Freesound Explorer: Make Music While Discovering Freesound!

Frederic Font
Music Technology Group
Universitat Pompeu Fabra
frederic.font@upf.edu

Giuseppe Bandiera
Music Technology Group
Universitat Pompeu Fabra
giuseppe.bandiera@live.com

ABSTRACT

Freesound Explorer is a visual interface for exploring Freesound content in a two-dimensional space and creating music by linking content in that space. Freesound Explorer is implemented as a web application which takes advantage of modern web technologies including the Web Audio API and the Web MIDI API. This extended abstract describes Freesound Explorer's features and provides some technical details about its implementation.

1. SYSTEM DESCRIPTION

Freesound Explorer[1] is a web application for exploring Freesound[2] content and making music at the same time. The core idea behind the Freesound Explorer is the projection of sounds as points in a two-dimensional space (or map). Sounds in that space are organised by similarity, i.e., similar-sounding sounds are closer in the map than non-similar-sounding ones. This idea was originally introduced in the CataRT system by Schwarz et al. [5] and has already been explored in a number of projects such as the SoundTorch [2], the SongExplorer [3], and others, including previous works by the authors [1]. Similar ideas have also been recently explored in projects such as the Infinite Drum Machine[3] and Bird Sounds[4].

In Freesound Explorer, users start by entering some query terms which are then used to retrieve sounds from Freesound and project them as points in a map. As in previous work by the authors [1], points are given a colour which summarises their timbral properties as per the *tristimulus*[5] audio descriptor (see Fig. 1). Sounds can be played on mouse over. This allows users to quickly have an idea of what kinds of sounds are projected at each part of the map. Further queries can be performed by entering new query terms, and the results are also projected in the map, grouping the results of each query in different *Spaces*. In this way, the results of several queries can be simultaneously represented in a single map (see Fig. 2).

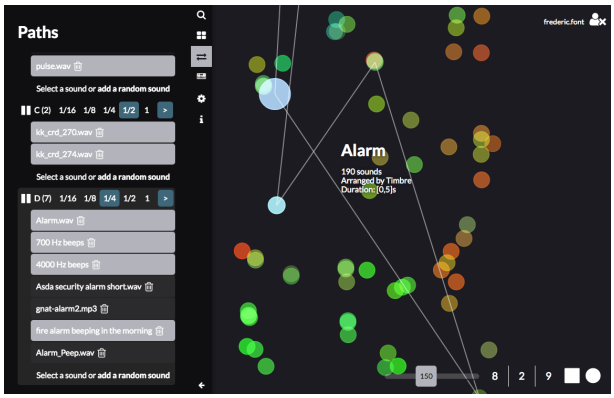

Figure 1: Screenshot of the Paths creation interface

Music making in the Freesound Explorer is done by the creation of what we call *Paths* (see Fig. 1). Paths consist of lists of sounds (chosen from the sounds in the map) which are played in sequence and (optionally) synchronised to a global metronome. This idea is inspired by the graph grammar for music creation described by Roma and Herrera [4]. Several paths can be played simultaneously, allowing users to create multiple sound streams (e.g. bass line, drum sequence, sound fx) which together conform what we call a *Session*.

Besides paths, Freesound Explorer also allows to create mappings between sounds in the map and specific MIDI note events. The mapping is created by a standard MIDI learn mechanism in which a sound is set to *learn* mode and linked with a MIDI key once pressed. Playback speed of the sounds triggered by MIDI events is modified according to the distance to the closest learnt note. In this way users can easily play melodies and harmonies with *ready made instruments* based on arbitrary Freesound samples.

Sessions, along with MIDI mappings and the content of the Spaces, can be saved in a remote server and later reloaded (see Sec. 2 for more details). Furthermore, audio output can be recorded and downloaded as 16bit PCM WAV file.

[1] http://labs.freesound.org/fse/. The code is released under a MIT open source license and can be found at https://github.com/ffont/freesound-explorer.
[2] http://freesound.org
[3] https://aiexperiments.withgoogle.com/drum-machine
[4] https://aiexperiments.withgoogle.com/bird-sounds
[5] https://en.wikipedia.org/wiki/Timbre#Tristimulus_timbre_model

Web Audio Conference WAC-2017, August 21–23, 2017, London, UK.

