Synthalize: Semantic Sound Synthesis

Ahren Alexander
Northwestern University
MaDE 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 musical instrument that can be used to imitate other instruments or generate brand new sounds. Since the late 20th century, the synthesizer has grown in popularity and is now one of the most important tools of the music industry. Synthesizers have countless adjustable settings that allow for limitless exploration of timbres, making synths 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 related 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 often complicated instrument. Their design frequently models the physical synthesizers produced by the likes of Robert Moog and David Buchla in the 1960s. 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 This was the 60s -- why does software today recapitulate these outdated and dizzingly complicated synthesizer interfaces?

The most problematic issue with such interfaces is the "semantic gap" between the language used by the machine and the language used by the user [2]. Synthesizer systems provide no clear means for a user to produce high-level creative intent. Users are provided low-level generating and editing tools such as LFOs, envelope generators, and various patch modules to meddle with in attempting to make particular sounds [Figure 1].

The challenges of designing one's own sounds can be circumvented by using the synthesizer's provided factory presets--sounds that have been pre-synthesized for use. The "semantic gap" carries over to these presets, however. Hundreds of sounds such as "DramaQueen" and "Break it Down" populate the sound bank [4]. When a user is looking for a particular type of sound, these non sequiter names make exploring the synth patch space and finding their target difficult. SynthAssist addressed this problem by asking users to imitate the sound they are looking for [1]. We propose a different solution.

In this research, we intend to gather several pre-synthesized patches and their acoustic properties, crowdsource descriptions of these sounds, and map acoustic properties to the particular descriptors used to describe them. 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. Our interface implementation, testing procedures, results, and suggestions for future work are also outlined in this work.

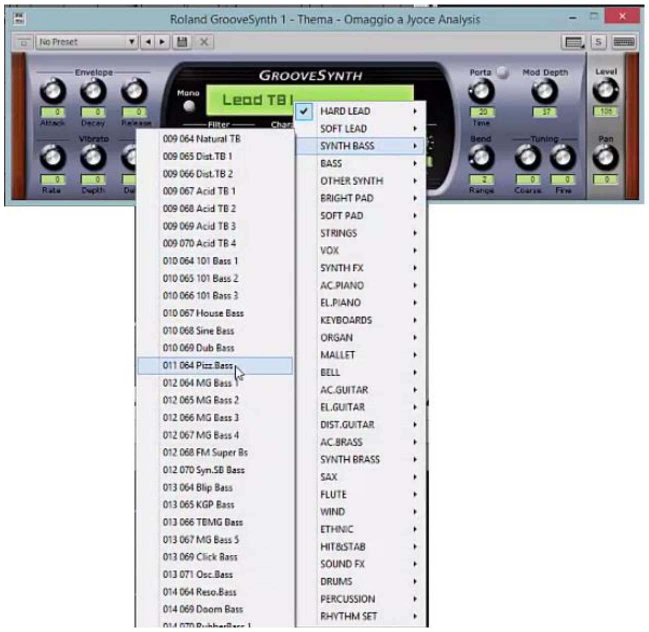


Figure 1: A standard synthesizer interface and presets

2. GOALS

Ultimately, we seek a human-centered synthesizer interface that allows users to specify high-level sonic intent and rapidly find a synthesizer preset which matches that intent. This will be accomplished through establishing relationships between human vocabulary and synthesizer language. Our concrete goals are:

-Develop a novel, human-centered synthesizer interface
-Cut time required to find satisfactory sounds by 50%

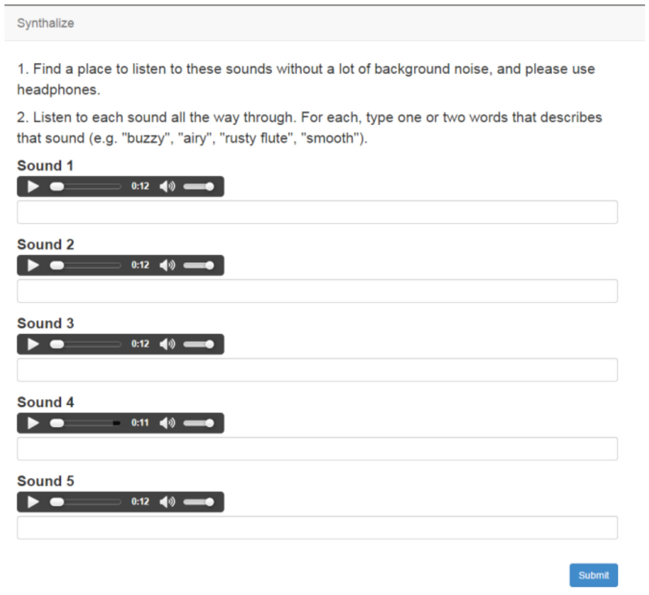
3. PROCESS AND IMPLEMENTATION

3.1 Developing the 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.


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

Sound 2

Sound 3

Sound 4

Sound 5

Figure 2: Mechanical Turk Survey

3.2 Implementing the Synthesizer Interface

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

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

Synthalize requires further development before the effectiveness of searching for sounds semantically can be determined; we believe the project holds great potential. We succeeded in building a platform of our own, and glimmers of promise showed through during user observation. When subjects searched for well-mapped tags in our database, Synthalize was very effective in isolating the types of sounds the user was looking for; however, too many search queries went unanswered, lacking a connection to the available synth patches. The database of semantic tags requires expansion to account for the variance in descriptions of particular sounds. Furthermore, the quality of data gathered needs to be more robust. Some descriptions provided by Mechanical Turk workers proved inhibitive to the appropriate mapping of tags to sounds. To avoid inhibitive responses, what makes an appropriate sound description needs to be more clearly defined. In addition to more robust data gathering, the Synthalize interface requires a facelift to better communicate the purpose of its elements. Provided in Figure 3 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. In order to get there, it will require further development of the user interaction experience, further semantic data collection, and greater exploration of the relationship between human language and sonic phenomena.

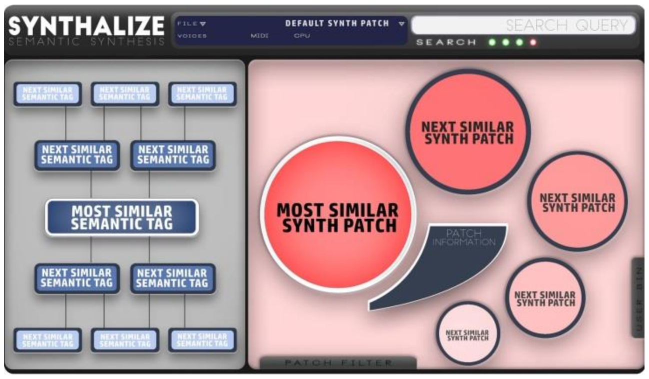


Figure 3: Proposed future interface

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

[4] Massive Synthesizer, Native Instruments