



- (51) **International Patent Classification:**
G06T 1/00 (2006.01)
- (21) **International Application Number:**
PCT/CN2015/081148
- (22) **International Filing Date:**
10 June 2015 (10.06.2015)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (71) **Applicant:** INTEL CORPORATION [US/US]; 2200 Mission College Blvd., Santa Clara, California 95054 (US).
- (72) **Inventors:** YANG, Jin; Room 901, No. 1 Building, Lane 599, Zhoudong Nan Road, Shanghai 200241 (CN). HAN, Junchao; Room 503, No 4 Building, Lane 1041, Hongqiao Road, Shanghai 200240 (CN). JIANG, Zidong; Room 204, Building 81, Zhenhua Road 1801, Shanghai 200000 (CN). LE, Yongnian; Room 1801, No. 11 Building, Lane 29, Puxiong Road, Shanghai 200233 (CN).
- (74) **Agent:** CHINA PATENT AGENT (H.K.) LTD.; 22/F., Great Eagle Center, 23 Harbour Road, Wanchai, Hong Kong (CN).
- (81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM,

AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

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

(54) **Title:** WEBGL APPLICATION ANALYZER

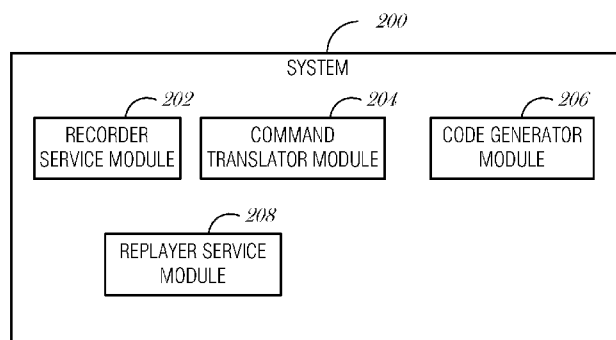


FIG. 2

(57) **Abstract:** Various systems and methods for analyzing WebGL applications are described herein. A system comprises a recorder service module to intercept a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context; a command translator module to translate the plurality of graphics API functions calls to a set of generic API invocations; a code generator module to generate executable code from the set of generic API invocations; and a replayer service module to test the executable code.



WEBGL APPLICATION ANALYZER

5 TECHNICAL FIELD

[0001] Embodiments described herein generally relate to software performance profiling and in particular, to a system for analyzing WebGL applications.

10 BACKGROUND

[0002] HTML5 is a markup language that builds on the technologies provided in HTML 4.01 and XHTML 1.1. WebGL (Web Graphics Library) is a JavaScript application programming interface (API) for rendering computer graphics within a web browser without the use of a plug-in. WebGL is an integrated technology
15 in HTML5. Through the use of HTML5's canvas object and scripts that use the WebGL API, developers are able to provide rich two-dimensional (2D) and three-dimensional (3D) graphics.

BRIEF DESCRIPTION OF THE DRAWINGS

20 [0003] In the drawings, which are not necessarily drawn to scale, like numerals may describe similar components in different views. Like numerals having different letter suffixes may represent different instances of similar components. Some embodiments are illustrated by way of example, and not limitation, in the figures of the accompanying drawings in which:

25 [0004] FIG. 1 is a flowchart illustrating control and data flow during operation, according to an embodiment;

[0005] FIG. 2 is a block diagram illustrating a system for analyzing WebGL applications, according to an embodiment;

[0006] FIG. 3 is a flowchart illustrating a method of analyzing WebGL
30 applications, according to an embodiment; and

[0007] FIG. 4 is a block diagram illustrating an example machine upon which any one or more of the techniques (e.g., methodologies) discussed herein may perform, according to an example embodiment.

DETAILED DESCRIPTION

[0008] Systems and methods described herein provide mechanisms for analyzing WebGL applications. A WebGL application runs in a web browser.

5 [0009] With the WebGL JavaScript API, developers are able to create fantastic 3D games or applications. Because of the popularity and the inherent advantage of HTML5 (e.g., write once, run within many browsers), more developers are producing 3D games with either the WebGL API directly or with various WebGL-based game engines.

[0010] When a WebGL-based 3D application is not performing well, it is difficult to identify the source of the problem. Because each browser may implement its own
10 platform to handle WebGL commands, it may be difficult to determine whether a particular 3D application is limited by hardware (e.g., graphics processing unit (GPU)) or software (e.g., some WebGL implementation problem inside the browser).

[0011] To date, there are no existing tools for testing or optimizing WebGL programs. While some browsers provide programmers with tracing tools (e.g., Google
15 Chrome™ browser), such tools do not provide detailed insight into the WebGL program's operation independent of the browser's environment. Thus, for example, a tracing tool may indicate that a WebGL program is GPU bound, but may not be able to indicate whether the same WebGL program run on another browser would also be GPU bound. Browser implementations may change frequently, further complicating analysis.
20 To perform proper analysis, the graphics commands initiated by WebGL calls should be separately analyzed from graphics commands issued by the browser. Thus, there is a need to analyze a WebGL program independent from the underlying browser platform.

[0012] FIG. 1 is a flowchart illustrating control and data flow during
25 operation, according to an embodiment. A script 102 is executed within a browser 104 on a user device 106. The script 102 may be considered a WebGL program if it invokes at least one WebGL function from a WebGL API. WebGL programs includes control code, which may be written in JavaScript or another scripting language, and shader code that is executed on a GPU. While JavaScript
30 is presently used for WebGL, it is understood that any client-side scripting language may be adapted to use a WebGL API. For example, a VBScript library may be developed to provide a WebGL API to VBScript client-side scripts.

