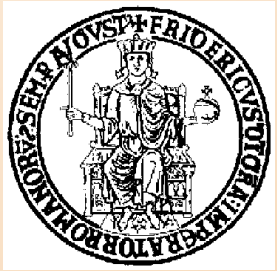
Università degli studi di Napoli Federico II



Dipartimento di Ingegneria Elettrica e
delle Tecnologie dell'informazione

Classe di Laurea in Ingegneria dell'Informazione, Classe n. L.8

Corso di Laurea in Ingegneria Elettronica

Elaborato di Laurea

ANDROID NDK:

SVILUPPO DI APPLICAZIONI NATIVE CON ANDROID NDK

Relatore:
Ch.mo Prof. PORFIRIO TRAMONTANA

Candidato:
GIOVANNI CAMPOLATTANO
Matr.N43000980

Il seguente lavoro di tesi è finalizzato all'analisi dell'aspetto e dei contenuti dell'Android *Native Development Kit,* cioè un set di strumenti che consente di utilizzare codici C/C ++ con Android. Per effettuare tale analisi è necessario descrivere dettagliatamente il sistema operativo Android e le prestazioni del kit di sviluppo nativo. Per tale motivo, i primi due capitoli sono finalizzati alla presentazione dell'architettura Android e delle caratteristiche principali dell'Android NDK. Nei capitoli successivi, per mostrare il modo in cui è strutturata un'applicazione native con Android NDK, verrà analizzato lo sviluppo di un'applicazione nativa: Sensor-Graph. Tale applicazione gestisce i dati del sensore d'accelerazione di un dispositivo mobile e li rappresenta graficamente. Per analizzare tale applicazione, verrà fatta preventivamente una panoramica sui principali sensori disponibili per i dispositivi mobili.

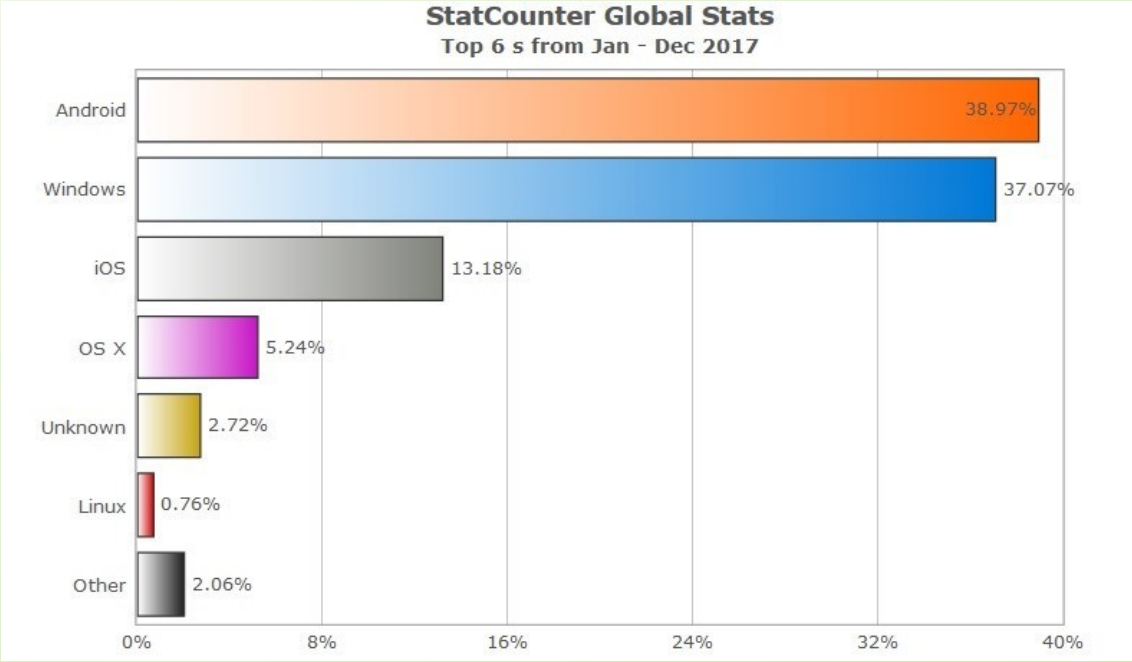


Figura 1- sistemi operativi più diffusi

Prima di analizzare l'architettura Android è utile notare che esso è il sistema operativo per dispositivi mobili più diffuso ed utilizzato. In particolare, è un insieme di componenti software, comprendente un sistema operativo, un *middleware* e un insieme di applicazioni basilari, utilizzato in *smartphones*, *tablets*, *smart watches*, ma potrebbe essere esteso a supportare qualsiasi dispositivo, ipoteticamente anche un PC. Android è compatibile con un numero elevato di dispositivi e sistemi, ed è un sistema *open source.* Con il termine open si vogliono intendere diverse cose:

- È *open* in quanto il suo codice è liberamente consultabile, migliorabile, ma soprattutto non è necessario pagare nessuna *royalty* per utilizzarlo sui dispositivi;
- È *open* in quanto le librerie e le *API* utilizzate per la sua realizzazione sono le stesse che possono essere sfruttate per la creazione di nostre applicazioni;
- È *open* in quanto si possono utilizzare diversi linguaggi di programmazione, anche se il più usato e supportato è Java.

