Sifting Sound

Interactive Extraction, Exploration, and Expressive Recombination of Large and Heterogeneous Audio Collections

by

Nikhil Singh
B.M. Berklee College of Music, 2017

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning in partial fulfillment of the requirements for the degree of

Master of Science
at the
Massachusetts Institute of Technology
September 2020

Author . . . . . .
Program in Media Arts and Sciences
August 17th 2020

Certified by . . . . . .
Tod Machover
Muriel R. Cooper Professor of Music and Media
Thesis Supervisor

Accepted by . . . . . .
Tod Machover
Academic Head, Program in Media Arts and Sciences

Sifting Sound

Interactive Extraction, Exploration, and Expressive Recombination of Large and Heterogeneous Audio Collections

by

Nikhil Singh

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning in partial fulfillment of the requirements for the degree of

Master of Science

Abstract:

How can the tasks of searching, finding, and combining sounds be transformed into acts of exploration, discovery, and design? In the age of big data, recorded sound abounds in both volume and variety. When data is used not for the passive extraction of knowledge, but as aesthetic material for creative media production, the human role in curating and shaping its contents is even more essential. While access to this growing expanse of material is nominally more available than ever, this access is encumbered by the time, technical facility, and experienced aesthetic judgment it takes to effectively scan large amounts of unedited and unstructured media for contextually interesting elements, discover connections among them, and recombine them intentionally into new compositions, leaving extraordinary creative resources unexplored.

This thesis offers a new platform for audio exploration, discovery, and creativity that addresses this problem by developing tools that build on, rather than replace, human creative judgment in extracting interesting samples from long audio recordings, navigating across a large number of these to form subcollections, and guiding sounds back into expressive musical forms.

Thesis advisor:
Tod Machover
Muriel R. Cooper Professor of Music and Media

Sifting Sound

Interactive Extraction, Exploration, and Expressive Recombination of Large and Heterogeneous Audio Collections

by

Nikhil Singh

This thesis has been reviewed and approved by the following committee member

Tod Machover . . . . . .
Muriel R. Cooper Professor of Music and Media
MIT Media Lab

Sifting Sound

Interactive Extraction, Exploration, and Expressive Recombination of Large and Heterogeneous Audio Collections

by

Nikhil Singh

This thesis has been reviewed and approved by the following committee member

Joseph A. Paradiso . . . . . .
Alexander W Dreyfoos (1954) Professor
MIT Media Lab

Sifting Sound

Interactive Extraction, Exploration, and Expressive Recombination of Large and Heterogeneous Audio Collections

by

Nikhil Singh

This thesis has been reviewed and approved by the following committee member

Tristan Jehan . . . . . .
Independent
ex-Director of Research, Spotify
PhD, MIT Media Lab

Contents

Figures

0

Introduction

The composer Edgard Varèse once asked the question "what is music but organized noises?" The music of our time is a music of sounds. The proliferation of broadcast and mechanical reproduction technologies in the mid-twentieth century gave the designation of music as organized sound new meaning, ultimately giving rise to a diverse array of musical styles in which sounds, not notes, form the essential units from which music is built. Collecting sounds becomes an important process for sound-based composition, because their sources are conceptually infinite. In contrast to the finite number of sound-producing instruments available to the acoustic composer, even with their innumerable techniques, selecting and collecting sounds to compose with is an essential first step, in a way defining an instrumentation to work with. Additionally, sonic materials also play an essential role in other media arts, including films, theater, and games.

As collections of audio content, whether recordings, synthesized materials, archives, music libraries, speech corpora, and otherwise scale upwards in size, diversity, and number, creative projects that employ them are increasingly encumbered by the time, energy, effort, and attention required to effectively access and shape them using traditional media production software environments. These traditional tools are often useful for precise, deterministic actions with small media collections. While this exciting and increasing wealth of content should perhaps offer ever more creative possibilities, audio software tools for editing and creating with them lack support for spontaneity, agility, and flexibility needed in proportion. Instead, they rely on intensive and manual processes whose outcomes

are difficult to adjust.

Conversely, tools that are able to manage large scales of content often operate autonomously, are difficult to interpret and control, and are designed to accomplish specific tasks. Chapter 1 looks at this and other related work, and reviews some background information. Additionally, in chapter 2 we survey several past projects involving the author, and how they contributed, through their ideas, implementations, and limitations, to the goals and methods formulated and developed in this thesis work.

The *Sifting Sound* project proposes a new model that connects these two approaches, and focuses them around augmenting and extending human creative actions across large and unstructured audio corpora. More specifically, it consists of models, interfaces, and infrastructure to support three areas of sound-based creative activity: curation, collection, and composition through combination.

The *Sifter* addresses curation, in the context of discovering interesting or useful segments in long soundfiles. The *Observatory* builds on this to support navigation and cataloguing tasks, transforming them into exploratory interactions. Finally, the *Sketchpad* is focused on composition, assembling sounds into collages guided by audiovisual sketches provided by the user. Chapter 3 looks in detail at the design and implementation of these modules, as well as the rest of the supporting system.

These modules are composed into the *Sifting Sound* software ecosystem, which is designed to be deployed in settings ranging from personal sound libraries to large-scale collaborative projects. In chapter 4, we discuss two planned and imagined application scenarios. As a whole, *Sifting Sound* aims to be a creative environment for extracting and assembling meaningful aesthetic material from large audio databases and leveraging their size as a creative asset, enabling new forms of sonic creativity at new scales.

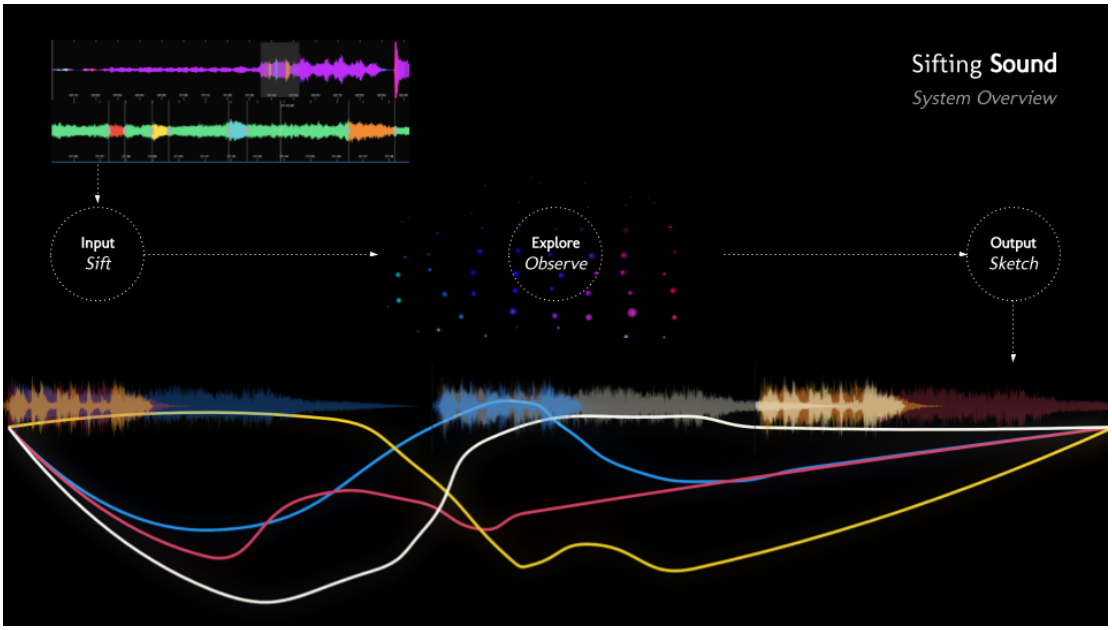


Figure 1: Overview of the *Sifting Sound* system.

1 Background

1.1 Artistic Context

1.1.1 From Notes to Sounds

Given the focus of this project, it is important to consider the history and development of music-making that focuses on sounds as basic units, rather than on notes. The category of sounds admits a much broader range of acoustic phenomena than notes, as produced by traditional musical instruments. Musicians have drawn on this rich, expansive domain for centuries, imitating natural or animal sounds with instruments or the voice, for example, in a variety of musical cultures. In 1913, painter and composer Luigi Russolo wrote:

> The variety of noises is infinite. If today, when we have perhaps a thousand different machines, we can distinguish a thousand different noises, tomorrow, as new machines multiply, we will be able to distinguish ten, twenty, or thirty thousand different noises, not merely in a simply imitative way, but to combine them according to our imagination.[73]

Russolo realized his vision by constructing a number of noise generating devices called Intonarumori. With the advent of electronic sound technologies allowing

for both the recording and synthesis of acoustic materials, the idea of a music based on sounds was led forward in a few different ways. Perhaps most relevant of these is the development of musique concrète, a hugely influential style of music based on the manipulation and application of recorded sounds. This style and its methods emerged first in the 1940s, beginning with both Halim El-Dabh in Cairo, and Pierre Schaeffer in Paris. Schaeffer described its goals as being:

> Instead of notating musical ideas on paper with the symbols of solfege and entrusting their realization to well-known instruments, the question was to collect concrete sounds, wherever they came from, and to abstract the musical values they were potentially containing.[25]

Already, we can see a clear integration of the world of sounds into musical composition, and a particular emphasis on collection. This pattern of collecting and combining sounds has seen development and application through a number of different styles and communities of music since, and with source materials including both acoustic and electronic (i.e. synthetic) sources. Specifically within the domain of acoustic sources, an important movement is the study of acoustic ecology, and its integration with the practice of soundscape composition[88] beginning in the 1970s. This approach to composition helped to motivate the use of field recording, a type of sound capture in which sources of interest are not brought into a studio environment, but addressed in their natural environment through the use of mobile recording technologies.

