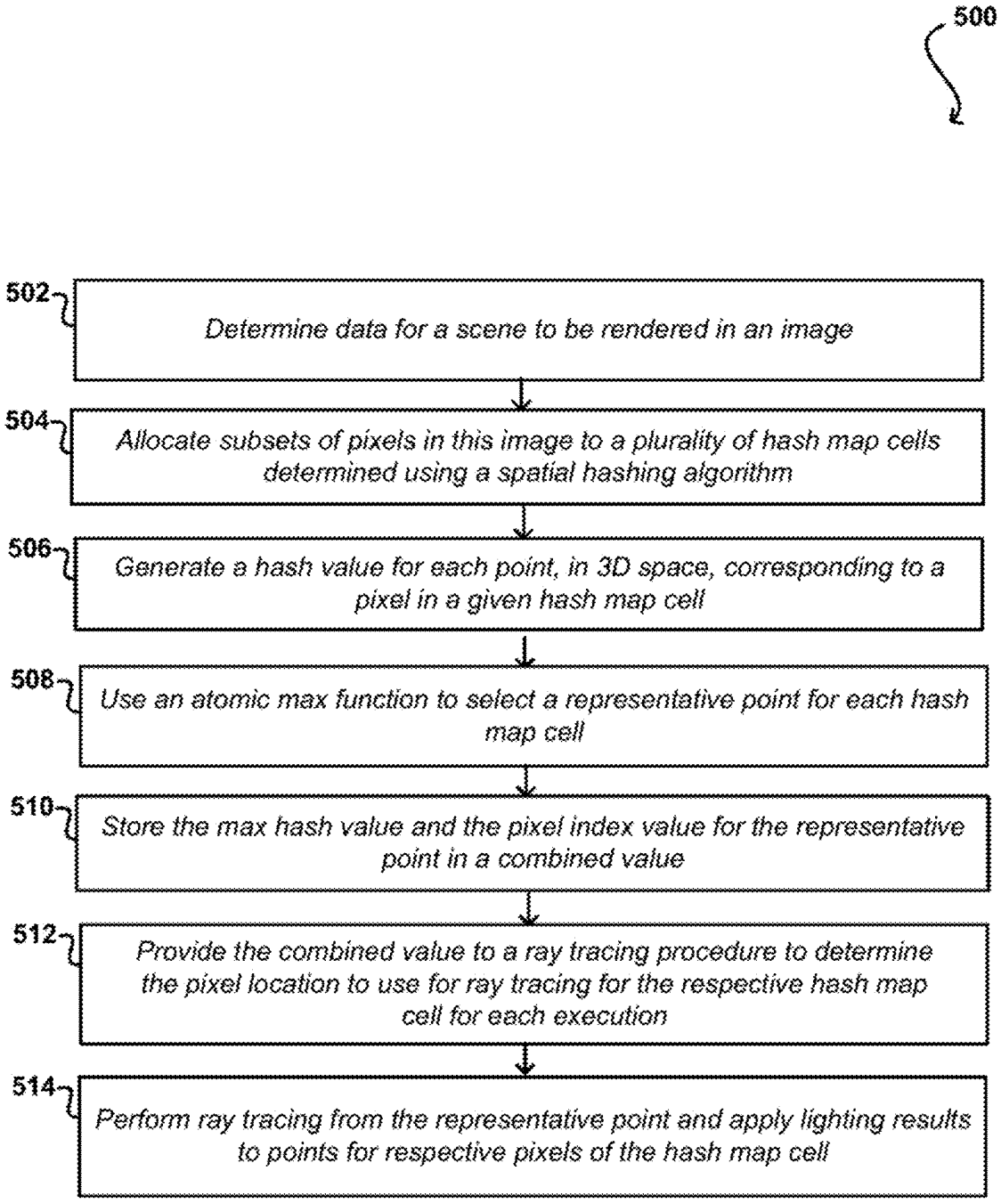

500
502
Determine data for a scene to be rendered in an image
504
Allocate subsets of pixels in this image to a plurality of hash map cells determined using a spatial hashing algorithm
506
Generate a hash value for each point, in 3D space, corresponding to a pixel in a given hash map cell
508
Use an atomic max function to select a representative point for each hash map cell
510
Store the max hash value and the pixel index value for the representative point in a combined value
512
Provide the combined value to a ray tracing procedure to determine the pixel location to use for ray tracing for the respective hash map cell for each execution
514
Perform ray tracing from the representative point and apply lighting results to points for respective pixels of the hash map cell

FIG. 5

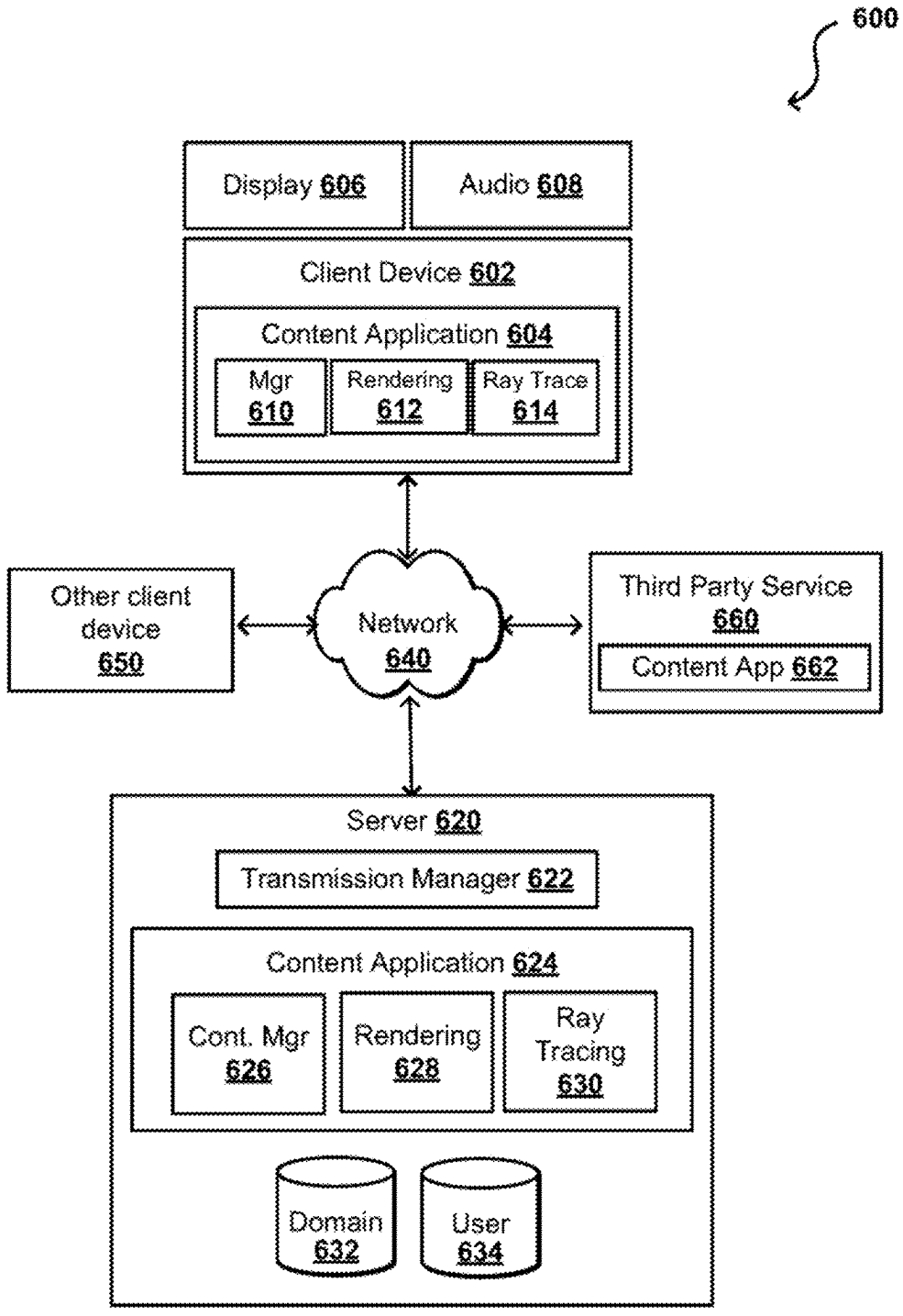

600
Display 606
Audio 608
Client Device 602
Content Application 604
Mgr 610
Rendering 612
Ray Trace 614
Other client device 650
Network 640
Third Party Service 660
Content App 662
Server 620
Transmission Manager 622
Content Application 624
Cont. Mgr 626
Rendering 628
Ray Tracing 630
Domain 632
User 634

FIG. 6

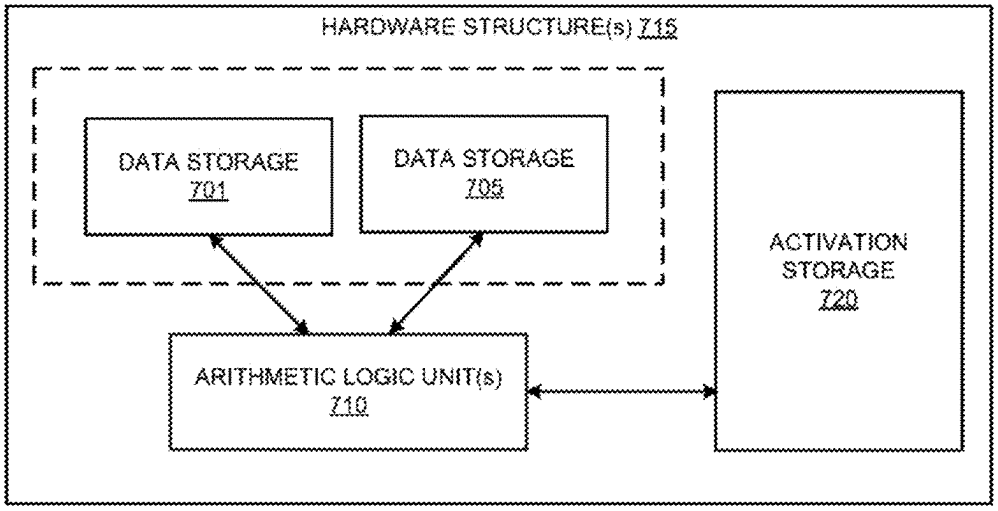

HARDWARE STRUCTURE(s) 715
DATA STORAGE 701
DATA STORAGE 705
ACTIVATION STORAGE 720
ARITHMETIC LOGIC UNIT(s) 710

FIG. 7A

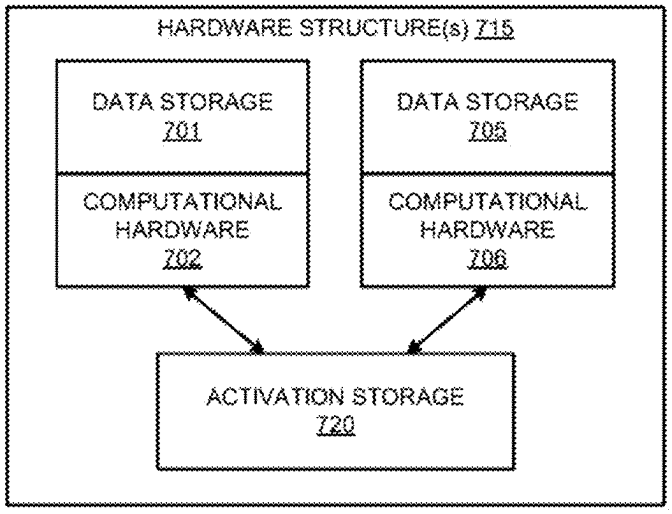

HARDWARE STRUCTURE(s) 715
DATA STORAGE 701
COMPUTATIONAL HARDWARE 702
DATA STORAGE 705
COMPUTATIONAL HARDWARE 706
ACTIVATION STORAGE 720

FIG. 7B

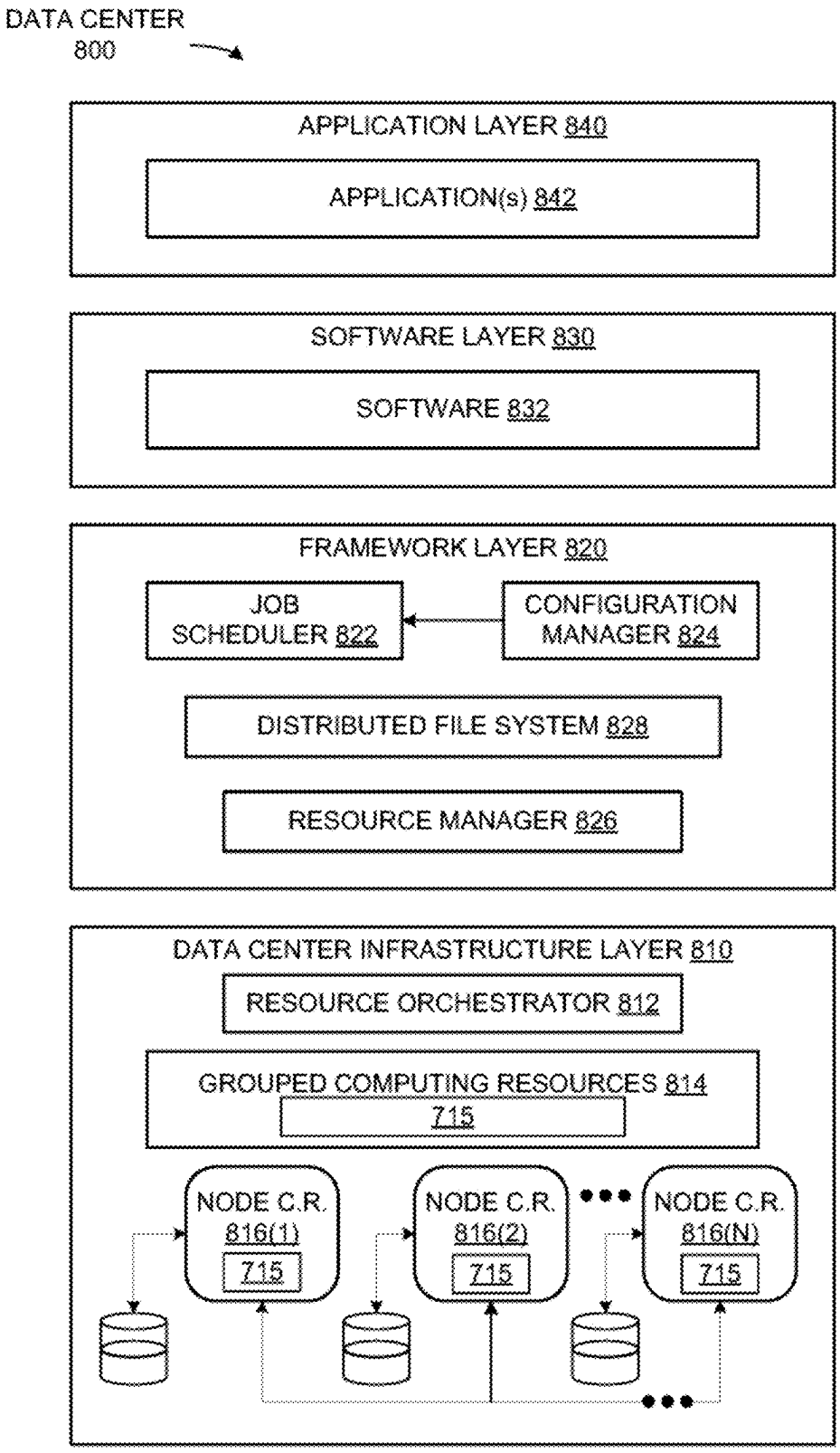

DATA CENTER
800
APPLICATION LAYER 840
APPLICATION(s) 842
SOFTWARE LAYER 830
SOFTWARE 832
FRAMEWORK LAYER 820
JOB SCHEDULER 822
CONFIGURATION MANAGER 824
DISTRIBUTED FILE SYSTEM 828
RESOURCE MANAGER 826
DATA CENTER INFRASTRUCTURE LAYER 810
RESOURCE ORCHESTRATOR 812
GROUPED COMPUTING RESOURCES 814
715
NODE C.R. 816(1)
715
NODE C.R. 816(2)
715
NODE C.R. 816(N)
715

FIG. 8

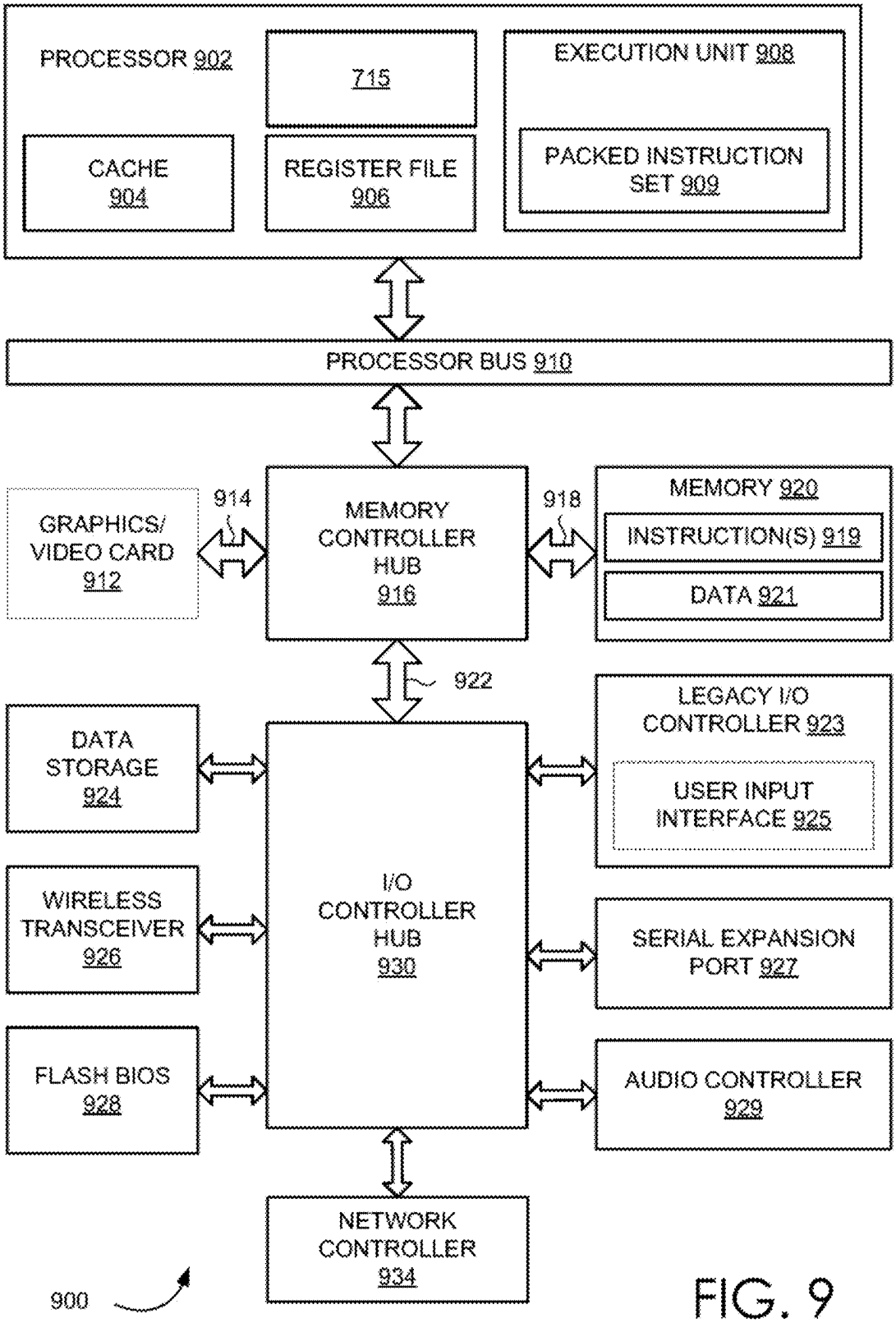

PROCESSOR 902
715
EXECUTION UNIT 908
CACHE 904
REGISTER FILE 906
PACKED INSTRUCTION SET 909
PROCESSOR BUS 910
GRAPHICS/ VIDEO CARD 912
914
MEMORY CONTROLLER HUB 916
918
MEMORY 920
INSTRUCTION(S) 919
DATA 921
922
DATA STORAGE 924
LEGACY I/O CONTROLLER 923
USER INPUT INTERFACE 925
WIRELESS TRANSCEIVER 926
I/O CONTROLLER HUB 930
SERIAL EXPANSION PORT 927
FLASH BIOS 928
AUDIO CONTROLLER 929
NETWORK CONTROLLER 934
900

FIG. 9

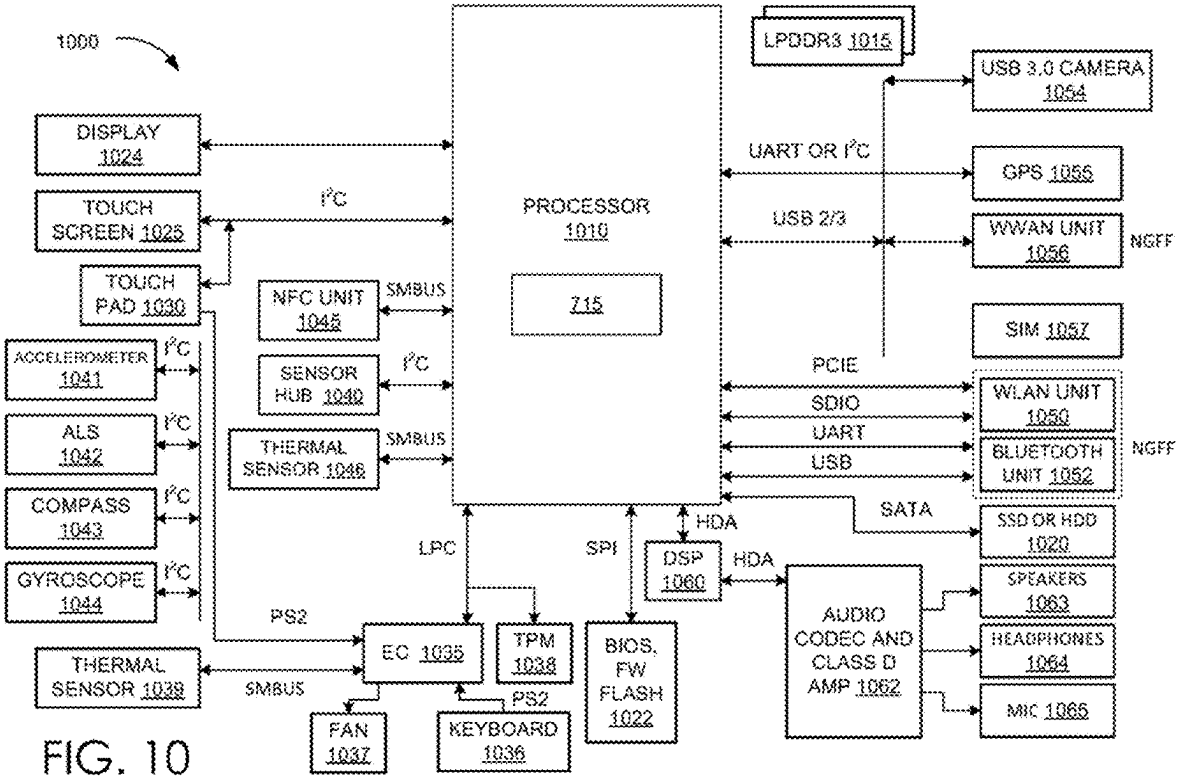

1000
DISPLAY 1024
TOUCH SCREEN 1025
TOUCH PAD 1030
ACCELEROMETER 1041
ALS 1042
COMPASS 1043
GYROSCOPE 1044
THERMAL SENSOR 1039
I²C
PS2
SMBUS
NFC UNIT 1045
SENSOR HUB 1040
THERMAL SENSOR 1046
EC 1035
FAN 1037
KEYBOARD 1036
TPM 1038
PS2
PROCESSOR 1010
715
LPC
SPI
BIOS, FW FLASH 1022
DSP 1060
HDA
HDA
AUDIO CODEC AND CLASS D AMP 1062
LPDDR3 1015
USB 3.0 CAMERA 1054
UART OR I²C
GPS 1055
USB 2/3
WWAN UNIT 1056
NGFF
SIM 1057
PCIE
SDIO
UART
USB
WLAN UNIT 1050
BLUETOOTH UNIT 1052
NGFF
SATA
SSD OR HDD 1020
SPEAKERS 1063
HEADPHONES 1064
MIC 1065

