ISBN: 978-981-18-1791-5
2021 the 11th International Workshop on Computer Science and Engineering (WCSE 2021)
doi: 10.18178/wcse.2021.06.003

B2GPAPI: an Efficient Framework Providing C/C++ Runtime in Web-based OS

Hao Xu[1★], Long Peng[1★], Tao Mao[2], Jun Ma[1], Shasha Li[1], Jie Yu[1+], Qingbo Wu[1]

[1] College of Computer, National University of Defense Technology, China
[2] Kylin Software Co., Ltd., China

Abstract. In the Internet era, mobile devices have become more and more popular because of their portability. Web-based operating system (OS) is a mobile operating system based on web technologies and has great potential for growth. However, mobile applications are mainly developed for Android and iOS, and cannot run in a Web-based OS due to the incompatibility between different platforms. Aiming to expand the application ecosystem of the Web-based OS and based on the fact that many mature applications' low-level code is written in C/C++, we propose B2GPAPI, a framework that provides a runtime for C/C++ applications in the Web-based OS. B2GPAPI makes it possible that existing mature mobile applications can directly run in a Web-based OS. To implement the framework, on the one hand, we employ the <object> html tag in web applications as a connection with the dynamic link library of the C/C++ application; on the other hand, in Web engine of the Web-based OS, we provide an event interface to deliver the event information from web application to the C/C++ application and a graphic rendering interface to display the interface of the C/C++ applications. In order to verify the efficiency of the framework, we compare B2GPAPI's graphics rendering interface with the WebGL interface. Experimental results show that B2GPAPI has good practicability and outperforms than WebGL.

Keywords: HTML5, WebGL, Open GL ES, C/C++, Web-based OS

1. Introduction

At present the number of applications for mobile devices dramatically increases along with the market demands in the Internet era. Compared with desktop applications, people prefer mobile applications that are more convenient and accessible. HTML5 is a new web standard proposed by W3C in 2014 [1], and compared with previous versions of HTML standards, it could support multimedia on mobile devices and greatly improve system performance, which provides a new solution for mobile application development. Applications developed under HTML5 can be deployed on all platforms, thus the uniform set of code is needed for maintenance and version update. Web-based operating system (OS) is a kind of mobile operating system, on which the application is developed by web languages including HTML5, JavaScript and CSS. The architecture of Web-based OS contains three layers: Application, Web engine and Kernel driver [2], as shown in Fig.1.

Application layer. This layer is composed of web applications and each web application can be regarded as a single page app (SPA) normally.

Web engine layer. This layer provides runtime and infrastructure for the application layer. It employs a browser engine and adds additional components, such as Web APIs [3], which makes that web applications can directly communicate with mobile hardware. This layer contains rendering engine, JavaScript engine, security module, and Web APIs etc.

Kernel driver layer. This layer is the bottom of the Web-based OS, which contains a Linux kernel and the user space hardware abstraction layer (HAL).

+ Corresponding author. Tel.: 86-13487592344;
E-mail address: yj@nudt.edu.cn
★ Indicates equal contribution

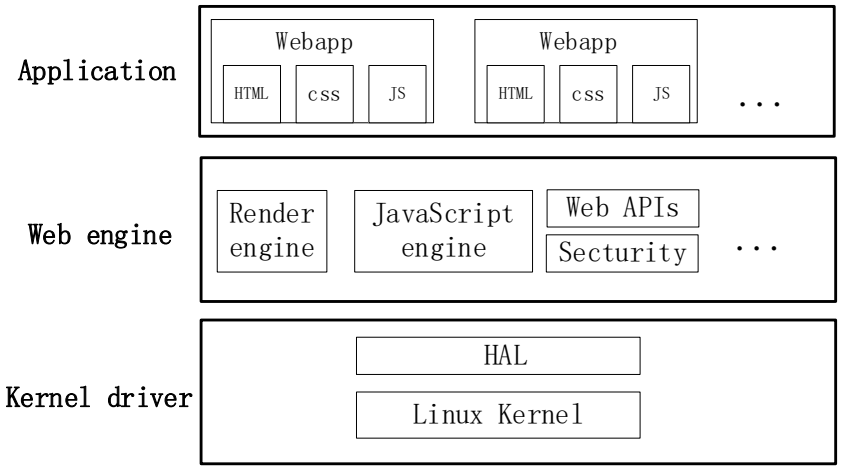

Fig. 1: The architecture of Web-based OS

However, the market of operating system for mobile device is almost wholly occupied by Android and iOS, and there are few applications developed for Web-based OS. It will be easier to expand the application ecosystem of the Web-based OS if mature applications developed for Android and iOS could be transplanted to Web-based OS. The work for transplanting is not easy due to the platform heterogeneity. While previous studies have shown that lots of applications (e.g., input method and the map application) are developed in an identical mode, of which the low-level implementation is written in C/C++ and being packaged as a dynamic library for third party calls. So, it would greatly reduce the workload for application transplant that we provide a C/C++ runtime in Web-based OS. Inspired by this, we propose B2GPAPI, a framework that provides a runtime for C/C++ applications in a Web-based OS, which aims to expand the application ecosystem of the Web-based OS.