Field recording, as it is employed now, often generates large amounts of audio content. In many cases, not all of this content is used; particular recordings may be selected, or, commonly, long recordings may be edited to retrieve moments or segments of interest or contextual relevance. Another area in which these techniques are very often applied is sound design, as used in multimedia projects like film or video games. The purpose of sound design may be to create or re-create so-called diegetic sounds, which may be seen visually, or to produce sound effects for commentary or dramatic effect. A sound designer might conduct exploratory recordings, to later be examined for useful elements to extract and process.

1.1.2 Audio Sampling and Collage

The term sampling refers to two interrelated practices, in music. The first of these refers to capturing a specific segment of audio, often a produced piece of music, and integrating it into a composition. This is similar to musical quotation, in which themes or ideas may be borrowed from existing repertoire and developed into new pieces. The second form of sampling refers to collecting a number of segments and mapping them onto an instrumental controller, often by pitch but sometimes other parameters or even arbitrarily.

The former type of sampling has largely proliferated through hip hop. Beginning in the 1980s, hip hop DJs excerpted, reused, and combined segments of other recordings to produce new forms.[59] Paul D. Miller aka DJ Spooky That Subliminal Kid notes "DJ culture—urban youth culture—is all about recombinant potential."[61] Miller makes a comparison to collage:

> For the early filmmakers, such as Georges Melies and the Lumiere brothers, editing and being able to splice film was part of how to put scenes together. Related to this was the collage culture of Pablo Picasso and the poetry of Guillaume Apollinaire, who were juxtaposing phrases and pulling random elements together to make language poems.

A number of related practices deal with this collage-like sound construction process. John Oswald's plunderphonics[67] involves combining many segments together into new compositions. Mashups are a more general form of recombinant media, though the term is often used in musical contexts as well. Mashup artists may alter recordings or excerpts in order to re-contextualize them. Microsampling, as exemplified by the DJ and producer Akufen, involves recording, editing, and arranging very brief samples into phrases and ultimately larger musical structures. Akufen notes: "with the PC you could now record hours of sound matter. So the possibilities became endless. The world became our sound source."[41] Even further along the time-scale of collage materials, the micromontage technique[70], used by composer Horacio Vaggione, edits sound into small sound particles, and assembles these into collage-like structures. Through a variety of techniques, collage makes available an expansive and growing ecology and economy of sonic resources in ever larger quantities.

1.2 Audio Analysis

Audio analysis techniques give us a way to computationally derive and estimate physical and sometimes even perceptual qualities of sounds. In many cases, this takes the form of applying transformations to the audio signal to retrieve information we expect to be correlated to its physical or perceptual attributes.

1.2.1 Feature Extraction

In this project, we deal predominantly (though not exclusively) with signal representations that are explicitly designed, often called *engineered* or *hand-crafted* features[16]. Significantly, these features apply domain knowledge to produce quantities that describe meaningful aspects of the signal.

These features can serve as a form of dimensionality reduction[33], making them more compact and less intensive to process than the original data. They are also usually interpretable as descriptions of what we see, hear, and understand, allowing for physically- or perceptually-motivated decision-making in systems that use them, though they do range into more abstract territory as well, in which case interpreting them through such a lens may be challenging. Even so, these measurements are often more useful to work with than the original audio signals.

Within the modality of sound specifically, a very large number of such features have accumulated in the literature, and especially in the Music Information Retrieval (MIR) and Automatic Speech Recognition (ASR) communities. Later in this thesis, we explore the features chosen for this project and describe their interpretation where possible, though several excellent reviews of common and useful audio features are available.[68 82]

In recent years, much attention has been focused on *learned*[9] features. In contrast to engineered features, these representations are learned automatically from data, increasingly using deep neural networks. For example, deep autoencoders are very effective at mapping data to a low-dimensional latent space, because their objective is related to the quality of the reconstruction from it. Classifiers that are trained with annotated data similarly may have compact and effective representations that are produced before their output layers, once their weights are optimized.

Two challenges of these learned features are that they are difficult to interpret, if

interpretation is even possible, and that in the case of DNN-based features, they are often very slow to learn and require a lot of data. There are some kinds of learned features, specifically the output of dimensionality reduction methods like Principal Components Analysis (PCA) that we use in this project, which are much faster to compute in cases where interpretability isn't required. Additionally, a few DNN architectures have become popularly used as feature extractors because of their ability to generalize, and our analysis pipeline does employ one such pre-trained model (VGGish)[32], as an additionally available collection of features.

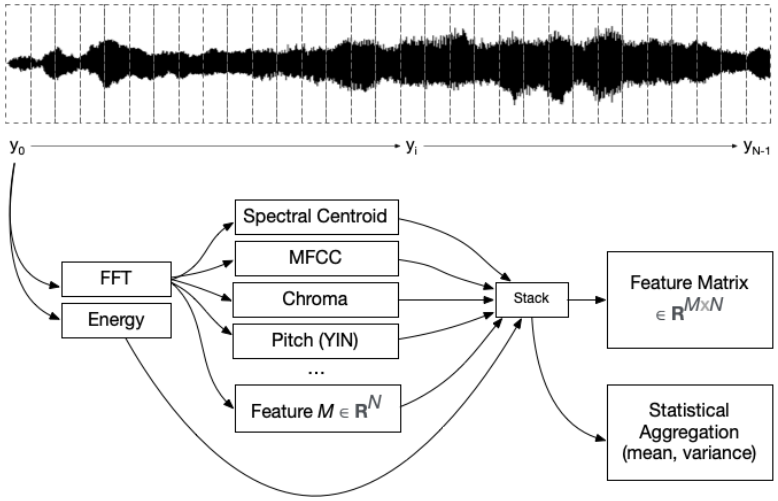


Figure 1.1: An example audio feature extraction process.

1.2.2 Methods for Data Analysis

Here we look generally at a few specific methods for data analysis used in this thesis, to better motivate their applications discussed in the following chapter.

Variable Selection

Variable or feature selection is the process of selecting a subset of available features, typically explicitly performed as a pre-processing step for training a predictive model[39]. A number of domain-general, data-driven techniques have been developed for accomplishing this step, often based on either statistical measures of feature independence, or evaluation of models using feature subsets.

For this project, rather than adapt standard feature selection techniques, we apply domain knowledge alongside dimensionality reduction methods. This allows us to impose semantic constraints on the problem, while simplifying and lowering data dimensionality when needed.

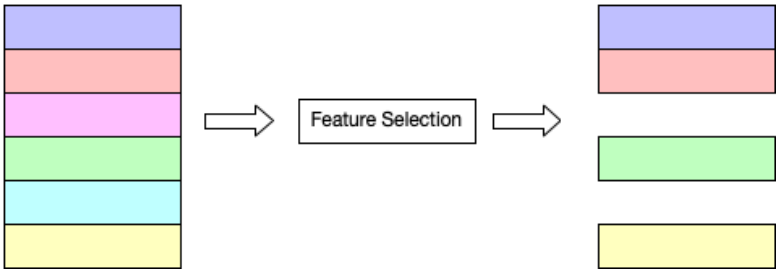


Figure 1.2: An illustration of the process of selecting features.

Dimensionality Reduction

While feature selection, as noted above, can reduce data dimensionality, we employ two methods for producing compact representations, that are abstract but significantly task-optimized. These methods may also be considered *learned features*, as discussed earlier in this chapter.

The first of these is a simple, efficient, and common method: Principal Components Analysis (PCA)[78]. PCA calculates a set of N linearly independent variables from multidimensional data that most explain its variance (where N is a parameter that indicates the output dimensionality).

The second is a recent and increasingly popular non-linear, graph-based manifold learning approach called Uniform Manifold Approximation and Projection (UMAP)[58]. UMAP seeks to capture a representation of the source data's underlying topological structure, and optimize a lower-dimensional embedding that preserves it. We use it here for its favorable balance of local and global structure maintenance in comparison with other popular methods like t-SNE.

Since UMAP is a stochastic algorithm (it uses stochastic gradient descent for optimization) and the results are difficult to interpret and depend to a great degree on chosen hyper-parameters, we use this algorithm primarily for visualization, and secondarily to find approximate relations between samples.

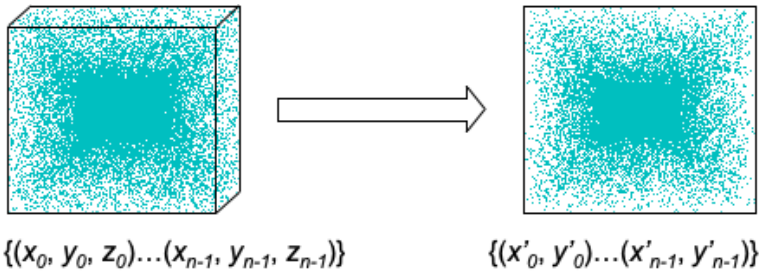


Figure 1.3: A graphical depiction of dimensionality reduction from three dimensions to two dimensions.

Time-Series Analysis

The two time-series problems that arise in the context of this work are outlier detection and subsequence-similarity search. For the former, we propose one simple peak-based algorithm, and specify another method that is feature-based, and uses the Matrix Profile.[90]

The Matrix Profile is a data representation and a collection of algorithms to efficiently compute it. The resulting representation, given a time-series and a window length, is the Euclidean distance under z-normalization between each window in the time-series and its nearest neighbor (itself another time-series window). To do this, it uses Mueen's Algorithm for Similarity Search (MASS), which is a fast method for computing distance profiles, or sliding-window z-normalized Euclidean distances, between a query time-series and each query-sized window in another time-series. We also use MASS directly for subsequence-similarity measurement, in addition to a similar feature-based method that we develop.

