High Quality Audio for Mobile Applications

Authors: Jennifer Hruska and Dave Sparks

Music based applications consistently perform in the top five for producing revenue on mobile devices. Likewise, engaging music and audio in games can increasingly propel a game into extreme popularity. Take Guitar Hero, Rock Band and even Angry Birds for examples. The audio synthesizer is the engine that propels this engaging music and therefore integrating a high-quality audio synthesizer into an application is well worth the effort.

In order to meet this effort on battery powered devices with limited processors, the audio synthesizer, and by extension application designers, should be aware of how audio synthesizers work, including some of the tradeoffs between application size, processor load and the quality of the audio. We will explore theses issues in this article.

MIDI - The Format Standard

There are dozens of different file formats that have been developed over the years for storing data for audio synthesis. The most widely accepted standards-based format is the Standard MIDI File (SMF), a standard jointly overseen by the MIDI Manufacturers Association (MMA) and the Association of Musical Electronics Industry (AMEI). The MIDI standard originated as a protocol to transmit a musical performance over a 31.25Kbps serial cable. With limited bandwidth available, the protocol comprises primarily control data such as *Note-On*, signifying the moment when a performer presses a note on a musical keyboard, and *Note-Off*, signifying when the note is released.

The SMF format was created in the 1980's as a means to capture the MIDI data stream into a file for editing on a computer. Events are stored in a stream format with the addition of delta timestamps to mark the amount of time since the last event. The file format is very compact, with a typical files size between 10K and 100K bytes. A similar file stored in a perceptive audio coder format such as MP3 would be 4M bytes. While MIDI behaves similar to a codec at the receiver end, an encoder that can encode an arbitrary audio stream is impractical today.

Synthesizing Audio

MIDI is able to achieve these levels of "compression" because the audio data itself is not stored in the file, only the actions of the performer, similar to the concept of a player piano roll with its coded instructions of what notes to play when, how hard to play them and for what duration. The audio synthesis engine then interprets this data into audio producing actions using a synthesis algorithm. There are two common forms of synthesis used today: Frequency Modulation (FM) and analog synthesis, both of which store just a few simple waveforms and then modulate or change those waveforms in order to simulate real instruments. Because, however, the stored waveforms are very simple the realism of the resulting sound is severely reduced. Real musical instruments have very complex waveforms, which change over time depending on how much force is applied to a string, or air is blown through a mouthpiece, etc.

In contrast, a sampling synthesizer, often referred to as a wavetable synthesizer, utilizes recordings ("samples") of actual instruments as well as synthetic sounds. By varying the playback speed through interpolation, a single recording can be used to synthesis a range of frequencies, thereby spreading a single recording over several notes on a musical keyboard. To simulate the changes a player may employ on a real instrument, the sound is often further manipulated using filters and volume envelopes to dynamically vary the output spectrum.

Sampling synthesizers produce much more realistic sounding instruments than FM or analog synthesizers, but the realism comes at the cost of additional read-only memory for the wavetable data. In contrast, FM and analog synthesizers require less memory to store the algorithm parameters for their sounds, but signal processing requirements tend to be much higher, and, as we've said, the realism of the resulting instrument in compromised.

Audio Synthesis Components

The process of reading a MIDI file and synthesizing an audio output stream from it can be broken into three distinct components: The file parser, the MIDI interpreter, and the synthesis engine.

The file parser reads MIDI data from a file or input stream and reconstructs the timeline from the delta timestamps stored in the file. Timestamps are generally specified relative to the tempo of the musical piece although they can also be specified relative to SMTPE time code. The file parser converts the relative timestamps in the file to absolute time so that events can be fed to the MIDI interpreter at the appropriate time.

The MIDI interpreter acts on the performance data in the MIDI stream. For example, when a *Note-On* event is received, the MIDI interpreter must locate the algorithm parameters that characterize the musical instrument to be synthesized, allocate resources ("voice") to synthesize the note and start the process of synthesizing the note.

The performance data may occasionally request more voices than are available, in which case the MIDI interpreter must determine which notes have priority. "Voice stealing" occurs when an active voice is reallocated to synthesize a new note.

The synthesis engine receives control data from the MIDI interpreter and synthesizes the audio based on the supplied parameters ("program") and, in the case of a sampling synthesizer, the sample data. The output of all the voices is mixed together based on the MIDI controls to render the final audio output.

