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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 June 2011 (03.06.2011)

PCT

(10) International Publication Number
WO 2011/065929 A1

(51) International Patent Classification:
G06T 1/00 (2006.01)

(21) International Application Number:
PCT/US2008/084282

(22) International Filing Date:
21 November 2008 (21.11.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/989,881 23 November 2007 (23.11.2007) US

(71) Applicant *(for all designated States except US)*: **MERCURY COMPUTER SYSTEMS, INC.** [US/US]; 199 Riverneck Road, Chelmsford, MA 01824 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **WESTERHOFF, Malte** [DE/DE]; Leo-Baeck-str. 70, D-14165 Berlin (DE). **STALLING, Detlev** [DE/DE]; Garystrabe 20, D-14195 Berlin (DE).

(74) Agents: **POWSNER, David, J.** et al.; Nutter Mcclennen & Fish Llp, World Trade Center West, 155 Seaport Boulevard, Boston, MA 02210-2604 (US).

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

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

Published:

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

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) **Title**: MULTI-USER MULTI-GPU RENDER SERVER APPARATUS AND METHODS

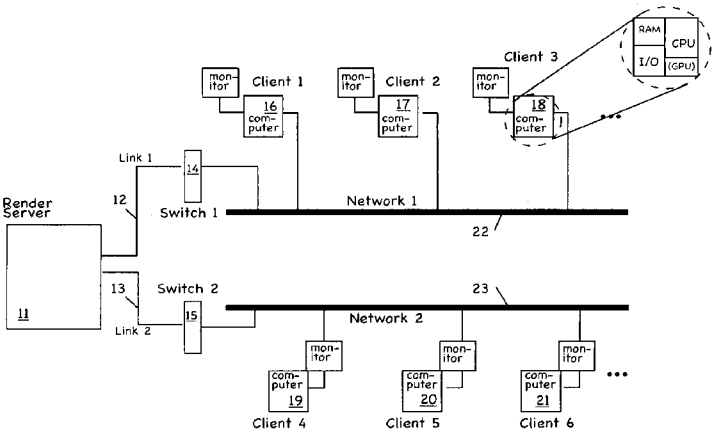

Figure 1

(57) **Abstract**: The invention provides a system for rendering images, having one or more client digital data processors and a server digital data processor in communications coupling with the one or more client digital data processors, the server digital data processor having one or more graphics processing units. The system additionally comprises a render server module executing on the server digital data processor and coupling with the graphics processing units, where the render server module issues a command in response to a request from a first client digital data processor. The graphics processing units on the server digital data processor simultaneously process image data in response to interleaved commands from (i) the render server module on behalf of the first client digital data processor, and (ii) one or more requests from the render server module on behalf of any of the other client digital data processors.

WO 2011/065929 A1

MULTI-USER MULTI-GPU RENDER SERVER APPARATUS AND METHODS

Background of the Invention

This application claims the benefit of priority of United States Patent Application Serial No. 60/989,881, filed November 23, 2007, the teachings of which are incorporated herein by reference.

The invention pertains to digital data processing and, more particularly, by way of example, to the visualization of image data. It has application to areas including medical imaging, atmospheric studies, astrophysics, and geophysics.

3D and 4D image data is routinely acquired with computer tomographic scanners (CT), magnetic resonance imaging scanners (MRI), confocal microscopes, 3D ultrasound devices, positron emission tomographics (PET) and other imaging devices. The medical imaging market is just one example of a market that uses these devices. It is growing rapidly, with new CT scanners collecting ever greater amounts of data even more quickly than previous generation scanners. As this trend continues across many markets, the demand for better and faster visualization methods that allow users to interact with the image data in real-time will increase.

Standard visualization methods fall within the scope of volume rendering techniques (VRT), shaded volume rendering techniques (sVRT), maximum intensity projection (MIP), oblique slicing or multi-planar reformats (MPR), axial/sagittal and coronal slice display, and thick slices (also called slabs). In the following, these and other related techniques are collectively referred to as "volume rendering." In medical imaging, for example, volume rendering is used to display 3D images from 3D image data sets, where a typical 3D image data set is a large number of 2D slice images acquired by a CT or MRI scanner and stored in a data structure.

The rendition of such images can be quite compute intensive and therefore takes a long time on a standard computer, especially, when the data sets are large. Too long compute times can, for example, prevent the interactive exploration of data sets, where a user wants to change viewing parameters, such as the viewing position interactively, which requires several screen updates per second (typically 5 – 25 updates/second), thus requiring rendering times of fractions of a second or less per image.

Several approaches have been taken to tackle this performance problem. Special-purchase chips have been constructed to implement volume rendering in hardware. Another approach is to employ texture hardware built into high-end graphics workstations or graphics super-computers, such as for example Silicon Graphics Onyx computers with Infinite Reality and graphics. More recently, standard graphics boards, such as NVIDIA's Geforce and Quadro FX series, as well as AMD/ATI's respective products, are also offering the same or greater capabilities as far as programmability and texture memory access are concerned.

Typically hardware for accelerated volume rendering must be installed in the computer (e.g., workstation) that is used for data analysis. While this has the advantage of permitting ready visualization of data sets that are under analysis, it has several drawbacks. First of all, every computer which is to be used for data analysis needs to be equipped with appropriate volume-rendering hardware, as well as enough main memory to handle large data sets. Second the data sets often need to be transferred from a central store (e.g.,, a main enterprise server), where they are normally stored, to those local workstations prior to analysis and visualization, thus potentially causing long wait times for the user during transfer.

Several solutions have been proposed in which data processing applications running on a server are controlled from a client computer, thus, avoiding the need to equip it with the full hardware needed for image processing/visualization and also making data transfer to the client unnecessary. Such solutions include Microsoft's Windows 2003 server (with the corresponding remote desktop protocol (RDP)), Citrix Presentation Server, VNC, or SGI's OpenGL Vizserver. However, most of these solutions do not allow applications to use graphics hardware acceleration. The SGI OpenGL Vizserver did allow hardware accelerated

graphics applications to be run over the network: it allocated an InfiniteReality pipeline to an application controlled over the network. However that pipeline could then not be used locally any longer and was also blocked for other users. Thus effectively all that the Vizserver was doing was extending a single workplace to a different location in the network. The same is true for VNC.

For general graphics applications (i.e., not specifically volume rendering applications), such as computer games, solutions have been proposed to combine two graphics cards on a single computer (i.e., the user's computer) in order to increase the rendering performance, specifically NVIDIA's SLI and AMD/ATI's Crossfire products. In these products, both graphics cards receive the exact same stream of commands and duplicate all resources (such as textures). Each of the cards then renders a different portion of the screen — or in another mode one of the cards renders every second image and the other card renders every other image. While such a solution is transparent to the application and therefore convenient for the application developers it is very limited, too. Specifically the duplication of all textures effectively eliminates half of the available physical texture memory.

An object of the invention is to provide digital data processing methods and apparatus, and more particularly, by way of example, to provide improved such methods and apparatus for visualization of image data.

A further object of the invention is to provide methods and apparatus for rendering images.

A still further object of the invention is to provide such methods and apparatus for rendering images as have improved real-time response to a user's interaction.

Yet a still further object of the invention is to provide such methods and apparatus as allow users to interactively explore the rendered images.

Summary of the Invention

The aforementioned are among the objects attained by the invention, which provides, in one aspect, a graphics system including a render server that has one or more graphics boards in one or more host systems. One or more client computers can simultaneously connect to the render server, which receives messages from the client computers, creates rendered images of data set and sends those rendered images to the client computers for display.

Related aspects of the invention provide a graphics system, for example, as described above in which rendered data sets are kept in memory attached to the render server, such as RAM memory installed in the host systems, e.g., for reuse in response to subsequent messaging by the client computers.

Further related aspects of the invention provide a graphics system, for example, as described above in which the render server maintains a queue of so-called render requests, i.e., a list of images to render. These can comprise render requests received directly in messages from the client computers and/or they can comprise requests generated as a result of such messages. One message received from the client computer can result in zero, one, or multiple render requests being generated.

A further aspect of the invention provides a graphics system, for example, of the type described above, in which the render server breaks down selected ones of the render requests into multiple smaller requests, i.e., requests which require less compute time and/or less graphics resources. A related aspect of the invention provides for scheduling the smaller (and other) requests so as to minimize an average time that a client computer waits for a response to a request. This allows (by way of non-limiting example) for concurrent treatment of requests and for serving multiple client computers with a single GPU without compromising interactivity.

Another aspect of the invention provides a graphics system, for example, of the type described above, that processes render requests in an order determined by a prioritization

function that takes into account the nature of the request (e.g., interactive rendering vs. non-interactive), the client from which the request was received, the order in which the requests were received, the resources currently allocated on the graphics boards, and/or other parameters.

Yet another aspect of the invention provides a graphics system, for example, of the type described above that processes multiple render requests simultaneously. The render server of such a system can, for example, issue multiple render commands to a single graphics board and process them in time slices (in a manner analogous to a multi-tasking operating system on a CPU), thereby switching between processing different render requests multiple times before a single render request is completed.

A related aspect of the invention provides a system, for example, as described above wherein the render server combines render requests for simultaneous processing in such a way, that their total graphics resource requirements can be satisfied by resources (e.g., texture and frame buffer memory) on-board a single graphics board. This allows (by way of example) time-slicing between the simultaneously processed render requests without the computationally expensive swapping of graphics memory chunks in and out of main memory of the host (i.e., "host memory").

Another aspect of the invention provides a graphics system, for example, of the type described above, that renders images at different resolution levels, e.g., rendering a low-resolution image from a low-resolution version of the input data while rotating the data set, thus enabling faster rendering times and thereby smoother interaction. A related aspect of the invention provides such a system that adapts the resolution to the network speed and or the available processing resources. Another related aspect of the invention provides such a system wherein the render server continuously monitors one or more of these parameters and thereby allows for continuous adaptation of the resolution.

Another aspect of the invention provides a graphics system, for example, of the type described above, wherein the render server keeps local resources (such as texture memory) on one of the graphics boards allocated for the processing of a particular set of related render

requests. Related aspects of the invention provide (for example) for re-use of such allocated resources for the processing of a subsequent render request in the set, thus eliminating the need to re-upload the data from host memory to texture memory for such subsequent render requests. By way of example, the render server of such a system can keep the texture memory of a graphics board allocated to the rendition of interactive render requests for low resolution versions of a data set (e.g., user-driven requests for rotation of the data set), which need to be processed with a minimal latency to allow for smooth interaction but only require a small amount of texture memory.

Another aspect of the invention provides a graphics system, for example, of the type described above, wherein the render server dispatches render commands to different graphics boards. A related aspect provides such a system that takes into account the data sets resident on these different graphics boards and uses this information to optimize such dispatching.

Further aspects of the invention provide systems employing combinations of the features described above.

Further aspects of the invention provide methods for processing images that parallel the features described above.