1.3 Selecting Sounds

1.3.1 Auditory Salience

Auditory saliency is a complex phenomenon. Generalized models, those that are not narrowly domain-specific, of auditory salience often either employ psychoacoustic models, or rely on annotations to determine ground truth[47]. If we understand the salience of a sound segment as its perceived prominence in relation to its surroundings, then perhaps a simple starting point is loudness.

Intuitively, we might expect that loud sounds stand out more. The usefulness of loudness as a predictor of auditory prominence has been studied, and found to be performant in a range of auditory settings (e.g. speech, music, and natural sounds). It has also been shown that loudness combined with other perceptual aspects of a sound may yield more optimal measures of salience[79], though the effects of other features may also be context-dependent.

A central challenge to employing models of salience in creative projects is that salience may not predict the usefulness of a sound for a creative task. This is perhaps difficult to verify experimentally for the same reason that we might expect it to be the case: the introduction of aesthetic preference, a highly irregular domain, to the already complex sensory, perceptual, and cognitive challenges involved in models of auditory salience and attention.

Instead, we will discuss four simple computational techniques that rely on modeling manual audio editing practices themselves based on loudness (encoded visually as waveforms), on multi-feature novelty measures (considering the link between novelty and creativity[34]), referential methods that seek sonic queries from users, and interaction to make all of this fundamentally led by human exploration. Importantly, the outcomes are also selected by human judgment; the software we describe provides a suggestive layer that partially automates this process, and can ultimately optionally admit more sophisticated measures of salience as well.

1.3.2 Searching Sound

Searching through auditory streams to retrieve segments is often framed as a Sound Event Detection (SED)[60] task. SED typically deals with identifying onsets and offsets for brief occurrences of identifiable sound-sources (e.g. car-horns) in (especially non-speech) long audio signals or real-time audio streams. Solutions are typically implemented as autonomous processes that annotate signals with the relevant details.

A related area is sound annotation. Sound annotation tools are typically developed as interactive interfaces that allow human annotation of sound events[17], often used as datasets in training or evaluating SED algorithms and models. Two tools in this area that relate to the work in this thesis are *SoundsLike*[37] and *I-SED*[46]. *SoundsLike* uses measurements of audio similarity to augment the audio annotation task, specifically within the context of SED in movie soundtracks. *I-SED*

formulates sound annotation as a human-in-the-loop process to scale human annotation to very long audio files, with few concept labels available. Similar to *SoundsLike*, *I-SED* uses similarity to make the annotation process quicker, but also includes relevance feedback in order to update its similarity metric by re-weighting audio features based on user interaction.

The *Sifter*'s graphical interface is designed similarly to an annotation tool, but the goal is not to annotate. Rather, we expect the outputs to be sound segments extracted from the source audio. As such, the underlying computational methods employ different techniques to perform the SED-like but concept-free task of finding sound events of interest without a sonic prompt, and the augmented-annotation-like task of finding segments that relate to a query, but allowing these to coexist and operate interactively.

Another important difference to consider with regard to the design goals of the *Sifter* is that its purpose is different: it is focused on retrieving interesting sounds for creative media production projects. As such, it is considered more important to facilitate the user learning the tool's consistent behaviors, characteristics, and limitations, than for the tool to learn a user model and adapt its behavior in accordance. Practically, this means that we avoid using implicit relevance feedback. Instead, we develop a purpose-designed typology of sounds and audio features, a method for automatically estimating feature relevance, and provide the basis for an explicit relevance feedback system that might then allow these automatically estimated feature weights to be adjusted by category (though not currently implemented).

Projects similarly motivated by creative goals sometimes use audio segmentation techniques to transform long audio recordings into collections of segments for application. One example of this is the Fluid Decomposition Toolbox, which conceives of sounds as dissoluble into slices, layers, and objects.[81] In this project, we take a different search-like approach motivated by human exploratory behavior, in attempting to build a platform that naturally extends and supports it.

1.4 Sketching Interfaces for Sound and Music

1.4.1 Drawing to Generate and Impose Structure

In *The Dialectics of Sketching*, architect and researcher Gabriela Goldschmidt writes:

> …by sketching, the designer does not represent images held in the mind, as is often the case in lay sketching, but creates visual displays which help induce images of the entity that is being designed.[36]

Goldschmidt designs a study to investigate the role of sketching in the architectural design process, ultimately conceiving of sketching as a form of "interactive imagery", a process of visual reasoning through which form develops. This stands in contrast to the notion of sketching as simply conveying, through graphical rendering, formed images held in the minds of the designers.

In music composition, sketches are well-documented as a method by which composers gradually develop their ideas into larger, coherent forms. Sketching has also manifested in the form of various graphic notational techniques, used by composers both in place of and in addition to standard notational praxis. One such technique particularly relevant here is decoupling[20], which involves parameterizing sound production, and then notating with the resulting parameters. Later, we will explore the development of a parametric visual language for electronic sound production that employs some similar implements.

A number of sketching interfaces have operated directly with sound, as far back as the first half of the twentieth century. One notable, relatively early realization of this idea is British composer and electroacoustic pioneer Daphne Oram's *Oramics*[56] technique. Oram's optical control and synthesis system grew out of observing an oscilloscope, which performs the reverse transcription (i.e. graphical depiction of sound), and the earlier development and proliferation of optical film soundtracks. Related systems were developed early in Russia, including Evgeny Sholpo's *Variophone* and Evgency Murzin's *ANS* synthesizer[6], in Germany by Oskar Fischinger , in Canada by composer Hugh Le Caine as well as by filmmakers Norman McLaren and Evelyn Lambart with composer Maurice Blackburn, and by others in these countries and around the world.[71]

One graphical sound system, developed later in the twentieth century, has been

particularly influential. The development of the *UPIC* system was led by composer Iannis Xenakis, in his role as the director of *CEMAMu* (now *CCMIX*). Cornelia Colyer, then a staff member working with Xenakis, describes the system in a studio report:

> The composer designs his graphic score on the table as a series of "music pages", where a "page" is simply a musical sequence of from one to several thousand notes (constant pitch or as complicated a glissando as desired) with a duration that he specifies and may respecify at will. Each note, or "arc", on a page is drawn with the pen in the large, millimetered "design zone" on the table with time as the x axis and pitch as the y axis, and is displayed simultaneously on the system's CILT screen. To each note the composer attributes a timbre, an envelope, and an average intensity. [22]

This sophisticated system, with its time-frequency representation, has set the stage for much work since in developing "drawn sound" interfaces and computational systems, including digital versions of the *UPIC* system and elaborations upon it.

A key feature of these devices is that the sketches or drawings act as control signals for synthesized sounds. There exist other systems that design sketching gestures as interfaces to sound recordings, as ways to impose structure onto collections rather than generate structure by synthesis. Several interactive sound systems use scatterplot-like representations to display large collections of sounds or sound segments, a number of which also use the Corpus-Based Concatenative Synthesis procedure outlined later in this thesis (including projects by the author, detailed in the following section). Examples of such projects include *CataRT* [77] by Diemo Schwarz et al., *earGram* [10] by Gilberto Bernardes, the *Infinite Drum Machine* [57] by McDonald, Tan, and Mann, and a number of others, including the recent commercial software product *XO* by XLN Audio [48]. One interface that explicitly treats sketches as scores, i.e. discrete and repeatable structures that can persist, is van Troyer's *Constellation* [84] (fig. 1.4). Drawing in this environment imposes structure by connecting sounds linearly and forming visual paths.

Some drawing-based media composition systems instead opt for parametric forms of drawing. Significantly, Farbood's *Hyperscore* [28] environment assigns parameters to drawn freehand contours, interpreting them as statements and elabo-

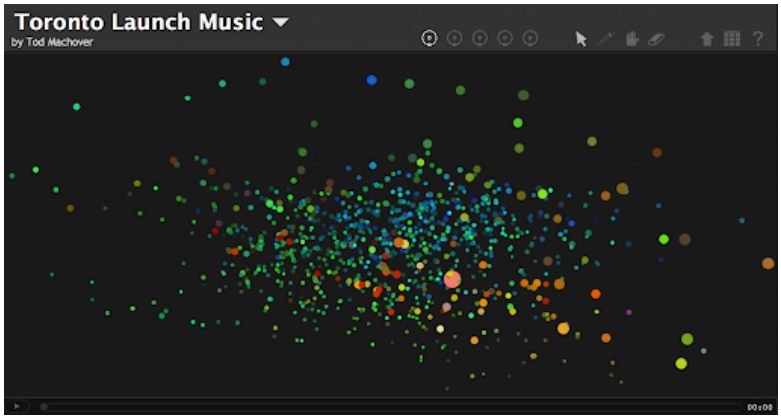


Figure 1.4: A screenshot of van Troyer's *Constellation*.

rations of pre-determined motifs, with systems that help to produce contextually relevant music. The parametric approach mediates between user control and interface simplicity by developing an underlying model that establishes a shared stylistically appropriate language. In the multimedia setting, Peter Torpey's *Media Scores*[80] establishes a language of "expression parameters", and a system for notating these that deploys color to differentiate them. We will later see how these ideas are adapted to sketching in the *Sifting Sound* project's *Sketchpad* interface, within the specific modality of sound.

