International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 4 (April 2014), PP.30-35

A study on the OpenGL ES and the OpenSL ES in the Android NDK paradigm

Sabyasachi Patra[1], Karishma Velisetty[2], Prathamesh Patel[3], Mr. Abhay Kolhe[4]
[1]*Mukesh Patel school of Technology and Management, Computer science Mumbai, India*
[2]*Mukesh Patel school of Technology and Management, Computer science, Mumbai, India*
[3] *Mukesh Patel school of Technology and Management, Computer science, Mumbai, India*
[4] *Faculty, Mukesh Patel school of Technology and Management, Computer science, Mumbai, India*

Abstract: -When it comes to beautiful visual rendering on the Android handsets that we use today, much of that credit has to be given to the various graphic libraries which come along with programming paradigms. Just like anything in programming and technology, there are good and bad ways to implement and get certain things done both at the front end and the backend. What the Android Native Development Kit (NDK) does is that it works alongside the Software Development Kit (SDK) and injects the native powers of any C/C++ application into your Android application which can be packaged as any normal application and run on an emulator/device of choice. The SDK, the NDK, the ADT and Eclipse are primarily what one requires to directly hit on towards Android Native Development. The Android NDK basically is a companion toolset for the Android SDK, designed to augment the SDK to allow developers to implement and embed performance-critical portions of their applications using 'machine code' generating programming languages like C, C++, and Assembly. So now how does one enjoy the seamless graphics and sound in multimedia applications on an Android phone? It is the Native Graphics/ Sound API's which come as a result of relying on performance critical native code which makes this possible. There are various levels of detail to which this can be carried out. In this paper we delve into the introductory portions of what the Open Graphics Library and the Open Sound Library for Android actually are and the important steps which broadly need to be followed, in order to get such applications up and running. We also talk about the basic Hardware and Software implementations of each of these along with the Configurations, simplified diagrams as well as the role of sensors.

Keywords: NDK, Companion toolset, Native Programming, Open GL, Open SL, C++ Android, Graphics

I. INTRODUCTION

Multimedia is defined as a system, which equally encompasses text, images, sound, animation and video. Multimedia Applications are one of the most popular in the Android market. One of the reasons for this is the Music. Most mobile phones are sold because of only the sound quality to many music lovers. Also, almost every complex application in the market requires embedding of sound into it.

Various sound related - APIs are Media Player, Sound Pool, Audio Track, Jet Player and Open SL for embedded systems.

- MediaPlayer: It is a high-level API and hence easy to use. It handles music and video. It is the way to go when simple file playback is sufficient.
- 'SoundPool' and 'AudioTrack': These are low – level API's and closer to low latency when playing sound. This API is the most flexible yet complex to use and allows sound buffer modifications on the fly.
- JetPlayer is more dedicated to the playback of MIDI files.
- OpenSL ES basically aims at offering a cross-platform API to manage audio on embedded systems.On Android, OpenSL ES is in fact implemented on top of the AudioTrack API.
- OpenSL ES was first released on the Android 2.3 Gingerbread OS version. While there is a profusion of APIsin Java, OpenSL ES is the only one provided on the native side and is exclusivelyavailable on it.

However, the point to be noted on a broader scale when it comes to OpenSL ES is that it is still immature, not to say that it is a nemesis of course. The OpenSL specification is still incompletely supported and several limitations shall be expected of it. In addition, OpenSL specification is implemented in its version 1.0.1

on Android although version 1.1 is already out. Thus, OpenSL ES implementation is not frozen at the highest echelons yet and should continue evolving.