FIG. 10

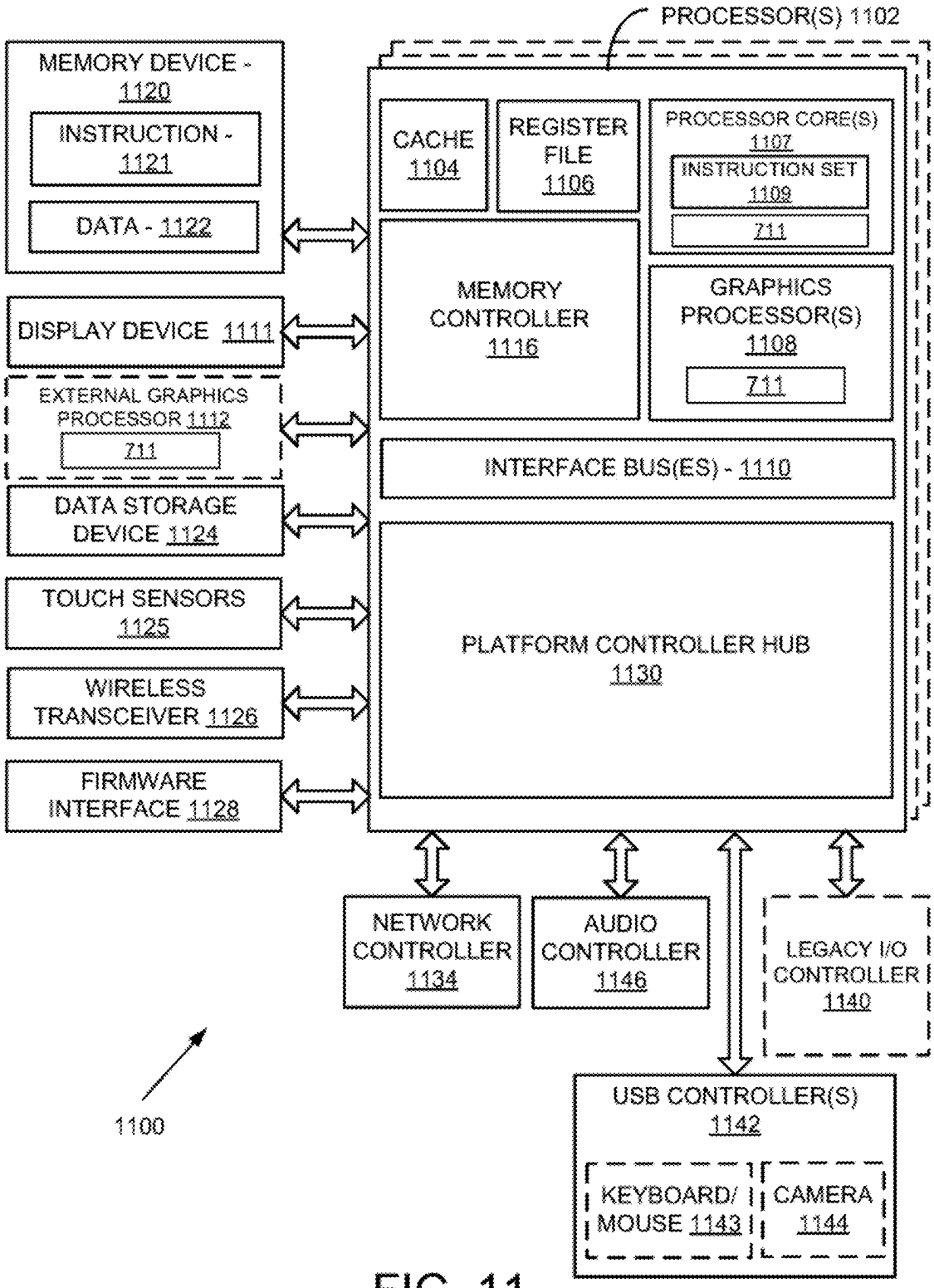

PROCESSOR(S) 1102
MEMORY DEVICE - 1120
INSTRUCTION - 1121
DATA - 1122
CACHE 1104
REGISTER FILE 1106
PROCESSOR CORE(S) 1107
INSTRUCTION SET 1109
711
MEMORY CONTROLLER 1116
GRAPHICS PROCESSOR(S) 1108
711
DISPLAY DEVICE 1111
EXTERNAL GRAPHICS PROCESSOR 1112
711
INTERFACE BUS(ES) - 1110
DATA STORAGE DEVICE 1124
TOUCH SENSORS 1125
PLATFORM CONTROLLER HUB 1130
WIRELESS TRANSCEIVER 1126
FIRMWARE INTERFACE 1128
NETWORK CONTROLLER 1134
AUDIO CONTROLLER 1146
LEGACY I/O CONTROLLER 1140
USB CONTROLLER(S) 1142
KEYBOARD/ MOUSE 1143
CAMERA 1144
1100

FIG. 11

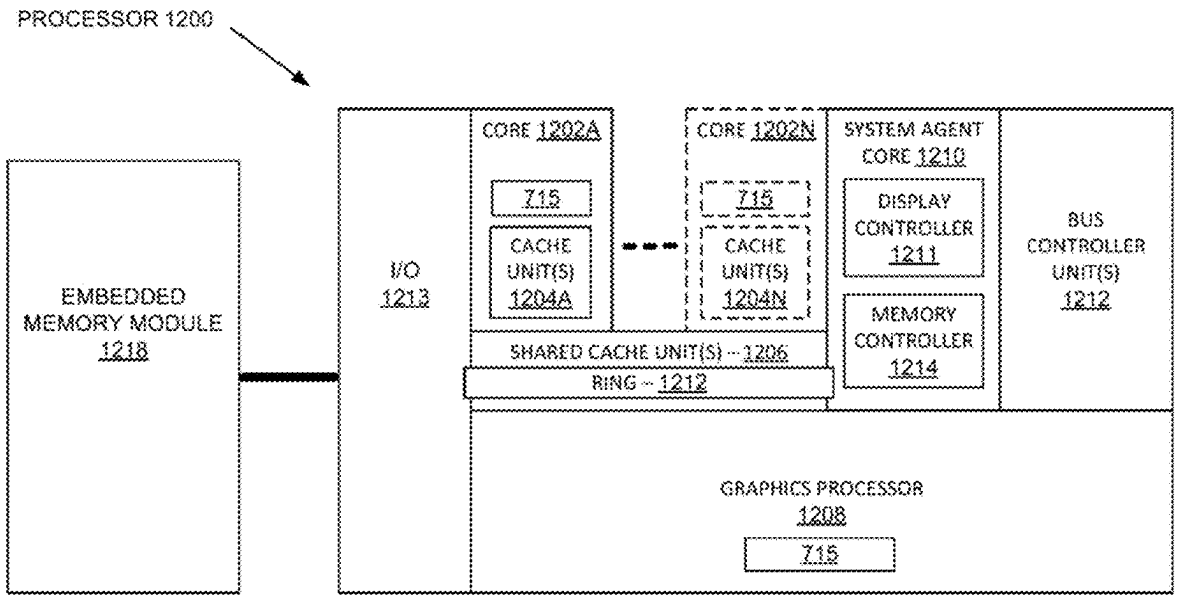

PROCESSOR 1200
EMBEDDED MEMORY MODULE 1218
I/O 1213
CORE 1202A
715
CACHE UNIT(S) 1204A
CORE 1202N
715
CACHE UNIT(S) 1204N
SHARED CACHE UNIT(S) – 1206
RING – 1212
SYSTEM AGENT CORE 1210
DISPLAY CONTROLLER 1211
MEMORY CONTROLLER 1214
BUS CONTROLLER UNIT(S) 1212
GRAPHICS PROCESSOR 1208
715

FIG. 12

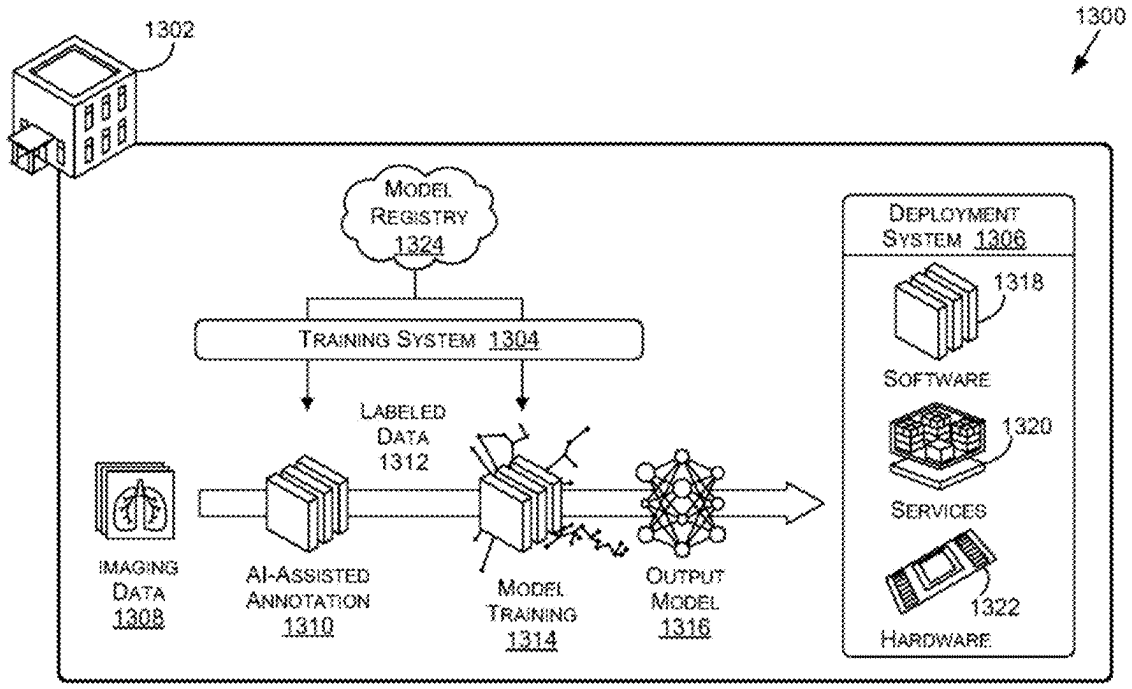

1302
1300
MODEL REGISTRY 1324
TRAINING SYSTEM 1304
LABELED DATA 1312
IMAGING DATA 1308
AI-ASSISTED ANNOTATION 1310
MODEL TRAINING 1314
OUTPUT MODEL 1316
DEPLOYMENT SYSTEM 1306
1318
SOFTWARE
1320
SERVICES
1322
HARDWARE

FIG. 13

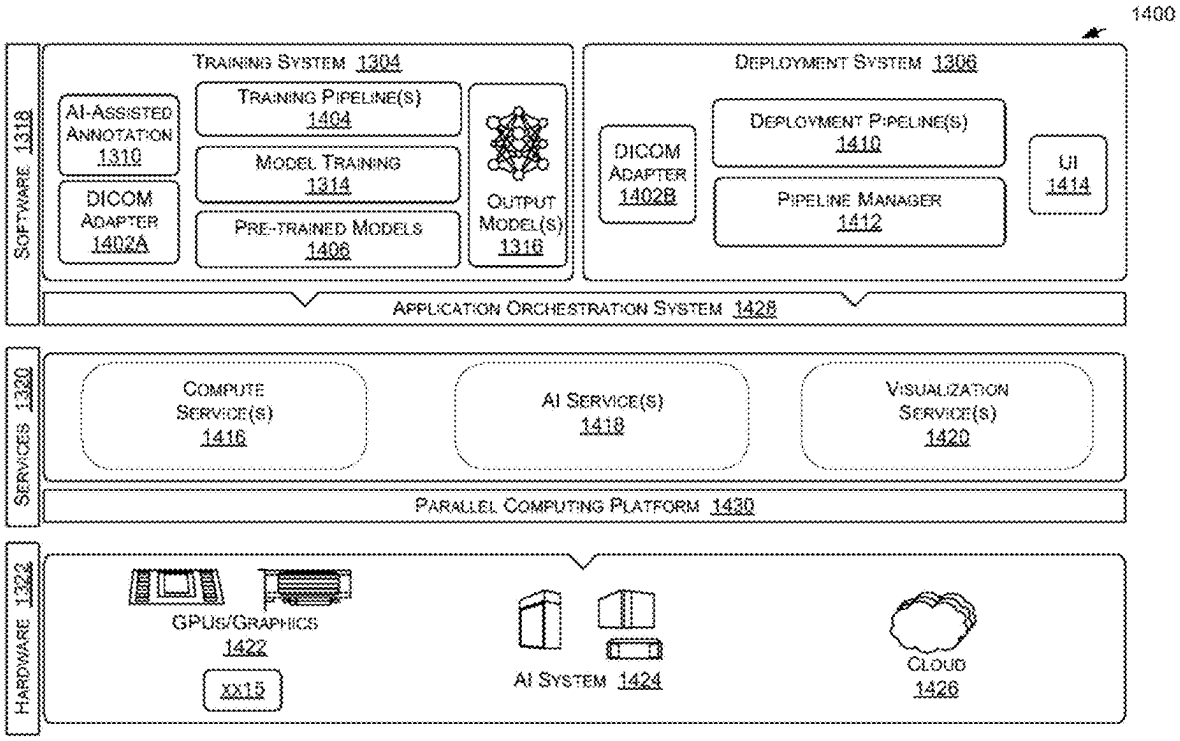

1400
SOFTWARE 1318
TRAINING SYSTEM 1304
AI-ASSISTED ANNOTATION 1310
DICOM ADAPTER 1402A
TRAINING PIPELINE(S) 1404
MODEL TRAINING 1314
PRE-TRAINED MODELS 1408
OUTPUT MODEL(S) 1316
DEPLOYMENT SYSTEM 1306
DICOM ADAPTER 1402B
DEPLOYMENT PIPELINE(S) 1410
PIPELINE MANAGER 1412
UI 1414
APPLICATION ORCHESTRATION SYSTEM 1428
SERVICES 1330
COMPUTE SERVICE(S) 1416
AI SERVICE(S) 1418
VISUALIZATION SERVICE(S) 1420
PARALLEL COMPUTING PLATFORM 1430
HARDWARE 1322
GPUS/GRAPHICS 1422
XX15
AI SYSTEM 1424
CLOUD 1426

FIG. 14

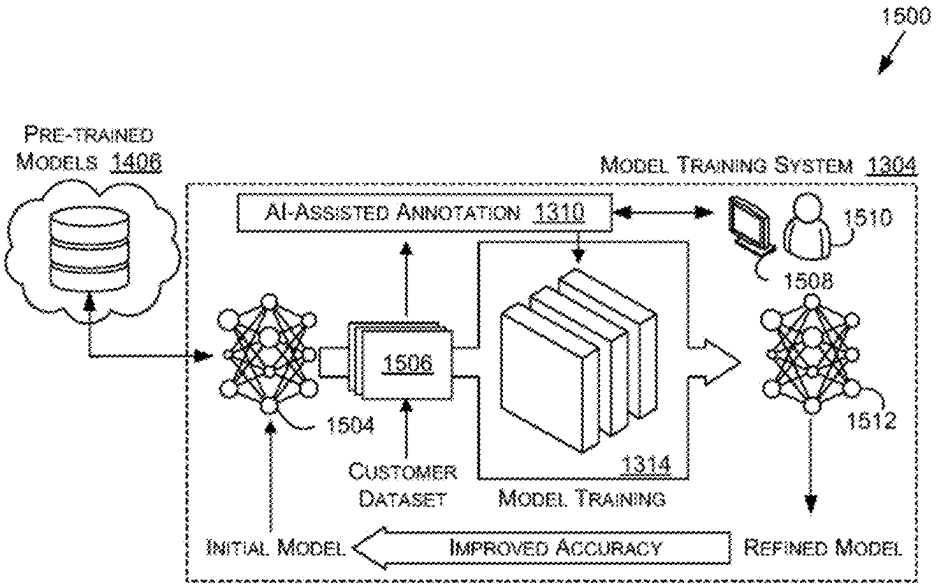

1500
PRE-TRAINED MODELS 1406
MODEL TRAINING SYSTEM 1304
AI-ASSISTED ANNOTATION 1310
1510
1508
1506
1504
1512
1314
CUSTOMER DATASET
MODEL TRAINING
INITIAL MODEL
IMPROVED ACCURACY
REFINED MODEL

FIG. 15A

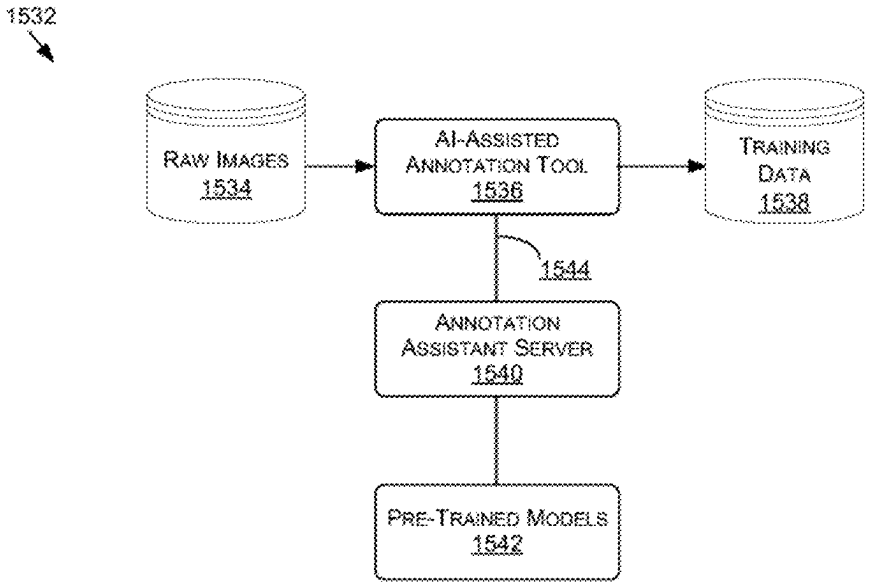

1532
Raw Images
1534
AI-Assisted
Annotation Tool
1536
Training
Data
1538
1544
Annotation
Assistant Server
1540
Pre-Trained Models
1542

FIG. 15B

CONSISTENT SAMPLING FOR SPATIAL HASHING

BACKGROUND

Spatial hashing is a practical tool that can be used to efficiently store and retrieve sparse spatial data in massively parallel environments. Hash functions, by their nature, can spread information relatively uniformly throughout a hash map. In the case of rendering an image, a spatial hash map entry may cover several pixels in a final image to be generated. Each of these pixels may be processed in a separate computational step, with each of these threads adding information to the hash map. As an example, each thread may contribute to a Monte-Carlo estimate of the integral of a hemispherical function by tracing sample rays through the virtual scene. When those samples are taken from a pseudorandom or deterministic sequence, such as quasi-Monte-Carlo (QMC), each thread will cover a segment of that sequence. Depending on the scheduling of these threads, the mapping of the QMC sequence segment to threads may change from one execution to the other due to the different temporal order, as the scheduling may be non-deterministic. Since each pixel corresponds to at least one respective point in a scene to be rendered, such changes between executions often results in a different set of rays being traced, which can lead to temporal artifacts such as image flickering when the same process is executed repeatedly.

BRIEF DESCRIPTION OF THE DRAWINGS

Various embodiments in accordance with the present disclosure will be described with reference to the drawings, in which:

FIGS. **1**A, **1**B, and **1**C illustrate images containing flickering artifacts, in accordance with various embodiments;

FIGS. **2**A, **2**B, and **2**C illustrate hash cells that can be determined for surfaces in an image, in accordance with various embodiments;

FIGS. **3**A and **3**B illustrate an example approach to determining representative points to be used for ray tracing, according to at least one embodiment;

FIGS. **4**A and **4**B illustrate approaches for storing point determination data, according to at least one embodiment;

FIG. **5** illustrates a process for performing ray tracing in image rendering, according to at least one embodiment;

FIG. **6** illustrates components of a distributed system that can be utilized to determine and/or perform tasks, according to at least one embodiment;

FIG. **7**A illustrates inference and/or training logic, according to at least one embodiment;

FIG. **7**B illustrates inference and/or training logic, according to at least one embodiment;

FIG. **8** illustrates an example data center system, according to at least one embodiment;

FIG. **9** illustrates a computer system, according to at least one embodiment;

FIG. **10** illustrates a computer system, according to at least one embodiment;

FIG. **11** illustrates at least portions of a graphics processor, according to one or more embodiments;

FIG. **12** illustrates at least portions of a graphics processor, according to one or more embodiments;

FIG. **13** is an example data flow diagram for an advanced computing pipeline, in accordance with at least one embodiment;

FIG. **14** is a system diagram for an example system for training, adapting, instantiating and deploying machine learning models in an advanced computing pipeline, in accordance with at least one embodiment; and

FIGS. **15**A and **15**B illustrate a data flow diagram for a process to train a machine learning model, as well as client-server architecture to enhance annotation tools with pre-trained annotation models, in accordance with at least one embodiment.

DETAILED DESCRIPTION

In the following description, various embodiments will be described. For purposes of explanation, specific configurations and details are set forth in order to provide a thorough understanding of the embodiments. However, it will also be apparent to one skilled in the art that the embodiments may be practiced without the specific details. Furthermore, well-known features may be omitted or simplified in order not to obscure the embodiment being described.

Approaches in accordance with various embodiments can be used to generate content that is substantially free of at least certain types of artifacts, where this generated content may include one or more images, video, texture maps, augmented reality (AR) or virtual reality (VR) content, or other such two- or three-dimensional content, as well as other types of output such as, for example, one or more light probes. In at least one embodiment, this can include reducing a presence of artifacts, such as flickering, when using spatial hashing with a simulation of light transport, or other forms of computer-generated visuals. Flickering can be avoided, at least in part, by ensuring that the same set of rays is traced in separate executions of an algorithm, even where the execution order can vary. This can involve selecting a single representative point for each of a set of hash cells. A spatial hash map entry may cover a number of pixels, or other points or regions, of an instance of content such as an image. A single point can be selected within an area spanned by that hash map entry, with all rays being traced (or light transport being simulated) from that location, regardless of thread scheduling or computational ordering. Both a pixel index and a selector value can be stored for this hash map entry. A pseudo-random function can be utilized to compute a distinct selector value for each point covered by the hash map entry. The selector value of the entry stores the maximum (or minimum) selector value encountered so far, and the pixel index stores the index of the pixel for which this selector has been obtained. In order to update both the selector and pixel index, both values can be bundled together into a single 32-bit value, where the lower 25 bits store the pixel index and the higher 7 bits represent the selector value. Storing the selector value in the higher-weight bits ensures an atomic Max operation will primarily consider the selector value, and resort to the pixel index only in the event of equal selector values. During simulation or rendering, for example, the respective entry can be located for each thread of execution, the pixel index corresponding to that entry can be determined, and additional spatial information of the virtual scene, such as the surface position and normal at that pixel, can be fetched accordingly. This thread can then atomically reserve a segment of the pseudo-random or deterministic sequence and trace its sample rays. Since all threads for a given entry will trace rays from the same location, the scheduling order becomes irrelevant, and the resulting images will be consistent if re-running the same computational steps in a different order.

Various other such functions can be used as well within the scope of the various embodiments as would be apparent to one of ordinary skill in the art in light of the teachings and suggestions contained herein.

FIGS. **1**A-**1**C illustrate an example of flickering artifacts in a sequence of images or video frames. In a first image **100** in FIG. **1**A, a cube **102** is illustrated that is illuminated by one or more virtual light sources in a scene. As mentioned, light simulation can be used to determine how to light or shade each pixel of this cube, which includes determining a color or pixel value based at least in part upon an estimation of a computed integral, such as for an amount of illumination at that point on the cube. Such a simulation can be used to determine other information as well, as may relate to global illumination, ambient occlusion, shader effects, and the like. It should be understood that a cube-type object may not frequently exhibit flickering as illustrated, but this example is presented for simplicity of explanation. Since it will not be practical in many instances to sample all incoming light rays or to compute the integral analytically for all pixels, particularly for time-sensitive applications such as online gaming, some amount of sampling is typically performed that can serve as a representative measure of an aspect such as illumination, for example, that can then be applied to nearby pixels as well. This can include applying a representative illumination value to pixels in a hash cell of an object. As illustrated in the subsequent images **120**, **140** of FIGS. **1**B and **1**C, however, sampling different rays when generating different images or video frames can result in different amounts of representative illumination being determined, which can cause pixels within different hash cells **122**, **142** to have slightly different shading between frames. This frequent adjustment in color is often referred to as flickering, an effect of which can depend at least in part upon differences in illumination values between different rays that are incident on a given pixel location.

In order to avoid such issues, at least some embodiments presented herein provide the ability to redo the same computation steps, independent of order, in separate executions of an algorithm, as may take advantage of spatial hashing. FIG. **2**A illustrates an image **200** of a portion of an object, here an underside of a vehicle. Spatial hashing can be applied to such an image to generate a representation **220** of this image as illustrated in FIG. **2**B, wherein groups of adjacent cells are apportioned to different hash cells. In one such approach, a spatial hash map entry E can cover n pixels of an image to be generated, such as an array **240** or grid of 3×3 pixels **242**, as illustrated in FIG. **2**C. An efficient way to sample directions involves storing a counter value c that represents the current index in a sequence, such as a quasi-Monte-Carlo (QMC) sequence, used to sample directions above its surface covered by spatial hash map entry E. During the raytracing phase, n threads are launched for entry E, such as one thread for each pixel in that entry. Each thread can reserve a segment of the sampling sequence by, for example, atomically increasing the value of the counter c. Then, each thread can trace the sample rays from the point visible through its corresponding pixel, and add their contribution to the hash map entry E. Use of such an atomic function can ensure the atomicity of the operation, but does not provide any guarantee on the order in which this operation will be executed. If each thread traces 10 rays, then in one execution thread 0 may be the first to be scheduled to issue an atomic addition call. Thread 0 would then change the value of c from 0 to 10, and trace the first 10 samples in the sequence. Thread 1 could trace samples 10 to 19, and so on. In another execution, thread 1 may be scheduled first, and trace rays 0 to 9, while thread 0 would trace rays 10 to 19. Since the rays originate from slightly different locations within the area covered by the entry E, the results will vary from one execution to another, resulting in flickering artifacts in animations Approaches in accordance with various embodiments can attempt to address these and other deficiencies in existing approaches by, for example, efficiently selecting a single representative point for each hash cell. In this way, a parallel sampling process can be used that provides consistent, repeatable results. The providing of such results can be particularly important when the hash map cannot be reused across frames, such as during an animation process.

In at least one embodiment, a single point can be selected, during allocation of entries in the hash map, from within an area spanned by spatial hash map entry E. All computations, such as may involve the tracing of rays based on the spatial position of a single point, can then be traced from a location of that single point, regardless of the thread scheduling. This can be accomplished in at least some embodiments by storing two additional values in the individual hash entries, including a pixel index and a selector value. A trivial way of selecting a single index is to start with a pixel index set to a specific value, and use a simple atomic compare and swap to store the index of the first thread touching that entry. However, this can result in the same non-deterministic issue due to scheduling. It would also be possible, for example, to keep the point closest to the center of the volume spanned by the entry. This, in turn, may result in aliasing artifacts, especially in scenes that feature chaotic or high-frequency content.

