Rendering Subtitles In Android

Shashanka K and Ravindra GV, Ittiam Systems Pvt Ltd

Abstract

The convergence of Mobile phone and Portable Media Player has resulted in the emergence of powerful media centric smart phones. The multimedia capability of these smart phones is playing a significant role in their success. Support for subtitles is one of the key multimedia features, especially in regions, where consumers like to watch non-native content with the help of subtitles.

Android has become one of the most popular smartphone operating systems. However, Android, as of Gingerbread version, does not have any support for rendering subtitles. So smartphone vendors have to either add subtitle support by themselves or integrate a third party solution. In this paper, we have attempted to capture major challenges in supporting subtitles in general and on Android in particular and described possible approaches to overcome those challenges.

Specifically, in this paper, we have explained subtitles, various types of subtitles, and challenges in supporting subtitles in a multimedia player. Later, we have described architecture of a typical subtitle rendering system. We have also explained, various blocks in the architecture. Thereafter, we have provided two methods of implementing subtitle system on Android, highlighting their advantages and disadvantages. Finally, we have provided an overview of Ittiam's solution for rendering subtitles on Android.

Contents

Introduction

Subtitles can be defined as the textual representation of dialogues in films and television programs. They are usually displayed at the bottom of the video screen in synchronization with audio and video. Subtitles are typically available in multiple languages. They either help viewers with disabilities or viewers who cannot understand the spoken dialogue or accent.

Types of subtitles

Subtitles are always used along with the relevant video content or movie. Based on the locality of the subtitle data with respect to video content, subtitles can be categorized into three types, embedded, internal and external subtitles.

- **Embedded** - Video data will have subtitle data embedded onto it. Therefore, video rendering itself takes care of subtitle rendering as well.

- **Internal** - Internal subtitles exist as a separate stream along with audio and video streams in the media content. For example, DivX content can have XSUB as a separate subtitle stream in the same file. Subtitle stream could be in either text format or bitmap format. Media content can have multiple streams of subtitles, one for each language.

- **External -** External subtitles exist as separate files, not tied to particular media content. These provide user the freedom of using subtitle content of preferred language. External subtitles mostly exist in text format. Some of the popular external subtitle file-formats are SAMI, SUBRIP, SUBVIEWER, MICRODVD and SSA. Among the various types of subtitles, external subtitle is the most common.

Contents of a subtitle file

The typical subtitle file contains the subtitle text and the corresponding presentation time. The timing information is used to display the subtitle data at appropriate time, such that audio, video and subtitles are synchronized. The subtitle files can have additional information pertaining to
- Language of subtitle data. Number of languages present.
- Font information like size, color and typeface.
- Formatting of subtitle data.
- Position on screen where subtitle needs to be displayed.

Character encoding format of subtitles

The character encoding format of subtitle text content is typically ASCII for English and Unicode for other languages. Unicode supports the character sets of all languages. However, this may not always be the case. For example Korean subtitles are mostly available in CP949 character encoding format.

Multiple subtitle tracks

Some subtitle file formats like SAMI support more than one track, allowing a single file to contain subtitles of different languages. Internal subtitles can contain multiple tracks (for e.g. DivX content can support up-to 8 different subtitle tracks).

Challenges in rendering subtitles

Rendering of subtitle involves fetch stage (retrieval of subtitle data), prepare stage (preparation of subtitle data according to the configuration), present stage (timed presentation on device screen) and finally control stage (control of subtitle playback). Functionalities and challenges involved in each of these stages are addressed in the following sub-sections.

> **Fetch, Prepare, Present and Control are the different stages in subtitle rendering**

Fetch

Fetch is the first stage in subtitle rendering. The core functionality is to fetch the subtitle data along with timing and other information from the subtitle content and pass it to next stage. In short, it acts as a subtitle data source to the system. The first challenge is to locate the subtitle content. For external subtitles, content could be in the same location as that of corresponding video content or elsewhere.

Following are other important challenges involved in fetch stage

- **Support for multiple file encoding formats.** - Generally external subtitle files are encoded in ASCII or Unicode format like UTF-8, UTF-16, UTF-32. To retrieve information, data must be decoded before interpretation. The module implementing fetch operation must be aware of byte order masking and Unicode character set.

- **Support for multiple character encoding formats -** Subtitle content creators use different character encoding format for different languages based on the popularity of the format in respective language speaking countries. This stage therefore has to understand individual encoding formats. For e.g. as mentioned earlier, Korean subtitles are mostly encoded using CP949 format.