[0013] The script 102 executes within the browser 104. The browser 104 may be any web browser application capable of executing client-side scripts and
35 presenting visual images. Examples of browsers include, but are not limited to

Mozilla Firefox®, Google Chrome™, Apple Safari®, and Microsoft Internet Explorer®. The browser 104 executes on a user device 106. The user device 106 may be any type of compute device including, but not limited to an onboard vehicle system, set-top box, wearable device, personal computer (PC), a tablet
5 PC, a hybrid tablet, a personal digital assistant (PDA), a mobile telephone, or the like.

[0014] A user may or may not interact with the script 102 via the browser 104. In either case, a recorder service 108 is used to monitor and record graphics commands issued by the script 102. The recorder service 108 may exist in
10 various layers of the software stack, for example, in the scripting engine (e.g., JavaScript engine), in the browser as a native function or an extension, or as a standalone application. In an example, the recorder service 108 is implemented as a JavaScript module and is injected into the HTML page before the WebGL script 102 is executed.

[0015] The recorder service 108 records the graphics commands issued by the script 102, saving them to a data store 110. The recorder service 108 may buffer a number of API calls before dumping them to the data store 110. The recorder service 108 records the graphics commands and the associated context of the API call. The context of the API call may include object properties and attributes
15 at the time of the call, parameter values, values of global variables, values of local variables, or information on the function call stack or execution stack.

[0016] A command translator 112 parses the information stored in the data store 110 by the recorder service 108 and translates the graphics commands to a set of generic API invocations. The generic API invocations are in an
25 intermediate format, which is not specific to any programming language. In an example, the script 102 is a WebGL script that invokes a WebGL API, where the commands are captured and recorded by the recorder service 108 and later translated to an OpenGL command. The OpenGL command may be any of OpenGL command implementations, such as OpenGL ES 2.0.

[0017] By performing the translation, native OpenGL commands may be mapped to WebGL commands line by line so that developers may isolate and analyze browser implementations of graphics commands quickly and easily. For example, developers may utilize tools that already exist for analyzing OpenGL application to conduct functionality debugging and performance analysis with
30

the translated applications. Such analysis may provide insight into browser limitations or other implementation bottlenecks, which may impact the WebGL-based application.

[0018] Thus, after the command translator 112 parses the information in the data store 110 to output generic API invocations, a code generator 114 is used to interpret the intermediate format of generic API invocations and generate corresponding native OpenGL code. The code may be in a certain language for a certain OpenGL platform. For example, the code generator 114 may generate Android Java SDK-based OpenGL ES 2.0 code or pure native C++ code based on Android NDK. Alternatively, the code generator 114 may generate native OpenGL ES 2.0 C++ code for Linux or Windows platforms, or for other platforms.

[0019] Once the code is created, it is available to a replayer service 116. The replayer service 116 may be an Android Application Package (APK) on Android or an executable binary on Linux or Windows, for either an Intel platform or an ARM platform. Alternatively, the replayer service 116 may be implemented with a virtual machine that executes the code created by the code generator 114. The replayer service 116 is used to replay one or more commands frame-by-frame.

[0020] The command translator 112 and code generator 114 may be implemented as a single software object. In an example implementation, the command translator 112 and code generator 114 are implemented using a single Python module that translates WebGL commands into native OpenGL ES 2.0 C++ commands, and then wraps them into an Android project with the Make file. In this example implementation, the replayer service 116 may generate the Android native application for either an x86 or ARM platform for further debugging or performance analysis.

[0021] FIG. 2 is a block diagram illustrating a system 200 for analyzing WebGL applications, according to an embodiment. The system 200 includes a recorder service module 202, a command translator module 204, a code generator module 206, and a replayer service module 208. The recorder service module 202 may be configured to intercept a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context. The command translator module 204 may be configured to translate the plurality of graphics API

functions calls to a set of generic API invocations. The code generator module 206 may be configured to generate executable code from the set of generic API invocations. The replayer service module 208 may be configured to test the executable code.

5 **[0022]** In an embodiment, to intercept the plurality of graphics API function calls, the recorder service module 202 is to load a script in a browser page load operation, the script configured to record the plurality of graphics API function calls. In a further embodiment, to translate the plurality of graphics API functions calls to the set of generic API invocations, the command translator
10 module 204 is to perform the translation while the browser is executing the plurality of API function calls.

[0023] In an embodiment, the recorder service module 202 is to buffer a plurality of graphics API function calls into a buffer and save the plurality of graphics API function calls to a log when the buffer is full.

15 **[0024]** In an embodiment, the plurality of graphics API function calls are a subset of WebGL commands. In an embodiment, the set of generic API invocations are a subset of OpenGL ES 2.0 commands.

[0025] In an embodiment, to generate executable code from the set of generic API invocations, the code generator module 206 is to generate an executable
20 program for a target platform. In a further embodiment, the executable program is one of: a C program, a Java program, or a C++ program. In a further embodiment, the target platform is one of: a Windows® platform, an Android™ platform, or an Apple® platform.

[0026] In an embodiment, to test the executable code, the replayer service
25 module 208 is to compile the executable code into an executable program and test the executable program. In an embodiment, to test the executable code, the replayer service module 208 is to instantiate a virtual machine and test the executable code in the virtual machine. Testing may include using a profiler to compare the execution of the code under test against a reference code set.

30 Testing may be partially or fully automated to limit user interaction and intervention. The output from testing may include a report to highlight or identify sections of code that are operating at less than an optimal performance, causing bottlenecks, sections that have high processor (e.g., CPU or GPU) utilization, high memory utilization, or the like.