An approach in accordance with at least one embodiment can instead leverage a pseudo-random or deterministic function. In such an approach, a distinct value (e.g., a selector value), can be computed for each point covered by the entry, and the index of the pixel with the highest (or smallest) selector value can be selected, such as by using an atomic Max/Min function. For example, this value can be computed using a spatial hashing scheme without discretization. In another example, a pseudo-random generator can be used that provides deterministic results. The selector value of the entry can store the maximum (or minimum) selector value encountered thus far, and the pixel index can be used to store the index of the pixel for which this selector has been obtained.

Atomically updating both the selector and pixel index values may prove challenging on certain graphics hardware. In an attempt to address such challenges, both values can be bundled into a single value. This may be a 32-bit value, for example, where the lower 25 bits are used to store the pixel index, and the higher 7 bits are used to represent the selector value. As an example, 25 bits allows for pixel indices up to 33 megapixels of a 2D image, which can support up to 8K resolution, among other such options. Storing the selector value in the higher-weight bits also helps to ensure that an atomic Min/Max operation will mainly consider the selector value, and resort to the pixel index only in the event of equal selector values. Some embodiments may instead utilize a 64-bit value to embed both the selector and the pixel identifier, at the potential cost of extra memory and processing time.

During the raytracing phase, an individual thread can find its entry and obtain the pixel index corresponding to that entry. This thread can also fetch the surface position and normal at that pixel location. This thread can then atomically reserve a segment of the pseudo-random sequence and trace its sample rays. Since all threads for a given entry will trace

rays from the same location in at least this embodiment, the scheduling order becomes irrelevant and the resulting images should remain consistent. Thus, while prior approaches focus on progressively refining estimates over many frames in a static scene, they do not tackle the particular case of dynamic scenes for which the hash map (or portions thereof) has to be reset on each frame. Approaches presented herein can solve at least this issue for with respect to, for example, real-time rendering of animated scenes using spatial hashing.

An advantage to using spatial hashing is that data can be stored in three-dimensional space instead of storing it within a 2D image. A number of cells, or hash entries, can be generated in a regular hash map where each region covers a region in space. Such an approach can be highly practical for purposes such as determining lighting for a scene, as lighting tends to be relatively smooth over all surfaces in a scene. Information for pixels in a given hash cell can be gathered, combined, and reused to provide a very smooth estimate of the lighting for those pixels, where otherwise there might be very little information per pixel which could result in images with substantial noise or other such artifacts. In the situation **300** of FIG. **3**, a portion **310** of a surface is illustrated that includes a number of points. The positions of these points can be mapped to a three-dimensional (3D) space that generalizes to n dimensions. A hashing function can take the coordinates of these points and generate one-dimensional (1D) values of, for example, 32 bits in length. This one-dimensional value provides a location to store information about that point, such as how much light reaches that point. This can be done for other points as well, limited in part by a number of entries in this particular hash map. In such a map, two points that may be unrelated may share the same hash key. Such an approach can also be used to detect or avoid collisions, as may utilize another hash key calculated on the same input data. This second hash key can be analyzed to determine whether there is a collision, and if so then another location in the hash map can be identified to be used to store the respective data.

A number of different approaches can be utilized to hash points in three dimensions. In at least one embodiment, a first coordinate of a point can be hashed, followed by a second coordinate being added then another hash performed, then a third coordinate and another hashing, which results in a three-dimensional hash value. In at least one embodiment, the position information can also be discretized such that all points in a same cell end up in a same hash map entry. As mentioned, a problem with existing approaches is that lighting can be estimated at different points (e.g., p_1, p_2, p_3) inside a given hash cell, which can lead those points to being treated independently. In at least one embodiment, there can be one thread of execution for each point. There may be a number of samples determined to be taken in a sequence of samples, which for ray tracing may involve a sequence of pseudo-random rays. This may include tracing rays 1, 2 in for a first point, rays 3, 4 for a second point, and rays 5, 6 for a third point. In certain hardware, such as within a GPU, however, there may be no guarantee that the first point will trace rays 1, 2 or the second point will trace rays 3, 4 for different samplings. These tracings can depend upon factors such as a scale needed and aspects of the hardware, such that between two different executions, the application may not end up tracing the same rays for individual points. As mentioned, differences in traced rays can result in artifacts such as flickering when that sequence of images or video frames is presented. The same results may thus not be determined for different frames depending in part upon the scheduling of the rays to be traced. These cells can also flicker independently, which may be visually disturbing for at least some presentations or viewers.

Each of these points can correspond to a different pixel inside a given hash entry, which can cover multiple pixels. Each of the points in a given hash cell can then contribute to the lighting estimate computed for that hash cell. In some applications there may be one such point for each pixel, while in other applications there might be one point for every two or four pixels, among other such options. In at least one embodiment, a number of pixels to be included in a given hash cell can be configurable, as may be set by a user or application, such as to select a 3×3 or 9×9 pixel array. Each pixel in a hash cell can also correspond to an independent thread. Each pixel can pick a number of samples within a sequence of pseudo-random samples. A counter can be associated with a given hash cell, which can start at a default value such as 0. A first thread to execute for a given point may want to trace ten rays, which will then cause this counter to be incremented 0 to 10, with rays 0-9 being traced. A next thread to be executed will atomically increment the counter to 10 and trace samples 10-19, and similarly for a third thread. Each of these threads can refer to a single value in memory, which can be updated using atomic functions. These atomic values allow look ups to be performed for specific values, as well as additions made to those values, in ways in which other threads cannot interfere. When there are several threads running in parallel on the GPU, for example, there may be no guarantee as to the order in which those atomic adds will be executed. There may be no schedule that guarantees thread execution will be scheduled in the same way each time to perform those atomic additions.

Approaches in accordance with various embodiments can attempt to ensure that the same rays are traced for each execution of a given thread. Instead of sampling from all the various points within a given hash cell, an approach can instead select one point from which all rays will be traced for that cell. This can be performed even though different pixels correspond to different points for a given hash cell. An approach can therefore select one point of interest from among the various points within a cell, then trace rays only from that selected point. Each thread may still trace a determined number of rays, but those rays will then all originate from the same point in space. Such an approach may still have at least some amount of randomness in scheduling, but an impact of this randomness is minimized because all rays for a hash cell originate from the same place, such that the same rays can be traced for each execution for a given hash cell.

Various approaches can be used to select a representative point, or point of interest, within a given hash cell. If there were just a single cell, a choice could be made to always select the first point, such as an upper left corner point as illustrated in cell **400** of FIG. **4**A, and all tracing can be performed from that location. A potential problem with such an approach, however, is that taking this same corner point for multiple hash cells across an image can result in some aliasing issues, as there may be certain features that are never sampled which can result in a loss of resolution or image quality. In order to avoid missing small or fine features, for example, it can be desirable in at least some embodiments to provide at least some amount of randomization in the choice of representative points.

In at least one embodiment, an approach to point randomization can be taken that is similar to spatial hashing. A hash value can be computed for each point of interest. This

can be represented by $H_i(p_i)$ as illustrated in FIG. **3**A. There can be a pseudorandom index associated with each point, and a value can be chosen arbitrarily or according to a selection criterion, such as to select a point that has a largest hash value. As illustrated in the example situation **450** of FIG. **4**B, this can correspond to different point locations **452** in each cell being selected as having the maximum hash value. The rays for each respective cell can then be traced from those locations. Such an approach can provide variation that can help to create a more uniform pattern that is more likely to capture information for fine features or details. In at least one embodiment, pseudo-random generator, or other technique that will remain consistent over time and be deterministic, can be utilized in place of a hash function for assigning values to individual points. In at least one embodiment, there are at least two pieces of information that need to be cached (or otherwise stored) for each pixel, including a hash value (or pseudo-random number, etc.) and a pixel index. These values can be used to locate or identify a point of interest. A potential problem, however, is that atomic functions typically only act on small values, as for applications such as DirectX or Vulkan these can only be 32 bit values. If it is determined that p_1 is a maximum value point in a cell, for example, that can be stored atomically using a function such as AtomicMax, as illustrated in FIG. **4**B. This 32 bit AtomicMax value can be used to store two separate values, including the respective hash or pseudo-random value (here, MaxH or the maximum hash value that caused this point to be selected as the point of interest) and the pixel index (here, Indx). In at least one embodiment, this can include using 7 bits for a MaxH value, and 25 bits for an index value for a respective point. Such an approach enables these values to be bundled together, and updated using a single AtomicMax function call. It can be assumed that this 32 bit value with the maximum hash value will be kept, and the AtomicMax function can then operate on the strongest bits of the value. If two points have the same hash, then the process can take the maximum of the two pixel indices as well, which enables having a single atomic call in at least some embodiments. It should be understood that other bit sizes can be used as well, such as 64 bit atomic values which might use a 32 bit number to store both the MaxH and Indx values. In at least some embodiments, however, smaller bit sizes may be preferred to reduce space, time, and processing requirements. A hashing algorithm can be utilized, as such an algorithm can provide values that are relatively random and independent, or will at least be primarily different for each point value and can generate a nice pattern in image space. In at least one embodiment, the same spatial hashing scheme that is used for the hash cell can be applied to the floating point coordinates of an image point of a hash cell. In at least one embodiment, the MaxH value is the selector value, such that it is completely independent of the lighting. Lighting and other information would be stored someplace else in the hash map, with this value enabling determination of a point from which to sample.

In at least one embodiment, a hash cell can have up to 128 pixels. Any more pixels can cause the approach to revert to also using the pixel index to manage colliding values. The pixel index can then be used to determine which pixel or point to select, as a selection process can select the largest pixel index. The size of the hash element itself, as mentioned, can be configurable, although sizes above 10×10 pixels may be too large in some implementations and may end up leaving out too many fine details.

Such an approach can also utilize a component that separates these two values that were blended. A spatial hashing scheme can first compute all cells for an image, then a pass can perform an operation such as to trace rays from the various identified points of those cells. Before tracing any ray, the system can determine which pixel within a given cell is the max pixel, or one with the highest hash or pseudo-random value. The system can then attempt to determine which point in 3D space is viewable through that pixel, and can then trace rays from that pixel. In many implementations, it will be relatively straightforward to look up that value and extract the **25** weaker bits to obtain the index of the pixel of interest.

FIG. **5** illustrates an example process **500** for performing ray tracing for an image to be rendered, which can be performed in accordance with various embodiments. It should be understood that for this and other processes discussed herein, there can be additional, fewer, or alternative steps performed in similar or alternative orders, or at least partially in parallel, within the scope of the various embodiments unless otherwise specifically stated. Further, although this process is described with respect to rendering an image, it should be understood that advantages of consistent sampling with spatial hashing can be utilized advantageously for other applications, content types, or uses as well. In this example, data is determined **502** for a view of a scene to be rendered in an image. This can include, for example, data for one or more objects that are to be at least partially represented in the image. A spatial hashing algorithm can be used to generate a hash map for the image, and subsets of pixels in that image can be allocated **504** to respective hash map cells. In at least one embodiment, each cell can include up to a specified number of pixels or points, such as nine points in a 3×3 grid. Each pixel in a hash cell can correspond to a corresponding point in 3D space, and a hash value or pseudo random number can be generated **506** for each of these points (among other such options discussed and suggested herein). An atomic maximum function can then be used **508** to select a representative point, or point of interest, for each of these hash map cells. This approach enables the point to be selected within a cell relatively randomly, but then consistently selected across multiple executions independent of thread scheduling. The maximum hash value (or other selector value) can then be stored **510**, along with a pixel index value, in a combined value, such as a single 32-bit value. The combined value can be provided **512** to a ray tracing procedure (or hardware, etc.) to determine the pixel location to use for ray tracing for the respective hash map cell for each execution. Ray tracing can then be performed **514** from the representative point and the lighting applied to respective pixels of the hash map cell for use in rendering an image for presentation, or other such purposes.

As discussed, various approaches presented herein are lightweight enough to execute on a client device, such as a personal computer or gaming console, in real time. Such processing can be performed on content that is generated on that client device or received from an external source, such as streaming content received over at least one network. The source can be any appropriate source, such as a game host, streaming media provider, third party content provider, or other client device, among other such options. In some instances, the processing and/or rendering of this content may be performed by one of these other devices, systems, or entities, then provided to the client device (or another such recipient) for presentation or another such use.

As an example, FIG. **6** illustrates an example network configuration **600** that can be used to provide, generate, modify, encode, and/or transmit content. In at least one embodiment, a client device **602** can generate or receive content for a session using components of a content application **604** on client device **602** and data stored locally on that client device. In at least one embodiment, a content application **624** (e.g., an image generation or editing application) executing on content server **620** (e.g., a cloud server or edge server) may initiate a session associated with at least client device **602**, as may utilize a session manager and user data stored in a user database **634**, and can cause content **632** to be determined by a content manager **626**. A content generation or management application **626** can determine data for an image to be generated, which can be passed to a ray tracing component **630** and rendering engine **628**, among other such components, modules, or processes, to generate image data, video frames, or other such content to be transmitted to a client device **602** or other such recipient. At least a portion of that content may be transmitted to client device **602** using an appropriate transmission manager **622** to send by download, streaming, or another such transmission channel. An encoder may be used to encode and/or compress at least some of this data before transmitting to the client device **602**. In at least one embodiment, client device **602** receiving such content can provide this content to a corresponding content application **604**, which may also or alternatively include a content manager **610**, rendering engine **612**, or ray tracing hardware or software **614**. A decoder may also be used to decode data received over the network(s) **640** for presentation via client device **602**, such as image or video content through a display **606** and audio, such as sounds and music, through at least one audio playback device **608**, such as speakers or headphones. In at least one embodiment, at least some of this content may already be stored on, rendered on, or accessible to client device **602** such that transmission over network **640** is not required for at least that portion of content, such as where that content may have been previously downloaded or stored locally on a hard drive or optical disk. In at least one embodiment, a transmission mechanism such as data streaming can be used to transfer this content from server **620**, or user database **634**, to client device **602**. In at least one embodiment, at least a portion of this content can be obtained or streamed from another source, such as a third party service **660** that may also include a content application **662** for generating or providing content. In at least one embodiment, portions of this functionality can be performed using multiple computing devices, or multiple processors within one or more computing devices, such as may include a combination of CPUs and GPUs.

In this example, these client devices can include any appropriate computing devices, as may include a desktop computer, notebook computer, set-top box, streaming device, gaming console, smartphone, tablet computer, VR headset, AR goggles, wearable computer, or a smart television. Each client device can submit a request across at least one wired or wireless network, as may include the Internet, an Ethernet, a local area network (LAN), or a cellular network, among other such options. In this example, these requests can be submitted to an address associated with a cloud provider, who may operate or control one or more electronic resources in a cloud provider environment, such as may include a data center or server farm. In at least one embodiment, the request may be received or processed by at least one edge server, that sits on a network edge and is outside at least one security layer associated with the cloud provider environment. In this way, latency can be reduced by enabling the client devices to interact with servers that are in closer proximity, while also improving security of resources in the cloud provider environment.

In at least one embodiment, such a system can be used for performing graphical rendering operations. In other embodiments, such a system can be used for other purposes, such as for providing image or video content to test or validate autonomous machine applications, or for performing deep learning operations. In at least one embodiment, such a system can be implemented using an edge device, or may incorporate one or more Virtual Machines (VMs). In at least one embodiment, such a system can be implemented at least partially in a data center or at least partially using cloud computing resources.

Inference and Training Logic

FIG. **7**A illustrates inference and/or training logic **715** used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B.

In at least one embodiment, inference and/or training logic **715** may include, without limitation, code and/or data storage **701** to store forward and/or output weight and/or input/output data, and/or other parameters to configure neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one embodiment, training logic **715** may include, or be coupled to code and/or data storage **701** to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs). In at least one embodiment, code, such as graph code, loads weight or other parameter information into processor ALUs based on an architecture of a neural network to which the code corresponds. In at least one embodiment, code and/or data storage **701** stores weight parameters and/or input/output data of each layer of a neural network trained or used in conjunction with one or more embodiments during forward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, any portion of code and/or data storage **701** may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

In at least one embodiment, any portion of code and/or data storage **701** may be internal or external to one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or code and/or data storage **701** may be cache memory, dynamic randomly addressable memory ("DRAM"), static randomly addressable memory ("SRAM"), non-volatile memory (e.g., Flash memory), or other storage. In at least one embodiment, choice of whether code and/or code and/or data storage **701** is internal or external to a processor, for example, or comprised of DRAM, SRAM, Flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

In at least one embodiment, inference and/or training logic **715** may include, without limitation, a code and/or data storage **705** to store backward and/or output weight and/or

input/output data corresponding to neurons or layers of a neural network trained and/or used for inferencing in aspects of one or more embodiments. In at least one embodiment, code and/or data storage **705** stores weight parameters and/or input/output data of each layer of a neural network trained or used in conjunction with one or more embodiments during backward propagation of input/output data and/or weight parameters during training and/or inferencing using aspects of one or more embodiments. In at least one embodiment, training logic **715** may include, or be coupled to code and/or data storage **705** to store graph code or other software to control timing and/or order, in which weight and/or other parameter information is to be loaded to configure, logic, including integer and/or floating point units (collectively, arithmetic logic units (ALUs). In at least one embodiment, code, such as graph code, loads weight or other parameter information into processor ALUs based on an architecture of a neural network to which the code corresponds. In at least one embodiment, any portion of code and/or data storage **705** may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory. In at least one embodiment, any portion of code and/or data storage **705** may be internal or external to on one or more processors or other hardware logic devices or circuits. In at least one embodiment, code and/or data storage **705** may be cache memory, DRAM, SRAM, non-volatile memory (e.g., Flash memory), or other storage. In at least one embodiment, choice of whether code and/or data storage **705** is internal or external to a processor, for example, or comprised of DRAM, SRAM, Flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors.

In at least one embodiment, code and/or data storage **701** and code and/or data storage **705** may be separate storage structures. In at least one embodiment, code and/or data storage **701** and code and/or data storage **705** may be same storage structure. In at least one embodiment, code and/or data storage **701** and code and/or data storage **705** may be partially same storage structure and partially separate storage structures. In at least one embodiment, any portion of code and/or data storage **701** and code and/or data storage **705** may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory.

In at least one embodiment, inference and/or training logic **715** may include, without limitation, one or more arithmetic logic unit(s) ("ALU(s)") **710**, including integer and/or floating point units, to perform logical and/or mathematical operations based, at least in part on, or indicated by, training and/or inference code (e.g., graph code), a result of which may produce activations (e.g., output values from layers or neurons within a neural network) stored in an activation storage **720** that are functions of input/output and/or weight parameter data stored in code and/or data storage **701** and/or code and/or data storage **705**. In at least one embodiment, activations stored in activation storage **720** are generated according to linear algebraic and or matrix-based mathematics performed by ALU(s) **710** in response to performing instructions or other code, wherein weight values stored in code and/or data storage **705** and/or code and/or data storage **701** are used as operands along with other values, such as bias values, gradient information, momentum values, or other parameters or hyperparameters, any or all of which may be stored in code and/or data storage **705** or code and/or data storage **701** or another storage on or off-chip.

In at least one embodiment, ALU(s) **710** are included within one or more processors or other hardware logic devices or circuits, whereas in another embodiment, ALU(s) **710** may be external to a processor or other hardware logic device or circuit that uses them (e.g., a co-processor). In at least one embodiment, ALUs **710** may be included within a processor's execution units or otherwise within a bank of ALUs accessible by a processor's execution units either within same processor or distributed between different processors of different types (e.g., central processing units, graphics processing units, fixed function units, etc.). In at least one embodiment, code and/or data storage **701**, code and/or data storage **705**, and activation storage **720** may be on same processor or other hardware logic device or circuit, whereas in another embodiment, they may be in different processors or other hardware logic devices or circuits, or some combination of same and different processors or other hardware logic devices or circuits. In at least one embodiment, any portion of activation storage **720** may be included with other on-chip or off-chip data storage, including a processor's L1, L2, or L3 cache or system memory. Furthermore, inferencing and/or training code may be stored with other code accessible to a processor or other hardware logic or circuit and fetched and/or processed using a processor's fetch, decode, scheduling, execution, retirement and/or other logical circuits.

In at least one embodiment, activation storage **720** may be cache memory, DRAM, SRAM, non-volatile memory (e.g., Flash memory), or other storage. In at least one embodiment, activation storage **720** may be completely or partially within or external to one or more processors or other logical circuits. In at least one embodiment, choice of whether activation storage **720** is internal or external to a processor, for example, or comprised of DRAM, SRAM, Flash or some other storage type may depend on available storage on-chip versus off-chip, latency requirements of training and/or inferencing functions being performed, batch size of data used in inferencing and/or training of a neural network, or some combination of these factors. In at least one embodiment, inference and/or training logic **715** illustrated in FIG. **7***a* may be used in conjunction with an application-specific integrated circuit ("ASIC"), such as Tensorflow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., "Lake Crest") processor from Intel Corp. In at least one embodiment, inference and/or training logic **715** illustrated in FIG. **7***a* may be used in conjunction with central processing unit ("CPU") hardware, graphics processing unit ("GPU") hardware or other hardware, such as field programmable gate arrays ("FPGAs").

FIG. **7***b* illustrates inference and/or training logic **715**, according to at least one or more embodiments. In at least one embodiment, inference and/or training logic **715** may include, without limitation, hardware logic in which computational resources are dedicated or otherwise exclusively used in conjunction with weight values or other information corresponding to one or more layers of neurons within a neural network. In at least one embodiment, inference and/or training logic **715** illustrated in FIG. **7***b* may be used in conjunction with an application-specific integrated circuit (ASIC), such as Tensorflow® Processing Unit from Google, an inference processing unit (IPU) from Graphcore™, or a Nervana® (e.g., "Lake Crest") processor from Intel Corp. In at least one embodiment, inference and/or training logic **715**

illustrated in FIG. **7***b* may be used in conjunction with central processing unit (CPU) hardware, graphics processing unit (GPU) hardware or other hardware, such as field programmable gate arrays (FPGAs). In at least one embodiment, inference and/or training logic **715** includes, without limitation, code and/or data storage **701** and code and/or data storage **705**, which may be used to store code (e.g., graph code), weight values and/or other information, including bias values, gradient information, momentum values, and/or other parameter or hyperparameter information. In at least one embodiment illustrated in FIG. **7***b*, each of code and/or data storage **701** and code and/or data storage **705** is associated with a dedicated computational resource, such as computational hardware **702** and computational hardware **706**, respectively. In at least one embodiment, each of computational hardware **702** and computational hardware **706** comprises one or more ALUs that perform mathematical functions, such as linear algebraic functions, only on information stored in code and/or data storage **701** and code and/or data storage **705**, respectively, result of which is stored in activation storage **720**.

In at least one embodiment, each of code and/or data storage **701** and **705** and corresponding computational hardware **702** and **706**, respectively, correspond to different layers of a neural network, such that resulting activation from one "storage/computational pair **701**/**702**" of code and/or data storage **701** and computational hardware **702** is provided as an input to "storage/computational pair **705**/**706**" of code and/or data storage **705** and computational hardware **706**, in order to mirror conceptual organization of a neural network. In at least one embodiment, each of storage/computational pairs **701**/**702** and **705**/**706** may correspond to more than one neural network layer. In at least one embodiment, additional storage/computation pairs (not shown) subsequent to or in parallel with storage computation pairs **701**/**702** and **705**/**706** may be included in inference and/or training logic **715**.

Data Center

FIG. **8** illustrates an example data center **800**, in which at least one embodiment may be used. In at least one embodiment, data center **800** includes a data center infrastructure layer **810**, a framework layer **820**, a software layer **830**, and an application layer **840**.

In at least one embodiment, as shown in FIG. **8**, data center infrastructure layer **810** may include a resource orchestrator **812**, grouped computing resources **814**, and node computing resources ("node C.R.s") **816**(**1**)-**816**(N), where "N" represents any whole, positive integer. In at least one embodiment, node C.R.s **816**(**1**)-**816**(N) may include, but are not limited to, any number of central processing units ("CPUs") or other processors (including accelerators, field programmable gate arrays (FPGAs), graphics processors, etc.), memory devices (e.g., dynamic read-only memory), storage devices (e.g., solid state or disk drives), network input/output ("NW I/O") devices, network switches, virtual machines ("VMs"), power modules, and cooling modules, etc. In at least one embodiment, one or more node C.R.s from among node C.R.s **816**(**1**)-**816**(N) may be a server having one or more of above-mentioned computing resources.

In at least one embodiment, grouped computing resources **814** may include separate groupings of node C.R.s housed within one or more racks (not shown), or many racks housed in data centers at various geographical locations (also not shown). Separate groupings of node C.R.s within grouped computing resources **814** may include grouped compute, network, memory or storage resources that may be configured or allocated to support one or more workloads. In at least one embodiment, several node C.R.s including CPUs or processors may grouped within one or more racks to provide compute resources to support one or more workloads. In at least one embodiment, one or more racks may also include any number of power modules, cooling modules, and network switches, in any combination.

In at least one embodiment, resource orchestrator **812** may configure or otherwise control one or more node C.R.s **816**(**1**)-**816**(N) and/or grouped computing resources **814**. In at least one embodiment, resource orchestrator **812** may include a software design infrastructure ("SDI") management entity for data center **800**. In at least one embodiment, resource orchestrator may include hardware, software or some combination thereof.