These and other aspects of the invention are evident in the drawings and in the description that follows.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a client-server system according to one practice of the invention;

Figure 2 depicts the host system of the render server of the type used in a system of the type shown in Figure 1;

Figure 3 depicts a timeline of incoming render requests from client computers in a system of the type shown in Figure 1;

Figures 4 – 6 depict timelines for processing requests of the type shown in Figure 3;

Figure 7 depicts a 3D data set of the type suitable for processing in a system according to the invention;

Figure 8 depicts sub-volumes making up the data set of Figure 7;

Figures 9 – 12 depict images resulting from MIP renderings of an image utilizing sub-volumes of the type shown in Figure 8;

Figures 13 is a flowchart illustrating a method of operation of the system of the type shown in Figure 1;

Figure 14 is a flowchart illustrating a method of utilizing bricking to perform rendering in a system of the type shown in Figure 1;

Figure 15 is a flowchart illustrating a method of multi-resolution rendering in a system of the the type shown in Figure 1; and

Figures 16a – 16b are flowcharts illustrating data upload from host memory to graphics memory in a host system of the type shown in Figure 2; and

Figure 17 are flow charts illustrating a method of breaking down render requests into smaller requests in connection with concurrent rendering.

Detailed Description of the Illustrated Embodiment

Overview

Figure 1 depicts a system 10 according to one practice of the invention. A render server (or server digital data processor) 11, which is described in more detail below, is connected via one or more network interfaces 12, 13 and network devices such as switches or hubs 14, 15 to one or more networks 22, 23. The networks 22, 23 can be implemented utilizing Ethernet, WIFI, DSL and/or any other protocol technologies and they can be part of the Internet and/or form WANs (wide area networks), LANs (local area networks), or other types of networks known in the art.

One or more client computers (or "client digital data processors") 16 – 21 are coupled to render server 11 for communications via the networks 22, 23. Client software running on each of the client computers 16 – 21 allows the respective computers 16 – 21 to establish a network connection to render server 11 on which server software is running. As the user interacts with the client software, messages are sent from the client computers 16 – 21 to the render server 11. Render server 11, generates render commands in response to the messages, further processing the render requests to generate images or partial images, which are then sent back to the respective client computer s 16 – 21 for further processing and/or display.

The make-up of a typical such client computer is shown, by way of example, in the break-out on Figure 1. As illustrated, client computer 18 includes CPU 18a, dynamic memory (RAM) 18b, input/output section 18c and optional graphics processing unit 18d, all configured and operated in the conventional manner known in the art — as adapted in accord with the teachings hereof.

The components illustrated in Figure 1 comprise conventional components of the type known in the art, as adapted in accord with the teachings hereof. Thus, by way of non-limiting example, illustrated render server 11 and client computers 16 – 21 comprise conventional workstations, personal computers and other digital data processing apparatus of the type available in the market place, as adapted in accord with the teachings hereof.

It will be appreciated that the system 10 of Figure 1 illustrates just one configuration of digital data processing devices with which the invention may be practiced. Other embodiments may, for example, utilize greater or fewer numbers of client computers, networks, networking apparatus (e.g., switches or hubs) and so forth. Moreover, it will be appreciated that the invention may be practiced with additional server digital data processors. Still further, it will be appreciated that the server digital data processor 11 may, itself, function — at least in part — in the role of a client computer (e.g., generating and servicing its own requests and/or generating requests for servicing by other computers) and vice versa.

Render server

In the following section we describe the render server in more detail and how it is used to perform volume rendering.

Figure 2 depicts render server 11, which includes one or more host systems 30, each equipped with one or more local graphics (GPU) boards 33, 34. As those skilled in the art will appreciate, a host system has other components as well, such as a chipset, I/O components, etc., which are not depicted in the figure. The host system contains one or more central processing units (CPU) 31, 32, for example AMD Opteron or Intel Xeon CPUs. Each CPU 31, 32 can have multiple CPU cores. Connected to CPUs 31, 32 is a host memory 41.

GPU Boards 33, 34. can be connected to other system components (and, namely, for example, to CPUs 31, 32) using the PCI-Express bus, but other bus systems such as PCI or AGP can be used as well, by way of non-limiting example. In this regard, standard host mainboards exist, which provide multiple PCI-Express slots, so that multiple graphics cards can be installed. If the host system does not have sufficient slots, a daughter card can be used (e.g., of a type such as that disclosed in co-pending commonly assigned United States Patent Application Serial No. 11/129,123, entitled "Daughter Card Approach to Employing Multiple Graphics Cards Within a System," the teachings of which are incorporated herein by reference). Alternatively, or in addition, such cards can be provided via external cable-connected cages.

Each graphics board 33, 34 has amongst other components local, on-board memory 36, 38, coupled as shown (referred to elsewhere herein as "graphics memory," "Graphics Memory," "texture memory," and the like) and a graphics processing unit (GPU) 35, 37. In order to perform volume rendering of a data set, the data set (or the portion to be processed) preferably resides in graphics memories 36, 38.

The texture (or graphics) memory 36, 38 is normally more limited than host memory 41 and often smaller than the total amount of data to be rendered, specifically for example, as in the case of the illustrated embodiment, if server 11 is used by multiple users concurrently visualizing different data sets. Therefore not all data needed for rendering can, at least in the illustrated embodiment, be kept on graphics boards 33, 34.

Instead, in the illustrated embodiment, in order to render an image, the respective portion of the data set is transferred from either an external storage device or, more typically, host memory 41 into the graphics memories 36, 38 via the system bus 42. Once the data is transferred, commands issued to GPUs 35, 37 by Render Server Software (described below) cause it to render an image with the respective rendering parameters. The resulting image is generated in graphics memories 36, 38 on graphics boards 33, 34 and once finished can be downloaded from graphics boards 33, 34, i.e., transferred into host memory 41, and then after optional post-processing and compression be transferred via network interfaces 39,40 to client computer s 16 – 21.

The components of host 30 may be interconnected by a system bus 42 as shown. Those skilled in the art will appreciate that other connections and interconnections may be provided as well or in addition.

Render Server Software and Client Software

The process described above, as well as aspects described subsequently, is controlled by software, more specifically software running on Render Server 11 ("Render Server Software") and software running on client computers 16 – 21 ("Client Software"). The Render Server Software handles network communication, data management, actual

rendering, and other data processing tasks such as filtering by way of employing CPUs 31, 32, GPUs 35, 37, or a combination thereof. The Client Software is responsible for allowing the user to interact, for example, to choose a data set to visualize, to choose render parameters such as color, data window, or the view point or camera position when e.g., rotating the data set. The client software also handles network communication with server 11 and client side display.

In the following we describe one way how the Render Server Software and Client software can be implemented. In this regard, see, for example, Figure 13, steps 1301 – 1310. A component of the Render Server software listens for incoming network connections. Once a Client computers attempts to connect, the Render Server Software may accept or reject that connection potentially after exchanging authentication credentials such as a username and password and checking whether there are enough resources available on the render server. The Render Server software listens on all established connections for incoming messages. This can be implemented for example by a loop sequentially checking each connection or by multiple threads, one for each connection, possibly being executed simultaneously on different CPUs or different CPU cores. Once a message is received, it is either processed immediately or added to a queue for later processing. Depending on the message type a response may be sent. Examples for message types are: (i) Request for a list of data sets available on the server – potentially along with filter criteria, (ii) Request to load a data set for subsequent rendering, (iii) Request to render a data set with specified rendering parameters and a specified resolution level, (iv) Message to terminate a given connection, (v) message to apply a filter (for example noise removal or sharpening) etc.

Figure 13, steps 1311–1315, illustrate the typical case in which the client computer sends a render request and the Render Server Software handles the render request using GPU 35, 37. The Render Server Software transfers the data set in question (or, as is discussed below, portions of it) into local graphics memories 36, 38 via the system bus 42, issues commands to GPUs 35, 37 to create a rendered image in graphics memories 36, 38 and transfers the rendered image back into host memory 41 for subsequent processing and network transfer back to the requesting client computer.

In the illustrated embodiment, a component (e.g., software module) within the Render Server Software prioritizes the requests added to the queue of pending requests thereby determining the order in which they are executed. Other such components of the illustrated embodiment alter requests in the queue, i.e., remove requests which are obsoleted or break down requests into multiple smaller ones (*see,* step 1311b). In these and other embodiments, still another such component of the Render Server Software determines which resources are used to process a request. Other embodiments may lack one or more of these components and/or may include additional components directed toward image rendering and related functions.

In the following, details of these components as well as other aspects are described.

Bricking

When the Render Server Software handles a render request by way of using the GPU, it transfers the data set in question (or, as is discussed below, portions of it) into the local Graphics Memory via the system bus, then issues the commands necessary to create a rendered image, and then transfers back the rendered image into main memory for subsequent processing and network transfer. Even a single data set can exceed the size of the graphics memory. In order to render such a data set efficiently, it is broken down into smaller pieces which can be rendered independently. We refer to this process as bricking. As discussed later, the ability to break down one render request into multiple smaller requests, where smaller can mean that less graphics memory and/or less GPU processing time is required, is also helpful for efficiently handling multiple requests concurrently.

We now describe how such a break down can be performed. As an example, we first discuss the MIP rendering mode, though, it will be appreciated that such a methodology can be used with other rendering modes. The 3D data set can be viewed as a cuboid in three-space, consisting of a number of voxels carrying gray values. Figure 7 depicts that data volume viewed from a certain camera position by way of displaying a bounding box. Referring to Figure 14 (which illustrates a method for bricking according to one practice of the invention), for a given camera position, each pixel on a computer screen (screen pixel)

can be associated with a viewing ray. *See,* step 1402a. The voxels intersected by each such viewing ray which intersects the cuboid are then determined. *See,* step 1402b. In the MIP rendering mode, the screen pixel is assigned the maximum gray value of any of the voxels, which the viewing ray corresponding to the screen pixel intersects. *See,* step 1402c. The resulting rendered image can be seen in Figure 9.

If the Render Server Software subdivides the original data volume into multiple smaller data volumes — for example if it divides the data volume into four sub volumes — then each of the sub volumes can be rendered independently, thus, effectively producing four rendered images. *See,* Figure 14, steps 1401 and 1402. The subdivision for this example is illustrated in Figure 8 by way of showing the bounding boxes of the four sub-volumes. Figure 10 shows the individual MIP rendition of each of the four sub volumes for an example data set depicting an Magnet Resonance Angiography image. For better orientation, the bounding box of the original data volume is shown as well. If the rendered images are then composed in such a way that for each pixel in the composed image the brightest value for that pixel from the four rendered images is chosen (*see,* Figure 14, step 1403), then the resulting composed image, which is shown in Figure 11, is identical to the MIP rendition of the full data set, seen in Figure 8.

