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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 June 2014 (26.06.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/099301 A1

(51) International Patent Classification:
G06F 3/14 (2006.01) *G06F 11/34* (2006.01)

(21) International Application Number:
PCT/US2013/071913

(22) International Filing Date:
26 November 2013 (26.11.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/718,199 18 December 2012 (18.12.2012) US

(71) Applicant: **MOTOROLA MOBILITY LLC** [US/US]; 600 North US Highway 45, Libertyville, IL 60048 (US).

(72) Inventor: **ELMIEH, Baback**; 3388 Bryant Street, Palo Alto, CA 94306 (US).

(74) Agents: **BRETSCHER, John, T.** et al.; 600 North US Highway 45, Libertyville, IL 60048 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, 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, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, 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, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

(54) Title: METHODS AND SYSTEMS FOR OVERRIDING GRAPHICS COMMANDS

(57) Abstract: Disclosed are "graphics overrides." An override accepts (300) a stream of graphics commands as produced by an application (108, 110, 112) and then modifies (304) the stream before it is rendered (310) by the GPU (120). Different overrides perform different modifications. One override can modify a stream of graphics commands in response to another stream. Overrides can enforce conformity with a visual paradigm and, by being modified, can support a change to that paradigm without requiring the applications to change. Overrides can monitor the entire computing environment and improve the response to that environment of a particular application: For example, an override monitors (506) frames as they are produced by an application. If the application cannot keep up (510) with a fixed frame rate, then the override produces (508) "synthetic" frames (604, 608, 610, 612) to take the place of missing frames. Overrides are not restricted to fixing existing problems. Rather, applications can be developed that depend upon (700, 800) the presence of overrides.

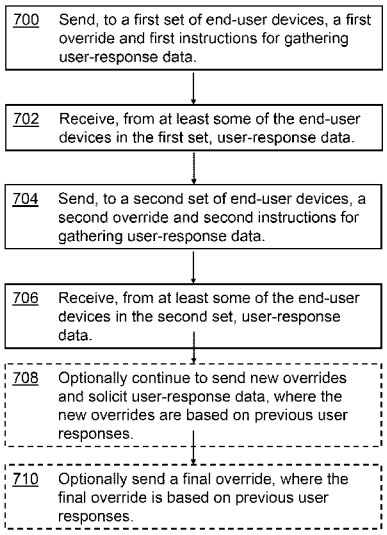


FIG. 7

WO 2014/099301 A1

METHODS AND SYSTEMS FOR OVERRIDING GRAPHICS COMMANDS

TECHNICAL FIELD

[0002] The present disclosure is related generally to computer graphics processing and, more particularly, to displaying information on a computer screen.

BACKGROUND

[0003] In a modern computing device, applications, utilities, and the operating system all run on, and share the processing power of, a central processing unit ("CPU").

[0004] (Note 1: Computing devices now commonly parse out their work among multiple processors. For clarity's sake, the present discussion uses the term "CPU" to cover all single- and multiple-processor computing architectures.)

[0005] (Note 2: The phrase "applications, utilities, and the operating system" is shorthand for the set of all entities that may present information to a computer screen. Distinctions among these entities are, for the most part, of little import to the present discussion. Therefore, for clarity's sake, all of these entities are all usually called "applications" from here on unless a particular distinction is being drawn.)

[0006] When an "application" needs to present visual information to a user of a computing device, rather than directly issuing commands to the hardware that controls a display screen of that device, the application writes a series of graphics commands. These commands are sent to a graphics processing unit ("GPU") which interprets the commands and then draws on the display screen.

[0007] Because the applications, utilities, and the operating system running on the device may all send graphics commands to the GPU, conflicts among the commands may arise. As a simple example, an application may command that a window be opened in a portion of the screen in which the operating system wishes to draw a background image (or "wallpaper"). The GPU knows the presentation hierarchy established among the applications (i.e., which one's display information should be "on top" and should thus overwrite that of the others) and resolves any conflicts accordingly. In the present example, the application is on top, so its window occludes

a portion of the background wallpaper displayed by the operating system. Instead of occlusion, the GPU can be commanded to merge disparate images to make, for example, an "upper" image semi-transparent so that a "lower" image can be seen through it.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0008] While the appended claims set forth the features of the present techniques with particularity, these techniques, together with their objects and advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

[0009] Figure 1 is a generalized schematic of a computing device in which the present techniques may be practiced;

[0010] Figure 2 is a dataflow diagram showing overrides being applied to streams of graphics commands;

[0011] Figure 3 is a flowchart of a representative method for applying a graphics override;

[0012] Figure 4 is a flowchart of an alternative method for applying a graphics override;

[0013] Figures 5a and 5b together form a flowchart of a representative method for creating synthetic frames;

[0014] Figure 6 is a rendering timeline for a series of application-produced and synthetic frames;

[0015] Figure 7 is a flowchart of a representative method for a server to gather user-response data;

[0016] Figure 8 is a flowchart of a representative method for an end-user device to gather user-response data; and

[0017] Figure 9 is a schematic diagram of a selection process driven by user-response data.

DETAILED DESCRIPTION

[0018] Turning to the drawings, wherein like reference numerals refer to like elements, techniques of the present disclosure are illustrated as being implemented in a suitable environment. The following description is based on embodiments of the claims and should not be taken as limiting the claims with regard to alternative embodiments that are not explicitly described herein.

[0019] Powerful as it is, the existing model in which applications send graphics commands to be implemented by a GPU presents significant limitations. A first limitation is based on what may be called the "too-many-authors" problem: In current open computing architectures, applications are developed by a disparate set of authors, and the applications are hosted by a computing device running an operating system and utilities written by yet other authors. It would be surprising if all of these authors together accidentally presented a unified visual scene. Existing GPUs are not intelligent enough to radically transform the visual information they are commanded to present, therefore if a designer of a computing device wishes to have all of the applications running on the device conform to a given visual paradigm, then one option would be to constrain the applications to use only a restricted set of graphics commands (a set that, for example, does not allow changing certain fixed parameters). Of course, this reduces the flexibility of the developed applications, and that could be annoying to authors and to users alike. Also, because there are so many different applications controlled by no single authority, it may become essentially impossible to update all of the applications when the visual paradigm is changed.

[0020] A second, and more reasonable, response to the too-many-authors problem is to allow the authors the use of the full set of graphics commands but to publish guidelines for implementing the visual paradigm. Often, however, application authors fail to follow the guidelines, either through a misunderstanding or in order to achieve an effect not allowed for by the guidelines. And, as with the first option, there may be no feasible update path when the paradigm shifts.

[0021] A second limitation posed by the application-to-GPU model may be called the "too many externalities" problem: It is very difficult to draft an application that can account for everything in the computing environment outside of the application itself that may be running concurrently on the computing device, especially as that "everything" may include applications and other software written by the disparate set of authors mentioned above. When computing resources are in high demand, a given application may begin to fail or to operate in an unanticipated fashion. As the computing environment changes with the addition of more and different applications, it becomes unrealistic to assume that all of the applications are written in such a manner that they can handle the strain. Because of the too-many-authors problem, it is also unrealistic to assume that these applications can all be upgraded as needed whenever the computing environment changes.

[0022] To address these and other limitations of the application-to-GPU model, the present disclosure introduces the concept of a "graphics override." The override accepts a stream of graphics commands as produced by an application and then modifies the stream before it is rendered by the GPU.

[0023] Generally speaking, the earliest overrides are implemented as code (or as parameters for a more generic code) to run on the CPU. In the future, it is expected that some overrides will be hosted directly by the GPU.

[0024] Different overrides perform different modifications. As a simple example, one override can review all of the incoming graphics commands from all applications and other software, find the commands that call for text to be displayed, and modify those commands so that they conform to a standard (e.g., they all use the same font and contrast).

[0025] Overrides can be very general or very specific in their focus. One override can modify a stream of graphics commands in response to another stream. Overrides can enforce conformity with a visual paradigm and, by being modified, can support a change to that paradigm without requiring the applications to change. Overrides can monitor the entire computing environment and improve the response to that environment of a particular application. In an example explained in detail below, an

application is supposed to produce frames at a fixed rate. An override monitors the frames as they are produced by the application. If the application cannot keep up with the fixed rate, then the override produces "synthetic" frames to take the place of missing frames. Thus, the overrides prevents the application from producing jerky output when the load on the CPU is heavy.

[0026] Overrides are not restricted to fixing existing problems. Rather, applications can be developed that depend upon the presence of overrides. In another example detailed below, overrides support "A/B testing" of a new software product. A product is sent to two sets of users. Each set is also given a particular override. The first set of users experiences the product as modified by a first override, and the second set experiences the same product but with a second override in place. User-response data are gathered to see which override is preferred. The "winning" override is then incorporated into the release version of the product. Future overrides can be sent to further modify the user experience, either to support further testing or to implement an altered experience.

