

The Acustick: Game Command Extraction from Audio Input Stream

Giordano Cabral¹

Roberto Cássio Júnior²

DEINFO / UFRPE

MusiGames Studio

Abstract

Recent years have witnessed the boom of musical games. These games associate the commands of the player with musical or sonorous events. While the creation and edition of these associations remain a key factor for the musical game industry, digital signal processing techniques continue evolving, providing very useful information about songs. Even though these techniques cannot provide (nowadays) a perfect transcription of songs, they can be successfully mapped into game commands if a proper strategy is applied. This paper discusses some strategies to answer questions like "how can the information automatically retrieved from an audio source be used in a game?". More specifically, "how can this information be used to determine which commands should a player hit on specific moments of a game?".

Keywords: audio processing, music games, alternative inputs

Authors' contact:

¹giordano@deinfo.ufrpe.br

²roberto@musigames.com

1. Introduction

Audio has always been a crucial element on games. Environment, emotions, feedback, and communication are some of the features that a good soundtrack or sound effect set may provide. Nevertheless, the song traditionally only reflects or supports the game. In recent years, however, a new type of game has appeared, going on the opposite direction. Titles such as Guitar Hero, Guitar Freaks, and Rock Band are examples of games where music plays the key role in the gameplay, and they are probably just the first from a lot of musical games to come.

On the other hand, digital signal processing techniques can be used to understand and extract information from sound files, hopefully transcribing their major musical components. These components may appear in different levels: at the highest, most abstract level,

there are elements such as tempo, tonality, and timbre; at a mid-level, traditional musical elements, such as notes, chords, and percussive sounds; and at the lowest level, information such as attacks, loudness, and spectrum. The occurrence (or abrupt changes) of these components can be automatically tracked (depending on the case, in real-time), using standard techniques. In most cases, a sliding window is applied in order to distinctly analyse each part of the signal (a frame). For each frame, some audio features are observed. For example, one can use spectral features for note, chord, or tonality estimation, and amplitude features for detecting attacks or tempo. As these are computed for each frame, a history can be stored and tracked to perform automatic music transcription.

The actual resulting events can be subsequently converted into commands in a game. The goal of this paper is to discuss how to automatically generate levels of a video game solely based on an audio source (e.g. mp3 file) or an audio input stream (e.g. microphone). This implies some kind of mapping strategy between the given events and the game commands. The paper explains in more detail the issues involved in this process.

Finally, it presents some (non exhaustive) strategies we conceived to perform the mentioned mapping.

The paper also shows some examples that illustrate where each strategy is best suited, bringing some light into their advantages and drawbacks.

2. Transforming events into commands

As stated before, our objective is to extract a set of playable commands from a song. To accomplish this goal, the gameplay should be well defined based on the features we can automatically compute from music. We may consider musical notes, percussive sounds, the level of bass/treble sounds, samples which look similar to something, and even a higher level information inferred from the basic info, like tonality, speed (beats per minute), rhythm, etc.

Not only the extractable acoustic properties should be considered, but also the commands that the player will be able to execute - which varies with the game and the interface used (ex.: keyboard, joystick, mouse, microphone, wiimote). One or more events extracted from the sound can be mapped onto commands. In some cases, we can work in a 1:1 relation - for example, a guitar hero-like game could extract each note from a guitar solo soundtrack and each note detected would be mapped as a note command on the game. But this isn't the only possible way: just consider that we could detect the BPM (beats per minute) from the song and assign commands to the player, expecting him to correctly follow the sound beats. Higher level information, like tonality or rhythm may not be directly mapped into commands, but can be used to adjust other properties like difficulty or to bring a different atmosphere to the game.

3. State of the Art

Current technology is able to transcribe many musical elements, like notes, chords, tempo, tonality, timbre, with more or less accuracy according to the technique, examples, and audio features used. The most basic audio features include frequency and loudness, but many others can be computed, mainly those analysing the behaviour of the spectrum over time.

The spectrum is often regarded via some Fourier Transformation [1], but other methods can be used. The spectrum can also be reprocessed to compute intensities of frequency bands, like bark bands [2], mel bands, and pitchgrams. Unpitched information can also be retrieved using appropriate features, such as the zero-crossing rate, spectral kurtosis, and spectral centroid.