Sweetening the Audio Output

In addition to the basic algorithmic processing required to synthesize a note, other signal processing may take place including audio filters; chorus and reverb (typically called "side-chain effects"); audio exciter, compressor/limiter, and EQ (typically called "post-processing effects"). These effects are often referred to as "audio sweeteners" and they can greatly enhance the quality of the audio. This is another opportunity for application designers to differentiate and add more value to their applications.

Audio filters are used to vary the spectrum of the synthesizer to simulate changes in brightness such as the natural decay of a piano string. A chorus is a delay line with a variable tap used to simulate multiple voices, providing a richer tone to brass and string section sounds. A reverb is a combination of delay lines and all-pass filters used to simulate the reverberation of different environments such as a concert hall or stadium. All of these effects are normally controlled on an individual instrument level, for example, the brass section can have chorus effects applied without affecting the piano.

An audio exciter brightens the audio by adding harmonics to fill in the upper frequency range, an effect that can help make up for harmonics that may be lacking in the original samples (more on this later). A compressor/limiter maximizes the output signal level by increasing the output gain when the overall volume of the synthesizer drops, a useful effect for audio coming out of mobile speakers in a noisy environment. EQ or equalization can be used to compensate for characteristics of the transducer and acoustics of the mobile device itself or simply for user preferences.

Performance Optimization

Software-based audio synthesis requires a fair bit of processor bandwidth. The actual bandwidth required is highly influenced by the polyphony of the synthesizer (the number of simultaneous notes that can be synthesized), specifics of the algorithm, such as the sample rate, whether it includes additional signal processing stages including individual voice filters, chorus or reverb effects, post-processing enhancements such as an audio exciter or compressor/limiter.

The processor architecture is the next most significant factor. The number of registers, availability of zero wait-state memory such as cache or tightly-coupled memory (TCM), and signal processing capability such as MAC pipelines and saturating arithmetic can all significantly influence performance.

The synthesizer engine should be one that is designed from the beginning specifically for embedded applications. Retrofitted PC or hardware synthesizers generally do not perform as well.

Size IS Important

As previously mentioned, FM and analog synthesizers use a purely algorithmic method of synthesizing instrument sounds, while sampling synthesizers use a mixture of algorithms and recorded audio. Since sample-based audio synthesis has at its core a wavetable of recorded sounds to drive the oscillators, the size and quality of the wavetable is crucial to the resulting quality of the synthesized sound. Therefore, the process of wavetable creation or selection is considered by many to be the most important aspect of a successful MIDI solution. After all, you can have the most elegant synthesizer design possible but if the samples you are playing back are of poor quality, the entire solution will sound bad.

The samples must be free of any background or player noise, of adequate dynamic range, and consistent in loudness and timbre across the range of notes sampled. It is not enough to have balance across one instrument's scale – all instruments must balance with each other when played together in the context of a musical piece. This requires more than engineering finesse and is often a process undertaken by professionally trained musicians with highly discerning ears.

Once the instruments have been sampled, the resulting recordings need to be key-mapped. This is a process in which individual samples are assigned a range of notes they are used for playback on. After the key mapping process is completed, the task of "voicing" or adding the synthesizer control structures is done. This involves musical decisions and programming to take the final set of recordings to a playable state. Time and velocity variant filters are added, amplitude envelopes to modulate the volume over time, pitch modulation, layering of sounds for synthesizer voices, etc. all are done at this stage. In order for the final wavetable to sound correctly with the standard MIDI files available, careful attention should be made towards volume balancing and "mixing" the instrument set so it plays well in a multi-timbral, musical setting.

Small footprint wavetables for mobile handsets and audio players take on an extra set of important tasks that involve several techniques to reduce the size of the wavetable, while maintaining a high quality output. These tasks may involve pitch and time compression techniques, specialized looping and sampling rate reduction, equalization, and many others. To ensure the best results, special consideration should be taken with small footprint wavetables to optimize them for the playback synthesizer and final product application.

Related Standards

Unlike a typical codec, MIDI does not guarantee a specific output for a specific input. Synthesizers from different vendors will produce different outputs from the same MIDI file based on their own algorithms and samples. In contrast, General MIDI (GM) is a joint MMA/AMEI standard that defines a common set of 128 instruments and 47 percussion sounds and the means to select them on any platform that supports it. This gives the author of a music file some assurance that when his or her composition requires a violin, that the platform will attempt to reproduce a violin sound. General MIDI 2 increases the number of sounds available and further defines the behaviors of a compliant platform. However, even the combination of General MIDI and SMF files still cannot assure the quality of the sound that will be reproduced on a particular platform.