[0027] To understand these concepts more fully, first consider the representative computing device 100 of Figure 1. The device 100 is meant to represent any computing device that presents visual information. It could be, for example, a personal communications device with a display screen, a mobile telephone, a personal digital assistant, or a personal or tablet computer. In some embodiments, the device 100 presents visual information to be displayed on a screen separate from the device 100 itself, such as a set-top box, gaming console, or server. The device 100 could even be a plurality of servers working together in a coordinated fashion.

[0028] The CPU 102 of the device 100 includes one or more processors (i.e., any of microprocessors, controllers, and the like) or a processor and memory system which processes computer-executable instructions to control the operation of the device 100. The device 100 can be implemented with a combination of software, hardware, firmware, and fixed-logic circuitry implemented in connection with processing and control circuits, generally identified at 104. Although not shown, the device 100 can include a system bus or data transfer system that couples the various

components within the device 100. A system bus can include any combination of different bus structures, such as a memory bus or memory controller, a peripheral bus, a universal serial bus, and a processor or local bus that utilizes any of a variety of bus architectures.

[0029] The computing device 100 also includes one or more memory devices 106 that enable data storage, examples of which include random-access memory, non-volatile memory (e.g., read-only memory, flash memory, EPROM, and EEPROM), and a disk storage device. A disk storage device may be implemented as any type of magnetic or optical storage device, such as a hard disk drive, a recordable or rewriteable disc, any type of a digital versatile disc, and the like. The device 100 may also include a mass-storage media device.

[0030] The memory system 106 provides data storage mechanisms to store device data 114, other types of information and data, and various device applications 112. An operating system 108 can be maintained as software instructions within the memory 106 and executed by the CPU 102. The device applications 112 may also include a device manager, such as any form of a control application or software application. The utilities 110 may include a signal-processing and control module, code that is native to a particular component of the device 100, a hardware-abstraction layer for a particular component, and so on.

[0031] The computing device 100 also includes an audio-processing system 116 that processes audio data and controls an audio system 118 (which may include, for example, speakers). A GPU 120 processes graphics commands and visual data and controls a display system 122 that can include, for example, a display screen. The audio system 118 and the display system 122 may include any devices that process, display, or otherwise render audio, video, display, or image data. Display data and audio signals can be communicated to an audio component or to a display component via a radio-frequency link, S-video link, High-Definition Multimedia Interface, composite-video link, component-video link, Digital Video Interface, analog audio connection, or other similar communication link, represented by the media-data ports 124. In some implementations (e.g., a set-top box which drives a television monitor),

the audio system 118 and the display system 122 are components external to the device 100. Alternatively (e.g., in a cellular telephone), these systems 118, 122 are integrated components of the device 100.

[0032] The computing device 100 can include a communications interface which includes communication transceivers 126 that enable wired or wireless communication. Example transceivers 126 include Wireless Personal Area Network radios compliant with various IEEE 802.15 standards, Wireless Local Area Network radios compliant with any of the various IEEE 802.11 standards, Wireless Wide Area Network cellular radios compliant with 3GPP standards, Wireless Metropolitan Area Network radios compliant with various IEEE 802.16 standards, and wired Local Area Network Ethernet transceivers.

[0033] The computing device 100 may also include one or more data-input ports 128 via which any type of data, media content, or inputs can be received, such as user-selectable inputs (e.g., from a keyboard, from a touch-sensitive input screen, or from another user-input device), messages, music, television content, recorded video content, and any other type of audio, video, or image data received from any content or data source. The data-input ports 128 may include USB ports, coaxial-cable ports, and other serial or parallel connectors (including internal connectors) for flash memory, storage disks, and the like. These data-input ports 128 may be used to couple the device 100 to components, peripherals, or accessories such as microphones and cameras.

[0034] Before considering detailed techniques for using overrides (see the discussion below accompanying Figures 3 through 9), turn to Figure 2 which gives examples of where overrides can be implemented in the processing architecture. Running on the CPU 102 of the computing device 100 are an operating system 108, a utility program 110, and an application 112.

[0035] In Figure 2, the operating system 108 is operating according to prior conventions, that is, without an override. The operating system 108 produces a stream of graphics commands 200a that are sent to the GPU 120. The GPU 120 implements these commands 200a as appropriate and thereby controls the display system 122.

(Note: As discussed above, commands from various sources 108, 110, 112 running on the CPU 102 may conflict, so the GPU 120 implements all commands "as appropriate" by considering the display context and the hierarchy of the sources 108, 110, 112 of the graphics commands. Thus, the GPU 120 may perform some graphics commands exactly as specified, may ignore others, and may perform still others in a manner different from what the commands explicitly call for.)

[0036] Next consider the utility 110. It produces its own stream of graphics commands 200b. However, before those commands 200b get to the GPU 120, they are intercepted by the CPU-hosted override 202. That override 202 examines and amends its incoming stream 200b. It then sends an amended stream 200c to the GPU 120 which processes the amended stream 200c just as it would process any other stream of graphics commands (paying attention, of course, to the source and context of that stream).

[0037] Finally, the application 112 produces its stream of graphics commands 200d which are dutifully sent to the GPU 120. Before implementing these commands 200d, however, the GPU-hosted override 204 intercepts them and modifies them to produce the amended stream 200e. This amended stream 200e is then processed within the GPU 120. (Note that most current GPUs cannot host overrides such as 204. However, as GPUs become more powerful and flexible, it is anticipated that this will become a useful adjunct to CPU-hosted overrides like 202.)

[0038] Note that the assignment of the three techniques in Figure 2, i.e., no override to the operating system 108, a CPU-hosted override 202 to the utility 110, and a GPU-hosted override 204 to the application 112 is completely arbitrary. Any type of software process 108, 110, 112 may use any of these techniques and may even use more than one technique at the same time. Thus, the amended stream 200c produced by the CPU-hosted override 202 may itself be intercepted by another CPU-hosted override or even by a GPU-hosted override. Also, some of the graphics commands 200d produced by the application 112 may go to the override 204 while other commands go to another override. These are straightforward matters of implementation and management of the presently disclosed techniques.

[0039] With this general background in mind, turn to Figure 3 which presents a representative method for using a CPU-hosted override such as the override 202 of Figure 2.

[0040] The method begins at step 300 where a stream of graphics commands 200b is produced as output by an source application 108, 110, 112. In some embodiments, the source 108, 110, 112 is not aware of the fact that its output is going to an override. In contrast with this, the discussion accompanying Figures 7 through 9 presents a circumstance in which the source application is written to depend upon the presence of an override.

[0041] In optional step 302, a second stream of graphics commands is received. This step (along with optional step 306 discussed below) emphasizes that the relation between sources 108, 110, 112 and overrides need not be one-to-one: A single override may accept graphics commands from multiple sources 108, 110, 112.

[0042] In step 304, the override 202 is applied to its input stream of graphics commands 200b and produces an amended output stream of graphics commands 200c.

[0043] The override 202 is a code instance separate from the source 108, 110, 112 of the stream of graphics commands 200b. Although separate, in some embodiments, the override 202 can be given full access to information about the internal status of the source 108, 110, 112 and about the context in which that source 108, 110, 112 is running (see the example of Figures 5 and 6, discussed below), including, for example, the relationship of the current source to other applications and possibly the environment beyond the computing device 100 (e.g., an override may access the Web to gather important information). On the other hand, other overrides may be limited in their knowledge (for security's sake) and may not even be aware of the source 108, 110, 112 of the input stream 200b but may simply process the input stream 200b without regard to the identity or nature of the source 108, 110, 112 of that stream 200b.

[0044] In general, an override 202 can be developed to perform any type of amendment to the input stream 200b, including adding, deleting, modifying, and re-arranging graphics commands. The amendment may simply change a single parameter in a single graphics command as, for example, when the input "stream" 200b has a single command to draw a blue box, and the amended output stream 200c has a single command to draw a red box. Or the override 202 can change all text commands so that the output 200c (and potentially the output from all sources 108, 110, 112) has a consistent font or other appearance parameter.

[0045] In some cases, the output stream 200c stream may be null, that is, the override 202 cancels, at least for a time, the visual output of the source 108, 110, 112.

[0046] The processing performed by a given override 202 can be very simple, as shown by the examples given above, or very complex. For example, an override 202 may receive as input a static image 200b and change the image's perspective (e.g., tilt or scale it) or even add motion to the image 200b to produce a video output 200c. A sophisticated override 202 may generate, from the input graphics command stream 200b, a scene graph (as known in the art). The scene graph may then be modified, and the output stream of graphics commands 200c may then be created directly from the modified scene graph and, thus, only indirectly from the input stream of graphics commands 200b. Although creating and interpreting a scene graph requires a good deal of processing power, it is a good option for some overrides 202 because the override 202 can then apply existing graphics tools (e.g., an element identifier, shader, properties module, or physics engine) to the created scene graph to produce sophisticated effects. The override 202 may be implemented as a set of instructions or rules that take account of the current context and of past developments (i.e., the override 202 can have a memory). The rules of the override 202 may even specify when to invoke itself or another override.