Then, the features to be used can be quite complex. Some will have a respective musical concept, such as percussivity, others may not be associated with any perceivable acoustic elements (we will therefore call them abstract features). Here, we show features and transcriptions (frequency, loudness, pitch, tempo) which can be mapped to game commands, providing a fun gameplay. However, we remark that more complex solutions will probably lead to more precise solutions.

3.1 Loudness

Loudness is probably the simplest and most intuitive audio property. It's a measure of how intense we perceive the sound, highly correlated to the amount of

energy contained in the audio signal (which physically result on more the level of air pressure in the ears). Loudness can be estimated based on the signal amplitude behaviour (ex.: take the maximum at certain interval and you'll may have a good representant for the expected intensity), or summing the audio intensity of its multiple frequency components (the frequency spectrum got from the FFT).

3.2 Frequency

Frequency - as the name suggests - is the measure of how fast a repetitive oscillation occurs (in our case, in an audio signal). Audio can be a random signal with no repeating patterns at all, but this is usually not true for most cases - specially music. Multiple simultaneous frequencies composes an audio track, and each one of them appears with certain intensity. This distribution of frequency/intensities is called Frequency Spectrum, and is most usually computed using an widely-known algorithm called Fast Fourier Transform (FFT).

3.3 Pitch

Pitch is related to the frequencies of a sound. Just as said, multiple repeating patterns at various frequencies composes the sound. Pitch comes from the perception of those multiple patterns together as matching some pre-defined musical note - slightly different audio frequencies or the multiple frequencies that compose an audio signal may be perceived as the same pitch. Transcribing the melody of a song (i.e., extracting the musical notes) requires detecting the audio pitch in the signal along the time.

3.4 Onset detection

The need to detect features as pitch on some kind of audio source comes together with the need of detecting when this feature has started to appear. The exact instant where an event like a note starts to be played or a hit on the drums occur is called onset.

Onset detection algorithms can use various methods to get the onset times. Frequency spectrum, amplitude, or signal phase evolution on time are some of the most used features as input to the onset detection algorithms, which work analysing at the variation (derivative) of those measures. Usually, before detecting one of those features, some preprocessing/filtering is done in order to give the signal better quality for the desired purpose. Afterwards, an evaluation function will compute the chosen input functions used in the algorithm (amplitude, spectrum, etc.), and its results will be

compared on the timeline to determine what looks like an onset and what does not. [3]

3.5 Beat tracking

Music has usually well-defined time divisions. Those divisions are aligned with the stronger attacks on the musical instruments (percussive or not). They can be also divided in more subdivisions, and the main characteristic present on each of these time divisions is that there is rhythmic coherence if you play a "strong" musical note or percussive sound in one of those times. The most intuitive way to understand this is observing how people tap their feet while following a song.

Beat tracking tries to extract this pulse from the original audio, i.e. to find a function which indicates exactly what would an rhythm-aligned foot be doing while hearing the sound. In other words, it marks the time of all the "strong beats", in the music, which can be used to track the sound speed (beats per minute), making an accompaniment, for instance.

Beat tracking algorithms commonly rely on onset detection (although this is not necessarily the only input). Knowing the times of entrant sounds / spectral changes in a music piece is the most basic information that you can use to extract the beat. AI techniques are also widely used to find the best matching in the beat for the whole song (you may have lots of possibilities to match the beat, given an onset timestamp set). Also, some previous known information about the kind of music being played may be useful.

3.6 Pitch tracking, spectral flux

Generic music properties are not the only useful information to get from audio. It is possible to look for more specific elements: imagine a game which requires the player to trigger some action when he listens a whistle. Or a game which is played using a microphone, where the player must imitate some specific sounds (ex.: beatbox/drum sounds). On those situations, proper musical information is not really needed, but the game should be able to find pieces in the audio stream being played which simply look similar to a desired pattern. This can be achieved by some statistical techniques, trying to find signals highly-correlated to the searched pattern. This correlation may be calculated considering the pure signal, or some transformed version of it. Results may not be perfect, but there are lots of information which can be good enough for a nice gameplay.

4. Strategies

After the main elements were transcribed, based on the chosen features, a mapping strategy shall take place to actually generate the game commands. Next sections present some of them.