[0027] FIG. 3 is a flowchart illustrating a method 300 of analyzing WebGL applications, according to an embodiment. At block 302, on a computing platform, a plurality of graphics application programming interface (API) function calls are intercepted, each of the plurality of graphics API functions
5 calls having an associated execution context. At block 304, the plurality of graphics API functions calls are translated to a set of generic API invocations. At block 306, executable code is generated from the set of generic API invocations and at block 308, the executable code is tested.

[0028] In an embodiment, intercepting the plurality of graphics API function
10 calls comprises loading a script in a browser page load operation, the script configured to record the plurality of graphics API function calls. In a further embodiment, translating the plurality of graphics API functions calls to the set of generic API invocations occurs while the browser is executing the plurality of API function calls.

[0029] In an embodiment, the method 300 further comprises buffering a
15 plurality of graphics API function calls into a buffer and saving the plurality of graphics API function calls to a log when the buffer is full.

[0030] In an embodiment, the plurality of graphics API function calls are a
20 subset of WebGL commands. In an embodiment, the set of generic API invocations are a subset of OpenGL ES 2.0 commands.

[0031] In an embodiment, generating executable code from the set of generic
API invocations comprises generating an executable program for a target platform. In an embodiment, the executable program is one of: a C program, a Java program, or a C++ program. In an embodiment, the target platform is one
25 of: a Windows platform, an Android platform, or an Apple platform.

[0032] In an embodiment, testing the executable code comprise compiling the
executable code into an executable program and testing the executable program. In an embodiment, testing the executable code comprises instantiating a virtual machine and testing the executable code in the virtual machine.

[0033] Embodiments may be implemented in one or a combination of
30 hardware, firmware, and software. Embodiments may also be implemented as instructions stored on a machine-readable storage device, which may be read and executed by at least one processor to perform the operations described herein. A machine-readable storage device may include any non-transitory mechanism for

storing information in a form readable by a machine (e.g., a computer). For example, a machine-readable storage device may include read-only memory (ROM), random-access memory (RAM), magnetic disk storage media, optical storage media, flash-memory devices, and other storage devices and media.

- 5 **[0034]** Examples, as described herein, may include, or may operate on, logic or a number of components, modules, or mechanisms. Modules may be hardware, software, or firmware communicatively coupled to one or more processors in order to carry out the operations described herein. Modules may be hardware modules, and as such modules may be considered tangible entities
- 10 capable of performing specified operations and may be configured or arranged in a certain manner. In an example, circuits may be arranged (e.g., internally or with respect to external entities such as other circuits) in a specified manner as a module. In an example, the whole or part of one or more computer systems (e.g., a standalone, client or server computer system) or one or more hardware
- 15 processors may be configured by firmware or software (e.g., instructions, an application portion, or an application) as a module that operates to perform specified operations. In an example, the software may reside on a machine-readable medium. In an example, the software, when executed by the underlying hardware of the module, causes the hardware to perform the specified
- 20 operations. Accordingly, the term hardware module is understood to encompass a tangible entity, be that an entity that is physically constructed, specifically configured (e.g., hardwired), or temporarily (e.g., transitorily) configured (e.g., programmed) to operate in a specified manner or to perform part or all of any operation described herein. Considering examples in which modules are
- 25 temporarily configured, each of the modules need not be instantiated at any one moment in time. For example, where the modules comprise a general-purpose hardware processor configured using software; the general-purpose hardware processor may be configured as respective different modules at different times. Software may accordingly configure a hardware processor, for example, to
- 30 constitute a particular module at one instance of time and to constitute a different module at a different instance of time. Modules may also be software or firmware modules, which operate to perform the methodologies described herein.

[0035] FIG. 4 is a block diagram illustrating a machine in the example form of a computer system 400, within which a set or sequence of instructions may be executed to cause the machine to perform any one of the methodologies discussed herein, according to an example embodiment. In alternative
5 embodiments, the machine operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine may operate in the capacity of either a server or a client machine in server-client network environments, or it may act as a peer machine in peer-to-peer (or distributed) network environments. The machine may be an onboard vehicle
10 system, set-top box, wearable device, personal computer (PC), a tablet PC, a hybrid tablet, a personal digital assistant (PDA), a mobile telephone, or any machine capable of executing instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of
15 machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein. Similarly, the term “processor-based system” shall be taken to include any set of one or more machines that are controlled by or operated by a processor (e.g., a computer) to individually or jointly execute instructions to perform any one or
20 more of the methodologies discussed herein.

[0036] Example computer system 400 includes at least one processor 402 (e.g., a central processing unit (CPU), a graphics processing unit (GPU) or both, processor cores, compute nodes, etc.), a main memory 404 and a static memory 406, which communicate with each other via a link 408 (e.g., bus). The
25 computer system 400 may further include a video display unit 410, an alphanumeric input device 412 (e.g., a keyboard), and a user interface (UI) navigation device 414 (e.g., a mouse). In one embodiment, the video display unit 410, input device 412 and UI navigation device 414 are incorporated into a touch screen display. The computer system 400 may additionally include a storage
30 device 416 (e.g., a drive unit), a signal generation device 418 (e.g., a speaker), a network interface device 420, and one or more sensors (not shown), such as a global positioning system (GPS) sensor, compass, accelerometer, or other sensor.

[0037] The storage device 416 includes a machine-readable medium 422 on which is stored one or more sets of data structures and instructions 424 (e.g., software) embodying or utilized by any one or more of the methodologies or functions described herein. The instructions 424 may also reside, completely or at least partially, within the main memory 404, static memory 406, and/or within the processor 402 during execution thereof by the computer system 400, with the main memory 404, static memory 406, and the processor 402 also constituting machine-readable media.