[0047] In optional step 306, the override 202 is applied, at least in part, to the second input stream (see step 302) to produce a second output stream. Note that this technique can be applied to allow the override 202 to coordinate among multiple instances of the same, or different, sources 108, 110, 112. For example, multiple

players of a game may each run a separate instance of a game application 112. The override 202 receives the output graphics streams of all of the game-application instances, harmonizes them, and then either produces a separate output graphics stream 200c for each user, or a single output stream 200c for all the users (i.e., the first and second output streams 200c of steps 304 and 306 may be the same stream 200c, or the output streams 200c may each be based on the totality of the input streams 200b). In another example, an override 202 can accept the outputs of multiple applications 112 and modify those outputs so that they fit into a unified user interface, perhaps by tiling the separate inputs or by blending them together.

[0048] The output streams 200c produced by the override 202 are sent in step 308 to the GPU 120. The streams 200c may be identified to the GPU 120 as produced by their ultimate sources 108, 110, 112 or as produced by the override 202 itself. (As discussed above, the GPU 120 uses this source information for resolving conflicts among graphics commands.)

[0049] The GPU 120, which in the method of Figure 3 may be an existing GPU, renders, in step 310, the graphics streams it receives, including non-amended streams such as 200a (e.g., a status update message from the operating system 108; see Figure 2) and amended streams such as 200c. The rendering may be done directly to a display screen on, or hosted by, the computing device 100. In some situations, the rendered output of the GPU 120 may be sent to a communications interface 126 (as when a server performs graphics processing too burdensome for an end-user device) or stored in a memory system 106.

[0050] Figure 4 presents a method for implementing a GPU-hosted override, such as the override 204 of Figure 2. The considerations and details discussed above in relation to the CPU-hosted override 202 apply as well to this case and need not be repeated. In step 400, the GPU 120 receives an input stream of graphics commands 200d (see Figure 2). In step 402, the GPU 120 receives the override 204 from the CPU 102. (There is no significance to the ordering of steps 400 and 402: The override 204 may have been received in step 402 long before the stream of graphics commands 200d arrives and may even reside permanently on the GPU 120.) The override 204

amends the incoming stream of graphics commands 200d in step 404 to produce an amended output stream (not shown in Figure 2). The GPU 120 renders that output stream as appropriate.

[0051] It is anticipated that, in particular situations, hosting an override on the GPU 120 will have certain advantages over hosting a similar override on the CPU 102. Different GPUs support different features and have different strengths and weaknesses, so an override tailored to a particular GPU architecture may be more efficient than a CPU-hosted override. In a computing architecture where multiple CPUs provide input to a single GPU, the GPU may be the best place to coordinate a unified display paradigm.

[0052] Figures 5 and 6 present the case of a particular override that accounts for factors outside of the control of the source application 112 itself. Consider an application 112 that produces frames at a fixed rate. In some examples, the application 112 need only retrieve already existing frames from, say, the memory system 106 and present them to the GPU 120. In other examples, such as an interactive computer game or an enhanced reality display, the application 112 creates each frame, potentially using significant computational resources in order to do so. Because of resource competition and other conflicts, it is well know that in some situations an application 112 may fail to present its frames at the specified rate. Then, when resources become available, the application 112 may produce all of the frames that it missed earlier and send this series of frames to the GPU 120. These frames may then be rendered at an inappropriate rate. When viewed by a user, the resulting motion can be jerky or may suffer from other annoyances. An override can be developed to address these issues.

[0053] In step 500 of Figure 5a, a frame produced by the application 112 is received by the CPU 102. This application-produced frame is sent to the GPU 120 in step 502 and is rendered in step 504. Turning to Figure 6, this application-produced frame 600 is rendered at the first frame time.

[0054] As steps 500 through 504 are performed (perhaps repeatedly for frame after frame), an override 202 watches over this process. In step 506, the override 202

calculates (or otherwise determines) the time when a next frame needs to be presented. (In a very simple example, this would be the frame time of the next frame, based solely on the frame time of the first frame and the known frame rate. In other embodiments, the override 202 may be working further ahead in the anticipated sequence of frames.)

[0055] Step 508 is, in some situations, technically optional. In this step, the override 202 produces its own frame to be displayed at the next frame time. There are several possible procedures for producing such a "synthetic" frame. In a sophisticated example, the override 202 could examine the past few application-produced frames, note any consistent movement or other change in that short sequence, and then produce the synthetic frame by predicting a continuation of that movement into the next frame time. While the motion prediction may turn out to be incorrect, the viewed result would, in some situations, not be as jarring as simply repeating the last application-produced frame. With even more sophistication, the override 202 could gather further information from the hosting device 100 such as, for example, user-input data, and produce a scene graph based on the gathered information as well as on the previously received application-produced frames. From the scene graph, the override 202 could predict the visual effects that the application 112 is trying to produce. The override 202 could then produce the synthetic frame accordingly.

[0056] The override 202 can use information beyond the previously received application-generated frames when creating the synthetic frame. As one example, the override 202 can monitor user inputs to the computing device 100. If a user moves the computing device 100 away from himself (as detected by, say, an accelerometer), then the override 202 can create a synthetic frame that is a zoomed-out perspective of the previous application-produced frame. In another example, if the user touches a position on a touch-sensitive display screen, then the override 202 can shift the viewer's point-of-view based on that touch (or on a more elaborate gesture). (This can be enabled, for example, when the input stream of graphics commands 200b includes enough information to draw a scene larger than that currently displayed or includes enough information for the override 202 to create a scene graph of a fully three-

dimensional scene.) In sum, the override 202 can apply user-input information independently of mediation by the source application 112.

[0057] In step 510, the override 202, as it watches over the stream of application-produced frames, decides whether or not the application 112 is going to produce the next frame in time (an analysis based, in part, on the known frame rate). If it does, then everything is well, and the synthetic frame produced in step 508 is discarded. (Another procedure would be to only create the synthetic frame in step 508 if step 510 determines that the synthetic frame is needed. That is why step 508 is technically optional.)

[0058] If, on the other hand, the next application-produced frame is not available in time, then in step 512 of Figure 5b, the synthetic frame produced in step 508 is sent to the GPU 120 where it is rendered at the appropriate frame time in step 514.

[0059] This method is illustrated in Figure 6 where the application 112, after producing two frames 600 and 602 for the first and second frame times, fails to produce the next frame in time. The override 202 notes this and produces a synthetic frame 604 for the third frame time. By the fourth frame time, the application 112 catches up and produces frame 606. (If the application 112 also belatedly produces a frame for the third frame time, then in some embodiments, that frame is discarded.) After that, the application 112 fails to produce the next three frames in time, so the override 202 produces synthetic frames 608, 610, 612.

[0060] The process of Figures 5a and 5b continues, looping as long as the application 112 is attempting to produce frames. Note, however, that in some situations, the process may not be looping on a strictly frame-by-frame basis. Some applications 112 are able to produce a number of frames at one time and to send the series of frames to the GPU 120 to be rendered at the correct times. The teachings of Figures 5a and 5b apply to this scenario as well.

[0061] Finally, Figures 7 through 9 portray a scenario where an application 112 is developed that depends upon the presence of an override. This scenario is called "A/B" testing. This is a way to guide product development by soliciting the input of

users. In particular, two (or more) sets of users are given copies of an application 112. However, not all copies are identical. Normally, one set of users receives a copy that embodies one choice for a design or functional element (e.g., the "A" version), while the other set of users gets a copy that embodies a different choice (the "B" version). Users provide feedback, and from that feedback it is determined which choice is more popular. An advantage of using an override is that the application 112 need not be changed during testing: Only the overrides are different.

[0062] For more specific details of how overrides can assist in A/B testing, turn to Figure 7 which shows a method for a server to gather user-response data in an A/B testing scenario. The application 112 is sent to a first set of end-user devices 100 in step 700 along with instructions for gathering user-response data. (In some embodiments, a separate utility may be developed for gathering user-response data, and this utility need not be sent along with the application 112 being tested.) (In some embodiments, the application 112 is not sent but is otherwise made available to the end-user devices 100.) In the method of Figure 7, the design or function choice being tested is embodied in a first override (version "A"), sent along with the application 112. For example, the override can turn the backgrounds behind all text messages produced by the application 112 into a particular shade of blue.

