Audio effects with G-Streamer

By
Sharath Holla K,
Ittiam Systems Pvt. Ltd.
Bangalore, India
sharath.holla@ittiam.com

ARM dominates the market space for the consumer electronic devices with a wide range of ARM® processor cores such as ARM9E (ARMv5T), ARM11 (ARMv6) and Cortex™ A8 with Neon™ (ARMv7). These devices use Multimedia Framework to easily develop simple applications like media player/recorder as well as complex ones like multi-track player, media converter, video conferencing etc.

A wide range of multimedia frameworks have evolved in the last few years to enable these applications. One of the widely used frameworks is G-Streamer. It can support applications ranging from simple audio mixer to complex A/V player.

One of the most popular applications is that of a music player. It has become a norm to not only support popular music format, but also to enhance the listening experience of the end user. This is done by using audio post processing effects which ranges from equalizer to 3D surround effects.

This paper discusses the fundamental aspects of some of the audio post processing effects and integrating them to G-Streamer on Linux.

G-Streamer

G-Streamer is a popular multimedia framework, which allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming, and editing. G-Streamer allows any audio/video codecs to be plugged in their framework.

G-Streamer's development framework is suitable for any type of streaming multimedia application. The G-Streamer framework is suited to easily to write applications that handle audio and video. It isn't restricted to audio and video, and can process any kind of data flow. The framework is based on plug-ins that will provide the various codecs and other functionality.

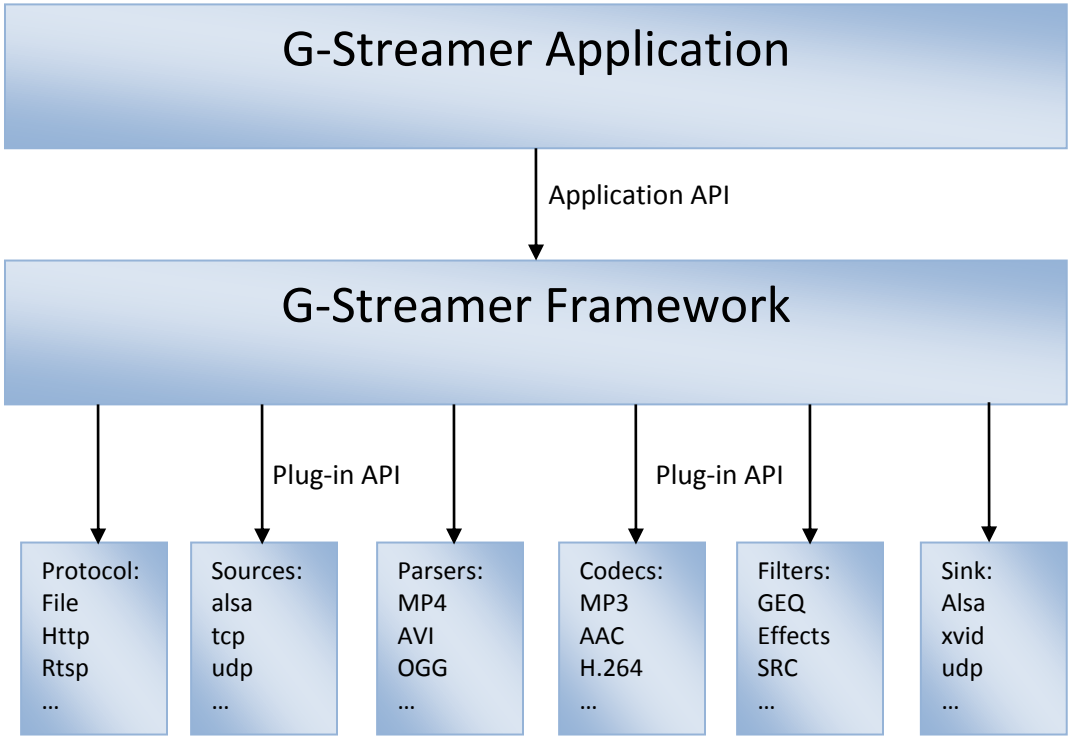

Figure 1: G-Steamer design

The Plug-ins can be classified into following groups.

- Protocol handlers
- Sources for A/V
- Formats which include parsers, creators, Muxers, Demuxers, metadata handlers, etc
- Codecs which include audio and video encoders and decoders
- Filters which include mixers, convertors, effects
- Sink for A/V

Plug-ins are shared libraries that are loaded at runtime whose properties can be set using the GObject properties.