B2GPAPI is implemented in two layers. 1) Application layer: The C/C++ application will be compiled as a dynamic link library. We use a web application in a Web-based OS as the container for C/C++ applications, and employ the <object> tag in html file as a connection with the dynamic link library of the C/C++ application. 2) Web engine layer: B2GPAPI is divided into two sides, i.e., B2G side for browser and Plugin side for C/C++ application. B2G side receives the event information from the web application and transfers them to Plugin side. Then the C/C++ application handles the information from Plugin side and draws the interface image by OpenGL ES [4]. The drawn image data will be read from the VRAM (Video Random Access Memory) to RAM by glReadPixels[2] () and be transferred from Plugin side to B2G side. In B2G side, the drawn image data will update the layer of the <object> tag in the layer tree. Eventually, the render thread in Web engine completes rendering <object> tag and the UI image of the C/C++ application will be eventually displayed.

Rendering of graphics is a complex and intensive work for any platform [5]. B2GPAPI provides a graphic rendering interface for C/C++ applications in the Web-based OS. To verify B2GPAPI's performance, we make a comparison with WebGL [6], a 3D graphic API based on OpenGL ES. We implement the same graphic rendering web application (including 2D and 3D) in the Web-based OS by B2GPAPI and WebGL respectively. The experimental results show that B2GPAPI performs better.

2. Related Work

Platform heterogeneity. We study three of the main mobile platforms (Android, iOS, Web-based OS) to show the incompatibility of applications between these platforms. Android is released by Google in 2007 and application for Android is written in JAVA. The Android platform provides not only the mobile operating system itself, but also a custom built virtual machine (Dalvik Virtual Machine [7]) for the applications to run

[2] glReadPixels () is a function in OpenGL ES that could read the drawn pixels from VRAM into RAM

on as well as acting as the middleware between code and operating system. iOS is the operating system for Apple devices, and the application for iOS is written in objective-C [8]. Web-based OS is an operating system on which the application developed by HTML, JavaScript，CSS. The Web-based OS series mainly include Firefox OS [9] that is proposed by Mozilla, and Kai OS that is popular in the low-end market. With the above description about Android and iOS, we learn that the Web-based OS does not provide the runtime for Android application which is written in JAVA and iOS application which is written in objective-C. Accordingly, application developed for Android and iOS cannot directly run in a Web-based OS.

NPAPI (Netscape Plugin Application Programming Interface). NPAPI [10] allows native code to be loaded and run as part of a web application. NPAPI is designed for the desktop browser and its graphic rendering interface relies on the GUI library for the desktop, such as Windows, Gtk, and MacOS X. When it comes to mobile OS, NPAPI is not involved in. It is feasible to provide a runtime for native code in mobile web application by means of the idea of NPAPI. Web-based OS is just right a mobile OS on which the application is developed by web, so the idea of NPAPI can be used for transplanting native application into a web application to expand the application ecosystem of Web-based OS. Inspired by this, we propose B2GPAPI.

The mode for mobile application development. There are currently three types of applications in the market: native application, web application and hybrid application. Different applications adapt to different scenarios: 1) Native application runs efficiently on the platform it was developed for, while it cannot run on other platforms; 2) Web application [11] is based on HTML5 and can achieve cross-platform using the browser, but there are limitations for the application by calling the bottom system's API; 3) Hybrid application [12, 13] takes the longest of both, and it could enable web application run on the native platform. On the one hand, the application could achieve cross-platform using web; on the other hand, it could call the native APIs of the platform. There are several frameworks for developing hybrid application such as PhoneGap [14], AppCan [15], and Cordova [16]. Lots of e-commerce applications are developed based on the mode of hybrid application.