ARCHITETTURA ANDROID

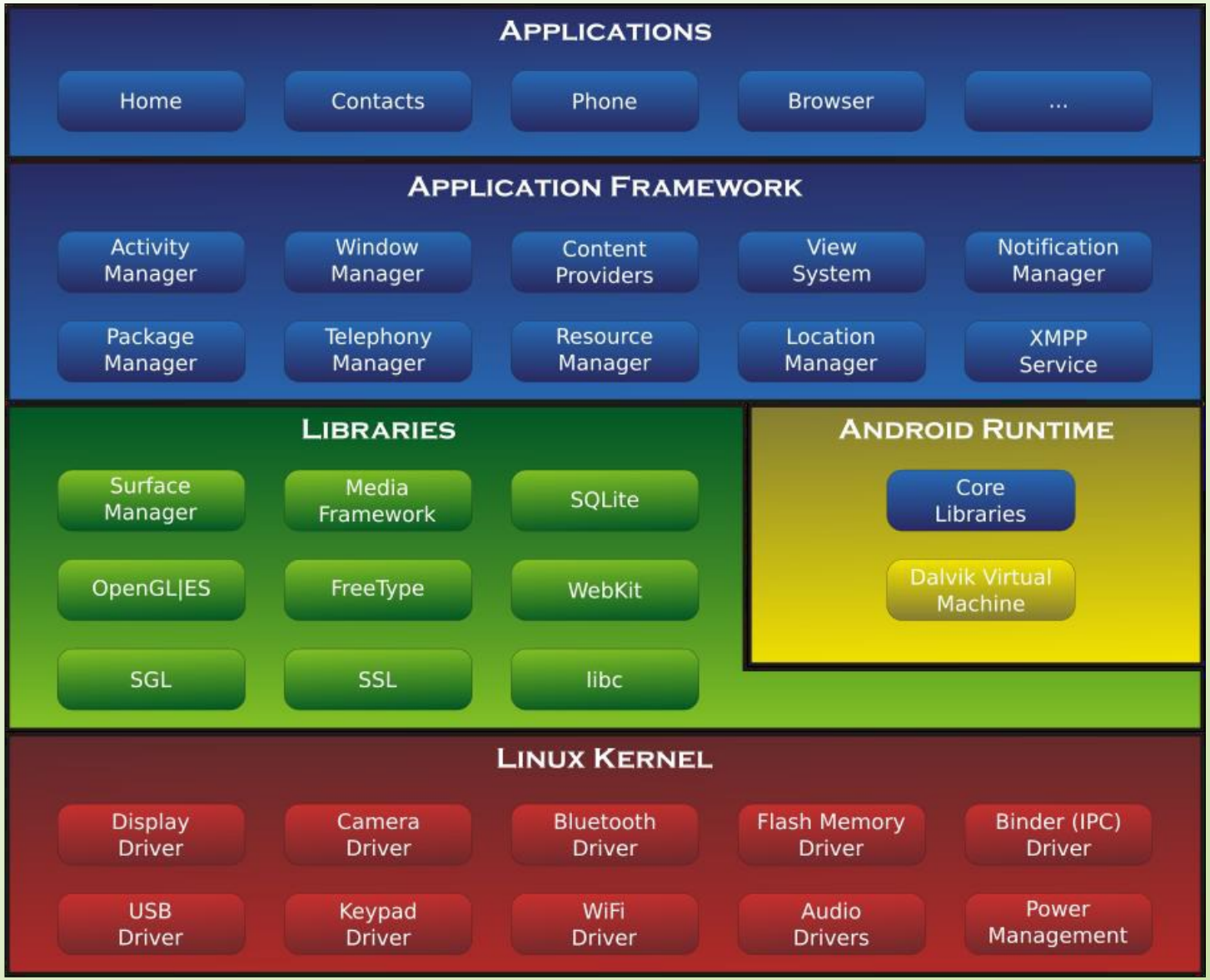


Figura 2- architettura Android

L'architettura Android è di tipo gerarchico ed è rappresentata dai seguenti strati:

- ***Linux Kernel****:* è uno strato di astrazione tra l'hardware sottostante (che comprende il GPS, la fotocamera, il touchscreen) ed il resto dello stack software.
- ***Android Runtime****:* strato dedicato all'esecuzione delle applicazioni.
- ***Libraries****:* le librerie sono sviluppare perlopiù in C/C++. Tra esse vi sono: *Surface Manager, OpenGL ES, SQLite,* etc.
- ***Application Framework****:* mette a disposizione degli sviluppatori componenti riutilizzabili per lo sviluppo delle proprie applicazioni. In questo strato vi sono i seguenti moduli: *Activity Manager, Telephony Manager, Notification Manager,* etc.
- ***Applications****:* è costituito dalle applicazioni sia native (gestione dei contatti, calendario, etc) che provenienti da terze parti.

ANDROID NATIVE DEVELOPMENT KIT

Si può ora presentare il *Native Development Kit (NDK)* cioè un set di strumenti che consente di utilizzare codici C/C ++ con Android e fornisce librerie di piattaforme che è possibile utilizzare per gestire attività native e accedere ai componenti fisici del dispositivo, come sensori e input tattili. Il codice Java può chiamare le funzioni contenute nelle librerie native attraverso il framework *JNI (Java Native Interface)*. Il kit di sviluppo nativo Android è uno dei più efficienti sistemi operativi per multimedia e giochi.

Figura 3- robottino Android

NDK è utile per applicazioni nelle quali è necessario eseguire una o più delle seguenti operazioni:

- Spremere le prestazioni di un dispositivo per ottenere una bassa latenza o eseguire applicazioni ad alta intensità di calcolo, come giochi o simulazioni fisiche.
- Riutilizzare le librerie C/C++ migliorando la velocità dell' applicazione, in quanto le librerie di base di C/C++ sono più veloci perché, un eseguibile in C++ è eseguibile direttamente sul processore del dispositivo mobile.
- Per ridurre notevolmente il processo di sviluppo del progetto per applicazioni con molte righe di codice C/C++.

STRUTTURA DI UN'APPLICAZIONE NATIVA

La realizzazione di un' applicazione nativa con Android NDK si basa sui seguenti elementi:

- *Activity:* è responsabile dell'interazione diretta con l'utente mediante i dispositivi di input. Il ciclo di vita di un'activity è una struttura a pila e l'attività in cima è sempre quella attiva (fig. 4).
- *Intent:* è un meccanismo di messaggi che può attivare tre dei componenti di base di un'applicazione: *activity, service* e *broadcast receiver.*
- *Boradcast Receiver:* permette di eseguire un codice in seguito ad un evento esterno improvviso, ad esempio una chiamata in arrivo.
- *Service:* prepara i dati che le *activity* devono mostrare all'utente, permettendo una reattività maggiore nel momento della visualizzazione.
- *Content Provider:* permette la condivisione di dati tra applicazioni.

