Synthalize: A Crowdsourced Synthesizer

Ahren Alexander
Northwestern University
EECS Department
ahren@u.northwestern.edu

Noah Conley
Northwestern University
EECS Department
noahkconley@gmail.com

Thomas Huang
Northwestern University
EECS Department
thomas.yt.huang@gmail.com

ABSTRACT

The synthesizer is an electronic, and now virtual, musical instrument that can imitate other instruments or generate many new timbres of its own. Since the late 20th century, the synthesizer has grown in popularity and is now one of the most important instruments in the music industry. Modern synthesizers have many adjustable settings that provide for countless combinations of sounds, making them both powerful and versatile. In this work, we describe Synthalize, an interface for rapidly and efficiently choosing a desired synthesizer setting. To use Synthalize, the user searches for and selects a descriptive word that is similar to their desired timbre, and the synthesizer highlights appropriate sounds.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: User-centered design; H.5.5 [**Sound and Music Computing**]: Signal analysis, synthesis, and processing

Keywords

Crowdsourcing, human computation, audio engineering, user interfaces, design.

1. INTRODUCTION

Software synthesizers, used by music professionals and hobbyists alike, are prone to follow the model set forth by the physical synthesizers produced by the likes of Robert Moog and David Buchla. The interface of these analog behemoths required hours of familiarizing oneself before meaningful sound synthesis could be achieved by the user. This was the 60s -- why does software today recapitulate this outdated and dizzyingly complicated conceptual model?

The most salient problem with most synthesizers, analog or digital, is the “semantic gap” between the language used by the machine and the language used by the user. While users are apt to describe sounds as “shrill” or “warm,” synthesizer systems have no adjustable parameters which directly influence such characteristics of a sound. Instead, users are provided complicated LFOs, envelope generators, and various patch modules to meddle with in attempting to make particular sounds.

This “gap” carries over to the factory presets often installed by the manufacturer as sounds like “DramaQueen” and “Break it Down” populate the synthesizer sound bank [Massive, NI]. When a user is looking for a particular type of sound, these nonsensical names make exploring the synth patch space and finding a setting that matches the user’s desired sound very difficult. SynthAssist has looked into this problem and approached it by having users make imitations of the sound they are looking for [1].

In this research we intend to gather several pre-synthesized patches and their acoustic properties, crowdsource descriptions of said sounds, and map these acoustic properties to particular descriptors as determined by the data gathered from those surveyed. For example, if a particular sound is collectively described as “brittle,” anytime users search for “brittle” or its synonyms, that sound patch and those with similar acoustic properties will populate the search results.

By establishing relationships between human vocabulary and synthesizer language, users will be able to more quickly or produce the sounds they are looking for.

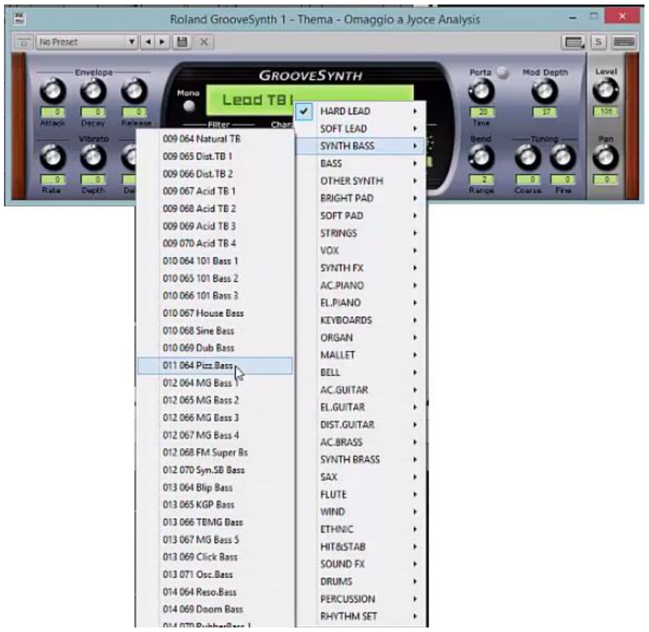


Figure 1: A standard synthesizer interface and presets

2. GOALS

As music producers, the intellectual interest and practical utility of this project is to both save time (practical) and foster creativity (intellectual) by having the workflow proceed smoothly without being interrupted as often.