B2GPAPI contributes a new type of application of which the low-level implementation is written in C/C++ while its runtime is in a web environment. The new type of application belongs to a hybrid application, while it is different from the traditional hybrid application, as shown in Fig.2. The traditional hybrid application provides a container for web application on the native platform, by which the web application could communicate with the mobile hardware. The container for web application is platform-dependent. By contrast, B2GPAPI provides a container for native applications on the web platform. Native application could be runnable on web platform by means of B2GPAPI.

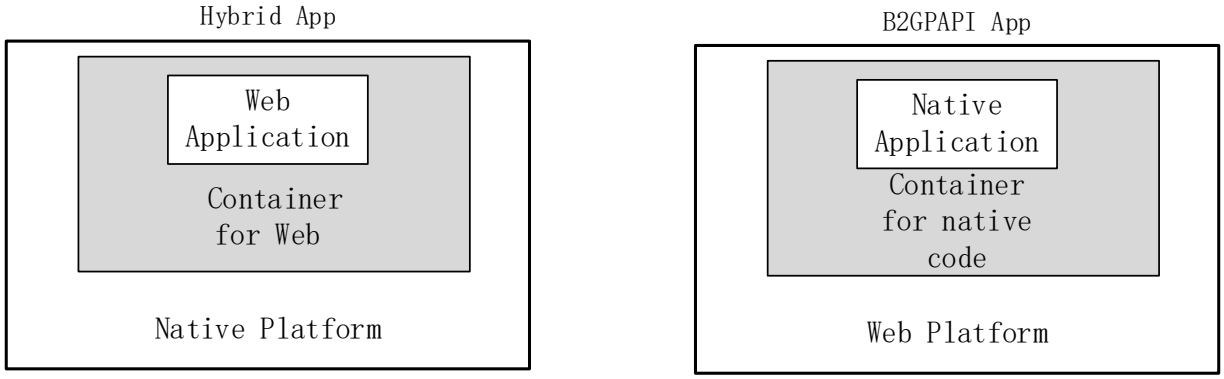

Fig. 2: Comparison between Hybrid app and B2GPAPI app

3. The B2GPAPI Framework

To enable the C/C++ application runnable in the Web-based OS, there are two works for B2GPAPI to be done, as shown in Fig.3.

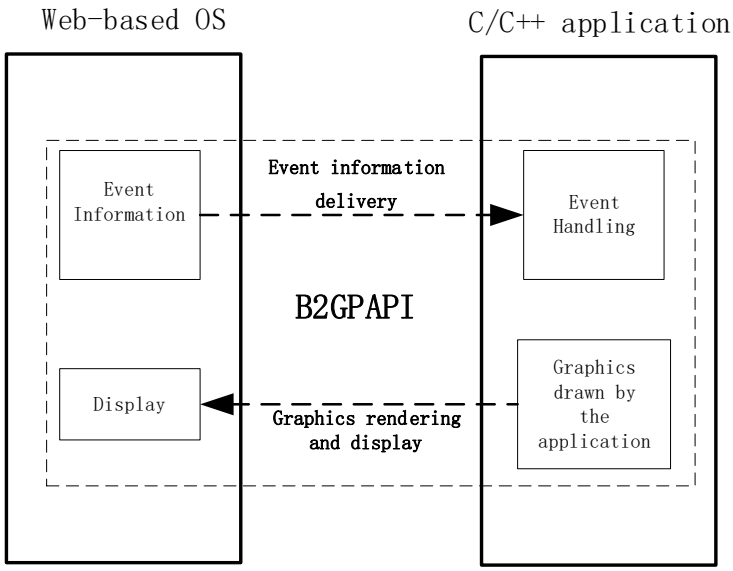

Fig. 3: The work for B2GPAPI to be done

Event information delivery. The user interacts with applications by touching the screen or other operations, and these operations are recognized as events by Web-based OS. Transferring these events to C/C++ applications for subsequent calculation or graphics drawing is to be done.

Graphics rendering and display. C/C++ application draws its UI image by native graphics drawing interface such as OpenGL ES. The drawn image needs to be transferred to the Web-based OS for display.

B2GPAPI is implemented in two layers: Application layer and Web engine layer. The application layer employs <object> tag in a web application as a connection with the C/C++ application's dynamic link library. The Web engine layer completes the specific implementation for the event information delivery, graphics rendering and display.

3.1. B2GPAPI in Application layer

A web application in a Web-based OS consists of HTML, CSS and JavaScript. We modify the <object> tag in Web engine layer and employ it as a connection with the dynamic link library of the C/C++ application in the web application. The modified <object> tag has two essential properties, "type" for identifying whether this tag is used for B2GPAPI and "plugin" for identifying the name of the dynamic link library. Fig.4 shows the specific implementation.