In at least one embodiment, as shown in FIG. **8**, framework layer **820** includes a job scheduler **822**, a configuration manager **824**, a resource manager **826** and a distributed file system **828**. In at least one embodiment, framework layer **820** may include a framework to support software **832** of software layer **830** and/or one or more application(s) **842** of application layer **840**. In at least one embodiment, software **832** or application(s) **842** may respectively include web-based service software or applications, such as those provided by Amazon Web Services, Google Cloud and Microsoft Azure. In at least one embodiment, framework layer **820** may be, but is not limited to, a type of free and open-source software web application framework such as Apache Spark™ (hereinafter "Spark") that may utilize distributed file system **828** for large-scale data processing (e.g., "big data"). In at least one embodiment, job scheduler **822** may include a Spark driver to facilitate scheduling of workloads supported by various layers of data center **800**. In at least one embodiment, configuration manager **824** may be capable of configuring different layers such as software layer **830** and framework layer **820** including Spark and distributed file system **828** for supporting large-scale data processing. In at least one embodiment, resource manager **826** may be capable of managing clustered or grouped computing resources mapped to or allocated for support of distributed file system **828** and job scheduler **822**. In at least one embodiment, clustered or grouped computing resources may include grouped computing resource **814** at data center infrastructure layer **810**. In at least one embodiment, resource manager **826** may coordinate with resource orchestrator **812** to manage these mapped or allocated computing resources.

In at least one embodiment, software **832** included in software layer **830** may include software used by at least portions of node C.R.s **816**(**1**)-**816**(N), grouped computing resources **814**, and/or distributed file system **828** of framework layer **820**. The one or more types of software may include, but are not limited to, Internet web page search software, e-mail virus scan software, database software, and streaming video content software.

In at least one embodiment, application(s) **842** included in application layer **840** may include one or more types of applications used by at least portions of node C.R.s **816**(**1**)-**816**(N), grouped computing resources **814**, and/or distributed file system **828** of framework layer **820**. One or more types of applications may include, but are not limited to, any number of a genomics application, a cognitive compute, and a machine learning application, including training or inferencing software, machine learning framework software

(e.g., PyTorch, TensorFlow, Caffe, etc.) or other machine learning applications used in conjunction with one or more embodiments.

In at least one embodiment, any of configuration manager **824**, resource manager **826**, and resource orchestrator **812** may implement any number and type of self-modifying actions based on any amount and type of data acquired in any technically feasible fashion. In at least one embodiment, self-modifying actions may relieve a data center operator of data center **800** from making possibly bad configuration decisions and possibly avoiding underutilized and/or poor performing portions of a data center.

In at least one embodiment, data center **800** may include tools, services, software or other resources to train one or more machine learning models or predict or infer information using one or more machine learning models according to one or more embodiments described herein. For example, in at least one embodiment, a machine learning model may be trained by calculating weight parameters according to a neural network architecture using software and computing resources described above with respect to data center **800**. In at least one embodiment, trained machine learning models corresponding to one or more neural networks may be used to infer or predict information using resources described above with respect to data center **800** by using weight parameters calculated through one or more training techniques described herein.

In at least one embodiment, data center may use CPUs, application-specific integrated circuits (ASICs), GPUs, FPGAs, or other hardware to perform training and/or inferencing using above-described resources. Moreover, one or more software and/or hardware resources described above may be configured as a service to allow users to train or performing inferencing of information, such as image recognition, speech recognition, or other artificial intelligence services.

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B. In at least one embodiment, inference and/or training logic **715** may be used in system FIG. **8** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

Computer Systems

FIG. **9** is a block diagram illustrating an exemplary computer system, which may be a system with interconnected devices and components, a system-on-a-chip (SOC) or some combination thereof **900** formed with a processor that may include execution units to execute an instruction, according to at least one embodiment. In at least one embodiment, computer system **900** may include, without limitation, a component, such as a processor **902** to employ execution units including logic to perform algorithms for process data, in accordance with present disclosure, such as in embodiment described herein. In at least one embodiment, computer system **900** may include processors, such as PENTIUM® Processor family, Xeon™, Itanium®, XScale™ and/or StrongARM™, Intel® Core™, or Intel® Nervana™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and like) may also be used. In at least one embodiment, computer system **900** may execute a version of WINDOWS' operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems (UNIX and Linux for example), embedded software, and/or graphical user interfaces, may also be used.

Embodiments may be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants ("PDAs"), and handheld PCs. In at least one embodiment, embedded applications may include a microcontroller, a digital signal processor ("DSP"), system on a chip, network computers ("NetPCs"), set-top boxes, network hubs, wide area network ("WAN") switches, or any other system that may perform one or more instructions in accordance with at least one embodiment.

In at least one embodiment, computer system **900** may include, without limitation, processor **902** that may include, without limitation, one or more execution units **908** to perform machine learning model training and/or inferencing according to techniques described herein. In at least one embodiment, computer system **900** is a single processor desktop or server system, but in another embodiment computer system **900** may be a multiprocessor system. In at least one embodiment, processor **902** may include, without limitation, a complex instruction set computer ("CISC") microprocessor, a reduced instruction set computing ("RISC") microprocessor, a very long instruction word ("VLIW") microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. In at least one embodiment, processor **902** may be coupled to a processor bus **910** that may transmit data signals between processor **902** and other components in computer system **900**.

In at least one embodiment, processor **902** may include, without limitation, a Level 1 ("L1") internal cache memory ("cache") **904**. In at least one embodiment, processor **902** may have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory may reside external to processor **902**. Other embodiments may also include a combination of both internal and external caches depending on particular implementation and needs. In at least one embodiment, register file **906** may store different types of data in various registers including, without limitation, integer registers, floating point registers, status registers, and instruction pointer register.

In at least one embodiment, execution unit **908**, including, without limitation, logic to perform integer and floating point operations, also resides in processor **902**. In at least one embodiment, processor **902** may also include a microcode ("ucode") read only memory ("ROM") that stores microcode for certain macro instructions. In at least one embodiment, execution unit **908** may include logic to handle a packed instruction set **909**. In at least one embodiment, by including packed instruction set **909** in an instruction set of a general-purpose processor **902**, along with associated circuitry to execute instructions, operations used by many multimedia applications may be performed using packed data in a general-purpose processor **902**. In one or more embodiments, many multimedia applications may be accelerated and executed more efficiently by using full width of a processor's data bus for performing operations on packed data, which may eliminate need to transfer smaller units of

data across processor's data bus to perform one or more operations one data element at a time.

In at least one embodiment, execution unit **908** may also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. In at least one embodiment, computer system **900** may include, without limitation, a memory **920**. In at least one embodiment, memory **920** may be implemented as a Dynamic Random Access Memory ("DRAM") device, a Static Random Access Memory ("SRAM") device, flash memory device, or other memory device. In at least one embodiment, memory **920** may store instruction(s) **919** and/or data **921** represented by data signals that may be executed by processor **902**.

In at least one embodiment, system logic chip may be coupled to processor bus **910** and memory **920**. In at least one embodiment, system logic chip may include, without limitation, a memory controller hub ("MCH") **916**, and processor **902** may communicate with MCH **916** via processor bus **910**. In at least one embodiment, MCH **916** may provide a high bandwidth memory path **918** to memory **920** for instruction and data storage and for storage of graphics commands, data and textures. In at least one embodiment, MCH **916** may direct data signals between processor **902**, memory **920**, and other components in computer system **900** and to bridge data signals between processor bus **910**, memory **920**, and a system I/O **922**. In at least one embodiment, system logic chip may provide a graphics port for coupling to a graphics controller. In at least one embodiment, MCH **916** may be coupled to memory **920** through a high bandwidth memory path **918** and graphics/video card **912** may be coupled to MCH **916** through an Accelerated Graphics Port ("AGP") interconnect **914**.

In at least one embodiment, computer system **900** may use system I/O **922** that is a proprietary hub interface bus to couple MCH **916** to I/O controller hub ("ICH") **930**. In at least one embodiment, ICH **930** may provide direct connections to some I/O devices via a local I/O bus. In at least one embodiment, local I/O bus may include, without limitation, a high-speed I/O bus for connecting peripherals to memory **920**, chipset, and processor **902**. Examples may include, without limitation, an audio controller **929**, a firmware hub ("flash BIOS") **928**, a wireless transceiver **926**, a data storage **924**, a legacy I/O controller **923** containing user input and keyboard interfaces **925**, a serial expansion port **927**, such as Universal Serial Bus ("USB"), and a network controller **934**. Data storage **924** may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

In at least one embodiment, FIG. **9** illustrates a system, which includes interconnected hardware devices or "chips", whereas in other embodiments, FIG. **9** may illustrate an exemplary System on a Chip ("SoC"). In at least one embodiment, devices may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of computer system **900** are interconnected using compute express link (CXL) interconnects.

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B. In at least one embodiment, inference and/or training logic **715** may be used in system FIG. **9** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

FIG. **10** is a block diagram illustrating an electronic device **1000** for utilizing a processor **1010**, according to at least one embodiment. In at least one embodiment, electronic device **1000** may be, for example and without limitation, a notebook, a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

In at least one embodiment, system **1000** may include, without limitation, processor **1010** communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. In at least one embodiment, processor **1010** coupled using a bus or interface, such as a 1° C. bus, a System Management Bus ("SMBus"), a Low Pin Count (LPC) bus, a Serial Peripheral Interface ("SPI"), a High Definition Audio ("HDA") bus, a Serial Advance Technology Attachment ("SATA") bus, a Universal Serial Bus ("USB") (versions 1, 2, 3), or a Universal Asynchronous Receiver/Transmitter ("UART") bus. In at least one embodiment, FIG. **10** illustrates a system, which includes interconnected hardware devices or "chips", whereas in other embodiments, FIG. **10** may illustrate an exemplary System on a Chip ("SoC"). In at least one embodiment, devices illustrated in FIG. **10** may be interconnected with proprietary interconnects, standardized interconnects (e.g., PCIe) or some combination thereof. In at least one embodiment, one or more components of FIG. **10** are interconnected using compute express link (CXL) interconnects.

In at least one embodiment, FIG. **10** may include a display **1024**, a touch screen **1025**, a touch pad **1030**, a Near Field Communications unit ("NFC") **1045**, a sensor hub **1040**, a thermal sensor **1046**, an Express Chipset ("EC") **1035**, a Trusted Platform Module ("TPM") **1038**, BIOS/firmware/flash memory ("BIOS, FW Flash") **1022**, a DSP **1060**, a drive **1020** such as a Solid State Disk ("SSD") or a Hard Disk Drive ("HDD"), a wireless local area network unit ("WLAN") **1050**, a Bluetooth unit **1052**, a Wireless Wide Area Network unit ("WWAN") **1056**, a Global Positioning System (GPS) **1055**, a camera ("USB 3.0 camera") **1054** such as a USB 3.0 camera, and/or a Low Power Double Data Rate ("LPDDR") memory unit ("LPDDR3") **1015** implemented in, for example, LPDDR3 standard. These components may each be implemented in any suitable manner.

In at least one embodiment, other components may be communicatively coupled to processor **1010** through components discussed above. In at least one embodiment, an accelerometer **1041**, Ambient Light Sensor ("ALS") **1042**, compass **1043**, and a gyroscope **1044** may be communicatively coupled to sensor hub **1040**. In at least one embodiment, thermal sensor **1039**, a fan **1037**, a keyboard **1046**, and a touch pad **1030** may be communicatively coupled to EC **1035**. In at least one embodiment, speaker **1063**, headphones **1064**, and microphone ("mic") **1065** may be communicatively coupled to an audio unit ("audio codec and class d amp") **1062**, which may in turn be communicatively coupled to DSP **1060**. In at least one embodiment, audio unit **1064** may include, for example and without limitation, an audio coder/decoder ("codec") and a class D amplifier. In at least one embodiment, SIM card ("SIM") **1057** may be communicatively coupled to WWAN unit **1056**. In at least one embodiment, components such as WLAN unit **1050** and

Bluetooth unit **1052**, as well as WWAN unit **1056** may be implemented in a Next Generation Form Factor ("NGFF").

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7***a* and/or **7***b*. In at least one embodiment, inference and/or training logic **715** may be used in system FIG. **10** for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

FIG. **11** is a block diagram of a processing system, according to at least one embodiment. In at least one embodiment, system **1100** includes one or more processors **1102** and one or more graphics processors **1108**, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors **1102** or processor cores **1107**. In at least one embodiment, system **1100** is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

In at least one embodiment, system **1100** can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming console, a handheld game console, or an online game console. In at least one embodiment, system **1100** is a mobile phone, smart phone, tablet computing device or mobile Internet device. In at least one embodiment, processing system **1100** can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In at least one embodiment, processing system **1100** is a television or set top box device having one or more processors **1102** and a graphical interface generated by one or more graphics processors **1108**.

In at least one embodiment, one or more processors **1102** each include one or more processor cores **1107** to process instructions which, when executed, perform operations for system and user software. In at least one embodiment, each of one or more processor cores **1107** is configured to process a specific instruction set **1109**. In at least one embodiment, instruction set **1109** may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). In at least one embodiment, processor cores **1107** may each process a different instruction set **1109**, which may include instructions to facilitate emulation of other instruction sets. In at least one embodiment, processor core **1107** may also include other processing devices, such a Digital Signal Processor (DSP).

In at least one embodiment, processor **1102** includes cache memory **1104**. In at least one embodiment, processor **1102** can have a single internal cache or multiple levels of internal cache. In at least one embodiment, cache memory is shared among various components of processor **1102**. In at least one embodiment, processor **1102** also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores **1107** using known cache coherency techniques. In at least one embodiment, register file **1106** is additionally included in processor **1102** which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). In at least one embodiment, register file **1106** may include general-purpose registers or other registers.

In at least one embodiment, one or more processor(s) **1102** are coupled with one or more interface bus(es) **1110** to transmit communication signals such as address, data, or control signals between processor **1102** and other components in system **1100**. In at least one embodiment, interface bus **1110**, in one embodiment, can be a processor bus, such as a version of a Direct Media Interface (DMI) bus. In at least one embodiment, interface **1110** is not limited to a DMI bus, and may include one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express), memory busses, or other types of interface busses. In at least one embodiment processor(s) **1102** include an integrated memory controller **1116** and a platform controller hub **1130**. In at least one embodiment, memory controller **1116** facilitates communication between a memory device and other components of system **1100**, while platform controller hub (PCH) **1130** provides connections to I/O devices via a local I/O bus.

In at least one embodiment, memory device **1120** can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In at least one embodiment memory device **1120** can operate as system memory for system **1100**, to store data **1122** and instructions **1121** for use when one or more processors **1102** executes an application or process. In at least one embodiment, memory controller **1116** also couples with an optional external graphics processor **1112**, which may communicate with one or more graphics processors **1108** in processors **1102** to perform graphics and media operations. In at least one embodiment, a display device **1111** can connect to processor(s) **1102**. In at least one embodiment display device **1111** can include one or more of an internal display device, as in a mobile electronic device or a laptop device or an external display device attached via a display interface (e.g., DisplayPort, etc.). In at least one embodiment, display device **1111** can include a head mounted display (HMD) such as a stereoscopic display device for use in virtual reality (VR) applications or augmented reality (AR) applications.

In at least one embodiment, platform controller hub **1130** enables peripherals to connect to memory device **1120** and processor **1102** via a high-speed I/O bus. In at least one embodiment, I/O peripherals include, but are not limited to, an audio controller **1146**, a network controller **1134**, a firmware interface **1128**, a wireless transceiver **1126**, touch sensors **1125**, a data storage device **1124** (e.g., hard disk drive, flash memory, etc.). In at least one embodiment, data storage device **1124** can connect via a storage interface (e.g., SATA) or via a peripheral bus, such as a Peripheral Component Interconnect bus (e.g., PCI, PCI Express). In at least one embodiment, touch sensors **1125** can include touch screen sensors, pressure sensors, or fingerprint sensors. In at least one embodiment, wireless transceiver **1126** can be a Wi-Fi transceiver, a Bluetooth transceiver, or a mobile network transceiver such as a 3G, 4G, or Long Term Evolution (LTE) transceiver. In at least one embodiment, firmware interface **1128** enables communication with system firmware, and can be, for example, a unified extensible firmware interface (UEFI). In at least one embodiment, network controller **1134** can enable a network connection to a wired network. In at least one embodiment, a high-performance network controller (not shown) couples with

interface bus **1110**. In at least one embodiment, audio controller **1146** is a multi-channel high definition audio controller. In at least one embodiment, system **1100** includes an optional legacy I/O controller **1140** for coupling legacy (e.g., Personal System 2 (PS/2)) devices to system. In at least one embodiment, platform controller hub **1130** can also connect to one or more Universal Serial Bus (USB) controllers **1142** connect input devices, such as keyboard and mouse **1143** combinations, a camera **1144**, or other USB input devices.

In at least one embodiment, an instance of memory controller **1116** and platform controller hub **1130** may be integrated into a discreet external graphics processor, such as external graphics processor **1112**. In at least one embodiment, platform controller hub **1130** and/or memory controller **1116** may be external to one or more processor(s) **1102**. For example, in at least one embodiment, system **1100** can include an external memory controller **1116** and platform controller hub **1130**, which may be configured as a memory controller hub and peripheral controller hub within a system chipset that is in communication with processor(s) **1102**.

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B. In at least one embodiment portions or all of inference and/or training logic **715** may be incorporated into graphics processor **1500**. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in a graphics processor. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. **7**A or **7**B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of a graphics processor to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

FIG. **12** is a block diagram of a processor **1200** having one or more processor cores **1202**A-**1202**N, an integrated memory controller **1214**, and an integrated graphics processor **1208**, according to at least one embodiment. In at least one embodiment, processor **1200** can include additional cores up to and including additional core **1202**N represented by dashed lined boxes. In at least one embodiment, each of processor cores **1202**A-**1202**N includes one or more internal cache units **1204**A-**1204**N. In at least one embodiment, each processor core also has access to one or more shared cached units **1206**.

In at least one embodiment, internal cache units **1204**A-**1204**N and shared cache units **1206** represent a cache memory hierarchy within processor **1200**. In at least one embodiment, cache memory units **1204**A-**1204**N may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as a Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, where a highest level of cache before external memory is classified as an LLC. In at least one embodiment, cache coherency logic maintains coherency between various cache units **1206** and **1204**A-**1204**N.

In at least one embodiment, processor **1200** may also include a set of one or more bus controller units **1216** and a system agent core **1210**. In at least one embodiment, one or more bus controller units **1216** manage a set of peripheral buses, such as one or more PCI or PCI express busses. In at least one embodiment, system agent core **1210** provides management functionality for various processor components. In at least one embodiment, system agent core **1210** includes one or more integrated memory controllers **1214** to manage access to various external memory devices (not shown).

In at least one embodiment, one or more of processor cores **1202**A-**1202**N include support for simultaneous multi-threading. In at least one embodiment, system agent core **1210** includes components for coordinating and operating cores **1202**A-**1202**N during multi-threaded processing. In at least one embodiment, system agent core **1210** may additionally include a power control unit (PCU), which includes logic and components to regulate one or more power states of processor cores **1202**A-**1202**N and graphics processor **1208**.

In at least one embodiment, processor **1200** additionally includes graphics processor **1208** to execute graphics processing operations. In at least one embodiment, graphics processor **1208** couples with shared cache units **1206**, and system agent core **1210**, including one or more integrated memory controllers **1214**. In at least one embodiment, system agent core **1210** also includes a display controller **1211** to drive graphics processor output to one or more coupled displays. In at least one embodiment, display controller **1211** may also be a separate module coupled with graphics processor **1208** via at least one interconnect, or may be integrated within graphics processor **1208**.

In at least one embodiment, a ring based interconnect unit **1212** is used to couple internal components of processor **1200**. In at least one embodiment, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques. In at least one embodiment, graphics processor **1208** couples with ring interconnect **1212** via an I/O link **1213**.

In at least one embodiment, I/O link **1213** represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module **1218**, such as an eDRAM module. In at least one embodiment, each of processor cores **1202**A-**1202**N and graphics processor **1208** use embedded memory modules **1218** as a shared Last Level Cache.

In at least one embodiment, processor cores **1202**A-**1202**N are homogenous cores executing a common instruction set architecture. In at least one embodiment, processor cores **1202**A-**1202**N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores **1202**A-**1202**N execute a common instruction set, while one or more other cores of processor cores **1202**A-**1202**N executes a subset of a common instruction set or a different instruction set. In at least one embodiment, processor cores **1202**A-**1202**N are heterogeneous in terms of microarchitecture, where one or more cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. In at least one embodiment, processor **1200** can be implemented on one or more chips or as an SoC integrated circuit.

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7***a* and/or **7***b*. In at least one embodiment portions or all of

inference and/or training logic **715** may be incorporated into processor **1200**. For example, in at least one embodiment, training and/or inferencing techniques described herein may use one or more of ALUs embodied in graphics processor **1512**, graphics core(s) **1202**A-**1202**N, or other components in FIG. **12**. Moreover, in at least one embodiment, inferencing and/or training operations described herein may be done using logic other than logic illustrated in FIG. **7**A or **7**B. In at least one embodiment, weight parameters may be stored in on-chip or off-chip memory and/or registers (shown or not shown) that configure ALUs of graphics processor **1200** to perform one or more machine learning algorithms, neural network architectures, use cases, or training techniques described herein.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

Virtualized Computing Platform

FIG. **13** is an example data flow diagram for a process **1300** of generating and deploying an image processing and inferencing pipeline, in accordance with at least one embodiment. In at least one embodiment, process **1300** may be deployed for use with imaging devices, processing devices, and/or other device types at one or more facilities **1302**. Process **1300** may be executed within a training system **1304** and/or a deployment system **1306**. In at least one embodiment, training system **1304** may be used to perform training, deployment, and implementation of machine learning models (e.g., neural networks, object detection algorithms, computer vision algorithms, etc.) for use in deployment system **1306**. In at least one embodiment, deployment system **1306** may be configured to offload processing and compute resources among a distributed computing environment to reduce infrastructure requirements at facility **1302**. In at least one embodiment, one or more applications in a pipeline may use or call upon services (e.g., inference, visualization, compute, AI, etc.) of deployment system **1306** during execution of applications.

In at least one embodiment, some of applications used in advanced processing and inferencing pipelines may use machine learning models or other AI to perform one or more processing steps. In at least one embodiment, machine learning models may be trained at facility **1302** using data **1308** (such as imaging data) generated at facility **1302** (and stored on one or more picture archiving and communication system (PACS) servers at facility **1302**), may be trained using imaging or sequencing data **1308** from another facility (ies), or a combination thereof. In at least one embodiment, training system **1304** may be used to provide applications, services, and/or other resources for generating working, deployable machine learning models for deployment system **1306**.

In at least one embodiment, model registry **1324** may be backed by object storage that may support versioning and object metadata. In at least one embodiment, object storage may be accessible through, for example, a cloud storage (e.g., cloud **1426** of FIG. **14**) compatible application programming interface (API) from within a cloud platform. In at least one embodiment, machine learning models within model registry **1324** may uploaded, listed, modified, or deleted by developers or partners of a system interacting with an API. In at least one embodiment, an API may provide access to methods that allow users with appropriate credentials to associate models with applications, such that models may be executed as part of execution of containerized instantiations of applications.

In at least one embodiment, training pipeline **1404** (FIG. **14**) may include a scenario where facility **1302** is training their own machine learning model, or has an existing machine learning model that needs to be optimized or updated. In at least one embodiment, imaging data **1308** generated by imaging device(s), sequencing devices, and/or other device types may be received. In at least one embodiment, once imaging data **1308** is received, AI-assisted annotation **1310** may be used to aid in generating annotations corresponding to imaging data **1308** to be used as ground truth data for a machine learning model. In at least one embodiment, AI-assisted annotation **1310** may include one or more machine learning models (e.g., convolutional neural networks (CNNs)) that may be trained to generate annotations corresponding to certain types of imaging data **1308** (e.g., from certain devices). In at least one embodiment, AI-assisted annotations **1310** may then be used directly, or may be adjusted or fine-tuned using an annotation tool to generate ground truth data. In at least one embodiment, AI-assisted annotations **1310**, labeled clinic data **1312**, or a combination thereof may be used as ground truth data for training a machine learning model. In at least one embodiment, a trained machine learning model may be referred to as output model **1316**, and may be used by deployment system **1306**, as described herein.

In at least one embodiment, training pipeline **1404** (FIG. **14**) may include a scenario where facility **1302** needs a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system **1306**, but facility **1302** may not currently have such a machine learning model (or may not have a model that is optimized, efficient, or effective for such purposes). In at least one embodiment, an existing machine learning model may be selected from a model registry **1324**. In at least one embodiment, model registry **1324** may include machine learning models trained to perform a variety of different inference tasks on imaging data. In at least one embodiment, machine learning models in model registry **1324** may have been trained on imaging data from different facilities than facility **1302** (e.g., facilities remotely located). In at least one embodiment, machine learning models may have been trained on imaging data from one location, two locations, or any number of locations. In at least one embodiment, when being trained on imaging data from a specific location, training may take place at that location, or at least in a manner that protects confidentiality of imaging data or restricts imaging data from being transferred off-premises. In at least one embodiment, once a model is trained—or partially trained—at one location, a machine learning model may be added to model registry **1324**. In at least one embodiment, a machine learning model may then be retrained, or updated, at any number of other facilities, and a retrained or updated model may be made available in model registry **1324**. In at least one embodiment, a machine learning model may then be selected from model registry **1324**—and referred to as output model **1316**—and may be used in deployment system **1306** to perform one or more processing tasks for one or more applications of a deployment system.

In at least one embodiment, training pipeline **1404** (FIG. **14**), a scenario may include facility **1302** requiring a machine learning model for use in performing one or more processing tasks for one or more applications in deployment system **1306**, but facility **1302** may not currently have such a machine learning model (or may not have a model that is

optimized, efficient, or effective for such purposes). In at least one embodiment, a machine learning model selected from model registry **1324** may not be fine-tuned or optimized for imaging data **1308** generated at facility **1302** because of differences in populations, robustness of training data used to train a machine learning model, diversity in anomalies of training data, and/or other issues with training data. In at least one embodiment, AI-assisted annotation **1310** may be used to aid in generating annotations corresponding to imaging data **1308** to be used as ground truth data for retraining or updating a machine learning model. In at least one embodiment, labeled data **1312** may be used as ground truth data for training a machine learning model. In at least one embodiment, retraining or updating a machine learning model may be referred to as model training **1314**. In at least one embodiment, model training **1314**—e.g., AI-assisted annotations **1310**, labeled clinic data **1312**, or a combination thereof—may be used as ground truth data for retraining or updating a machine learning model. In at least one embodiment, a trained machine learning model may be referred to as output model **1316**, and may be used by deployment system **1306**, as described herein.