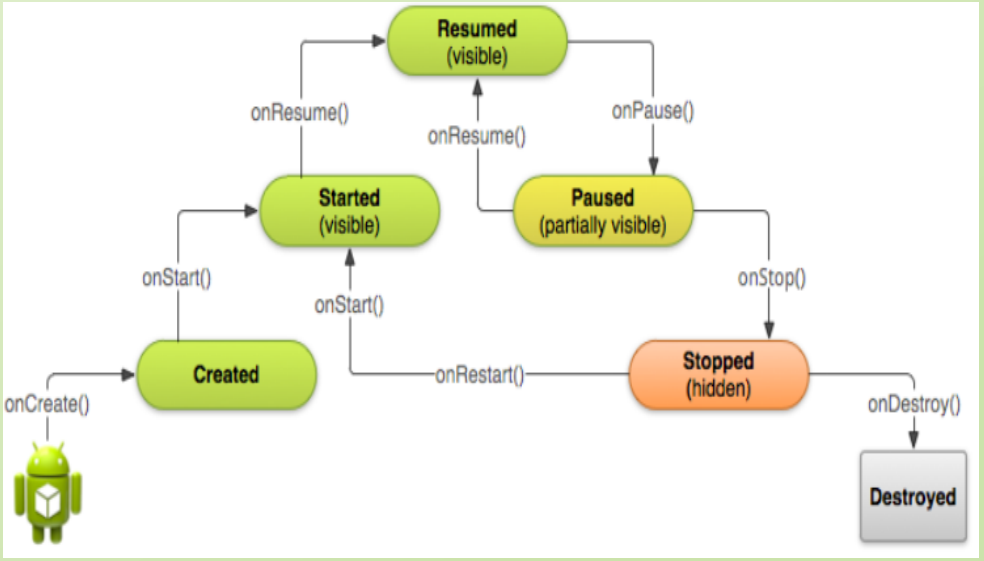


Figura 4- ciclo di vita di un'Activity

SENSORI PRESENTI NEI DISPOSITIVI MOBILI

La maggior parte dei dispositivi Android ha sensori integrati che misurano il movimento, l'orientamento e varie condizioni ambientali. Questi sensori sono in grado di fornire dati con precisione elevata e sono utili se si desidera monitorare il movimento o il posizionamento di un dispositivo tridimensionale o se si desidera monitorare i cambiamenti nell'ambiente circostante al dispositivo. Alcuni di questi sensori sono basati su hardware e alcuni sono basati su software. I sensori basati su hardware sono componenti elettronici fisicamente inseriti in un dispositivo mobile; essi ricavano i loro dati misurando direttamente specifiche proprietà ambientali, quali accelerazione o intensità del campo geomagnetico. I sensori basati su software non sono dispositivi fisici, sebbene imitino i sensori basati su hardware. Essi derivano i dati da uno o più sensori basati sull'hardware e sono talvolta denominati sensori virtuali o sensori sintetici. Il sensore di accelerazione lineare e il sensore di gravità sono esempi di sensori basati su software. Per gestire il funzionamento dei sensori di un dispositivo mobile esistono alcuni metodi come:

- getSensorList(): serve ad identificare i sensori presenti in un dispositivo.
- getMaximumRange(): serve per ottenere l'intervallo di misurazione massimo.
- getMinDelay(): serve per conoscere l'intervallo di tempo minimo necessario ad un sensore per rilevare i dati.

In generale, i sensori di un dispositivo mobile utilizzano un sistema di coordinate a 3 assi per esprimere i valori dei dati. Per la maggior parte dei sensori, in particolare per i sensori d'accelerazione, gravità e campo magnetico, il sistema di coordinate è definito in relazione allo schermo del dispositivo quando esso è nell'orientamento predefinito, nel quale:

- Asse x: è orizzontale e diretto verso destra.
- Asse y: è verticale e diretto verso l'alto.
- Asse z: punta verso l'esterno della faccia dello schermo.

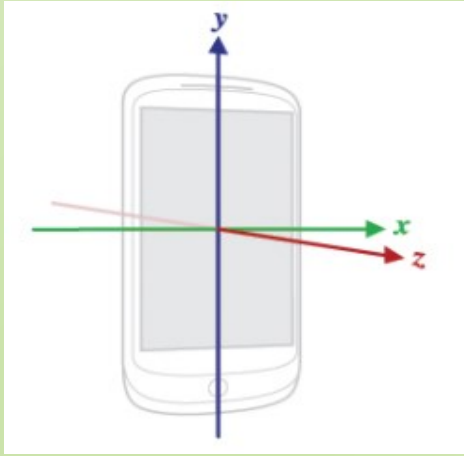


Figura 5- sistema di coordinate dei sensori di un dispositivo mobile

ACCELEROMETRO

Tra i vari sensori disponibili per i dispositivi mobili, si analizza l'accelerometro, in quanto è oggetto di studio dell'applicazione Sensor-Graph che sarà analizzata al termine di questa presentazione. Esso è un sensore di movimento che sfrutta le variazioni d'accelerazione per determinare se ci si sta spostando o meno, in quale direzione ci si sta spostando ed altre informazioni riguardanti la posizione ed il movimento del dispositivo mobile o di chi lo indossa e lo sta utilizzando. Può comporsi di elementi che misurano gli spostamenti su due o tre assi. Nel primo caso, i movimenti saranno registrati esclusivamente su un piano orizzontale (o verticale) e dunque bidimensionale; nel secondo caso si avrà la misurazione della variazione d'accelerazione tridimensionale e l'accelerometro registra gli spostamenti dell'oggetto in qualunque direzione.