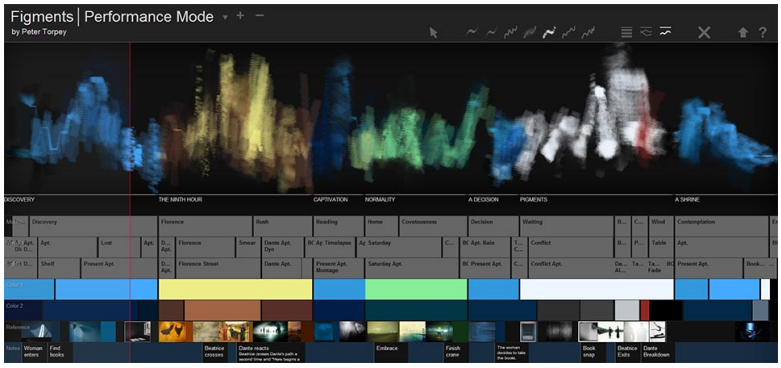


Figure 1.5: Torpey's *Media Scores* system.

In digital audio workstations (DAWs), parametric notation is commonplace in the form of automation. Automation allows time-varying parameter forms to be written and read to a persistent store. In a sense, this mirrors a similar workflow found in hardware-based audio processing and mixing practices: the adjustment

of tool parameters in real-time, and the recording of the output of the associated processes to a fixed store, such as tape. However, digital systems allow automation to be drawn by hand, interactively edited, added, and removed in large numbers at any point in the process before the outputs are rendered, facilitating the kind of dialectical sketching patterns alluded to earlier in this section. Automation patterns take on a new role, that of interactively sculpting sounds.

In the sketching system we describe, characteristics of a number of these diverse systems are combined, and added to through "sonic sketching", a different modality for expressing ideas. We will explore the intersection of the generative, parametric time-based notational systems with the structure-imposing characteristics of those that work directly with sound collections.

1.4.2 Sonic "Sketching"

Another approach to designing sound with provisional representations is directly with sonic materials themselves. There is a considerable range of prior and active research and development related to this topic. Information retrieval systems may implement query-by-example (QBE)[43] or query-by-vocalization (QBV)[27] techniques for retrieving sounds, allowing representations from users to retrieve similar sounds. QBV systems explicitly rely on performed vocal mimicry, typically non-speech utterances intended as direct acoustic representations, rather than linguistic forms like onomatopoeia.

Some systems have used similar vocalizations not to retrieve sounds, but rather to parameterize sound synthesis engines. One project building on this idea is *SkAT-VG*[72], which aims to combine vocal sketching with gestural articulation to produce a bimodal interface for creative sound design.

Another way that audio examples are used for sound design is in systems that assemble sounds together to match a target sound, which is particularly relevant to the *Sketchpad* model. One way this is done is by retrieving brief, typically uniform-length, segments matched by feature values to a time-varying target, as is typically done by CBCS-based systems that use a target sound. A related approach is found in the *AudioGuide*[40] system by Hackbarth, Schnell, and Schwarz, which combines sound segmentation with a matching pursuit algorithm to best approximate target segments with combinations of time-varying database sound segment features.

The research area of computational or computer-aided orchestration usually needs

to approximate a sound target with a combination of database sound units, and often formulates this as a combinatorial optimization problem. The popular *Orchidée* tool, for example, uses a multiobjective constrained optimization process, with objective functions computing feature dissimilarities, and constraints that help to supply ecologically valid solutions, given musical context and the composer's requirements.[19]

The *Sketchpad* offers multiple methods for approximating sonic sketches with sound combinations, but it is expected that the solutions will be quite dissimilar from the sketches because of the diversity and variability of available sound sets, since these materials are not pre-determined. This is desirable in our case: sketches set the scene for resulting collage pieces, but much precision is neither required nor expected. Instead, we will adapt similar methods to the general situation of creative sound-based composition, introducing additional ideas and steps to match the methods to the task, and pairing this system with graphical input methods, of the kind discussed in the previous section.

1.5 Summary

In this section, we have noted several artistic and technical precedents for specific areas, ideas, and techniques to be detailed later in this thesis. In addition to the ways that this work builds on these independently to produce useful functionality, a significant goal implicit in its design is to integrate these diverse practices and processes into a synergistic environment guided by artistic background, and connect them through content, purpose, and creative application.

In the introduction, we considered a comparison to conventional digital audio composition and media production tools. The Digital Audio Workstation (DAW) is the state-of-the-art for integrated sonic creative production, and while some of the techniques described in this section have been used to augment this environment, they largely exist in parallel, deployed in projects and limited in their scope of application. *Sifting Sound* envisions a new workflow, environment, and toolkit for partially replacing and partially augmenting this traditional platform through collaboration between human and machine processes, and points to future work designing powerful new musical tools for the data age.

2

Motivation and Initial Explorations

2.1 Early Projects

In this section we explore a small selection of previous projects that have helped to motivate both the broad design goals of *Sifting Sound*, as well as the specific techniques developed and used in working to accomplish these goals. These projects represent a range of ways in which to creatively manage and transform sound collections, mutually contributing ideas, techniques, and lessons learned, and each pointing to useful paths forward.

The projects are first described, both in concept and implementation, with the following section examining specific ways in which they connect to one another and influence the primary work detailed in this thesis through their methods, designs, and limitations.

2.1.1 Audio Editing for *Gammified*

Gammified[4] is a piece by Tod Machover for string quartet and electronics, commissioned by the Kronos Quartet as part of their *50 for the Future*[50f] project, and premiered by them at the MIT Media Lab in April 2019.

The piece is built around a Gamma frequency electronic pulse wave signal, which acts as an underlying drone that both the live quartet and the produced electronic

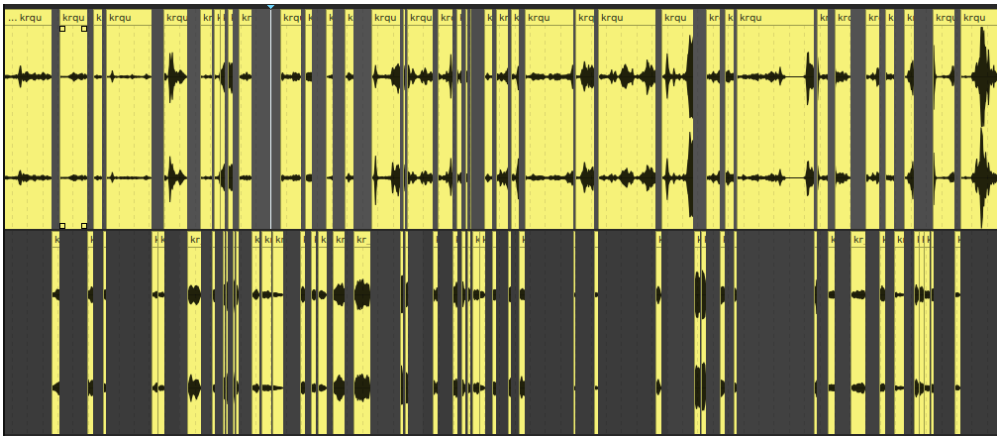


Figure 2.1: Audio editing for *Gammified*: source material is in the top track and excerpted segments in the bottom track.

layers build on. A number of the electronic sounds used are taken from a three-hour-long recording of the Kronos Quartet made at MIT in 2018, in which they experiment with and demonstrate various instrumental techniques, improvise around suggestions from the composer, and even perform some written material. Part of preparing the materials for the piece's electronic layers was to review the contents of the aforementioned recording in detail, and extract a collection of diverse sound excerpts that were both independently salient and also mutually compelling.

Analysis

To accomplish this with traditional audio editing tools, as used by the author in the project, it is necessary to rely on a largely linear workflow. This could mean listening to the audio file from beginning to end, marking segments to retrieve and perhaps even making the corresponding edits along the way, which would be a globally linear process with respect to the content. Alternatively, it is possible to observe regions of the audio with large amplitudes or that contain patterns in the amplitude, through the use of waveform displays. Identifying these can act as anchor points, around which a locally linear search-by-listening process can then be employed.

An advantage of the latter, bi-sensory approach is the distribution of attention over the media's duration by partial observation of its contents (i.e. amplitude, shown by waveform displays), rather than by assumption or circumstance in the

case of searching linearly from beginning to end. Another advantage of this approach is that it may allow the user to find connections between sounds available in different parts of the audio file, yielding collections that may have more internal logic, or allowing them to have a more specific character.

Even so, the use of waveform displays doesn't scale well to very long audio content, because the resolution available for decision-making reduces with the length of the audio, challenging both of the observed advantages. Traditionally, this might be managed by zooming in to an arbitrary section of the waveform, performing the search process, and then repeating for remaining sections. Such an approach was necessary in this project, as was periodically reviewing the collection of already segmented sounds for context.

The entire process took hours, and about 75 segments were ultimately selected through careful auditioning, comparison, and editing. These segments were then organized into a keyboard-based sampler instrument, so they might be more easily combined through performative gestures. While this does make them playable, a major limitation is that keys on a keyboard are pitch representations. They communicate little about the sounds assigned to them, and so provide a somewhat obscure and inconsistent mechanism for composition.

2.1.2 *Kronospaces*

Kronospaces is a visual interface for exploring a significant number of audio excerpts, similar to those mentioned in the previous section. The segments are taken from the same three-hour-long recording by the Kronos Quartet, as well as variously sampled from the Kronos Quartet's entire recorded catalog through a similar process. In this case, the corpus consists of several long audio files from which excerpts needed to be retrieved.