The parsers, codecs etc. are plugged into G-Streamer, with a set of plug-in API. G-Streamer also provides a set of API's for writing applications using G-Streamer framework. In the application, the plug-ins can be linked and arranged in a pipeline. This pipeline defines the flow of the data.

In G-Streamer, a module which generates or processes or consumes the data is called as an **element**. An element has one specific function which can be read from source or decode an AAC data.

Each element has one or more **pad**s, through which the data flows. Pads are element's input and output, where you can connect other elements. A **sink pad** accepts the incoming data, whereas a **source pad** outputs the data. Links are only allowed between two pads when the allowed data types of the two pads are compatible. Data types are negotiated between pads using a process called caps negotiation. Source and sink elements have only source and sink pads, respectively.

Various elements can be linked together using their pads to form a **pipeline**. Once started, pipelines will run in a separate thread until you stop them or the end of the data stream is reached. An example pipeline is shown below.

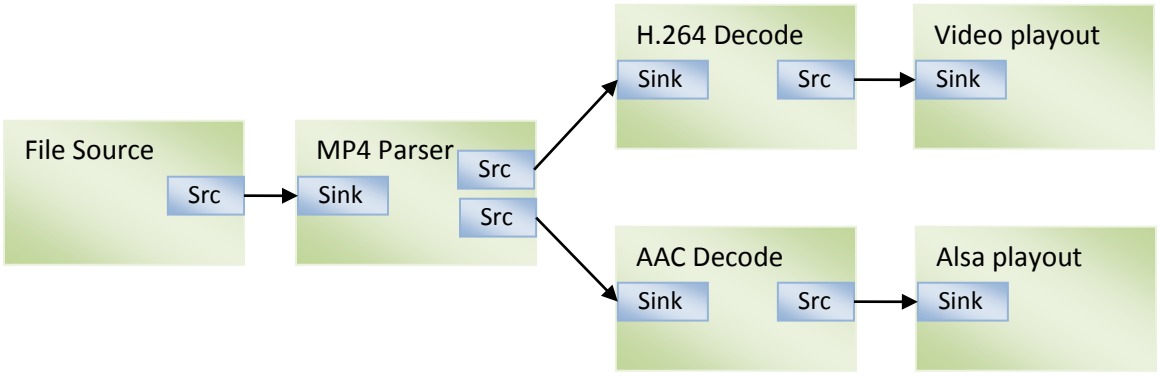

Figure 2: G-Streamer pipeline

In the above figure, Elements file source, MP4 parser, H.264 decode etc. are elements which perform operations like file read, mp4 parsing, component decodes and device playout.

The G-Streamer provides essential media features. It enables developers to build system that supports music applications, video creation and playback, video telephony, real time streaming and more. The framework package includes a huge range of open-source multimedia codecs like MP3, AAC and Vorbis for audio, H.263, MPEG4 for video. Parsers to parse MP4, AVI, OGG, FLV streams are also provided. These would help in developing high end products with minimal time to market.

Addition of audio post processing effects would include addition of another combined element performing different audio effects or series of elements each performing an audio effect.

Audio effects

Apart from traditional devices like Portable media players, media players are used in other devices like mobile phones, STBs, etc. With large number of people using their mobile phones as major source of entertainment, line separating mobile phones from media players is getting thinner by the day. Owing to small sizes of these devices, they are not ideally suited for great listening experience. These hardware limitations necessitate the use of software techniques to provide hi-fi audio quality. A well designed set of audio effects can help the manufacturer achieve high quality audio without increase the cost of the device significantly.

Major limitations of these devices in providing good quality audio playout are as follows.

- Due to their small size, the low frequency response is generally very poor
- The mid and high range frequency responses of the speakers are not flat and exhibit significant fluctuations, thus altering the spectral balance.
- A few devices have two speakers to provide stereo effect, but close proximity of the speakers causes audio signal

A suite of audio effects can be used to solve the above limitations.

Overview of some of the audio effect algorithms are given in the following sections.

Loudness Compensation

The spectral balance of an audio content is perceived to be different when played out at a level different from that used during recording. This need for correction arises due to the inherent non linearity in human aural perception. This is evident in the bass region especially when listening at lower volume levels. To improve the listening experience at low volume levels, algorithms like loudness compensation can be used.

Stereo Widening

Few high end devices have two speakers placed very close to each other due to space constraint. This causes a significant portion of the audio signal from the left speaker to reach the right ear of the listener and vice versa. This is termed as “acoustic crosstalk” and results in a mono like sound image.