To address this limitation, the Downloadable Sounds (DLS) standard was jointly created by the MMA and AMEI to allow content authors to create files of instrument sounds that can be downloaded to a compliant synthesizer. DLS gives the author a standardized method to control the sound of the instruments used to reproduce a musical performance. DLS-2 increases the capability of DLS-compatible synthesizer and provides for both forward- and backward-compatibility. DLS-2 (under the moniker SASBF) was adopted by the MPEG standards body in a joint effort with the MMA as part of MPEG-4 Structured Audio.

Content Authoring Tools

One very important aspect often overlooked by application designers when choosing an audio synthesizer is whether there are software tools available for creating the instruments that the audio synthesizer will playback. Again, you can have the most elegant synthesizer design possible but if the samples and instruments you are playing back are of poor quality, the entire solution will sound bad. A good software tool to create these samples and instruments is paramount to ensuring this quality. Similarly, once the samples and instruments are created, the musician or composer must have a software application that allows them to access those instruments while writing the music that uses them. The software components of a good set of content creation tools includes the following.

Audio Recording Hardware:
Excellent quality microphones, analog to digital converters, speakers and amplifiers are critical to getting a high quality recording at the beginning of the content creation process. The same holds true for the listening room the audio engineer is working in. Good quality recording should be done in a professional studio with access to this equipment.

Audio Processing Software:
This software allows the sound designer to modify the original recordings to reduce noise, remove artifacts, equalize, volume adjust, etc. This is important to ensure that all the final waveforms chosen for an instrument are clean and well balanced well played together.

Instrument Design:
After a collection of instrument recordings or samples have been chosen, the sound designer must then map those recordings so they span the range of a full musical keyboard. After that, envelopes to control the way a sound starts, holds and decays over time must be added. Filters must be added

to control how bright or dull the recordings sound as they are played. Low Frequency Oscillators (LFOs) must be added to simulate vibrato or tremolo. All of this sound design is done in a sound design editor. The editor also saves this information, often referred to as articulation data, so that the audio synthesizer can access it to play it back on the device.

Music Composition:
Finally, once the instrument design is complete, a composer must have access to the instruments AND the audio synthesizer to play the instruments, in a MIDI sequencing application. There are several standard MIDI sequencing applications that composers today tend to use. Some of these include Logic, LIVE, ProTools, Cubase, and others. Most of these applications follow a plug-in protocol that allows audio synthesizers to be "plugged-in" and accessed for composing. These protocols include VST, AU, ProTools and Standalone. Once an audio synthesizer is loaded into a MIDI sequencing program, the composer can select instruments from within that plugin and write their music using those instruments.

Often, however, audio synthesizers coded to run on mobile devices will not run on Macs and PCs and visa versa. It is therefore important for an application designer to know that the company providing the audio synthesizer for a mobile application has solved this problem and created a desktop version of the synthesizer for content developers.

Summary

As mentioned at the beginning of this article, music applications are consistently in the top five revenue producing applications on mobile devices. And games that use high quality audio almost always do better than games that do not. Therefore, great care should be given towards choosing an audio synthesizer that can deliver high quality and engaging sounds and music. In choosing a solution, attention must be given not just to the features of the mobile synthesizer itself, but also how well the wavetable instruments have been designed and whether the solution has content authoring tools which will allow a sound designer and composer to easily make the sounds and music for the application itself.

BIOS

Jennifer Hruska is the founder and president of SONiVOX (www.sonivoxmi.com). She began her career in 1984 where she served as Senior Sound Designer for Kurzweil Music Systems co-developing the company's award winning K2000, K2500, and K2600 music synthesizers. In 1993 Ms. Hruska founded SONiVOX and spearheaded the company's line of mobile audio products as well as a retail line of professional virtual instruments sold internationally. Ms. Hruska speaks regularly at industry conventions and events, participates in leading standards organizations and was recently featured in the book "Art of Digital Music" as one of the industry's top 56 visionaries.

Dave Sparks was a senior software architect with SONiVOX from 2005-2009 and currently heads the Android audio division for Google. Previous to this he was head of the research group for EMU and Creative Labs audio division. He has chaired the MMA committee that drafted the DLS standard, authored the DLS-2 standard and has served as a liaison to the MPEG Structured Audio committee during its adoption into the MPEG-4 standard.