Implementation

The *Kronospaces* visual interface is implemented in C++ with openFrameworks, and the audio analysis and interaction system is implemented in Python and Max/MSP with the MuBu[75] package. The visual interface consists of six vertically-delineated sections, each representing a different category of sounds.

To derive these categories, sounds were manually sorted into one of six groups. The group identities were derived from early experimentation with automated au-

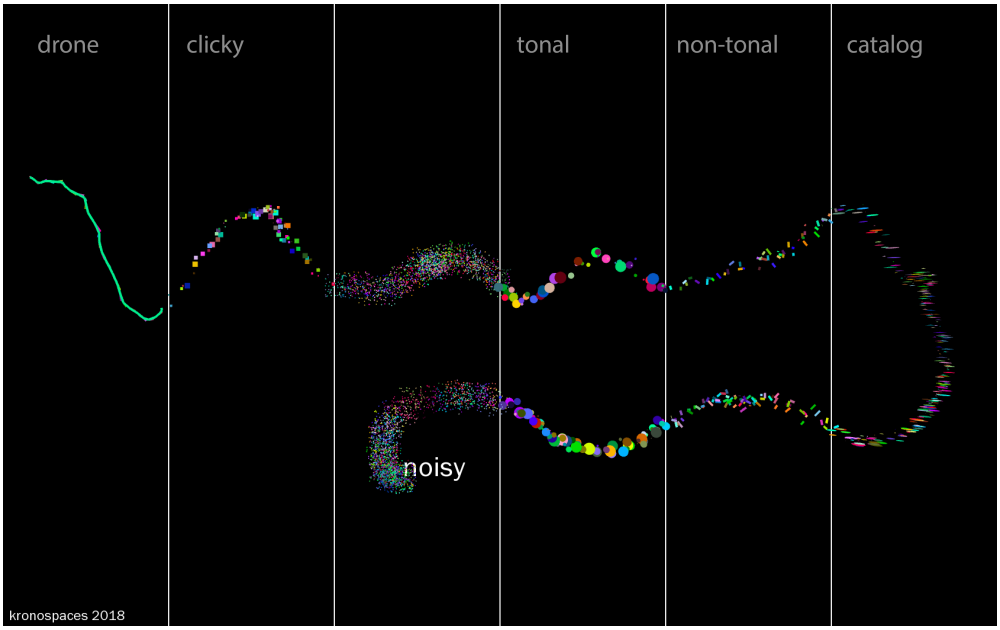


Figure 2.2: *Kronospaces* interface.

dio analysis and clustering pipelines that employed audio feature extraction and K-Means clustering to produce distinct collections. An early set of these was labeled by hand based on predominant acoustic qualities, and these collections were added to and re-organized as necessary with the addition of new sounds. The resulting six categories, and corresponding visual spaces, are "drone", "clicky", "noisy", "tonal", "non-tonal", and "catalog". With the exception of the last one, which consists of a diverse group of sound excerpts representing the range of the Quartet's recorded repertoire, each category name identifies its sounds predominant character.

The visual interface allows the mouse to be dragged around the screen to perform these sounds, with a path drawn that morphs into representations of each section's general sonic qualities. Under the hood, these visual gestures control a simple sound synthesis engine that produces a target signal for a Corpus-Based Concatenative Synthesis (CBCS)[76] system, a data-driven method adapted from text-to-speech synthesis. CBCS generates a continuous sound surface by concatenating sound segments together, often based on matching analyzed audio features to those of a target signal, as in this case.

The target here is a combination of a sinusoid and white noise, mixed together

based on the spatial position of the mouse. The horizontal axis within each section is mapped to the mix between the sine-tone and noise, and the vertical axis is mapped to the frequency of the sine-tone. Moving the mouse to a different section switches the underlying audio corpus, but replicates the control mappings for the target signal synthesizer.

Analysis

This project introduces the idea of automating some aspects of sound collection management and organization. It follows a similar manual curation process to the editing for *Gammified* to collect sounds, but then combines human judgment with computational assistance to uncover structure and organize collected sounds into creatively meaningful groups.

The connections between the human-led and computer-led tasks were largely ad-hoc; programs were written to accomplish specific jobs as directed by manual exploration of the corpus. Additionally, manual adjustments made afterwards removed the content from the computational domain and back into the realm of manual editing and preparation, since there was no interactive system to incrementally make with.

2.1.3 *Scream*

Scream is an audiovisual installation, also based on a CBCS system, that uses face-tracking as a control input, and a corpus built from the audio portion of the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)[53] by Livingstone and Russo (licensed under CC BY-NC-SA 4.0), which is a database of emotional speech and singing performed by actors.

Scream uses perhaps the most primal sound-making device, the mouth, in a silent, visual capacity; the installation's title doubles as an instruction, encouraging participants to scream for whatever reason they may want to through the voices of others without needing to make any sound acoustically.

Implementation

The software architecture for *Scream* is similar to that of *Kronospaces*. The graphical interface and face-feature extraction system are written in C++ with openFrameworks, along with OpenCV for implementing CLAHE (a popular local

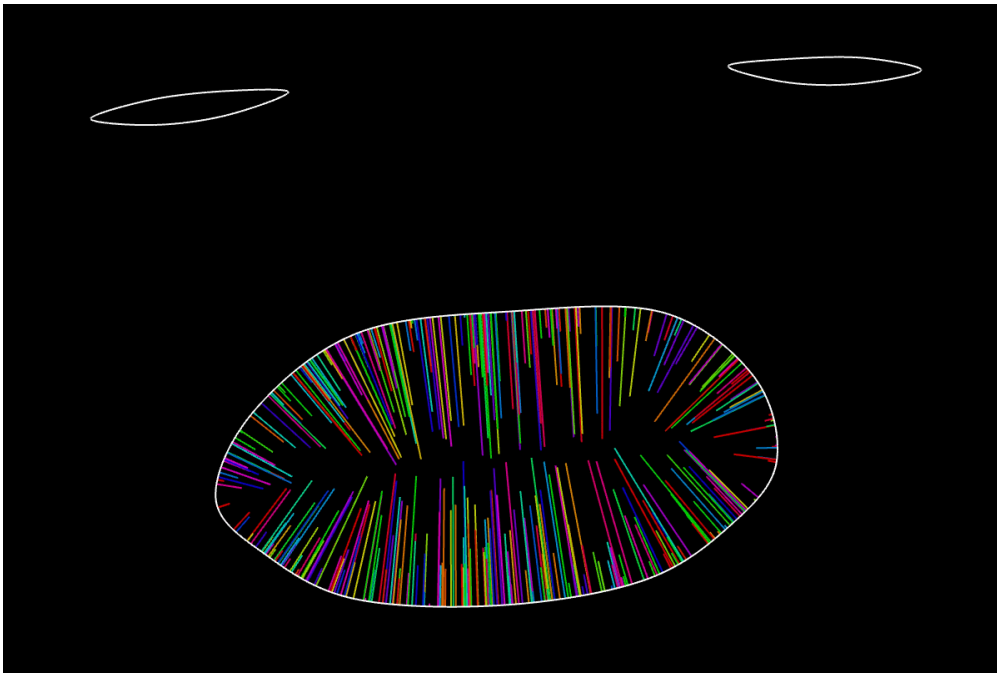


Figure 2.3: *Scream* interface.

contrast enhancement method, as a pre-processing step) and Kyle McDonald's ofxFaceTracker addon. The audio system is implemented in Max/MSP, again with the MuBu package.

A camera tracks the face of a user, and measures relating to the openness of the user's mouth and eyes are obtained. A simplified outline representation is drawn on the screen, for visual feedback. The openness of the mouth, as measured, is mapped to the amplitude of the CBCS system's unit selection algorithm, and the openness of the eyes is mapped to the estimated fundamental frequency. In contrast to the *Kronospaces* system, there is no sonic target (the target is instead a two-dimensional feature vector), and there is only one corpus.

Analysis

This project, despite its similarity to *Kronospaces*, follows a different organizational paradigm. First, it begins with a pre-assembled corpus. Second, rather than place sounds in discrete categories, it creates a continuous feature-based map of sound segments for navigation, and leverages this structure to design an

interaction.

There are benefits to this continuous approach: it creates an easily traversable representation of the entire space of available sounds, and requires much less human work. This has the limitation of not allowing more sophisticated or precise applications; the interaction is essentially limited to exactly such a traversal. Given the large number of sounds (about 2500), most other kinds of production with them would likely require an intensive manual search and selection process, as an initial step to produce a smaller collection more feasible to work further with.

2.1.4 *ObjectSounds*

ObjectSounds is a project, developed as an iOS application, that analyzes a visual scene and produces sonic collages. It uses a pre-trained model for camera-based object recognition, then searches the freesound.org[30] sound database for recorded sounds that represent the identified objects in a scene, and then puts these together, forming a representation of the latent sonic potential in the camera's view.

The project's goal is to spark sonic imagination by illustrating everyday objects' possible sounds, and to encourage exploration and imagination of sounding objects' potentiality. Everyday objects have been realized as musical materials in a range of creative practices, ranging from spoons[Spo] and glasses or bowls[21] in several musical traditions, to twentieth-century concert and electroacoustic music[52], to new electronic instruments designed specifically for this purpose.[85]

Analysis

ObjectSounds draws on a large online sound repository, greatly expanding the range of available sound materials. Since options in this case number in the hundreds of thousands, it is important to introduce some form of constraints that limit the actual composition process, which produces heard output, to work with some subset of these options. Additionally, limiting the number and diversity of options also lends specificity to the compositions, allowing each one to have a unique character.