- **Detection of file format for external subtitle content -** There is no international body to govern subtitle file format specification. So subtitles vary widely, making file format detection an issue. In addition, the file extension is not reliable.

- **Retrieval of correct language information -** The metadata in the subtitle content are not always reliable. Wrong information could affect functioning of other stages in the subtitle rendering system.

- **Retrieval of correct subtitle data -** Subtitle data pertaining to different languages could be interleaved in the content. Badly created subtitle content may have characters belonging to different languages in the same track.

- **Retrieval of subtitle data for internal subtitle -** The existing extractors may not always support internal subtitles. For example native MP4 extractor available in Google's Multimedia framework, Stagefright, does not support text stream. The challenge is to add support in MP4 extractor for subtitle data retrieval or to use custom MP4 extractor having the text support.

- **Support for multiple subtitle formats -** The module implementing fetch operation has to interact with rest of the system to provide the subtitle data. These interactions are independent of the subtitle file-formats. Therefore it is expected to have a generic interface for interaction.

Prepare

This is the second stage in subtitle rendering system. It receives subtitle information and data from the fetch stage. The core functionality is to prepare the subtitle data so that it can be used with the video for rendering in the next stage. Following are the key challenges encountered.

> **Translation of text to bitmap is a major challenge in Prepare stage.**

- **Translation of text data to bitmap -** Text has to be converted to raw video format like YUV or RGB, before it can be rendered over video screen.

- **Support for multiple character encoding formats** - The module implementing prepare stage must understand various encoding formats. Font rendering libraries that translate text to bitmap may support only a few of the encoding formats. Hence the module may have to map encoding format in the file to a standard encoding format like Unicode.

- **Support for compressed bitmap decoding -** Internal subtitle stream can exist as a compressed bitmaps, for example XSUB in DivX content. This stream must be decoded to generate a bitmap.

- **Support for dynamic font configuration -** The user at any point of time can change the font size, color and font type of the subtitle during the playback. Users would expect to see these changes immediately.

Present

This stage receives prepared subtitle data along with timing information. This stage renders the subtitle data on the display devices while maintaining time synchronization with video. Following are the key challenges encountered.

- **Synchronization with Audio and Video -** The timestamp associated with subtitle data may not always be in sync with media clock time. Therefore subtitle time needs to be converted so that subtitle is presented along with video in synchronization.

- **Support for subtitle data having multiple lines -** Subtitle data can either be a single line or multiple lines. If it is multiline, care must be taken to format each line with respect to each other and with respect to the video dimensions as well.

> **Real time blending of subtitle data onto the video is a major challenge in Present stage.**

- **Support for Centering -** The subtitle data needs to be put onto the video so that it comes at the center of the screen.

- **Support for Wrapping -** Sometimes long subtitle data can get sliced off. In such cases, subtitle data needs to be wrapped around. This is achieved through breaking of single line into multiple lines. However, wrapping should not result in slicing of the words in the middle.

- **Support for Bordering -** If the color of the subtitle is white and the video also contains a lot of white content then the subtitle data tends to disperse into the background and cannot be differentiated from the video.

- **Support for blending of subtitle on video frame -** The bitmap of the subtitle data will typically be in grayscale format while the video will be in YUV format. There is a need to convert the bitmap to the correct format before the blending can be performed.

Control

User through the application may want to control the subtitle display. The module that implements control stage provides a mechanism to the application to control subtitle display. It acts as a messenger between application and the other modules implementing earlier three stages. This section deals with the challenges involved in controlling the subtitles during the playback.

Some of the core functionalities are

- **Providing URL of subtitle content –** Application should be able to provide URL or path of the subtitle file to be rendered.

- **Supporting multiple tracks -** For subtitle content with multiple tracks, user must be given control to choose the preferred track during initialization as well as run time. Seamless transition of tracks is another challenge.

- **Trick mode support -** During video playback, user may choose to alter or trick the play position. User may want to seek, fast forward, rewind or even pause. In case of pause current subtitle should continue to be displayed. For rest of the modes, subtitle data should come from file position corresponding to the new play position. This requires appropriate communication between modules through controller.

> **The control module manages the fetch, prepare and present operations**

- **Dynamic font configuration -** The font parameters selected by the user, has to be communicated to appropriate modules. Care must be taken towards immediate application of the new configuration.

- **Support for subtitle toggle -** The user may choose to toggle subtitles during playback. Controller needs to communicate this to different stages of subtitle rendering pipeline. Key challenge here is to make sure that effect takes place immediately.