In at least one embodiment, deployment system **1306** may include software **1318**, services **1320**, hardware **1322**, and/or other components, features, and functionality. In at least one embodiment, deployment system **1306** may include a software "stack," such that software **1318** may be built on top of services **1320** and may use services **1320** to perform some or all of processing tasks, and services **1320** and software **1318** may be built on top of hardware **1322** and use hardware **1322** to execute processing, storage, and/or other compute tasks of deployment system **1306**. In at least one embodiment, software **1318** may include any number of different containers, where each container may execute an instantiation of an application. In at least one embodiment, each application may perform one or more processing tasks in an advanced processing and inferencing pipeline (e.g., inferencing, object detection, feature detection, segmentation, image enhancement, calibration, etc.). In at least one embodiment, an advanced processing and inferencing pipeline may be defined based on selections of different containers that are desired or required for processing imaging data **1308**, in addition to containers that receive and configure imaging data for use by each container and/or for use by facility **1302** after processing through a pipeline (e.g., to convert outputs back to a usable data type). In at least one embodiment, a combination of containers within software **1318** (e.g., that make up a pipeline) may be referred to as a virtual instrument (as described in more detail herein), and a virtual instrument may leverage services **1320** and hardware **1322** to execute some or all processing tasks of applications instantiated in containers.

In at least one embodiment, a data processing pipeline may receive input data (e.g., imaging data **1308**) in a specific format in response to an inference request (e.g., a request from a user of deployment system **1306**). In at least one embodiment, input data may be representative of one or more images, video, and/or other data representations generated by one or more imaging devices. In at least one embodiment, data may undergo pre-processing as part of data processing pipeline to prepare data for processing by one or more applications. In at least one embodiment, post-processing may be performed on an output of one or more inferencing tasks or other processing tasks of a pipeline to prepare an output data for a next application and/or to prepare output data for transmission and/or use by a user (e.g., as a response to an inference request). In at least one embodiment, inferencing tasks may be performed by one or more machine learning models, such as trained or deployed neural networks, which may include output models **1316** of training system **1304**.

In at least one embodiment, tasks of data processing pipeline may be encapsulated in a container(s) that each represents a discrete, fully functional instantiation of an application and virtualized computing environment that is able to reference machine learning models. In at least one embodiment, containers or applications may be published into a private (e.g., limited access) area of a container registry (described in more detail herein), and trained or deployed models may be stored in model registry **1324** and associated with one or more applications. In at least one embodiment, images of applications (e.g., container images) may be available in a container registry, and once selected by a user from a container registry for deployment in a pipeline, an image may be used to generate a container for an instantiation of an application for use by a user's system.

In at least one embodiment, developers (e.g., software developers, clinicians, doctors, etc.) may develop, publish, and store applications (e.g., as containers) for performing image processing and/or inferencing on supplied data. In at least one embodiment, development, publishing, and/or storing may be performed using a software development kit (SDK) associated with a system (e.g., to ensure that an application and/or container developed is compliant with or compatible with a system). In at least one embodiment, an application that is developed may be tested locally (e.g., at a first facility, on data from a first facility) with an SDK which may support at least some of services **1320** as a system (e.g., system **1400** of FIG. **14**). In at least one embodiment, because DICOM objects may contain anywhere from one to hundreds of images or other data types, and due to a variation in data, a developer may be responsible for managing (e.g., setting constructs for, building pre-processing into an application, etc.) extraction and preparation of incoming data. In at least one embodiment, once validated by system **1400** (e.g., for accuracy), an application may be available in a container registry for selection and/or implementation by a user to perform one or more processing tasks with respect to data at a facility (e.g., a second facility) of a user.

In at least one embodiment, developers may then share applications or containers through a network for access and use by users of a system (e.g., system **1400** of FIG. **14**). In at least one embodiment, completed and validated applications or containers may be stored in a container registry and associated machine learning models may be stored in model registry **1324**. In at least one embodiment, a requesting entity—who provides an inference or image processing request—may browse a container registry and/or model registry **1324** for an application, container, dataset, machine learning model, etc., select a desired combination of elements for inclusion in data processing pipeline, and submit an imaging processing request. In at least one embodiment, a request may include input data (and associated patient data, in some examples) that is necessary to perform a request, and/or may include a selection of application(s) and/or machine learning models to be executed in processing a request. In at least one embodiment, a request may then be passed to one or more components of deployment system **1306** (e.g., a cloud) to perform processing of data processing pipeline. In at least one embodiment, processing by deployment system **1306** may include referencing selected elements (e.g., applications, containers, models, etc.) from a container registry and/or model registry **1324**. In at least one

embodiment, once results are generated by a pipeline, results may be returned to a user for reference (e.g., for viewing in a viewing application suite executing on a local, on-premises workstation or terminal).

In at least one embodiment, to aid in processing or execution of applications or containers in pipelines, services **1320** may be leveraged. In at least one embodiment, services **1320** may include compute services, artificial intelligence (AI) services, visualization services, and/or other service types. In at least one embodiment, services **1320** may provide functionality that is common to one or more applications in software **1318**, so functionality may be abstracted to a service that may be called upon or leveraged by applications. In at least one embodiment, functionality provided by services **1320** may run dynamically and more efficiently, while also scaling well by allowing applications to process data in parallel (e.g., using a parallel computing platform **1430** (FIG. **14**)). In at least one embodiment, rather than each application that shares a same functionality offered by a service **1320** being required to have a respective instance of service **1320**, service **1320** may be shared between and among various applications. In at least one embodiment, services may include an inference server or engine that may be used for executing detection or segmentation tasks, as non-limiting examples. In at least one embodiment, a model training service may be included that may provide machine learning model training and/or retraining capabilities. In at least one embodiment, a data augmentation service may further be included that may provide GPU accelerated data (e.g., DICOM, RIS, CIS, REST compliant, RPC, raw, etc.) extraction, resizing, scaling, and/or other augmentation. In at least one embodiment, a visualization service may be used that may add image rendering effects—such as ray-tracing, rasterization, denoising, sharpening, etc.—to add realism to two-dimensional (2D) and/or three-dimensional (3D) models. In at least one embodiment, virtual instrument services may be included that provide for beam-forming, segmentation, inferencing, imaging, and/or support for other applications within pipelines of virtual instruments.

In at least one embodiment, where a service **1320** includes an AI service (e.g., an inference service), one or more machine learning models may be executed by calling upon (e.g., as an API call) an inference service (e.g., an inference server) to execute machine learning model(s), or processing thereof, as part of application execution. In at least one embodiment, where another application includes one or more machine learning models for segmentation tasks, an application may call upon an inference service to execute machine learning models for performing one or more of processing operations associated with segmentation tasks. In at least one embodiment, software **1318** implementing advanced processing and inferencing pipeline that includes segmentation application and anomaly detection application may be streamlined because each application may call upon a same inference service to perform one or more inferencing tasks.

In at least one embodiment, hardware **1322** may include GPUs, CPUs, graphics cards, an AI/deep learning system (e.g., an AI supercomputer, such as NVIDIA's DGX), a cloud platform, or a combination thereof. In at least one embodiment, different types of hardware **1322** may be used to provide efficient, purpose-built support for software **1318** and services **1320** in deployment system **1306**. In at least one embodiment, use of GPU processing may be implemented for processing locally (e.g., at facility **1302**), within an AI/deep learning system, in a cloud system, and/or in other processing components of deployment system **1306** to improve efficiency, accuracy, and efficacy of image processing and generation. In at least one embodiment, software **1318** and/or services **1320** may be optimized for GPU processing with respect to deep learning, machine learning, and/or high-performance computing, as non-limiting examples. In at least one embodiment, at least some of computing environment of deployment system **1306** and/or training system **1304** may be executed in a datacenter one or more supercomputers or high performance computing systems, with GPU optimized software (e.g., hardware and software combination of NVIDIA's DGX System). In at least one embodiment, hardware **1322** may include any number of GPUs that may be called upon to perform processing of data in parallel, as described herein. In at least one embodiment, cloud platform may further include GPU processing for GPU-optimized execution of deep learning tasks, machine learning tasks, or other computing tasks. In at least one embodiment, cloud platform (e.g., NVIDIA's NGC) may be executed using an AI/deep learning supercomputer(s) and/or GPU-optimized software (e.g., as provided on NVIDIA's DGX Systems) as a hardware abstraction and scaling platform. In at least one embodiment, cloud platform may integrate an application container clustering system or orchestration system (e.g., KUBERNETES) on multiple GPUs to enable seamless scaling and load balancing.

FIG. **14** is a system diagram for an example system **1400** for generating and deploying an imaging deployment pipeline, in accordance with at least one embodiment. In at least one embodiment, system **1400** may be used to implement process **1300** of FIG. **13** and/or other processes including advanced processing and inferencing pipelines. In at least one embodiment, system **1400** may include training system **1304** and deployment system **1306**. In at least one embodiment, training system **1304** and deployment system **1306** may be implemented using software **1318**, services **1320**, and/or hardware **1322**, as described herein.

In at least one embodiment, system **1400** (e.g., training system **1304** and/or deployment system **1306**) may implemented in a cloud computing environment (e.g., using cloud **1426**). In at least one embodiment, system **1400** may be implemented locally with respect to a healthcare services facility, or as a combination of both cloud and local computing resources. In at least one embodiment, access to APIs in cloud **1426** may be restricted to authorized users through enacted security measures or protocols. In at least one embodiment, a security protocol may include web tokens that may be signed by an authentication (e.g., AuthN, AuthZ, Gluecon, etc.) service and may carry appropriate authorization. In at least one embodiment, APIs of virtual instruments (described herein), or other instantiations of system **1400**, may be restricted to a set of public IPs that have been vetted or authorized for interaction.

In at least one embodiment, various components of system **1400** may communicate between and among one another using any of a variety of different network types, including but not limited to local area networks (LANs) and/or wide area networks (WANs) via wired and/or wireless communication protocols. In at least one embodiment, communication between facilities and components of system **1400** (e.g., for transmitting inference requests, for receiving results of inference requests, etc.) may be communicated over data bus(ses), wireless data protocols (Wi-Fi), wired data protocols (e.g., Ethernet), etc.

In at least one embodiment, training system **1304** may execute training pipelines **1404**, similar to those described

herein with respect to FIG. **13**. In at least one embodiment, where one or more machine learning models are to be used in deployment pipelines **1410** by deployment system **1306**, training pipelines **1404** may be used to train or retrain one or more (e.g. pre-trained) models, and/or implement one or more of pre-trained models **1406** (e.g., without a need for retraining or updating). In at least one embodiment, as a result of training pipelines **1404**, output model(s) **1316** may be generated. In at least one embodiment, training pipelines **1404** may include any number of processing steps, such as but not limited to imaging data (or other input data) conversion or adaption In at least one embodiment, for different machine learning models used by deployment system **1306**, different training pipelines **1404** may be used. In at least one embodiment, training pipeline **1404** similar to a first example described with respect to FIG. **13** may be used for a first machine learning model, training pipeline **1404** similar to a second example described with respect to FIG. **13** may be used for a second machine learning model, and training pipeline **1404** similar to a third example described with respect to FIG. **13** may be used for a third machine learning model. In at least one embodiment, any combination of tasks within training system **1304** may be used depending on what is required for each respective machine learning model. In at least one embodiment, one or more of machine learning models may already be trained and ready for deployment so machine learning models may not undergo any processing by training system **1304**, and may be implemented by deployment system **1306**.

In at least one embodiment, output model(s) **1316** and/or pre-trained model(s) **1406** may include any types of machine learning models depending on implementation or embodiment. In at least one embodiment, and without limitation, machine learning models used by system **1400** may include machine learning model(s) using linear regression, logistic regression, decision trees, support vector machines (SVM), Naïve Bayes, k-nearest neighbor (Knn), K means clustering, random forest, dimensionality reduction algorithms, gradient boosting algorithms, neural networks (e.g., auto-encoders, convolutional, recurrent, perceptrons, Long/Short Term Memory (LSTM), Hopfield, Boltzmann, deep belief, deconvolutional, generative adversarial, liquid state machine, etc.), and/or other types of machine learning models.

In at least one embodiment, training pipelines **1404** may include AI-assisted annotation, as described in more detail herein with respect to at least FIG. **15**B. In at least one embodiment, labeled data **1312** (e.g., traditional annotation) may be generated by any number of techniques. In at least one embodiment, labels or other annotations may be generated within a drawing program (e.g., an annotation program), a computer aided design (CAD) program, a labeling program, another type of program suitable for generating annotations or labels for ground truth, and/or may be hand drawn, in some examples. In at least one embodiment, ground truth data may be synthetically produced (e.g., generated from computer models or renderings), real produced (e.g., designed and produced from real-world data), machine-automated (e.g., using feature analysis and learning to extract features from data and then generate labels), human annotated (e.g., labeler, or annotation expert, defines location of labels), and/or a combination thereof. In at least one embodiment, for each instance of imaging data **1308** (or other data type used by machine learning models), there may be corresponding ground truth data generated by training system **1304**. In at least one embodiment, AI-assisted annotation may be performed as part of deployment pipelines **1410**; either in addition to, or in lieu of AI-assisted annotation included in training pipelines **1404**. In at least one embodiment, system **1400** may include a multi-layer platform that may include a software layer (e.g., software **1318**) of diagnostic applications (or other application types) that may perform one or more medical imaging and diagnostic functions. In at least one embodiment, system **1400** may be communicatively coupled to (e.g., via encrypted links) PACS server networks of one or more facilities. In at least one embodiment, system **1400** may be configured to access and referenced data from PACS servers to perform operations, such as training machine learning models, deploying machine learning models, image processing, inferencing, and/or other operations.

In at least one embodiment, a software layer may be implemented as a secure, encrypted, and/or authenticated API through which applications or containers may be invoked (e.g., called) from an external environment(s) (e.g., facility **1302**). In at least one embodiment, applications may then call or execute one or more services **1320** for performing compute, AI, or visualization tasks associated with respective applications, and software **1318** and/or services **1320** may leverage hardware **1322** to perform processing tasks in an effective and efficient manner.

In at least one embodiment, deployment system **1306** may execute deployment pipelines **1410**. In at least one embodiment, deployment pipelines **1410** may include any number of applications that may be sequentially, non-sequentially, or otherwise applied to imaging data (and/or other data types) generated by imaging devices, sequencing devices, genomics devices, etc.—including AI-assisted annotation, as described above. In at least one embodiment, as described herein, a deployment pipeline **1410** for an individual device may be referred to as a virtual instrument for a device (e.g., a virtual ultrasound instrument, a virtual CT scan instrument, a virtual sequencing instrument, etc.). In at least one embodiment, for a single device, there may be more than one deployment pipeline **1410** depending on information desired from data generated by a device. In at least one embodiment, where detections of anomalies are desired from an Mill machine, there may be a first deployment pipeline **1410**, and where image enhancement is desired from output of an Mill machine, there may be a second deployment pipeline **1410**.

In at least one embodiment, an image generation application may include a processing task that includes use of a machine learning model. In at least one embodiment, a user may desire to use their own machine learning model, or to select a machine learning model from model registry **1324**. In at least one embodiment, a user may implement their own machine learning model or select a machine learning model for inclusion in an application for performing a processing task. In at least one embodiment, applications may be selectable and customizable, and by defining constructs of applications, deployment and implementation of applications for a particular user are presented as a more seamless user experience. In at least one embodiment, by leveraging other features of system **1400**—such as services **1320** and hardware **1322**—deployment pipelines **1410** may be even more user friendly, provide for easier integration, and produce more accurate, efficient, and timely results.

In at least one embodiment, deployment system **1306** may include a user interface **1414** (e.g., a graphical user interface, a web interface, etc.) that may be used to select applications for inclusion in deployment pipeline(s) **1410**, arrange applications, modify or change applications or parameters or constructs thereof, use and interact with deployment pipeline(s) **1410** during set-up and/or deployment, and/or to otherwise interact with deployment system **1306**. In at least

one embodiment, although not illustrated with respect to training system **1304**, user interface **1414** (or a different user interface) may be used for selecting models for use in deployment system **1306**, for selecting models for training, or retraining, in training system **1304**, and/or for otherwise interacting with training system **1304**.

In at least one embodiment, pipeline manager **1412** may be used, in addition to an application orchestration system **1428**, to manage interaction between applications or containers of deployment pipeline(s) **1410** and services **1320** and/or hardware **1322**. In at least one embodiment, pipeline manager **1412** may be configured to facilitate interactions from application to application, from application to service **1320**, and/or from application or service to hardware **1322**. In at least one embodiment, although illustrated as included in software **1318**, this is not intended to be limiting, and in some examples (e.g., as illustrated in FIG. **12***cc*) pipeline manager **1412** may be included in services **1320**. In at least one embodiment, application orchestration system **1428** (e.g., Kubernetes, DOCKER, etc.) may include a container orchestration system that may group applications into containers as logical units for coordination, management, scaling, and deployment. In at least one embodiment, by associating applications from deployment pipeline(s) **1410** (e.g., a reconstruction application, a segmentation application, etc.) with individual containers, each application may execute in a self-contained environment (e.g., at a kernel level) to increase speed and efficiency.

In at least one embodiment, each application and/or container (or image thereof) may be individually developed, modified, and deployed (e.g., a first user or developer may develop, modify, and deploy a first application and a second user or developer may develop, modify, and deploy a second application separate from a first user or developer), which may allow for focus on, and attention to, a task of a single application and/or container(s) without being hindered by tasks of another application(s) or container(s). In at least one embodiment, communication, and cooperation between different containers or applications may be aided by pipeline manager **1412** and application orchestration system **1428**. In at least one embodiment, so long as an expected input and/or output of each container or application is known by a system (e.g., based on constructs of applications or containers), application orchestration system **1428** and/or pipeline manager **1412** may facilitate communication among and between, and sharing of resources among and between, each of applications or containers. In at least one embodiment, because one or more of applications or containers in deployment pipeline(s) **1410** may share same services and resources, application orchestration system **1428** may orchestrate, load balance, and determine sharing of services or resources between and among various applications or containers. In at least one embodiment, a scheduler may be used to track resource requirements of applications or containers, current usage or planned usage of these resources, and resource availability. In at least one embodiment, a scheduler may thus allocate resources to different applications and distribute resources between and among applications in view of requirements and availability of a system. In some examples, a scheduler (and/or other component of application orchestration system **1428**) may determine resource availability and distribution based on constraints imposed on a system (e.g., user constraints), such as quality of service (QoS), urgency of need for data outputs (e.g., to determine whether to execute real-time processing or delayed processing), etc.

In at least one embodiment, services **1320** leveraged by and shared by applications or containers in deployment system **1306** may include compute services **1416**, AI services **1418**, visualization services **1420**, and/or other service types. In at least one embodiment, applications may call (e.g., execute) one or more of services **1320** to perform processing operations for an application. In at least one embodiment, compute services **1416** may be leveraged by applications to perform super-computing or other high-performance computing (HPC) tasks. In at least one embodiment, compute service(s) **1416** may be leveraged to perform parallel processing (e.g., using a parallel computing platform **1430**) for processing data through one or more of applications and/or one or more tasks of a single application, substantially simultaneously. In at least one embodiment, parallel computing platform **1430** (e.g., NVIDIA's CUDA) may enable general purpose computing on GPUs (GPGPU) (e.g., GPUs **1422**). In at least one embodiment, a software layer of parallel computing platform **1430** may provide access to virtual instruction sets and parallel computational elements of GPUs, for execution of compute kernels. In at least one embodiment, parallel computing platform **1430** may include memory and, in some embodiments, a memory may be shared between and among multiple containers, and/or between and among different processing tasks within a single container. In at least one embodiment, inter-process communication (IPC) calls may be generated for multiple containers and/or for multiple processes within a container to use same data from a shared segment of memory of parallel computing platform **1430** (e.g., where multiple different stages of an application or multiple applications are processing same information). In at least one embodiment, rather than making a copy of data and moving data to different locations in memory (e.g., a read/write operation), same data in same location of a memory may be used for any number of processing tasks (e.g., at a same time, at different times, etc.). In at least one embodiment, as data is used to generate new data as a result of processing, this information of a new location of data may be stored and shared between various applications. In at least one embodiment, location of data and a location of updated or modified data may be part of a definition of how a payload is understood within containers.

In at least one embodiment, AI services **1418** may be leveraged to perform inferencing services for executing machine learning model(s) associated with applications (e.g., tasked with performing one or more processing tasks of an application). In at least one embodiment, AI services **1418** may leverage AI system **1424** to execute machine learning model(s) (e.g., neural networks, such as CNNs) for segmentation, reconstruction, object detection, feature detection, classification, and/or other inferencing tasks. In at least one embodiment, applications of deployment pipeline(s) **1410** may use one or more of output models **1316** from training system **1304** and/or other models of applications to perform inference on imaging data. In at least one embodiment, two or more examples of inferencing using application orchestration system **1428** (e.g., a scheduler) may be available. In at least one embodiment, a first category may include a high priority/low latency path that may achieve higher service level agreements, such as for performing inference on urgent requests during an emergency, or for a radiologist during diagnosis. In at least one embodiment, a second category may include a standard priority path that may be used for requests that may be non-urgent or where analysis may be performed at a later time. In at least one embodiment, application orchestration system **1428** may

distribute resources (e.g., services **1320** and/or hardware **1322**) based on priority paths for different inferencing tasks of AI services **1418**.

In at least one embodiment, shared storage may be mounted to AI services **1418** within system **1400**. In at least one embodiment, shared storage may operate as a cache (or other storage device type) and may be used to process inference requests from applications. In at least one embodiment, when an inference request is submitted, a request may be received by a set of API instances of deployment system **1306**, and one or more instances may be selected (e.g., for best fit, for load balancing, etc.) to process a request. In at least one embodiment, to process a request, a request may be entered into a database, a machine learning model may be located from model registry **1324** if not already in a cache, a validation step may ensure appropriate machine learning model is loaded into a cache (e.g., shared storage), and/or a copy of a model may be saved to a cache. In at least one embodiment, a scheduler (e.g., of pipeline manager **1412**) may be used to launch an application that is referenced in a request if an application is not already running or if there are not enough instances of an application. In at least one embodiment, if an inference server is not already launched to execute a model, an inference server may be launched. Any number of inference servers may be launched per model. In at least one embodiment, in a pull model, in which inference servers are clustered, models may be cached whenever load balancing is advantageous. In at least one embodiment, inference servers may be statically loaded in corresponding, distributed servers.

In at least one embodiment, inferencing may be performed using an inference server that runs in a container. In at least one embodiment, an instance of an inference server may be associated with a model (and optionally a plurality of versions of a model). In at least one embodiment, if an instance of an inference server does not exist when a request to perform inference on a model is received, a new instance may be loaded. In at least one embodiment, when starting an inference server, a model may be passed to an inference server such that a same container may be used to serve different models so long as inference server is running as a different instance.

In at least one embodiment, during application execution, an inference request for a given application may be received, and a container (e.g., hosting an instance of an inference server) may be loaded (if not already), and a start procedure may be called. In at least one embodiment, pre-processing logic in a container may load, decode, and/or perform any additional pre-processing on incoming data (e.g., using a CPU(s) and/or GPU(s)). In at least one embodiment, once data is prepared for inference, a container may perform inference as necessary on data. In at least one embodiment, this may include a single inference call on one image (e.g., a hand X-ray), or may require inference on hundreds of images (e.g., a chest CT). In at least one embodiment, an application may summarize results before completing, which may include, without limitation, a single confidence score, pixel level-segmentation, voxel-level segmentation, generating a visualization, or generating text to summarize findings. In at least one embodiment, different models or applications may be assigned different priorities. For example, some models may have a real-time (TAT<1 min) priority while others may have lower priority (e.g., TAT<10 min). In at least one embodiment, model execution times may be measured from requesting institution or entity and may include partner network traversal time, as well as execution on an inference service.

In at least one embodiment, transfer of requests between services **1320** and inference applications may be hidden behind a software development kit (SDK), and robust transport may be provide through a queue. In at least one embodiment, a request will be placed in a queue via an API for an individual application/tenant ID combination and an SDK will pull a request from a queue and give a request to an application. In at least one embodiment, a name of a queue may be provided in an environment from where an SDK will pick it up. In at least one embodiment, asynchronous communication through a queue may be useful as it may allow any instance of an application to pick up work as it becomes available. Results may be transferred back through a queue, to ensure no data is lost. In at least one embodiment, queues may also provide an ability to segment work, as highest priority work may go to a queue with most instances of an application connected to it, while lowest priority work may go to a queue with a single instance connected to it that processes tasks in an order received. In at least one embodiment, an application may run on a GPU-accelerated instance generated in cloud **1426**, and an inference service may perform inferencing on a GPU.

In at least one embodiment, visualization services **1420** may be leveraged to generate visualizations for viewing outputs of applications and/or deployment pipeline(s) **1410**. In at least one embodiment, GPUs **1422** may be leveraged by visualization services **1420** to generate visualizations. In at least one embodiment, rendering effects, such as ray-tracing, may be implemented by visualization services **1420** to generate higher quality visualizations. In at least one embodiment, visualizations may include, without limitation, 2D image renderings, 3D volume renderings, 3D volume reconstruction, 2D tomographic slices, virtual reality displays, augmented reality displays, etc. In at least one embodiment, virtualized environments may be used to generate a virtual interactive display or environment (e.g., a virtual environment) for interaction by users of a system (e.g., doctors, nurses, radiologists, etc.). In at least one embodiment, visualization services **1420** may include an internal visualizer, cinematics, and/or other rendering or image processing capabilities or functionality (e.g., ray tracing, rasterization, internal optics, etc.).

