



- (51) International Patent Classification:  
G06T 1/00 (2006.01) G06T 9/00 (2006.01)
- (21) International Application Number:  
PCT/US2013/036182
- (22) International Filing Date:  
11 April 2013 (11.04.2013)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:  
13/445,104 12 April 2012 (12.04.2012) US
- (71) Applicant: ACTIVEVIDEO NETWORKS, INC.  
[US/US]; 333 W. San Carlos Street, Suite 400, San Jose, CA 95110 (US).
- (72) Inventors: BROCKMANN, Ronald, A.; Bilstraat 140, NL-3572 BL Utrecht (NL). DEV, Anuj; Knsm-Iaan 566, NL-1019 LP Amsterdam (NL). GORTER, Onne; Ruiterweg 70-1, NL-1211 KX Hilversum (NL). HIDDINK, Gerrit; Beukstraat 54, NL-3581 XH Utrecht (NL). HOEBEN, Maarten; Muurhuizen 199, NL-3811 EH Amersfoort (NL).
- (74) Agents: SUNSTEIN, Bruce, D. et al.; Sunstein Kann Murphy & Timbers LLP, 125 Summer Street, Boston, MA 02110 (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, 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, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, ML, MR, NE, SN, TD, TG).

Published:  
— with international search report (Art. 21(3))

(54) Title: GRAPHICAL APPLICATION INTEGRATION WITH MPEG OBJECTS

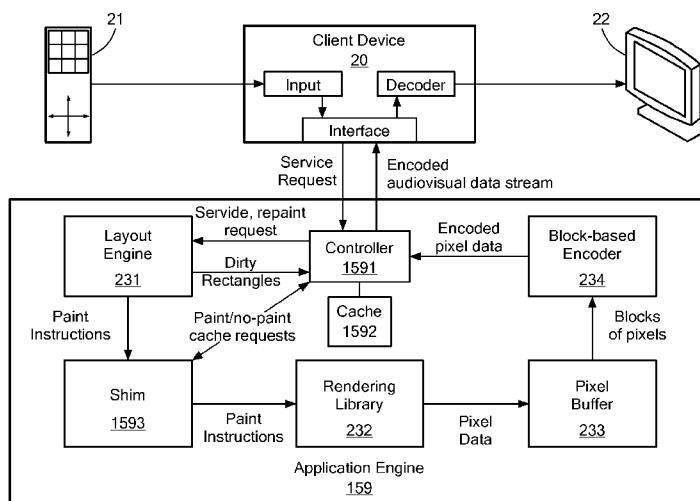


FIG. 3

(57) Abstract: System and methods are provided to cache encoded graphical objects that may be subsequently combined with other encoded video data to form a data stream decodable by a client device according to a format specification. Paint instructions relating to a graphical object are sent from a layout engine to a rendering library. A shim intercepts these instructions and determines whether the graphical object already has been rendered and encoded. If so, a cached copy of the object is transmitted to the client device. If not, the shim transparently passes the instructions to the rendering library, and the object is rendered, encoded, and cached. Hash values are used for efficiency. Methods are disclosed to detect and cache animations, and to cut and splice cached objects into encoded video data.

WO 2013/155310 A1

## **Graphical Application Integration with MPEG Objects**

### **Technical Field**

[0001] The present invention relates to computer graphics display memory systems and methods, and more particularly to providing a graphical user interface having cached graphical elements.

### **Background Art**

[0002] Content providers are experiencing a growth in demand for interactive applications, such as interactive menus, games, web browsing, and the like. Each such interactive application must provide an output that is tailored to the individual requesting it. This is done by establishing a session between the content provider and a client device over a data network, for example the Internet or a cable television system. Furthermore, the audiovisual data for each application is typically encoded or compressed according to an encoding scheme, such as MPEG, to reduce the amount of data that must be transferred. However, encoding audiovisual data for transmission over such a data network is computationally expensive. As the number of requests for interactive sessions grows, it becomes problematic to both render and encode the output of a large number of application sessions, each output destined for a different viewer.

[0003] It is known in the art to reuse audiovisual content by caching it. In this way, a frame of video content may be produced once, and sent to as many client devices as required. However, many applications generate reusable images that are smaller than a full frame of video. For example, a menuing application may generate a pulsating menu button animation, or a video game may draw a spaceship image at nearly any location on the screen. Prior art systems must re-render and re-encode these sub-frame images for each video frame produced. Caching mechanisms cannot be used, because the encoding process often uses a state-based data compression system that does not permit insertion of images into an existing

data stream. As rendering and encoding are computationally expensive operations, prior art systems require a large hardware and software investment to keep up with demand.

### **Summary of Illustrated Embodiments**

[0004] To solve the aforementioned problems, various embodiments of the present invention permit caching of encoded or compressed images that can be composited together with an audiovisual data source. In particular, for each application that defines a graphical user interface, various embodiments insert a small software hook, or shim, between layers in the application execution environment that intercepts rendering commands and determines whether the image to be rendered is already cached in an encoded state. If so, the encoded image is inserted into the video without being completely decoded and re-encoded. Slice cutting and slice linking techniques as separately disclosed herein may be used to accomplish such an insertion.

[0005] Thus, in a first embodiment there is given a method of providing an image to a client device from an application execution environment having a layout engine that assembles graphical components into a graphical user interface screen for a graphical application, and a rendering library that renders graphical components into pixels. The method includes receiving, from the layout engine, one or more paint instructions having parameters that pertain to a given graphical object. Next, the method requires computing a hash value based on the received one or more paint instructions. There are two paths, depending on whether the hash value is contained within a cache memory. If so, the method calls for retrieving, from the cache, encoded audiovisual data that are uniquely associated with the hash value, and transmitting the retrieved audiovisual data to the client device. If not, the method requires several more steps. The first such step is forwarding the received one or more paint instructions to the rendering library for rendering the graphical object into pixels according to the paint instruction. The second such step is encoding the rendered pixels into encoded audiovisual data. The third such step is storing the hash value and the encoded audiovisual data in the cache, whereby the hash value and the encoded audiovisual data are uniquely associated. Finally, the fourth such step is transmitting the encoded audiovisual data to the client device. Determining that the hash value is contained within the

cache may be done by comparing the hash value to a stored hash value of a cached image that forms part of an animation.

[0006] The client device may be a television, a television set-top box, a tablet computer, a laptop computer, a desktop computer, or a smartphone. The graphical application may be, for example, a web browser or a menu interface.

[0007] Encoding may include dividing the screen into blocks of pixels. In one such related embodiment, the method may be extended, after receiving the painting data and before computing the hash value, by determining the smallest rectangle consisting of whole blocks of pixels that surrounds the at least one graphical object; requesting that the layout engine repaint the smallest surrounding rectangle; and receiving, from the layout engine, second painting data that include at least one paint instruction having parameters that reflect the smallest surrounding rectangle, wherein computing the hash value is based on the second painting data.

[0008] In a separate related embodiment, the method may be extended by determining the smallest rectangle consisting of whole blocks of pixels that surrounds the at least one graphical object; copying current image data into a pixel buffer having the size and shape of the smallest surrounding rectangle; and requesting that the rendering library render the graphical object into the pixel buffer according to the painting data, wherein computing the hash value is based on the pixel data in the pixel buffer.

[0009] Sometimes an interactive application will provide a repeating sequence of images that forms an animation, and images in the sequence may benefit from other optimizations. For example, regarding these sequences of images as an animation allows motion detection to be performed, resulting in much more efficient inter-encoding (e.g., producing P-frames and B-frames). This increase in efficiency may manifest as, for example, a lower bandwidth required to transmit a video that includes the animation, or a higher quality for the same bandwidth.

[0010] Thus, in a second embodiment there is provided a method of transmitting, to a client device, images that comprise an animation. The method requires first receiving a current image into a computing processor. As with the first method embodiment, there are two paths. When the current image is identical to a previously rendered image, the previously rendered image being uniquely associated with an encoded image in a cache

memory, the method concludes by transmitting to the client device the cached, encoded image without encoding the current image. However, when the current image is not identical to a previously rendered image, but shares at least a given minimum percentage of its pixels with a given, previously rendered image, the method continues with a number of additional steps. The first such step is identifying the current image and the given, previously rendered image as belonging to a common animation. The second such step is encoding the current image according to a predictive encoding scheme. The third such step is storing the encoded current image in the cache memory. The fourth such step is transmitting to the client device the encoded current image.

[0011] The predictive encoding scheme may be an MPEG encoding scheme. The previously rendered image may not have been rendered immediately previously to the current image, but may be an image rendered earlier. The previously rendered image may be uniquely associated with a predictively encoded image in the cache memory. This second method may be extended by computing a hash value for each unique chain of images that forms an animation, the hash value being a function of all images in the chain of images and a screen displacement between two consecutive images in the chain.

[0012] On occasion, it is more efficient to form a row of encoded data by combining currently-displayed visual data with newly rendered rectangles or animations than it is to re-render and re-encode an entire screen. Thus, it is necessary to develop methods for cutting rows of the currently-displayed data into slices, and methods for combining slices of data together again to form whole rows.

[0013] Therefore, in a third embodiment there is provided a method of forming two encoded slices from data comprising a given encoded slice, each encoded slice comprising a sequence of macroblocks that are encoded according to a variable length code. This method includes locating, in the given slice, a location of a macroblock. Then, the method requires altering a DC luma value or a DC chroma value of the located macroblock without fully decoding the macroblock according to the variable length code. The first formed slice consists of the data of the given slice up to but not including the altered macroblock, and the second formed slice consists of the encoded macroblock and any subsequent encoded macroblocks in the given slice. Altering the DC luma value or the DC chroma value may be performed through a bit-shifting operation.

[0014] Further, in a fourth embodiment there is provided a method of combining a first encoded slice and a second encoded slice to form a third encoded slice, each encoded slice comprising a sequence of macroblocks that are encoded according to a variable length code. The method first requires altering a DC luma value or a DC chroma value in the first macroblock of the second slice without fully decoding the macroblock according to the variable length code. The method ends by concatenating the data of the first slice with the altered macroblock and the undecoded data of the second slice to form the third encoded slice. As before, altering the DC luma value or the DC chroma value may be performed through a bit-shifting operation.

[0015] It is contemplated that the invention may be embodied in a tangible medium on which is stored non-transitory computer program code for performing any of the above methods.

[0016] It is also contemplated that the invention may be embodied in a system for providing an image to a client device from an application execution environment having a layout engine that assembles graphical components into a graphical user interface screen for a graphical application, and a rendering library that renders graphical components into pixels. The system may include a memory. The system may also include a shim comprising hardware or a combination of hardware and software that is configured to: receive, from the layout engine, one or more paint instructions having parameters that pertain to a given graphical object, compute a hash value based on the received one or more paint instructions, and, when the hash value is not contained within the memory, forward the received one or more paint instructions to the rendering library for rendering the graphical object into pixels according to the one or more paint instructions. The system may also include a controller comprising hardware or a combination of hardware and software that is configured to:

retrieve, from the memory, encoded audiovisual data that are uniquely associated with the hash value, and transmit the retrieved audiovisual data to the client device when the hash value is contained within the memory; and transmit, to the client device, encoded audiovisual data comprising a rendering of the graphical object into pixels according to the received one or more paint instructions when the hash value is not contained within the memory.

[0017] The client device may be a television, a television set-top box, a tablet computer, a laptop computer, a desktop computer, or a smartphone. The graphical application may be, for example, a web browser or a menu interface. The memory may store a sequence of images that collectively form an animation, in which case the controller is further configured to determine that the hash value is contained within the cache by comparing the hash value to a stored hash value of a cached image that forms part of the animation. The audiovisual data may be encoded according to an MPEG encoding scheme.

[0018] The system may also include a block-based encoder that is configured to form two encoded MPEG slices from data comprising a given encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks. Forming the slices may be performed by locating, in the given MPEG slice, a location of a macroblock that is encoded according to a variable length code; then decoding the encoded macroblock according to the variable length code; then altering a DC luma value in the decoded macroblock; and finally encoding the altered macroblock according to the variable length code, wherein the first formed MPEG slice consists of the data of the given MPEG slice up to but not including the encoded macroblock, and the second formed MPEG slice consists of the encoded macroblock and any subsequent encoded macroblocks in the given MPEG slice.

[0019] The system may also include a block-based encoder that is configured to combine a first encoded MPEG slice and a second encoded MPEG slice to form a third encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks. Combining the slices may be performed by decoding the first macroblock of the second slice according to a variable length code; then altering a DC luma value in the decoded macroblock; then encoding the altered macroblock according to the variable length code; and finally concatenating the data of the first slice with the encoded macroblock and the undecoded data of the second slice to form the third slice.

### **Brief Description of the Drawings**

[0020] The foregoing features of embodiments will be more readily understood by reference to the following detailed description, taken with reference to the accompanying drawings, in which:

[0021] Fig. 1 is a schematic diagram of a typical system in which various embodiments of the invention may be used;

[0022] Fig. 2 is a block diagram showing functional modules and data flow in a prior art web browser system;

[0023] Fig. 3 is a block diagram showing functional modules and data flow in accordance with an embodiment of the invention;

[0024] Fig. 4 is a flowchart showing a method of generating an initial screen for a graphical user interface in accordance with an embodiment of the invention;

[0025] Figs. 5A-5C collectively comprise a flowchart showing a method of generating a screen update in accordance with the embodiment of Fig. 4;

[0026] Figs. 6A-6D show an exemplary screen area that is being updated at various stages of the methods of Figs. 4 and 5;

[0027] Fig. 6E shows a pixel buffer relating to the exemplary screen area of Fig. 6;

[0028] Fig. 7 is a flowchart showing a method of detecting an animation in accordance with an embodiment of the invention;

[0029] Figs. 8A-8C show a “rolling update” of several rows of macroblocks;

[0030] Figs. 9A-9E illustrate the concept of slice cutting and slice linking, as used in accordance with an embodiment of the invention;

[0031] Fig. 10 is a flowchart showing a method of cutting an MPEG slice in accordance with an embodiment of the invention;

[0032] Figs. 11A-11D show the effects of slice cutting at the level of slice data;

[0033] Fig. 12 is a flowchart showing a method of linking MPEG slices in accordance with an embodiment of the invention; and

[0034] Figs. 13A-13D show the effects of slice linking at the level of slice data.

### **Detailed Description of Specific Embodiments**

[0035] Definitions. As used in this description and the accompanying claims, the following terms shall have the meanings indicated, unless the context otherwise requires:

[0036] The term “application” refers to an executable program, or a listing of instructions for execution, that defines a graphical user interface (“GUI”) for display on a display device. An application may be written in a declarative language such as HTML or



CSS, a procedural language such as C, JavaScript, or Perl, any other computer programming language, or a combination of languages.

[0037] A “rectangle” is a rectangular area on a screen of the display device. The screen area may in fact reside within a window in a windowed user interface.

[0038] A rectangle is “clean” if its contents match what is currently being output to the display device, and “dirty” if its contents do not match what is currently being output.

[0039] A “layout engine” is a computing service that is used to convert a document into graphical objects placed on a display screen. For example, Trident, WebKit, and Gecko are software layout engines that convert web pages into a collection of graphical objects (text strings, images, and so on) arranged, according to various instructions, within a page display area of a web browser. The instructions may be static, as in the case of parts of HTML, or dynamic, as in the case of JavaScript or other scripting languages, and the instructions may change as a function of user input. Trident is developed by Microsoft Corporation and used by the Internet Explorer web browser; WebKit is developed by a consortium including Apple, Nokia, Google and others, and is used by the Google Chrome and Apple Safari web browsers; Gecko is developed by the Mozilla Foundation, and is used by the Firefox web browser.

[0040] A “rendering library” is a computing service that is used by a layout engine to convert graphical objects into images. Graphical objects include, without limitation, alphanumeric symbols, shapes such as circles and rectangles, and images defined according to an image format such as GIF or JPEG. For example, Cairo is a software rendering library that converts two-dimensional objects defined using vector graphics into either pixel data or into drawing commands for underlying graphical systems such as X Windows, the Windows 32-bit graphics device interface, or OpenGL. Cairo is developed by Carl Worth of Intel Corporation, Behdad Esfahbod of Google (Waterloo, Canada), and a host of others.