Subtitle Rendering Architecture

In the previous section, we described various stages involved in subtitle rendering. In this section we mention different modules in the subtitle system and the interaction between those modules. A typical subtitle rendering architecture is as shown in figure below.
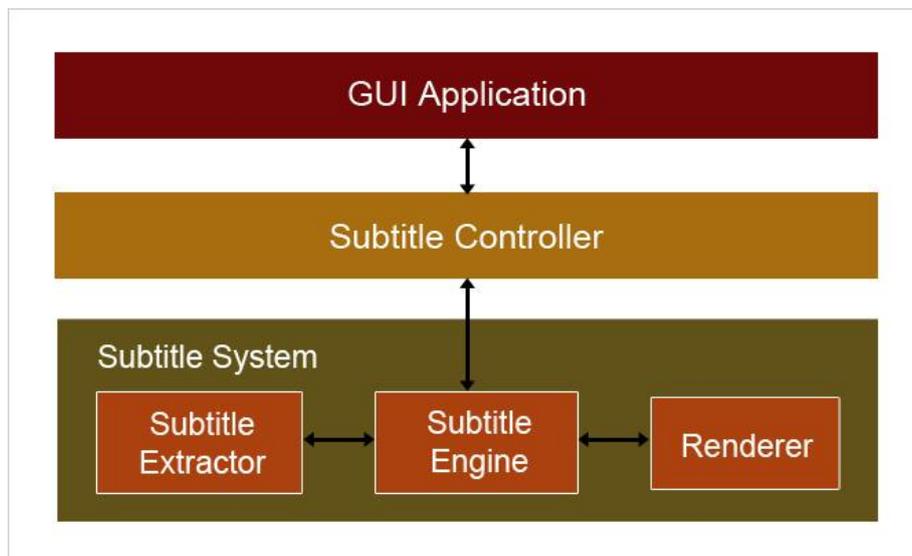

Figure 1 Subtitle Rendering Architecture

GUI Media Player Application

This represents the GUI application that end user uses for video playback with subtitles. This interacts with subtitle system for rendering subtitle. Scope of the application is defined by the subtitle rendering use case i.e. whether the rendering occurs at application layer or the framework layer.

Subtitle Controller

This module implements control operations. Controller sits outside the subtitle system and controls the subtitle system. All interactions with the subtitle system happen through the Subtitle Controller.

Subtitle Extractor

Subtitle Extractor module provides the interface to Subtitle Engine to access and control subtitle data. It implements the Fetch functionality discussed in the previous section.

Following are the responsibilities of this module.
- Detection of Subtitle content
- Locating the subtitle file (external).

- Detection of subtitle file format.
- Providing metadata information.
  - Language information
  - Font information
- File encoding information
- Language encoding information
- Retrieving subtitle data along with timing information.
- Repositioning within the stream / file on seek (trick mode) based on the new play position provided by the Subtitle Engine.
- Dynamic Language / track selection.

To support multiple file formats, it is implemented as a subsystem, abstracting multiple subtitle extractors like SAMI, SRT, SSA and providing a generic interface.

Subtitle Engine

Engine module is the heart of the subtitle system. It interacts with the controller, extractor and the renderer module and manages the subtitle system. The core functionality of this module is to prepare render-able subtitle data.

Following are the responsibilities of this module.
- Procuring subtitle data from Subtitle Extractor.
- Generation of grayscale bitmap for text subtitle data. Open source libraries like FreeType can be used for bitmap generation. These utilities usually support Unicode. Therefore to support non-Unicode character set like CP949, mapping to Unicode character set is required.
- Generation of grayscale bitmap from compressed bitmap through decoding.
- Applying font configuration during bitmap generation for text subtitle data.
- Providing data to renderer.

Renderer

Renderer module provides an interface to display subtitle data on the device screen. The core functionality is to present the subtitle data. Following are the responsibilities of this module.
- Apply formatting to the subtitle data received from subtitle engine.
  - Formatting involves centering, wrapping, bordering and coloring.
- Blending of bitmap onto video.
- Timed presentation of subtitle data to display device.

Adding Subtitle support to Android

There are two approaches to adding subtitle support to Android multimedia stack
- At media framework layer
- At application layer

In this section we will see how the typical architecture diagram can be applied to both the approaches.