[0038] While the machine-readable medium 422 is illustrated in an example embodiment to be a single medium, the term “machine-readable medium” may include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more instructions 424. The term “machine-readable medium” shall also be taken to include any tangible medium that is capable of storing, encoding or carrying instructions for execution by the machine and that cause the machine to perform any one or more of the methodologies of the present disclosure or that is capable of storing, encoding or carrying data structures utilized by or associated with such instructions. The term “machine-readable medium” shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media. Specific examples of machine-readable media include non-volatile memory, including but not limited to, by way of example, semiconductor memory devices (e.g., electrically programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM)) and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks.

[0039] The instructions 424 may further be transmitted or received over a communications network 426 using a transmission medium via the network interface device 420 utilizing any one of a number of well-known transfer protocols (e.g., HTTP). Examples of communication networks include a local area network (LAN), a wide area network (WAN), the Internet, mobile telephone networks, plain old telephone (POTS) networks, and wireless data networks (e.g., Wi-Fi, 3G, and 4G LTE/LTE-A or WiMAX networks). The term “transmission medium” shall be taken to include any intangible medium that is

capable of storing, encoding, or carrying instructions for execution by the machine, and includes digital or analog communications signals or other intangible medium to facilitate communication of such software.

5 Additional Notes & Examples:

- [0040] Example 1 includes subject matter for analyzing WebGL applications (such as a device, apparatus, or machine) comprising: a recorder service module to intercept a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context; a command translator module to translate the plurality of graphics API functions calls to a set of generic API invocations; a code generator module to generate executable code from the set of generic API invocations; and a replayer service module to test the executable code.
- 10 [0041] In Example 2, the subject matter of Example 1 may include, wherein to intercept the plurality of graphics API function calls, the recorder service module is to load a script in a browser page load operation, the script configured to record the plurality of graphics API function calls.
- [0042] In Example 3, the subject matter of any one of Examples 1 to 2 may include, wherein to translate the plurality of graphics API functions calls to the set of generic API invocations, the command translator module is to perform the translation while the browser is executing the plurality of API function calls.
- 20 [0043] In Example 4, the subject matter of any one of Examples 1 to 3 may include, wherein the recorder service module is to: buffer a plurality of graphics API function calls into a buffer; and save the plurality of graphics API function calls to a log when the buffer is full.
- [0044] In Example 5, the subject matter of any one of Examples 1 to 4 may include, wherein the plurality of graphics API function calls are a subset of WebGL commands.
- [0045] In Example 6, the subject matter of any one of Examples 1 to 5 may include, wherein the set of generic API invocations are a subset of OpenGL ES 2.0 commands.
- 30 [0046] In Example 7, the subject matter of any one of Examples 1 to 6 may include, wherein to generate executable code from the set of generic API

invocations, the code generator module is to generate an executable program for a target platform.

[0047] In Example 8, the subject matter of any one of Examples 1 to 7 may include, wherein the executable program is one of: a C program, a Java program,
5 or a C++ program.

[0048] In Example 9, the subject matter of any one of Examples 1 to 8 may include, wherein the target platform is one of: a Windows platform, an Android platform, or an Apple platform.

[0049] In Example 10, the subject matter of any one of Examples 1 to 9 may
10 include, wherein to test the executable code, the replayer service module is to: compile the executable code into an executable program; and test the executable program.

[0050] In Example 11, the subject matter of any one of Examples 1 to 10 may include, wherein to test the executable code, the replayer service module is to:
15 instantiate a virtual machine; and test the executable code in the virtual machine.

[0051] Example 12 includes subject matter for analyzing WebGL applications (such as a method, means for performing acts, machine readable medium including instructions that when performed by a machine cause the machine to performs acts, or an apparatus to perform) comprising: intercepting, on a
20 computing platform, a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context; translating the plurality of graphics API functions calls to a set of generic API invocations; generating executable code from the set of generic API invocations; and testing the executable code.

[0052] In Example 13, the subject matter of Example 12 may include, wherein the intercepting the plurality of graphics API function calls comprises loading a script in a browser page load operation, the script configured to record the plurality of graphics API function calls.

[0053] In Example 14, the subject matter of any one of Examples 12 to 13
30 may include, wherein translating the plurality of graphics API functions calls to the set of generic API invocations occurs while the browser is executing the plurality of API function calls.

[0054] In Example 15, the subject matter of any one of Examples 12 to 14 may include, wherein the method further comprises: buffering a plurality of

graphics API function calls into a buffer; and saving the plurality of graphics API function calls to a log when the buffer is full.

5 [0055] In Example 16, the subject matter of any one of Examples 12 to 15 may include, wherein the plurality of graphics API function calls are a subset of WebGL commands.

[0056] In Example 17, the subject matter of any one of Examples 12 to 16 may include, wherein the set of generic API invocations are a subset of OpenGL ES 2.0 commands.

10 [0057] In Example 18, the subject matter of any one of Examples 12 to 17 may include, wherein generating executable code from the set of generic API invocations comprises generating an executable program for a target platform.

[0058] In Example 19, the subject matter of any one of Examples 12 to 18 may include, wherein the executable program is one of: a C program, a Java program, or a C++ program.

15 [0059] In Example 20, the subject matter of any one of Examples 12 to 19 may include, wherein the target platform is one of: a Windows platform, an Android platform, or an Apple platform.

20 [0060] In Example 21, the subject matter of any one of Examples 12 to 20 may include, wherein testing the executable code comprises: compiling the executable code into an executable program; and testing the executable program.