The way this is accomplished here is through using detected objects in a scene to gather sounds from. More generally, it introduces specificity through determining

context supplied by the user. Context specification can be powerful, but in this case the input is concept-based rather than content-based, resulting in a search for semantic categories of sound. While this may be useful, it offers little in the way of control for the output compositions, leaving them largely unshaped by craft and intent.

2.1.5 The *AudioObservatory*

The *AudioObservatory* is an interactive platform for immersively exploring large audio collections. It uses a variety of analytic techniques to characterize audio collections, and latent relationships and structures that exist within them, using this information to place sounds into a dynamic three-dimensional visual space that adapts to the user's interest.

The *AudioObservatory* builds a model of the user's preferences, allowing new paths to be charted through the space to guide the user's exploration. As such, while similar sounds are closer together, the interface allows meaningful connections to be made across the entire collection. Furthermore, the interface can dynamically reshape itself by changing its underlying model of similarity between sounds, combining various measured parameters of the signals and, for example, weighing timbre, or pitch, or rhythmic features more or less. The positions of the sounds are changed, yielding new patterns, subcollections, and connections. The interface both guides and is guided by the user.

Implementation

The underlying systems were developed in C++ with openFrameworks for the visual interface, dimensionality reduction, and real-time audio code, as well as Python for the preprocessing and audio analysis software.

The preprocessing software extracts a number of features from each audio file, and then compares these using feature-defined distance functions (for example, Euclidean distance or Dynamic Time Warping[63]). These distances, produced pairwise and symmetric, are organized into matrices; one distance matrix for each feature of interest. In the real-time software, these matrices are loaded into memory as Eigen[38] matrices, linearly combined (allowing feature-relevance weighting), and passed to either the Multidimensional Scaling (MDS)[49] or Laplacian Eigenmaps[8] algorithms, to get three-dimensional embeddings that can be scaled into virtual positions for each sound.

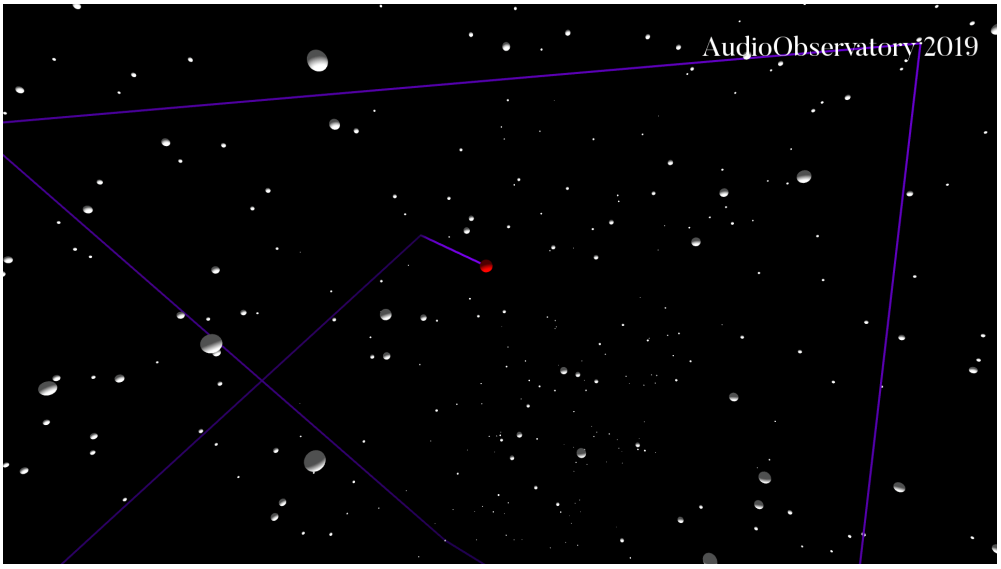


Figure 2.4: *AudioObservatory* prototype.

This map of sounds can be explored in a game-like manner, by using common peripherals like a mouse and keyboard, as well as multiple other inputs (such as a camera interface with face tracking), to navigate in desired directions. Additionally, user actions (such as explicitly marking two sounds as related) alter an underlying Markov transition matrix, which can then be used to generate paths across the space, beyond similarity alone. These paths are used to connect sounds and ultimately form meaningful subcollections to work further with.

Analysis

The *AudioObservatory* aims to build on the ideas of these past projects to offer a reusable software environment for sound exploration. Notably, it isn't tailored to a specific sound collection, but uses relevance feedback to alter its underlying model of similarity between sounds, and thereby its organization of sounds. This resulted in a more flexible system, but one with several parameters to tune. Additionally, while the system for charting compositional paths allowed for some interesting connections to be made, providing transition probabilities for all constituent sounds was an incredibly laborious process. Furthermore, both the transition and similarity matrices made handling growing collections quite resource-intensive.

With regard to the audio output, the system allowed individual sounds and clusters of sounds to be played in a few different ways, offering position-based combinatorial possibilities. While this allowed sounds to be combined both linearly and vertically, it also biased these combinations toward similarity and again didn't allow for much "expressive match", or articulation of ideas and shaping of materials. Even so, the idea of a re-configurable space of sounds, as well as some of the technical approaches, were helpful aspects of the project.

2.1.6 *voicecoil*

voicecoil is an electroacoustic piece, partly inspired by Vladimir Ussachevsky's *Wireless Fantasy*[83], but explores synthetic signals with a much colder and harsher "digital" edge. These signals are taken apart into phrases and patterns, which are recombined through motion, interruption, and superposition.

Generating the sonic raw materials for this piece involved performing lengthy improvisations with a suite of digital synthesis and sound alteration tools. Similarly to the editing work involved in *Gammified*, this process was conducted by hand. In contrast, however, this wasn't a one-time editing task: after an initial pass-through, sounds were excerpted and retrieved incrementally, as needed for the piece during the course of its assembly.

Compositional decisions sparked ideas and necessities for related or specific types of sounds, which prompted a search process that built both on some general awareness of the media, as well as a sense of character to search for: some kind of "target", perhaps a sound with a large transient and drum-like envelope, or one with a little pitch but largely noisy timbre, for example. Or, as often was the case, segments that share some aspects of a sound already segmented from the source signal.

As such, this required some persistent representation of the source media to be available for multiple search sessions, as well as a way to identify materials that had already been selected and their original positions.

Analysis

This project is different from the others in the sense that the material was generated, edited, and applied in composition as part of a whole, integrated workflow. The initial content sources were made through long-form improvisation, an ex-

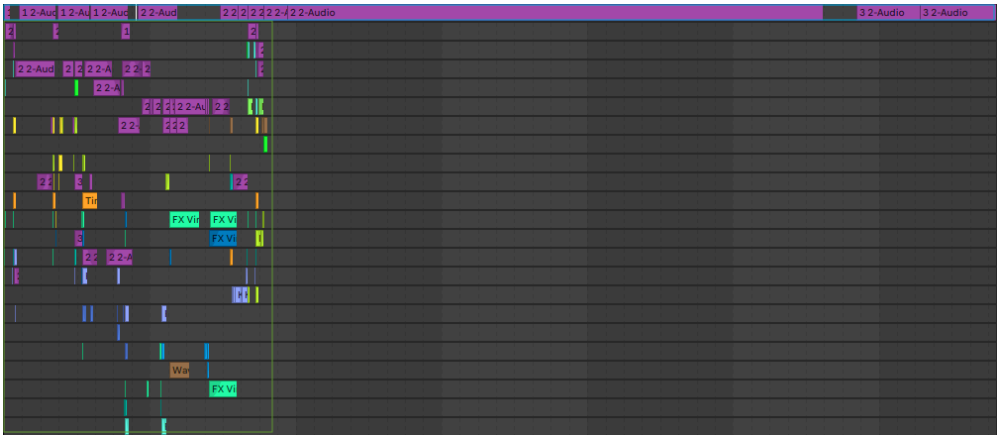


Figure 2.5: Audio editing for *voicecoil*. The blue box contains the source audio content, and the green box shows the arrangement of excerpts incrementally assembled by putting sounds together and processing them in context.

panding collection of sound segments was built incrementally through editing, and sounds were shaped and combined in such a way that prompted revisiting the source materials for more sounds. Selected segments were routinely combined into hybrid structures, and reused and modified. The piece itself was built by arranging such structures in time, and constructing and shaping material around them.

The piece itself is relatively brief (under four minutes long), as is the source audio, which was under half-an-hour in duration. Even so, the process of carefully selecting sounds, combining them in various configurations to try them, finding desirable patterns, and then returning to retrieve more material based on progress, took a great deal of time. Much of this time was spent on mechanical tasks, such as searching, organizing, and combining sounds, that depended on artistic impetus and judgment to evaluate and ultimately apply. For longer source materials and works, this process can take much longer, and may even only address part of the content available, to maintain the creative focus of the project.

2.2 Connection to Present Work

While the analysis for each project outlines its major contributions and limitations, and hints at how these relate to one another and to the formulation and development of *Sifting Sound*, in this section we look at themes that clearly connect the

projects in other ways, and establish important ideas we can build on later in this thesis.

2.2.1 Excerpting Long Audio Recordings

As noted, both the audio editing for *Gammified* and for *voicecoil* involved discovering and selecting interesting segments from a long audio file, albeit slightly differently; the former involved one session in which a large number of possible sounds of interest were identified and retrieved. The latter, conversely, treated the long audio file as a resource to search incrementally, with reference to external sound contexts.

These tasks provide the primary motivation for the design of the *Sifter*. Later, we will discuss how the implemented techniques support both context-free and context-driven (i.e. query-based) methods for sound extraction, which builds on the shared objectives and different approaches involved in these two projects.