[0041] A “pixel buffer” is a data buffer used to temporarily store the pixel data of a screen rectangle.

[0042] A “pixel hash” is a hash value that is calculated over all pixels in a pixel buffer.

[0043] A “repaint request” is a request from a controller to a layout engine to repaint the contents of a rectangle for output. Repaint requests may be used to “clean” a dirty rectangle.

[0044] A “graphical object” is a collection of data that permits a shape to be drawn on a display. For example, a graphical object that represents a square may include data pertaining to coordinates of the square’s vertices, a line thickness, a line color, and so on. A graphical object that represents a text character may include data pertaining to a font name, a letter height, a color, a font weight, and so on. A graphical object may contain other graphical objects; for example, a text string may include a number of letters.

[0045] A “paint instruction” is an instruction from the layout engine to a rendering library to generate pixel data, in a pixel buffer, that relates to a given graphical object.

[0046] A “paint hash” is a hash value that is calculated as a function of a sequence of paint instructions that are generated to repaint a rectangle’s content, including their parameters (or certain appropriately chosen representations of their parameters).

[0047] An “MPEG fragment” is one or more MPEG-encoded macroblocks, as disclosed in U.S. patent application 12/443,571, filed October 1, 2007, the contents of which are incorporated by reference in their entirety.

[0048] “Audiovisual data” are data that represent audio, video, or a combination of audio and video.

[0049] An “animation” is a repeating sequence of individual images.

[0050] A “slice”, in the context of video encoding and especially in the context of a H.264 / MPEG-4 encoding format, is a group of one or more horizontally contiguous macroblocks, in raster order, that can be encoded independently from other slices according to the encoding format.

[0051] Fig. 1 is a schematic diagram of a typical system in which various embodiments of the invention may be used. These embodiments transmit streaming audiovisual data to a variety of client devices for playback, including a smart television, cable set top box, or a desktop computer in house 11, a tablet computer 12, a laptop computer 13, and a smartphone 14. The audiovisual data are typically streamed from an operator headend 15. The operator may obtain content via a public data network, shown here as the

Internet 16, from a content provider, shown here as a web server 17. The operator also may obtain the content from an operator-controlled web server via a private data network.

**[0052]** The operator headend 15 is connected to each of the various client devices via a gateway. Thus, the headend is connected to house 11 through a cable gateway 151, which may be, for example, a cable modem termination system for terminating a cable system 1511. The headend is connected to the tablet computer 12 via a wireless gateway 152, such as an antenna, that transmits and receives on a wireless data network 1521. The headend is connected to the laptop computer 13 via a wired network gateway 153, such as a router, that uses a wired data network 1531. And the headend is connected to the smartphone 14 via a cellular network gateway 154 that uses a cellular telephone network 1541. Similarly, the headend is connected to the Internet 16 via a network gateway 155 (which typically includes a firewall, as indicated, to prevent unauthorized access). The headend may be connected to other client devices known in the art using similar, ordinary means.

**[0053]** All of these gateways are connected, typically via one or more firewalls or data routing devices (not shown), to a central headend data network 150. Also connected to the central network are various other useful headend systems, such as an administrative system 156 and media storage server 157. Various embodiments of the invention are particularly directed to the creation and use of transcoders and image scalers 158, and application engine and session manager 159. These functional components are described in more detail in connection with Figs. 3-6 below. The administrative functions 157, media storage 157, transcoders and scalers 158, and application engine and session manager 159 may be implemented in software and/or hardware using general purpose computers or special-purpose computing systems. It will be appreciated that any or all of these components may be implemented in parallel to handle large numbers of concurrent users. Thus, for example, a given headend 15 may execute a plurality of transcoder instances, scaler instances, and/or application engine instances at any given time. Moreover, these instances need not be executed within one physical premises, but may be distributed as required by the service provider.

**[0054]** Transcoders may be used to re-encode data from a first data format (such as a broadcast format or storage format) into a second data format (such as a data streaming format). Scalers may be used to dynamically resize video streams, for example to provide a

“mosaic” of multiple video streams on a single display. An application engine may be used to run an application having a graphical user interface, such as an HTML page or a web browser, in a user session with a particular client device. Such user sessions may be managed by the session manager.

[0055] Typically, a client device forms a data connection to the operator headend and requests a particular interactive service, such as a menuing interface or a web browser. In response, the headend requests a new session from the session manager, and allocates an application engine associated with the requested service. If the particular service requires transcoding or scaling, the session manager will also allocate these resources. The application engine communicates with the client device, and requests transcoding and scaling operations (as well as access to administrative functions 156 such as billing, and stored media 157) to provide an enjoyable interactive experience to a user of the client device. When the service is terminated, either by the headend or the client device, the session manager frees up the allocated resources. In accordance with these processes, many thousands of client devices may be simultaneously supported.

[0056] For purposes of illustration, and not by way of limitation, one service that may be requested is web browsing. Fig. 2 is a block diagram showing functional modules and data flow in a prior art web browser system having a remote browser engine. In this system, a client device 20, such as a cable set top box, is coupled to an input device, such as video keyboard 21, and a display device, such as monitor 22. It will be understood that these components are shown separately for clarity, but they may be integrated into a single form factor, such as a tablet computer or other computing device.

[0057] The input device 21 transmits a request for a web page through the client device 20 to a remote browser 23. The remote browser includes four components: a layout engine 231, one or more rendering libraries 232, a pixel buffer 233, and a block-based streaming data encoder 234. The layout engine receives the request and downloads the linked content. This content must be rendered, and when the layout engine wishes to render a graphical object, such as a text string or an image file, it issues one or more paint instructions to a rendering library 232 using an application programming interface (API) for the library. The rendering library then renders the graphical object into a pixel buffer 233 at a location determined by the layout engine.

[0058] File formats for encoded image data may be recognized by humans using a (e.g. three or four letter) filename extension such as GIF or JPEG. However, often these extensions are incorrect, so the layout engine may resort to reading a “magic number” inside the file itself at industry-standard byte offsets. Such magic numbers are well known in the art, and their careful management across the industry permits unambiguous identification of file formats by the application execution environment. Correct identification of the file format for an image graphical object permits the layout engine to invoke the proper rendering library 232 to draw its encoded data.

[0059] Once the pixel data have been drawn into the pixel buffer 233, the block-based encoder 234 receives blocks of pixels from the buffer and encodes them according to an encoding. Encodings are used to compress the data for transmission, as it is often the case that data transmission capabilities between the remote browser and the client device are limited. One encoding used in the art is the MPEG encoding, although it will be understood that the scope of the invention is not limited only to MPEG. Once the pixel data are encoded, they are transmitted from the remote browser 23 to the client device 20, where they are decoded and displayed on the display 22.

[0060] Interactive behavior typically is controlled from the client device as part of a session established between the client device and the remote browser. Further input received from the client device, such as a repeated key press or a held key on a remote control or a keyboard, causes the layout engine to execute any application logic (e.g., JavaScript). If the application logic requires the screen output to change in response to this interactive input, as it often does, the process may begin again as if a new page request (or update request) were received, thereby causing a modified pixel buffer to be encoded and sent to the client device.

### Screen Updates

[0061] Fig. 3 is a block diagram showing functional modules and data flow in accordance with an embodiment of the invention. As can be seen, the application engine 159 of this embodiment, also referred to as the application execution environment, differs substantially from the remote browser of Fig. 2. Some of the components of the remote browser 23 (i.e., the layout engine 231, rendering library 232, pixel buffer 233, and block-based encoder 234) operate as described above in connection with Fig. 2. However, the

application engine 159 adds a controller 1591, a data cache 1592, and a “shim” 1593, that cooperate to perform novel functionality as described below. Therefore, the application engine leverages the functions of the remote browser components 231-234 without modifying them. Because of this design, when newer and improved versions of remote browser components are released by third party developers, this embodiment advantageously may be adapted to integrate with the new components without requiring substantial modification.

**[0062]** The controller 1591 is responsible for controlling and optimizing the encoding of portions of the graphical user interface of an application. For purposes of concreteness, the application execution environment described herein provides a web browser, but the invention may be used with other application engines having modules that interact via an API. The controller receives service requests from a client device 20 and returns encoded audiovisual data.

**[0063]** The controller is coupled to a data cache 1592. This cache stores encoded audiovisual data that may be decoded by the client device 20 for display on a display device 22. For example, and not by way of limitation, the audiovisual data may be encoded according to an MPEG standard. The cached data may include either full frame, intracoded data (I-frames), intercoded data (P-frames, or B-frames) or MPEG fragments as disclosed in U.S. patent application 12/443,571. It will be appreciated that the data cache 1592 may be shared between application engine instances, so that it may be accessed by any number of controllers.

**[0064]** A shim 1593 is a software mechanism that is interposed between the layout engine 231 and the rendering library 232. As described above in connection with Fig. 2, a prior art layout engine sends paint instructions to a rendering library according to the library’s API. However, in accordance with the embodiment shown in Fig. 3, the shim intercepts these instructions and processes them. The shim passes some instructions through to the rendering library automatically, so that the instructions appear to have been issued by the layout engine. For example, if the paint instruction modifies a state in the library (e.g., instructs the library to use a particular coordinate system), or obtains information from the rendering library, then the shim forwards the instruction and returns any response to the layout engine. However, the shim may or may not forward certain other paint instructions to

the rendering library, such as rendering instructions, depending on whether it is in a 'forwarding' state or a 'non-forwarding' state. The controller instructs the shim as to which of these two states it should have, as described below. By avoiding unnecessary rendering, the shim advantageously saves processing time and memory.

**[0065]** The operation of the embodiment of Fig. 3 is now explained in more detail with reference to Figs. 4, 5, and 6. Fig. 4 is a flowchart showing a method of generating an initial screen for a graphical user interface in accordance with an embodiment of the invention. Figs. 5A-5C collectively comprise a flowchart showing a method of generating a screen update. Figs. 6A-6D show various screen areas affected by these methods, and Fig. 6E shows an exemplary pixel buffer.

**[0066]** With reference to Fig. 4, a method to generate an initial screen for a client device begins in process 40, in which the controller receives a page request from the client device. This request may be generated, for example, when an individual presses a button on remote control 21, thereby activating the requested application. In process 41, the layout engine (having performed any necessary preprocessing such as retrieving HTML data associated with a URL) determines and positions graphical objects according to methods known in the art. After the data are properly positioned based on their dimensions and other factors, they are rendered in process 42, in which the layout engine requests rendering from one or more rendering libraries. In process 43, the initial pixel buffer data are populated with the drawing outputs of the one or more rendering libraries. In process 44, the pixel buffer data are encoded according to an audiovisual encoding scheme. As known in the art, this scheme may be a block-based encoding scheme such as MPEG. In process 45, the encoded data are sent to the client device 20 for eventual display on display device 22. An example of an initial screen generated as a result of this method is shown in Fig. 6A.

**[0067]** A method of providing a screen update to a client device begins in Fig. 5A. This method may be triggered when an individual activates a control in the graphical user interface that causes a portion of the screen to be updated. The method causes only the portion of the screen to be encoded, and a new image to be transmitted to the client device 20, thereby saving memory and computing resources in the application engine. For encoding schemes other than MPEG, the method potentially saves bandwidth between the application engine and the client device. An example of a screen update request is shown by comparing

Fig. 6A to Fig. 6B. Fig. 6A represent an initial screen prompting an individual to choose between tomatoes and potatoes. Fig. 6B represents the desired output of highlighting a button around the “potatoes” element. Highlighting the button is a “screen update” that does not require a full-screen refresh.

**[0068]** The screen update method begins in process 50, in which the application engine receives a screen update request from the client device. Upon receiving the user input, the controller passes it to the layout engine. In process 51, the layout engine creates and returns to the controller a list of dirty rectangles; i.e., rectangular areas of the screen that must be repainted (redrawn) in response to the request. Fig. 6C shows an example of such a dirty rectangle that corresponds to the button of Fig. 6B. This dirty rectangle is the smallest rectangle that may be drawn completely around the affected button. The size and location of dirty rectangles may be determined in accordance with methods known in the art of layout engines.

**[0069]** In process 52, the controller instructs the shim to prevent rendering; that is, to enter the ‘non-forwarding’ state. Therefore, any rendering paint instructions received by the shim from the layout engine will not be sent to the rendering library.

**[0070]** In process 53, the controller determines whether any rectangles need resizing. This determination is made with knowledge of the size of the blocks of pixels encoded by the block-based encoder. Thus, if the encoder operates on MPEG macroblocks that are 16 pixels by 16 pixels (256 pixels in each block), the controller optionally may determine whether each dirty rectangle is aligned on 16 pixel boundaries. If a rectangle is not so aligned, the controller may determine to resize the dirty rectangles, and proceed to a process 531 in which the controller snaps the rectangles to pixel block boundaries. Fig. 6D shows the dirty rectangle of Fig. 6C, expanded to align with 16 pixel macroblocks. If one or more rectangles were resized, then the controller modifies the received repaint request (or creates a new repaint request) in a process 532, so that the layout engine will cause the proper screen area to be repainted. Thus, in accordance with these optional latter two processes 531, 532, the controller determines the smallest rectangle consisting of macroblocks that surrounds the graphical object being repainted. In this case, the repaint request sent to the layout engine reflects this smallest surrounding rectangle, and the output of the layout engine will include parameters that reflect the smallest surrounding rectangle. The above processes may be



performed using a pixel buffer provided by the controller and having the size and shape of the smallest surrounding rectangle, into which current screen image data have been copied, so that any newly rendered image will be drawn on top of the current screen image.

Alternately, the above processes may be performed without such a pixel buffer.

[0071] Whether or not the controller determines to resize any rectangles, in process 54 the layout engine processes the list of dirty rectangles to produce one or more paint instructions. These instructions have parameters that indicate how the instructions should be executed. For example, the parameters may define the size and coordinates of a dirty rectangle having an image to be re-rendered, and they may define properties of a graphical object, such as a font, weight, and size for a text string. In prior art systems, these instructions would be sent from the layout engine 231 directly to the rendering library 232, but in accordance with this embodiment of the invention, the shim 1593 instead intercepts the instructions.

[0072] Continuing the method in Fig. 5B as indicated, recall that the shim is in the 'non-forwarding' state. Thus, in process 55, rather than forwarding the instruction to the rendering library, instead the shim computes a hash value based on the received painting data. This hash value may be computed using a hash function known in the art for producing a small number (a hash) based on a large number according to a computationally inexpensive algorithm that deterministically distributes hash values uniformly and approximately randomly across the set of small output numbers. Because hash values are calculated deterministically, applying the function to the same input twice will yield the same output both times. Because hash values are distributed approximately randomly, applying the function to different inputs will yield different outputs in all but a vanishing number of cases. Thus, hash values are small numbers that may be used to discriminate between large data sets without requiring expensive comparison of the large data sets themselves.

[0073] The hash value may be calculated based on the painting data received by the shim, and especially the parameters of at least one paint instruction. In one embodiment, pixel data pertaining to a graphical object are used to produce the hash value. In another embodiment, the hash is calculated as a function of a series of incremental paint instructions that pertain to a particular rectangle. Other variations are contemplated, so long as the hash function is applied uniformly to paint instructions that would result in identical output

graphics. Thus, if multiple users of the same menuing interface, accessing the menu at different times, request identical behaviors of the interface, then the same hash value is produced for both users. This is true even if the two users access different application engine instances, and even if some of the parameters (such as a session identifier) are different. Moreover, such identical output graphics could occur at different locations on the screen. For example, a menu button may be rendered at different locations in different menu screens, but otherwise appear identical.