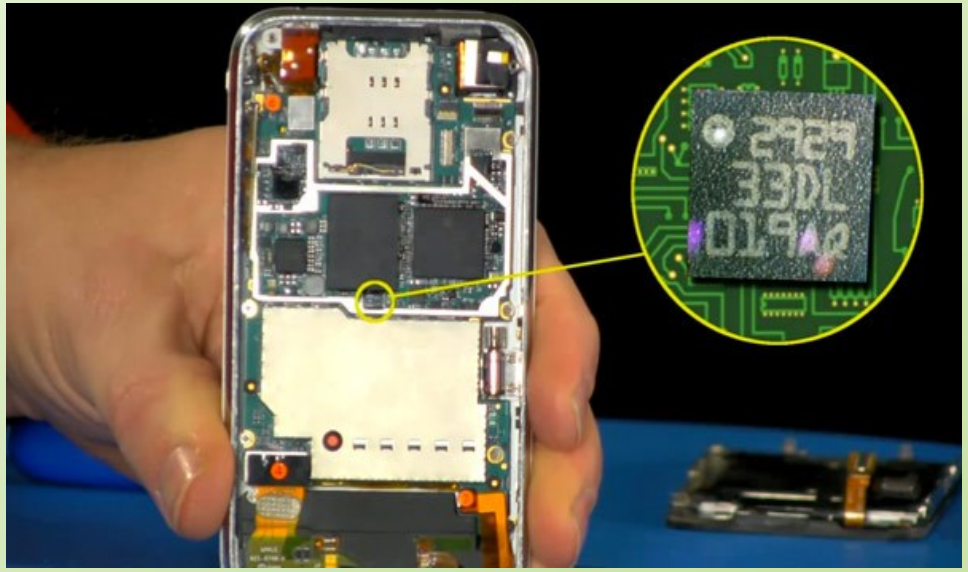


Figura 6- locazione dell'accelerometro in un dispositivo mobile

SENSOR-GRAPH

Dunque, dopo aver presentato la struttura generale di Android, le prestazioni e gli utilizzi dell' Android NDK, le caratteristiche dei sensori di un dispositivo mobile, in particolare l'accelerometro, si utilizzano le informazioni ricavate dall'analisi di questi componenti per analizzare lo sviluppo dell'applicazione Sensor-Graph. Essa legge in tempo reale i valori dell'accelerometro, captando i movimenti di uno smartphone e li rappresenta graficamente tramite OpenGL. Il progetto è contenuto in una cartella denominata accelerometer, che rappresenta il modulo di base e che include quattro file principali: file manifest, cartella con codice java, cartella con codice C++ e cartella res (contenente risorse realizzare in xml). Dopo tale modulo vi è la sezione Gradle Scripts in cui ci sono i file di build che saranno utilizzati da gradle per trasformare il progetto in un'applicazione funzionante.

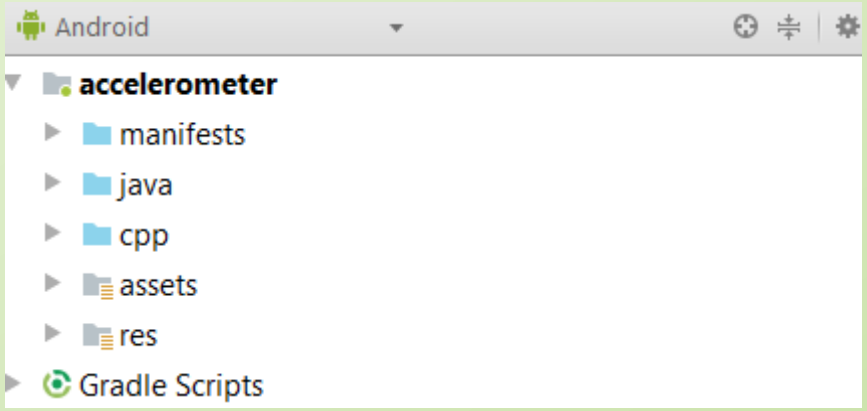


Figura 7- progetto Sensor-Graph

ANALISI CODICE C++

Il codice C++ si collega tramite Java native interface (JNI) al codice java per realizzare l'applicazione. Il codice inizia con l'inclusione di alcune librerie native, come dlfcn.h che definisce alcune macro-funzioni da utilizzare, tra le quali vi sono:

- RTLD_LAZY: fa sì che le ricollocazioni vengano eseguite in un tempo dipendente dall'implementazione.
- RTLD_NOW: fa sì che le ricollocazioni vengano eseguite quando l'oggetto viene caricato.

Dopo le librerie si dichiarano le costanti contenenti informazioni come la durata dell'attività del sensore, la frequenza di aggiornamento del sensore (in Hz) e il periodo di campionamento (in µs).

```
#include <dlfcn.h>
#include <jni.h>
#include <GLES2/gl2.h>

#include <android/log.h>
#include <android/asset_manager_jni.h>
#include <android/sensor.h>

#include <cstdint>
#include <cassert>
#include <string>

#define  LOG_TAG    "accelerometergraph"
#define  LOGI(...)  __android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__)

const int LOOPER_ID_USER = 3;
const int SENSOR_HISTORY_LENGTH = 100;
const int SENSOR_REFRESH_RATE_HZ = 100;
constexpr int32_t SENSOR_REFRESH_PERIOD_US = int32_t(1000000 / SENSOR_REFRESH_RATE_HZ);
const float SENSOR_FILTER_ALPHA = 0.1f;
```