4.1 Direct mapping

Audio events are associated to commands in a 1:1 relation, for each type of audio feature detected. Ex.: pitch changes always trigger command A, beat tracking command B, energy peaks implies command C; or percussive sounds induces one type of command, harmonic changes implies on a second type.

Direct mapping the commands can give a high level of fidelity to the song being played, but this doesn't imply directly in good gameplay. Two problems appear on this strategy: first, depending of the used musical elements, the commands may be simply unsuitable to play, providing a poor or too-hard game. Second, people may not perceive all elements playing in the song. This may lead the player to be unable to see the correlation between the commands and the music.

As a refinement of the above technique, some post-filtering may be applied to the obtained events. In order to achieve better playability, the commands may be filtered, favouring or preventing the appearing of sequences with certain properties.

4.2 Meta information

Using some methods cited on the state of art, it is possible to extract informations about a soundtrack in a wide view. Important characteristics like tonality, beats per minute, the relation between the intensity of percussive/harmonic periods in a song, dissonance, and others can be used to guide the choices for the expected player commands.

Periods characterized by tensions (with dissonances in the harmony) may trigger a higher difficulty, as well as repetitive presence of percussive elements in a short period of time.

Other utility to this information is to enhance the coherence of events found by direct mapping. Consider the set of all energy peaks in a soundtrack; for sure, there are many peaks which are not too representative of the audio, and even peaks which may be mere noise. Now if we have a more general purpose info – like the

beat tracking giving the music tempo – the energy peaks can be aligned according to the song's rhythm and the unaligned ones are just discarded.

Analogously, if when getting events on direct mapping for chord changes, having song's tonality may be useful to determine if the changes are happening in a coherent way.

This analysis can also be useful beyond mere command generation. The game atmosphere or scenario, or even difficulty may be guided by the song, instead of being restricted to the input commands which the player should trigger.

4.3 Pre-defined phrases

This strategy is focused on creating a gameplay with a strong musical feeling. A huge set of possible playing commands is recorded, preferably with volunteer players with good musical sensibility. This set intends to contain the most intuitive, frequent and enjoyable sequences to be executed on the game. Notice that those sequences will be usually done having the expected interface in mind (like a wiimote, a microphone, or a simple joystick).

The basic point is to privilege the “nice-to-play” command sequences. Those sequences should make the player feel integrated to the song when executing them, and they must be generic enough to be compatible with many songs. Once we have this sequence database and the audio features, the initial event list is generated. Then an algorithm is run to adjust the sequences on the database to “fit” into real song events. On this process, the recorded phrases may be expanded or reduced if necessary (but always maintaining musical coherence).

4.4 Pattern Extraction

Machine learning algorithms can be applied to the audio data, extracting the most recurrent patterns in a sequence. In comparison to the anterior technique, this attenuates the need to construct big datasets, as they'd be provided by the song itself.

Well calibrated, this way can allow better fidelity for the commands regarding the soundtrack, while it gives the to the user better facility to associate the commands with the playing tune, due to exploring of the patterns found.

5. Conclusions

This paper has described the overall method used to map sound analysis into playable commands the games developed by us. The method has three parts: 1) the discovery of the most significant features to analyse sounds; 2) the transcription of musical elements, based on that features; and 3) the strategies that can be used to convert a sequence of events into playable game commands.

As a graphical representation of playability is hard to achieve, we preferred to let video examples of actual conversions of music into playable games available in a web link: <http://www.musigames.com/>.

Our future work is quite open. We will continue exploring fun ways to convert audio into games, but we are willing to collaborate with other developers to enlarge our experience with more games.

We hope that this paper will be useful for the game development community.

References

- KLAPURI ANSSI / DAVY, MANUEL, SIGNAL PROCESSING METHODS FOR MUSIC TRANSCRIPTION
- GERARD ROMA TREPAT, FREESOUND RADIO: SUPPORTING COLLECTIVE ORGANIZATION OF SOUNDS, MASTER THESIS PP. 14
- SIMON DIXON, ONSET DETECTION REVISITED, PROC. OF THE 9TH INT. CONFERENCE ON DIGITAL AUDIO EFFECTS (DAFx'06), MONTREAL, CANADA, SEPTEMBER 18-20, 2006