[0074] In process 56, the shim transmits the hash value to the controller. The controller 1591 then consults the cache 1592 using the received hash value to determine whether there is an associated entry in the cache. If the data are determined to be in the cache in process 57, then in process 571 the controller immediately retrieves the encoded audiovisual data from the cache, and in process 572 the controller transmits the retrieved data to the client device. Because MPEG does not allow a system to send encoded images that represent less than a full frame to a client device, and because the encoded audiovisual data may represent less than a full frame, the encoded data may be stitched or composited into other encoded data to form a full frame prior to transmission, in accordance with methods known in the art. In process 573, the controller instructs the shim to discard the paint instruction it received from the layout engine, as it is no longer needed.

[0075] Thus, if the data are already cached, no further rendering or encoding is necessary to deliver the content to the client device that requested it. If, however, in process 57 the data are determined not to be in the cache, then they must be rendered and encoded. In this case, in process 58 the controller instructs the shim to permit painting (that is, to enter the 'forwarding' state), and in process 59 the controller resends the previous repaint request to the layout engine. At this point, the controller also temporarily stores the received hash value for later use as described below.

[0076] Continuing the process in Fig. 5C as indicated, in process 510 the layout engine resends the repaint request to the shim. Unlike previously, the shim now has been configured to forward the received paint instruction to the rendering library, which it does in process 511. This pass-through effect may be accomplished using the rendering library API in the same manner as the layout engine would if the shim were not present. In process 512, the rendering library creates a pixel buffer having the appropriate pixel data. For example,

Fig. 6E shows a pixel buffer associated with the (expanded) dirty rectangle of Fig. 6D. In Fig. 6E, the word “potatoes” is visible along with the button around it. Therefore, this rectangle corresponds to the pixel data (of Fig. 6B) that must be encoded by the encoder.

[0077] At this point in the process, an optional animation detection method may be invoked. The purpose of the optional method is to determine whether any optimizations may be made to the encoding process. This optional method is described below in connection with Fig. 7.

[0078] In process 513, the encoder encodes the rendered pixel data in the pixel buffer to form encoded audiovisual data. Process 513 may be performed according to methods known in the art, or it may be performed according to methods described in further detail below in connection with detecting and encoding animations, and/or performing slice linking and cutting. In process 514, the controller receives the encoded pixel data and stores it in the screen update cache 1592. These encoded data are stored in unique association with the hash value previously received by the controller in process 56. Thus, if a future screen update request causes the shim 1593 to generate an identical hash value, the encoded data will be available in the cache for immediate retrieval. Next, in process 515, the encoded pixel data are formed into an audiovisual data stream. This process may include generating a continuous stream of frames according to a fixed number of frames per second, in accordance with an industry encoding standard such as MPEG. During this process, any number (zero or more) MPEG fragments may be combined with output from a scaled and/or transcoded input video stream to form the final encoded audiovisual data stream. Finally, in process 516 the controller transmits the encoded audiovisual data stream to the client device. Advantageously, this method does not require an MPEG motion search on the entire displayed screen, but only the “dirty” rectangle that is being updated. The method therefore requires less processing power than in the prior art.

[0079] The above method may be modified as follows. In process 58, the shim receives a command from the controller to permit painting. The purpose of this command is to permit the system to render the received painting data. However, these painting data already are stored in the shim. Therefore, in an alternate embodiment, rather than executing processes 59, 510, and 511 (which collectively require a further repaint request being issued

to the layout engine), the shim may forward the painting data directly to the rendering library in process 58 upon receiving notification that there was a cache “miss”.

[0080] The above method also may be modified in a different manner. Some paint instructions read back pixel information from the pixel buffer used by the rendering library. However, the pixel buffer may include incorrect data (i.e., data of a previously rendered image) if the controller and shim bypassed the previous paint instruction because the image was found in the cache. In this case, the cached image may be retrieved, and the shim may either simulate the effect of the paint instruction directly, or update the state of the rendering library to use the retrieved, cached image and then pass the paint instruction to the library for execution. The information read from the pixel buffer might also be cached for later retrieval if a similar sequence of paint commands is issued.

#### Detecting Animations

[0081] According to the embodiments described above, each image is individually compressed in isolation; for example, the images may be compressed using MPEG intra-encoding. However, sometimes an application will provide a repeating sequence of images that forms an animation, and images in the sequence may benefit from other optimizations. For example, regarding these sequences of images as an animation allows motion detection to be performed, resulting in much more efficient inter-encoding (e.g., producing P-frames and B-frames). This increase in efficiency may manifest as, for example, a lower bandwidth required to transmit a video that includes the animation, or a higher quality for the same bandwidth.

[0082] Fig. 7 is a flowchart showing a method of detecting an animation in accordance with an embodiment of the invention. The method may be applied for any given screen update during or just before process 513 (in which the encoder encodes the frame pixel data).

[0083] The method begins with process 70, in which the controller compares the current rendered image with a previously rendered image to determine screen area overlap. The locations and sizes of the two images, but not necessarily their content, are compared to determine a percentage overlap in their respective pixel “surface area”. For example, a 50x100 pixel image having upper left coordinate (100,100) and a 50x100 pixel image having

upper left coordinate (105,95) have an overlap of 45x95 pixels, or a percentage surface area overlap of  $4275 / 5000 = 85.5\%$ . A sequence of screen updates for a flashing button, or a graphical object that is simply changing color, will have rectangles that do not change position on the screen, and will therefore have 100% screen area overlap. The controller stores a list including coordinates of previously rendered rectangles for this purpose. Because such a list includes only coordinate data, it may include data pertaining to a large number of previously rendered frames; therefore, the two images being compared need not be in consecutively rendered frames.

**[0084]** In process 71, a choice is made depending on whether the percentage overlap is substantial, as defined by a given minimum percentage. For illustrative purposes, and not by way of limitation, the minimum percentage may be 50%, so that two rectangles that share at least half of their pixel coordinates in common are considered to contain images that are part of a single animation. If there is not a substantial overlap, then in process 711 the controller determines whether there are any other previously rendered images in the list against which to compare the current image. If so, the method restarts at process 70 using a different previously rendered image, but if not, then the method ends.

**[0085]** However, if there is substantial overlap between the two compared image coordinates, then the algorithm concludes that the images form part of a single animation. To prevent loops, in process 72 a choice is made depending on whether the currently rendered image is identical to a first image in a previously-rendered chain of overlapping images. Rather than comparing the image pixel data directly, the hash values of the two images may be compared for improved efficiency. If the hash values are equal, then the current image is the first image of the animation cycle, and it does not need to be re-encoded. Thus, in process 721 the cached, encoded image is transmitted and the method ends.

**[0086]** If the image was not previously animated, then in process 73 the current image is intra-encoded. Further images that are determined to belong to the same animation chain are subsequently inter-encoded with respect to the previous image in the animation. Once the controller has determined that an animation is ongoing, new images generated by an application are checked against corresponding images, in sequence, in the stored animation. In case the current image does not match the corresponding stored image, a new animation sequence is started, and the first image in the sequence is intra-coded.

[0087] In accordance with the above discussion, an animation starts with intra-coded macroblocks, and subsequent images are generated as predictive macroblocks (P or B). It is sometimes the case that an animation starts at an intermediate image that has been predictively encoded, rather than the first, intra-coded image. Such an animation has a unique encoder history, so it needs to be identified as a different object in the cache. In particular, it has a different hash value than an animation that begins with the “first” image in the chain. Therefore, each chain of images in an animation is assigned a unique hash, calculated over the pixels of all individual images that are part of the chain. The displacement on the screen between images is also included in the hash calculation.

#### Slice cutting and slice linking

[0088] By way of background to inform another aspect of the invention, it is known in prior art MPEG systems to perform a periodic refresh of a screen by providing, to a client device, an entirely intra-coded frame (I-frame) of image data. Such refreshes eliminate screen artifacts caused by errors in the transmission of audiovisual data. However, intra-coded frames (I-frames) encode all pixel data in the image, and therefore require the use of more data than inter-coded frames (e.g. P-frames and B-frames) that merely encode the differences between successive images. I-frame transmissions therefore use more bandwidth than predictively coded frame transmissions. Moreover, they must be transmitted on a regular basis, or accumulating screen artifacts will eventually degrade the displayed image beyond usefulness.

[0089] Typically the high peak bitrate of an I-frame is handled by large buffers in the client, however this is detrimental for latency sensitive applications such as the interactive TV services that are the subject of the present invention. As a result of this problem, it is known to spread out the bitrate of a single I-frame across multiple transmitted frames by using a “rolling update”. In a rolling update, sometimes also called a “curtain refresh”, each consecutive frame updates a portion of the screen area using intra-encoded macroblocks. For example, each consecutive frame may update two or more rows of macroblocks, starting from the middle of the screen and progressing upwards and downwards simultaneously. The advantage to this type of refresh is that a rolling update distributes the large, intra-encoded macroblocks over multiple frames. As a result, the bitrate is slightly elevated over multiple

frames, instead of spiking as it would if all intra-encoded data were transmitted in a single frame. An alternative method of handling bitrate spikes by encoding I-frames at a very low bitrate, known as “I-frame pumping”, is known in the art but not discussed further herein.

**[0090]** An example of a vertical rolling update is shown graphically in Figs. 8A-8C. The example screen here consists of 10 rows of macroblocks, where each macroblock is a square of pixels. Rows having a right-slanted appearance represent predictively encoded image data from before a screen update, rows that are unshaded represent intra-encoded rows used in the rolling update, and rows having a left-slanted appearance represent predictively encoded image data having updated image data.

**[0091]** In Fig. 8A, central rows 5 and 6 are updated with intra-encoded macroblock data. As is known in the art, rows 5 and 6 may be represented by an intra-encoded MPEG slice (an I-slice). During this update, rows 1-4 and 7-10 may be updated with inter-encoded macroblock data pertaining to the current image (i.e., the image that is in the process of being replaced). Thus, each of these other rows may be represented by a P-slice or a B-slice. In Fig. 8B, rows 5 and 6 are updated with data (a P-slice or a B-slice) pertaining to the updated image, while rows 4 and 7 are updated with intra-encoded data (an I-slice) pertaining to the updated image, and the other rows are updated with inter-encoded data pertaining to the current image. In Fig. 8C, rows 3 and 8 are updated with intra-encoded data, while the other rows are updated with inter-encoded data. This process continues until each row has received intra-encoded macroblock data. It should be noted that newly refreshed slices can only perform motion searching and prediction within the refreshed screen area, and cannot refer to the non-refreshed areas.

**[0092]** One system in accordance with the invention stores screen objects as intra-encoded macroblocks, called “MPEG fragments”. To generate I-frames or intra-refresh rows based upon stored MPEG fragments, slices of one or more rows have to be cut and linked. The cutting and linking methods described below may be used during active periods where there are many screen updates.

**[0093]** The cutting and linking principles are illustrated with reference to Figs. 9A and 9B. Fig. 9A represents a “current image” displayed on a screen that is 14 rows of macroblocks in height and 24 columns of macroblocks (only the rows are marked). Thus, if a macroblock is a square 16 pixels on a side, this screen has a resolution of 384 by 224

pixels. Fig. 9B shows an “updated image” on the same screen, obtained by performing a screen update in accordance with an embodiment of the invention, has caused a rectangle 91 to be displayed. Rectangle 91 is five rows tall and 10 rows wide.

**[0094]** A method for integrating the image data of rectangle 91 into the rows of the screen is illustrated using Figs. 9C-9E. While these figures show the method as applied to only one row of macroblocks, it should be understood that this method must be repeated for each row of macroblocks that is covered (or partially covered) by rectangle 91. Fig. 9C shows one full-row slice of the screen 92. Logically superimposed on this slice is a slice 91a of MPEG fragments that represents a single row of macroblocks of the rectangle 91. To insert slice 91a into the row, the slice 92 is cut using a slice cutting method to form two partial-row slices 92a, 92b as shown in Fig. 9D. The slice cutting method is described in more detail below in connection with Figs. 10 and 11. Note that the three slices 92a, 91a, 92b together form 24 macroblocks; that is, when placed side-by-side, they have the width of a single row. However, they do not yet form a single slice. While the MPEG standard permits a row of macroblocks to be described by multiple slices, some display devices place a limit on the number of slices that may be used in a given frame (or the number of slices per second). In some extreme cases, a given frame of data may only permit as many slices as there are rows of macroblocks. Therefore, to account for such limitations, these three slices (or any two adjacent slices) may be linked to form a single slice, as shown in Fig. 9E. Slice linking is performed according to a slice linking method, described in more detail in connection with Figs. 12 and 13.

**[0095]** Slice cutting is a procedure that is required to perform an intra-refresh of the entire screen, built up of several possibly overlapping MPEG fragments. To compose the intra-encoded frame, only the non-obscured macroblocks of fragments are needed. Consequently, the slices in such fragments are cut.

**[0096]** Fig. 10 is a flowchart showing a method of cutting an MPEG4 slice in accordance with an embodiment of the invention. An MPEG slice includes macroblock image data. For sake of terminology, an original slice ‘S’ is cut to form two slices ‘S1’ and ‘S2’, where slice ‘S1’ includes those macroblocks earlier in the data stream and slice ‘S2’ includes those macroblocks later in the data stream. It will be understood that this method may be applied to standards other than MPEG4 by appropriate modification.



[0097] The method begins with a slice encoded (compressed) using a variable-length code (VLC) for transmission over a data network. For example, the slice shown in Fig. 11A is compressed, as indicated by the slanted lines, and contains 13 macroblocks. An arrow indicates where the slice should be cut. In process 1001, metadata are added to the slice S, for example in its elementary stream, as shown in Fig. 11B. In particular, these metadata pertain at least to the DC context of each macroblock in the slice. Next, in process 1002, the location in the compressed data stream of the start of the first macroblock of the new slice S2 is determined. This may be done by either VLC decoding the entire slice, or, if present, using macroblock pointers in the slice metadata. In process 1003, the found (compressed) macroblock is partially VLC decoded to produce uncompressed macroblock data, as shown in Fig. 11C. However, only DC luma and DC chroma information needs to be decoded; the full image data of the macroblock should not be decoded in the interest of efficiency. In process 1004, the DC luma and DC chroma information is located in the uncompressed data. Locating these data values may be done using methods known in the art. For example, in the H.264 standard, this information is stored in the Intra16x16DCLevel data block. The method only requires decoding of this information; other image data may remain compressed. In process 1005, the primary coefficient of the DC luma or DC chroma level is patched to match the DC context of the default slice start context, as shown in Fig. 11C. In this way, the macroblock may act as the first macroblock of an entire slice, namely the new slice S2. Patching may be accomplished using a bit-shifting operation; that is, the bits of the DC luma value or the DC chroma value may be shifted according to low-level, efficient bit-shifting instructions. In process 1006, the decoded portions of the patched macroblock are VLC re-encoded, as shown in Fig. 11D. Note that, in embodiments in which the slice metadata includes pointers to macroblocks in the compressed data stream, only the data of the patched macroblock must be VLC decoded and re-encoded; data of the other macroblocks in original slice S (including all data of slice S1 and the other macroblocks of slice S2) remain undisturbed by the method.

[0098] Fig. 12 is a flowchart showing a method of linking MPEG slices in accordance with an embodiment of the invention. Screen updates that consist of multiple fragments may result in more slices per line than can be permitted for certain end devices, especially for H.264 encodings. The purpose of slice linking is to reduce the number of slices

by linking two or more slices together. For the sake of simplicity, the process is described with respect to only two slices; those having ordinary skill in the art should understand that the process may be repeated to operate on more than two slices.

**[0099]** This method begins with two VLC-encoded slices S1' and S2' that must be linked, as shown in Fig. 13A. In process 1201, metadata are added to the slices, as shown in Fig. 13B. These metadata comprise at least the DC context of the last macroblock (right-most) of slice S1', the VLC state of this macroblock, and the DC context of the first macroblock (left-most) of the slice S2'. In process 1202, the first macroblock of slice S2' is partially VLC decoded using the VLC state of the last macroblock of slice S1'. As with the method of Fig. 10, only the Intra16x16DCLevel data block needs to be decoded. In process 1203, the Intra16x16DCLevel block is obtained for the first macroblock of slice S2'. In process 1204, the primary coefficient of this block is patched, using the metadata, to match the DC context of the last macroblock of the slice S1', as shown in Fig. 13C. The VLC tables for the left row of AC blocks are modified correspondingly. After patching, in process 1205 the decoded portions of the macroblock are VLC re-encoded. In process 1206, the compressed data are concatenated to form a new compressed slice S', as shown in Fig. 13D. As before, only the data of the patched macroblock must be VLC decoded and re-encoded; all data of slice S1' and data of the other macroblocks of slice S2' appear unchanged (and compressed) in the new slice S'.