Stereo widening post processing algorithm uses sophisticated signal processing to cancel the acoustic crosstalk at the listener’s ears, thus creating a wide stereo image. Also, it is possible to create virtual speakers at wide angles with respect to listener, by means of HRTF (Head Related Transfer Function) processing to further enhance the listening experience.

Virtual Surround

Experiencing true surround sound requires 5.1 speaker setup that consumes a lot of space and also adds clutter due to all the wiring. Such a setup may be undesirable due to either space constraint or aesthetic reasons. If devices like mobile phones are used, then such a setup is not possible. In such cases it is required to render multichannel content over just two speakers. Downmixing of multichannel audio to two channels by simple matrixing results in a loss of spatial information embedded in the multichannel content.

When music is listened through a headphone the position of the original performance and the richness provided by room acoustics are eliminated. This creates the impression that the source of sound is present inside the head and makes the perception of the performance sound artificial. This general lack of position and richness also creates a fatigue to the listener after long periods of listening. Hence there exists a great gap between listening through headphone or stereo speakers and listening through home theatre system

Sophisticated signal processing like Virtual Surround can be used to create an illusion that the sound is emanating from 5 speakers around the listener when listening over two speakers. In the case of headphones, Virtual Surround pushes the sound image outside the listener's head, thus avoiding listening fatigue that is so typical of headphone listening experience. This makes the sound reproduction through headphone sound natural and comfortable

Parametric equalizer

Equalization is a process of modifying the frequency envelope of the audio signal. This is needed in scenarios where the audio signal needs tuning to make it more suitable to the preference of the listener or to make it suitable to the speakers in use. Parametric equalizer gains over conventional graphic equalizer in allowing the user not only to change the boost or cut level of the frequency range but also the center frequency and the range bandwidth.

It is used to compensate for uneven frequency response of the built in speakers. It is made up of one or more parametric filters connected in cascade. By varying the gain of these filters, it is possible to generate the inverse of the speaker frequency response, in effect achieving flat overall response. The number of filters required would depend upon the speaker response to be compensated and the accuracy desired. Once the speaker response is known, then the filters can be designed manually by trial and error approach or the design could be automated by means of a software program.

Dynamic Range Compression

Dynamic Range is difference between the highest and the lowest level in the signal. The dynamic range of audio is reduced in order to make the overall audio level more even sounding and audible in usual listening environments. Under natural conditions, ears can hear soft sounds in a quiet environment, but not in a noisy environment. In a moving car for example, background noise can overpower soft sounds. Turning the volume up louder makes the soft sounds audible, but the loud sounds would become too loud. However, making the volume

lower, and thus acceptable for the loud sounds, would make the soft sounds inaudible. To make both the soft and loud parts of a sound audible at the same time, compression is used. Compression (when combined with an overall gain increase) increases the volume of both the soft and loud parts of sound; though the soft parts are increased more than the loud parts. The overall effect is that a compressor makes softer sounds relatively louder and louder sounds seem relatively softer in comparison.

DRC can also be used in portable players and in-car entertainment systems where the external noise level is high and the soft sounds in the music become in-audible.

Enabling Audio effects on G-Streamer

Integration of effects into can be split into two broad efforts.

- Creation of G-Streamer plug-in for audio effects
- Integration of the plug-in to the application

Creation of G-Streamer plug-in for audio effects

An element can be plugged into G-Streamer framework using Plug-in APIs. For complete details on creation of plug-in, please refer to plug-in writer's guide [*].

G-Streamer is implemented as object oriented framework. Each element has one or more pads. Elements receive data on their sink pads and generate data on their source pads. Since the pads play a very important role in how the element is viewed by the outside world, a mechanism is implemented to describe the data that can flow or currently flows through the pad by using capabilities.

An element may be required set/get some parameter related to the stream. For example, the parametric equalizer might require details on the frequency bands. These are communicated using the properties of the element. Each property should be installed into the element before it is used.

A G-Streamer element can be in any of the four states mentioned below.

• GST_STATE_NULL
• GST_STATE_READY
• GST_STATE_PAUSED
• GST_STATE_PLAYING

GST_STATE_NULL is the default state of an element. In this state, it has not allocated any runtime resources and cannot handle any data.
GST_STATE_READY is the next state that an element can be in. In the READY state, an element has all default resources allocated. However, it has not yet allocated or defined anything that is stream-specific

GST_STATE_PAUSED, GST_STATE_PLAYING is the state in which an element is ready to accept and handle data. Both states are same for all elements, except sink elements (such as audio playout element). Sink elements only accept one single buffer of data and then block, so that pipeline stops issuing buffers for other elements and hence playout gets paused.