[0063] Figure 8 shows what can happen on the end-user device 100. In step 800, the end-user device 100 receives the application 112 along with the override (and, potentially, the instructions for gathering user-response data). The override is applied to the stream of graphics commands produced by the application 112 in step 802. User-response data (which may be explicitly queried for or may be implicitly derived from behavioral observations of the user) are gathered in step 804 and returned in step 806.

[0064] Returning to Figure 7, the user-response data generated by the end-user device 100 using the method of Figure 8 are received by the server in step 702.

[0065] Steps 700 and 702 are repeated as steps 704 and 706 but with a different override (the "B" version that, for example, turns the backgrounds behind all text messages produced by the application 112 into a shade of red) and, possibly, with a

different set of end-user devices. This is illustrated at the first time increment of Figure 9: The application 112 is sent out with override A in 900 and with override B in 902.

[0066] The user responses to the "A" and "B" versions are compared to see which version is "better" in some sense (more popular, easier to use, maybe more efficient of computing resources). This is shown by 904 of Figure 9 where override B is preferred.

[0067] In optional step 708, the testing continues to refine the outcome by sending new overrides that embody new choices. In the case of Figure 9, the previous winner override B is sent out again (906) to be tested against a new override C (908).

[0068] When testing is complete, and a final choice is selected for implementation, that implementation may be coded into the application 112 itself or, per optional step 710, the final implementation may be left as an override shipped with the application 112. In Figure 9, override C is the final winner (910).

[0069] In view of the many possible embodiments to which the principles of the present discussion may be applied, it should be recognized that the embodiments described herein with respect to the drawing figures are meant to be illustrative only and should not be taken as limiting the scope of the claims. Therefore, the techniques as described herein contemplate all such embodiments as may come within the scope of the following claims and equivalents thereof.

CLAIMS

We claim:

1. A method on a server (100) for gathering user-response data, the method comprising:

sending (700), by the server (100) to a first set of end-user devices (100), a first override, the first override comprising instructions for at least one graphics modification;

sending (700), by the server (100) to the first set of end-user devices (100), first instructions for gathering user-response data;

receiving (702), by the server (100) from at least some of the first set of end-user devices (100), first user-response data;

sending (704), by the server (100) to a second set of end-user devices (100), a second override, the second set of end-user devices (100) comprising at least one end-user device (100) not in the first set of end-user devices (100);

sending (704), by the server (100) to the second set of end-user devices (100), second instructions for gathering user-response data; and

receiving (706), by the server (100) from at least some of the second set of end-user devices (100), second user-response data.

2. The method of claim 1 wherein the first override comprises instructions for a single graphics modification.

3. The method of claim 1 wherein the first override comprises instructions for a graphics modification selected from the group consisting of: an addition of a new graphics command, a deletion of a graphics command, an amendment to a graphics command, and a re-ordering of graphics commands.

4. The method of claim 1 further comprising:

sending, to a third set of end-user devices, a third override, the third override comprising instructions for at least one graphics modification, the third override based, at least in part, on the received first and second user-response data;

sending, to the third set of end-user devices, third instructions for gathering user-response data; and

receiving, from at least some of the third set of end-user devices, third user-response data.

5. The method of claim 1 further comprising:

sending, to a third set of end-user devices, a third override, the third override comprising instructions for at least one graphics modification, the third override based, at least in part, on the received first and second user-response data.

6. A server (100) configured for gathering user-response data, the server (100) comprising:

a communications interface (126) configured for sending (700), to a first set of end-user devices, a first override, the first override comprising instructions for at least one graphics modification; and

a processor (102) operatively connected to the communications interface (126) and configured for:

sending (700), via the communications interface (126) to the first set of end-user devices (100), first instructions for gathering user-response data;

receiving (702), via the communications interface (126) from at least some of the first set of end-user devices (100), first user-response data;

sending (704), via the communications interface (126) to a second set of end-user devices (100), a second override, the second set of end-user devices (100) comprising at least one end-user device (100) not in the first set of end-user devices (100);

sending (704), via the communications interface (126) to the second set of end-user devices (100), second instructions for gathering user-response data; and

receiving (706), via the communications interface (126) from at least some of the second set of end-user devices (100), second user-response data.

7. A method on an end-user device (100) for gathering user-response data, the method comprising:

receiving (800), by the end-user device (100), an override;

receiving (800), by the end-user device (100), instructions for gathering user-response data;

applying (802), by the end-user device (100), the override, wherein the override comprises instructions for at least one graphics modification;

performing (804), by the end-user device (100), the instructions to gather user-response data, the gathered user-data produced, at least in part, in response to the override; and

sending (806), by the end-user device (100), the gathered user-response data.

8. An end-user device (100) configured for gathering user-response data, the end-user device (100) comprising:

a communications interface (126) configured for receiving (802) an override; and

a processor (102) operatively connected to the communications interface (126) and configured for:

receiving (800), via the communications interface (126), instructions for gathering user-response data;

applying (802) the override, the override comprising instructions for at least one graphics modification;

performing (804) the instructions to gather user-response data, the gathered user-data produced, at least in part, in response to the override; and

sending (806), via the communications interface (126), the gathered user-response data.

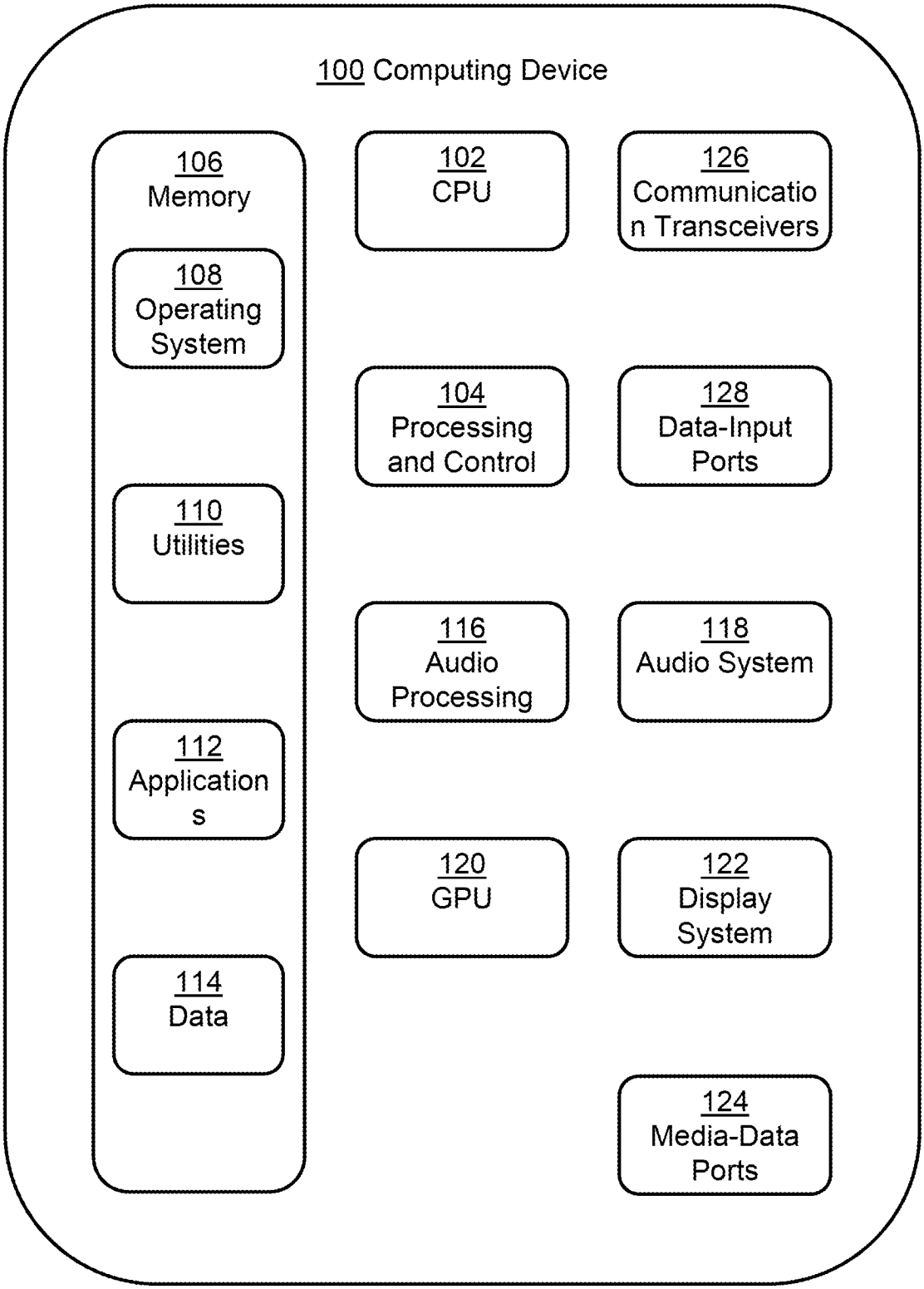

100 Computing Device
106 Memory
108 Operating System
110 Utilities
112 Applications
114 Data
102 CPU
104 Processing and Control
116 Audio Processing
120 GPU
126 Communication Transceivers
128 Data-Input Ports
118 Audio System
122 Display System
124 Media-Data Ports