**[0100]** The embodiments of the invention described above are intended to be merely exemplary; numerous variations and modifications will be apparent to those skilled in the art. All such variations and modifications are intended to be within the scope of the present invention as defined in any appended claims. For example, while H.264 stores DC luma and DC chroma information in a Intra16x16DCLevel data block, other standards such as MPEG2 and VC-1 store this data elsewhere; the methods and systems described above may be modified accordingly.

**[0101]** It should be noted that the logic flow diagrams are used herein to demonstrate various aspects of the invention, and should not be construed to limit the present invention to any particular logic flow or logic implementation. The described logic may be partitioned into different logic blocks (e.g., programs, modules, functions, or subroutines) without changing the overall results or otherwise departing from the true scope of the invention.

Often times, logic elements may be added, modified, omitted, performed in a different order, or implemented using different logic constructs (e.g., logic gates, looping primitives, conditional logic, and other logic constructs) without changing the overall results or otherwise departing from the true scope of the invention.

**[0102]** The present invention may be embodied in many different forms, including, but in no way limited to, computer program logic for use with a processor (e.g., a microprocessor, microcontroller, digital signal processor, or general purpose computer), programmable logic for use with a programmable logic device (e.g., a Field Programmable Gate Array (FPGA) or other PLD), discrete components, integrated circuitry (e.g., an Application Specific Integrated Circuit (ASIC)), or any other means including any combination thereof.

**[0103]** Computer program logic implementing all or part of the functionality previously described herein may be embodied in various forms, including, but in no way limited to, a source code form, a computer executable form, and various intermediate forms (e.g., forms generated by an assembler, compiler, linker, or locator). Source code may include a series of computer program instructions implemented in any of various programming languages (e.g., an object code, an assembly language, or a high-level language such as Fortran, C, C++, JAVA, or HTML) for use with various operating systems or operating environments. The source code may define and use various data structures and communication messages. The source code may be in a computer executable form (e.g., via an interpreter), or the source code may be converted (e.g., via a translator, assembler, or compiler) into a computer executable form.

**[0104]** The computer program may be fixed in any form (e.g., source code form, computer executable form, or an intermediate form) either permanently or transitorily in a tangible storage medium, such as a semiconductor memory device (e.g., a RAM, ROM, PROM, EEPROM, or Flash-Programmable RAM), a magnetic memory device (e.g., a diskette or fixed disk), an optical memory device (e.g., a CD-ROM), a PC card (e.g., PCMCIA card), or other memory device. The computer program may be fixed in any form in a signal that is transmittable to a computer using any of various communication technologies, including, but in no way limited to, analog technologies, digital technologies, optical technologies, wireless technologies (e.g., Bluetooth), networking technologies, and

internetworking technologies. The computer program may be distributed in any form as a removable storage medium with accompanying printed or electronic documentation (*e.g.*, shrink wrapped software), preloaded with a computer system (*e.g.*, on system ROM or fixed disk), or distributed from a server or electronic bulletin board over the communication system (*e.g.*, the Internet or World Wide Web).

**[0105]** Hardware logic (including programmable logic for use with a programmable logic device) implementing all or part of the functionality previously described herein may be designed using traditional manual methods, or may be designed, captured, simulated, or documented electronically using various tools, such as Computer Aided Design (CAD), a hardware description language (*e.g.*, VHDL or AHDL), or a PLD programming language (*e.g.*, PALASM, ABEL, or CUPL).

**[0106]** Programmable logic may be fixed either permanently or transitorily in a tangible storage medium, such as a semiconductor memory device (*e.g.*, a RAM, ROM, PROM, EEPROM, or Flash-Programmable RAM), a magnetic memory device (*e.g.*, a diskette or fixed disk), an optical memory device (*e.g.*, a CD-ROM), or other memory device. The programmable logic may be fixed in a signal that is transmittable to a computer using any of various communication technologies, including, but in no way limited to, analog technologies, digital technologies, optical technologies, wireless technologies (*e.g.*, Bluetooth), networking technologies, and internetworking technologies. The programmable logic may be distributed as a removable storage medium with accompanying printed or electronic documentation (*e.g.*, shrink wrapped software), preloaded with a computer system (*e.g.*, on system ROM or fixed disk), or distributed from a server or electronic bulletin board over the communication system (*e.g.*, the Internet or World Wide Web).

What is claimed is:

1. A method of providing an image to a client device from an application execution environment having a layout engine that assembles graphical components into a graphical user interface screen for a graphical application, and a rendering library that renders graphical components into pixels, the method comprising:
  - receiving, from the layout engine, one or more paint instructions having parameters that pertain to a given graphical object;
  - computing a hash value based on the received one or more paint instructions;
  - when the hash value is contained within a cache memory, retrieving, from the cache, encoded audiovisual data that are uniquely associated with the hash value, and transmitting the retrieved audiovisual data to the client device; and
  - when the hash value is not contained within the cache,
    - forwarding the received one or more paint instructions to the rendering library for rendering the graphical object into pixels according to the paint instruction,
    - encoding the rendered pixels into encoded audiovisual data,
    - storing the hash value and the encoded audiovisual data in the cache, whereby the hash value and the encoded audiovisual data are uniquely associated, and
    - transmitting the encoded audiovisual data to the client device.
2. The method of claim 1, wherein the client device is one of the group consisting of: a television, a television set-top box, a tablet computer, a laptop computer, a desktop computer, and a smartphone.
3. A method according to claim 1, wherein the graphical application is one of the group consisting of: a web browser and a menu interface.
4. A method according to claim 1, wherein encoding comprises dividing the screen into blocks of pixels, the method further comprising:
  - after receiving the painting data and before computing the hash value, determining the smallest rectangle consisting of whole blocks of pixels that surrounds the at least one graphical object;
  - requesting that the layout engine repaint the smallest surrounding rectangle; and
  - receiving, from the layout engine, second painting data that include at least one paint

instruction having parameters that reflect the smallest surrounding rectangle,

wherein computing the hash value is based on the second painting data.

5. A method according to claim 1, further comprising:

determining that the hash value is contained within the cache by comparing the hash value to a stored hash value of a cached image that forms part of an animation.

6. A method of transmitting, to a client device, images that comprise an animation, the method comprising:

receiving a current image into a computing processor;

when the current image is identical to a previously rendered image that is uniquely associated with an encoded image in a cache memory, transmitting to the client device the cached, encoded image without encoding the current image; and

when the current image is not identical to a previously rendered image, but the current image shares at least a given minimum percentage of its pixels with a given, previously rendered image:

identifying the current image and the given, previously rendered image as belonging to a common animation;

encoding the current image according to a predictive encoding scheme, storing the encoded current image in the cache memory, and transmitting to the client device the encoded current image.

7. A method according to claim 6, wherein the predictive encoding scheme is an MPEG encoding scheme.

8. A method according to claim 6, wherein the given previously rendered image was not rendered immediately previously to the current image.

9. A method according to claim 6, wherein the given previously rendered image is uniquely associated with a predictively encoded image in the cache memory.

10. A method according to claim 6, further comprising computing a hash value for each unique chain of images that forms an animation, the hash value being a function of all images in the chain of images and a screen displacement between two consecutive images in the chain.

11. A method of forming two encoded slices from data comprising a given encoded slice, each encoded slice comprising a sequence of macroblocks that are encoded according to a

variable length code, the method comprising:

locating, in the given slice, a location of a macroblock; and

altering a DC luma value or a DC chroma value of the located macroblock without fully decoding the macroblock according to the variable length code;

wherein the first formed slice consists of the data of the given slice up to but not including the altered macroblock, and the second formed slice consists of the altered macroblock and any subsequent macroblocks in the given slice.

12. The method of claim 11, wherein altering the DC luma value or the DC chroma value is performed using a bit-shifting operation.

13. A method of combining a first encoded slice and a second encoded slice to form a third encoded slice, each encoded slice comprising a sequence of macroblocks that are encoded according to a variable length code, the method comprising:

altering a DC luma value or a DC chroma value in the first macroblock of the second slice without fully decoding the macroblock according to the variable length code; and

concatenating the data of the first slice with the altered macroblock and the data of the second slice to form the third encoded slice.

14. The method of claim 13, wherein altering the DC luma value or the DC chroma value is performed through a bit-shifting operation.

15. A tangible medium on which is stored non-transitory computer program code for providing an image to a client device from an application execution environment having a layout engine that assembles graphical components into a graphical user interface screen for a graphical application, and a rendering library that renders graphical components into pixels, the medium comprising:

program code for receiving, from the layout engine, one or more paint instructions having parameters that pertain to a given graphical object;

program code for computing a hash value based on the received one or more paint instructions;

program code for retrieving, from the cache, encoded audiovisual data that are uniquely associated with the hash value, and transmitting the retrieved audiovisual data to the client device when the hash value is contained within a cache memory; and

program code for:

forwarding the received one or more paint instructions to the rendering library for rendering the graphical object into pixels according to the one or more paint instructions, encoding the rendered pixels into encoded audiovisual data, storing the hash value and the encoded audiovisual data in the cache, whereby the hash value and the encoded audiovisual data are uniquely associated, and transmitting the encoded audiovisual data to the client device, when the hash value is not contained within the cache.

16. A medium according to claim 15, wherein the client device is one of the group consisting of: a television, a television set-top box, a tablet computer, a laptop computer, a desktop computer, and a smartphone.

17. A medium according to claim 15, wherein the graphical application is one of the group consisting of: a web browser and a menu interface.

18. A medium according to claim 15, wherein the program code for encoding comprises program code for dividing the screen into blocks of pixels, the medium further comprising:

program code for determining the smallest rectangle consisting of whole blocks of pixels that surrounds the at least one graphical object after receiving the painting data and before computing the hash value;

program code for requesting that the layout engine repaint the smallest surrounding rectangle; and

program code for receiving, from the layout engine, second painting data that include at least one paint instruction having parameters that reflect the smallest surrounding rectangle,

wherein computing the hash value is based on the second painting data.

19. A medium according to claim 15, further comprising:

program code for determining that the hash value is contained within the cache by comparing the hash value to a stored hash value of a cached image that forms part of an animation.

20. A medium according to claim 15, further comprising:

program code for receiving a current image into a computing processor;

program code for receiving a previously rendered image into the computer processor, the previously rendered image being uniquely associated with an encoded image in a cache



memory;

program code for transmitting to the client device the cached, encoded image without encoding the current image when the current image and the previously rendered image are identical; and

program code for:

encoding the current image according to a predictive encoding scheme,  
storing the encoded current image in the cache memory, and

transmitting to the client device the encoded current image

when the current image and the previously rendered image are not identical but share at least a given minimum percentage of their pixels.

21. A medium according to claim 20, wherein the predictive encoding scheme is an MPEG encoding scheme.

22. A medium according to claim 20, wherein the previously rendered image was not rendered immediately previously to the current image.

23. A medium according to claim 20, wherein the previously rendered image is uniquely associated with a predictively encoded image in the cache memory.

24. A medium according to claim 20, further comprising program code for computing a hash value for each unique chain of images that forms an animation, the hash value being a function of all images in the chain of images and a screen displacement.

25. A medium according to claim 15, further comprising program code for forming two encoded MPEG slices from data comprising a given encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks, the program code comprising:

program code for locating, in the given MPEG slice, a location of a macroblock that is encoded according to a variable length code;

program code for decoding the encoded macroblock according to the variable length code;

program code for altering a DC luma value in the decoded macroblock; and

program code for encoding the altered macroblock according to the variable length code,

wherein the first formed MPEG slice consists of the data of the given MPEG slice up to but not including the encoded macroblock, and the second formed MPEG slice consists of the

encoded macroblock and any subsequent encoded macroblocks in the given MPEG slice.

26. A medium according to claim 15, further comprising program code for combining a first encoded MPEG slice and a second encoded MPEG slice to form a third encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks, the program code comprising:

- program code for decoding the first macroblock of the second slice according to a variable length code;

- program code for altering a DC luma value in the decoded macroblock;

- program code for encoding the altered macroblock according to the variable length code; and

- program code for concatenating the data of the first slice with the encoded macroblock and the undecoded data of the second slice to form the third slice.

27. A system for providing an image to a client device from an application execution environment having a layout engine that assembles graphical components into a graphical user interface screen for a graphical application, and a rendering library that renders graphical components into pixels, the system comprising:

- a memory;

- a shim comprising hardware or a combination of hardware and software that is configured to:

- receive, from the layout engine, one or more paint instructions having parameters that pertain to a given graphical object,

- compute a hash value based on the received one or more paint instructions,

and

- when the hash value is not contained within the memory, forward the received one or more paint instructions to the rendering library for rendering the graphical object into pixels according to the one or more paint instructions; and

- a controller comprising hardware or a combination of hardware and software that is configured to:

- retrieve, from the memory, encoded audiovisual data that are uniquely associated with the hash value, and transmit the retrieved audiovisual data to the client device when the hash value is contained within the memory; and

transmit, to the client device, encoded audiovisual data comprising a rendering of the graphical object into pixels according to the received one or more paint instructions when the hash value is not contained within the memory.

28. A system according to claim 27, wherein the client device is one of the group consisting of: a television, a television set-top box, a tablet computer, a laptop computer, a desktop computer, and a smartphone.

29. A system according to claim 27, wherein the graphical application is one of the group consisting of: a web browser and a menu interface.

30. A system according to claim 27, wherein the memory stores a sequence of images that collectively form an animation, and wherein the controller is further configured to determine that the hash value is contained within the cache by comparing the hash value to a stored hash value of a cached image that forms part of the animation.

31. A system according to claim 27, wherein the audiovisual data are encoded according to an MPEG encoding scheme.

32. A system according to claim 27, further comprising a block-based encoder that is configured to form two encoded MPEG slices from data comprising a given encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks, by:

locating, in the given MPEG slice, a location of a macroblock that is encoded according to a variable length code;

decoding the encoded macroblock according to the variable length code;

altering a DC luma value in the decoded macroblock; and

encoding the altered macroblock according to the variable length code,

wherein the first formed MPEG slice consists of the data of the given MPEG slice up to but not including the encoded macroblock, and the second formed MPEG slice consists of the encoded macroblock and any subsequent encoded macroblocks in the given MPEG slice.