Tra la riga 55 e la riga 59 si nota la presenza di una struct, che serve per memorizzare i valori delle tre dimensioni in cui stiamo analizzando l'accelerazione provocata dal movimento dello smartphone. Gli attributi x, y, z sono di tipo Glfloat, che è simile al tipo float ma che si utilizza per le multi-piattaforme, per evitare conflitti con le altre piattaforme. Tra la riga 64 e la riga 107, viene impostato il modo in cui comparirà a video il grafico. Infatti, si nota la presenza di alcuni shader utilizzati dalla libreria grafica OpenGL che sfruttano le capacità di shading delle GPU delle schede video. In particolare, si nota VertexShader, che gestisce la posizione dei vertici di un oggetto e FragmentShader, che gestisce operazioni sui pixel dell'oggetto rasterizzato. Grazie alla classe SensorManager, dichiarata alla riga 88, è possibile accedere ai sensori del dispositivo. Nelle righe finali si nota il modo in cui il file.cpp viene esportato e collegato al codice java. Vi sono varie funzioni JNIEXPORT, ognuna delle quali ha un preciso scopo: creare il grafico, modificarlo, aggiornarlo, metterlo in pausa e riavviarlo.

```
55      struct AccelerometerData {
56          GLfloat x;
57          GLfloat y;
58          GLfloat z;
59      };
60      AccelerometerData sensorData[SENSOR_HISTORY_LENGTH*2];
61      AccelerometerData sensorDataFilter;
62      int sensorDataIndex;
63
64  public:
65      sensorgraph() : sensorDataIndex(0) {}
66
67      void init(AAssetManager *assetManager) {
68          AAsset *vertexShaderAsset = AAssetManager_open(assetManager, "shader.glslv",
69                                                         AASSET_MODE_BUFFER);
70          assert(vertexShaderAsset != NULL);
71          const void *vertexShaderBuf = AAsset_getBuffer(vertexShaderAsset);
72          assert(vertexShaderBuf != NULL);
73          off_t vertexShaderLength = AAsset_getLength(vertexShaderAsset);
74          vertexShaderSource = std::string((const char*)vertexShaderBuf,
75                                           (size_t)vertexShaderLength);
76          AAsset_close(vertexShaderAsset);
77
78          AAsset *fragmentShaderAsset = AAssetManager_open(assetManager, "shader.glslf",
79                                                           AASSET_MODE_BUFFER);
80          assert(fragmentShaderAsset != NULL);
81          const void *fragmentShaderBuf = AAsset_getBuffer(fragmentShaderAsset);
82          assert(fragmentShaderBuf != NULL);
83          off_t fragmentShaderLength = AAsset_getLength(fragmentShaderAsset);
84          fragmentShaderSource = std::string((const char*)fragmentShaderBuf,
85                                             (size_t)fragmentShaderLength);
86          AAsset_close(fragmentShaderAsset);
87
88          sensorManager = AcquireASensorManagerInstance();
89          assert(sensorManager != NULL);
90          accelerometer = ASensorManager_getDefaultSensor(sensorManager, ASENSOR_TYPE_ACCELEROMETER);
91          assert(accelerometer != NULL);
92          looper = ALooper_prepare(ALOOPER_PREPARE_ALLOW_NON_CALLBACKS);
93          assert(looper != NULL);
94          accelerometerEventQueue = ASensorManager_createEventQueue(sensorManager, looper,
```

```
extern "C" {
JNIEXPORT void JNICALL
    Java_com_android_accelerometergraph_AccelerometerGraphJNI_init(
                      JNIEnv *env, jclass type, jobject assetManager) {
        (void)type;
        AAssetManager *nativeAssetManager = AAssetManager_fromJava(env, assetManager);
        gSensorGraph.init(nativeAssetManager);
    }

    JNIEXPORT void JNICALL
    Java_com_android_accelerometergraph_AccelerometerGraphJNI_surfaceChanged(
            JNIEnv *env, jclass type, jint width, jint height) {
        (void)env;
        (void)type;
        gSensorGraph.surfaceChanged(width, height);
    }
```

Ho dunque analizzato il codice C++ che, insieme al resto del progetto, porterà alla creazione dell'applicazione. Dalla riga 17 alla 21 del codice C++ sono stati inseriti i valori riguardanti la frequenza di aggiornamento del grafico ed il relativo periodo (frequenza = 100 Hz, periodo = 1000000/frequenza). Eseguendo l'applicazione su un dispositivo mobile reale (Samsung Galaxy S7 edge), si ottiene la seguente interfaccia grafica.