[0061] In Example 22, the subject matter of any one of Examples 12 to 21 may include, wherein testing the executable code comprises: instantiating a virtual machine; and testing the executable code in the virtual machine.

25 [0062] Example 23 includes at least one machine-readable medium including instructions, which when executed by a machine, cause the machine to perform operations of any of the Examples 12-22.

[0063] Example 24 includes an apparatus comprising means for performing any of the Examples 12-22.

30 [0064] Example 25 includes subject matter for analyzing WebGL applications (such as a device, apparatus, or machine) comprising: means for intercepting, on a computing platform, a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context; means for translating the plurality of graphics API functions calls to a set of generic API invocations; means for generating

executable code from the set of generic API invocations; and means for testing the executable code.

[0065] In Example 26, the subject matter of Example 25 may include, wherein the means for intercepting the plurality of graphics API function calls
5 comprise means for loading a script in a browser page load operation, the script configured to record the plurality of graphics API function calls.

[0066] In Example 27, the subject matter of any one of Examples 25 to 26 may include, wherein the means for translating the plurality of graphics API functions calls to the set of generic API invocations are implemented while the
10 browser is executing the plurality of API function calls.

[0067] In Example 28, the subject matter of any one of Examples 25 to 27 may include, wherein the apparatus further comprises: means for buffering a plurality of graphics API function calls into a buffer; and means for saving the plurality of graphics API function calls to a log when the buffer is full.

15 [0068] In Example 29, the subject matter of any one of Examples 25 to 28 may include, wherein the plurality of graphics API function calls are a subset of WebGL commands.

[0069] In Example 30, the subject matter of any one of Examples 25 to 29 may include, wherein the set of generic API invocations are a subset of OpenGL
20 ES 2.0 commands.

[0070] In Example 31, the subject matter of any one of Examples 25 to 30 may include, wherein the means for generating executable code from the set of generic API invocations comprise means for generating an executable program for a target platform.

25 [0071] In Example 32, the subject matter of any one of Examples 25 to 31 may include, wherein the executable program is one of: a C program, a Java program, or a C++ program.

[0072] In Example 33, the subject matter of any one of Examples 25 to 32 may include, wherein the target platform is one of: a Windows platform, an
30 Android platform, or an Apple platform.

[0073] In Example 34, the subject matter of any one of Examples 25 to 33 may include, wherein the means for testing the executable code comprises: means for compiling the executable code into an executable program; and means for testing the executable program.

[0074] In Example 35, the subject matter of any one of Examples 25 to 34 may include, wherein the means for testing the executable code comprises: means for instantiating a virtual machine; and means for testing the executable code in the virtual machine.

- 5 [0075] The above detailed description includes references to the accompanying drawings, which form a part of the detailed description. The drawings show, by way of illustration, specific embodiments that may be practiced. These embodiments are also referred to herein as “examples.” Such examples may include elements in addition to those shown or described.
- 10 However, also contemplated are examples that include the elements shown or described. Moreover, also contemplated are examples using any combination or permutation of those elements shown or described (or one or more aspects thereof), either with respect to a particular example (or one or more aspects thereof), or with respect to other examples (or one or more aspects thereof)
- 15 shown or described herein.

[0076] Publications, patents, and patent documents referred to in this document are incorporated by reference herein in their entirety, as though individually incorporated by reference. In the event of inconsistent usages between this document and those documents so incorporated by reference, the

20 usage in the incorporated reference(s) are supplementary to that of this document; for irreconcilable inconsistencies, the usage in this document controls.

[0077] In this document, the terms “a” or “an” are used, as is common in patent documents, to include one or more than one, independent of any other

25 instances or usages of “at least one” or “one or more.” In this document, the term “or” is used to refer to a nonexclusive or, such that “A or B” includes “A but not B,” “B but not A,” and “A and B,” unless otherwise indicated. In the appended claims, the terms “including” and “in which” are used as the plain-English equivalents of the respective terms “comprising” and “wherein.” Also, in the

30 following claims, the terms “including” and “comprising” are open-ended, that is, a system, device, article, or process that includes elements in addition to those listed after such a term in a claim are still deemed to fall within the scope of that claim. Moreover, in the following claims, the terms “first,” “second,” and

“third,” etc. are used merely as labels, and are not intended to suggest a numerical order for their objects.

[0078] The above description is intended to be illustrative, and not restrictive.

For example, the above-described examples (or one or more aspects thereof)

- 5 may be used in combination with others. Other embodiments may be used, such as by one of ordinary skill in the art upon reviewing the above description. The Abstract is to allow the reader to quickly ascertain the nature of the technical disclosure. It is submitted with the understanding that it will not be used to interpret or limit the scope or meaning of the claims. Also, in the above Detailed
- 10 Description, various features may be grouped together to streamline the disclosure. However, the claims may not set forth every feature disclosed herein as embodiments may feature a subset of said features. Further, embodiments may include fewer features than those disclosed in a particular example. Thus, the following claims are hereby incorporated into the Detailed Description, with
- 15 a claim standing on its own as a separate embodiment. The scope of the embodiments disclosed herein is to be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

CLAIMS

What is claimed is:

1. A system for analyzing WebGL applications, the system comprising:
 - a recorder service module to intercept a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context;
 - a command translator module to translate the plurality of graphics API functions calls to a set of generic API invocations;
 - a code generator module to generate executable code from the set of generic API invocations; and
 - a replayer service module to test the executable code.
2. The system of claim 1, wherein to intercept the plurality of graphics API function calls, the recorder service module is to load a script in a browser page load operation, the script configured to record the plurality of graphics API function calls.
3. The system of claim 2, wherein to translate the plurality of graphics API functions calls to the set of generic API invocations, the command translator module is to perform the translation while the browser is executing the plurality of API function calls.
4. The system of claim 1, wherein the recorder service module is to:
 - buffer a plurality of graphics API function calls into a buffer; and
 - save the plurality of graphics API function calls to a log when the buffer is full.
5. The system of claim 1, wherein the plurality of graphics API function calls are a subset of WebGL commands.
6. The system of claim 1, wherein the set of generic API invocations are a subset of OpenGL ES 2.0 commands.