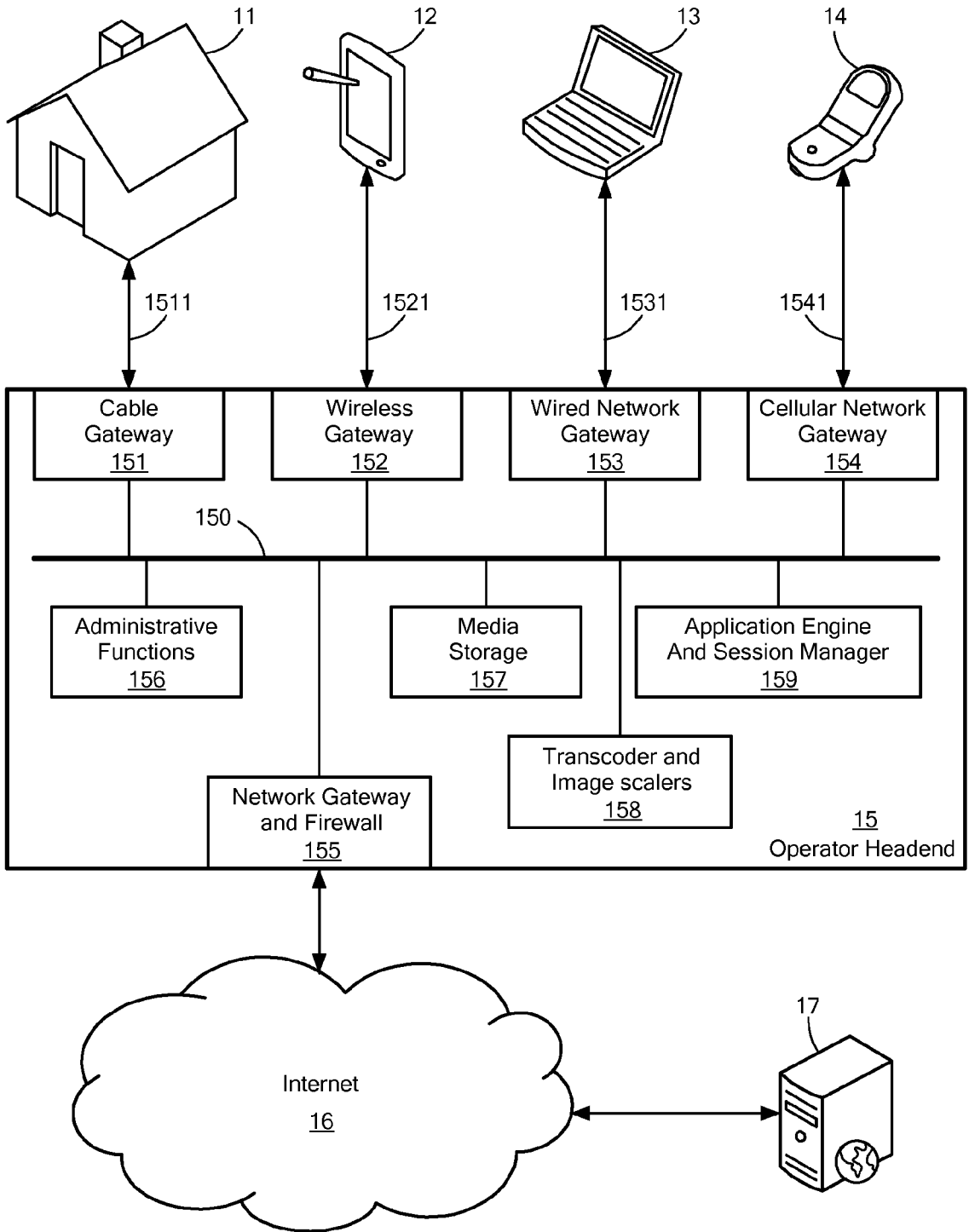
33. A system according to claim 27, further comprising a block-based encoder that is configured to combine a first encoded MPEG slice and a second encoded MPEG slice to form a third encoded MPEG slice, each encoded MPEG slice comprising a sequence of encoded macroblocks, by:

decoding the first macroblock of the second slice according to a variable length code;

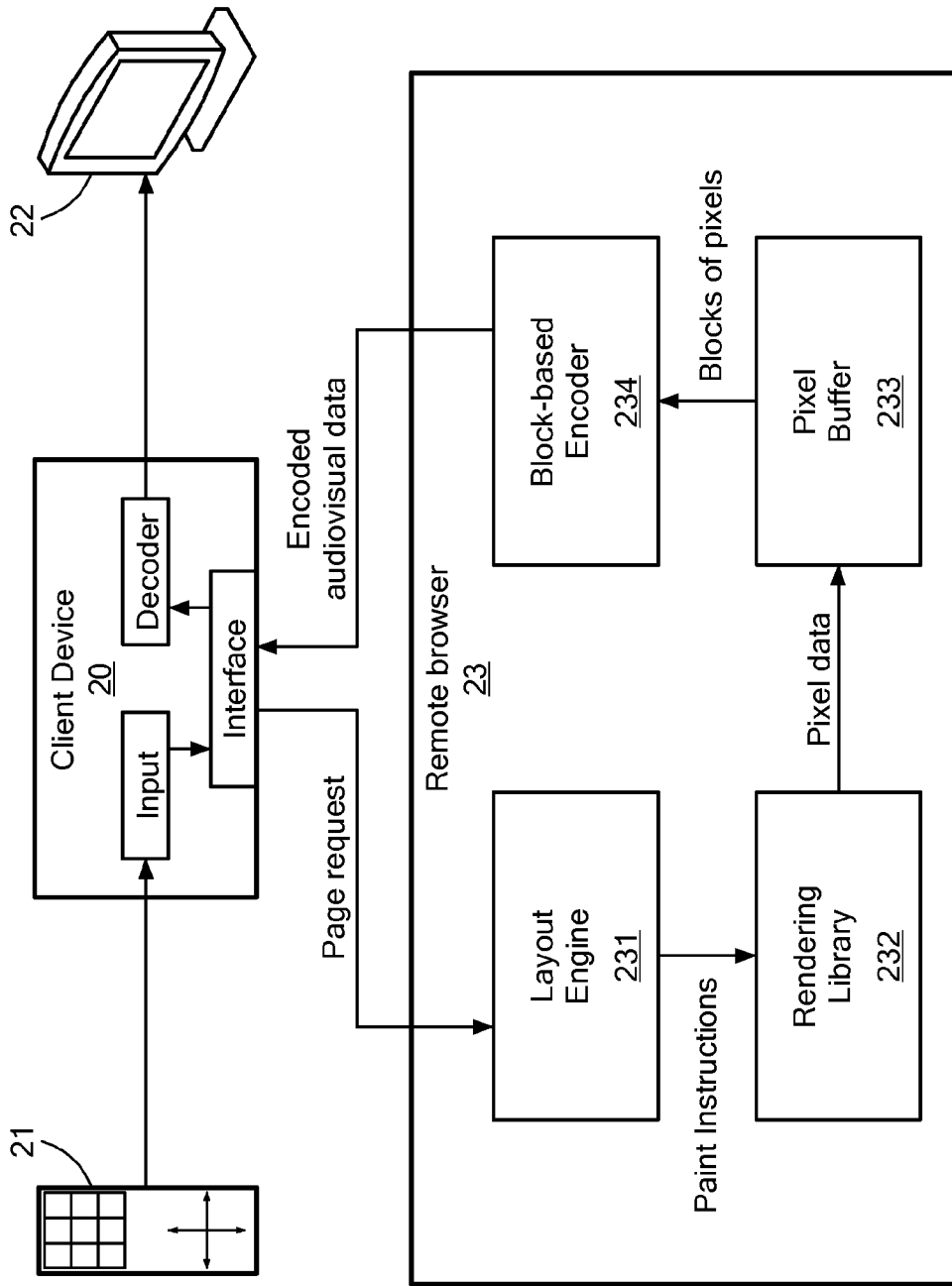
altering a DC luma value in the decoded macroblock;

encoding the altered macroblock according to the variable length code; and  
concatenating the data of the first slice with the encoded macroblock and the  
undecoded data of the second slice to form the third slice.

1/18



**FIG. 1**



**FIG. 2**

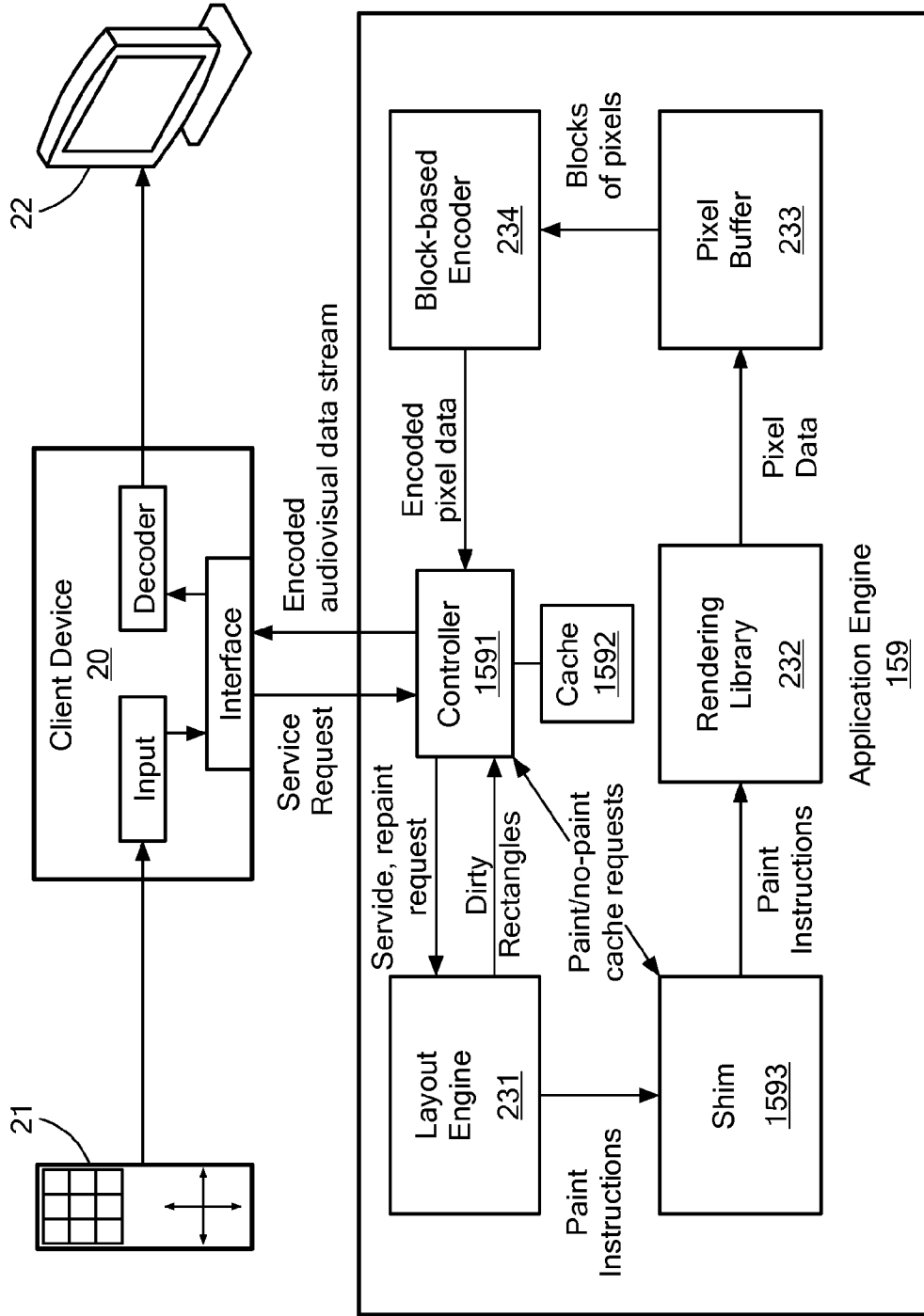
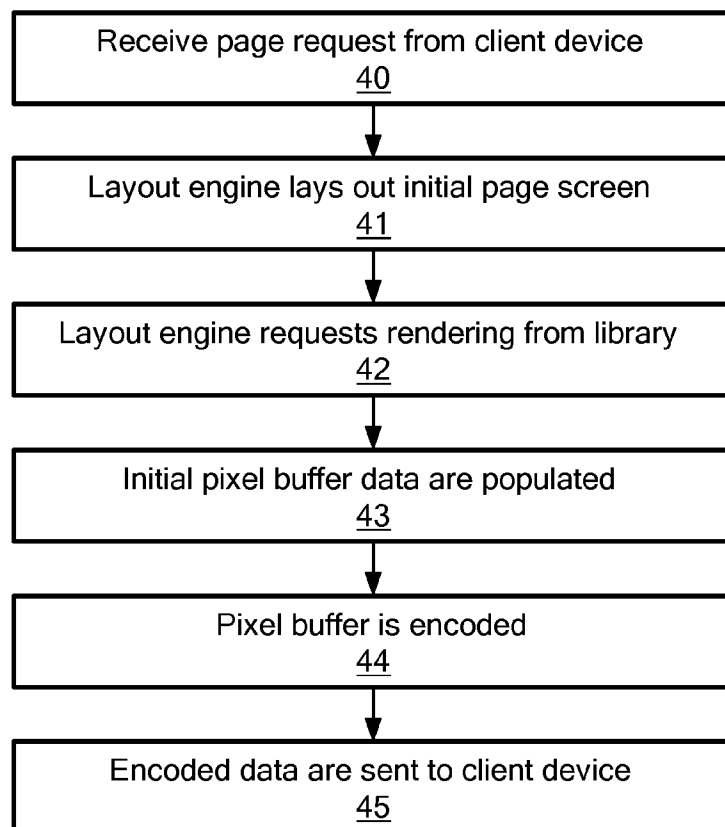
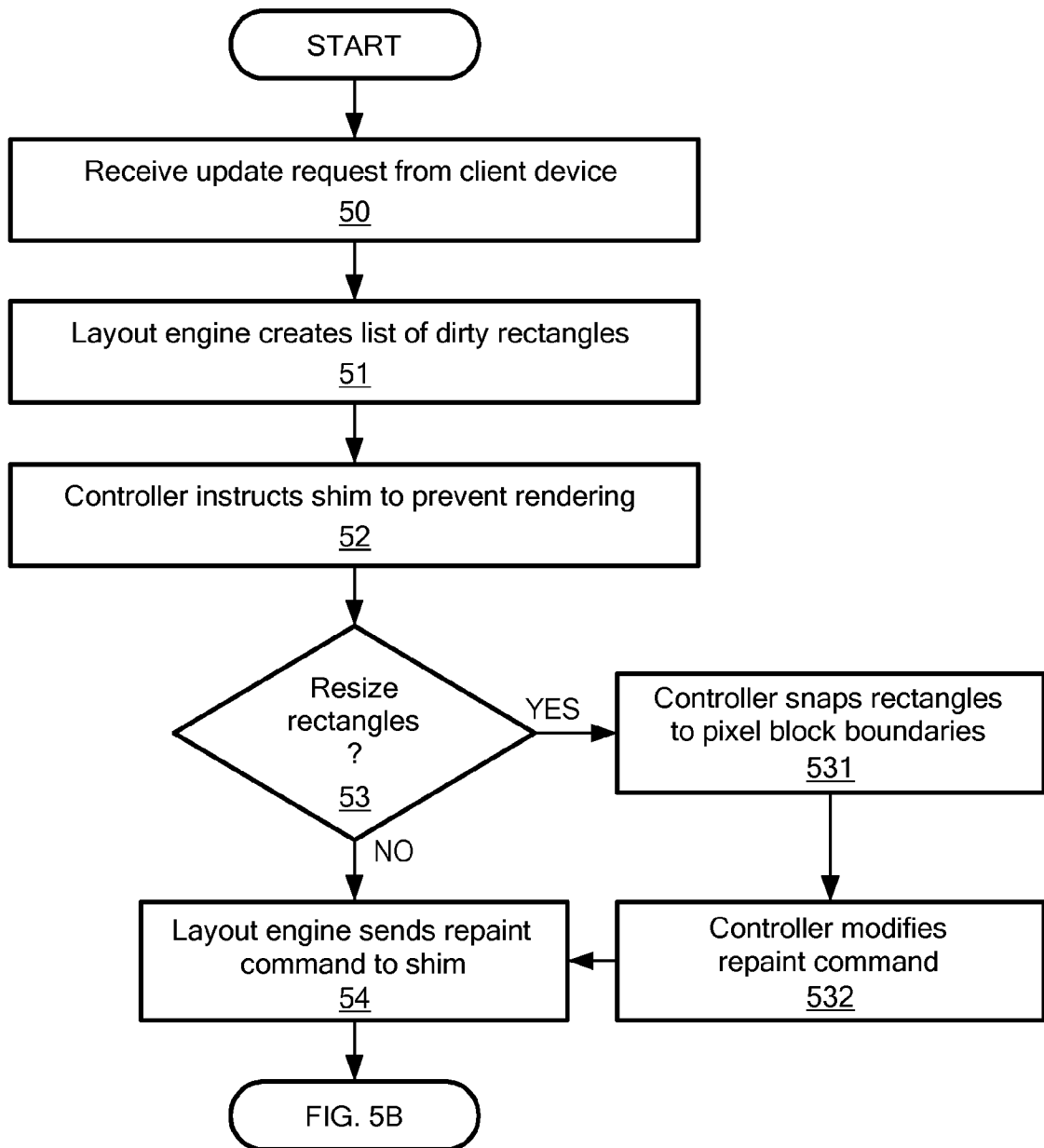