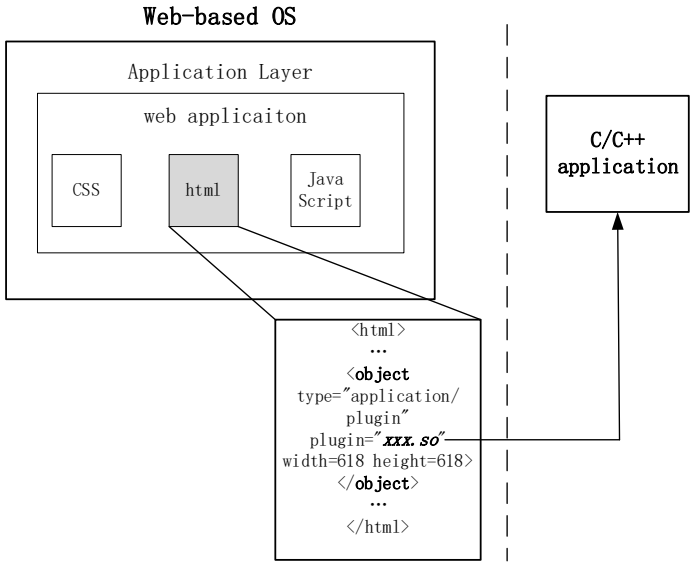

Fig. 4: B2GPAPI in Application layer

3.2. B2GPAPI in Web Engine Layer

The Web engine layer of the Web-based OS is written in C/C++, which lays the foundation for providing the C/C++ runtime in the OS. The implementation in Web engine layer is divided into two sides: B2G side in Web-based OS and Plugin side in a C/C++ application. The data structures of the B2GPAPI in Web engine are listed in Table1. In this section, we introduce B2GPAPI in Web engine by showing its lifetime in four phases.

Table 1: Data structures of B2GPAPI in Web engine

Data structures	Roles
B2GPluginInstance	The most essential data structure in B2GPAPI that inherits from DomEventListener, a class that monitors events in the browser; Includes a B2GPluginLibrary object and a B2GPluginRefreshObserver object.
B2GPluginLibrary	Load the dynamic link library and get the function pointers of initialize() and shutdown(), which are used to initialize and shutdown the framework; Store the B2GFuncs and the PluginFuncs.
B2GPluginRefreshObserver	Inherits from RefreshObserver, a class that monitors the screen refresh in the browser. It would notify B2GPAPI of displaying C/C++ application UI image as long as the screen refresh.
PluginFuncs	A list of functions that Plugin side provides, such as event handling function and graphics drawing function. The event handling function is used to receive events from the Web-based OS and transfer them to the C/C++ application. The graphic drawing function is used to draw the UI image of the C/C++ application when be notified by B2GPluginRefreshObserver. PluginFuncs is to be called by B2G side.
B2GFuncs	A list of functions that B2G side provides, such as function that register the B2GPluginRefreshObserver. B2GFuncs is to be called by Plugin side.
B2GEventStruct	Store the event information monitored by the B2GPluginInstance.

3.2.1. Initialization

When the <object> tag in a web application is identified, B2GPAPI will determine whether the <object> tag is used for B2GPAPI by the “type” property and whether the existing dynamic link library is consistent with the preset “plugin” property. If the two answers are yes, B2GPAPI initializes.

B2GPluginInstance is built first, and the B2GPluginLibrary is built by the B2GPluginInstance. B2GPluginLibrary loads the dynamic link library which is in <object> tag and stores the function pointers of the initialize() and the shutdown() of the dynamic link library. B2GPluginLibrary calls the initialize(), after which B2GFuncs is transferred from B2G side to Plugin side and PluginFuncs is transferred from Plugin side to B2G side.

B2GPluginInstance is responsible for monitoring the events from web application and the screen refresh from the Web-based OS. On the one hand, B2GPluginInstance inherits from DomEventlistener which monitors DomEvents such as load, click, blur etc. These events are to be delivered to the C/C++ application. B2GPluginInstance register event types needed to be monitored at the beginning. On the other hand, B2GPluginInstance builds B2GPluginRefreshObserver that inherits from RefreshObserver, a class that monitors the screen refresh. B2GPluginRefreshObserver would call the graphic drawing function in PluginFuncs to notify B2GPAPI of drawing the C/C++ application UI image as long as the screen refresh.

Finally, the initialization completes and the process is shown in Fig.5.