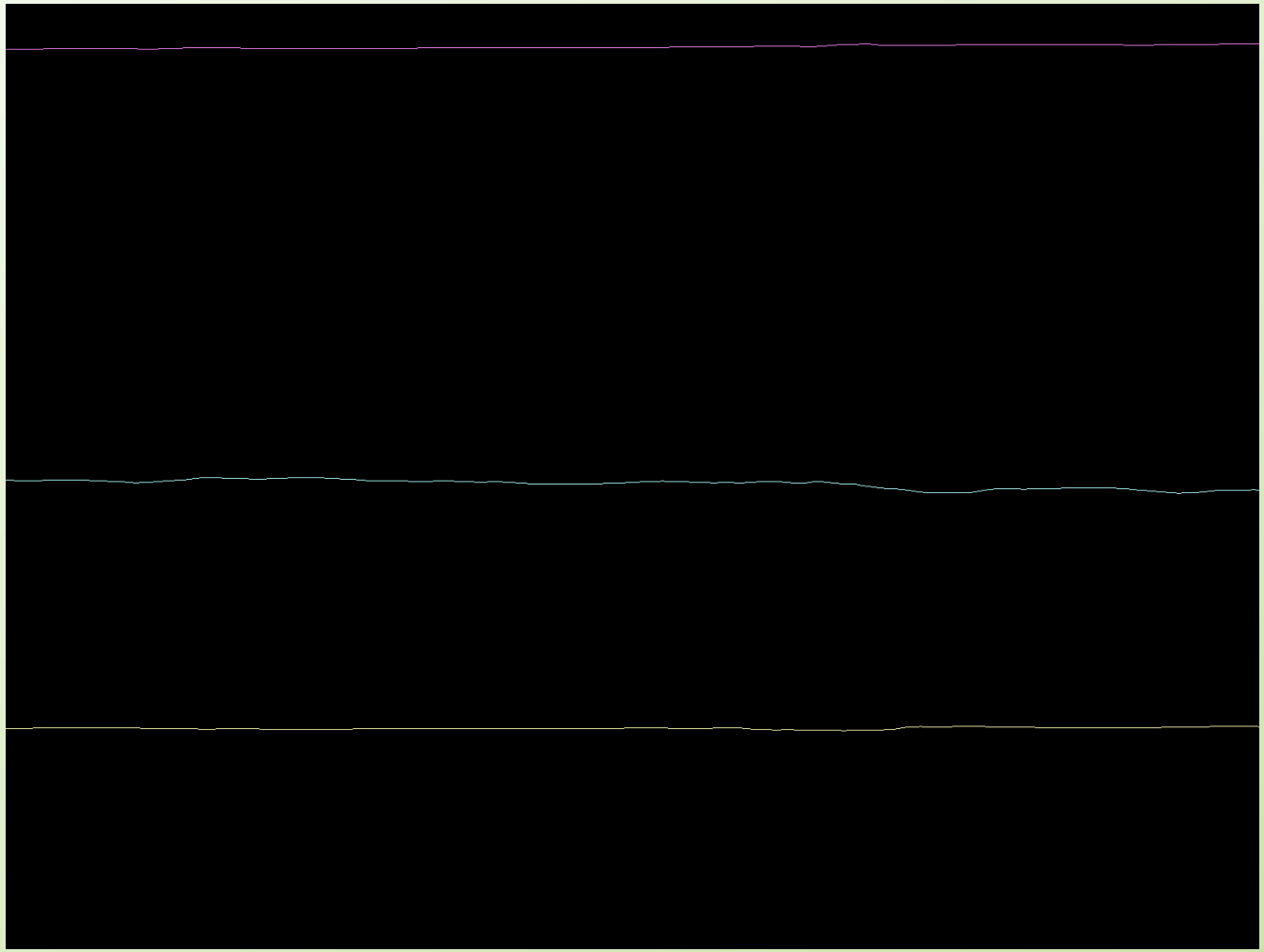


Figura 8- schermata iniziale con dispositivo statico

Nella prima schermata si notano tre caratteristiche che viaggiano quasi parallelamente, per poi notare degli sbalzi (più o meno ampi in base a quanta accelerazione applico al dispositivo) in verticale (lungo l'asse y, caratteristica viola) e/o in orizzontale (lungo l'asse x, caratteristica gialla) e/o in profondità (lungo l'asse z, caratteristica grigia).

Di seguito si riportano i grafici relativi alla variazione d’accelerazione lungo i singoli assi cartesiani.