FIG. 3

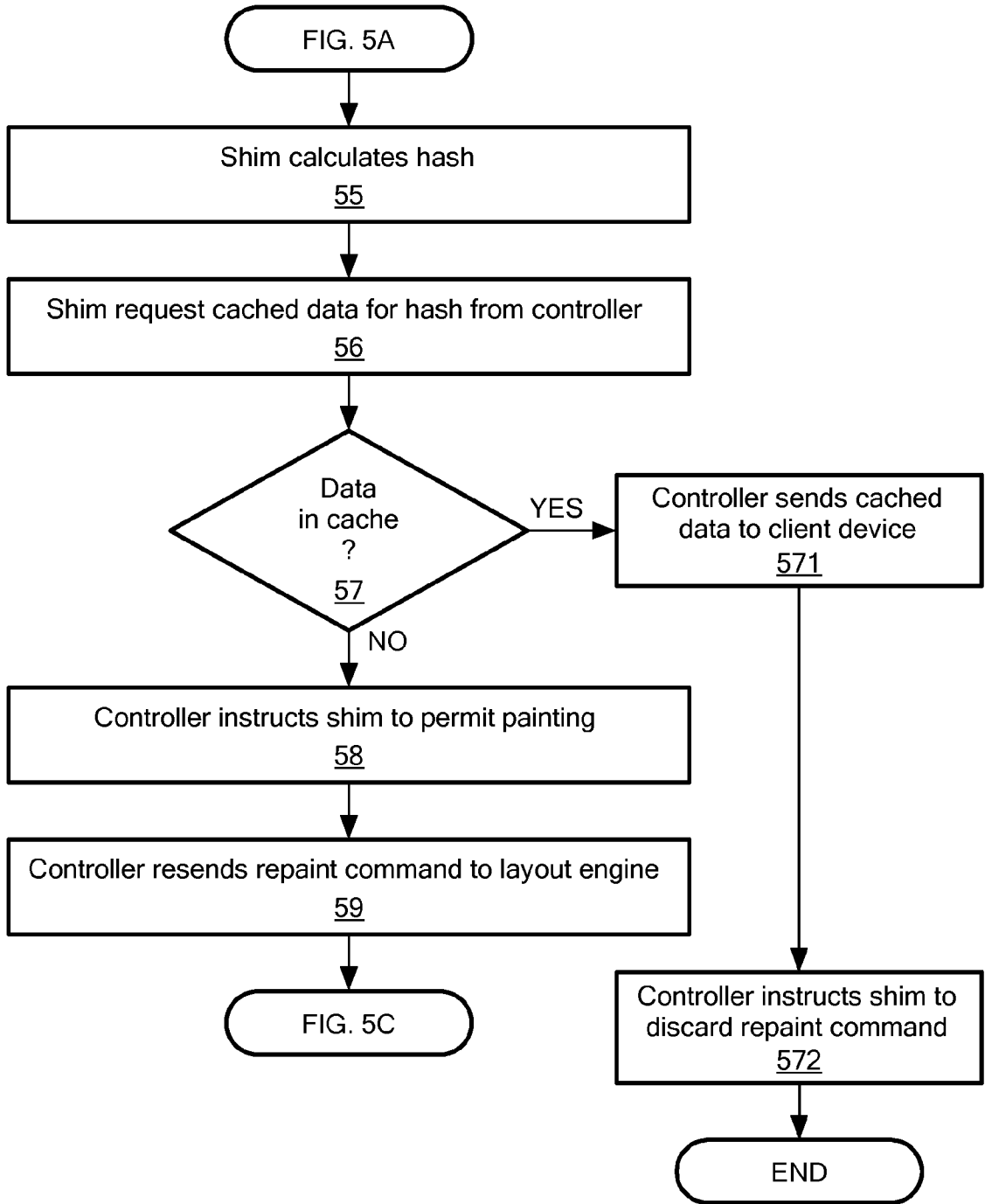
4/18

**FIG. 4**

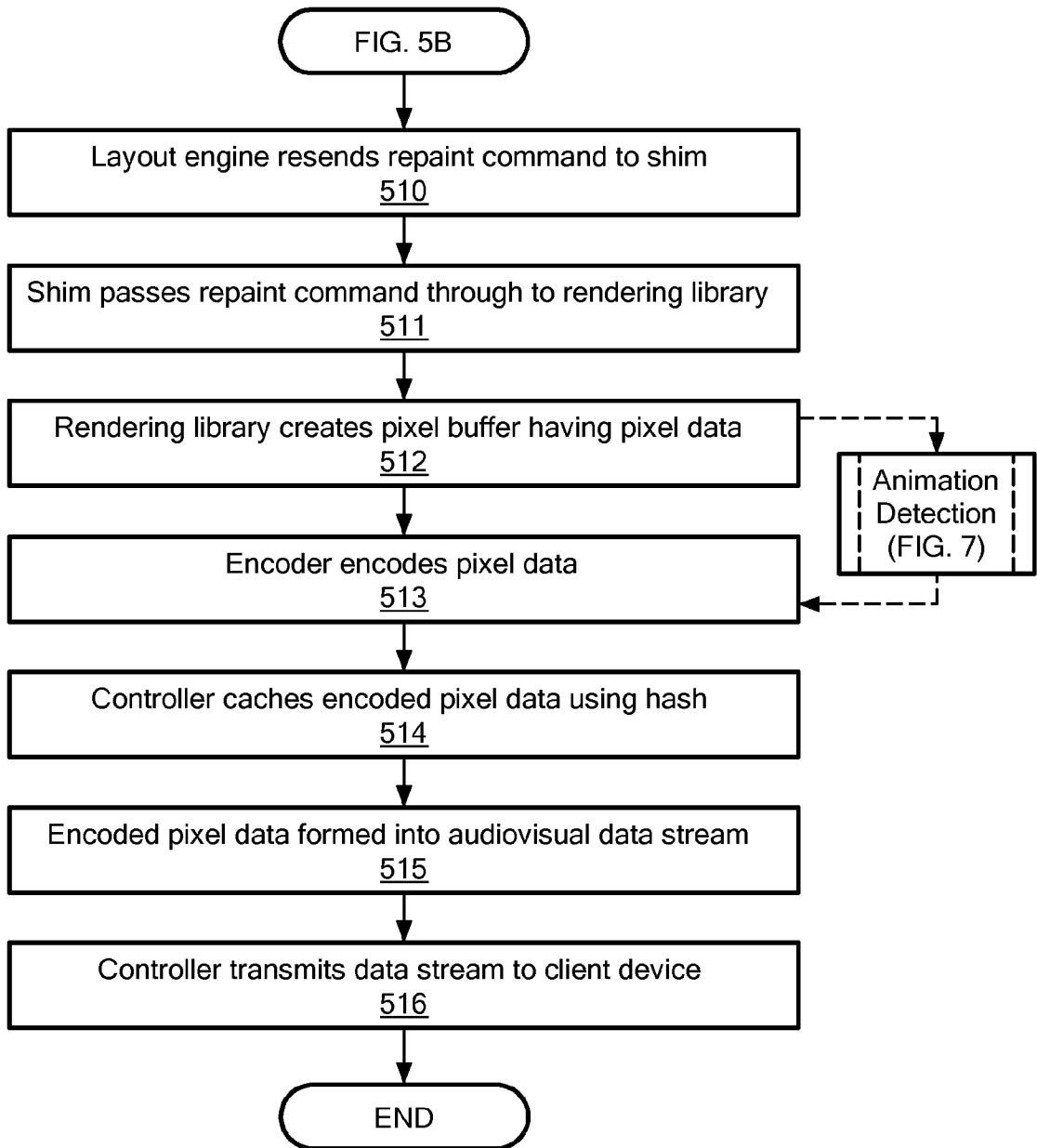




**FIG. 5A**

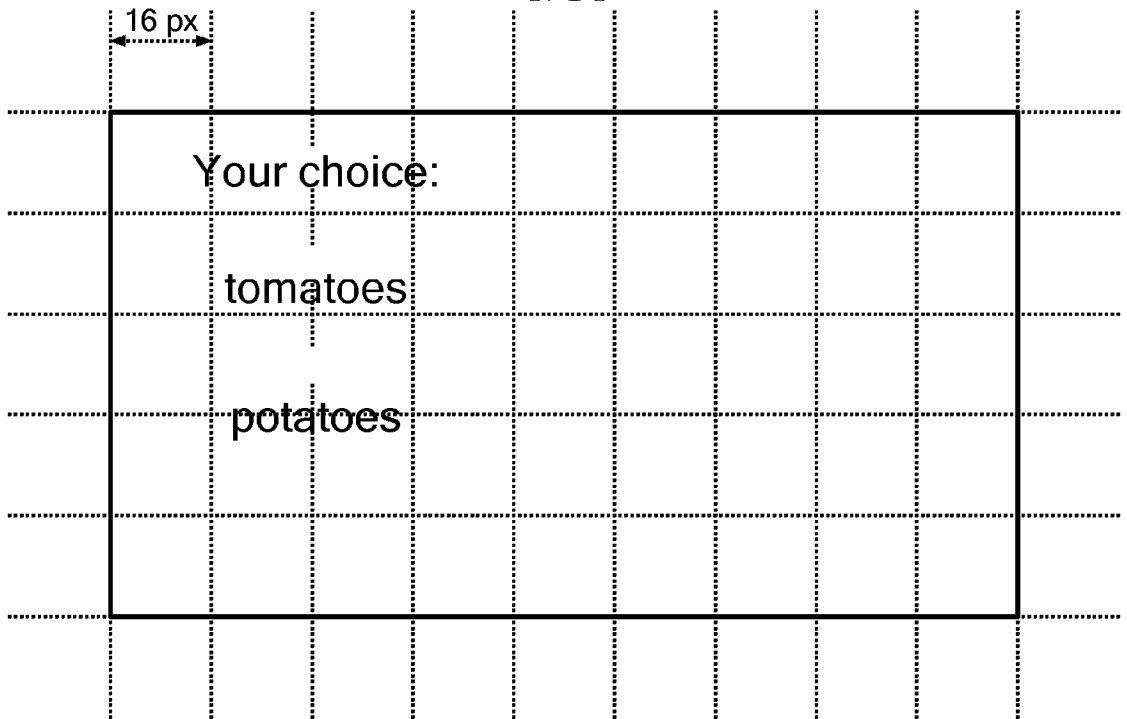


**FIG. 5B**

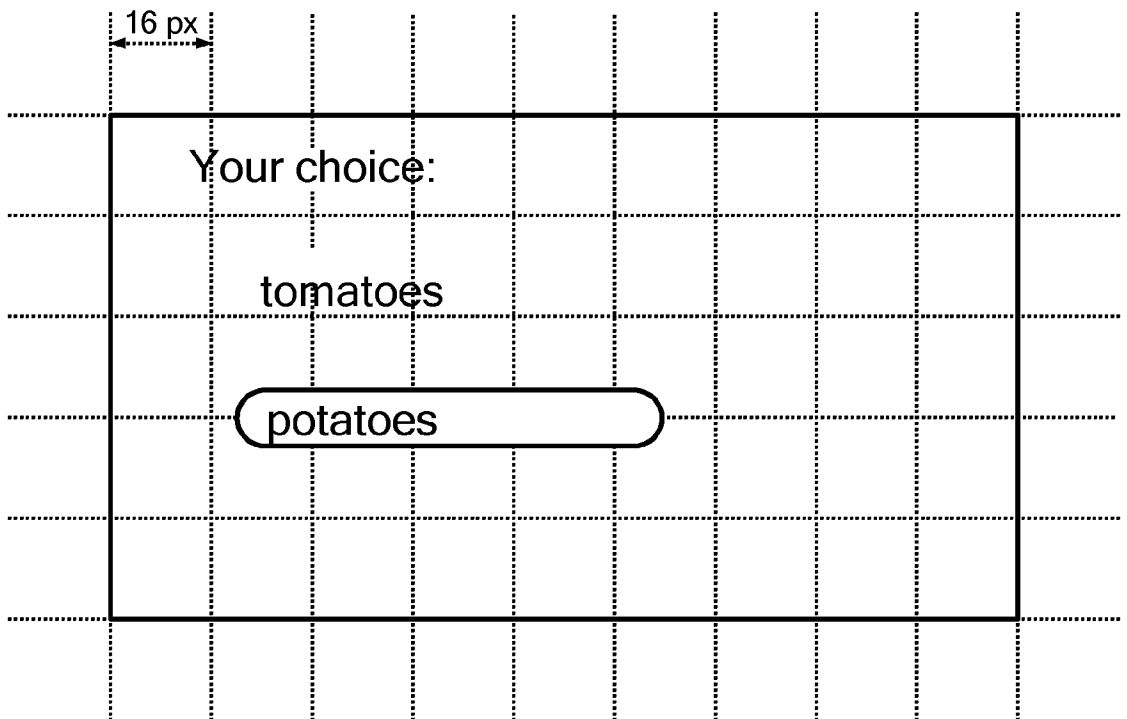


**FIG. 5C**

8/18

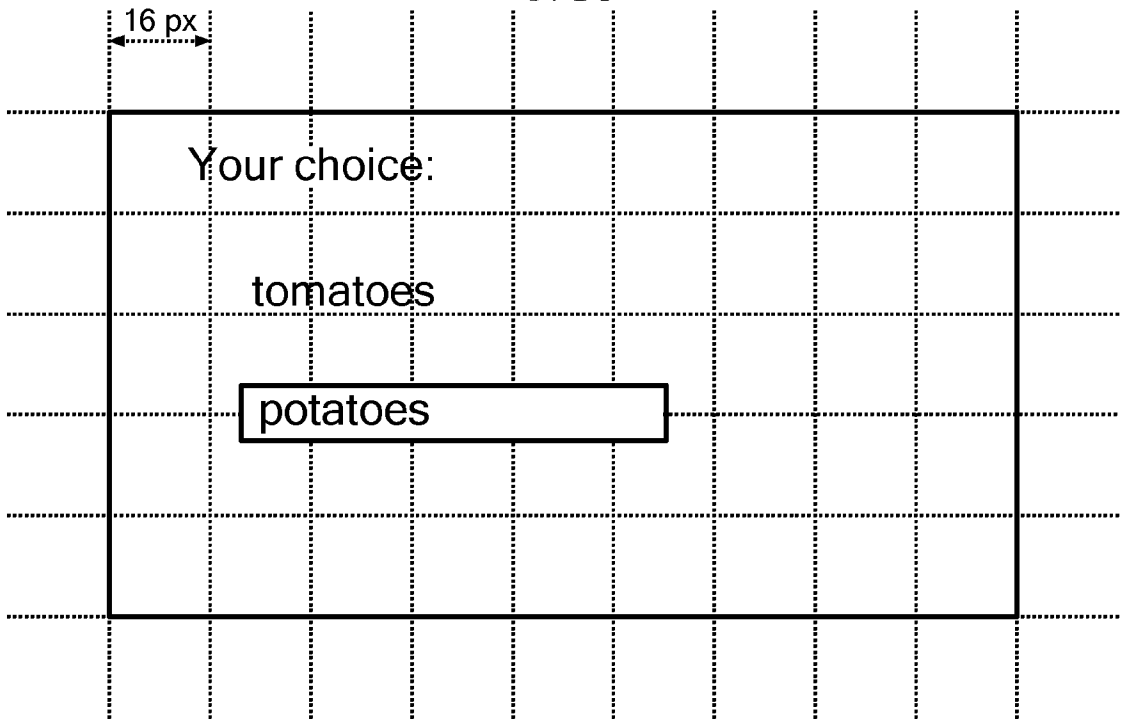


**FIG. 6A**

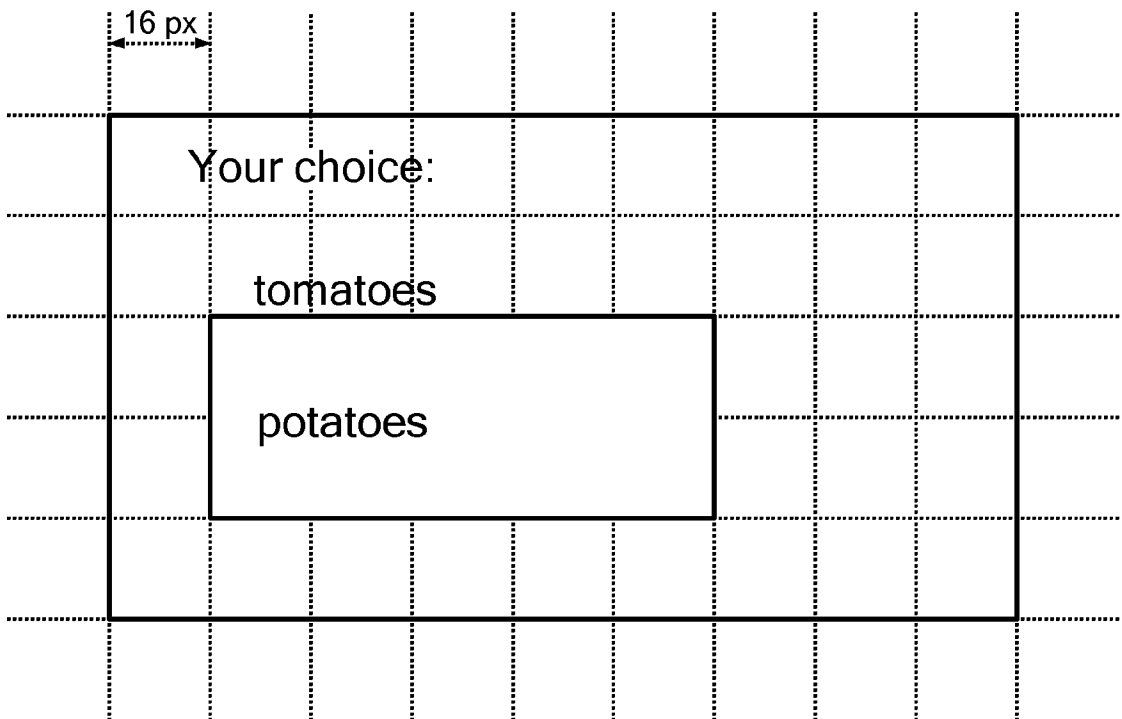


**FIG. 6B**

9/10

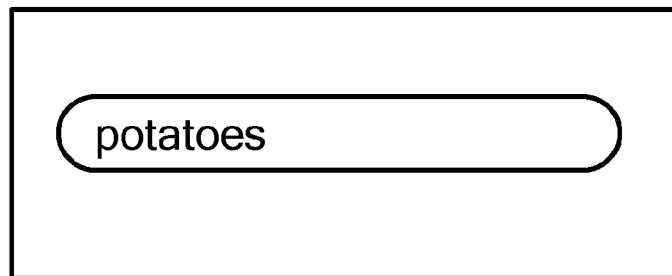


**FIG. 6C**

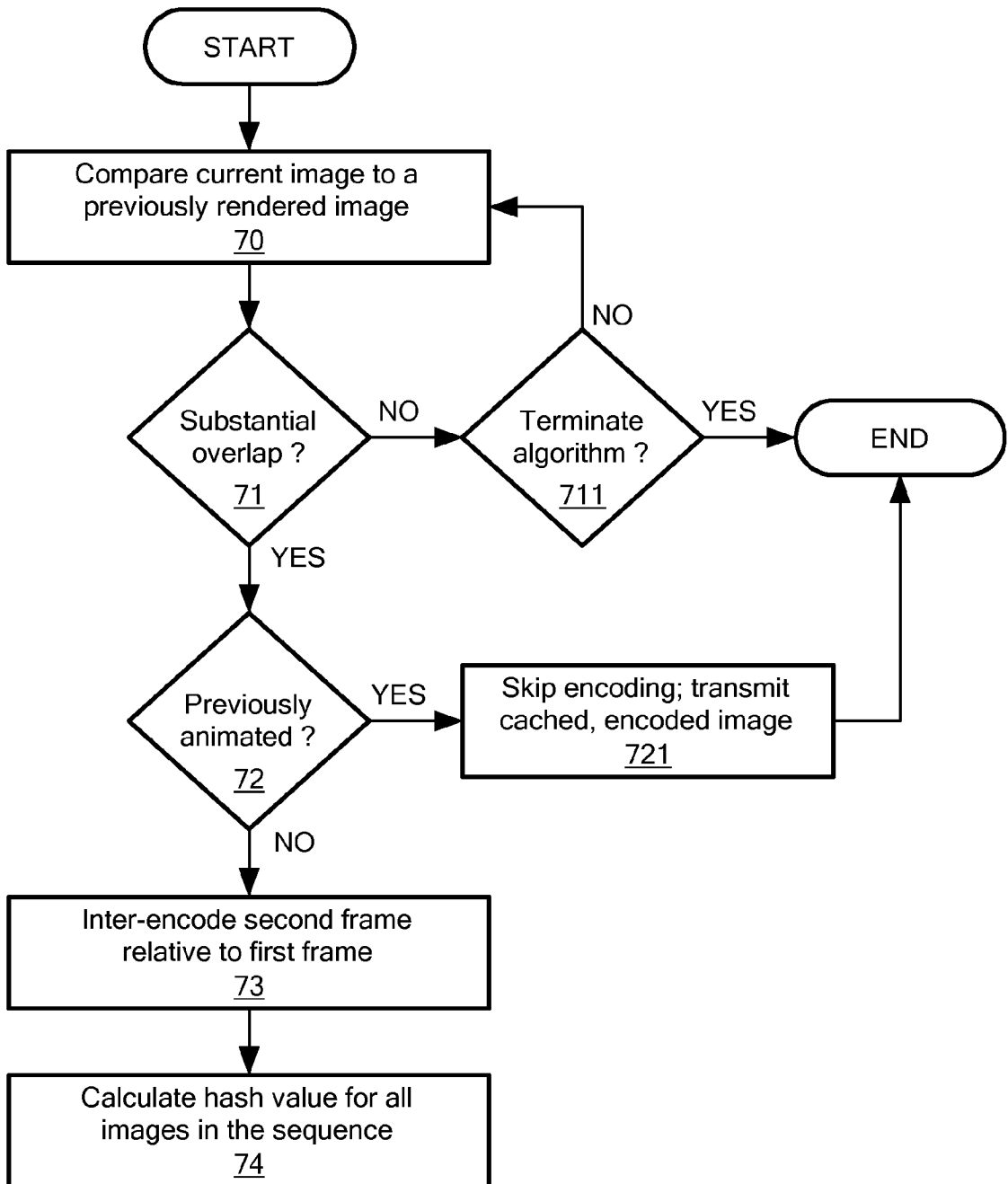


**FIG. 6D**

10/18

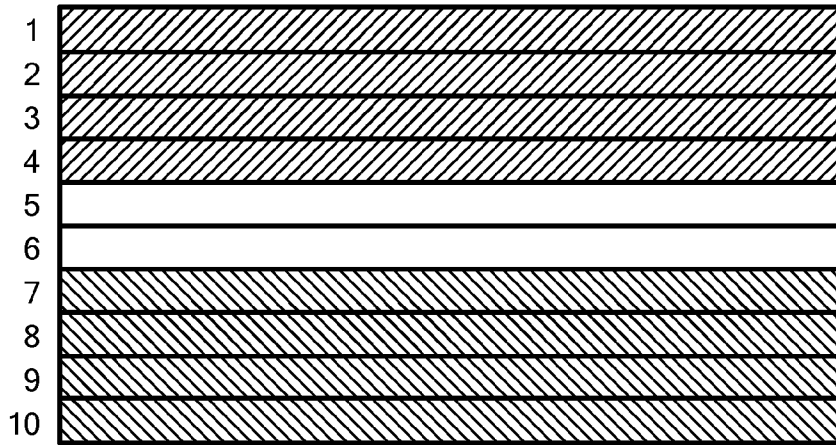


***FIG. 6E***

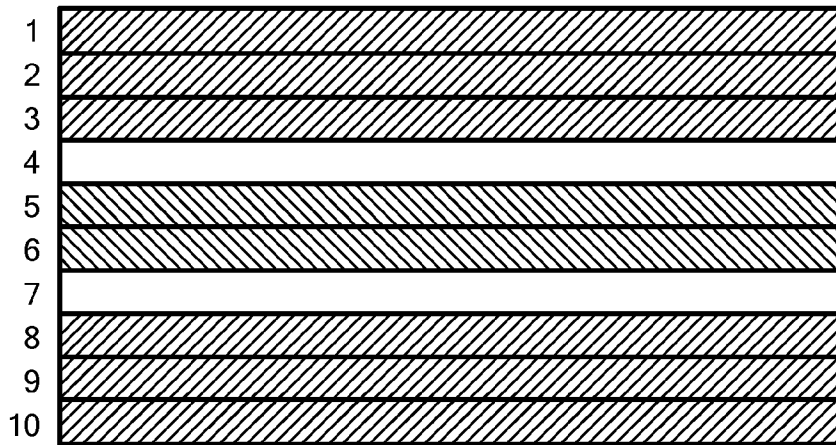


**FIG. 7**

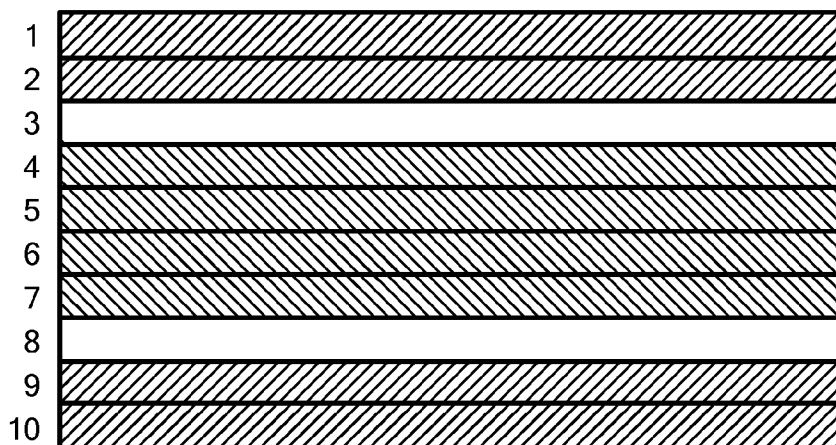
12/18



**FIG. 8A**



**FIG. 8B**



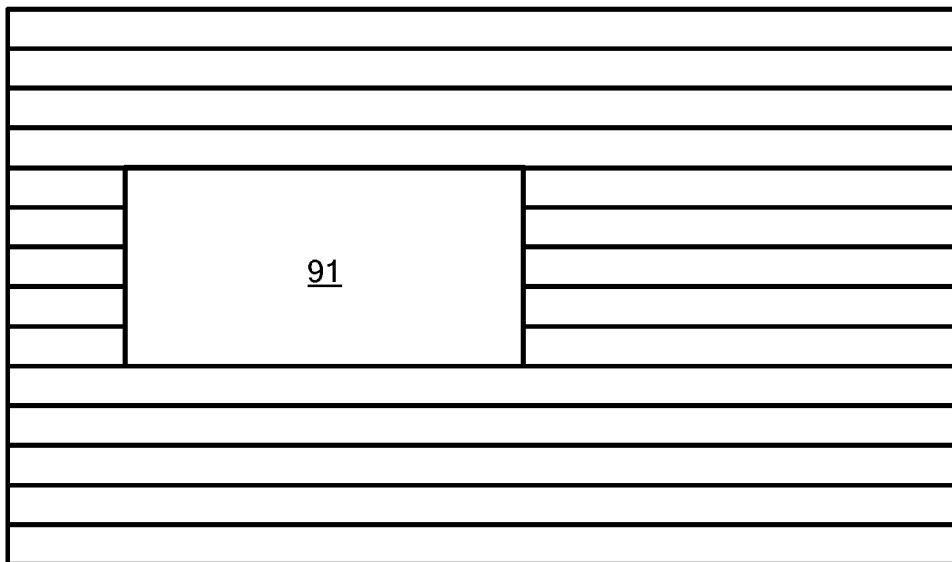
**FIG. 8C**



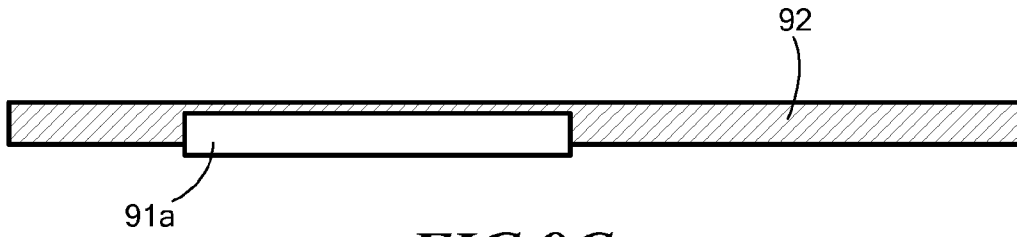
13/18



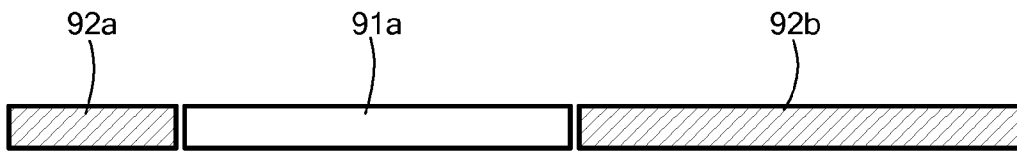
***FIG. 9A***



***FIG. 9B***



**FIG. 9C**

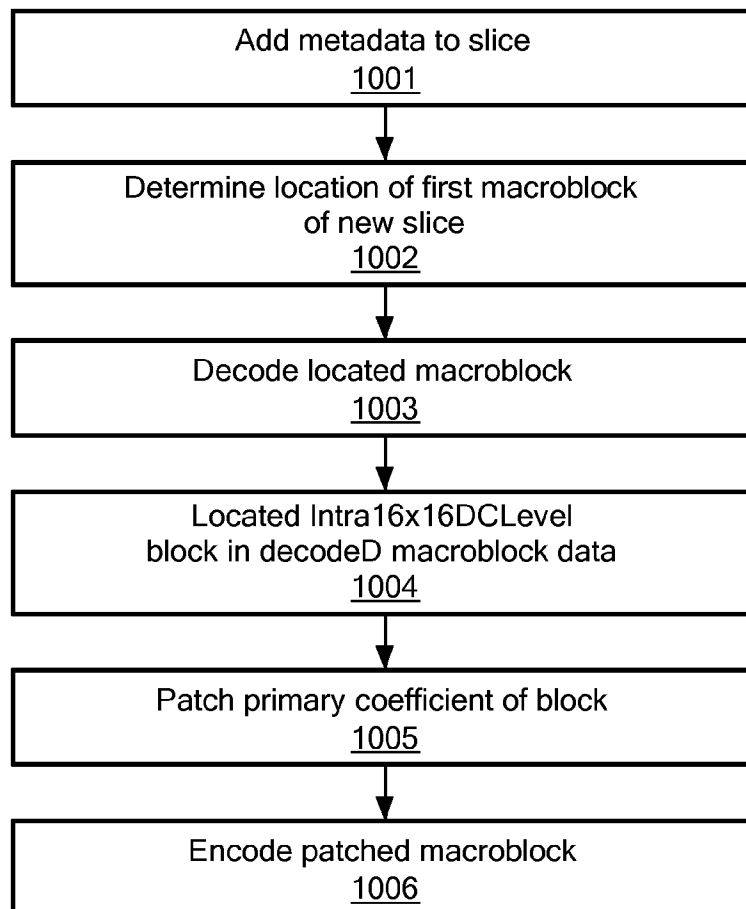


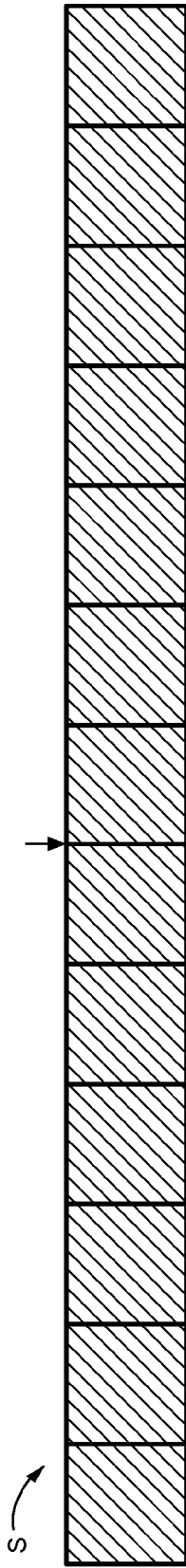
**FIG. 9D**



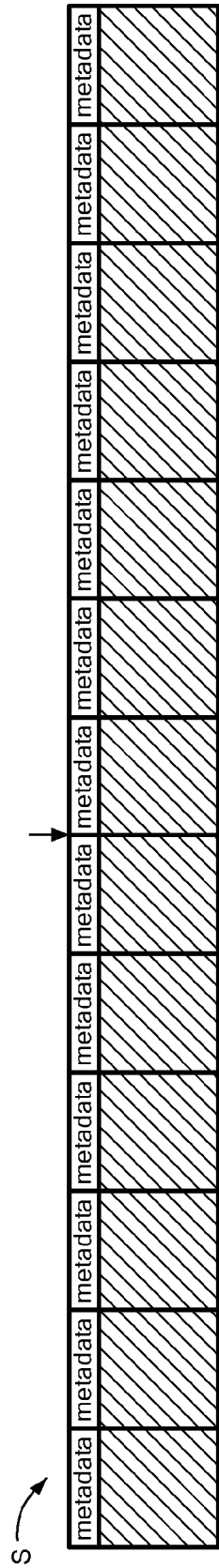
**FIG. 9E**

15/18

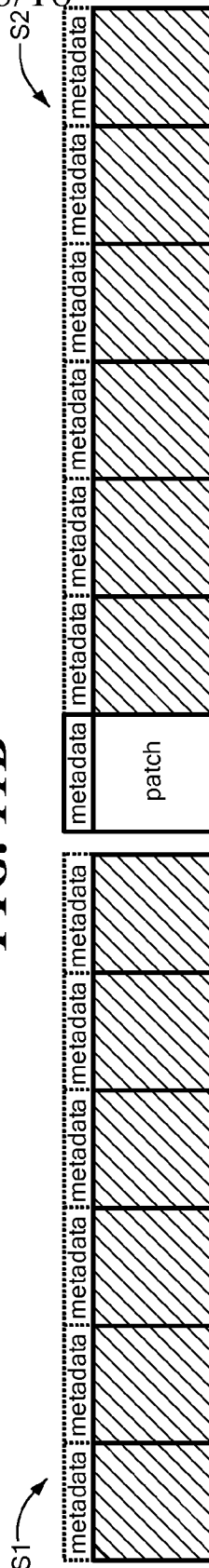
**FIG. 10**



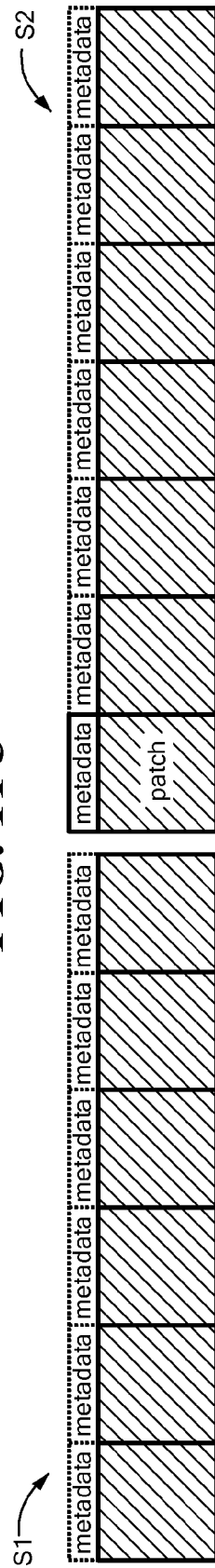
**FIG. 11A**



**FIG. 11B**

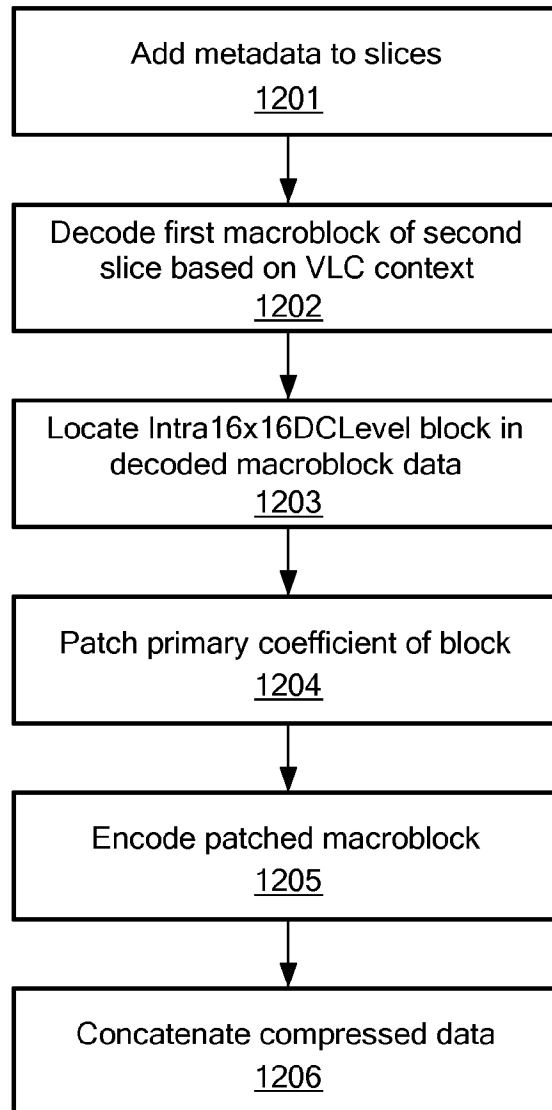


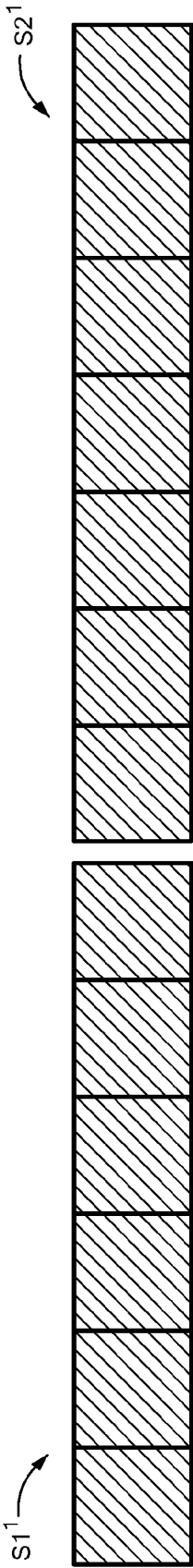
**FIG. 11C**



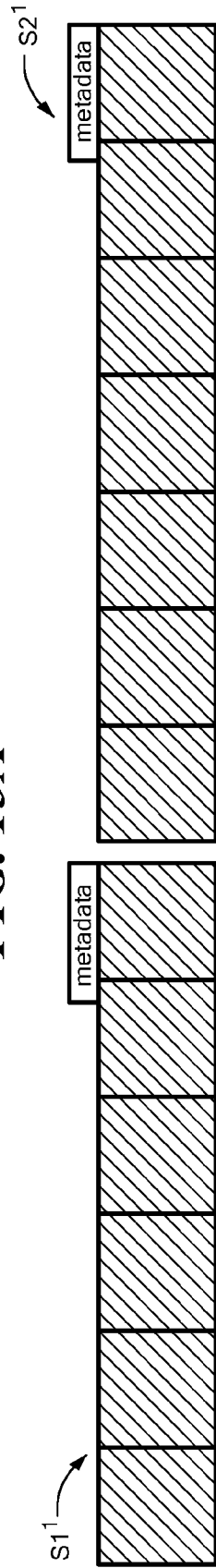
**FIG. 11D**

17/18

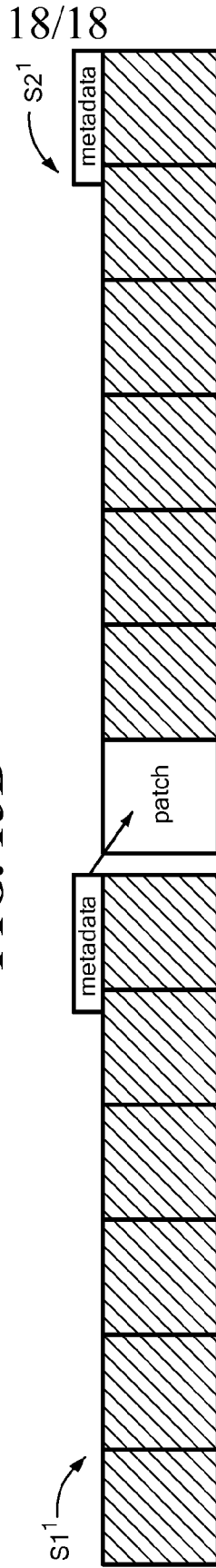
**FIG.12**



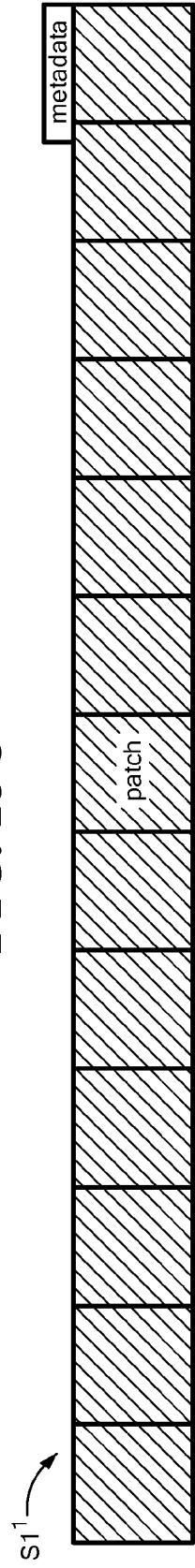
**FIG. 13A**



**FIG. 13B**



**FIG. 13C**



**FIG. 13D**

**A. CLASSIFICATION OF SUBJECT MATTER****G06T 1/00(2006.01)i, G06T 9/00(2006.01)i**

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)

G06T 1/00; G06F 15/16; G06F 15/00; G06F 17/00; G06T 15/00; H04B 1/66; G06T 9/00

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

**eKOMPASS(KIPO internal) & Keywords: paint instruction, hash value, rendering library, graphical object, audiovisual data, encoding, caching****C. DOCUMENTS CONSIDERED TO BE RELEVANT**

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 7830388 B1 (LU, YANG) 9 November 2010 See column 3, lines 57-67, column 7, lines 17-67; claims 1, 5, 6, 11; and figure 5.	1-33
A	US 2009-0003446 A1 (WU, YOUNGJUN et al.) 1 January 2009 See paragraphs [0020], [0731]; claims 1-5; and figure 53.	1-33