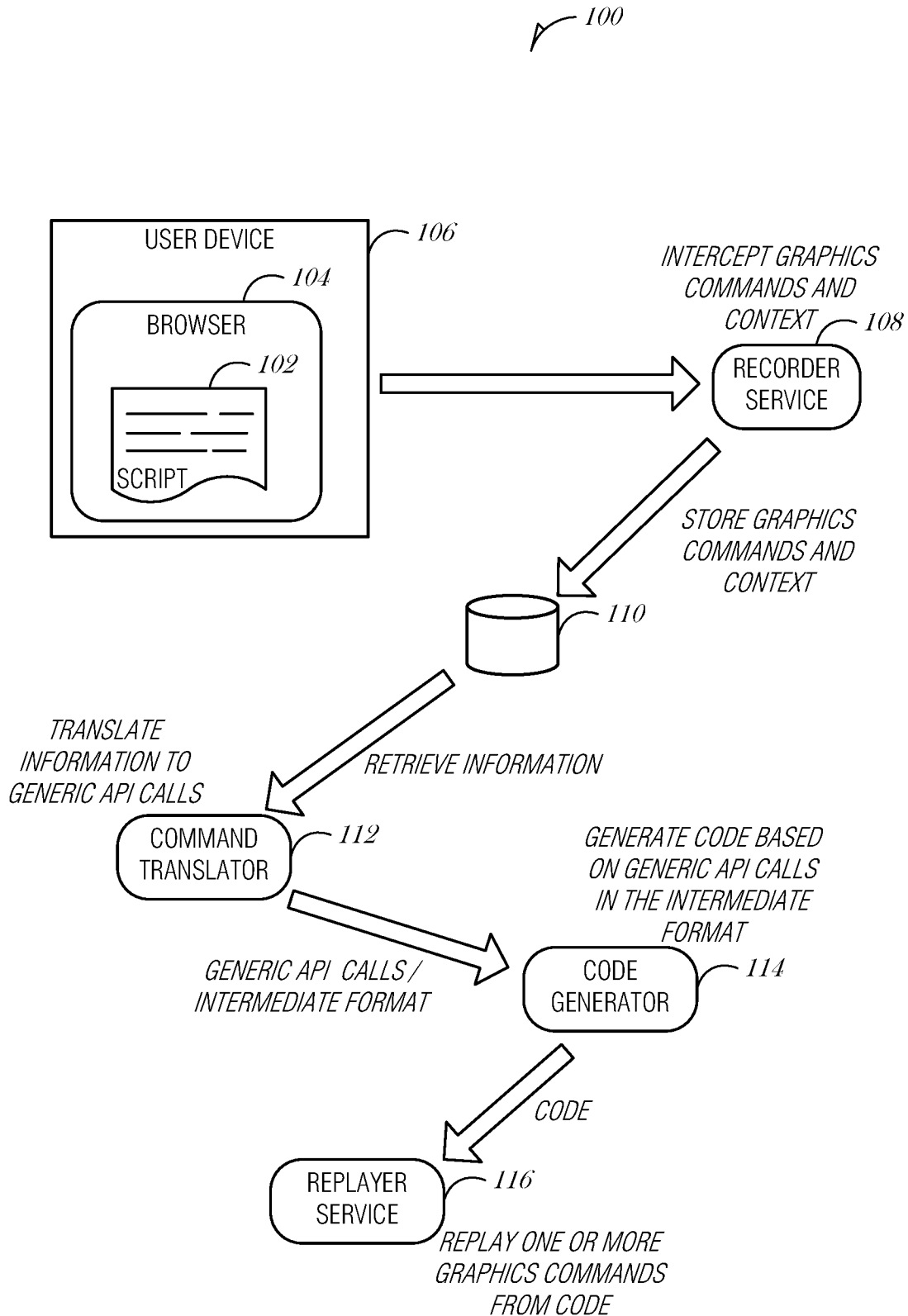
7. The system of claim 1, wherein to generate executable code from the set of generic API invocations, the code generator module is to generate an executable program for a target platform.
8. The system of claim 7, wherein the executable program is one of: a C program, a Java program, or a C++ program.
9. The system of claim 7, wherein the target platform is one of: a Windows platform, an Android platform, or an Apple platform.
10. The system of claim 1, wherein to test the executable code, the replayer service module is to:
 - compile the executable code into an executable program; and
 - test the executable program.
11. The system of claim 1, wherein to test the executable code, the replayer service module is to:
 - instantiate a virtual machine; and
 - test the executable code in the virtual machine.
12. A method of analyzing WebGL applications, the method comprising:
 - intercepting, on a computing platform, a plurality of graphics application programming interface (API) function calls, each of the plurality of graphics API functions calls having an associated execution context;
 - translating the plurality of graphics API functions calls to a set of generic API invocations;
 - generating executable code from the set of generic API invocations; and
 - testing the executable code.
13. The method of claim 12, wherein the intercepting the plurality of graphics API function calls comprises loading a script in a browser page load operation, the script configured to record the plurality of graphics API function calls.