In at least one embodiment, hardware **1322** may include GPUs **1422**, AI system **1424**, cloud **1426**, and/or any other hardware used for executing training system **1304** and/or deployment system **1306**. In at least one embodiment, GPUs **1422** (e.g., NVIDIA's TESLA and/or QUADRO GPUs) may include any number of GPUs that may be used for executing processing tasks of compute services **1416**, AI services **1418**, visualization services **1420**, other services, and/or any of features or functionality of software **1318**. For example, with respect to AI services **1418**, GPUs **1422** may be used to perform pre-processing on imaging data (or other data types used by machine learning models), post-processing on outputs of machine learning models, and/or to perform inferencing (e.g., to execute machine learning models). In at least one embodiment, cloud **1426**, AI system **1424**, and/or other components of system **1400** may use GPUs **1422**. In at least one embodiment, cloud **1426** may include a GPU-optimized platform for deep learning tasks. In at least one embodiment, AI system **1424** may use GPUs, and cloud **1426**—or at least a portion tasked with deep learning or inferencing—may be executed using one or more AI systems **1424**. As such, although hardware **1322** is illustrated as discrete components, this is not intended to be limiting, and any components of hardware **1322** may be combined with, or leveraged by, any other components of hardware **1322**.

In at least one embodiment, AI system **1424** may include a purpose-built computing system (e.g., a super-computer or an HPC) configured for inferencing, deep learning, machine learning, and/or other artificial intelligence tasks. In at least one embodiment, AI system **1424** (e.g., NVIDIA's DGX) may include GPU-optimized software (e.g., a software stack) that may be executed using a plurality of GPUs **1422**, in addition to CPUs, RAM, storage, and/or other components, features, or functionality. In at least one embodiment, one or more AI systems **1424** may be implemented in cloud **1426** (e.g., in a data center) for performing some or all of AI-based processing tasks of system **1400**.

In at least one embodiment, cloud **1426** may include a GPU-accelerated infrastructure (e.g., NVIDIA's NGC) that may provide a GPU-optimized platform for executing processing tasks of system **1400**. In at least one embodiment, cloud **1426** may include an AI system(s) **1424** for performing one or more of AI-based tasks of system **1400** (e.g., as a hardware abstraction and scaling platform). In at least one embodiment, cloud **1426** may integrate with application orchestration system **1428** leveraging multiple GPUs to enable seamless scaling and load balancing between and among applications and services **1320**. In at least one embodiment, cloud **1426** may tasked with executing at least some of services **1320** of system **1400**, including compute services **1416**, AI services **1418**, and/or visualization services **1420**, as described herein. In at least one embodiment, cloud **1426** may perform small and large batch inference (e.g., executing NVIDIA's TENSOR RT), provide an accelerated parallel computing API and platform **1430** (e.g., NVIDIA's CUDA), execute application orchestration system **1428** (e.g., KUBERNETES), provide a graphics rendering API and platform (e.g., for ray-tracing, 2D graphics, 3D graphics, and/or other rendering techniques to produce higher quality cinematics), and/or may provide other functionality for system **1400**.

FIG. **15**A illustrates a data flow diagram for a process **1500** to train, retrain, or update a machine learning model, in accordance with at least one embodiment. In at least one embodiment, process **1500** may be executed using, as a non-limiting example, system **1400** of FIG. **14**. In at least one embodiment, process **1500** may leverage services **1320** and/or hardware **1322** of system **1400**, as described herein. In at least one embodiment, refined models **1512** generated by process **1500** may be executed by deployment system **1306** for one or more containerized applications in deployment pipelines **1410**.

In at least one embodiment, model training **1314** may include retraining or updating an initial model **1504** (e.g., a pre-trained model) using new training data (e.g., new input data, such as customer dataset **1506**, and/or new ground truth data associated with input data). In at least one embodiment, to retrain, or update, initial model **1504**, output or loss layer(s) of initial model **1504** may be reset, or deleted, and/or replaced with an updated or new output or loss layer(s). In at least one embodiment, initial model **1504** may have previously fine-tuned parameters (e.g., weights and/or biases) that remain from prior training, so training or retraining **1314** may not take as long or require as much processing as training a model from scratch. In at least one embodiment, during model training **1314**, by having reset or replaced output or loss layer(s) of initial model **1504**, parameters may be updated and re-tuned for a new data set based on loss calculations associated with accuracy of output or loss layer(s) at generating predictions on new, customer dataset **1506** (e.g., image data **1308** of FIG. **13**).

In at least one embodiment, pre-trained models **1406** may be stored in a data store, or registry (e.g., model registry **1324** of FIG. **13**). In at least one embodiment, pre-trained models **1406** may have been trained, at least in part, at one or more facilities other than a facility executing process **1500**. In at least one embodiment, to protect privacy and rights of patients, subjects, or clients of different facilities, pre-trained models **1406** may have been trained, on-premise, using customer or patient data generated on-premise. In at least one embodiment, pre-trained models **1406** may be trained using cloud **1426** and/or other hardware **1322**, but confidential, privacy protected patient data may not be transferred to, used by, or accessible to any components of cloud **1426** (or other off premise hardware). In at least one embodiment, where a pre-trained model **1406** is trained at using patient data from more than one facility, pre-trained model **1406** may have been individually trained for each facility prior to being trained on patient or customer data from another facility. In at least one embodiment, such as where a customer or patient data has been released of privacy concerns (e.g., by waiver, for experimental use, etc.), or where a customer or patient data is included in a public data set, a customer or patient data from any number of facilities may be used to train pre-trained model **1406** on-premise and/or off premise, such as in a datacenter or other cloud computing infrastructure.

In at least one embodiment, when selecting applications for use in deployment pipelines **1410**, a user may also select machine learning models to be used for specific applications. In at least one embodiment, a user may not have a model for use, so a user may select a pre-trained model **1406** to use with an application. In at least one embodiment, pre-trained model **1406** may not be optimized for generating accurate results on customer dataset **1506** of a facility of a user (e.g., based on patient diversity, demographics, types of medical imaging devices used, etc.). In at least one embodiment, prior to deploying pre-trained model **1406** into deployment pipeline **1410** for use with an application(s), pre-trained model **1406** may be updated, retrained, and/or fine-tuned for use at a respective facility.

In at least one embodiment, a user may select pre-trained model **1406** that is to be updated, retrained, and/or fine-tuned, and pre-trained model **1406** may be referred to as initial model **1504** for training system **1304** within process **1500**. In at least one embodiment, customer dataset **1506** (e.g., imaging data, genomics data, sequencing data, or other data types generated by devices at a facility) may be used to perform model training **1314** (which may include, without limitation, transfer learning) on initial model **1504** to generate refined model **1512**. In at least one embodiment, ground truth data corresponding to customer dataset **1506** may be generated by training system **1304**. In at least one embodiment, ground truth data may be generated, at least in part, by clinicians, scientists, doctors, practitioners, at a facility (e.g., as labeled clinic data **1312** of FIG. **13**).

In at least one embodiment, AI-assisted annotation **1310** may be used in some examples to generate ground truth data. In at least one embodiment, AI-assisted annotation **1310** (e.g., implemented using an AI-assisted annotation SDK) may leverage machine learning models (e.g., neural networks) to generate suggested or predicted ground truth data for a customer dataset. In at least one embodiment, user **1510** may use annotation tools within a user interface (a graphical user interface (GUI)) on computing device **1508**.

In at least one embodiment, user **1510** may interact with a GUI via computing device **1508** to edit or fine-tune (auto)annotations. In at least one embodiment, a polygon

editing feature may be used to move vertices of a polygon to more accurate or fine-tuned locations.

In at least one embodiment, once customer dataset **1506** has associated ground truth data, ground truth data (e.g., from AI-assisted annotation, manual labeling, etc.) may be used by during model training **1314** to generate refined model **1512**. In at least one embodiment, customer dataset **1506** may be applied to initial model **1504** any number of times, and ground truth data may be used to update parameters of initial model **1504** until an acceptable level of accuracy is attained for refined model **1512**. In at least one embodiment, once refined model **1512** is generated, refined model **1512** may be deployed within one or more deployment pipelines **1410** at a facility for performing one or more processing tasks with respect to medical imaging data.

In at least one embodiment, refined model **1512** may be uploaded to pre-trained models **1406** in model registry **1324** to be selected by another facility. In at least one embodiment, his process may be completed at any number of facilities such that refined model **1512** may be further refined on new datasets any number of times to generate a more universal model.

FIG. **15**B is an example illustration of a client-server architecture **1532** to enhance annotation tools with pre-trained annotation models, in accordance with at least one embodiment. In at least one embodiment, AI-assisted annotation tools **1536** may be instantiated based on a client-server architecture **1532**. In at least one embodiment, annotation tools **1536** in imaging applications may aid radiologists, for example, identify organs and abnormalities. In at least one embodiment, imaging applications may include software tools that help user **1510** to identify, as a non-limiting example, a few extreme points on a particular organ of interest in raw images **1534** (e.g., in a 3D MRI or CT scan) and receive auto-annotated results for all 2D slices of a particular organ. In at least one embodiment, results may be stored in a data store as training data **1538** and used as (for example and without limitation) ground truth data for training. In at least one embodiment, when computing device **1508** sends extreme points for AI-assisted annotation **1310**, a deep learning model, for example, may receive this data as input and return inference results of a segmented organ or abnormality. In at least one embodiment, pre-instantiated annotation tools, such as AI-Assisted Annotation Tool **1536**B in FIG. **15**B, may be enhanced by making API calls (e.g., API Call **1544**) to a server, such as an Annotation Assistant Server **1540** that may include a set of pre-trained models **1542** stored in an annotation model registry, for example. In at least one embodiment, an annotation model registry may store pre-trained models **1542** (e.g., machine learning models, such as deep learning models) that are pre-trained to perform AI-assisted annotation on a particular organ or abnormality. These models may be further updated by using training pipelines **1404**. In at least one embodiment, pre-installed annotation tools may be improved over time as new labeled clinic data **1312** is added.

Such components can be used to render images using ray tracing-based importance sampling, which can be accelerated through hardware.

Automated Technology

FIG. **16**A is a block diagram illustrating an example system architecture for autonomous vehicle **1600** of FIG. **16**A, according to at least one embodiment. In at least one embodiment, each of components, features, and systems of vehicle **1600** in FIG. **16**A are illustrated as being connected via a bus **1602**. In at least one embodiment, bus **1602** may include, without limitation, a CAN data interface (alternatively referred to herein as a "CAN bus"). In at least one embodiment, a CAN bus may be a network inside vehicle **1600** used to aid in control of various features and functionality of vehicle **1600**, such as actuation of brakes, acceleration, braking, steering, windshield wipers, etc. In at least one embodiment, bus **1602** may be configured to have dozens or even hundreds of nodes, each with its own unique identifier (e.g., a CAN ID). In at least one embodiment, bus **1602** may be read to find steering wheel angle, ground speed, engine revolutions per minute ("RPMs"), button positions, and/or other vehicle status indicators. In at least one embodiment, bus **1602** may be a CAN bus that is ASIL B compliant.

In at least one embodiment, in addition to, or alternatively from CAN, FlexRay and/or Ethernet may be used. In at least one embodiment, there may be any number of busses **1602**, which may include, without limitation, zero or more CAN busses, zero or more FlexRay busses, zero or more Ethernet busses, and/or zero or more other types of busses using a different protocol. In at least one embodiment, two or more busses **1602** may be used to perform different functions, and/or may be used for redundancy. For example, a first bus **1602** may be used for collision avoidance functionality and a second bus **1602** may be used for actuation control. In at least one embodiment, each bus **1602** may communicate with any of components of vehicle **1600**, and two or more busses **1602** may communicate with same components. In at least one embodiment, each of any number of system(s) on chip(s) ("SoC(s)") **1604**, each of controller(s) **1636**, and/or each computer within vehicle may have access to same input data (e.g., inputs from sensors of vehicle **1600**), and may be connected to a common bus, such CAN bus.

In at least one embodiment, vehicle **1600** may include one or more controller(s) **1636**, such as those described herein with respect to FIG. **1**A. Controller(s) **1636** may be used for a variety of functions. In at least one embodiment, controller(s) **1636** may be coupled to any of various other components and systems of vehicle **1600**, and may be used for control of vehicle **1600**, artificial intelligence of vehicle **1600**, infotainment for vehicle **1600**, and/or like.

In at least one embodiment, vehicle **1600** may include any number of SoCs **1604**. Each of SoCs **1604** may include, without limitation, central processing units ("CPU(s)") **1606**, graphics processing units ("GPU(s)") **1608**, processor(s) **1610**, cache(s) **1612**, accelerator(s) **1614**, data store(s) **1616**, and/or other components and features not illustrated. In at least one embodiment, SoC(s) **1604** may be used to control vehicle **1600** in a variety of platforms and systems. For example, in at least one embodiment, SoC(s) **1604** may be combined in a system (e.g., system of vehicle **1600**) with a High Definition ("HD") map **1622** which may obtain map refreshes and/or updates via network interface **1624** from one or more servers (not shown in FIG. **16**A).

In at least one embodiment, CPU(s) **1606** may include a CPU cluster or CPU complex (alternatively referred to herein as a "CCPLEX"). In at least one embodiment, CPU(s) **1606** may include multiple cores and/or level two ("L2") caches. For instance, in at least one embodiment, CPU(s) **1606** may include eight cores in a coherent multi-processor configuration. In at least one embodiment, CPU(s) **1606** may include four dual-core clusters where each cluster has a dedicated L2 cache (e.g., a 2 MB L2 cache). In at least one embodiment, CPU(s) **1606** (e.g., CCPLEX) may be config-

ured to support simultaneous cluster operation enabling any combination of clusters of CPU(s) **1606** to be active at any given time.

In at least one embodiment, one or more of CPU(s) **1606** may implement power management capabilities that include, without limitation, one or more of following features: individual hardware blocks may be clock-gated automatically when idle to save dynamic power; each core clock may be gated when core is not actively executing instructions due to execution of Wait for Interrupt ("WFI")/Wait for Event ("WFE") instructions; each core may be independently power-gated; each core cluster may be independently clock-gated when all cores are clock-gated or power-gated; and/or each core cluster may be independently power-gated when all cores are power-gated. In at least one embodiment, CPU(s) **1606** may further implement an enhanced algorithm for managing power states, where allowed power states and expected wakeup times are specified, and hardware/microcode determines best power state to enter for core, cluster, and CCPLEX. In at least one embodiment, processing cores may support simplified power state entry sequences in software with work offloaded to microcode.

In at least one embodiment, GPU(s) **1608** may include an integrated GPU (alternatively referred to herein as an "iGPU"). In at least one embodiment, GPU(s) **1608** may be programmable and may be efficient for parallel workloads. In at least one embodiment, GPU(s) **1608**, in at least one embodiment, may use an enhanced tensor instruction set. In at least one embodiment, GPU(s) **1608** may include one or more streaming microprocessors, where each streaming microprocessor may include a level one ("L1") cache (e.g., an L1 cache with at least 96 KB storage capacity), and two or more of streaming microprocessors may share an L2 cache (e.g., an L2 cache with a 512 KB storage capacity). In at least one embodiment, GPU(s) **1608** may include at least eight streaming microprocessors. In at least one embodiment, GPU(s) **1608** may use compute application programming interface(s) (API(s)). In at least one embodiment, GPU(s) **1608** may use one or more parallel computing platforms and/or programming models (e.g., NVIDIA's CUDA).

In at least one embodiment, one or more of GPU(s) **1608** may be power-optimized for best performance in automotive and embedded use cases. For example, in on embodiment, GPU(s) **1608** could be fabricated on a Fin field-effect transistor ("FinFET"). In at least one embodiment, each streaming microprocessor may incorporate a number of mixed-precision processing cores partitioned into multiple blocks. For example, and without limitation, 64 PF32 cores and 32 PF64 cores could be partitioned into four processing blocks. In at least one embodiment, each processing block could be allocated 16 FP32 cores, 8 FP64 cores, 16 INT32 cores, two mixed-precision NVIDIA TENSOR CORES for deep learning matrix arithmetic, a level zero ("L0") instruction cache, a warp scheduler, a dispatch unit, and/or a 64 KB register file. In at least one embodiment, streaming microprocessors may include independent parallel integer and floating-point data paths to provide for efficient execution of workloads with a mix of computation and addressing calculations. In at least one embodiment, streaming microprocessors may include independent thread scheduling capability to enable finer-grain synchronization and cooperation between parallel threads. In at least one embodiment, streaming microprocessors may include a combined L1 data cache and shared memory unit in order to improve performance while simplifying programming.

In at least one embodiment, one or more of GPU(s) **1608** may include a high bandwidth memory ("HBM) and/or a 16 GB HBM2 memory subsystem to provide, in some examples, about 900 GB/second peak memory bandwidth. In at least one embodiment, in addition to, or alternatively from, HBM memory, a synchronous graphics random-access memory ("SGRAM") may be used, such as a graphics double data rate type five synchronous random-access memory ("GDDR5").

In at least one embodiment, GPU(s) **1608** may include unified memory technology. In at least one embodiment, address translation services ("ATS") support may be used to allow GPU(s) **1608** to access CPU(s) **1606** page tables directly. In at least one embodiment, embodiment, when GPU(s) **1608** memory management unit ("MMU") experiences a miss, an address translation request may be transmitted to CPU(s) **1606**. In response, CPU(s) **1606** may look in its page tables for virtual-to-physical mapping for address and transmits translation back to GPU(s) **1608**, in at least one embodiment. In at least one embodiment, unified memory technology may allow a single unified virtual address space for memory of both CPU(s) **1606** and GPU(s) **1608**, thereby simplifying GPU(s) **1608** programming and porting of applications to GPU(s) **1608**.

In at least one embodiment, GPU(s) **1608** may include any number of access counters that may keep track of frequency of access of GPU(s) **1608** to memory of other processors. In at least one embodiment, access counter(s) may help ensure that memory pages are moved to physical memory of processor that is accessing pages most frequently, thereby improving efficiency for memory ranges shared between processors.

In at least one embodiment, one or more of SoC(s) **1604** may include any number of cache(s) **1612**, including those described herein. For example, in at least one embodiment, cache(s) **1612** could include a level three ("L3") cache that is available to both CPU(s) **1606** and GPU(s) **1608** (e.g., that is connected both CPU(s) **1606** and GPU(s) **1608**). In at least one embodiment, cache(s) **1612** may include a write-back cache that may keep track of states of lines, such as by using a cache coherence protocol (e.g., MEI, MESI, MSI, etc.). In at least one embodiment, L3 cache may include 4 MB or more, depending on embodiment, although smaller cache sizes may be used.

In at least one embodiment, one or more of SoC(s) **1604** may include one or more accelerator(s) **1614** (e.g., hardware accelerators, software accelerators, or a combination thereof). In at least one embodiment, SoC(s) **1604** may include a hardware acceleration cluster that may include optimized hardware accelerators and/or large on-chip memory. In at least one embodiment, large on-chip memory (e.g., 4 MB of SRAM), may enable hardware acceleration cluster to accelerate neural networks and other calculations. In at least one embodiment, hardware acceleration cluster may be used to complement GPU(s) **1608** and to off-load some of tasks of GPU(s) **1608** (e.g., to free up more cycles of GPU(s) **1608** for performing other tasks). In at least one embodiment, accelerator(s) **1614** could be used for targeted workloads (e.g., perception, convolutional neural networks ("CNNs"), recurrent neural networks ("RNNs"), etc.) that are stable enough to be amenable to acceleration. In at least one embodiment, a CNN may include a region-based or regional convolutional neural networks ("RCNNs") and Fast RCNNs (e.g., as used for object detection) or other type of CNN.

In at least one embodiment, accelerator(s) **1614** (e.g., hardware acceleration cluster) may include a deep learning

accelerator(s) ("DLA(s)"). DLA(s) may include, without limitation, one or more Tensor processing units ("TPU(s)") that may be configured to provide an additional ten trillion operations per second for deep learning applications and inferencing. In at least one embodiment, TPU(s) may be accelerators configured to, and optimized for, performing image processing functions (e.g., for CNNs, RCNNs, etc.). DLA(s) may further be optimized for a specific set of neural network types and floating point operations, as well as inferencing. In at least one embodiment, design of DLA(s) may provide more performance per millimeter than a typical general-purpose GPU, and typically vastly exceeds performance of a CPU. In at least one embodiment, TPU(s) may perform several functions, including a single-instance convolution function, supporting, for example, INT8, INT16, and FP16 data types for both features and weights, as well as post-processor functions. In at least one embodiment, DLA(s) may quickly and efficiently execute neural networks, especially CNNs, on processed or unprocessed data for any of a variety of functions, including, for example and without limitation: a CNN for object identification and detection using data from camera sensors; a CNN for distance estimation using data from camera sensors; a CNN for emergency vehicle detection and identification and detection using data from microphones **1696**; a CNN for facial recognition and vehicle owner identification using data from camera sensors; and/or a CNN for security and/or safety related events.

In at least one embodiment, DLA(s) may perform any function of GPU(s) **1608**, and by using an inference accelerator, for example, a designer may target either DLA(s) or GPU(s) **1608** for any function. For example, in at least one embodiment, designer may focus processing of CNNs and floating point operations on DLA(s) and leave other functions to GPU(s) **1608** and/or other accelerator(s) **1614**.

In at least one embodiment, accelerator(s) **1614** (e.g., hardware acceleration cluster) may include a programmable vision accelerator(s) ("PVA"), which may alternatively be referred to herein as a computer vision accelerator. In at least one embodiment, PVA(s) may be designed and configured to accelerate computer vision algorithms for advanced driver assistance system ("ADAS") **1638**, autonomous driving, augmented reality ("AR") applications, and/or virtual reality ("VR") applications. PVA(s) may provide a balance between performance and flexibility. For example, in at least one embodiment, each PVA(s) may include, for example and without limitation, any number of reduced instruction set computer ("RISC") cores, direct memory access ("DMA"), and/or any number of vector processors.

In at least one embodiment, RISC cores may interact with image sensors (e.g., image sensors of any of cameras described herein), image signal processor(s), and/or like. In at least one embodiment, each of RISC cores may include any amount of memory. In at least one embodiment, RISC cores may use any of a number of protocols, depending on embodiment. In at least one embodiment, RISC cores may execute a real-time operating system ("RTOS"). In at least one embodiment, RISC cores may be implemented using one or more integrated circuit devices, application specific integrated circuits ("ASICs"), and/or memory devices. For example, in at least one embodiment, RISC cores could include an instruction cache and/or a tightly coupled RAM.

In at least one embodiment, DMA may enable components of PVA(s) to access system memory independently of CPU(s) **1606**. In at least one embodiment, DMA may support any number of features used to provide optimization to PVA including, but not limited to, supporting multi-dimensional addressing and/or circular addressing. In at least one embodiment, DMA may support up to six or more dimensions of addressing, which may include, without limitation, block width, block height, block depth, horizontal block stepping, vertical block stepping, and/or depth stepping.

In at least one embodiment, vector processors may be programmable processors that may be designed to efficiently and flexibly execute programming for computer vision algorithms and provide signal processing capabilities. In at least one embodiment, PVA may include a PVA core and two vector processing subsystem partitions. In at least one embodiment, PVA core may include a processor subsystem, DMA engine(s) (e.g., two DMA engines), and/or other peripherals. In at least one embodiment, vector processing subsystem may operate as primary processing engine of PVA, and may include a vector processing unit ("VPU"), an instruction cache, and/or vector memory (e.g., "VMEM"). In at least one embodiment, VPU may include a digital signal processor such as, for example, a single instruction, multiple data ("SIMD"), very long instruction word ("VLIW") digital signal processor. In at least one embodiment, a combination of SIMD and VLIW may enhance throughput and speed.

In at least one embodiment, each of vector processors may include an instruction cache and may be coupled to dedicated memory. As a result, in at least one embodiment, each of vector processors may be configured to execute independently of other vector processors. In at least one embodiment, vector processors that are included in a particular PVA may be configured to employ data parallelism. For instance, in at least one embodiment, plurality of vector processors included in a single PVA may execute same computer vision algorithm, but on different regions of an image. In at least one embodiment, vector processors included in a particular PVA may simultaneously execute different computer vision algorithms, on same image, or even execute different algorithms on sequential images or portions of an image. In at least one embodiment, among other things, any number of PVAs may be included in hardware acceleration cluster and any number of vector processors may be included in each of PVAs. In at least one embodiment, PVA(s) may include additional error correcting code ("ECC") memory, to enhance overall system safety.

In at least one embodiment, accelerator(s) **1614** (e.g., hardware acceleration cluster) may include a computer vision network on-chip and static random-access memory ("SRAM"), for providing a high-bandwidth, low latency SRAM for accelerator(s) **1614**. In at least one embodiment, on-chip memory may include at least 4 MB SRAM, consisting of, for example and without limitation, eight field-configurable memory blocks, that may be accessible by both PVA and DLA. In at least one embodiment, each pair of memory blocks may include an advanced peripheral bus ("APB") interface, configuration circuitry, a controller, and a multiplexer. In at least one embodiment, any type of memory may be used. In at least one embodiment, PVA and DLA may access memory via a backbone that provides PVA and DLA with high-speed access to memory. In at least one embodiment, backbone may include a computer vision network on-chip that interconnects PVA and DLA to memory (e.g., using APB).

In at least one embodiment, computer vision network on-chip may include an interface that determines, before transmission of any control signal/address/data, that both PVA and DLA provide ready and valid signals. In at least one embodiment, an interface may provide for separate

phases and separate channels for transmitting control signals/addresses/data, as well as burst-type communications for continuous data transfer. In at least one embodiment, an interface may comply with International Organization for Standardization ("ISO") 26262 or International Electrotechnical Commission ("IEC") 61508 standards, although other standards and protocols may be used.