A	US 2009-0189890 A1 (CORBETT, TIM et al.) 30 July 2009 See paragraphs [0008]-[0024], [0042], [0105], [0118]-[0125]; and figures 4-7.	1-33
A	US 8136033 B1 (BHARGAVA, GAURAV et al.) 13 March 2012 See column 8, line 9 - column 10, line 58; claims 1-3; and figures 2-3.	1-33
A	US 2006-0242570 A1 (CROFT, LAWRENCE et al.) 26 October 2006 See paragraphs [0075]-[0077], [0083]-[0084], [0202]-[0205]; and figures 3-4, 22.	1-33
A	US 2010-0146139 A1 (BROCKMANN, RONALD) 10 June 2010 See paragraphs [0096]-[0100]; and figure 5.	1-33



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 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

"&amp;" document member of the same patent family


Date of the actual completion of the international search

26 July 2013 (26.07.2013)

Date of mailing of the international search report

**29 July 2013 (29.07.2013)**

Name and mailing address of the ISA/KR


 Korean Intellectual Property Office  
 189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City,  
 302-701, Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

HWANG Yun Koo

Telephone No. +82-42-481-5715



**Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)**

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1.  Claims Nos.:  
because they relate to subject matter not required to be searched by this Authority, namely:
  
2.  Claims Nos.:  
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
  
3.  Claims Nos.:  
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

**Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)**

This International Searching Authority found multiple inventions in this international application, as follows:

This ISA found multiple inventions as follows:

Group I, claims 1-10 and 15-33, drawn to a method, a medium, and a system of providing an image to a client device comprising the hash value and the encoded audiovisual data.

Group II, claims 11-14, drawn to a method of forming two encoded slices, comprising altering a DC luma value or a DC chroma value.

The inventions listed as Groups I and II do not relate to a single general inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2 they lack the same or corresponding special technical features for the following reasons; they are separate inventions with distinct fields of search.