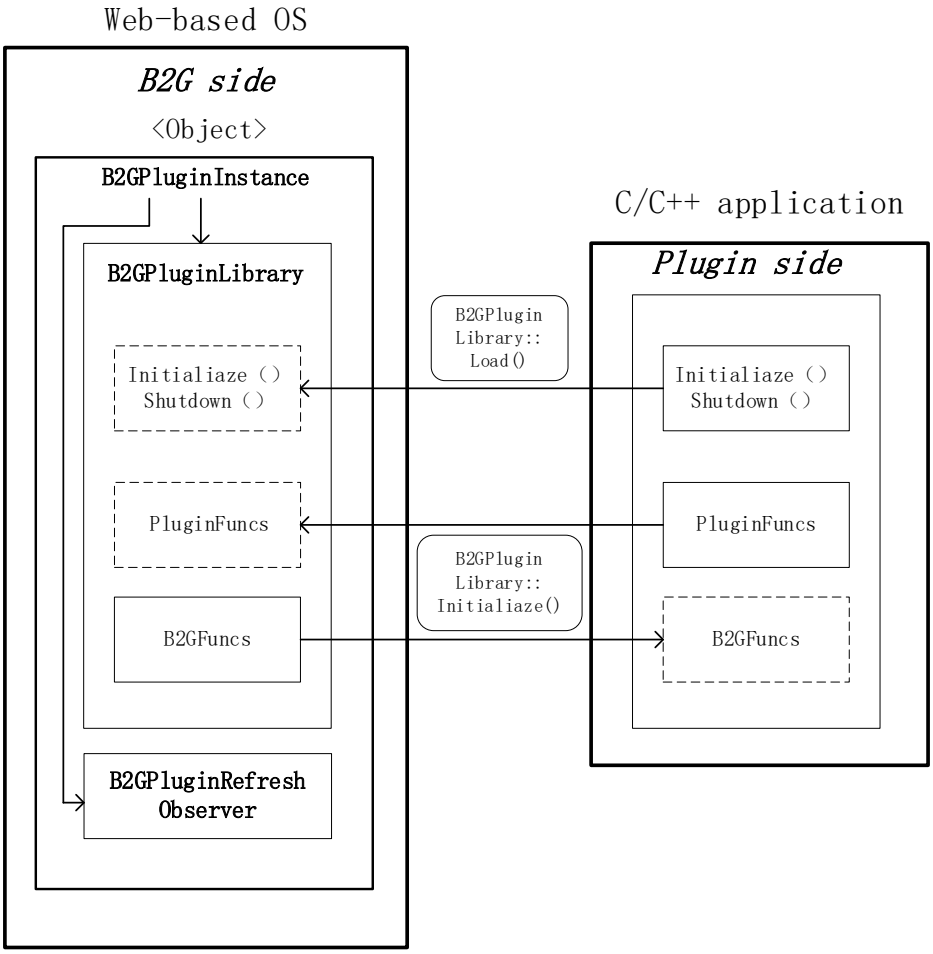

Fig. 5: Initialization of B2GPAPI

3.2.2. Event information delivery.

B2GPluginInstance monitors events from the web application, and identifies whether these events are the initially registered. If the answer is yes, B2GPluginLibrary calls the event handling function in PluginFuncs to store these events' information (including event types, coordinate information on the screen etc.) into B2GEventStruct. B2GEventStruct would be transferred to Plugin side and handled by the C/C++ application. The whole process is shown in Fig.6.

3.2.3. Graphics rendering and display

To illustrate this phase clearly, we introduce how the browser works in rendering first. The browser would parser the html to get the DOM tree. Each tag in the html corresponds to a content node in the DOM tree. The node in the DOM tree is combined with the CSSOM obtained by parsing CSS to build a render tree. A node in the render tree is called a frame, and the frame is not one-to-one with the content node in the DOM tree, in which some tags such as <head> wouldn't be rendered in the future. Render tree wouldn't be rendered at once, for the frame in the render tree needs to show the 3D effect and the relative position relationship between frames needs to be represented. So, frames in the render tree are divided into various layers, and all the layers make up a layer tree. While not all frames have corresponding layers in the layer tree, frame which does not have corresponding layer belongs to its parent frame's layer. The layers here are similar to the layers in Photoshop software. The browser would render the layer tree and the contents display on the screen finally.