In at least one embodiment, one or more of SoC(s) **1604** may include a real-time ray-tracing hardware accelerator. In at least one embodiment, real-time ray-tracing hardware accelerator may be used to quickly and efficiently determine positions and extents of objects (e.g., within a world model), to generate real-time visualization simulations, for RADAR signal interpretation, for sound propagation synthesis and/or analysis, for simulation of SONAR systems, for general wave propagation simulation, for comparison to LIDAR data for purposes of localization and/or other functions, and/or for other uses.

In at least one embodiment, accelerator(s) **1614** (e.g., hardware accelerator cluster) have a wide array of uses for autonomous driving. In at least one embodiment, PVA may be a programmable vision accelerator that may be used for key processing stages in ADAS and autonomous vehicles. In at least one embodiment, PVA's capabilities are a good match for algorithmic domains needing predictable processing, at low power and low latency. In other words, PVA performs well on semi-dense or dense regular computation, even on small data sets, which need predictable run-times with low latency and low power. In at least one embodiment, autonomous vehicles, such as vehicle **1600**, PVAs are designed to run classic computer vision algorithms, as they are efficient at object detection and operating on integer math.

For example, according to at least one embodiment of technology, PVA is used to perform computer stereo vision. In at least one embodiment, semi-global matching-based algorithm may be used in some examples, although this is not intended to be limiting. In at least one embodiment, applications for Level 3-5 autonomous driving use motion estimation/stereo matching on-the-fly (e.g., structure from motion, pedestrian recognition, lane detection, etc.). In at least one embodiment, PVA may perform computer stereo vision function on inputs from two monocular cameras.

In at least one embodiment, PVA may be used to perform dense optical flow. For example, in at least one embodiment, PVA could process raw RADAR data (e.g., using a 4D Fast Fourier Transform) to provide processed RADAR data. In at least one embodiment, PVA is used for time of flight depth processing, by processing raw time of flight data to provide processed time of flight data, for example.

In at least one embodiment, DLA may be used to run any type of network to enhance control and driving safety, including for example and without limitation, a neural network that outputs a measure of confidence for each object detection. In at least one embodiment, confidence may be represented or interpreted as a probability, or as providing a relative "weight" of each detection compared to other detections. In at least one embodiment, confidence enables a system to make further decisions regarding which detections should be considered as true positive detections rather than false positive detections. For example, In at least one embodiment, a system may set a threshold value for confidence and consider only detections exceeding threshold value as true positive detections. In an embodiment in which an automatic emergency braking ("AEB") system is used, false positive detections would cause vehicle to automatically perform emergency braking, which is obviously undesirable. In at least one embodiment, highly confident detections may be considered as triggers for AEB. In at least one embodiment, DLA may run a neural network for regressing confidence value. In at least one embodiment, neural network may take as its input at least some subset of parameters, such as bounding box dimensions, ground plane estimate obtained (e.g. from another subsystem), output from IMU sensor(s) **1666** that correlates with vehicle **1600** orientation, distance, 3D location estimates of object obtained from neural network and/or other sensors (e.g., LIDAR sensor(s) **1664** or RADAR sensor(s) **1660**), among others.

In at least one embodiment, one or more of SoC(s) **1604** may include data store(s) **1616** (e.g., memory). In at least one embodiment, data store(s) **1616** may be on-chip memory of SoC(s) **1604**, which may store neural networks to be executed on GPU(s) **1608** and/or DLA. In at least one embodiment, data store(s) **1616** may be large enough in capacity to store multiple instances of neural networks for redundancy and safety. In at least one embodiment, data store(s) **1616** may comprise L2 or L3 cache(s).

In at least one embodiment, one or more of SoC(s) **1604** may include any number of processor(s) **1610** (e.g., embedded processors). In at least one embodiment, processor(s) **1610** may include a boot and power management processor that may be a dedicated processor and subsystem to handle boot power and management functions and related security enforcement. In at least one embodiment, boot and power management processor may be a part of SoC(s) **1604** boot sequence and may provide runtime power management services. In at least one embodiment, boot power and management processor may provide clock and voltage programming, assistance in system low power state transitions, management of SoC(s) **1604** thermals and temperature sensors, and/or management of SoC(s) **1604** power states. In at least one embodiment, each temperature sensor may be implemented as a ring-oscillator whose output frequency is proportional to temperature, and SoC(s) **1604** may use ring-oscillators to detect temperatures of CPU(s) **1606**, GPU(s) **1608**, and/or accelerator(s) **1614**. In at least one embodiment, if temperatures are determined to exceed a threshold, then boot and power management processor may enter a temperature fault routine and put SoC(s) **1604** into a lower power state and/or put vehicle **1600** into a chauffeur to safe stop mode (e.g., bring vehicle **1600** to a safe stop).

In at least one embodiment, processor(s) **1610** may further include a set of embedded processors that may serve as an audio processing engine. In at least one embodiment, audio processing engine may be an audio subsystem that enables full hardware support for multi-channel audio over multiple interfaces, and a broad and flexible range of audio I/O interfaces. In at least one embodiment, audio processing engine is a dedicated processor core with a digital signal processor with dedicated RAM.

In at least one embodiment, processor(s) **1610** may further include an always on processor engine that may provide necessary hardware features to support low power sensor management and wake use cases. In at least one embodiment, always on processor engine may include, without limitation, a processor core, a tightly coupled RAM, supporting peripherals (e.g., timers and interrupt controllers), various I/O controller peripherals, and routing logic.

In at least one embodiment, processor(s) **1610** may further include a safety cluster engine that includes, without limitation, a dedicated processor subsystem to handle safety management for automotive applications. In at least one embodiment, safety cluster engine may include, without

limitation, two or more processor cores, a tightly coupled RAM, support peripherals (e.g., timers, an interrupt controller, etc.), and/or routing logic. In a safety mode, two or more cores may operate, in at least one embodiment, in a lockstep mode and function as a single core with comparison logic to detect any differences between their operations. In at least one embodiment, processor(s) **1610** may further include a real-time camera engine that may include, without limitation, a dedicated processor subsystem for handling real-time camera management. In at least one embodiment, processor(s) **1610** may further include a high-dynamic range signal processor that may include, without limitation, an image signal processor that is a hardware engine that is part of camera processing pipeline.

In at least one embodiment, processor(s) **1610** may include a video image compositor that may be a processing block (e.g., implemented on a microprocessor) that implements video post-processing functions needed by a video playback application to produce final image for player window. In at least one embodiment, video image compositor may perform lens distortion correction on wide-view camera(s) **1670**, surround camera(s) **1674**, and/or on in-cabin monitoring camera sensor(s). In at least one embodiment, in-cabin monitoring camera sensor(s) are preferably monitored by a neural network running on another instance of SoC(s) **1604**, configured to identify in cabin events and respond accordingly. In at least one embodiment, an in-cabin system may perform, without limitation, lip reading to activate cellular service and place a phone call, dictate emails, change vehicle's destination, activate or change vehicle's infotainment system and settings, or provide voice-activated web surfing. In at least one embodiment, certain functions are available to driver when vehicle is operating in an autonomous mode and are disabled otherwise.

In at least one embodiment, video image compositor may include enhanced temporal noise reduction for both spatial and temporal noise reduction. For example, in at least one embodiment, where motion occurs in a video, noise reduction weights spatial information appropriately, decreasing weight of information provided by adjacent frames. In at least one embodiment, where an image or portion of an image does not include motion, temporal noise reduction performed by video image compositor may use information from previous image to reduce noise in current image.

In at least one embodiment, video image compositor may also be configured to perform stereo rectification on input stereo lens frames. In at least one embodiment, video image compositor may further be used for user interface composition when operating system desktop is in use, and GPU(s) **1608** are not required to continuously render new surfaces. In at least one embodiment, when GPU(s) **1608** are powered on and active doing 3D rendering, video image compositor may be used to offload GPU(s) **1608** to improve performance and responsiveness.

In at least one embodiment, one or more of SoC(s) **1604** may further include a mobile industry processor interface ("MIPI") camera serial interface for receiving video and input from cameras, a high-speed interface, and/or a video input block that may be used for camera and related pixel input functions. In at least one embodiment, one or more of SoC(s) **1604** may further include an input/output controller(s) that may be controlled by software and may be used for receiving I/O signals that are uncommitted to a specific role.

In at least one embodiment, one or more of SoC(s) **1604** may further include a broad range of peripheral interfaces to enable communication with peripherals, audio encoders/decoders ("codecs"), power management, and/or other devices. SoC(s) **1604** may be used to process data from cameras (e.g., connected over Gigabit Multimedia Serial Link and Ethernet), sensors (e.g., LIDAR sensor(s) **1664**, RADAR sensor(s) **1660**, etc. that may be connected over Ethernet), data from bus **1602** (e.g., speed of vehicle **1600**, steering wheel position, etc.), data from GNSS sensor(s) **1658** (e.g., connected over Ethernet or CAN bus), etc. In at least one embodiment, one or more of SoC(s) **1604** may further include dedicated high-performance mass storage controllers that may include their own DMA engines, and that may be used to free CPU(s) **1606** from routine data management tasks.

In at least one embodiment, SoC(s) **1604** may be an end-to-end platform with a flexible architecture that spans automation levels 3-5, thereby providing a comprehensive functional safety architecture that leverages and makes efficient use of computer vision and ADAS techniques for diversity and redundancy, provides a platform for a flexible, reliable driving software stack, along with deep learning tools. In at least one embodiment, SoC(s) **1604** may be faster, more reliable, and even more energy-efficient and space-efficient than conventional systems. For example, in at least one embodiment, accelerator(s) **1614**, when combined with CPU(s) **1606**, GPU(s) **1608**, and data store(s) **1616**, may provide for a fast, efficient platform for level 3-5 autonomous vehicles.

In at least one embodiment, computer vision algorithms may be executed on CPUs, which may be configured using high-level programming language, such as C programming language, to execute a wide variety of processing algorithms across a wide variety of visual data. However, in at least one embodiment, CPUs are oftentimes unable to meet performance requirements of many computer vision applications, such as those related to execution time and power consumption, for example. In at least one embodiment, many CPUs are unable to execute complex object detection algorithms in real-time, which is used in in-vehicle ADAS applications and in practical Level 3-5 autonomous vehicles.

Embodiments described herein allow for multiple neural networks to be performed simultaneously and/or sequentially, and for results to be combined together to enable Level 3-5 autonomous driving functionality. For example, in at least one embodiment, a CNN executing on DLA or discrete GPU (e.g., GPU(s) **1620**) may include text and word recognition, allowing supercomputer to read and understand traffic signs, including signs for which neural network has not been specifically trained. In at least one embodiment, DLA may further include a neural network that is able to identify, interpret, and provide semantic understanding of sign, and to pass that semantic understanding to path planning modules running on CPU Complex.

In at least one embodiment, multiple neural networks may be run simultaneously, as for Level 3, 4, or 5 driving. For example, in at least one embodiment, a warning sign consisting of "Caution: flashing lights indicate icy conditions," along with an electric light, may be independently or collectively interpreted by several neural networks. In at least one embodiment, a sign itself may be identified as a traffic sign by a first deployed neural network (e.g., a neural network that has been trained) and a text "flashing lights indicate icy conditions" may be interpreted by a second deployed neural network, which informs vehicle's path planning software (preferably executing on CPU Complex) that when flashing lights are detected, icy conditions exist. In at least one embodiment, a flashing light may be identified by operating a third deployed neural network over multiple

frames, informing vehicle's path-planning software of presence (or absence) of flashing lights. In at least one embodiment, all three neural networks may run simultaneously, such as within DLA and/or on GPU(s) **1608**.

In at least one embodiment, a CNN for facial recognition and vehicle owner identification may use data from camera sensors to identify presence of an authorized driver and/or owner of vehicle **1600**. In at least one embodiment, an always on sensor processing engine may be used to unlock vehicle when owner approaches driver door and turn on lights, and, in security mode, to disable vehicle when owner leaves vehicle. In this way, SoC(s) **1604** provide for security against theft and/or carjacking.

In at least one embodiment, a CNN for emergency vehicle detection and identification may use data from microphones **1696** to detect and identify emergency vehicle sirens. In at least one embodiment, SoC(s) **1604** use CNN for classifying environmental and urban sounds, as well as classifying visual data. In at least one embodiment, CNN running on DLA is trained to identify relative closing speed of emergency vehicle (e.g., by using Doppler effect). In at least one embodiment, CNN may also be trained to identify emergency vehicles specific to local area in which vehicle is operating, as identified by GNSS sensor(s) **1658**. In at least one embodiment, when operating in Europe, CNN will seek to detect European sirens, and when in United States CNN will seek to identify only North American sirens. In at least one embodiment, once an emergency vehicle is detected, a control program may be used to execute an emergency vehicle safety routine, slowing vehicle, pulling over to side of road, parking vehicle, and/or idling vehicle, with assistance of ultrasonic sensor(s) **1662**, until emergency vehicle(s) passes.

In at least one embodiment, vehicle **1600** may include CPU(s) **1618** (e.g., discrete CPU(s), or dCPU(s)), that may be coupled to SoC(s) **1604** via a high-speed interconnect (e.g., PCIe). In at least one embodiment, CPU(s) **1618** may include an X86 processor, for example. CPU(s) **1618** may be used to perform any of a variety of functions, including arbitrating potentially inconsistent results between ADAS sensors and SoC(s) **1604**, and/or monitoring status and health of controller(s) **1636** and/or an infotainment system on a chip ("infotainment SoC") **1630**, for example.

In at least one embodiment, vehicle **1600** may include GPU(s) **1620** (e.g., discrete GPU(s), or dGPU(s)), that may be coupled to SoC(s) **1604** via a high-speed interconnect (e.g., NVIDIA's NVLINK). In at least one embodiment, GPU(s) **1620** may provide additional artificial intelligence functionality, such as by executing redundant and/or different neural networks, and may be used to train and/or update neural networks based at least in part on input (e.g., sensor data) from sensors of vehicle **1600**.

In at least one embodiment, vehicle **1600** may further include network interface **1624** which may include, without limitation, wireless antenna(s) **1626** (e.g., one or more wireless antennas **1626** for different communication protocols, such as a cellular antenna, a Bluetooth antenna, etc.). In at least one embodiment, network interface **1624** may be used to enable wireless connectivity over Internet with cloud (e.g., with server(s) and/or other network devices), with other vehicles, and/or with computing devices (e.g., client devices of passengers). In at least one embodiment, to communicate with other vehicles, a direct link may be established between vehicle **160** and other vehicle and/or an indirect link may be established (e.g., across networks and over Internet). In at least one embodiment, direct links may be provided using a vehicle-to-vehicle communication link. A vehicle-to-vehicle communication link may provide vehicle **1600** information about vehicles in proximity to vehicle **1600** (e.g., vehicles in front of, on side of, and/or behind vehicle **1600**). In at least one embodiment, aforementioned functionality may be part of a cooperative adaptive cruise control functionality of vehicle **1600**.

In at least one embodiment, network interface **1624** may include an SoC that provides modulation and demodulation functionality and enables controller(s) **1636** to communicate over wireless networks. In at least one embodiment, network interface **1624** may include a radio frequency front-end for up-conversion from baseband to radio frequency, and down conversion from radio frequency to baseband. In at least one embodiment, frequency conversions may be performed in any technically feasible fashion. For example, frequency conversions could be performed through well-known processes, and/or using super-heterodyne processes. In at least one embodiment, radio frequency front end functionality may be provided by a separate chip. In at least one embodiment, network interface may include wireless functionality for communicating over LTE, WCDMA, UMTS, GSM, CDMA2000, Bluetooth, Bluetooth LE, Wi-Fi, Z-Wave, ZigBee, LoRaWAN, and/or other wireless protocols.

In at least one embodiment, vehicle **1600** may further include data store(s) **1628** which may include, without limitation, off-chip (e.g., off SoC(s) **1604**) storage. In at least one embodiment, data store(s) **1628** may include, without limitation, one or more storage elements including RAM, SRAM, dynamic random-access memory ("DRAM"), video random-access memory ("VRAM"), Flash, hard disks, and/or other components and/or devices that may store at least one bit of data.

In at least one embodiment, vehicle **1600** may further include GNSS sensor(s) **1658** (e.g., GPS and/or assisted GPS sensors), to assist in mapping, perception, occupancy grid generation, and/or path planning functions. In at least one embodiment, any number of GNSS sensor(s) **1658** may be used, including, for example and without limitation, a GPS using a USB connector with an Ethernet to Serial (e.g., RS-232) bridge.

In at least one embodiment, vehicle **1600** may further include RADAR sensor(s) **1660**. RADAR sensor(s) **1660** may be used by vehicle **1600** for long-range vehicle detection, even in darkness and/or severe weather conditions. In at least one embodiment, RADAR functional safety levels may be ASIL B. RADAR sensor(s) **1660** may use CAN and/or bus **1602** (e.g., to transmit data generated by RADAR sensor(s) **1660**) for control and to access object tracking data, with access to Ethernet to access raw data in some examples. In at least one embodiment, wide variety of RADAR sensor types may be used. For example, and without limitation, RADAR sensor(s) **1660** may be suitable for front, rear, and side RADAR use. In at least one embodiment, one or more of RADAR sensors(s) **1660** are Pulse Doppler RADAR sensor(s).

In at least one embodiment, RADAR sensor(s) **1660** may include different configurations, such as long-range with narrow field of view, short-range with wide field of view, short-range side coverage, etc. In at least one embodiment, long-range RADAR may be used for adaptive cruise control functionality. In at least one embodiment, long-range RADAR systems may provide a broad field of view realized by two or more independent scans, such as within a 250 m range. In at least one embodiment, RADAR sensor(s) **1660** may help in distinguishing between static and moving objects, and may be used by ADAS system **1638** for emergency brake assist and forward collision warning. Sen-

sors **1660**(*s*) included in a long-range RADAR system may include, without limitation, monostatic multimodal RADAR with multiple (e.g., six or more) fixed RADAR antennae and a high-speed CAN and FlexRay interface. In at least one embodiment, with six antennae, central four antennae may create a focused beam pattern, designed to record vehicle **1600**'s surroundings at higher speeds with minimal interference from traffic in adjacent lanes. In at least one embodiment, other two antennae may expand field of view, making it possible to quickly detect vehicles entering or leaving vehicle **1600**'s lane.

In at least one embodiment, mid-range RADAR systems may include, as an example, a range of up to 160 m (front) or 80 m (rear), and a field of view of up to 42 degrees (front) or 150 degrees (rear). In at least one embodiment, short-range RADAR systems may include, without limitation, any number of RADAR sensor(s) **1660** designed to be installed at both ends of rear bumper. When installed at both ends of rear bumper, in at least one embodiment, a RADAR sensor system may create two beams that constantly monitor blind spot in rear and next to vehicle. In at least one embodiment, short-range RADAR systems may be used in ADAS system **1638** for blind spot detection and/or lane change assist.

In at least one embodiment, vehicle **1600** may further include ultrasonic sensor(s) **1662**. Ultrasonic sensor(s) **1662**, which may be positioned at front, back, and/or sides of vehicle **1600**, may be used for park assist and/or to create and update an occupancy grid. In at least one embodiment, a wide variety of ultrasonic sensor(s) **1662** may be used, and different ultrasonic sensor(s) **1662** may be used for different ranges of detection (e.g., 2.5 m, 4 m). In at least one embodiment, ultrasonic sensor(s) **1662** may operate at functional safety levels of ASIL B.

In at least one embodiment, vehicle **1600** may include LIDAR sensor(s) **1664**. LIDAR sensor(s) **1664** may be used for object and pedestrian detection, emergency braking, collision avoidance, and/or other functions. In at least one embodiment, LIDAR sensor(s) **1664** may be functional safety level ASIL B. In at least one embodiment, vehicle **1600** may include multiple LIDAR sensors **1664** (e.g., two, four, six, etc.) that may use Ethernet (e.g., to provide data to a Gigabit Ethernet switch).

In at least one embodiment, LIDAR sensor(s) **1664** may be capable of providing a list of objects and their distances for a 360-degree field of view. In at least one embodiment, commercially available LIDAR sensor(s) **1664** may have an advertised range of approximately 100 m, with an accuracy of 2 cm-3 cm, and with support for a 100 Mbps Ethernet connection, for example. In at least one embodiment, one or more non-protruding LIDAR sensors **1664** may be used. In such an embodiment, LIDAR sensor(s) **1664** may be implemented as a small device that may be embedded into front, rear, sides, and/or corners of vehicle **1600**. In at least one embodiment, LIDAR sensor(s) **1664**, in such an embodiment, may provide up to a 120-degree horizontal and 35-degree vertical field-of-view, with a 200 m range even for low-reflectivity objects. In at least one embodiment, front-mounted LIDAR sensor(s) **1664** may be configured for a horizontal field of view between 45 degrees and 135 degrees.

In at least one embodiment, LIDAR technologies, such as 3D flash LIDAR, may also be used. 3D Flash LIDAR uses a flash of a laser as a transmission source, to illuminate surroundings of vehicle **1600** up to approximately 200 m. In at least one embodiment, a flash LIDAR unit includes, without limitation, a receptor, which records laser pulse transit time and reflected light on each pixel, which in turn corresponds to range from vehicle **1600** to objects. In at least one embodiment, flash LIDAR may allow for highly accurate and distortion-free images of surroundings to be generated with every laser flash. In at least one embodiment, four flash LIDAR sensors may be deployed, one at each side of vehicle **1600**. In at least one embodiment, 3D flash LIDAR systems include, without limitation, a solid-state 3D staring array LIDAR camera with no moving parts other than a fan (e.g., a non-scanning LIDAR device). In at least one embodiment, flash LIDAR device(s) may use a 5 nanosecond class I (eye-safe) laser pulse per frame and may capture reflected laser light in form of 3D range point clouds and co-registered intensity data.

In at least one embodiment, vehicle may further include IMU sensor(s) **1666**. In at least one embodiment, IMU sensor(s) **1666** may be located at a center of rear axle of vehicle **1600**, in at least one embodiment. In at least one embodiment, IMU sensor(s) **1666** may include, for example and without limitation, accelerometer(s), magnetometer(s), gyroscope(s), magnetic compass(es), and/or other sensor types. In at least one embodiment, such as in six-axis applications, IMU sensor(s) **1666** may include, without limitation, accelerometers and gyroscopes. In at least one embodiment, such as in nine-axis applications, IMU sensor(s) **1666** may include, without limitation, accelerometers, gyroscopes, and magnetometers.

In at least one embodiment, IMU sensor(s) **1666** may be implemented as a miniature, high performance GPS-Aided Inertial Navigation System ("GPS/INS") that combines micro-electro-mechanical systems ("MEMS") inertial sensors, a high-sensitivity GPS receiver, and advanced Kalman filtering algorithms to provide estimates of position, velocity, and attitude. In at least one embodiment, IMU sensor(s) **1666** may enable vehicle **1600** to estimate heading without requiring input from a magnetic sensor by directly observing and correlating changes in velocity from GPS to IMU sensor(s) **1666**. In at least one embodiment, IMU sensor(s) **1666** and GNSS sensor(s) **1658** may be combined in a single integrated unit.

In at least one embodiment, vehicle **1600** may include microphone(s) **1696** placed in and/or around vehicle **1600**. In at least one embodiment, microphone(s) **1696** may be used for emergency vehicle detection and identification, among other things.

In at least one embodiment, vehicle **1600** may further include any number of camera types, including stereo camera(s) **1668**, wide-view camera(s) **1670**, infrared camera(s) **1672**, surround camera(s) **1674**, long-range camera(s) **1698**, mid-range camera(s) **1676**, and/or other camera types. In at least one embodiment, cameras may be used to capture image data around an entire periphery of vehicle **1600**. In at least one embodiment, types of cameras used depends on vehicle **1600**. In at least one embodiment, any combination of camera types may be used to provide necessary coverage around vehicle **1600**. In at least one embodiment, number of cameras may differ depending on embodiment. For example, in at least one embodiment, vehicle **1600** could include six cameras, seven cameras, ten cameras, twelve cameras, or another number of cameras. Cameras may support, as an example and without limitation, Gigabit Multimedia Serial Link ("GMSL") and/or Gigabit Ethernet. In at least one embodiment, each of camera(s) is described with more detail previously herein with respect to FIG. **16**A and FIG. **16**B.

In at least one embodiment, vehicle **1600** may further include vibration sensor(s) **1642**. In at least one embodiment, vibration sensor(s) **1642** may measure vibrations of components of vehicle **1600**, such as axle(s). For example,

in at least one embodiment, changes in vibrations may indicate a change in road surfaces. In at least one embodiment, when two or more vibration sensors **1642** are used, differences between vibrations may be used to determine friction or slippage of road surface (e.g., when difference in vibration is between a power-driven axle and a freely rotating axle).

In at least one embodiment, vehicle **1600** may include ADAS system **1638**. ADAS system **1638** may include, without limitation, an SoC, in some examples. In at least one embodiment, ADAS system **1638** may include, without limitation, any number and combination of an autonomous/adaptive/automatic cruise control ("ACC") system, a cooperative adaptive cruise control ("CACC") system, a forward crash warning ("FCW") system, an automatic emergency braking ("AEB") system, a lane departure warning ("LDW)" system, a lane keep assist ("LKA") system, a blind spot warning ("BSW") system, a rear cross-traffic warning ("RCTW") system, a collision warning ("CW") system, a lane centering ("LC") system, and/or other systems, features, and/or functionality.

In at least one embodiment, ACC system may use RADAR sensor(s) **1660**, LIDAR sensor(s) **1664**, and/or any number of camera(s). In at least one embodiment, ACC system may include a longitudinal ACC system and/or a lateral ACC system. In at least one embodiment, longitudinal ACC system monitors and controls distance to vehicle immediately ahead of vehicle **1600** and automatically adjust speed of vehicle **1600** to maintain a safe distance from vehicles ahead. In at least one embodiment, lateral ACC system performs distance keeping, and advises vehicle **1600** to change lanes when necessary. In at least one embodiment, lateral ACC is related to other ADAS applications such as LC and CW.