Thus, our ultimate goal is to cut down on the time it takes users to find a particular synthesizer patch by constructing an interface to bridge the “semantic gap” and make it possible to select synth presets via appropriate descriptor names rather than complicated settings or random labels (e.g., “DramaQueen”). We intend to do this by crowdsourcing data to tag non sequitur synth patch names with more familiar descriptors. We will use this data to design a unique synthesizer interface: users find a particular sound by looking up words which semantically relate to the sound they are looking for. Upon searching for a semantic descriptor, a list of descriptors will appear, and they can be selected to highlight appropriate sounds resembling the descriptor.

3. PROCESS AND IMPLEMENTATION

3.1 Development of Mechanical Turk Survey

1. Generate a simple 12-tone scale and chord for 48 different sound patches to play.
2. Record each sound playing the scale and chord to a 12 second audio file.

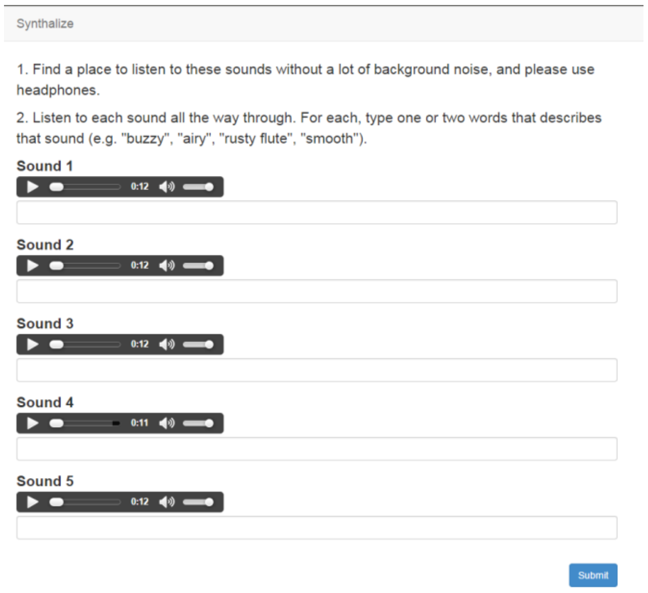
3. Within Mechanical Turk HIT, direct workers to http://synthalize.herokuapp.com/listen
4. Ask workers to describe 4 random sounds from our bank of 48 using one or two words
5. Workers then provided code to receive their compensation

150 workers completed this task resulting in 600 words or phrases used to describe our various sounds. Various others (friends, peers, etc.) also completed the task outside of Mechanical Turk increasing the number of samples in our description database above 600.

3.2 Implementation of Synthesizer and Collected Data

1. Our interface was built with HTML, CSS, and JavaScript and can be found at http://synthalize.herokuapp.com/
2. The 600+ descriptions were incorporated into the synthesizer as "semantic tags" for users to search through in finding the sound their search queries are related to. Using the Big Huge Thesaurus API [3], related semantic tags were also included in the descriptor database to help users find sounds described by words not found in our 600+ sample descriptions. This increased the semantic tag count to 1500.
3. The interface is separated into two sections: tag searching and patch selection marked by blue and red respectively. Users can hear a sample of a particular patch by clicking on its name in the patch selection area. By searching for descriptors such as "dark" or "warm," patch names change size in relation to their relevance to the search query.

Additional tools and resources used to construct Synthalize included Massive by Native Instruments, Node.js, AngularJS, and Heroku


Synthalize

1. Find a place to listen to these sounds without a lot of background noise, and please use headphones.

2. Listen to each sound all the way through. For each, type one or two words that describes that sound (e.g. "buzzy", "airy", "rusty flute", "smooth").