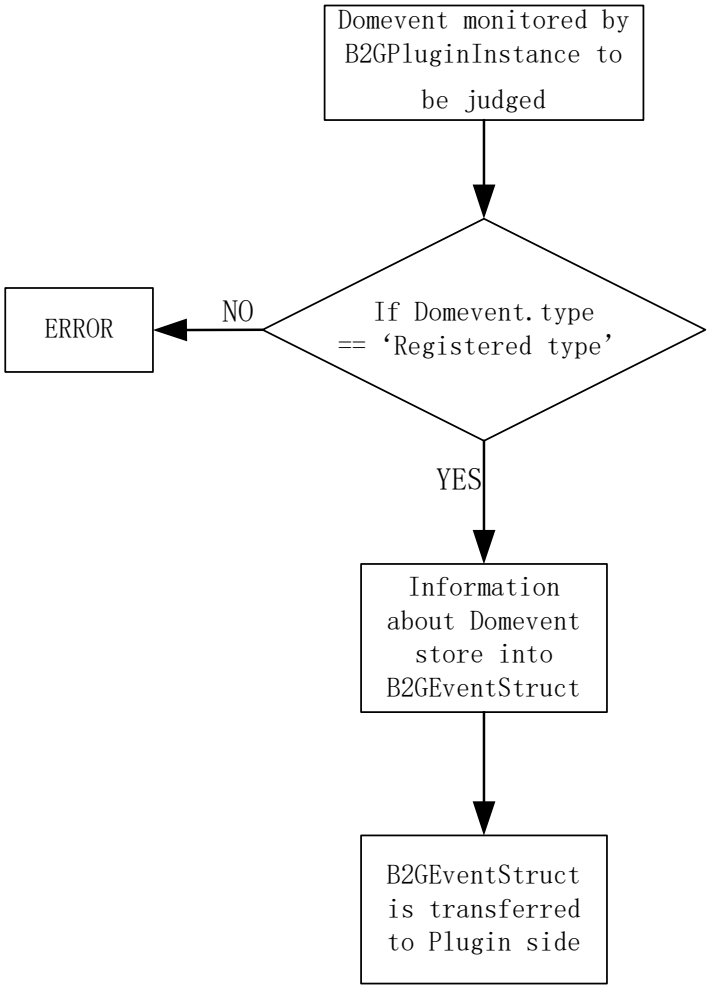

Fig. 6: Event delivery in B2GPAPI

In this phase, C/C++ application calls the corresponding function in B2GFuncs to register the B2GPluginRefreshObserver, as a result, B2GPluginRefreshObserver could monitor the screen fresh and notify the B2GPluginInstance as long as the fresh happens. Then the B2GPluginInstance gets the frame of <object> tag in the render tree, and gets its corresponding layer in the layer tree. So, all we need to do is updating that layer to display contents in <object> tag, which contains the C/C++ application.

Image is the basic class used for display in a layer, and its data maps into RAM by a data structure named GraphicBuffer. In other words, updating the layer is the same as updating the data in the GraphicBuffer.

To write data into the GraphicBuffer, we call lock() of the GraphicBuffer to lock this buffer in case of another writing. After the lock(),we get the Addr which is the address of the GraphicBuffer in RAM. Then we call graphic drawing function in PluginFuncs to deliver the Addr to Plugin side.

The C/C++ application draws its UI image using OpenGL ES and calls glReadPixels() to get the image data from VRAM into a block in RAM named gl_buffer. Then the data in gl_buffer will be copied to the block in RAM whose address is Addr and the GraphicBuffer calls unlock(). As a result, the data in GraphicBuffer updated.

Finally, the layer belonging to the <object> tag in the layer tree is updated, and the browser would render the layer tree to display the UI image of the C/C++ application. The phase is shown in Fig.7.