FIG. 1

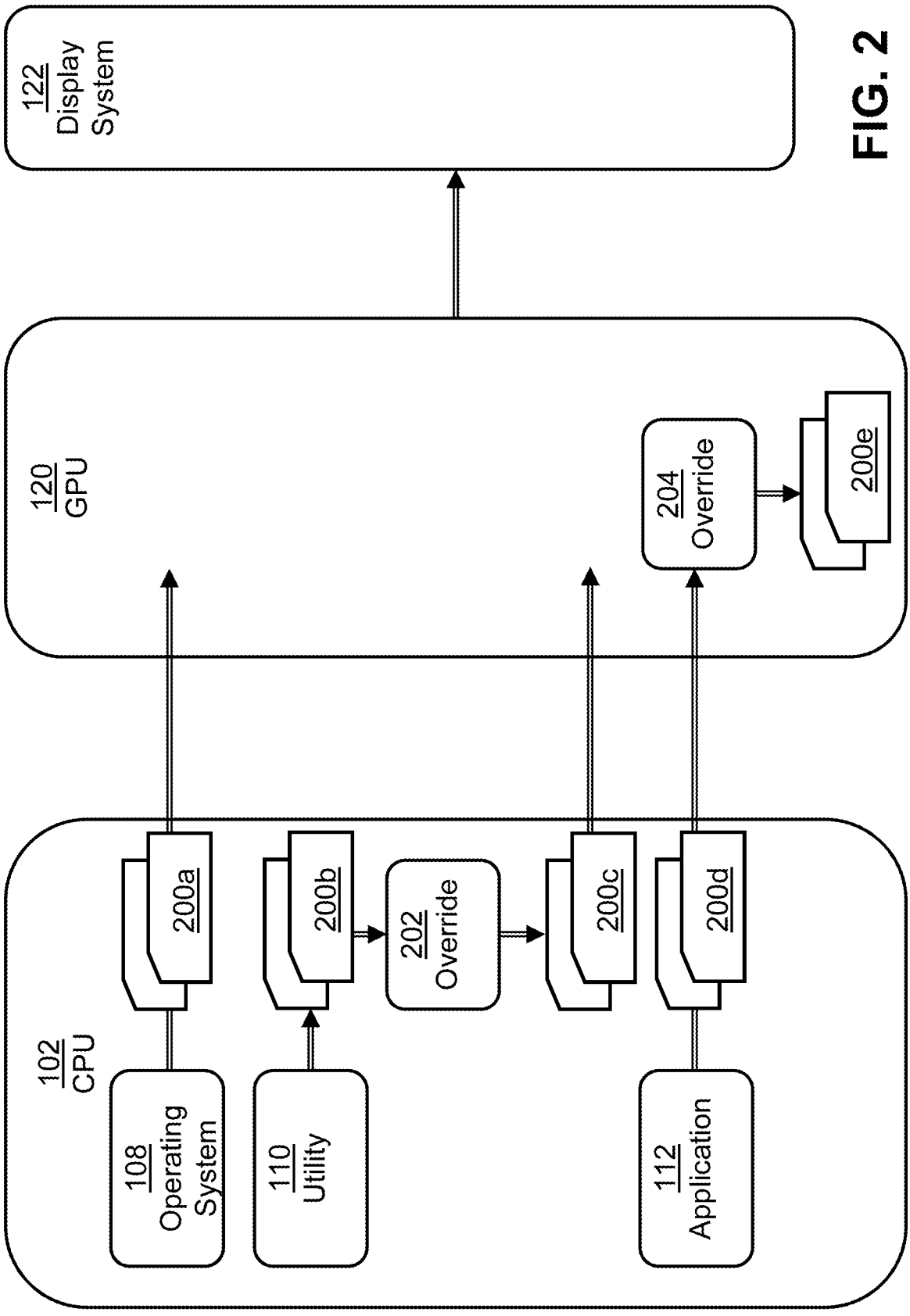

122
Display System
120
GPU
204
Override
200e
102
CPU
200a
200b
202
Override
200c
200d
108
Operating System
110
Utility
112
Application

FIG. 2

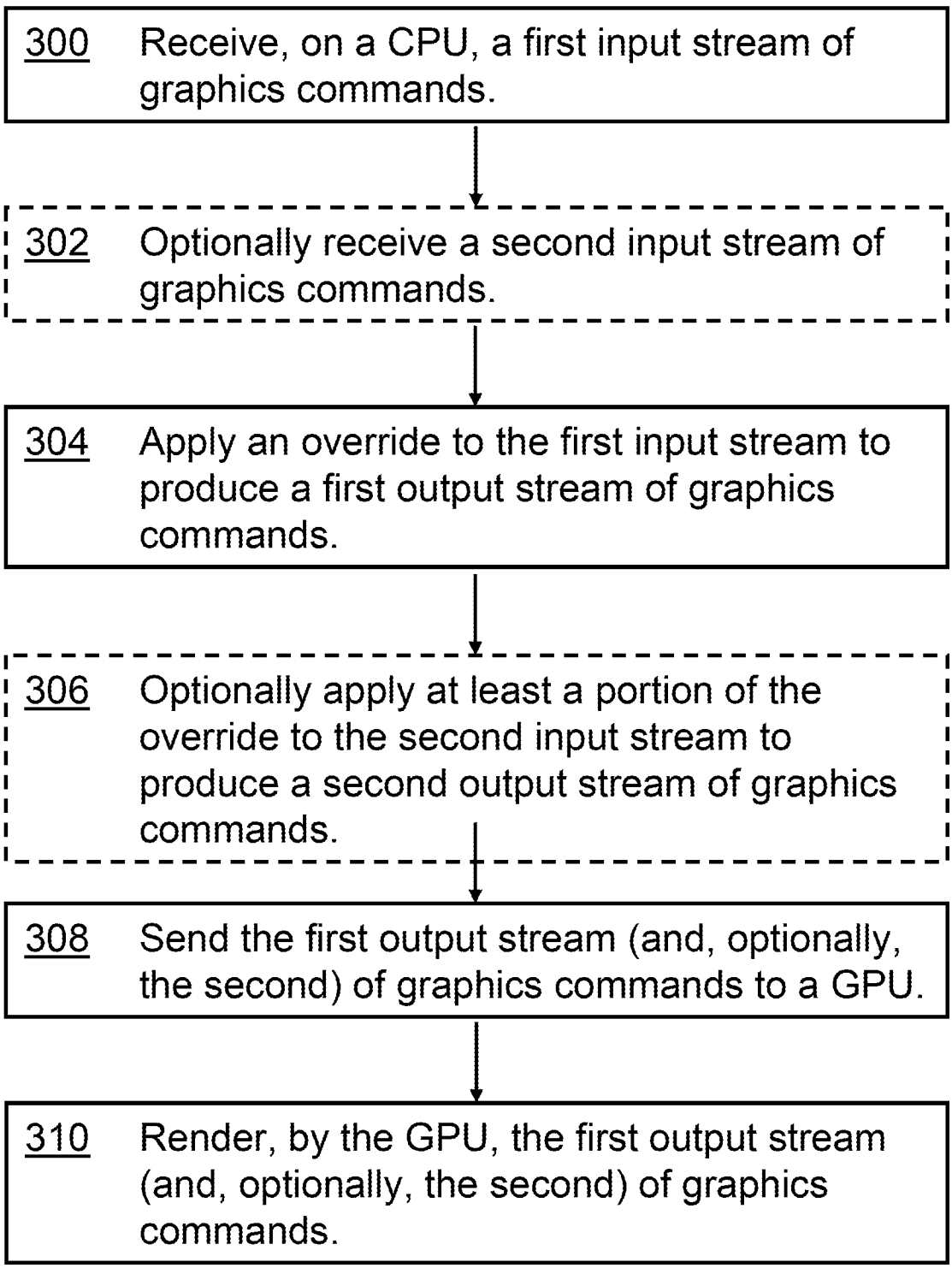

300 Receive, on a CPU, a first input stream of graphics commands.
302 Optionally receive a second input stream of graphics commands.
304 Apply an override to the first input stream to produce a first output stream of graphics commands.
306 Optionally apply at least a portion of the override to the second input stream to produce a second output stream of graphics commands.
308 Send the first output stream (and, optionally, the second) of graphics commands to a GPU.
310 Render, by the GPU, the first output stream (and, optionally, the second) of graphics commands.