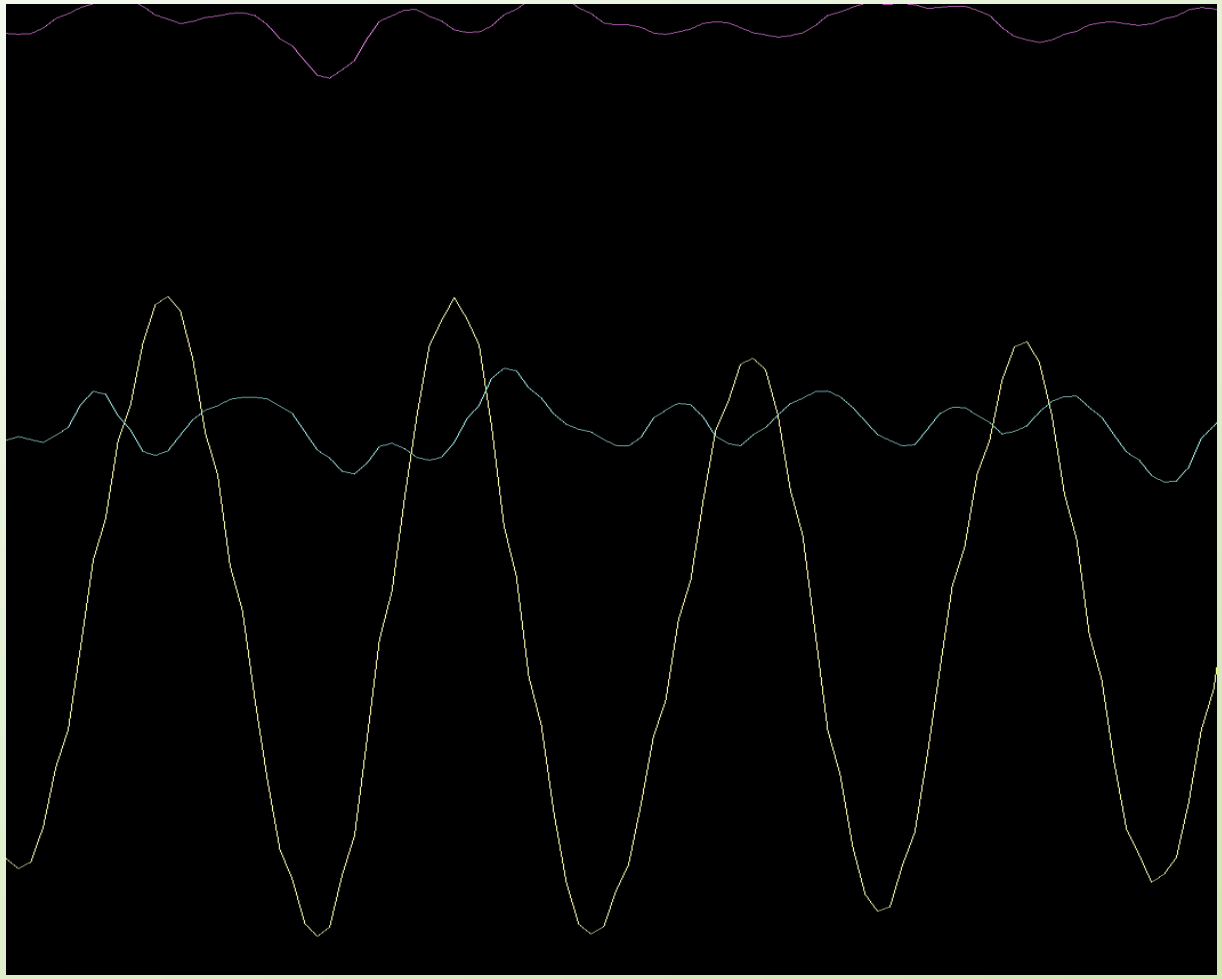


Figura 9- variazione d’accelerazione lungo l’asse x

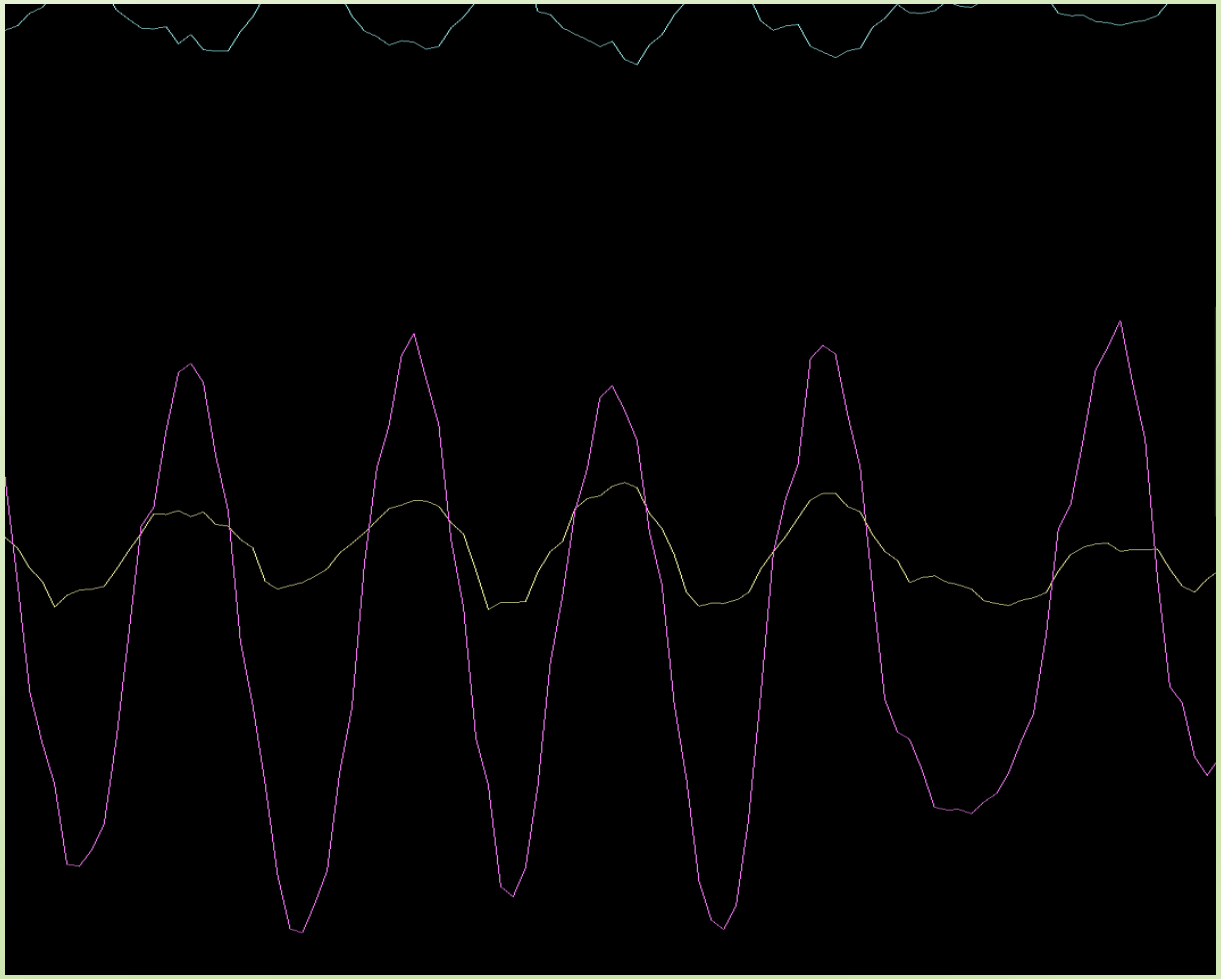


Figura 10- variazione d’accelerazione lungo l’asse y

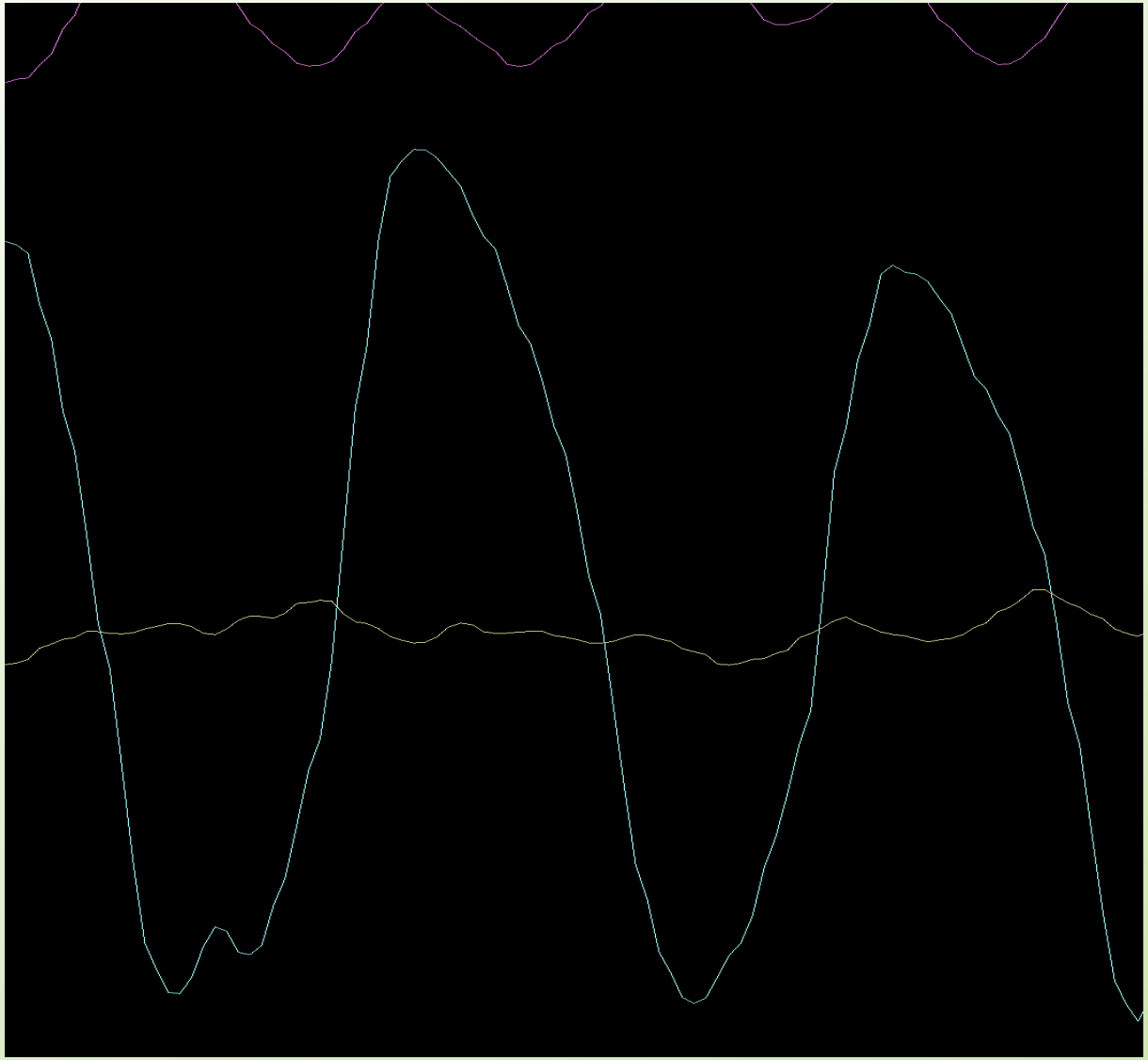


Figura 11- variazione d’accelerazione lungo l’asse z

Variando il valore della frequenza si ottiene una modifica