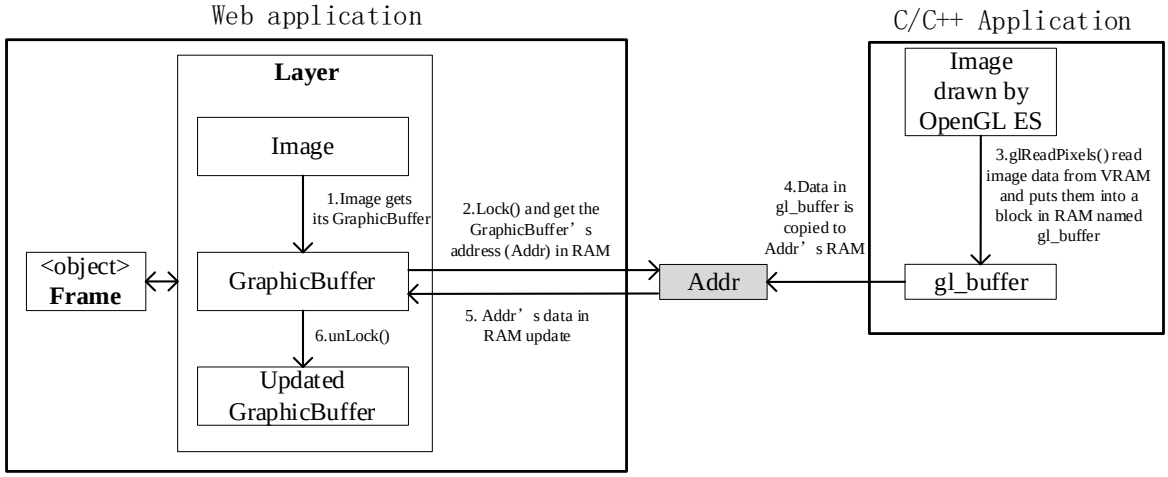

Fig. 7: Graphics rendering and display in B2GPAPI

3.2.4. Shutdown

B2GPAPI calls the shutdown() in the B2GPluginLibrary. The B2GPluginInstance unregisters the B2GPluginRefreshObserver and removes the added event type for monitoring. The instance of B2GPluginInstance in memory would be freed.

4. Experimental Analysis

Currently, there are two types of web application in the Web-based OS, original and compatible to C/C++ with B2GPAPI. The two applications use different graphic rendering interfaces. The former one uses WebGL, while the later one uses OpenGL ES to draw image in the C/C++ application and displays it in the Web-based OS with B2GPAPI. WebGL is a JavaScript API for 3D graphics rendering in the browsers. WebGL uses <canvas> tag in html to call OpenGL ES. To verify the execution efficiency of the B2GPAPI, we compared the graphics rendering interface of B2GPAPI with WebGL interface in a web application. We chose the Firefox OS as the Web-based OS.

4.1. Experimental Setup

- SoC RK3399
- ARMv8 Processor rev2 (v8l), 6 cores
- Memory 4GB
- Mozilla Firefox OS 44.0

4.2. 2D Test

Setup

We draw a 2D image in a web application of the Firefox OS with WebGL and B2GPAPI's graphics rendering interface respectively, and test the drawing speed. The 2D image is drawn 10, 100, 1000, 10000 and 100000 times respectively before display.

Firstly, we add a "click" event listener to the web application to get the drawing start time T1 when we touch the screen of the device, at the same time the 2D image starts drawing. Secondly, we add a timestamp to the code snippets which are responsible for the final display of the screen to get the end time T2 when the 2D image displays on the screen. The consumed time $\Delta T = T2 - T1$.

Tools

ADB（Android Debug Bridge）is a debug tool for Android devices. We use ADB to test because Firefox OS is developed on Android [2]. We connect the device with PC and use the "adb logcat" command to check the logs generated by the web application in real-time to get T1 and T2.

4.3. 3D Test

Setup

Similarly, we implement a 3D graphic program in a web application of the Firefox OS by WebGL and B2GPAPI's graphic rendering interface respectively. The 3D program shows a circle rotates around X, Y, Z-

axis in turn by a concrete angle between the two screen refreshes. To test the performance of WebGL and B2GPAPI, we observe the frame rate of the 3D program under different computational cost. We consider the rotate times (T) around axis as the computational cost and set the T as 100000, 150000, 200000, 250000 to observe the frame rate respectively. We add codes to the WebGL and OpenGL ES code snippets to get the frame rate. Images for test are shown in Fig. 8.

Tools

ADB tools and "adb logcat" are used again.

Fig. 8: Images for test. The left is for 2D test and the right is for 3D test.

4.4. Results Analysis

In the 2D test, when the drawing times before display is 10, B2GPAPI is 60.47% faster than WebGL, and when the drawing times before display is 100000, B2GPAPI is 18.76% faster than WebGL. When the drawing times is short, the upgrade with B2GPAPI is remarkable, with the drawing times get longer, the parallelism gets stronger and the optimization for parallelism starts to work, which conducts in the upgrade is not as remarkable as the former.

In the 3D test, with the cost of calculation increases, both the frame rate with B2GPAPI and WebGL decreases, while the B2GPAPI performs better. When the calculation cost is 100000, the framerate with B2GPAPI is 16.52% higher than WebGL, and when the calculation cost is 250000, the framerate with B2GPAPI is 25.67% higher than WebGL. Table 2 and Table 3 show the experimental results.