FIG. 3

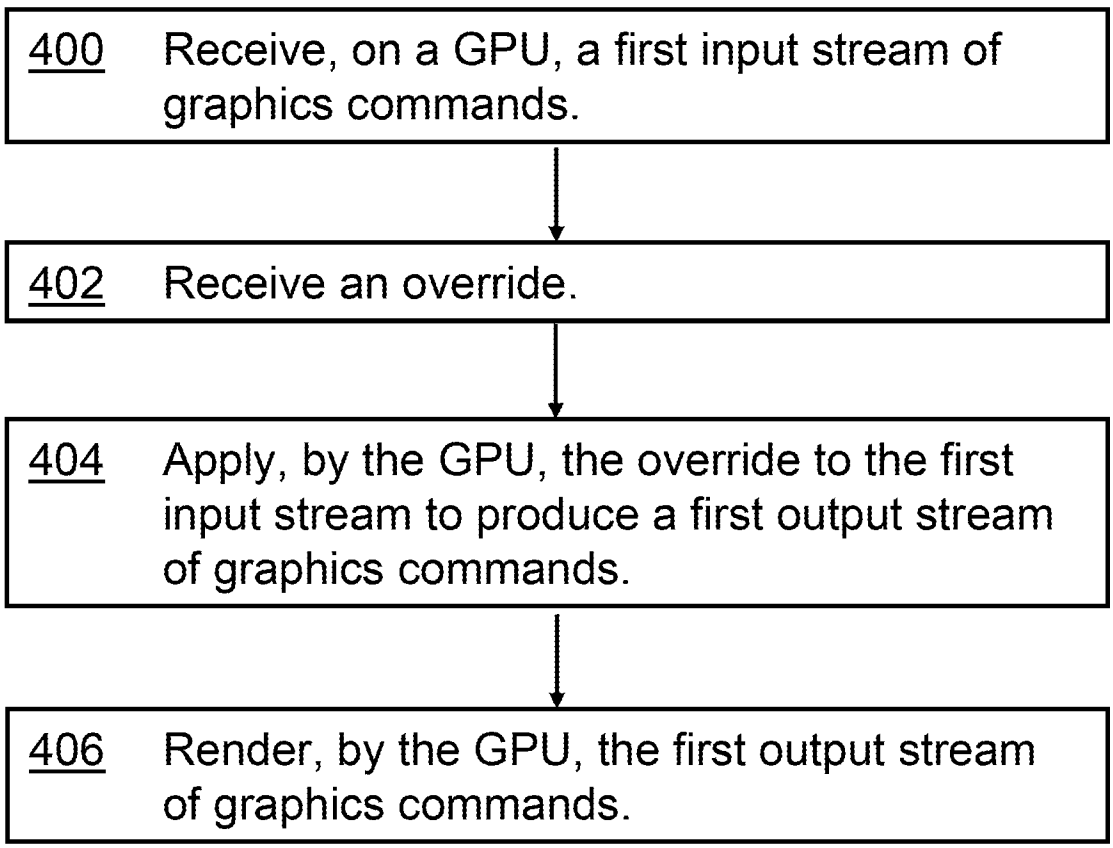

400 Receive, on a GPU, a first input stream of graphics commands.
402 Receive an override.
404 Apply, by the GPU, the override to the first input stream to produce a first output stream of graphics commands.
406 Render, by the GPU, the first output stream of graphics commands.

FIG. 4

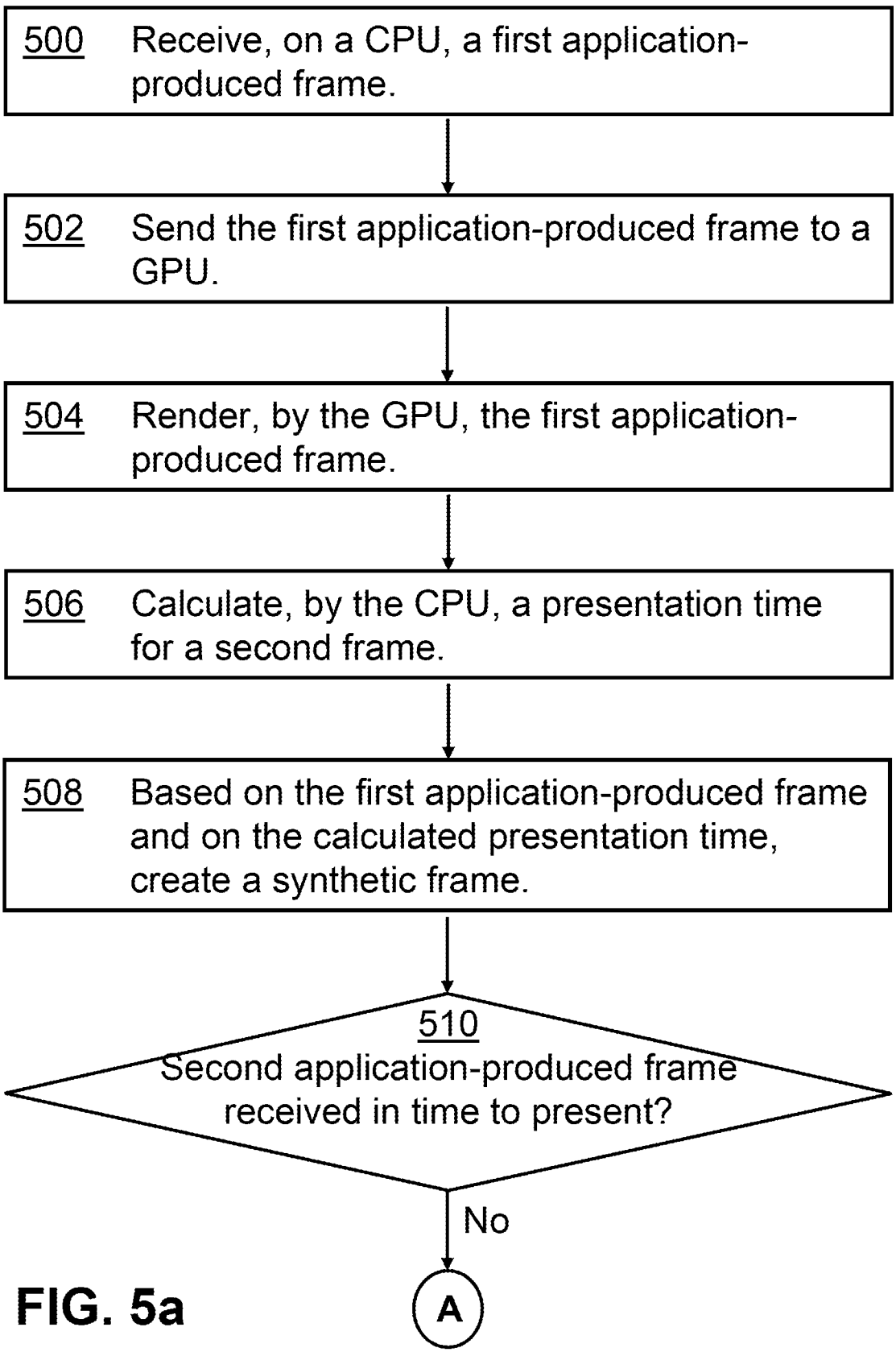

500 Receive, on a CPU, a first application-produced frame.
502 Send the first application-produced frame to a GPU.
504 Render, by the GPU, the first application-produced frame.
506 Calculate, by the CPU, a presentation time for a second frame.
508 Based on the first application-produced frame and on the calculated presentation time, create a synthetic frame.
510
Second application-produced frame received in time to present?
No
A

FIG. 5a

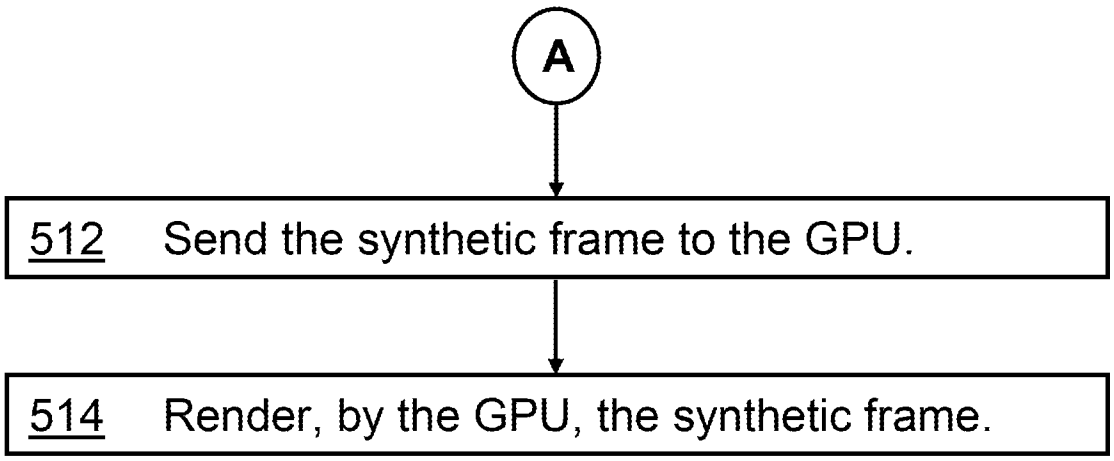

A
512 Send the synthetic frame to the GPU.
514 Render, by the GPU, the synthetic frame.