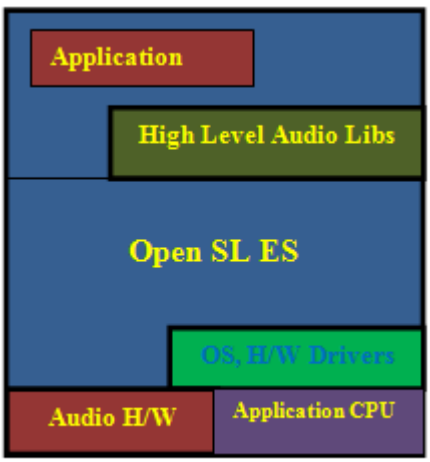

Fig 1: OpenSL ES H/W Implementation

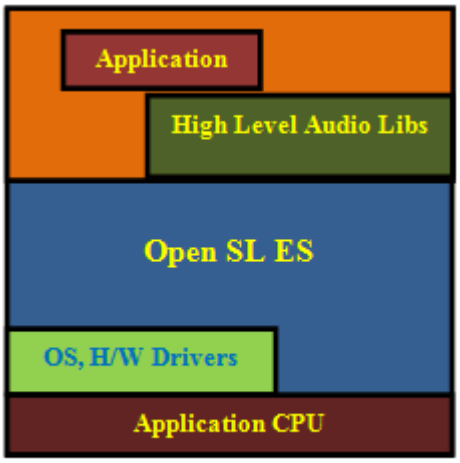

Fig 2: OpenSL ES H/W Implementation

In our research undertaking we identified, reviewed and tested some of the very important steps which are required for an introductory and in-depth implementation of the OpenSL ES.

These actions in the Android programming domain can be classified as:

- **Initialize OpenSL ES on Android**
- **Play background music**
- **Play sounds with a sound buffer queue**
- **Record sounds and play them**

II. CREATING OPENSL ES ENGINE AND OUTPUT

The 'SoundService' class does the following:

- **Initializing OpenSL ES by using method start()**
- **Similarly, By using method stop() ,sound can be stopped.**

Pseudo-object structures in OpenSL ES:

- Objects: These are represented by a 'SLObjectItf'. It provides common methods to get allocated resources and get object interfaces.
- Interfaces: These give access to object features. There can be several interfaces for an object. It depends on host devices, accordingly interfaces are available.

III. OPEN SL ES CONFIGURATION

```
#include <android_native_app_glue.h>
#include <SLES/OpenSLES.h>
#include <SLES/OpenSLES_Android.h>
#include <SLES/OpenSLES_AndroidConfiguration.h>
```

The above headers need to be added in the startof any OpenSL ES Application.

STEPS FOR SETTING UP AN 'OPENSL ES' OBJECT:

[1]. Instantiating it through a build method (belonging usually to the OpenGL engine).
[2]. Realizing it to allocate necessary resources.
[3]. Retrieving object interfaces. A basic object only has a very limited set of operations (Realize(), Resume(), Destroy(), and a few more). Interfaces give access to real object features and describes what operations can be performed on an object, for example, a Play interface to play or pause a sound. Any interfaces can be requested but only the one supported by the object is going to be successfully retrieved.

IV. HANDLING INPUT DEVICES AND SENSORS

Android is all about interaction. Admittedly, that means feedback, through graphics, audio, vibrations, and so on. The success of today's smart-phones takes its root in their multiple and modern input possibilities: touch screens, keyboard, mouse, GPS, sound recorder, and so on.

Examples of available devices are:

- Keyboard, either physical (with a slide-out keyboard) or virtual (which appears on screen)
- Directional pad (up, down, left, right, and action buttons), often abbreviated D-Pad
- Trackball (optical ones included)
- Touch screen, which has made the success of modern smart-phones
- Mouse or Track Pad (since NDK R5, but available on Honeycomb devices only)

Hardware sensors:

- Accelerometer
- Gyroscope
- Magnetometer
- Light
- Proximity sensor

Software sensors also have been introduced with the 'Gingerbread' version release.

Other sensors are:

- Gravity sensor: This helps to measure the gravity direction and magnitude
- Linear acceleration sensor: It helps to measures device "movement" excluding gravity
- Rotation vector: This indicates device orientation in space

Gravity sensor and linear acceleration sensor are derived from the accelerometer. On the other hand, rotation vector is derived from the magnetometer and the accelerometer. Because these sensors are generally computed over time, they usually incur a slight delay to get up-to-date values.