We infer the reason for the experimental results is that the B2GPAPI only needs to display the image that directly drawn by OpenGL ES, while the WebGL has to call the JavaScript engine to parser the corresponding codes before calling the OpenGL ES, which conducts in more time cost.

Table 2: Results for 2D test

Drawing times before display	**B2GPAPI** (s)	**WebGL** (s)
10	0.0713	0.1804
100	0.0844	0.1875
1000	0.1466	0.2696
10000	0.6278	0.8957
100000	5.3413	6.5750

Table 3: Results for 3D test

Calculational cost	**B2GPAPI** (FPS)	**WebGL** (FPS)
100000	60.8	50.8
150000	40.0	36.2
200000	33.1	27.0
250000	29.6	22.0

5. Conclusion

In order to expand the application ecosystem of the Web-based OS, we intend to transplant mature applications to the Web-based OS, while the transplant work is not easy because of the platform heterogeneity. Research found that lots of mature applications (e.g., input method and the map application)

of which the low-level implementation is written in C/C++, so we propose a framework named B2GPAPI to provide a runtime for C/C++ application in the Web-based OS to achieve the applications transplant. To verify the efficiency of the framework we make a comparison with WebGL, and experimental results show that B2GPAPI outperforms WebGL on graphics rendering interface. In the future, we plan to focus on the stability or other problems with B2GPAPI.

6. References

[1] HTML5.2: editor's draft. Retrieved April 5, 2021 from https://html.spec.whatwg.org/ Standard (whatwg.org).

[2] Firefox OS architecture. Retrieved April 5, 2021 from https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/Architecture

[3] Web API. Retrieved by April 15, 2021 from https://developer.mozilla.org/en-US/docs/Web/API

[4] Ed Angel and Dave Shreiner. 2008. An interactive introduction to OpenGL and OpenGL ES programming. In ACM SIGGRAPH ASIA 2008 courses (SIGGRAPH Asia 08). Association for Computing Machinery, New York, NY, USA, Article 2, 1–160.

[5] Daniele Nadalutti, Luca Chittaro, and Fabio Buttussi. 2006. Rendering of X3D content on mobile devices with OpenGL ES. In Proceedings of the eleventh international conference on 3D web technology (Web3D 06). Association for Computing Machinery, New York, NY, USA, 19–26.

[6] Ed Angel and Dave Shreiner. 2014. An introduction to WebGL programming. In ACM SIGGRAPH 2014 Courses (SIGGRAPH 14). Association for Computing Machinery, New York, NY, USA, Article 7, 1–105.

[7] Hyeong-Seok Oh, Beom-Jun Kim, Hyung-Kyu Choi, and Soo-Mook Moon. 2012. Evaluation of Android Dalvik virtual machine. In Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 12). Association for Computing Machinery, New York, NY, USA, 115–124.

[8] Shivi Garg and Niyati Baliyan. 2021. Comparative analysis of Android and iOS from security viewpoint. Computer Science Review, Volume 40.

[9] Firefox OS. Retrieved Mar 24, 2021 from https://developer.mozilla.org/en-US/docs/Archive/B2G_OS

[10] NPAPI. Retrieved Mar 27, 2021 from https://developer.mozilla.org/en-US/docs/Plugins

[11] Andre Charland and Brian LeRoux. 2011. Mobile Application Development: Web vs. Native: Web apps are cheaper to develop and deploy than native apps, but can they match the native user experience? Queue 9, 4 (April 2011), 20–28.

[12] Abeer AlJarrah and Mohamed Shehab. 2017. The Demon is in the Configuration: Revisiting Hybrid Mobile Apps Configuration Model. In Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES 17). Association for Computing Machinery, New York, NY, USA, Article 57, 1–10.

[13] David Hayes. 2018. Mobile Web App Development for All!. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE 18). Association for Computing Machinery, New York, NY, USA, 1060.

[14] Ville Ahti, Sami Hyrynsalmi, and Olli Nevalainen. 2016. An Evaluation Framework for Cross-Platform Mobile App Development Tools: A case analysis of Adobe PhoneGap framework. In Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (CompSysTech 16). Association for Computing Machinery, New York, NY, USA, 41–48.

[15] AppCan: An IDE for Mobile Application. Retrieved Apirl 24, 2021 http://newdocx. appcan.cn/

[16] Mohamed Shehab and Abeer AlJarrah. 2014. Reducing Attack Surface on Cordova-based Hybrid Mobile Apps. In Proceedings of the 2nd International Workshop on Mobile Development Lifecycle (MobileDeLi 14). Association for Computing Machinery, New York, NY, USA, 1–8.