del grafico. Infatti, diminuendo la frequenza di aggiornamento di una grandezza (10 Hz), si ottiene un andamento a gradino del grafico (fig.12), dovuto al relativo aumento del periodo a causa della sua inversa proporzionalità con la frequenza.

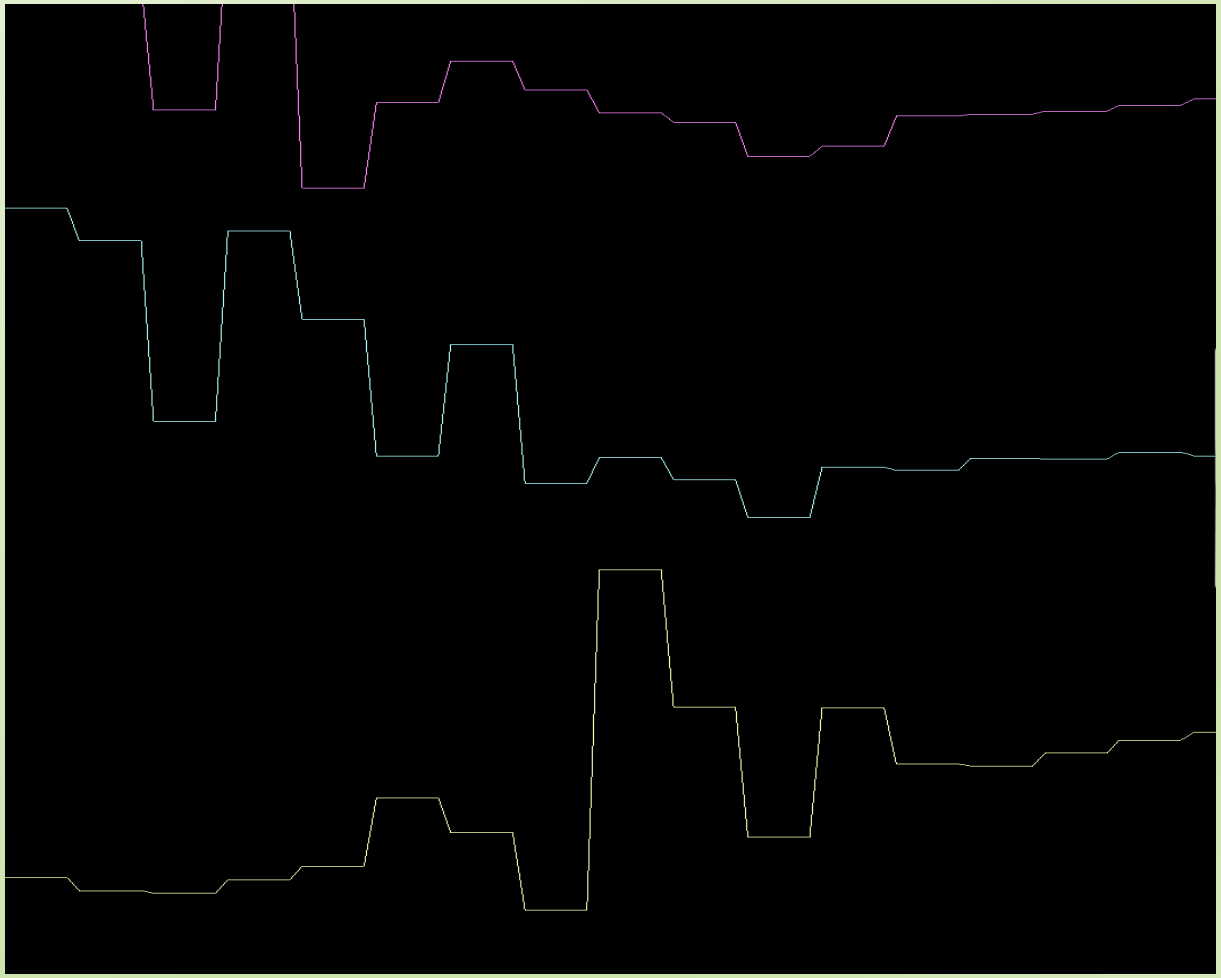


Figura 12- variazione d’accelerazione con andamento a gradino