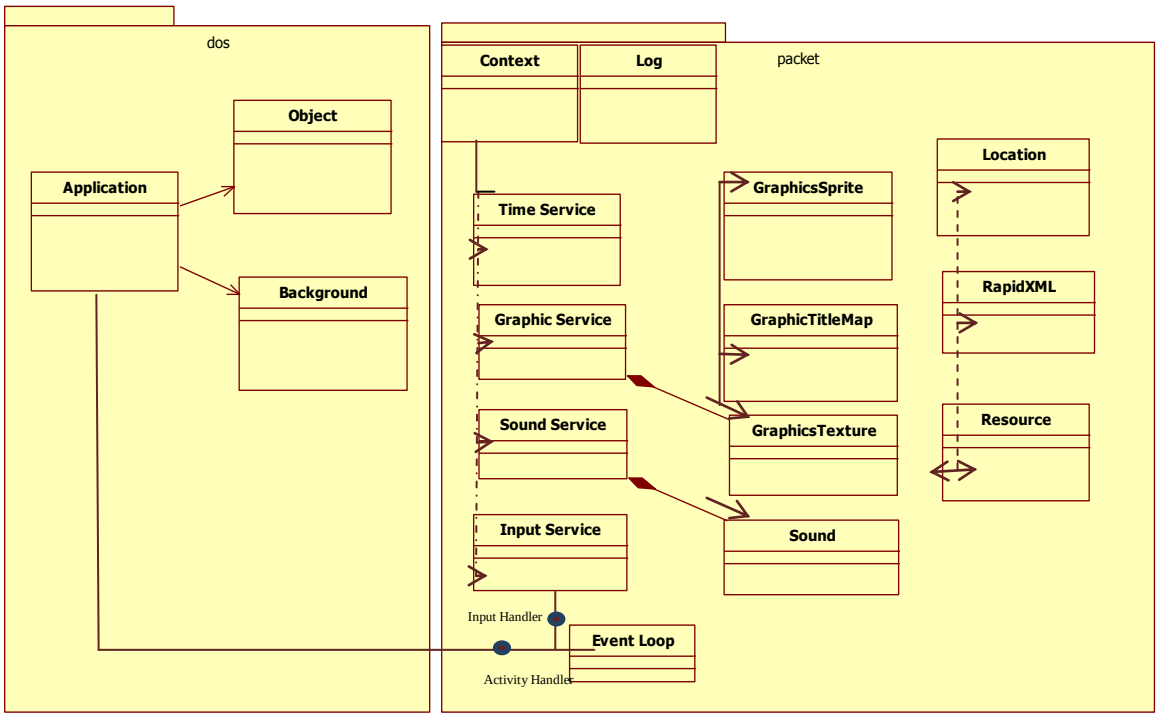

Fig 3: Android generic Application structure handling input devices and sensors

V. INTERACTING WITH ANDROID

The biggest innovation in technology in mobile/smart phones is electronic visual display also known as touch screen. It helps the user to interact with device with the help of simple gestures or by touching the screen. It uses coated gloves to understand the action through electronic visual display.

VI. PROBING DEVICE SENSORS

In anapplication, handling the input devices is very important, but probing sensors is more important for running smarter applications. The most used sensor in Android especially for games is the accelerometer which is basically used to measure the linear acceleration applied to a device.
When moving a device up, down, left, or right, the accelerometer gets excited and indicates an acceleration vector in 3D space. Vector is expressed relative to screen default orientation. The Coordinates system is relative to device natural orientation:

- X axis points left
- Y points up
- Z points from back to front

Axes become inverted if device is rotated (for example, Y points left if the device is rotated 90 degrees clockwise).

Accelerometers can also undergo a constant acceleration: gravity, around 9.8m/s2 on the Earth. For example, when lying flat on a table, acceleration vector indicates -9.8 on the Z-axis. When straight, it indicates the same value on Y axis. So assuming device position is fixed, device orientation on two axes in space can be deduced from the gravity acceleration vector. Magnetometer is still required to get full device orientation in 3D space.

VII. OPEN GL ES

The Android NDK provides OpenGL ES with the versions 1.x and 2.0 graphics API to the native code. Version 1.0 is supported from Android 1.6 and further versions. Version 1.1 is supported only on devices that have the corresponding GPU. Version 2.0 is supported on Android 2.0 and further versions. The <uses-feature> tag is used by the applications in android manifest file to indicate the preferred. 'android.opengl.GLSurfaceView' instance is used in Java code to use the OpenGL ES API. Native application can these functions to render graphics to the GLSurfaceView.

OpenGL basically is a standard API created by Silicon Graphics and now managed by the Khronos Group (http://www.khronos.org/). OpenGL ES derivative is available on many platforms such as iOS or Blackberry OS and is the best hope for writing portable and efficient graphics code. OpenGL can do both 2D and 3D graphics with programmable shaders (if hardware supports it). There are two main releases of OpenGL ES currently supported by Android:

- **OpenGL ES 1.1**: This is the most supported API on Android devices. It offers an old school graphic API with a fixed pipeline (that is, a fixedset of configurable operations to transform and render geometry).This is a good choice towrite 2D games or 3D games targeting older devices.