For audio effect plug-ins, we do the creation of codec, while changing from NULL state to READY state. Similarly de-allocation of all resources should happen when there is a state-change other way around
The data enters an element through a sink pad. After the processing the output data is pushed onto the source pad. The G-Streamer will call a specified function in whenever it wants any data to be processed by the element. This function is called 'chain' function and it should be specified for each sink pad during the init constructor function.

When end of stream is reached, the G-Streamer signals the event EOS to the pipeline, which flows through pads of each and every element, similar to how data flows. So an element receives the EOS event through its sink pad. We should register a function, which does the event handling for that sink pad, during the init constructor function.

Integration of the plug-in to the application

As G-Streamer is a pipeline based framework, effort involved in integration of the effects plug-in would be to add effects plug-in in the audio path of the pipeline. Typical pipeline with MP4 file as input is given below.

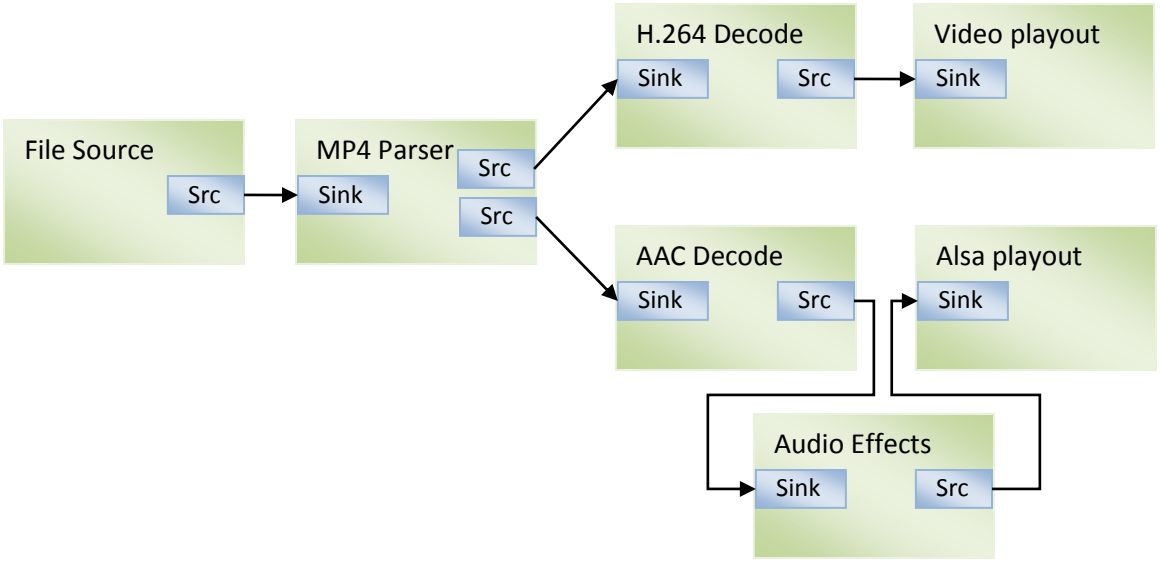

Figure 3: G-Streamer pipeline with Audio Effects

Care should be taken that the sink pad of audio effects should be compatible with source pad of AAC decoder and source pad of audio effect with sink pad of Alsasink.

Typically, a user would like to experience more than one audio effect at a time, and then determine which set of effects and presets suit them best. In such scenario, more effects can be added in series and use them without much of the effort. To avoid multiple plug-ins, in the same plug-in, different effects can be combined with APIs to enable or disable different effects.

Since features like audio effects are tied to the user's choice, it is a requirement to provide control in an application to enable/disable or change parameters of a particular effect while playing music. This is done using G-Streamer APIs to set/get parameters. Application can use or alter properties which are exposed by the plug-in. Graphical interface can be coded to call these functions to change the parameters during music playout.

Conclusion

With more and more devices supporting multimedia applications flooding the market, it has become imperative to develop new applications with minimal time to market. G-Streamer is one of the widely used frameworks which can support most of the multimedia application requirements. Being light weight, the component integration and application development is also very easy making it one of the first choices when selecting the framework for the application development. With the size of these devices shrinking, audio rendering with original quality is becoming a challenge. This can be overcome by using a well designed set of audio post processing effects which enhance the quality of audio output. The integration of these audio quality enhancement effects into a G-Streamer based application is achieved by adding the effect element before audio rendering as explained in the preceding sections.

References

1. G-Streamer Application Development Manual – http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf
2. GStreamer Plugin Writer's Guide – http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/pwg.pdf