2.2.2 Content-Based Organization of Sounds

In some of the projects described previously, audio analysis methods are needed for downstream sound organization tasks, which are themselves needed to support the designed interactions. Specifically, *Kronospaces*, *Scream*, and the *AudioObservatory* make use of such processes in related but also slightly distinct ways.

In the case of *Kronospaces*, we use audio feature extraction at two levels: the first one deals with discovering structure, i.e. helping to derive (along with human judgment) the sound categories, and the second supports the content-based retrieval for the real-time system. In this sense, we organize sounds in two ways; sounds are placed both in discrete groups, and annotated with extracted features to be easily retrieved by matching these values.

Scream works with audio features in primarily the latter sense, without explicitly organizing sounds into groups, but by labeling them with features for retrieval. Conversely, the *AudioObservatory* largely deals with the former, in the sense that the audio features are used to measure distances between sounds, in the perceptual sense, and mirror this internal structure in spatial positions assigned to each sound.

Two significant ways that the practices employed in these applications have influenced this thesis's project are: first, through the selection and parameterization of appropriate audio features for sound-based creative work, and, second, the design of a pipeline that integrates these into a system, and produces output in easily manageable formats.

2.2.3 Interfaces for Making Sound-Collages

All three interfaces just described produce sound-collages as their output, in the sense that they combine sounds vertically (sounding simultaneously) and linearly (in sequence) to produce artful composites, controlled by the user. A fourth of this collection of projects also produces such composites, *ObjectSounds*.

Kronospaces and *Scream* both use the Corpus-Based Concatenative Synthesis (CBCS) procedure outlined earlier, and as part of this, they segment audio files into brief segments for analysis and retrieval in real-time. They retrieve not only the very brief segments identified, but also surrounding material to obtain some of the identity of the source sounds. Even so, the structural integrity of the individual sounds isn't maintained, and isn't considered a priority in these projects; the aggregate space of sound in the collection(s) is explored. Additionally, it is the sequence of sounds that is controlled by the interaction; the layering is largely incidental, arising from overlaps between consecutive units.

In the other two projects, the *AudioObservatory* and *ObjectSounds*, the individual sounds are preserved. The *AudioObservatory* plays them as sound mixtures as the user traverses the visual space that represents the sounds, resulting in a somewhat arbitrary linear structure biased towards the underlying measure of similarity, and *ObjectSounds* assembles audio files into mixtures in real-time based on recognized visual objects, an even more arbitrary criteria for sonic combination. Still, each sound is able to be heard in its original form, allowing us to observe its shape, time-varying aspects, and duration.

In the current project, we take an approach that integrates the data-driven compositional process of the CBCS-based systems with the sound-preserving characteristics of the latter, to yield content-based composites that benefit from analysis, maintain the units' sonic identities, and allow complex but coherent sound collages to emerge, guided by creative input.

2.2.4 Stored Repositories of Sound

All the projects mentioned entail either the production, maintenance, or application of repositories of sound. While this is central to the *Sifting Sound* project, there are additional details about these repositories that are pertinent to the later discussion of how this thesis's repository is designed and engineered. The two editing projects, for *Gammified* and *voicecoil*, produce relatively small collections of sounds ($N < 100$). No special structure was required to contain these, because they can easily be searched by listening, and no additional information about the sounds needed to be stored.

In *Kronospaces* and *Scream*, sounds are stored as independent files, and analysis data that references these files is stored in specialized containers provided by the MuBu package (based on the SDIF[89] file format). For the *AudioObservatory*, independent files are used along with NumPy arrays written to disk, as well as JSON files for organizational data about the collection. This has the advantage of being purpose-designed for the application.

ObjectSounds relies on the existing Freesound database, which contains a large number of sounds searchable by various parameters, and accessible via a web API. In addition to being easily web-compatible and massively scalable, this provides a shared repository of sounds for clients to explore and select from as well as contribute new sounds to, offering the benefits of a growing ecosystem to individual content creators.

Sifting Sound once again attempts to synthesize qualities of both these types of structures in order to easily allow local deployment as well as server-contained databases shared across multiple clients, that benefit from the diversity and scale of contributions offered by users.

3
Sifting Sound

3.1 Project Definition

The *Sifting Sound* project is in a dense problem space, with much ongoing work. Consider the concept of auditory salience, for example: a great deal of research exists that addresses this from sensory, perceptual, statistical, neurocomputational, cognitive, and other perspectives[87 44 69 26]. Part of the challenge is that the general category of "sound" as a class of experiential objects is innumerably vast, diverse, and high-dimensional.

Much of the work in the field of Music Information Retrieval (MIR) operates under the constraints of solving specific problems with specific types of signals, corresponding to specialized subsets of recorded music (itself a specialized domain). Sound Event Detection, as described earlier, typically deals with semantically identifying, segmenting, and classifying audio streams from the natural and built environments. Part of developing this project is necessarily finding a way to unify these categories of sound for our purposes in such a way that seeks to recognize their differences as an aesthetic advantage.

A further challenge, which we have alluded to already, is that this project is a suite of creative tools. Creative process and artistic judgment add a highly variant and idiosyncratic dimension to the already complex perceptual and computational challenges. Given the ambiguous nature of both the problems being addressed and the outcomes, it is necessary to introduce a few constraints to focus the

project's scope and goals.

3.1.1 Problem Constraints

Sound Types

The range of sounds, both real and possible, is immense, especially when the boundaries of musicality and semantic salience are not imposed. In removing those constraints, we need to add others that, even if loosely, identify a subset of this immense domain that the proposed methods are intended to practically produce compelling results with. The primary way that we constrain the range of possible sounds is by duration. These are not hard constraints, but rather serve as guidelines for use.

There are three loosely-defined ranges for sounds in the *Sifting Sound* project. The *Sifter*, whose goal is to extract brief sounds from long streams, expects to work with signals at least a few minutes in length. This is when these automated methods can really begin to be helpful in augmenting manual search. The sounds output from the *Sifter* should ideally be between one and ten seconds long, brief enough that each makes a simple aesthetic statement by itself without accommodating too much variation; they should be independent, basic building blocks. These atomic sound units accumulate into the system's underlying audio database, and form the basis for the interactions in the *Observatory* and *Sketchpad* components. The *Sketchpad* also accepts audio sketches as guides for composing these underlying sounds into new collages, and these sketches should be between two and thirty seconds in duration. Those on the lower end of this range produce composites, designed sound hybrids that, like the atomic sound units, make a simple aesthetic statement. Sketches on the longer end produce collages, more akin to compositional praxis than sound design by combination.

These constraints are largely conceptual, though they have been reinforced through early experimentation while developing the project. We will also discuss empirical limitations in chapter 4, discovered by offering different types of input and applying the project's various methods, while analyzing and (where possible) measuring outcomes.

In addition to these limitations for the input and methods, we must also clarify the goals of the project. This helps us not only to guide the processes employed, but also to evaluate the quality of the outcomes and interaction. Significantly, the

project's goals are artistic, and it is understood that the methods designed may be perceptually sub-optimal. For example, the *Sifter*'s goal is not to design a model of auditory salience that maximally agrees with user judgment. Rather, the goal is to use plausible correlates of salience to augment and scale user judgment, as well as reduce and regularize the effort needed in applying it.

Additionally, the project is intended to support creative behavior roughly in real-time. This is not the real-time of performance, what Nilsson[65] calls *play time*, but of composition or *design time*. In multiple cases, this requires using approximate methods, or exact methods with subsampled or downsampled data. In the context of this project, these approximations are often also helpful for two reasons. First, they make our computations faster so as to better support more interactive and iterative workflows. Second, they allow us to make decisions with "bigger picture" data representations, and reflect the variability of making artistic decisions in general.

3.1.2 Key Contributions

This project is designed to make a few key contributions:

A software architecture and system for supporting creative work with large, expanding audio collections.

Sifting Sound integrates a number of common and less common sound-based computational tasks into a package including a database for storage and retrieval, extensive analysis and search functionality, and various data transformation methods. The resulting software system is designed to work at every level of the sound design process, as well as allow audio to be easily added and retrieved to enable workflows involving other tools.

Models and methods for swiftly and flexibly searching, relating, and combining diverse sounds.

As part of the software system, a number of computational techniques are developed and adapted to be effective in the audio domain.

Interfaces that extend and augment human creative judgment within the domain of sonic creativity, and enable it to scale to large archives, recordings, databases.

Prototype user interfaces are designed for each of the project's main modules.

These interfaces aim to facilitate and balance user input, partial automation, user creative judgment, and easy iteration and development of the user's resources and ideas.

A web-based ecosystem that combines these into a format deployable in individual and collective project settings.

The software system and methods described are part of a client-server design. The client is a browser application consisting of the user interfaces. The server is a web application that wraps the underlying software package and provides an API to its significant functionality. Together, these present an ecosystem for sound-based creative production that can be flexibly deployed in both personal and collaborative settings. Examples of each are considered in this thesis.

3.2 System Overview

The *Sifting Sound* software system is composed of two linked software projects. The first is a collection of computational tools and models, as well as additional infrastructure to support these, implemented as a Python 3 package.

The second is a web application, consisting of a client and server. The server application provides a REST API to the Python package's functionality, and the client application contains responsive user interfaces for interactively performing *Sifting Sound* tasks.