Using the correct composition function, the same break-down approach can be used for other rendering modes as well. For example, for VRT mode, standard alpha-blending composition can be used, i.e., for each pixel of the resulting image the color an opacity is computed as follows. The sub images are blended over each other in back to front order, one after the other using the formula c_result = (1 – a_front) * c_back + a_front * c_front, where, a_front and c_front denote the opacity and color of the front picture respectively, and c_back denotes the color of the back picture. As those skilled in the art will appreciate, other schemes such as front to back or pre-multiplied alpha may be used with the respective formulas found in general computer graphics literature. The resulting image for VRT rendering is shown in Figure 12.

Multi-Resolution Rendering

The time it takes to render an image depends on several criteria, such as the rendering mode, the resolution (i.e., number of pixels) of the rendered (target) image and the size of the input data set. For large data sets and high-resolution renditions, rendering can take up to several seconds, even on a fast GPU. However, when a user wants to interactively manipulate the data set, i.e., rotate it on the screen, multiple screen updates per second (typically 5 – 25 updates/second) are required to permit a smooth interaction. This means that the rendition of a single image must not take longer than few hundred milliseconds, ideally less than 100 milliseconds.

One way to ensure smooth rendering during users' interactive manipulations of data sets is by rendering images at a resolution according to the level of a user's interaction. One way to guarantee this is illustrated in Figure 15. Here, by way of example, the system checks whether the user is rotating the data set (*see,* Step 1502). If so, the render server uses a lower resolution version of the input data and renders the images at a lower target resolution. *See,* steps 1503b and 1504b. Once the user stops interacting, e.g., by releasing the mouse button, a full resolution image is rendered with the full-resolution data set and the screen is updated with that image, potentially a few seconds later. *See,* steps 1503a and 1504a. Schemes with more than two resolutions can be used in the same way.

In the subsequent discussion we refer to the above scenario to illustrate certain aspects of the invention. We refer to the low-resolution renderings as "interactive render requests" and to the larger full resolution renditions as "high-resolution render requests". The methodologies described below are not restricted to an interaction scheme which uses two resolutions in the way described above.

Scheduling Strategies

In order to build an effective multi-user multi-GPU render server, another component of the Render Server Software is provided which dispatches, schedules and processes the render requests in a way that maximizes rendering efficiency. For example, the number of

client computers which can access the render server concurrently may not be limited to the number of GPUs. That is, two or more clients might share one GPU. Render requests received by such clients therefore need to be scheduled. This section describes some factors that may be considered for the scheduling and illustrates why a trivial scheduling may not be sufficient in all cases.

Figure 3 illustrates, by way of non-limiting example, render requests coming in from three different client computers. The render requests A1, A2, ..., A5 shall come in from a client computer A, while the render requests B1 ... B5 come in from client computer B and the render request C1 comes from client computer C. The different sizes of the render requests in Figure 3 symbolize the different size in the sense that larger boxes (such as C1) require more processing time and require more graphics memory than smaller ones (such as for example A1). The horizontal axis symbolizes the time axis, depicting when the render requests have been received, i.e., render request A1 has been received first, then C1, then B1, then A2, then B2, and so forth.

In one example, the "smaller" render requests A1...A5 and B1...B5 are interactive render requests, e.g., requests received while the user is rotating the data set, while C1 may be a high-resolution render request. By way of example, the interactive render requests might require 50 ms to process, while the high-resolution render request might take 2 seconds to render. If only one GPU was available to handle these render requests, and if the render requests were scheduled in a trivial way, on a first come-first serve basis, the result would not yield a good user experience. Figure 4 illustrates such a case where request A1 is processed first, followed by C1, B1, A2, ... While render request C1 is processed, which in this example is assumed to take 5 seconds, no render requests for client A and client B would be processed. However this example assumes that the users using client A and client B are at this given time interactively manipulating, e.g., rotating, the data sets. Therefore if those clients would not receive a screen update for 2 seconds, the interaction would stall, prohibiting a smooth and interactive user experience.

An alternative strategy of not processing any high-resolution render requests as long as any interactive render requests are still pending also would not be optimal. If, in the above

example, the users using clients A or B rotated their data sets for a longer period of time, e.g., half a minute or longer, then during that time they would constantly generate render requests, effectively prohibiting the request from client C to be processed at all (until both other users have completed their interaction). This is also not desired.

Methods of improved scheduling to reduce average wait time for a response to a client computer's render request are needed. We are now going to describe two alternative strategies for a better scheduling and will later describe how a combination of both leads to even better results.

The first strategy, illustrated in Figures 5 and 6, involves the situation where "large" render requests are broken down into multiple smaller render requests which are processed individually. For example, here, request C1 is broken down into multiple smaller requests. Once this is done, those smaller requests can be scheduled more flexibly, for example as shown in Figure 6. Such a scheduling has the advantage that none of the clients would see any significant stalling — only a somewhat reduced rate of screen updates per second. Still however also the high-resolution render request would not be postponed indefinitely but be processed in a timely manner.

Concurrent Rendering

The second strategy is to issue multiple render commands to the same graphics board simultaneously, i.e., issue a first command (e.g., in response to a request received from a first client computer) and then issue a second command (e.g., in response to a request received from a second client computer) before the first request is completed. Preferably, this is done so as to interleave commands that correspond to different respective client requests so that the requests are processed in smaller time slices in an alternating fashion.

This can be done in multiple ways. One way is to use multiple processes or multiple threads, each rendering using the same graphics board. In this case the operating system and graphics driver respectively handle the "simultaneous" execution of the requests. In fact, of course, the execution is not really simultaneous but broken down into small time slices in

which the requests are processed in an alternating fashion. The same can be achieved by a single thread or process issuing the primitive graphics commands forming the render requests in an alternating fashion, thereby assuring that texture bindings and render target assignments are also switched accordingly.

The reason why it may be advantageous to issue multiple render commands simultaneously in contrast to a fully sequential processing as depicted, e.g., in Figure 6, is two-fold. First, it can be the case that, even after breaking down larger render requests into smaller ones, each request may still take more processing time than one would like to accept for stalling other, smaller, interactive requests. Second, a graphics board is a complex sub-system with many different processing and data transfer units, some of which can work in parallel. Therefore, certain aspects of two or more render requests being processed simultaneously can be executed truly simultaneously, e.g.,, while one render request consumes the compute resources on the GPU, the other consumes data transfer resources. Thus, executing the two requests simultaneously may be faster than executing them sequentially. Additionally, although the GPU simultaneously processes render commands issued by the render server CPU on behalf of multiple remote client computers, the GPU may also simultaneously process render requests (or other requests) issued by or on behalf of other functionality (e.g., requests issued by the render server CPU on behalf of a local user operating the server computer directly).

Another aspect taken into account by the Render Server Software when issuing render requests simultaneously is the total graphics resource consumption. If the sum of required graphics memory for all simultaneously processed render requests would exceed the total graphics resources on the graphics board, then a significant performance decrease would be the consequence. The reason is, that whenever the operating system or graphics driver switched from execution of request 1 to request 2, then first the data required for the processing of request 1 would have to be swapped out from graphics memory to host memory to make room for the data needed for request 2. Then the data needed for the processing of request 2 would have to be swapped in from host memory into graphics memory. This would be very time consuming and inefficient.

Figure 17 illustrates how the method described above of breaking down render requests into smaller requests can be used with concurrent rendering. Specifically, when scheduling requests, the Render Server Software insures that requests are broken down sufficiently so that the total resource requirements for all simultaneously processed requests do fit into the totally available graphics memory of the graphics board processing these requests. *See,* steps 1702 and 173b.

Persistent Data

The Render Server Software additionally implements schemes to take advantage of data persistency, during scheduling and/or dispatching of requests. Very often subsequent render requests use some of the same data. For example if a user rotates a data set, then many different images will be generated all depicting the same input data set only rendered from different viewing angles. Therefore, if one request has been processed, it can be of advantage to not purge the input data from the graphics memory, but instead keep it persistent in anticipation of a future render request potentially requiring the same data. As illustrated in Figure 16a, in this way a repeated data upload from host memory into graphics memory can be avoided. *See,* step 1606.

In single-GPU systems, a scheduler component of the Render Server Software may take data persistency into account and re-arrange the order of requests in such a way as to optimize the benefit drawn from persistency. In the case of Figure 16a, for example, the scheduler might rearrange the order of the requests so that render request 3 is processed immediately subsequent to render request 1.

In a multi-GPU system, on the other hand, the dispatcher component of the Render Server Software takes persistency into account when deciding which GPU to use to satisfy a specific render request. For example, as mentioned above and depicted in Figure 16b, render requests in multi-GPU systems are typically dispatched to all of the GPUs following the same basic scheme as described above. *See,* step 1652. To take advantage of data persistency, the dispatcher component attempts to dispatch the current request to a graphics processing unit in which the data set specified by the request is stored. *See,* steps 1653 and 1656. This will

often lead to subsequent interactive render requests from the same client computer being handled by the same GPUs.

But, not all render requests need to be executed on the GPUs. Depending on resource use and the type of request, it may also be feasible to use one or more CPU cores on one or more CPUs to process a render request, or a combination of CPU and GPU. For example, rendering requests for MPR mode and oblique slicing can be executed on the CPU unless the data required is already on the GPU. *See,* steps 1654 and 1655b.

Rendering requests are only one example. As those skilled in the art will appreciate, the described embodiment can also be used in the same way to perform other data processing tasks, such as filtering, feature detection, segmentation, image registration and other tasks.

Described above are methods and systems meeting the desired objects, among others. It will be appreciated that the embodiments shown and described herein are merely examples of the invention and that other embodiments, incorporating changes therein may fall within the scope of the invention, of which we claim:

1. A system for rendering images comprising:

A. one or more client digital data processors,

B. a server digital data processor in communications coupling with the one or more client digital data processors, the server digital data processor comprising one or more graphics processing units,

C. a render server, executing on the server digital data processor and in communications coupling with the graphics processing units, the render server responding to a render request from a said client digital data processor by issuing one or more render commands to the one or graphics processing units,

D. the render server responding to render requests from a plurality of said client digital data processors by issuing interleaved render commands to the one or more graphics processing units so that commands corresponding to different respective render are processed by the one or more graphics processing units in an alternating fashion.

2. The system of claim 1 wherein the server digital data processor further comprises one or more central processing units, in communications coupling with the render server, the one or more central processing units processing image data in response to plural interleaved commands from the render server.

3. The system of claim 1, wherein the server digital data processor comprises a host memory, in communications coupling with the render server, the host memory storing one or more data sets to be rendered.

4. The system of claim 1, wherein

the server digital data processor comprises one or more queues in communications coupling with the render server and with the one or more graphics processing units, and

the render server maintaining render requests in the one or more queues.

5. The system of claim 4, wherein the render server prioritizes render requests in the one or more queues.

6. The system of claim 5, wherein the render server prioritizes a said render request based on at least one of a rendering mode associated with that render request, a client digital data processor associated with that render request, an order of receipt of that render request, and available resources.