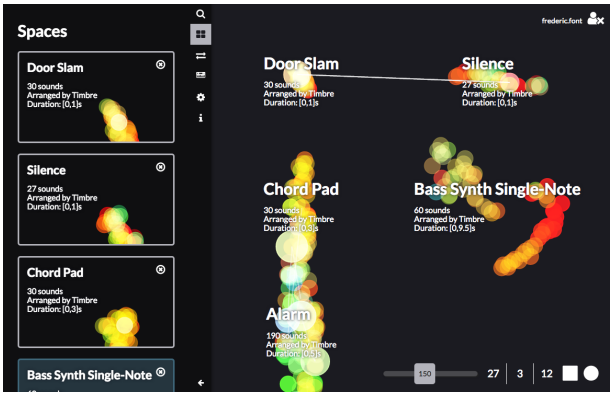

Figure 2: Screenshot of the list of Spaces and their arrangement on map

2. IMPLEMENTATION DETAILS

Freesound Explorer is implemented as a single page application built around two communicating modules:

- A front-end core written in Javascript and React[6] which does all UI and audio operations.
- A back-end server written in Python and powered by Flask[7], connected to a SQLite[8] database which stores sessions.

We handle audio playback and synchronisation by calling the Web Audio API[9]. The Web Audio API clock is used to provide accurate timing of the metronome following the popular method described by Chris Wilson[10]. Metronome *ticks* are dispatched as events that contain the AudioContext time of the next tick. These events can be listened from any component of the application to provide synchronisation (i.e. in the Paths component, to decide whether any sounds should be played in text tick and schedule them). Support for handling MIDI events is implemented using the Web MIDI API[11], with functions included for reloading MIDI devices and filtering by channel and device.

The code related to the UI has been written using the library React, providing high code reusability and scalability. To manage the state of the application, we decided to use the popular library Redux[12], that provides an organised way of accessing and modifying the state of application, stored as a plain Javascript object. The novelty in our approach stands in the usage of Redux not only for portions of the state related the UI, but most importantly for keeping a centralised store for audio and MIDI events of the current performance. This way, the application always has a strictly defined and predictable way to retrieve and manipulate past, present and future elements of the current track. Using the library Redux by this approach also facilitates the implementation of saving and loading functionalities.

In search mode, when user enters query terms, the system queries Freesound using the Freesound API[13], retrieving a number of sound results which are added to a Space. The computation of the position of the sounds in the Space is done using a Javascript implementation of the t-SNE dimensionality reduction method[14] and based on *MFCC*[15] audio features (or, alternatively, *HPCP*[16] for tonality-based arrangements). Therefore map computation is completely carried out in the client. The position of each sound in the space is stored in the Redux store. For each sound, we also store extra metadata such as the author, title or license (to provide attribution). When the user clicks on a sound, the application fetches the audio for it, decodes it and creates a new element on the Web Audio API pipeline. The decoded buffer is saved on the Redux store to avoid having to refetch the audio every time the user wants to play the sound.

The backend part of Freesound Explorer is only in charge of handling user accounts and storing user sessions data. Authentication is done using the Freesound API, therefore users can login using their Freesound credentials and do not need to create a specific account for Freesound Explorer. Sessions data is stored in JSON files which are linked to user accounts. This comes naturally as our Redux store in the front-end can dump all the data as a Javascript object which can be easily sent to the server and directly saved into a file. Because storing sessions in the remote server is not an essential feature, Freesound Explorer can also run *server-less*. In that case, sessions can still be stored and loaded using local storage of the browser, but can not be synced through the server with other instances of Freesound Explorer.

Technical requirements for the demo at WAC 2017: We will need two speakers and a table to put a laptop plus a small MIDI controller. Connection to the speakers will be through mini-jack. A power plug will also be needed for the laptop and for the speakers. The laptop and the MIDI controller will be provided by the authors.

3. REFERENCES

[1] F. Font. Design and evaluation of a visualization interface for querying large unstructured sound databases. *Master's thesis, Universitat Pompeu Fabra, Music Technology Group*, 2010.

[2] S. Heise, M. Hlatky, and J. Loviscach. Soundtorch: Quick browsing in large audio collections. In *Audio Engineering Society Convention 125*, 2008.

[3] C. F. Julià and S. Jorda. Songexplorer: A tabletop application for exploring large collections of songs. In *International Music Information Retrieval Conference (ISMIR)*, pages 675–680, 2009.

[4] G. Roma and P. Herrera. Graph grammar representation for collaborative sample-based music creation. In *Audio Mostly Conference*, 2010.

[5] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton. Real-time corpus-based concatenative synthesis with catart. In *International Conference on Digital Audio Effects (DAFx)*, pages 279–282, 2006.

[6] https://facebook.github.io/react/
[7] http://flask.pocoo.org
[8] https://www.sqlite.org
[9] https://www.w3.org/TR/webaudio/
[10] https://www.html5rocks.com/en/tutorials/audio/scheduling/
[11] https://www.w3.org/TR/webmidi/
[12] https://redux.js.org/
[13] https://freesound.org/docs/api/
[14] https://github.com/karpathy/tsnejs
[15] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
[16] https://en.wikipedia.org/wiki/Harmonic_pitch_class_profiles