Media Framework Layer

In this approach, media framework (either Stagefright or OpenCore) on one side acts as subtitle controller, interacting with the subtitle subsystem to achieve subtitle rendering and on the other side interacts with Android application through JNI layer. It takes requests from Android Application and executes them through subtitle system. Figure 2 shows the subtitle system interacting with the framework.
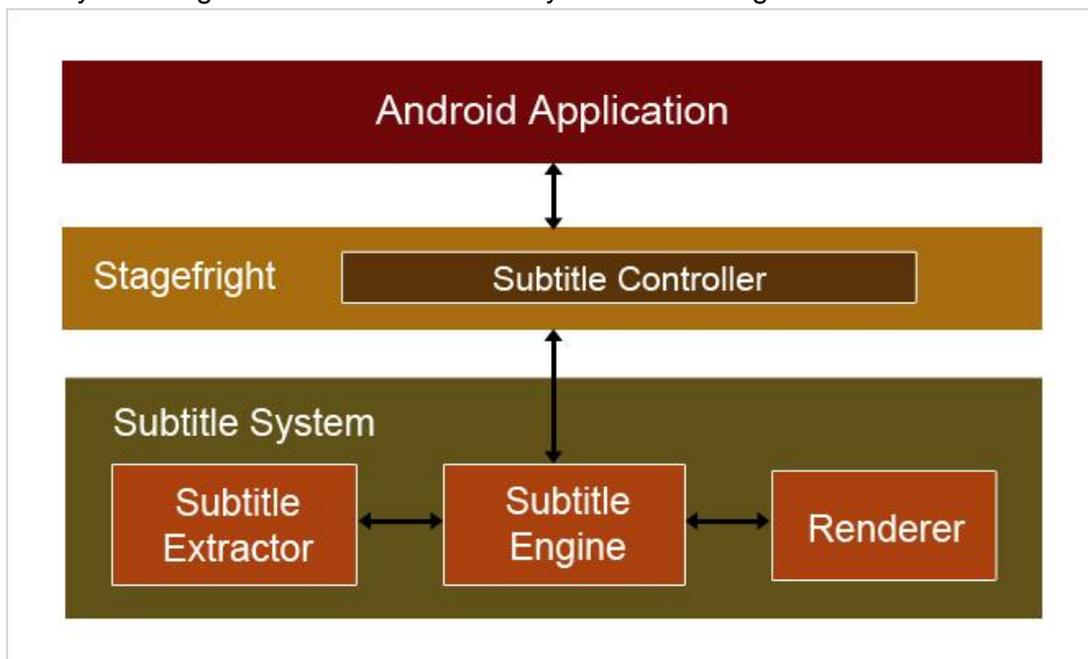

Figure 2 Media Framework Based Subtitle System

Awesome player is a media player engine available in Stagefright. This is extended to implement functionalities of the controller. The augmented Awesome player provides additional APIs to Android Application framework, which are used by the Android Application for subtitle rendering. The entire subsystem is implemented in native language (C / C++) and integrated into Awesome player.

Subtitle extractor and engine communicate through generic interface defined by the extractor. Subtitle engine receives text data from extractor and prepares the subtitle to be rendered. Renderer here is a thin

wrapper that takes subtitle bitmap from engine and video frame from awesome player as input. It then applies formatting and blends the subtitle data onto video before dispatching to device screen.

Pros
- Subtitle can be rendered even when the video is redirected to the video out. (TV Out use case)
- Enhances Android Application Framework, so that any Android application can make use of newly added subtitle APIs.

Cons
- Since software blending is performed it might not be the most efficient method.
- Adding support for new character set is cumbersome, since it involves upgrading subtitle extractor and engine.

Applicability
- This is the only approach supporting TV-out use case.
- The approach allows making use of internal subtitle supported by native parsers.