7. The system of claim 4, wherein the render server breaks down a said render request in a said queue into plural smaller render requests.

8. The system of claim 1, wherein the render server breaks down requests received from the one or more client digital data processors into multiple smaller render requests, each requiring less compute time and/or less graphics resources than the render request from which it was broken down.

9. The system of claim 1, wherein the render server schedules one or more of the smaller requests to minimize an average wait time.

10. The system of claim 1, wherein a said graphics processing unit renders an image at a rendering resolution determined by one or more parameters, including, at least one of a user interaction type, a network speed, and available processing resources.

11. The system of claim 10, wherein the render server

monitors at least one of user interaction type, network speed, and available processing resources, and

generates said one or more parameters in response thereto.

12. The system of claim 1, wherein the render server allocates at least a portion of one or more server digital data processor resources in response to one of the render requests.

13. The system of claim 12, wherein the one or more server digital data processor resources comprise a graphics memory that is coupled to any of said one or more graphics processing units.

14. The system of claim 13, wherein the render server allocates, as a said digital data processor resource, a said graphics memory having a data set specified by a said request.

15. The system of claim 14, wherein the render server causes a said graphics memory to maintain a said data set.

16. The system of claim 1, wherein a said graphics processing unit concurrently renders images in response to the one or more interleaved commands, each of the commands associated with a different request, by using one of multi-processing or multi-threading.

17. A system for rendering images comprising:

A. one or more client digital data processors,

B. a server digital data processor in communications coupling with the one or more client digital data processors, the server digital data processor comprising one or more graphics processing units,

C. a render server, executing on the server digital data processor and in communications coupling with the graphics processing units, the render server responding to a render request from a said client digital data processor by issuing one or more render commands to the one or graphics processing units,

D. the render server responding to render requests from a plurality of said client digital data processors by issuing interleaved render commands to the one or more graphics processing units so that commands corresponding to different respective render are processed by the one or more graphics processing units in an alternating fashion, and

E. the render server breaking down render requests received from the one or more client digital data processors into multiple smaller render requests, each requiring less compute time and/or less graphics resources than the render request from which it was broken down.

18. The system of claim 17, wherein

the render server breaks down render requests so that the amount of memory required for concurrent rendering of the smaller render requests generated as a result thereof is less than or equal to the amount of memory available on the graphics processing unit.

19. A method for rendering images comprising:

A. executing, on a server digital data processor, a render server,

B. issuing one or more interleaved commands with the render server in response to one or more render requests from one or more client digital data processors,

C. rendering images with one or more graphics processing units in response to the interleaved commands from the render server on behalf of the one or more client digital data processors.

20. The method of claim 19, comprising storing one or more data sets in a host memory associated with the server digital data processor.

21. The method of claim 19, comprising maintaining requests received from one or more said client digital data processors in one or more queues associated with the server digital data processor, such maintaining including any of prioritizing the requests,

removing requests, and/or breaking down one or more requests into two or more smaller requests.

22. The method of claim 21, wherein the prioritizing step includes any of prioritizing a said render request based on at least one of a rendering mode associated therewith, a client associated therewith, an order of receipt thereof, and available resources.

23. The method of claim 21, comprising breaking down render requests received from one or more client digital data processors into multiple smaller render requests, each requiring less compute time and/or fewer graphics resources than the request from which it was broken down.

24. The method of claim 19, comprising scheduling one or more of the smaller requests to minimize an average wait time.

25. The method of claim 23, wherein the rendering step comprises rendering images, with the one or more graphics processing units, in response to interleaved commands that are based on the multiple smaller render requests.

26. The method of claim 25, comprising processing, with the one or more graphics processing units, multiple interleaved commands, each based on smaller requests broken down from a said render request received from the one or more client digital data processors, before completing rendering of an image associated with any such received request.

27. The method of claim 19, comprising rendering with a said graphics processing unit an image at a rendering resolution determined by one or more parameters, including, at least one of a user interaction type, a network speed, and available processing resources.

28. The method of claim 19, comprising allocating at least a portion of one or more server digital data processor resources in response to one or more requests received from a said client digital data processor.

29. The method of claim 28, comprising allocating, as a said server digital data processor resource, a graphics memory that is coupled to any of said one or more graphics processing units.

30. A system for rendering images comprising:

A. one or more client digital data processors,

B. a server digital data processor in communications coupling with the one or more client digital data processors, the server digital data processor comprising one or more graphics processing units,

C. a render server, executing on the server digital data processor and in communications coupling with the graphics processing units, the render server issuing one or more interleaved commands in response to one or more render requests from one or more client digital data processors,

D. one or more graphics processing units rendering images in response to the one or more interleaved commands from the render server on behalf of the one or more client digital data processors.

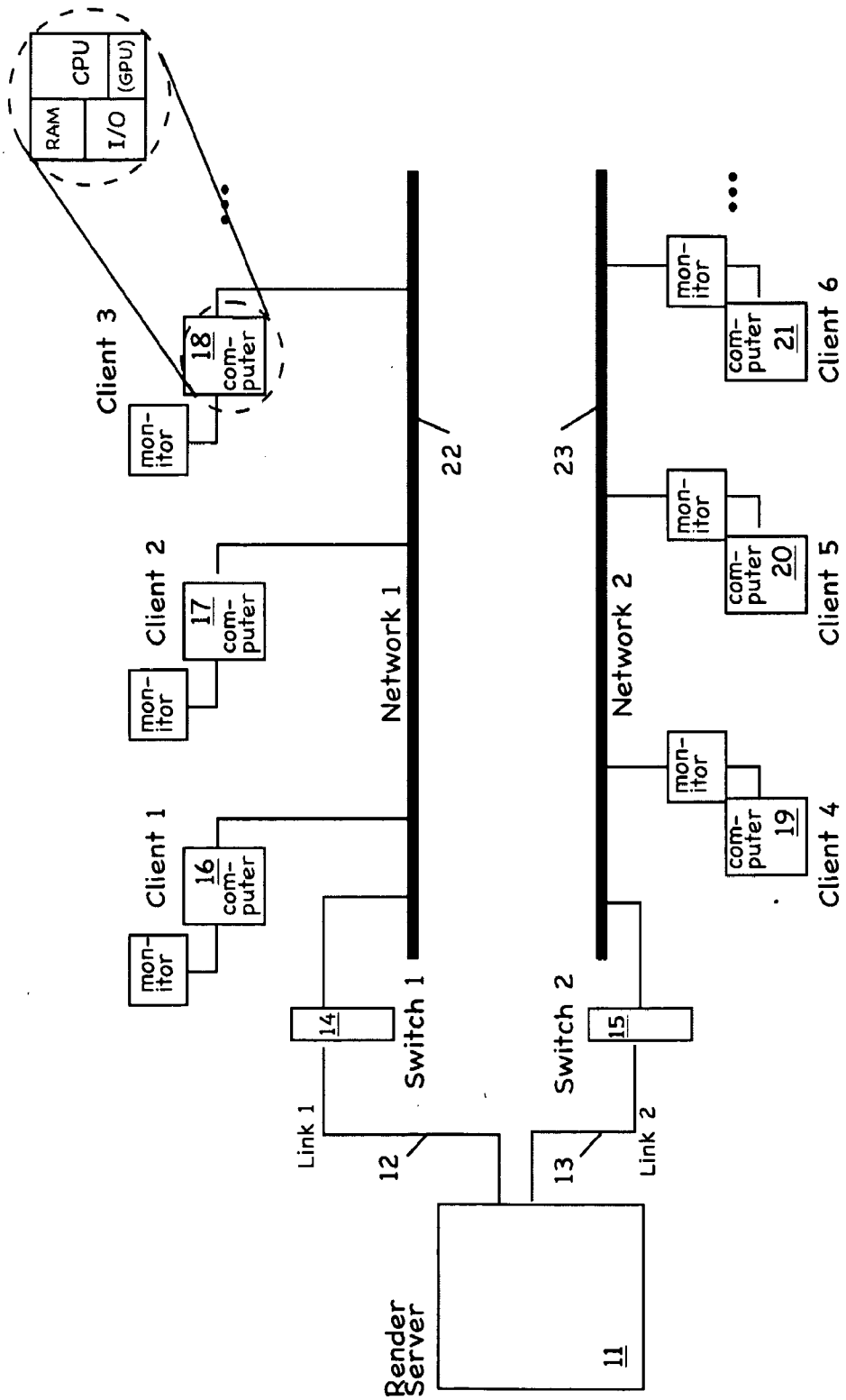
CPU
(GPU)
RAM
I/O
Client 3
18
com-puter
mon-itor
Client 2
17
com-puter
mon-itor
Client 1
16
com-puter
mon-itor
Network 1
22
23
Network 2
Client 6
21
com-puter
mon-itor
Client 5
20
com-puter
mon-itor
Client 4
19
com-puter
mon-itor
14
Switch 1
15
Switch 2
Link 1
12
13
Link 2
Render Server
11