In at least one embodiment, CACC system uses information from other vehicles that may be received via network interface **1624** and/or wireless antenna(s) **1626** from other vehicles via a wireless link, or indirectly, over a network connection (e.g., over Internet). In at least one embodiment, direct links may be provided by a vehicle-to-vehicle ("V2V") communication link, while indirect links may be provided by an infrastructure-to-vehicle ("I2V") communication link. In general, V2V communication concept provides information about immediately preceding vehicles (e.g., vehicles immediately ahead of and in same lane as vehicle **1600**), while I2V communication concept provides information about traffic further ahead. In at least one embodiment, CACC system may include either or both I2V and V2V information sources. In at least one embodiment, given information of vehicles ahead of vehicle **1600**, CACC system may be more reliable and it has potential to improve traffic flow smoothness and reduce congestion on road.

In at least one embodiment, FCW system is designed to alert driver to a hazard, so that driver may take corrective action. In at least one embodiment, FCW system uses a front-facing camera and/or RADAR sensor(s) **1660**, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, FCW system may provide a warning, such as in form of a sound, visual warning, vibration and/or a quick brake pulse.

In at least one embodiment, AEB system detects an impending forward collision with another vehicle or other object, and may automatically apply brakes if driver does not take corrective action within a specified time or distance parameter. In at least one embodiment, AEB system may use front-facing camera(s) and/or RADAR sensor(s) **1660**, coupled to a dedicated processor, DSP, FPGA, and/or ASIC. In at least one embodiment, when AEB system detects a hazard, AEB system typically first alerts driver to take corrective action to avoid collision and, if driver does not take corrective action, AEB system may automatically apply brakes in an effort to prevent, or at least mitigate, impact of predicted collision. In at least one embodiment, AEB system, may include techniques such as dynamic brake support and/or crash imminent braking.

In at least one embodiment, LDW system provides visual, audible, and/or tactile warnings, such as steering wheel or seat vibrations, to alert driver when vehicle **1600** crosses lane markings. In at least one embodiment, LDW system does not activate when driver indicates an intentional lane departure, by activating a turn signal. In at least one embodiment, LDW system may use front-side facing cameras, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component. In at least one embodiment, LKA system is a variation of LDW system. LKA system provides steering input or braking to correct vehicle **1600** if vehicle **1600** starts to exit lane.

In at least one embodiment, BSW system detects and warns driver of vehicles in an automobile's blind spot. In at least one embodiment, BSW system may provide a visual, audible, and/or tactile alert to indicate that merging or changing lanes is unsafe. In at least one embodiment, BSW system may provide an additional warning when driver uses a turn signal. In at least one embodiment, BSW system may use rear-side facing camera(s) and/or RADAR sensor(s) **1660**, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component.

In at least one embodiment, RCTW system may provide visual, audible, and/or tactile notification when an object is detected outside rear-camera range when vehicle **1600** is backing up. In at least one embodiment, RCTW system includes AEB system to ensure that vehicle brakes are applied to avoid a crash. In at least one embodiment, RCTW system may use one or more rear-facing RADAR sensor(s) **1660**, coupled to a dedicated processor, DSP, FPGA, and/or ASIC, that is electrically coupled to driver feedback, such as a display, speaker, and/or vibrating component.

In at least one embodiment, conventional ADAS systems may be prone to false positive results which may be annoying and distracting to a driver, but typically are not catastrophic, because conventional ADAS systems alert driver and allow driver to decide whether a safety condition truly exists and act accordingly. In at least one embodiment, vehicle **1600** itself decides, in case of conflicting results, whether to heed result from a primary computer or a secondary computer (e.g., first controller **1636** or second controller **1636**). For example, in at least one embodiment, ADAS system **1638** may be a backup and/or secondary computer for providing perception information to a backup computer rationality module. In at least one embodiment, backup computer rationality monitor may run a redundant diverse software on hardware components to detect faults in perception and dynamic driving tasks. In at least one embodiment, outputs from ADAS system **1638** may be provided to a supervisory MCU. In at least one embodiment, if outputs from primary computer and secondary computer conflict, supervisory MCU determines how to reconcile conflict to ensure safe operation.

In at least one embodiment, primary computer may be configured to provide supervisory MCU with a confidence

score, indicating primary computer's confidence in chosen result. In at least one embodiment, if confidence score exceeds a threshold, supervisory MCU may follow primary computer's direction, regardless of whether secondary computer provides a conflicting or inconsistent result. In at least one embodiment, where confidence score does not meet threshold, and where primary and secondary computer indicate different results (e.g., a conflict), supervisory MCU may arbitrate between computers to determine appropriate outcome.

In at least one embodiment, supervisory MCU may be configured to run a neural network(s) that is trained and configured to determine, based at least in part on outputs from primary computer and secondary computer, conditions under which secondary computer provides false alarms. In at least one embodiment, neural network(s) in supervisory MCU may learn when secondary computer's output may be trusted, and when it cannot. For example, in at least one embodiment, when secondary computer is a RADAR-based FCW system, a neural network(s) in supervisory MCU may learn when FCW system is identifying metallic objects that are not, in fact, hazards, such as a drainage grate or manhole cover that triggers an alarm. In at least one embodiment, when secondary computer is a camera-based LDW system, a neural network in supervisory MCU may learn to override LDW when bicyclists or pedestrians are present and a lane departure is, in fact, safest maneuver. In at least one embodiment, supervisory MCU may include at least one of a DLA or GPU suitable for running neural network(s) with associated memory. In at least one embodiment, supervisory MCU may comprise and/or be included as a component of SoC(s) **1604**.

In at least one embodiment, ADAS system **1638** may include a secondary computer that performs ADAS functionality using traditional rules of computer vision. In at least one embodiment, secondary computer may use classic computer vision rules (if-then), and presence of a neural network(s) in supervisory MCU may improve reliability, safety and performance. For example, in at least one embodiment, diverse implementation and intentional non-identity makes overall system more fault-tolerant, especially to faults caused by software (or software-hardware interface) functionality. For example, in at least one embodiment, if there is a software bug or error in software running on primary computer, and non-identical software code running on secondary computer provides same overall result, then supervisory MCU may have greater confidence that overall result is correct, and bug in software or hardware on primary computer is not causing material error.

In at least one embodiment, output of ADAS system **1638** may be fed into primary computer's perception block and/or primary computer's dynamic driving task block. For example, in at least one embodiment, if ADAS system **1638** indicates a forward crash warning due to an object immediately ahead, perception block may use this information when identifying objects. In at least one embodiment, secondary computer may have its own neural network which is trained and thus reduces risk of false positives, as described herein.

In at least one embodiment, vehicle **1600** may further include infotainment SoC **1630** (e.g., an in-vehicle infotainment system (IVI)). Although illustrated and described as an SoC, infotainment system **1630**, in at least one embodiment, may not be an SoC, and may include, without limitation, two or more discrete components. In at least one embodiment, infotainment SoC **1630** may include, without limitation, a combination of hardware and software that may be used to provide audio (e.g., music, a personal digital assistant, navigational instructions, news, radio, etc.), video (e.g., TV, movies, streaming, etc.), phone (e.g., hands-free calling), network connectivity (e.g., LTE, WiFi, etc.), and/or information services (e.g., navigation systems, rear-parking assistance, a radio data system, vehicle related information such as fuel level, total distance covered, brake fuel level, oil level, door open/close, air filter information, etc.) to vehicle **1600**. For example, infotainment SoC **1630** could include radios, disk players, navigation systems, video players, USB and Bluetooth connectivity, carputers, in-car entertainment, WiFi, steering wheel audio controls, hands free voice control, a heads-up display ("HUD"), HMI display **1634**, a telematics device, a control panel (e.g., for controlling and/or interacting with various components, features, and/or systems), and/or other components. In at least one embodiment, infotainment SoC **1630** may further be used to provide information (e.g., visual and/or audible) to user(s) of vehicle, such as information from ADAS system **1638**, autonomous driving information such as planned vehicle maneuvers, trajectories, surrounding environment information (e.g., intersection information, vehicle information, road information, etc.), and/or other information.

In at least one embodiment, infotainment SoC **1630** may include any amount and type of GPU functionality. In at least one embodiment, infotainment SoC **1630** may communicate over bus **1602** (e.g., CAN bus, Ethernet, etc.) with other devices, systems, and/or components of vehicle **1600**. In at least one embodiment, infotainment SoC **1630** may be coupled to a supervisory MCU such that GPU of infotainment system may perform some self-driving functions in event that primary controller(s) **1636** (e.g., primary and/or backup computers of vehicle **1600**) fail. In at least one embodiment, infotainment SoC **1630** may put vehicle **1600** into a chauffeur to safe stop mode, as described herein.

In at least one embodiment, vehicle **1600** may further include instrument cluster **1632** (e.g., a digital dash, an electronic instrument cluster, a digital instrument panel, etc.). In at least one embodiment, instrument cluster **1632** may include, without limitation, a controller and/or supercomputer (e.g., a discrete controller or supercomputer). In at least one embodiment, instrument cluster **1632** may include, without limitation, any number and combination of a set of instrumentation such as a speedometer, fuel level, oil pressure, tachometer, odometer, turn indicators, gearshift position indicator, seat belt warning light(s), parking-brake warning light(s), engine-malfunction light(s), supplemental restraint system (e.g., airbag) information, lighting controls, safety system controls, navigation information, etc. In some examples, information may be displayed and/or shared among infotainment SoC **1630** and instrument cluster **1632**. In at least one embodiment, instrument cluster **1632** may be included as part of infotainment SoC **1630**, or vice versa.

Inference and/or training logic **715** are used to perform inferencing and/or training operations associated with one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B. In at least one embodiment, inference and/or training logic **715** may be used in system FIG. **16**A for inferencing or predicting operations based, at least in part, on weight parameters calculated using neural network training operations, neural network functions and/or architectures, or neural network use cases described herein.

FIG. **16**B is a diagram of a system **1676** for communication between cloud-based server(s) and autonomous vehicle **1600** of FIG. **16**A, according to at least one embodiment. In at least one embodiment, system **1676** may include,

without limitation, server(s) **1678**, network(s) **1690**, and any number and type of vehicles, including vehicle **1600**. In at least one embodiment, server(s) **1678** may include, without limitation, a plurality of GPUs **1684**(A)-**1684**(H) (collectively referred to herein as GPUs **1684**), PCIe switches **1682**(A)-**1682**(D) (collectively referred to herein as PCIe switches **1682**), and/or CPUs **1680**(A)-**1680**(B) (collectively referred to herein as CPUs **1680**). GPUs **1684**, CPUs **1680**, and PCIe switches **1682** may be interconnected with high-speed interconnects such as, for example and without limitation, NVLink interfaces **1688** developed by NVIDIA and/or PCIe connections **1686**. In at least one embodiment, GPUs **1684** are connected via an NVLink and/or NVSwitch SoC and GPUs **1684** and PCIe switches **1682** are connected via PCIe interconnects. In at least one embodiment, although eight GPUs **1684**, two CPUs **1680**, and four PCIe switches **1682** are illustrated, this is not intended to be limiting. In at least one embodiment, each of server(s) **1678** may include, without limitation, any number of GPUs **1684**, CPUs **1680**, and/or PCIe switches **1682**, in any combination. For example, in at least one embodiment, server(s) **1678** could each include eight, sixteen, thirty-two, and/or more GPUs **1684**.

In at least one embodiment, server(s) **1678** may receive, over network(s) **1690** and from vehicles, image data representative of images showing unexpected or changed road conditions, such as recently commenced road-work. In at least one embodiment, server(s) **1678** may transmit, over network(s) **1690** and to vehicles, neural networks **1692**, updated neural networks **1692**, and/or map information **1694**, including, without limitation, information regarding traffic and road conditions. In at least one embodiment, updates to map information **1694** may include, without limitation, updates for HD map **1622**, such as information regarding construction sites, potholes, detours, flooding, and/or other obstructions. In at least one embodiment, neural networks **1692**, updated neural networks **1692**, and/or map information **1694** may have resulted from new training and/or experiences represented in data received from any number of vehicles in environment, and/or based at least in part on training performed at a data center (e.g., using server(s) **1678** and/or other servers).

In at least one embodiment, server(s) **1678** may be used to train machine learning models (e.g., neural networks) based at least in part on training data. In at least one embodiment, training data may be generated by vehicles, and/or may be generated in a simulation (e.g., using a game engine). In at least one embodiment, any amount of training data is tagged (e.g., where associated neural network benefits from supervised learning) and/or undergoes other pre-processing. In at least one embodiment, any amount of training data is not tagged and/or pre-processed (e.g., where associated neural network does not require supervised learning). In at least one embodiment, once machine learning models are trained, machine learning models may be used by vehicles (e.g., transmitted to vehicles over network(s) **1690**, and/or machine learning models may be used by server(s) **1678** to remotely monitor vehicles.

In at least one embodiment, server(s) **1678** may receive data from vehicles and apply data to up-to-date real-time neural networks for real-time intelligent inferencing. In at least one embodiment, server(s) **1678** may include deep-learning supercomputers and/or dedicated AI computers powered by GPU(s) **1684**, such as a DGX and DGX Station machines developed by NVIDIA. However, in at least one embodiment, server(s) **1678** may include deep learning infrastructure that use CPU-powered data centers.

In at least one embodiment, deep-learning infrastructure of server(s) **1678** may be capable of fast, real-time inferencing, and may use that capability to evaluate and verify health of processors, software, and/or associated hardware in vehicle **1600**. For example, in at least one embodiment, deep-learning infrastructure may receive periodic updates from vehicle **1600**, such as a sequence of images and/or objects that vehicle **1600** has located in that sequence of images (e.g., via computer vision and/or other machine learning object classification techniques). In at least one embodiment, deep-learning infrastructure may run its own neural network to identify objects and compare them with objects identified by vehicle **1600** and, if results do not match and deep-learning infrastructure concludes that AI in vehicle **1600** is malfunctioning, then server(s) **1678** may transmit a signal to vehicle **1600** instructing a fail-safe computer of vehicle **1600** to assume control, notify passengers, and complete a safe parking maneuver.

In at least one embodiment, server(s) **1678** may include GPU(s) **1684** and one or more programmable inference accelerators (e.g., NVIDIA's TensorRT 3). In at least one embodiment, combination of GPU-powered servers and inference acceleration may make real-time responsiveness possible. In at least one embodiment, such as where performance is less critical, servers powered by CPUs, FPGAs, and other processors may be used for inferencing. In at least one embodiment, inference and/or training logic **715** are used to perform one or more embodiments. Details regarding inference and/or training logic **715** are provided below in conjunction with FIGS. **7**A and/or **7**B.

Other variations are within spirit of present disclosure. Thus, while disclosed techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in drawings and have been described above in detail. It should be understood, however, that there is no intention to limit disclosure to specific form or forms disclosed, but on contrary, intention is to cover all modifications, alternative constructions, and equivalents falling within spirit and scope of disclosure, as defined in appended claims.

Use of terms "a" and "an" and "the" and similar referents in context of describing disclosed embodiments (especially in context of following claims) are to be construed to cover both singular and plural, unless otherwise indicated herein or clearly contradicted by context, and not as a definition of a term. Terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (meaning "including, but not limited to,") unless otherwise noted. Term "connected," when unmodified and referring to physical connections, is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within range, unless otherwise indicated herein and each separate value is incorporated into specification as if it were individually recited herein. Use of term "set" (e.g., "a set of items") or "subset," unless otherwise noted or contradicted by context, is to be construed as a nonempty collection comprising one or more members. Further, unless otherwise noted or contradicted by context, term "subset" of a corresponding set does not necessarily denote a proper subset of corresponding set, but subset and corresponding set may be equal.

Conjunctive language, such as phrases of form "at least one of A, B, and C," or "at least one of A, B and C," unless specifically stated otherwise or otherwise clearly contra-

dicted by context, is otherwise understood with context as used in general to present that an item, term, etc., may be either A or B or C, or any nonempty subset of set of A and B and C. For instance, in illustrative example of a set having three members, conjunctive phrases "at least one of A, B, and C" and "at least one of A, B and C" refer to any of following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B, and at least one of C each to be present. In addition, unless otherwise noted or contradicted by context, term "plurality" indicates a state of being plural (e.g., "a plurality of items" indicates multiple items). A plurality is at least two items, but can be more when so indicated either explicitly or by context. Further, unless stated otherwise or otherwise clear from context, phrase "based on" means "based at least in part on" and not "based solely on."

Operations of processes described herein can be performed in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. In at least one embodiment, a process such as those processes described herein (or variations and/or combinations thereof) is performed under control of one or more computer systems configured with executable instructions and is implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. In at least one embodiment, code is stored on a computer-readable storage medium, for example, in form of a computer program comprising a plurality of instructions executable by one or more processors. In at least one embodiment, a computer-readable storage medium is a non-transitory computer-readable storage medium that excludes transitory signals (e.g., a propagating transient electric or electromagnetic transmission) but includes non-transitory data storage circuitry (e.g., buffers, cache, and queues) within transceivers of transitory signals. In at least one embodiment, code (e.g., executable code or source code) is stored on a set of one or more non-transitory computer-readable storage media having stored thereon executable instructions (or other memory to store executable instructions) that, when executed (i.e., as a result of being executed) by one or more processors of a computer system, cause computer system to perform operations described herein. A set of non-transitory computer-readable storage media, in at least one embodiment, comprises multiple non-transitory computer-readable storage media and one or more of individual non-transitory storage media of multiple non-transitory computer-readable storage media lack all of code while multiple non-transitory computer-readable storage media collectively store all of code. In at least one embodiment, executable instructions are executed such that different instructions are executed by different processors—for example, a non-transitory computer-readable storage medium store instructions and a main central processing unit ("CPU") executes some of instructions while a graphics processing unit ("GPU") executes other instructions. In at least one embodiment, different components of a computer system have separate processors and different processors execute different subsets of instructions.

Accordingly, in at least one embodiment, computer systems are configured to implement one or more services that singly or collectively perform operations of processes described herein and such computer systems are configured with applicable hardware and/or software that enable performance of operations. Further, a computer system that implements at least one embodiment of present disclosure is a single device and, in another embodiment, is a distributed computer system comprising multiple devices that operate differently such that distributed computer system performs operations described herein and such that a single device does not perform all operations.

Use of any and all examples, or exemplary language (e.g., "such as") provided herein, is intended merely to better illuminate embodiments of disclosure and does not pose a limitation on scope of disclosure unless otherwise claimed. No language in specification should be construed as indicating any non-claimed element as essential to practice of disclosure.

All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

In description and claims, terms "coupled" and "connected," along with their derivatives, may be used. It should be understood that these terms may be not intended as synonyms for each other. Rather, in particular examples, "connected" or "coupled" may be used to indicate that two or more elements are in direct or indirect physical or electrical contact with each other. "Coupled" may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other.

Unless specifically stated otherwise, it may be appreciated that throughout specification terms such as "processing," "computing," "calculating," "determining," or like, refer to action and/or processes of a computer or computing system, or similar electronic computing device, that manipulate and/or transform data represented as physical, such as electronic, quantities within computing system's registers and/or memories into other data similarly represented as physical quantities within computing system's memories, registers or other such information storage, transmission or display devices.

In a similar manner, term "processor" may refer to any device or portion of a device that processes electronic data from registers and/or memory and transform that electronic data into other electronic data that may be stored in registers and/or memory. As non-limiting examples, "processor" may be a CPU or a GPU. A "computing platform" may comprise one or more processors. As used herein, "software" processes may include, for example, software and/or hardware entities that perform work over time, such as tasks, threads, and intelligent agents. Also, each process may refer to multiple processes, for carrying out instructions in sequence or in parallel, continuously or intermittently. Terms "system" and "method" are used herein interchangeably insofar as system may embody one or more methods and methods may be considered a system.

In present document, references may be made to obtaining, acquiring, receiving, or inputting analog or digital data into a subsystem, computer system, or computer-implemented machine. Obtaining, acquiring, receiving, or inputting analog and digital data can be accomplished in a variety of ways such as by receiving data as a parameter of a function call or a call to an application programming interface. In some implementations, process of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a serial or parallel interface. In another implementation, process of obtaining, acquiring, receiving, or inputting analog or digital data can be accomplished by transferring data via a computer network from providing entity to acquiring entity. References

may also be made to providing, outputting, transmitting, sending, or presenting analog or digital data. In various examples, process of providing, outputting, transmitting, sending, or presenting analog or digital data can be accomplished by transferring data as an input or output parameter of a function call, a parameter of an application programming interface or interprocess communication mechanism.

Although discussion above sets forth example implementations of described techniques, other architectures may be used to implement described functionality, and are intended to be within scope of this disclosure. Furthermore, although specific distributions of responsibilities are defined above for purposes of discussion, various functions and responsibilities might be distributed and divided in different ways, depending on circumstances.

Furthermore, although subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that subject matter claimed in appended claims is not necessarily limited to specific features or acts described. Rather, specific features and acts are disclosed as exemplary forms of implementing the claims.

What is claimed is:

1. A computer-implemented method, comprising:

allocating subsets of locations, in content to be rendered, to a plurality of hash map entries, individual locations of the subsets being associated with respective selector values and respective location index values, the location index values for a single instance of the content being different for each of the individual locations;

determining a representative location, within a respective hash map entry, from which to perform, independent of order, simulations for the remaining individual locations within the respective hash map entry, the representative location being one of the individual locations;

storing the selector value associated with the representative location for the respective hash map entry and the location index value associated with the representative location for the respective hash map entry in a combined value; and

providing the combined value to determine the representative location for a computing operation to be performed for the respective hash map entry during the simulations.

2. The computer-implemented method of claim **1**, wherein a lower number of bits in the combined value store the location index value and a higher number of bits store the selector value.

3. The computer-implemented method of claim **1**, further comprising:

determining the plurality of hash map entries by performing spatial hashing on a set of locations for the content to be rendered.

4. The computer-implemented method of claim **1**, wherein the locations correspond to pixels in an image to be rendered, and wherein the operation to be performed involves tracing rays for a respective hash map entry from a pixel location determined from the combined value.

5. The computer-implemented method of claim **1**, wherein the selector value is a maximum or minimum value generated using a hash function, or a pseudo-random or deterministic number generator.

6. The computer-implemented method of claim **1**, further comprising:

performing the simulation for the respective hash map entry from the determined location, wherein the determined location is the same for each of a set of instances of the content regardless of a schedule of execution for the computing operation.

7. The computer-implemented method of claim **1**, wherein data computed for the location is applied to related points for other locations in the respective hash map entry.

8. A system, comprising:

at least one processor; and

memory including instructions that, when executed by the at least one processor, cause the system to:

allocate subsets of locations, of content to be generated, to a plurality of hash map cells, individual locations of the subsets being associated with respective selector values and respective index values, the index values for a single instance of the content being different for each of the individual locations;

determine a representative location, within a respective hash map cell, from which to perform, independent of order, simulations for the remaining individual locations within the respective hash map cell, the representative location being one of the individual locations;

store the selector value associated with the representative location for the respective hash map cell and the index value associated with the representative location for the respective hash map cell in a combined value; and

provide the combined value to determine the representative location for a computing operation for the respective hash map cell during the simulation.

9. The system of claim **8**, wherein a lower number of bits in the combined value store the index value and a higher number of bits store the selector value.

10. The system of claim **8**, wherein the instructions when executed further cause the system to:

determine the plurality of hash map cells by performing spatial hashing on a set of locations for the content to be generated.

11. The system of claim **8**, wherein the locations correspond to pixels in an image to be rendered, and wherein the operation to be performed involves tracing rays for a respective hash map entry from a pixel location determined from the combined value.

12. The system of claim **8**, wherein the selector value is a maximum or minimum value generated using a hash function, or a pseudo-random or deterministic number generator.

13. The system of claim **8**, wherein the instructions when executed further cause the system to:

perform the simulation for the respective hash map cell from the determined location, wherein the determined location is the same for each of a set of instances of the content regardless of a schedule of execution for the computing operation.

14. The system of claim **8**, wherein simulation data determined for the location is applied to related points for other locations in the respective hash map cell.

15. The system of claim **8**, wherein the system comprises at least one of:

a system for performing simulation operations;

a system for performing simulation operations to test or validate autonomous machine applications;

a system for rendering graphical output;

a system for performing deep learning operations;

a system implemented using an edge device;

a system incorporating one or more Virtual Machines (VMs);

a system implemented at least partially in a data center; or

a system implemented at least partially using cloud computing resources.

16. A non-transitory computer-readable storage medium including instructions that, if executed by one or more processors, cause the one or more processors to:

allocate subsets of pixels, of an image to be rendered, to a plurality of hash map cells, individual pixels of the subsets being associated with respective selector values and respective pixel index values, the pixel index values for the image being different for each of the individual pixels;

determine a representative pixel, within a respective hash map cell, from which to perform, independent of order, ray tracing for the remaining individual pixels, within the respective hash map cell, the representative pixel being one of the individual pixels;

store the selector value associated with the representative pixel for the respective hash map cell and the pixel index value associated with the representative pixel for the respective hash map cell in a combined value; and

provide the combined value to determine the representative pixel for a thread of execution for the respective hash map cell during ray tracing, wherein rays for the respective hash map cell are to be traced from the representative pixel.

17. The non-transitory computer-readable storage medium of claim **16**, wherein the instructions, if executed, further cause the one or more processors to:

use a lower number of bits in the combined value to store the pixel index value and a higher number of bits to store the selector value.

18. The non-transitory computer-readable storage medium of claim **17**, wherein the instructions, if executed, further cause the one or more processors to:

determine the plurality of hash map cells by performing spatial hashing on a set of pixels for the image to be rendered.

19. The non-transitory computer-readable storage medium of claim **17**, wherein the instructions, if executed, further cause the one or more processors to:

determine a number of pixels to be included in individual cells of the plurality of hash map cells.

20. The non-transitory computer-readable storage medium of claim **17**, wherein the selector value is a maximum or minimum value generated using a hash function or pseudo-random number generator.

* * * * *