14. The method of claim 13, wherein translating the plurality of graphics API functions calls to the set of generic API invocations occurs while the browser is executing the plurality of API function calls.
15. The method of claim 12, wherein the method further comprises:
buffering a plurality of graphics API function calls into a buffer; and
saving the plurality of graphics API function calls to a log when the buffer is full.
16. The method of claim 12, wherein the plurality of graphics API function calls are a subset of WebGL commands.
17. The method of claim 12, wherein the set of generic API invocations are a subset of OpenGL ES 2.0 commands.
18. The method of claim 12, wherein generating executable code from the set of generic API invocations comprises generating an executable program for a target platform.
19. The method of claim 18, wherein the executable program is one of: a C program, a Java program, or a C++ program.
20. The method of claim 18, wherein the target platform is one of: a Windows platform, an Android platform, or an Apple platform.
21. The method of claim 12, wherein testing the executable code comprises:
compiling the executable code into an executable program; and
testing the executable program.
22. The method of claim 12, wherein testing the executable code comprises:
instantiating a virtual machine; and
testing the executable code in the virtual machine.

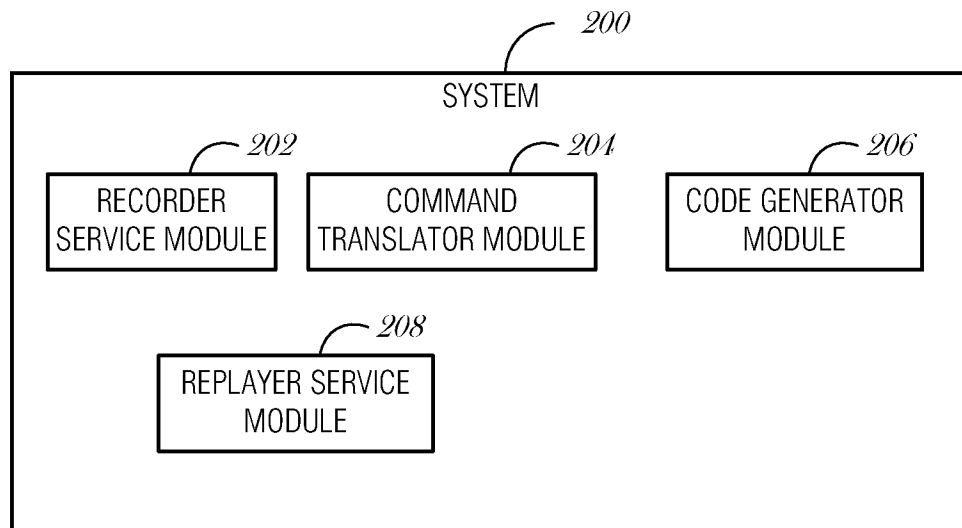
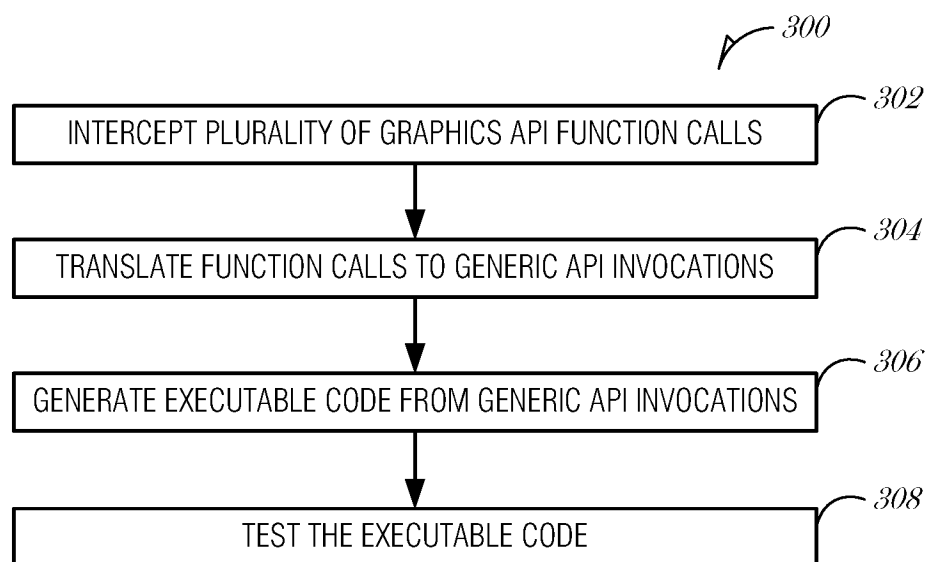
23. At least one machine-readable medium including instructions, which when executed by a machine, cause the machine to perform operations of any of the methods of claims 12-22.