Figure 1

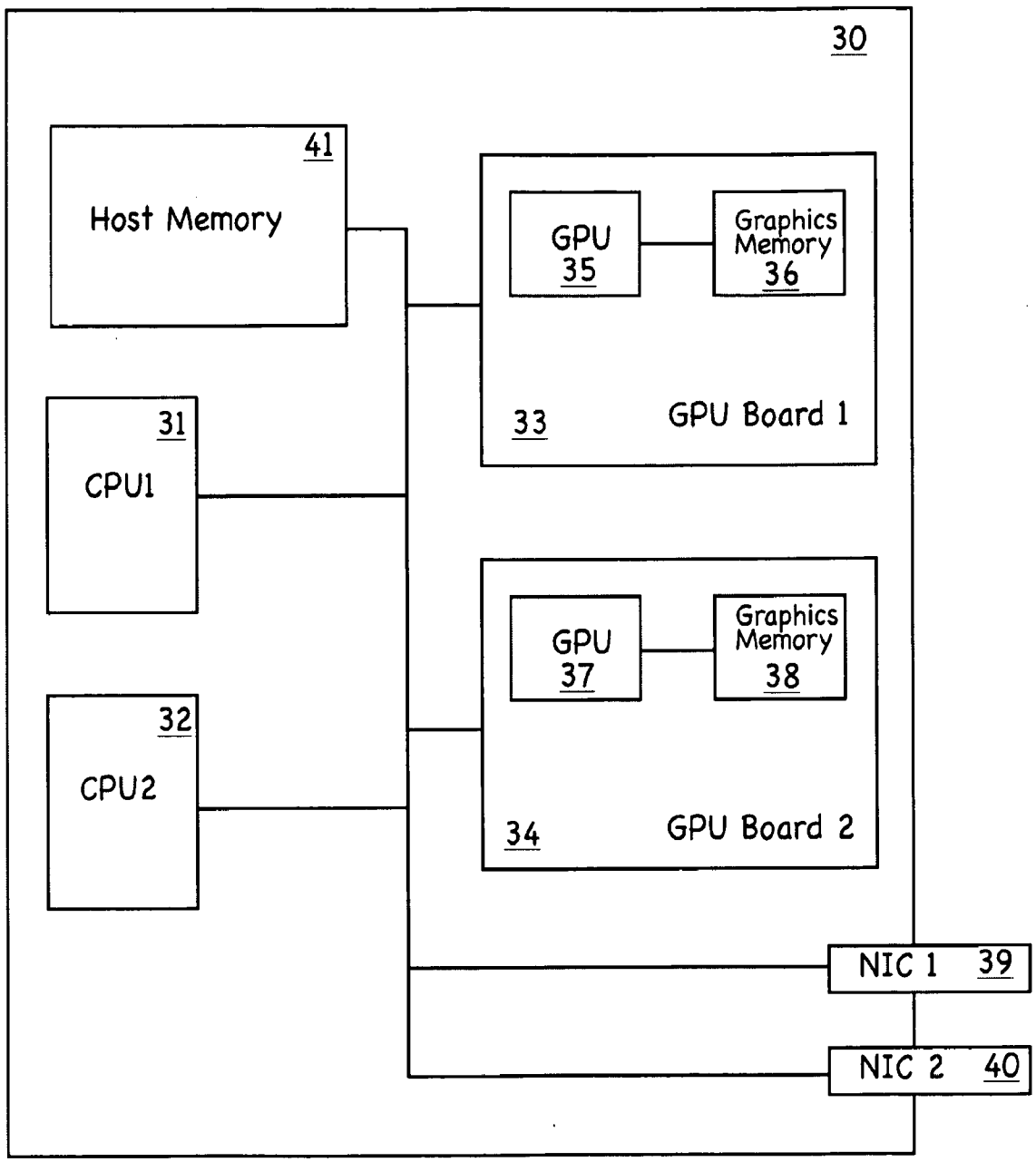
30
41
Host Memory
GPU
35
Graphics
Memory
36
33
GPU Board 1
31
CPU1
GPU
37
Graphics
Memory
38
32
CPU2
34
GPU Board 2
NIC 1 39
NIC 2 40

