Flexible Sound Effects

Lonce Wyse

Institute of Systems Science, National University of Singapore
lwyse@iss.nus.sg, http://www.iss.nus.sg/People/lwyse/lwyse.html

Abstract

Despite exponential growth in the power of consumer computers and despite the dollars generated by the "interactive" multimedia industry (particularly in the form of games), applications using more than trivially interactive sounds are almost non-existent. To address this state of affairs, we have developed a) a collection of controllable sound effects, b) a graphical interface for selecting sounds and controlling parameters in real-time, c) an API for using the sounds in other applications, and d) a manager for timing and queuing simultaneous embedded sound processes.

Our sound effects models are based on *wavetable event patterns* with table lengths and event rates that range between sample-file playback on one extreme and granular synthesis on the other. Most of the interactivity is at the level of controlling algorithms for event generation. The demonstration will focus on the range of sound classes for which this representation is effective.

The noise instruments of Luigi Russolo were among the first of a long line of ingenious contraptions used to create sound effects to accompany theatrical and film performances through the first half of this century. Through the careful attention and skills of the instrumentalist, sounds were intimately tied to the actions on the stage or screen, and were naturally "realtime interactive".

The current state-of-the-art of sound in multimedia typically involves the playback of prerecorded sounds, and interactivity is limited to initiation and volume. Furthermore, we are subject to waterfalls made of three second loops, and crashes that sound exactly the same at each occurrence. Our goal is to provide developers (non experts in synthesis) with tools to build more realistic audio experiences into any application on current popular computer platforms.

While CPU speed and lack of operating system support are obvious reasons why realtime general purpose synthesis is not a part of most applications, there is an equally fundamental non-technical reason for its absence, and that is the lack of tools to bridge the gap between sound synthesis and people who "don't know much about sound, but know what they like." The usability issues of sound control strategies and graphical and code interfaces for embedding flexible sounds in applications is addressed elsewhere (Wyse and Kellock, 1997). Here we focus on sound modeling techniques.

Our approach has been to construct sounds via the algorithmic generation of events. The atomic "events" are wavetables, so the CPU is burdened with sound construction primarily at the "event rate". This is significant, because our system is designed to be embedded in applications that are typically CPU intensive themselves, such as virtual environments and games. We have been able to construct a wide class of flexible sounds models within the event-generation paradigm.

The justification for this approach to sound modeling goes beyond computational costs. There are no sounds that fall outside the domain of sound effects. Some effects are not meant to be imitations of real-world sounds (e.g. cartoon whooshes, bonks, etc.), but many must not reveal a synthetic origin if they are to be effective. Decades of research have been devoted to developing digital synthesis techniques that produce "realistic" musical instrument sounds. While steady progress is being made with specific techniques for specific sound sources or types, there is nothing that approaches the generality and realism of using actual recordings in wavetable synthesis (*Figure 1*).

Recorded sounds are not models, and they provide little in the way of flexibility, but when constructing sounds out of patterns of many events, we have access to the parameters that control the event patterns. In general, the shorter the "atomic" events are, the more control we have over the constructed sound until we reach the far end of this scale where we have the complete generality of granular synthesis. Before we reach that extreme, however, we can get interesting handles on sound construction, and provide a degree of interactivity far beyond current practice for a wide range of sounds.

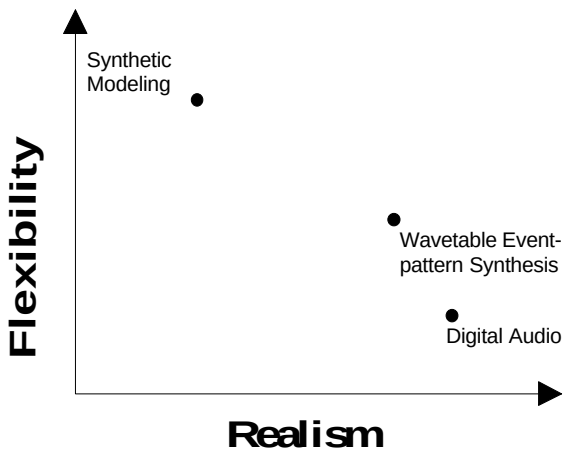

Figure 1: Recordings make up in realism what they lack in flexibility. Purely synthetic techniques offer realism in only isolated instances, and generally do so at great computational expense.

Sound models are each crafted individually, and each sound has idiomatic high-level control parameters. For example, crowd sounds have a parameter specifying the number of people, footsteps have a rate, and collisions have a force. Sounds are classed in an object hierarchy, but there is usually some code that must be written specifically for each sound model. We have an ever growing box of tools which is making development time grow shorter.

Event-oriented sounds such as applause, machine guns or bouncing objects are most obviously suited to this kind of wavetable event pattern synthesis. Many of the sounds use sets of wavetables that are similar. Applause, for example, necessitates the use of roughly 25 separate hand claps to avoid the "electronic" sound of perceivable exact repetitions. In some contexts, microvariations in pitch and timbre can also hide the fact that the same wavetable is being used repeatedly.

Another class of sound types we have had success modeling with these techniques are textured sounds. In this category are waterfalls, wind, machinery, engines, room tone, crowds, and traffic sounds. In most multimedia applications today, generating statistically stable sounds is done by creating loop points in a short recording. Unfortunately, the ear is very sensitive to the presence of such loops, the detection of which quickly destroys any sense of realism. Statistical event generating algorithms can be effectively used in most situations where loops are currently a ubiquitous technique.

Sounds that are perceived as single events (e.g. a gun shot or a pencil breaking) can also be modeled using synchronous or overlapping multiple wavetable events [1,2], with flexibility being the prime advantage over simple audio playback, (memory usage could even be greater than a single event recording). Sounds that are more difficult to model are those that do not have enough texture variation to permit undetected event changes (e.g. the whirr of a drill). Sounds that are impractical to model this way are those that have complex characteristic temporal sequences across many attributes (e.g. speech).

Game developers, use the GUI to choose a sound (type) from the database, and then alter the sound until it is what they are looking for. The parameterized sound model can then be saved, and loaded into the game environment where application object behaviors can be hooked to sound parameters. For example, the speed with which the user is moving through space might control the rate (and style) of footsteps, and different surfaces encountered (eg. carpet and stone), could control the character of the sound. Like in the good old days of sound effects artists, sound makers can once again "watch" behavior and respond appropriately.

Constructing sound models as flexible wavetable event patterns is perhaps more of a craft than a science, but interesting still, as every sound presents different challenges. Although it is a shortage of CPU cycles that provided the original motivation, the expressive power of the method suggests that it will be useful even when purely synthetic techniques run fast enough to be part of an embedded real time sound engine. The "atomic" elements of the models are generally easier to synthesize and then combine than the composite sound is to directly synthesize, and they don't need to be continuously recomputed. The "endless variation" that breathes life into these sounds comes very cheaply with computations only at the event rate.

References

[1] Horner, A, J. Beauchamp, and L. Haken. (1993). "Methods for multiple wavetable synthesis of musical instrument tones." *Journal of the Audio Engineering Society* 41(5):336-356.

[2] Roads, C. (1985). "The realization of nscor," in *Composers and the Computer*, edited by C. Roads (A-R Editions, Madison).