FIG. 5b

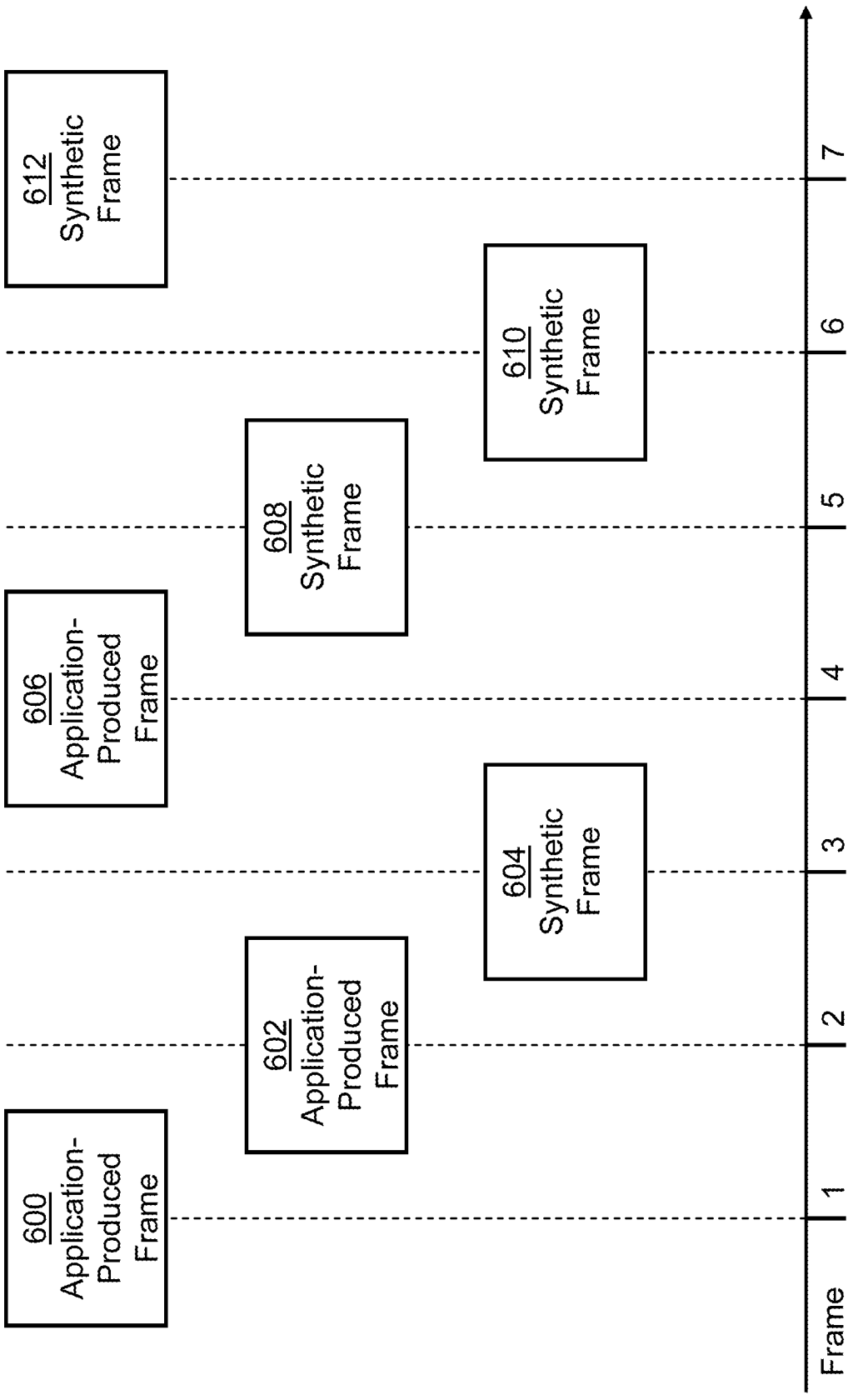

600 Application-Produced Frame
602 Application-Produced Frame
604 Synthetic Frame
606 Application-Produced Frame
608 Synthetic Frame
610 Synthetic Frame
612 Synthetic Frame
Frame Time:
1
2
3
4
5
6
7

FIG. 6

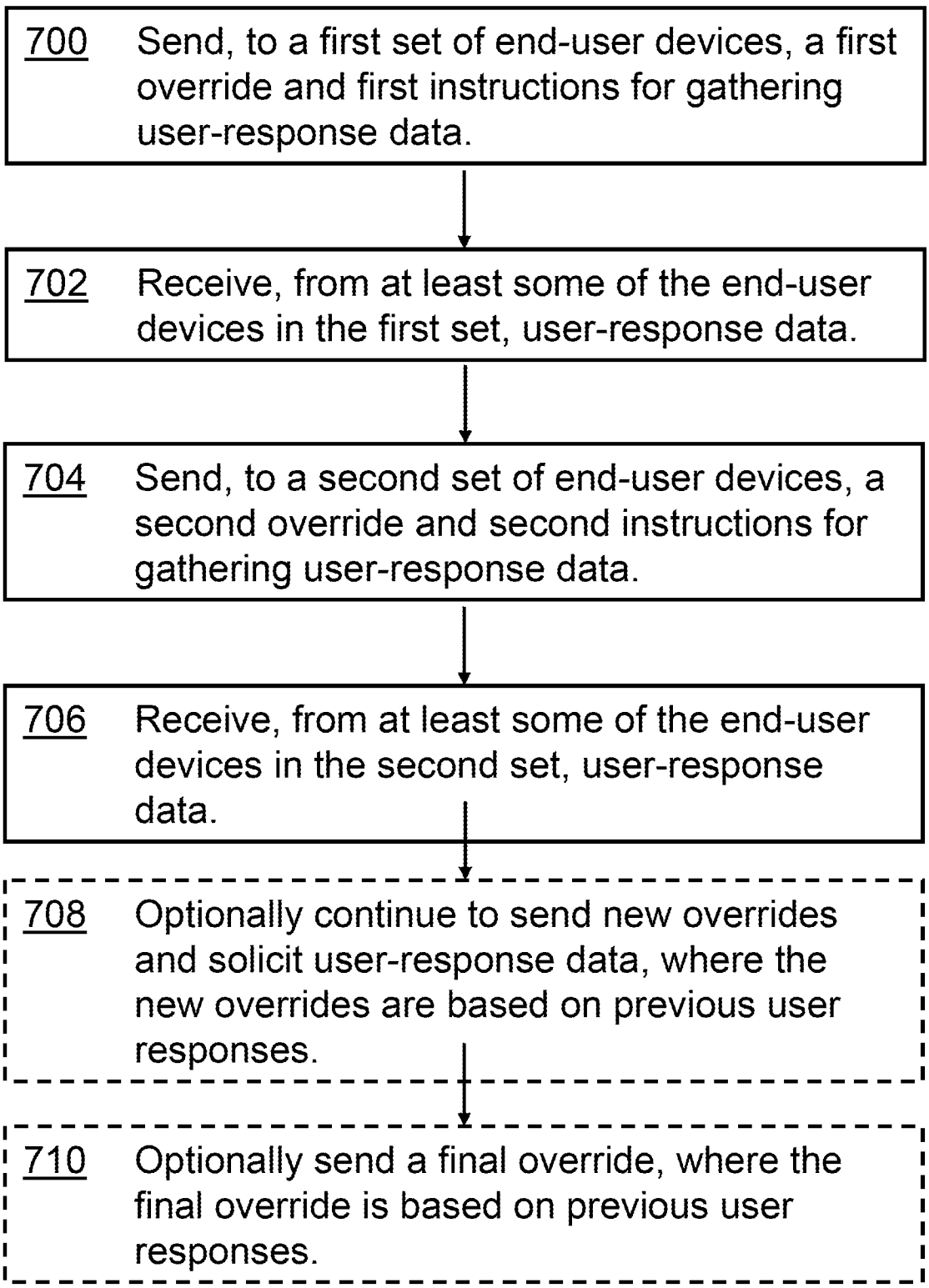

700 Send, to a first set of end-user devices, a first override and first instructions for gathering user-response data.
702 Receive, from at least some of the end-user devices in the first set, user-response data.
704 Send, to a second set of end-user devices, a second override and second instructions for gathering user-response data.
706 Receive, from at least some of the end-user devices in the second set, user-response data.
708 Optionally continue to send new overrides and solicit user-response data, where the new overrides are based on previous user responses.
710 Optionally send a final override, where the final override is based on previous user responses.