Figure 2

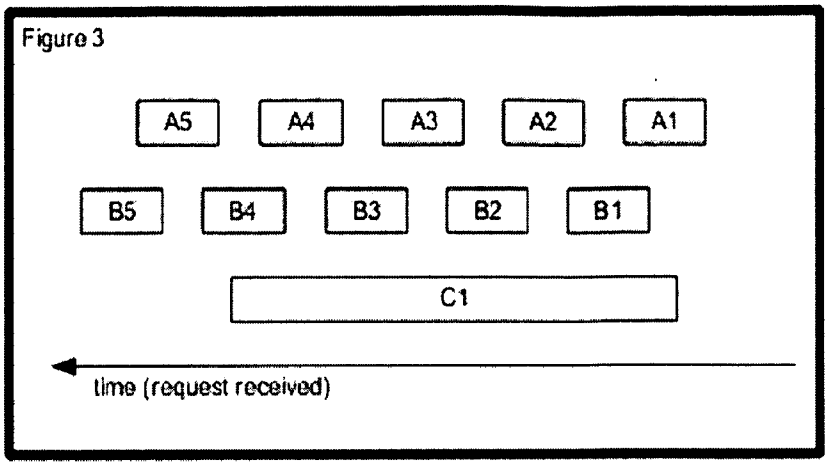

Figure 3

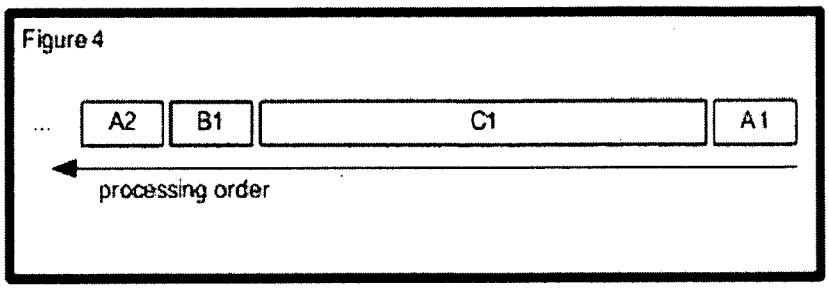

Figure 4

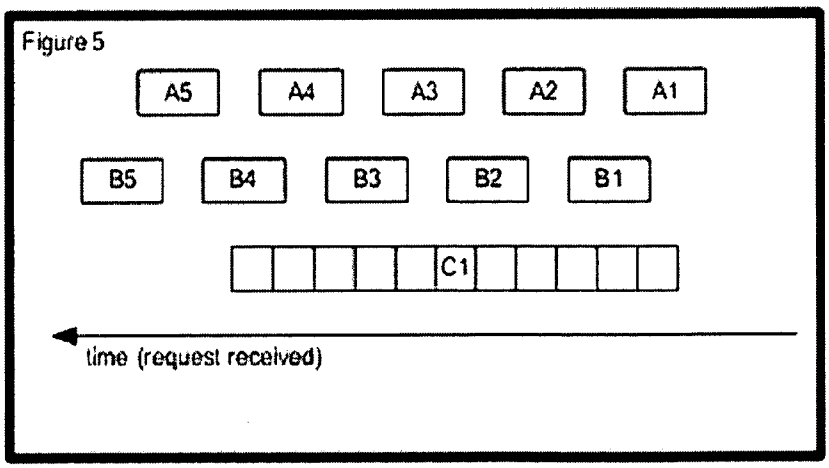

Figure 5

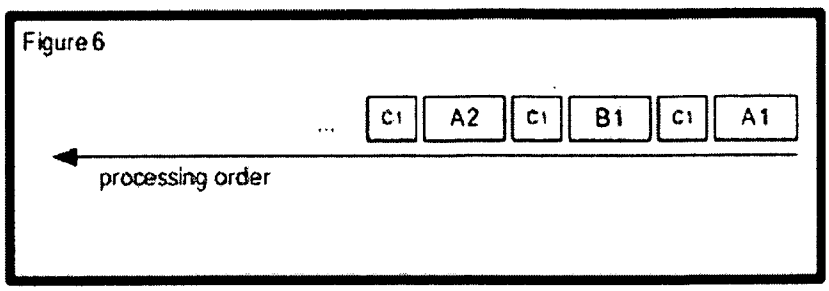

Figure 6

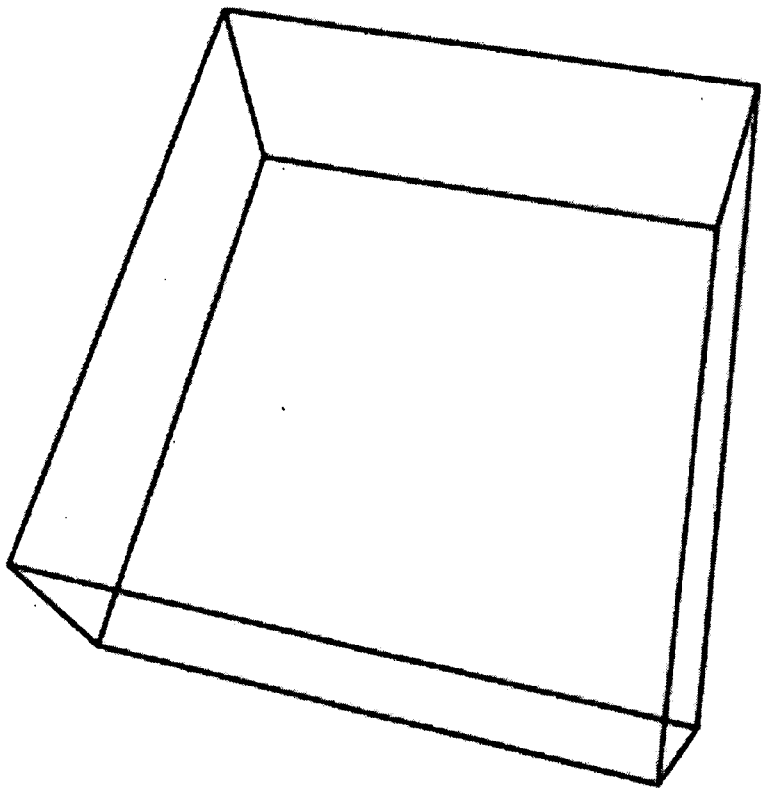

Figure 7

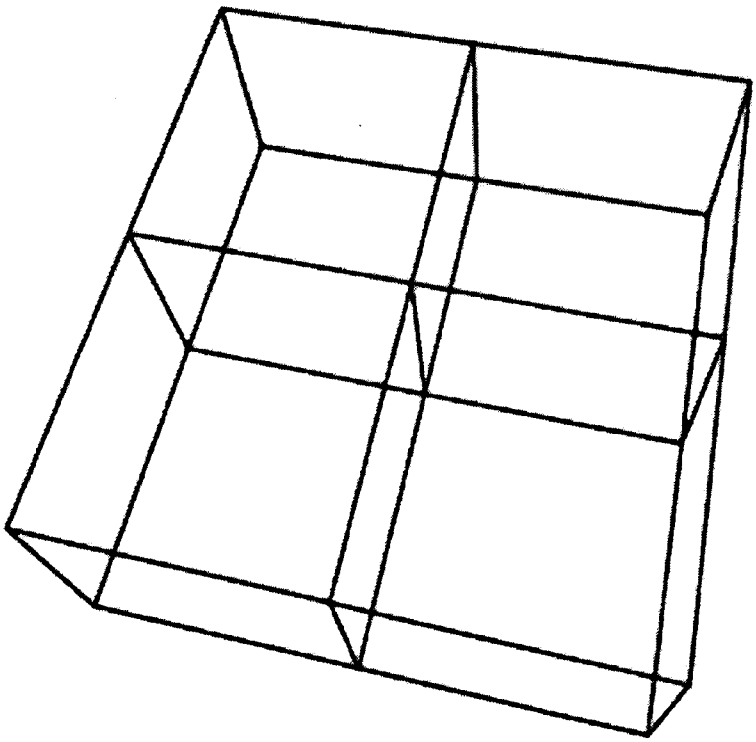

Figure 8

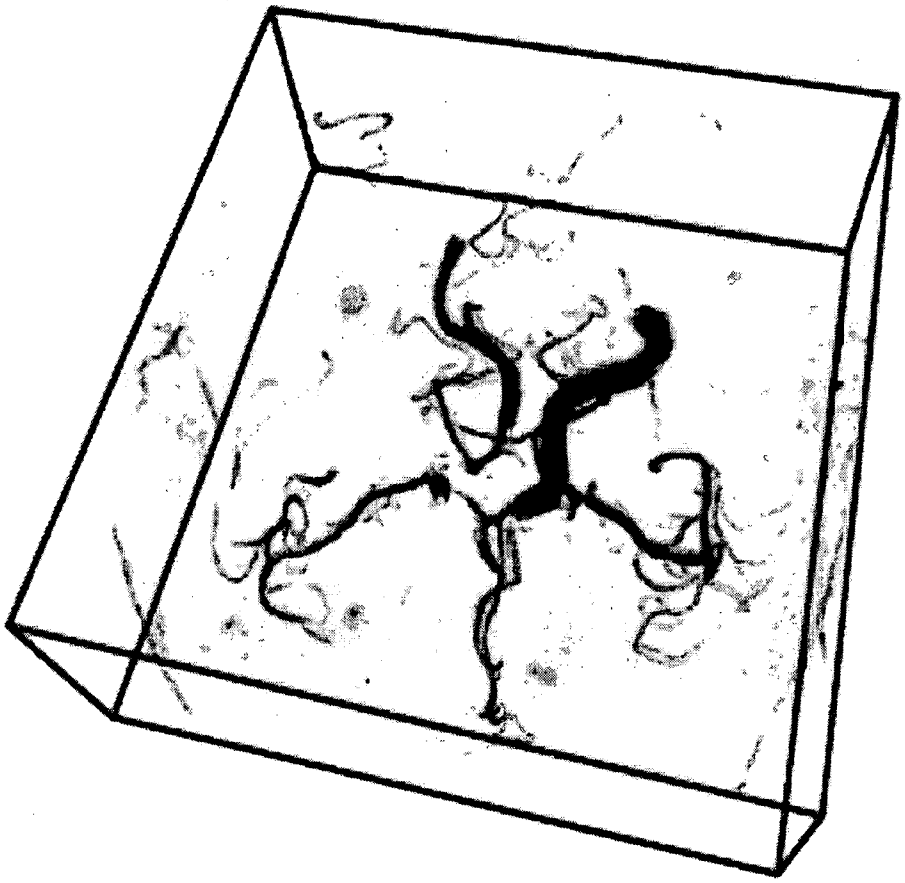

Figure 9

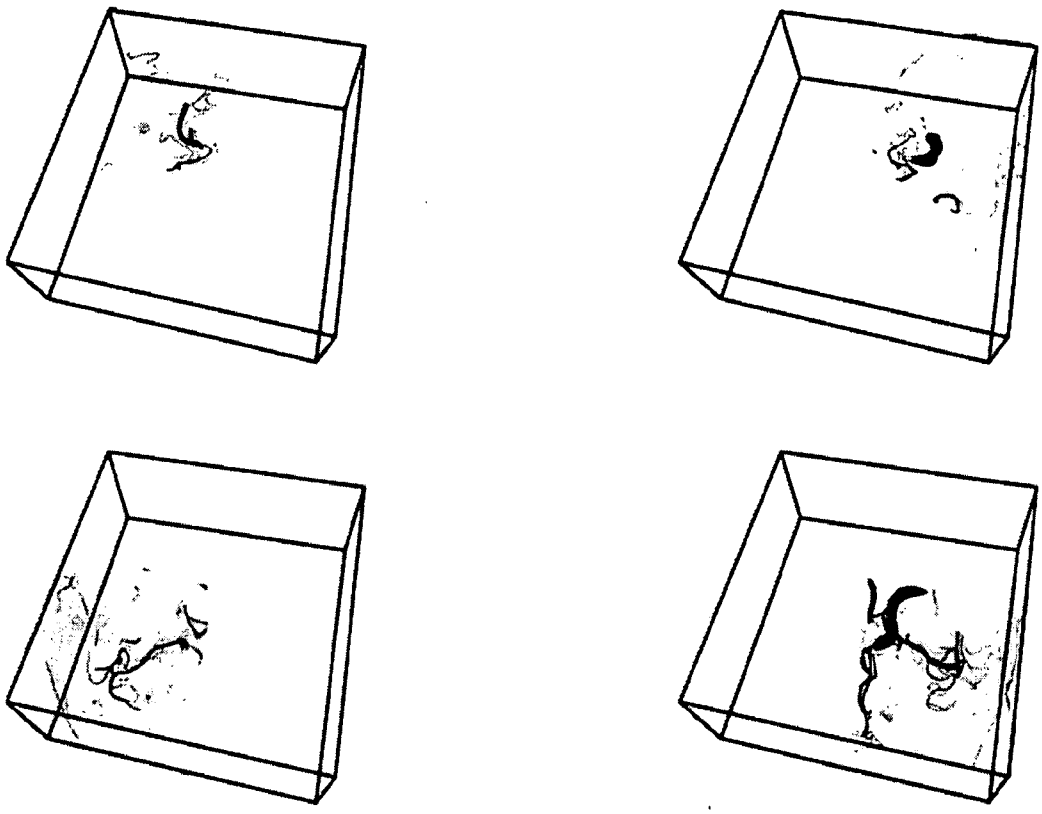

Figure 10

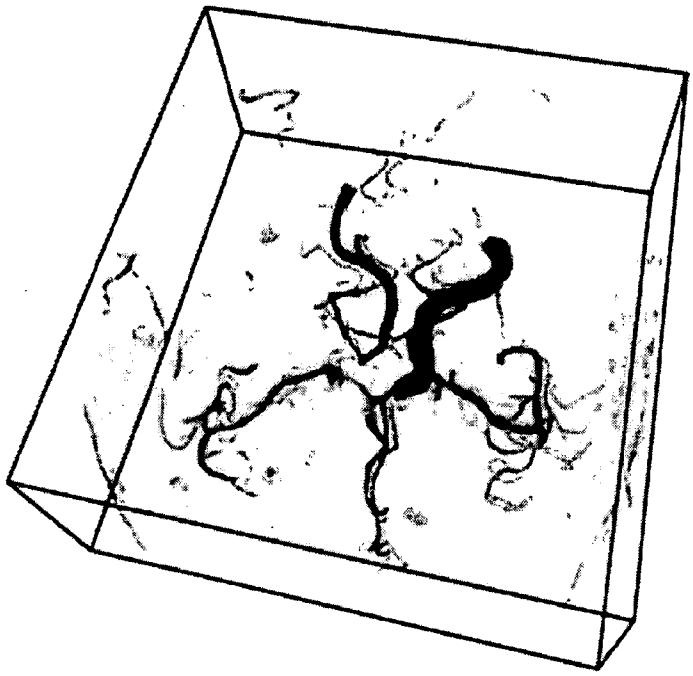

Figure 11

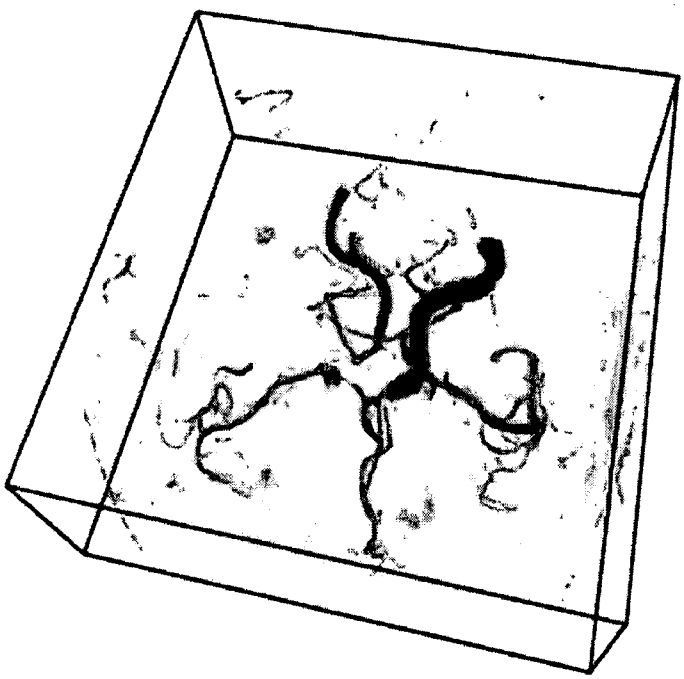

Figure 12

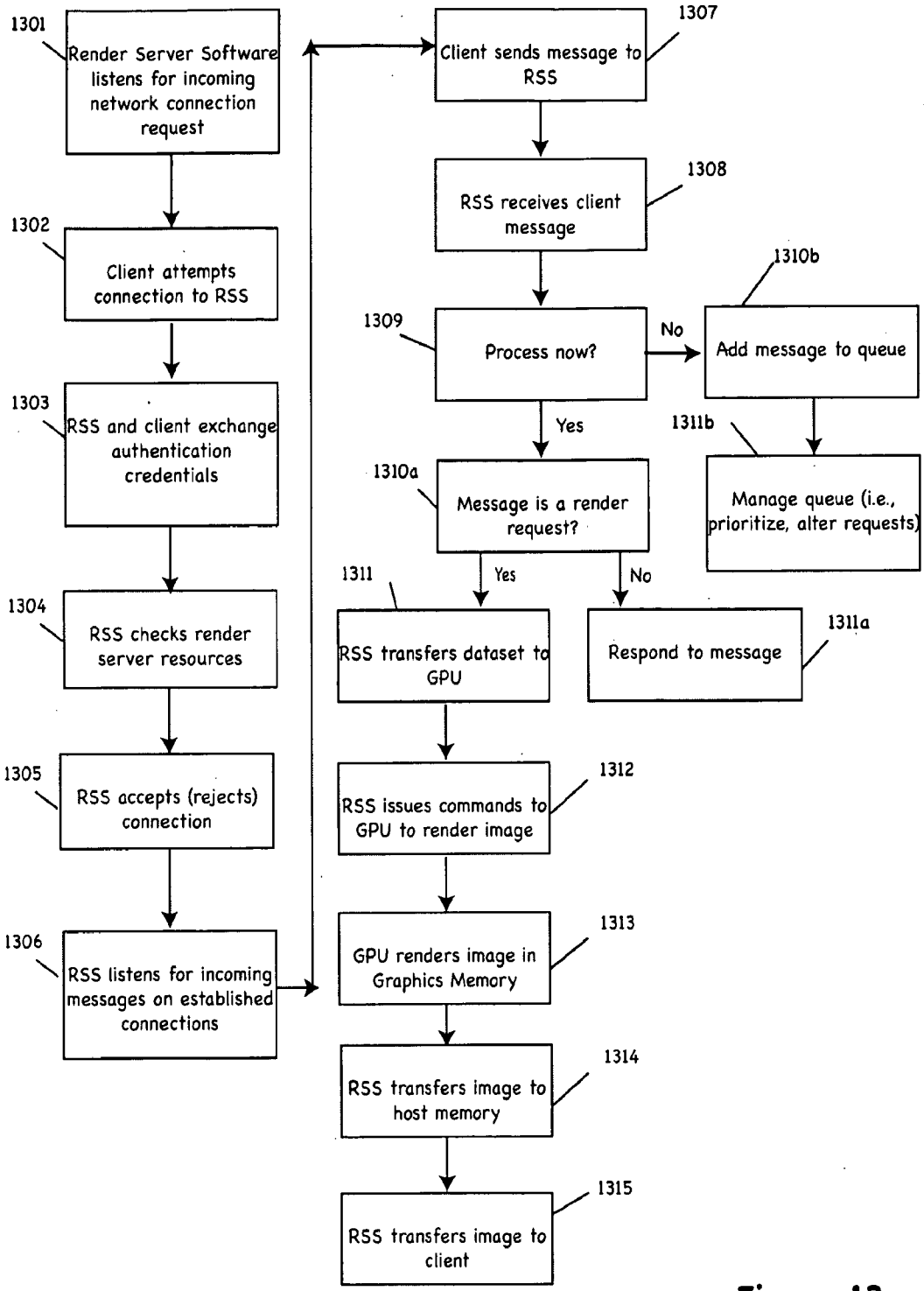
1301
Render Server Software listens for incoming network connection request
1302
Client attempts connection to RSS
1303
RSS and client exchange authentication credentials
1304
RSS checks render server resources
1305
RSS accepts (rejects) connection
1306
RSS listens for incoming messages on established connections
1307
Client sends message to RSS
1308
RSS receives client message
1309
Process now?
No
1310b
Add message to queue
1311b
Manage queue (i.e., prioritize, alter requests)
Yes
1310a
Message is a render request?
Yes
No
1311
RSS transfers dataset to GPU
Respond to message
1311a
1312
RSS issues commands to GPU to render image
1313
GPU renders image in Graphics Memory
1314
RSS transfers image to host memory
1315
RSS transfers image to client