Application Layer

In this approach, the subtitle rendering is achieved at the application layer. The GUI Application implemented in Java, houses the subtitle controller. On one side, it interacts with end user and on the other side it interacts with Subtitle Extractor and Subtitle Engine and uses Android SDK for rendering. The below figure depicts typical subtitle rendering architecture at application layer.
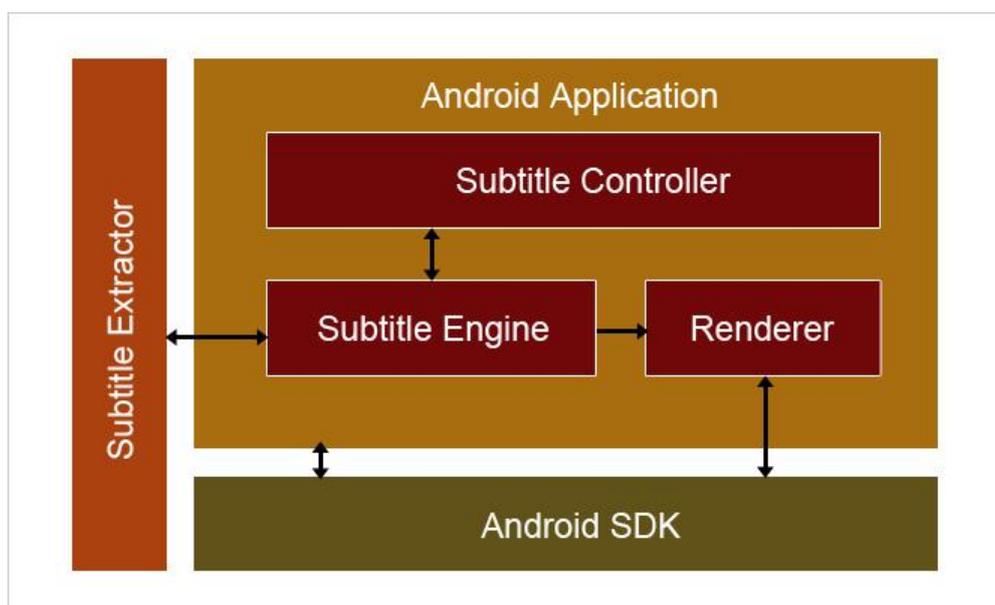

Figure 3 Application Based Subtitle System

12

Ittiam Systems Pvt Ltd
The Consulate, 1, Richmond Road
Bangalore - 560025, India

Subtitle Controller & Subtitle Engine are implemented as one of the classes in the Android application. The controller would interact with MediaPlayer class of Android and subtitle engine to achieve timed presentation. User requests like font configuration, trick mode are supported through this module. The application developer may choose to combine controller and engine for convenience.

Subtitle extractor here is implemented in Java and provides Java APIs to subtitle engine. This consists of individual Java subtitle extractors interacting through common interface.

Renderer, here, is a thin wrapper that uses appropriate Android SDK API to render subtitle data on video based on subtitle data type. Different APIs will be used to render bitmap and textual data. The actual blending occurs at hardware layer.

Pros
- Availability of core subtitle functionality at application makes this highly portable
- The hardware blending makes this approach efficient in terms of CPU usage.
- Availability of time tested Java Packages makes Subtitle Extractor implementation fast, scale-able and stable.

Cons
- This approach cannot be used to support TV-out use case.
- This approach can be used only for external subtitles.

Applicability
- This approach can be used for subtitle rendering on device screen.

Ittiam's subtitle solution for Android

Ittiam has been offering Media System SDK since 2005. This includes multimedia framework, engine, extractors, composers, network protocol stacks, audio and video decoders and many more. Ittiam's multimedia components have been integrated into products of leading consumer electronics giants.

Ittiam offers both application level and framework level solution for subtitle use case. Ittiam has

delivered the framework level subtitle rendering solution on Android, to leading Korean consumer electronics giant.

Salient feature of the solution are
- Support for Chinese, Korean, Japanese and English font.
- Support for SAMI, SRT, SSA, MicroDVD and Subviewer subtitle file formats.
- Support in OpenCore as well as Stagefright frameworks.
- Support for dynamic font configuration.

- Support for bordering, centering and wrapping.

Figure 4 English Subtitle displayed by Ittiam Solution

Figure 5 Korean Subtitle displayed by Ittiam solution

Conclusion

In this paper, we provided an introduction to subtitles, and challenges in supporting subtitles in a multimedia player. We also described typical subtitle rendering architecture, which can be implemented on any multimedia framework. We explained two methods of implementing subtitle system on Android, highlighting their advantages and disadvantages. In the end, we provided a high level overview of Ittiam's industry proven, multimedia solution on Android and how it handles the challenges of subtitle rendering.

References

1. Introduction to the subtitles - http://en.wikipedia.org/wiki/Subtitle_(captioning)
2. Android on Wikipedia - http://en.wikipedia.org/wiki/Android_(operating system)
3. Basics on Android - http://developer.android.com/guide/basics/what-is-android.html
4. FreeType - http://www.freetype.org/index2.html
5. CP949 format - http://en.wikipedia.org/wiki/Code_page_949

Contact

E-mail: **mkt@ittiam.com**
Website: **www.ittiam.com**

Disclaimer

This white paper is for informational purposes only. Ittiam makes no warranties, express, implied or statutory, as to the information in this document. The information contained in this document represents the current view of Ittiam Systems on the issues discussed as of the date of publication. It should not be interpreted to be a commitment on the part of Ittiam, and Ittiam cannot guarantee the accuracy of any information presented after the date of publication.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Ittiam Systems. Ittiam Systems may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Ittiam Systems, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.