- **OpenGL ES 2**: This is not supported on old phones (like the antic HTC G1)but more recent ones (at least not so old like the Nexus One… time goes fast in the mobile world) support it. OpenGL ES 2 replaces the fixed pipeline with a modern programmable pipeline with vertex and pixelshaders. This is the best choice for advanced 3D games. Note thatOpenGL ES 1.X is frequently emulated by an OpenGL 2 implementation behind the scene.

The most important point to be noted when developing applications using the NDK and Graphics is that though you can work with the emulator, but it's much better to have an actual device, because emulators do not accurately reflect real-world performance and results, and it can also be very slow, even on high-end hardware.

STEPS TO USE THE OPENGL ES 2.0 IN NATIVE APPLICATION:

1. Include the OpenGL ES 2.0 header files.

#include <GLES2/gl2.h>
#include <GLES2/gl2ext.h>

2. Update the Android.mk build file to dynamically link with GLESv2 library.

LOCAL_LDLIBS += −lGLESv2

Open Sound Library (OpenSL**):** It provides a native sound API that helps to play and record audio. This can be done without invoking any method at Java layer.

Obtaining a Device Supporting OpenGL ES 2.0.

The GLSurfaceView class makes Android activity life cycle easy to handle.. In Android activity lifecycle we can create, destroy, pause and resume activities.

Adding OpenGL Activity and initializing:

public class OpenGLActivity extends Activity {
private GLSurfaceView glSurfaceView;
private boolean rendererSet = false;

Checking If the System Supports OpenGL ES 2.0

In the Activity class

activityManager.getDeviceConfigurationInfo();
***final boolean** supportsEs2 = configurationInfo.reqGlEsVersion >= 0x20000;*