FIG. 7

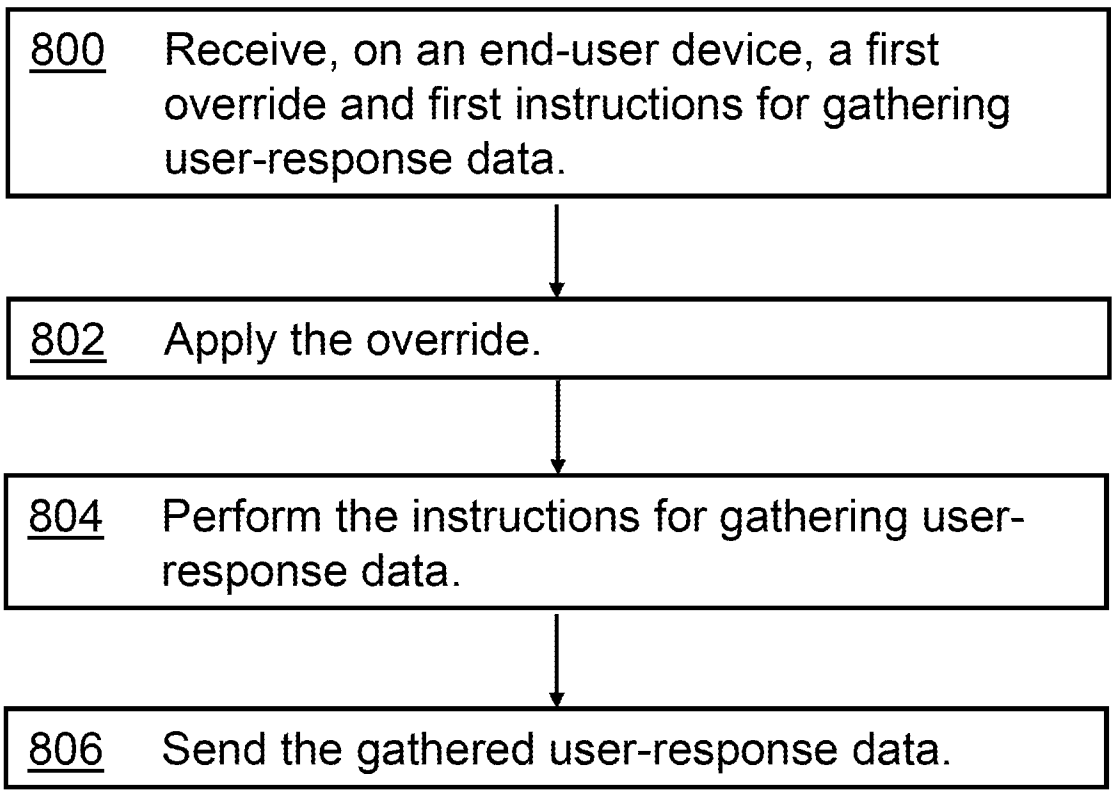

800 Receive, on an end-user device, a first override and first instructions for gathering user-response data.
802 Apply the override.
804 Perform the instructions for gathering user-response data.
806 Send the gathered user-response data.

FIG. 8

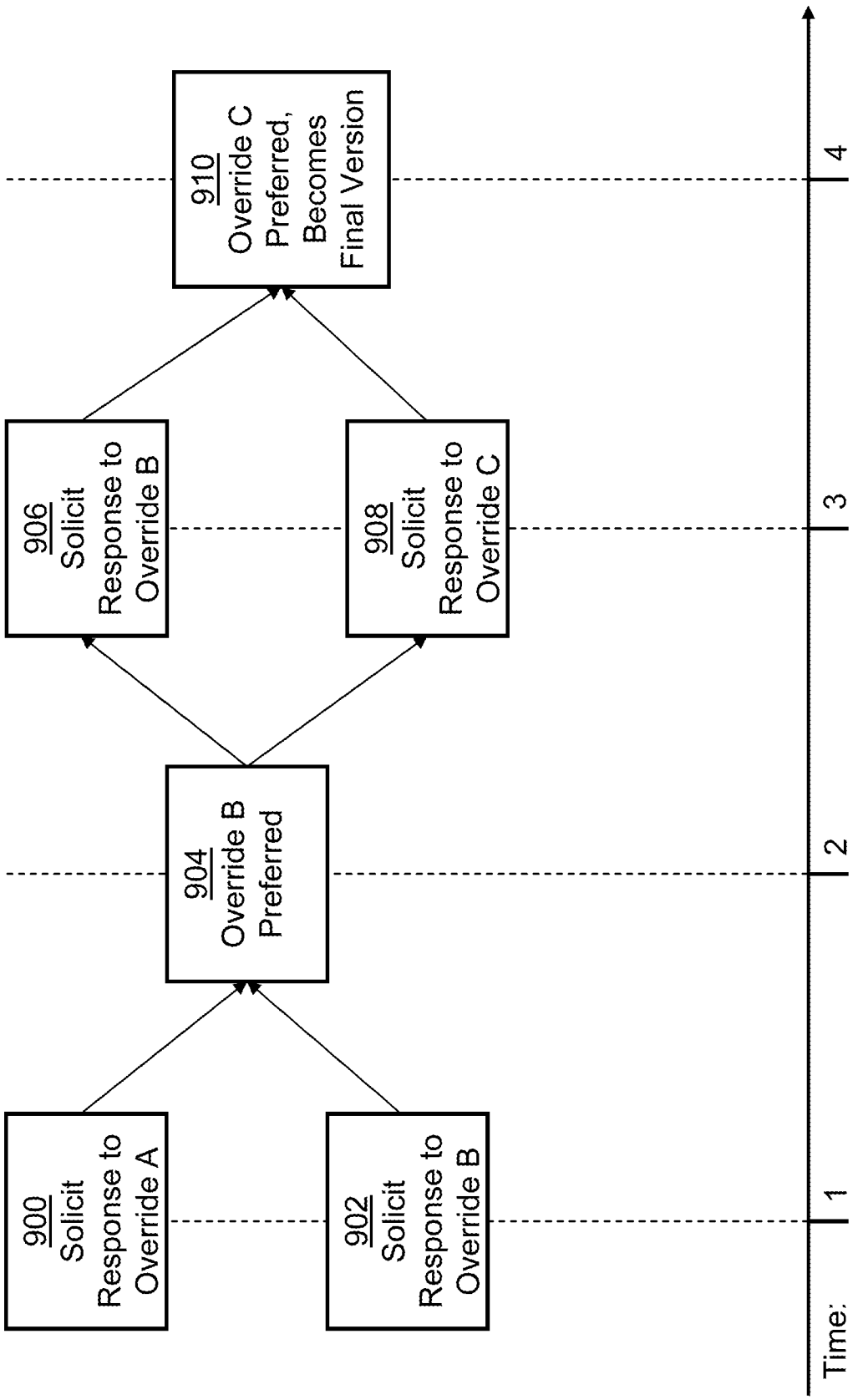

900 Solicit Response to Override A
902 Solicit Response to Override B
904 Override B Preferred
906 Solicit Response to Override B
908 Solicit Response to Override C
910 Override C Preferred, Becomes Final Version
Time:
1
2
3
4

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/071913

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F3/14 G06F11/34
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F G09G

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2012/081376 A1 (SOWERBY ANDREW M [US] ET AL) 5 April 2012 (2012-04-05) figures 1-3,8 paragraphs [0003], [0004], [0010] - [0012], [0035], [0102] - [0105], [0108] - [0110], [0114]	1-8
X	US 2011/018884 A1 (RITTS JAMES P [US] ET AL) 27 January 2011 (2011-01-27) figures 1-5,8 paragraphs [0026] - [0029], [0038] - [0045], [0049] - [0053], [0061], [0080], [0082], [0099], [0100]	1-8

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

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 February 2014	21/02/2014

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

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2013/071913

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012081376 A1	05-04-2012	NONE	
US 2011018884 A1	27-01-2011	CN 102449665 A	09-05-2012
		EP 2438576 A1	11-04-2012
		JP 2012529120 A	15-11-2012
		KR 20120026116 A	16-03-2012
		TW 201108152 A	01-03-2011
		US 2011018884 A1	27-01-2011
		WO 2010141612 A1	09-12-2010