24. An apparatus comprising means for performing any of the methods of claims 12-22.

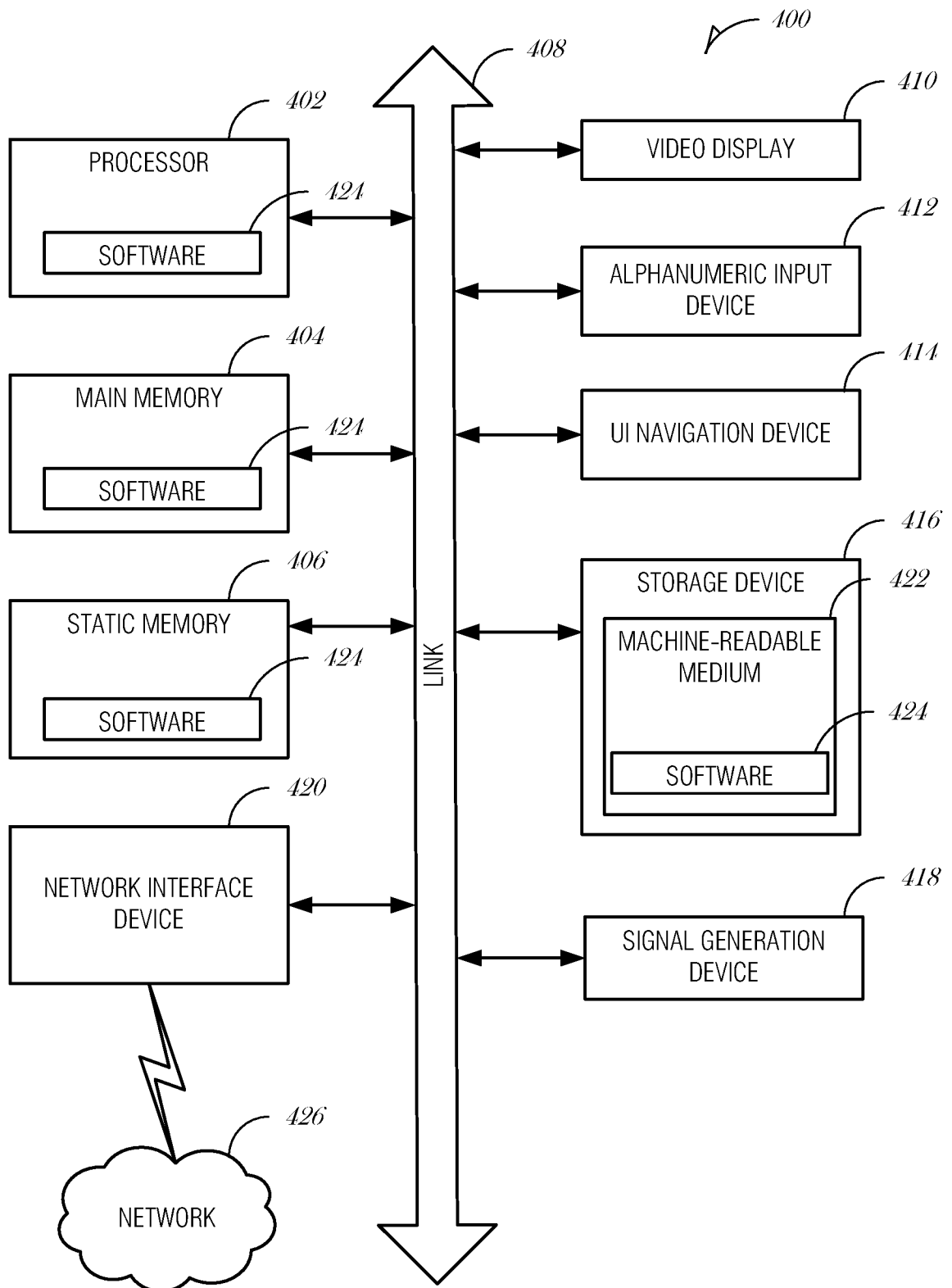
1/3

**FIG. 1**

2/3

**FIG. 2****FIG. 3**

3/3

**FIG. 4**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/CN2015/081148

A. CLASSIFICATION OF SUBJECT MATTER

G06T 1/00(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,G06F

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

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

CNPAT,WPI,EPODOC,IEEE,CNKI,GOOGLE:WebGL,API,application programming interface, execution,code,graphic, translate,transform,generate,test,analyze,script

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2013106877 A1 (WILLIAMS, ROY E.) 02 May 2013 (2013-05-02) claim 1, description, paragraphs [0030]-[0040]	1-24
X	US 2013227521 A1 (QUALCOMM INCORPORATED) 29 August 2013 (2013-08-29) description, paragraphs [0018]-[0119]	1-24
A	US 2013144566 A1 (DESIGN PLAY TECHNOLOGIES INC.) 06 June 2013 (2013-06-06) the whole document	1-24
A	CN 103608780 A (NOKIA CORPORATION) 26 February 2014 (2014-02-26) the whole document	1-24
A	US 2014130187 A1 (PRASHANT, DEWAN ET AL.) 08 May 2014 (2014-05-08) the whole document	1-24



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

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

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

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

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

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

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

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

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

“&” document member of the same patent family

Date of the actual completion of the international search

29 January 2016

Date of mailing of the international search report

24 February 2016

Name and mailing address of the ISA/CN

STATE INTELLECTUAL PROPERTY OFFICE OF THE
P.R.CHINA
6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing
100088, China

Facsimile No. (86-10)62019451

Authorized officer

QIN,Jingchan

Telephone No. (86-10)82245945

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.

PCT/CN2015/081148

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	2013106877	A1	02 May 2013	US	2013106880	A1	02 May 2013
				US	2013106878	A1	02 May 2013
				US	2013106879	A1	02 May 2013
				WO	2013062956	A1	02 May 2013
US	2013227521	A1	29 August 2013	WO	2013130212	A1	06 September 2013
				KR	20140139516	A	05 December 2014
				EP	2820544	A1	07 January 2015
				JP	2015511737	A	20 April 2015
				CN	104137076	A	05 November 2014
				US	2015261651	A1	17 September 2015
US	2013144566	A1	06 June 2013	WO	2013019961	A2	07 February 2013
CN	103608780	A	26 February 2014	US	2012324482	A1	20 December 2012
				EP	2721491	A1	23 April 2014
				WO	2012175786	A1	27 December 2012
US	2014130187	A1	08 May 2014	WO	2014074310	A1	15 May 2014
				GB	2523040	A	12 August 2015
				EP	2917869	A1	16 September 2015