Figure 13

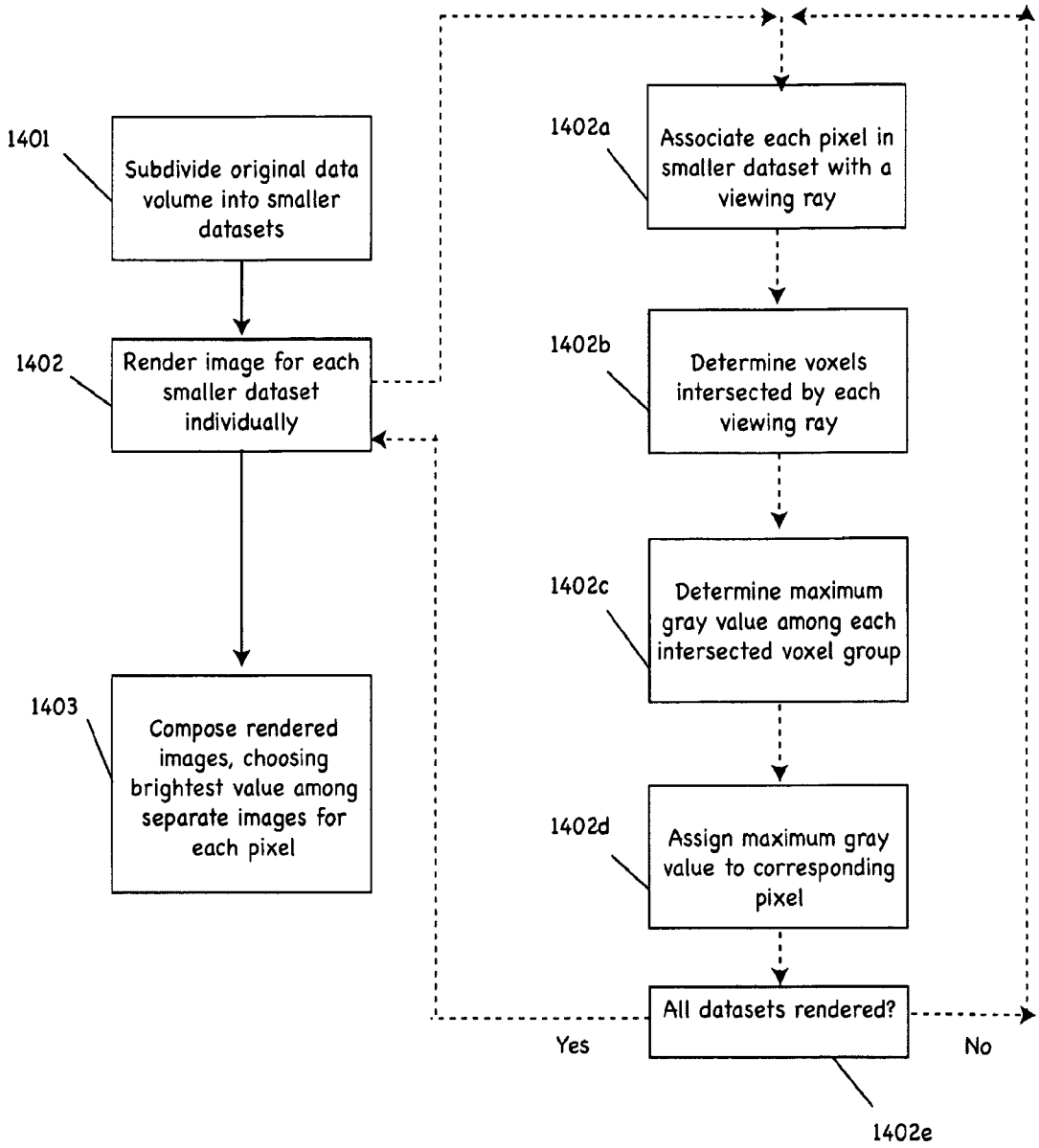
1401
Subdivide original data volume into smaller datasets
1402
Render image for each smaller dataset individually
1403
Compose rendered images, choosing brightest value among separate images for each pixel
1402a
Associate each pixel in smaller dataset with a viewing ray
1402b
Determine voxels intersected by each viewing ray
1402c
Determine maximum gray value among each intersected voxel group
1402d
Assign maximum gray value to corresponding pixel
All datasets rendered?
Yes
No
1402e

Figure 14

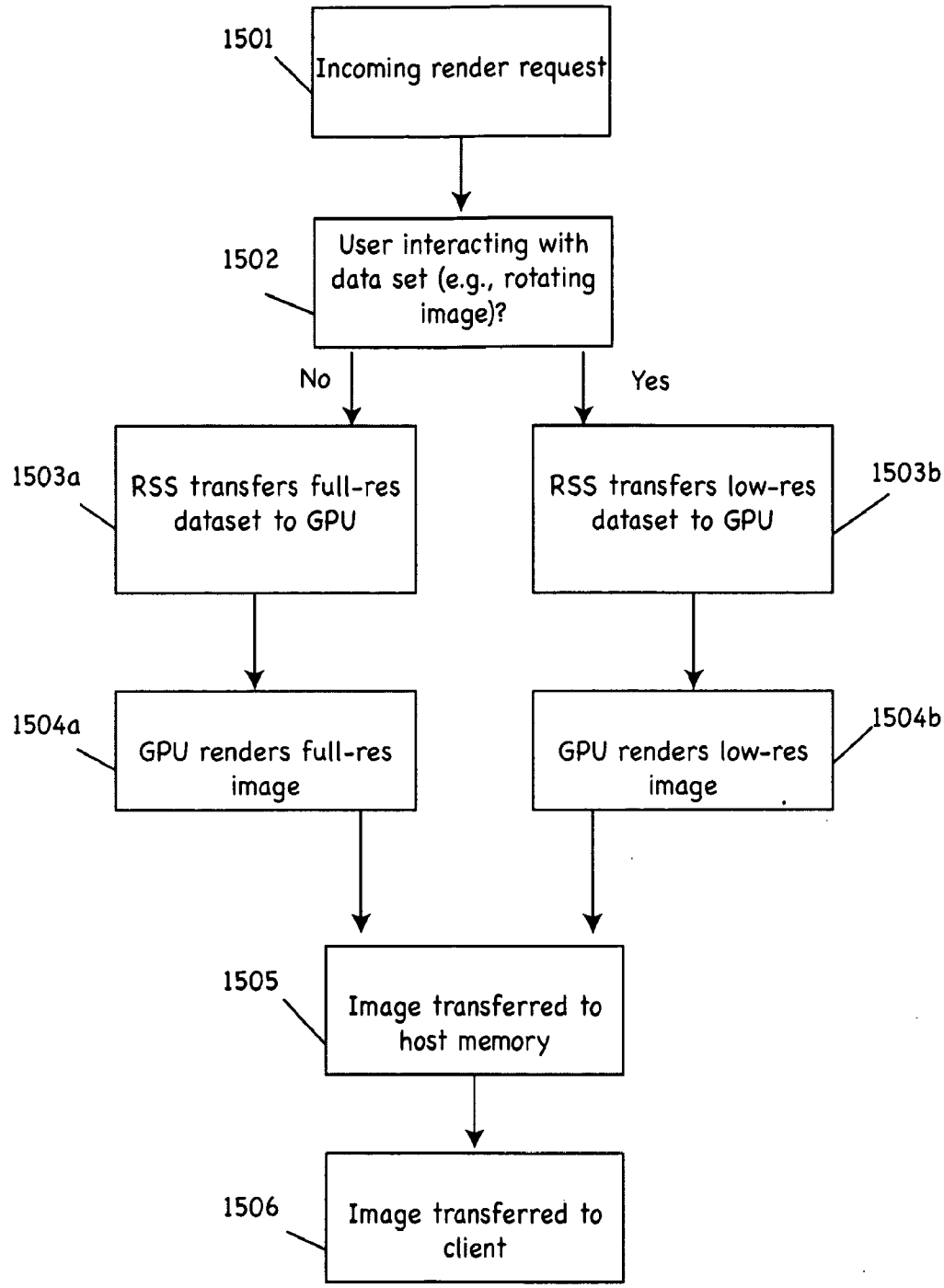
1501
Incoming render request
1502
User interacting with data set (e.g., rotating image)?
No
Yes
1503a
RSS transfers full-res dataset to GPU
1503b
RSS transfers low-res dataset to GPU
1504a
GPU renders full-res image
1504b
GPU renders low-res image
1505
Image transferred to host memory
1506
Image transferred to client