1.  As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2.  As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3.  As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
  
4.  No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

**Remark on Protest**

- The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- No protest accompanied the payment of additional search fees.



**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.

**PCT/US2013/036182**

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 7830388 B1	09/11/2010	US 2009-0305790 A1	10/12/2009
US 2009-0003446 A1	01/01/2009	None	
US 2009-0189890 A1	30/07/2009	AU 2009-206251 A1 CA 2700225 A1 CN 101918921 A EP 2245536 A2 EP 2293192 A2 EP 2293192 A3 EP 2315122 A2 EP 2315122 A3 EP 2315123 A2 EP 2315123 A3 EP 2315124 A2 EP 2315124 A3 IL 205332 D0 US 2009-0189891 A1 US 2009-0189892 A1 US 2009-0189893 A1 US 2009-0189894 A1 US 2012-218260 A1 US 8169436 B2 US 8350863 B2 US 8405654 B2 WO 2009-094673 A2 WO 2009-094673 A3	30/07/2009 30/07/2009 15/12/2010 03/11/2010 09/03/2011 04/05/2011 27/04/2011 04/05/2011 27/04/2011 11/05/2011 27/04/2011 04/05/2011 30/12/2010 30/07/2009 30/07/2009 30/07/2009 30/07/2009 30/08/2012 01/05/2012 08/01/2013 26/03/2013 30/07/2009 01/10/2009
US 8136033 B1	13/03/2012	None	
US 2006-0242570 A1	26/10/2006	EP 1741038 A1 EP 1875374 A1 JP 2007-535044 A JP 2008-539479 A JP 4448537 B2 JP 4448549 B2 US 2005-0240858 A1 US 2005-0262430 A1 US 2009-0193331 A1 US 2009-0222719 A1 US 7536636 B2 US 7555712 B2 US 7694217 B2 US 7925969 B2 US 8032824 B2 WO 2005-103935 A1 WO 2006-113989 A1	10/01/2007 09/01/2008 29/11/2007 13/11/2008 14/04/2010 14/04/2010 27/10/2005 24/11/2005 30/07/2009 03/09/2009 19/05/2009 30/06/2009 06/04/2010 12/04/2011 04/10/2011 03/11/2005 02/11/2006
US 2010-0146139 A1	10/06/2010	EP 2105019 A2	30/09/2009

**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.

**PCT/US2013/036182**

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
		EP 2477414 A2	18/07/2012
		EP 2487919 A2	15/08/2012
		JP 2010-505330 A	18/02/2010
		WO 2008-044916 A2	17/04/2008
		WO 2008-044916 A3	16/04/2009
		WO 2008-044916 A8	31/07/2008