AMEN, AMEN SEVER, AMEN.JS: TOOLS FOR ALGORITHMIC REMIXING

Thor Kell
Tide Pool Research Corps
thor@tide-pool.ca

Brian McFee
Center for Data Science +
Music and Audio Research Lab, NYU
brian.mcfee@nyu.edu

Ben Lacker
Sholto Labs
benlacker@gmail.com

ABSTRACT

We present a set of tools for algorithmic remixing: Amen, a Python analysis and remixing tool built on the librosa analyzer; Amen Server, a Python web server for reading and analyzing audio files,and server the analysis data via HTTP; and Amen.js, a JavaScript remixing tool for web applications that makes use of the Web Audio API. These tools are heavily influenced by the Echo Next Remix and its associated APIs, and are designed to replace this now defunct set of tools. In addition to discussing this motivation, we delineate the implementation details of these three tools, and discuss the difference in both implementation and features between Echo Nest Remix and Amen.

1. INTRODUCTION

Amen is an open source replacement for the Echo Nest Remix API [1]. That is to say, it allows for algorithmic manipulation of audio based on musical analysis. The related tools, Amen Server and Amen.js, provide HTTP access to Amen and the analysis that it generates, and a JavaScript version of the algorithmic manipulation tools. Amen is built on top of the librosa [2] analysis library.

The Echo Nest Remix API was the foremost algorithmic remixing tool in the world during its lifespan. It became unsupported in November of 2014, and the underlying Echo Nest / Spotify API analysis endpoint was removed in March of 2016. Amen aims to replace both the analysis API endpoint and the Remix API with open source, non-proprietary, locally-runnable tools.

2. PREVIOUS WORK

Tristan Jehan's Ph.D thesis [3] approaches the topic in great detail, and was the driving force behind the Echo Nest Remix API [1], which dated from 2008. William Sethares has also discussed beat-based signal processing, as well as "musical recomposition" in his book Rhythm & Transforms [4]. It is an unverifiable music technology legend that Sethares had a MATLAB implementation of this sort of algorithmic remixing as early as 1995.

3. IMPLEMENTATION DETAILS

3.1 Amen

Amen is built on top of the librosa analysis library, and builds its primary object, the Audio object, from librosa analysis. The Audio objects handles the combination of analysis features and musical timespans. For each feature, the data from librosa is loaded into a Pandas [5] dataframe. These dataframes allows each feature to be sampled at arbitrary times and durations, with a given aggregation function. (The fundamental unit of time is the librosa frame, which Amen leaves as the default of 512 samples). For example, the spectral centroid feature could be sampled per-beat, with an aggregation function that takes the mean of each frame within each beat. Or, the tempo for the entire track could be generated by taking the median tempo across every frame in the track. Amen currently provides beats, Echo Nest-style segments, and a timespan for the entire track. Bars and downbeat tracking are currently on the backlog.

3.2 Amen Server

Amen Server is a simple web application that allows end users to post an audio file to the server for analysis, and get a JSON file representing the analysis back. The JSON returned is in the same format as Echo Nest analysis JSON (many thanks to Colin Fahy for writing the conversion function). The server runs on the Tornado networking library, and provides a single endpoint that reads audio from the body of a POST request. The audio is read into a temporary file, and the server puts the path to the file on a queue. The queue (implemented with RQ and Redis) analyses the audio, creates an Audio object, converts it to JSON, and writes the JSON to a file. It then uploads the analysis file and the audio file to a cloud-based storage service (Amazon S3 is the current default). The entire server is packaged using Docker, to make installation and deployment as simple as possible.

The server is not, at time of writing, a production-ready API. It lacks user verification, error recovery, and easy horizontal scaling. These issues are on the backlog, and will

hopefully be completed soon.

3.3 Amen.js

Amen.js uses the JSON data returned from the Amen Server to manipulate the analyzed audio file via the Web Audio API. It is heavily based on The Echo Nest's remix.js and jremix.js. While it requires more setup than the Python version (loading audio to a client-side web browser can be surprisingly complicated), it is better for applications that require user interaction beyond the command line.

4. DIFFERENCES FROM ECHO NEST REMIX

The primary difference between Echo Nest Remix and Amen is the timespans and features that are available in each API.

Echo Nest Remix	Amen
-	Track
Sections	-
Bars	-
Beats	Beats
Tatums	-
Segments	Segments

Table 1. Differences in timespans between Echo Nest Remix and Amen.

Echo Nest Remix	Amen
Chroma	Chroma
Timbre (PCA)	Timbre (MFCC)
Loudness, Max Loudness	Amplitude
-	Spectral Centroid
Key, Mode	-
Time Signature	-

Table 2. Differences in features between Echo Nest Remix and Amen.

In addition to these feature differences, the Echo Nest analysis API provided feature-rich data at the segment level, and left it up to the user to build up features for bars or beats. By using pandas, Amen can instead dynamically build feature-rich data at any temporal level, using the aggregation function of the user's choice.

The process of generating audio output is also different. Echo Nest Remix is based on a concatenative synthesis model, with additive synthesis between two slices of time being done with helper functions before concatenation and output. In contrast, Amen makes use of an additive synthesis model. Slices of time can be placed at any point in time, and are summed with any audio that is already there. This requires a more complicated API, but is more flexible.

The Echo Nest Remix API made use of segments, which started and ended at zero-crossings, allowing them to be concatenated with no sonic artifacts. Amen does not require its timespans to be bound to zero crossings. Rather, Amen's synthesis function expands each timespan to the nearest zero crossing, and then does additive synthesis from there. This can lead to degenerate cases (e.g. audio with a DC offset), but in general this approach works well.

5. CONCLUSION & FUTURE WORK

Amen provides open-source, locally-runnable analysis, remixing, and synthesis tools that have been missing on the wider Internet since the Spotify / Echo Nest analysis endpoint was removed in March 2016. The server makes it possible for anyone to provide their own analysis endpoint, for their own project or for wider usage. Finally, the JavaScript version matches the functionality of remix.js, and enables all sorts of interactive web applications.

Future work is legion, but includes adding more audio features to Amen, linking to machine-learning models for downbeat and time signature detection, making the server more robust, modernizing the JavaScript, and so on. We'd love your help: we're at https://github.com/algorithmic-music-exploration.

6. THANKS

The Amen core team is Thor Kell, Brian McFee, and Ben Lacker. Special thanks to Colin Fahy for the shim to Echo Nest analysis format, Mike Kilmer for documentation fixes, CJ Carr for discussion about segments, and to Andrew Nesbit & Peter Sobot for inspiration.

7. REFERENCES

[1] Ochshorn, Rob, Ben Lacker, Jason Sundram, Thor Kell, Peter Sobot, Tristan Jehan, and Brian Whitman "Echo Nest Remix: The Internet Synthesizer", *http://echonest.github.io/remix*, retrieved August 2017

[2] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and Music Signal Analysis in Python", *Proceedings of the 14th Python in Science Conference*, pp. 18-25. 2015.

[3] Jehan, Tristan. "Creating Music By Listening", *Diss. Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences*, 2005.

[4] Sethares, William Arthur. "Rhythm and Transforms". *Springer Science & Business Media*, 2007.

[5] McKinney, Wes "pandas: a Foundational Python Library for Data Analysis and Statistics", *Python for High Performance and Scientific Computing*, 2011