Figure 15

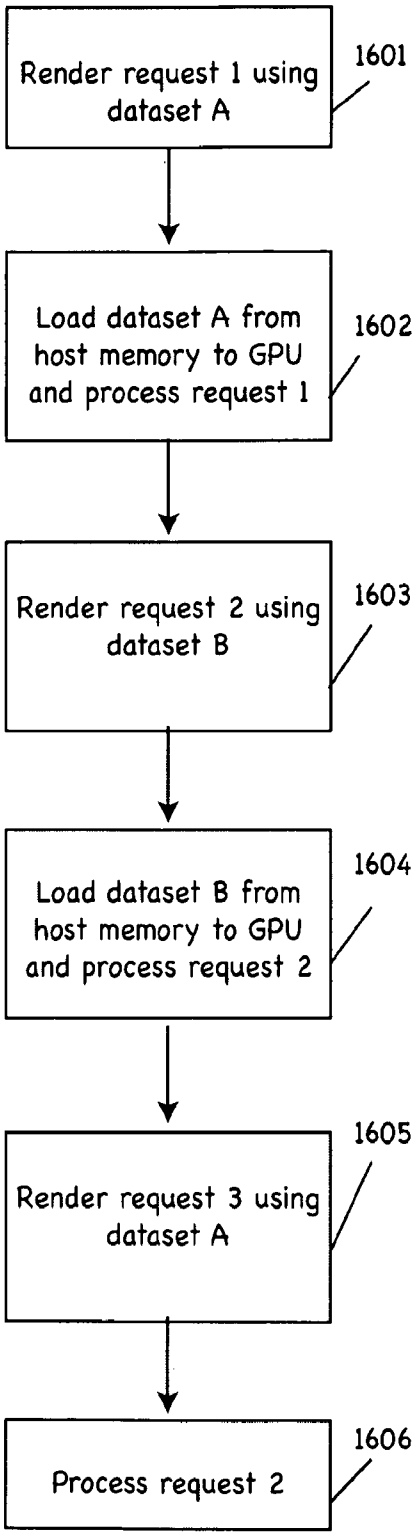
Render request 1 using dataset A
1601
Load dataset A from host memory to GPU and process request 1
1602
Render request 2 using dataset B
1603
Load dataset B from host memory to GPU and process request 2
1604
Render request 3 using dataset A
1605
Process request 2
1606

Figure 16a

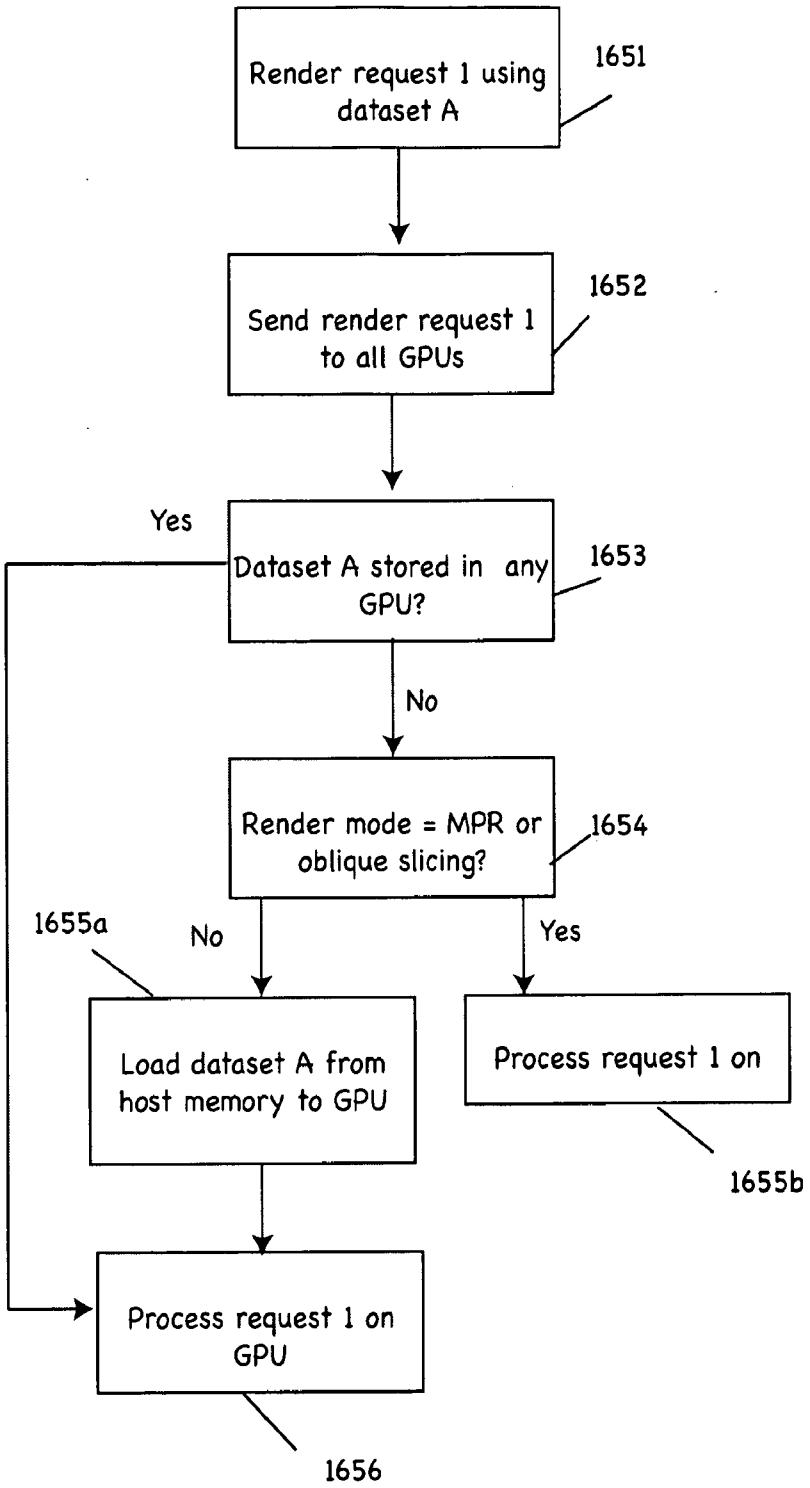
Render request 1 using dataset A
1651
Send render request 1 to all GPUs
1652
Yes
Dataset A stored in any GPU?
1653
No
Render mode = MPR or oblique slicing?
1654
1655a
No
Yes
Load dataset A from host memory to GPU
Process request 1 on
1655b
Process request 1 on GPU
1656

Figure 16b

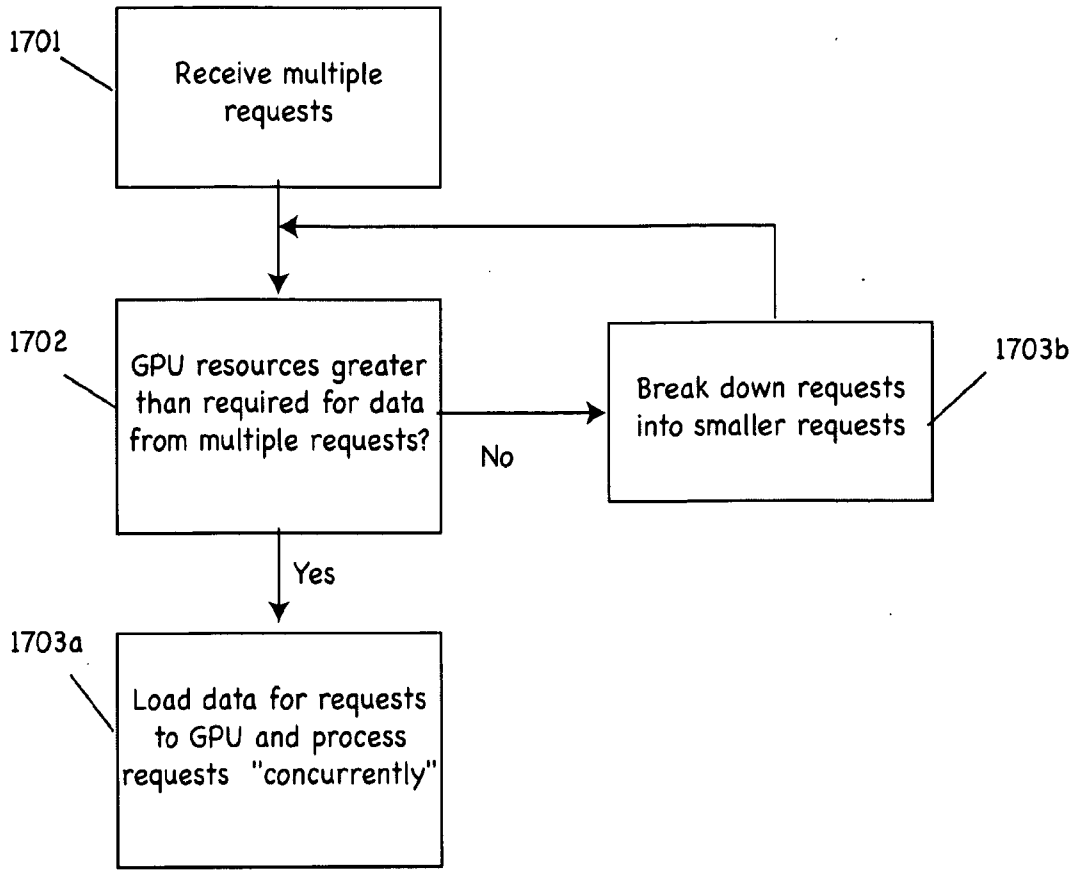
1701
Receive multiple requests
1702
GPU resources greater than required for data from multiple requests?
No
1703b
Break down requests into smaller requests
Yes
1703a
Load data for requests to GPU and process requests "concurrently"

Figure 17

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 08/84282

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06T 1/00 (2011.01)
USPC - 345/522
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
USPC 345/522

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC 345/420, 520, 522, 619; 709/207, 231; 715/740, 744 (text search--see below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PubWest (PGPB,USPT,EPAB,JPAB); Google Scholar (Patents,Articles)
Search terms: GPU, graphics, video, card, processor, accelerator, server, remote, render, command, call, request, break, split, brick, alternate, interleave, queue, buffer, FIFO, stream, pipeline, priority, preference, smaller, shorter, faster

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2006/0028479 A1 (CHUN et al.) 09 February 2006 (09.02.2006) entire document, especially Abstract; Figs. 3, 5, 10; para [0023], [0032]-[0036], [0074], [0075], [0097]	1-30
Y	US 2007/0097133 A1 (STAUFFER et al.) 03 May 2007 (03.05.2007) entire document, especially Abstract; Fig. 3; para [0031]	1-30
Y	US 2004/0066384 A1 (OHBA) 08 April 2004 (08.04.2004) entire document, especially Abstract; Figs. 1A, B; para [0015], [0061]	7, 8, 17, 18, 23, 25, 26
Y	US 2007/0156955 A1 (ROYER, JR. et al.) 05 July 2007 (05.07.2007) entire document, especially Abstract; Fig. 1; para [0009], [0010]	9, 24
Y	US 7,076,735 B2 (CALLEGARI) 11 July 2006 (11.07.2006) entire document, especially Abstract; Fig. 4; col. 8, ln 28-47	10, 11, 27
A	US 7,274,368 B1 (KESLIN) 25 September 2007 (25.09.2007) entire document, especially Abstract; Figs. 1, 2; col. 3, ln 30 to col. 4, ln 22	1-30
A	US 6,798,417 B1 (TAYLOR) 28 September 2004 (28.09.2004) entire document, especially Abstract; Figs. 4-6; col. 4, ln 46 to col. 5, ln 52	1-30

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

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 April 2011 (13.04.2011)	05 MAY 2011
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)