Sound 1 0:12

Sound 2 0:12

Sound 3 0:12

Sound 4 0:11

Sound 5 0:12

Submit

Figure 2: Mechanical Turk survey

4. TESTING AND RESULTS

Test subjects were asked to perform one of two versions of the same task. Subjects were to select two sounds of our 48 that might find in an orchestra and then one sound they might find in a science fiction action film. In the first version, subjects were not allowed to use the tag search functinoaltiy of the synthesizer, simulating the experience of using conventional synthesizer interfaces. In the second version subjects were allowed to use the search functionality. Subjects were timed on how long it took for them to pick sounds and were asked their satisfaction with their choices.

Results proved inconclusive given remaining work to be done in developing the synthesizer interface. During user testing and observation, subjects frequently skipped over using the search function altogether in favor of perusing the synth patch bank and sampling sounds one by one. This was due to user frustration in receiving no results when searching for tags they felt were related to the task as well as the lack of emphasis by the interface of the importance of search functionality. Even after reminding subjects about the semantic tag search function, they would often continue to explore the sound patch bank unassisted. This is due to the bank only possessing only 48 sounds to choose from. Commercially available synthesizers often have hundreds of factory presets users must sift through to find what they want. With only 48 sounds displayed all at once (as opposed to the standard drop down menu), searching through synth patches did not require much effort. By making it too easy for users to sort through all 48 sounds without the search function, the value of the semantic search function was ultimately trivial.

5. CONCLUSIONS AND FUTURE WORK

While this project holds great promise, Synthalize requires further development before the effectiveness of searching for sounds semantically can be determined. We succeeded in building a sound testing platform of our own, and glimmers of promise shown through during user observation when subjects happened to search for tags in our database that were mapped appropriately to their respective sound patch; however, too many search queries went unanswered, lacking a connection to the available synth patches. For a future release, the database of sound descriptors needs to be expanded to account for more encompassing descriptions of sounds. Furthermore, the quality of data gathered (as well as the specificity in what responses are needed) needs to be better. Some of the descriptions provided by Mechanical Turk workers proved useless and detrimental to the appropriate mapping of tags to sounds.

In addition to more robust data gathering, the Synthalize interface requires a facelift to better communicate the purpose of its elements. Provided below is a concept rendering of the next iteration of Synthalize and how it would appear to users.

Ideally, Synthalize would evolve into a platform where low-level synthesis parameters could be controlled by high-level directives made by the user. For example, a "warm/cold" slider could be adjusted to affect the filtering of the sound.

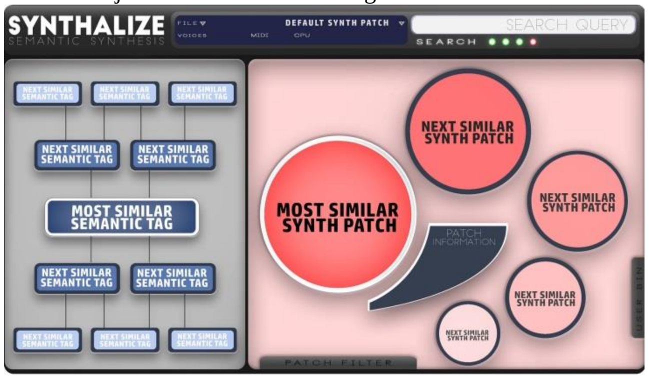


Figure 3: Proposed future interface

6. VIDEO LINK

A link to a video describing Synthalize can be found on its webpage at http://synthalize.herokuapp.com/about

References

[1] Cartwright, M. and Pardo, B. SynthAssist: An Audio Synthesizer Programmed with Vocal Imitation. http://music.cs.northwestern.edu/publications/cartwright_pardo_acmmm14.pdf

[2] Seetharaman, P. and Pardo, B. Reverbalize: A Crowdsourced Reverberation Controller. http://music.cs.northwestern.edu/publications/seetharaman_pardo_td_acmmm14.pdf

[3] Big Huge Thesaurus API. http://words.bighugelabs.com/api.php