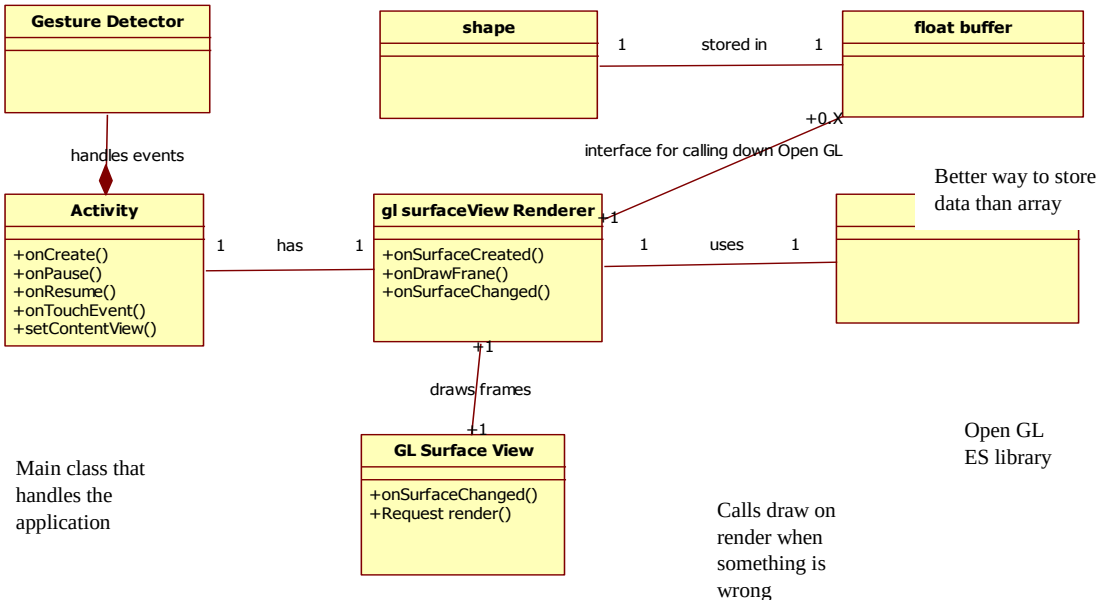

Fig 4: Android Open GL ES Class Diagram

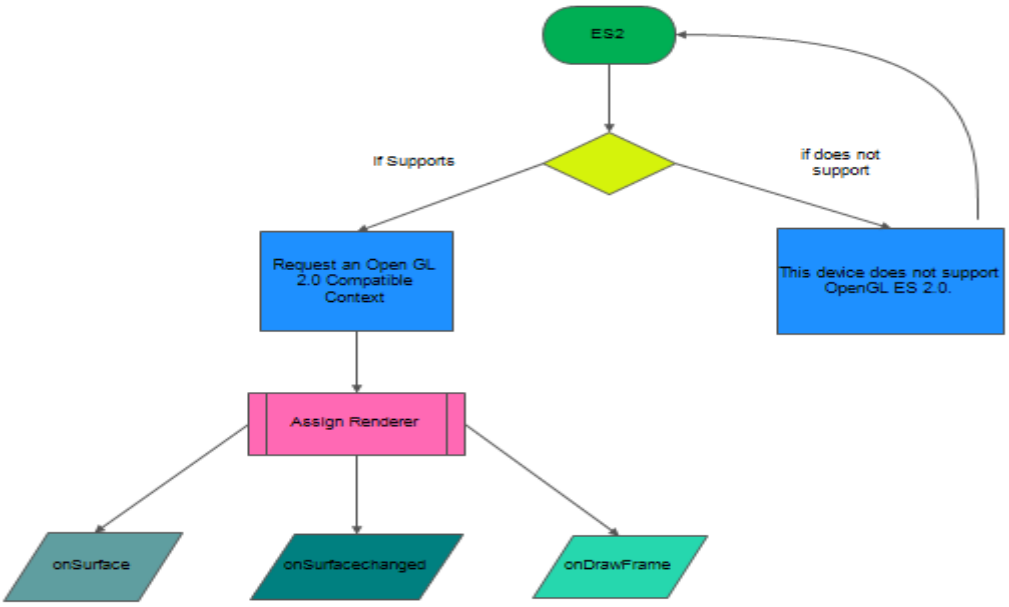

Fig 5: Flowchart| checking for compatibility and Configuring OPENGL ES Surface

VIII. CONCLUSION

Normal Android application development in both the industry as well as for personal purposes is done on the Android SDK using the various API's provided by Google and Android. The NDK is used for special purposes whenever required, as in for Physics simulations or certain kind of games. There are certain performance – critical portions where in the role of Native development with languages such as C/C++ comes into picture. The secret to 3D rendering, games, collisions and other simulations employ these libraries as enablers. In this paper we have carried out research on primarily the Android Graphics and Sound library and supported that with critical configurations which would enable strong and simple implementations.

ACKNOWLEDGEMENT

This research paper is made possible through the help and support of many people both inside and outside the domain of Engineering and Science.

Especially, please allow me to dedicate my acknowledgment of gratitude towards our college Librarian Mr. Pradip Das and his team. I would also like to thank Mr. Anand Gawadekar of the NMIMS IEEE Committee due to which we could get all requested references seamlessly without any trouble and on time.

A sincere thanks to our college and Computer Science department H.O.D. Professor Dhirendra Mishra for allowing us to enter the invaluable field of research in our so important final year of B.Tech.Our mentor, Mr. Abhay Kolhe also guided us on a weekly basis at periodic meets with him. Finally, we would like to thank our parents who always encouraged us to do as much research we could do in our capacity in the final year and extend an outside support whenever and wherever required.

REFERENES

[1] The Android Google group (http://groups.google.com/group/androiddevelopers)
[2] Android NDK group (http://groups.google.com/group/android-ndk)
[3] The Android Developer BlogSpot (http://android-developers.blogspot.com/)
[4] Google Code (http://code.google.com/hosting/) for lots of NDK exampleapplications.
[5] Stack Overflow (http://stackoverflow.com/)
[6] Sangchul Lee and Jae Wook Jeon "Evaluating Performance of Android Platform Using Native C for Embedded Systems" Control Automation and Systems (ICCAS), 2010 International Conference pages 1160 - 1163.
[7] Walter Binder, Jarle Hulaas and Philippe Moret "A Quantitative Evaluation of the Contribution of Native Code to Java Workloads" Workload Characterization, 2006 IEEE International Symposium pages 201-209. [8]S. Liang. The Java Native Interface: Programmer‟s Guide and Specification. Addison-Wesley, June 1999.
[8] Sun Microsystems.Integrating native methods into Java programs. http://java.sun.com/docs/books/ tutorialNB/download/tut- native1dot0.zip, May 1998.
[9] Java Native Interface. http://java.sun.com/j2se/1.3/docs/guide/jni/index.html.
[10] D. Bornstein, "Dalvik VM Internals," 2008.
[11] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform, Third Edition*. The Pragmatic Bookshelf, Raleigh, NC and Dallas,TX, 2010.
[12] M. B. Dillecourt, H. Samet and M. Tamminen, "A general approach to connected component labeling for arbitrary image representations", Journal of the ACM, vol. 39, no. 2, (1992), pp. 253-280.