3.2.1 **siftingsound** Package

The siftingsound package is written in Python 3, chosen as an implementation language for the extensive collection of audio and data analysis tools available for it. It consists of four main modules, which are:

- **siftingsound.core**: general functionality to support all the components, distributed into multiple sub-modules
- **siftingsound.sifter**: the Sifter class
- **siftingsound.observatory**: the Embedder and Collector classes
- **siftingsound.sketchpad**: the Approximator class

The elements of the three component-specific modules (i.e. the *Sifter*, *Observatory*, and *Sketchpad*) are discussed in detail in later sections. The core module implements a number of functions and classes needed across the system, and below we enumerate and discuss the most significant of these.

There are eleven sub-modules within the core module (each representing a Python source file). Below we describe the general contents of each module. Some of the elements referred to below will be discussed in much more detail in the following section ("Underlying Models"); here we just provide an overview to illustrate package structure and organization.

- **siftingsound.core.analysis**: SoundAnalyzer and BatchAnalyzer classes, for extracting and transforming audio descriptors. The SoundAnalyzer class depends most significantly on the Essentia[14] library, a popular open-source toolkit for audio feature extraction.
- **siftingsound.core.authentication**: Two functions for managing user accounts in the siftingsound system.
- **siftingsound.core.classification**: Sound classification tool, for categorizing sounds based on their most salient aspects.
- **siftingsound.core.dataset**: Utilities for generating synthetic datasets to test package functionality with.
- **siftingsound.core.db**: Functions for managing the package's database, and for adding information to it.
- **siftingsound.core.featureimportance**: Class for computing a few high-level features from extracted descriptors, designed for use in this project.
- **siftingsound.core.loader**: Class for loading audio files into NumPy arrays.
- **siftingsound.core.math**: A few useful mathematical utility functions.
- **siftingsound.core.query**: A number of functions for building queries to the database.
- **siftingsound.core.schema**: Classes and custom types that define the structure of the database, and the methods for interaction between Python ob-

jects and database entries.

- **siftingsound.core.writer**: Class for writing NumPy arrays to audio files.

3.2.2 Web Application

Server

The *Sifting Sound* server is implemented, also in Python 3, as a Flask application and a service layer that provides easy access to the siftingsound package. It includes two modules:

- **endpoints**: Contains the Flask application, and implements a number of API endpoints for making requests.
- **service**: Implements a service layer that wraps the siftingsound package's functionality, decoupling it from the Flask application.

Client

The client application is implemented as a React.js app, in JavaScript with the JSX extension. It consists of a number of components, written as functions, and supporting utility classes.

The application also uses Redux to model some of the application state into four stores, similar to the structure of the siftingsound Python package: one general store, and three specifically for the *Sifter*, *Observatory*, and *Sketchpad*.

Below are a list of general components, with tool-specific components discussed later in each tool's sections:

- **Main**: Handles routing to login, sifter, observatory, and sketchpad pages.
- **Login**: Handles user authentication.
- **Header**: Shown at the top, passed to sifter, observatory, and sketchpad base components for further customization and rendering.

- **Home**: Displays welcome message to an authenticated user.
- **utils**:
  - **ActivationPattern**: class that "activates" sounds in the Observatory APIClient: class for making requests to the backend.
  - **dataRetrievers**: functions for getting values from redux store.
  - **eventHandlers**: custom functions for handling user interface events.
  - **AudioSampler**: class to play audio files from database.
  - **styles**: contains styled components.
  - **Tree**: class acts as a wrapper around RBush, implementing visualization features for the Observatory.
  - **useMouseEvents**: a utility to track mouse actions, movements Value: various functions to transform data.

3.3 Underlying Models

This section details the underlying tools, models, algorithms, and methods that are shared across different parts of the siftingsound package and the web application.

3.3.1 Audio Analysis

The SoundAnalyzer class extracts a number of time-varying features and summary statistics from an audio signal. These features, along with brief descriptions, are listed below grouped by time-resolution:

Envelope

The envelope computation first uses Essentia's Envelope algorithm, which rectifies the signal and applies a low-pass filter, and then resamples it to (by default) 210Hz.

Frequency-Domain Descriptors

Many of these descriptors rely on representations of the signal that we do not store or use further downstream, but compute as intermediate features (magnitude spectrum, for example). The SoundAnalyzer class extracts these frame-by-frame, by default with a frame size of 2048 samples and a hop size of 512 samples, resulting in an effective sampling rate of $\frac{44100}{512} \approx 86Hz$ (if the audio signal's sample rate is 44100Hz).

- **Pitch**
  - YIN[24] algorithm, with bounds at 200Hz and 1200Hz (and the estimation's confidence).
  - MELODIA[74] algorithm, with bounds at 200Hz and 1200Hz (and the estimation's confidence). This is computed as an alternative to YIN that focuses on predominant melody extraction.
  - Filtered pitch using Essentia's PitchFilter algorithm, computed from YIN's output (a chunkwise algorithm that removes under-confident chunks and unexpected jumps).

- **MFCC**: 13 coefficients. Essentia's default parameters correspond to the popular MFCC-FB40[31] implementation.
- **Harmonic Pitch Class Profile (or Chroma[7])**: The spectral intensities of each of the 12 pitch classes in twelve-tone equal temperament.
- **Spectral Centroid**: Centroid of the magnitude spectrum, scaled to $\frac{SampleRate}{2}$.
- **Pitch Salience**: Feature indicating "pitchness" of the signal (from 200Hz to 1200Hz), based on autocorrelation function.

- **Spectral Flux**: Euclidean distance between consecutive frames of magnitude spectrum.
- **Spectral Rolloff**: 85th percentile frequency of the energy in the magnitude spectrum (i.e. above which 15 percent of the spectrum's energy is).
- **Spectral Entropy**: Shannon entropy of the magnitude spectrum (lower entropy indicates more "peakiness" in the distribution).
- **Spectral Spread, Skewness, Kurtosis**: These measures characterize the shape of the spectral energy distribution, all computed with Essentia's DistributionShape algorithm.
- **Spectral Contrast**: Measure of contrast in each of six sub-bands, by characterizing the distribution shape in each one.
- **Dissonance**: A measure of sensory dissonance, based on a perceptual model.
- **Inharmonicity**: The divergence of the peaks in the spectrum from the nearest integer multiples of the estimated fundamental frequency.
- **Tristimulus**: Three numbers that characterize the balance of harmonics in a given spectrum.

VGGish

The popular VGGish model for audio classification, trained on AudioSet, is often used to produce compact, semantically useful representations of audio signals. The SoundAnalyzer class also uses the pre-trained VGGish model to produce 128-dimensional embeddings at around 1Hz. We opt not to use these features for most downstream tasks in this project, though the *Sifter*'s implementation includes some methods that use them, as an initial attempt. They do, however, remain available to tasks that depend on the SoundAnalyzer.

Summary Statistics

The main summary statistics computed are the mean and variance along the time-axis for each feature described under "Frequency-Domain Descriptors". We also store a fixed-length version of the signal envelope, 12 samples long, by interpolation to act as a compact shape representation for easy storage, retrieval, and comparison.

Additionally, for storage in the database, we estimate a predominant MIDI pitch note number. To do this, we first transform the frequency vector computed by the MELODIA algorithm to the appropriate scale, filter out those with a confidence value c_i below the mean μ_c, and then take the median value:

$$p = med \left\{ 69 + \log_2(\frac{f_i}{440}) \; \forall_i \in \{0, \ldots, N-1\} : c_i > \mu_c \right\} \tag{3.1}$$

This feature may not be useful in many cases, and in fact several features relating to pitch may be unhelpful in processing and evaluating sounds that lack a clear pitch. We need some way to understand whether the pitch of a sound, as estimated by algorithms we use, is a relevant descriptor for a given sound.

The pitch salience function described previously was found, by trial-and-error, not to be indicative of the author's perceptual evaluation of "pitchness" in a collection of sounds chosen as test candidates. To generalize (and complicate) this further, we review some simple engineered summary features that produce importance scores for the feature categories of pitch, harmony, timbre, and signal envelope.

3.3.2 TEMP Model: Feature Selection and Classification

For the purposes of this project, there are four perceptual aspects of a sound we evaluate to estimate their importance. These, along with methods for computing importance scores, are below. The methods described are derived largely by trial and error, using a small dataset with elements selected to represent diverse positions on these axes by listening.

- **Pitch**: The pitch importance score is computed by measuring the portion ($\in [0, 1]$) of filtered pitch values that are non-zero.

- **Harmony**: The harmony importance score is based on the entropy H of the time-averaged chroma vector Ch: $(1 - \frac{H(Ch)}{3}) * (1 - P)$ where P denotes the pitch importance score.
- **Timbre**: The timbre importance score is $\left((\sigma_C^2 * (\frac{C}{2})^2) + (\frac{\mu_S}{3 \times 10^7})\right) * (1 - P)$ where σ_C^2 represents the variance of the spectral centroid, μ_S represents the mean spectral spread, and *P* denotes the pitch importance score.
- **Envelope**: The envelope importance score is given by $\frac{\tilde{\mu}_3 + \tilde{\mu}_4}{40}$, where $\tilde{\mu}_3$ and $\tilde{\mu}_4$ respectively denote the skew and kurtosis of the normalized envelope. The output is clipped to $[0, 1]$.

The TEMP model refers to a simple typology of sounds, designed for the *Sifting Sound* project. These basic types are listed here, along with features selected manually for each type.

- **Timbre-Centric Sounds**: MFCC, Spectral Centroid, Spectral Rolloff, Spectral Flux, Spectral Entropy, Spectral Spread, Spectral Skewness, Spectral Kurtosis, Inharmonicity, Dissonance.
- **Envelope-Centric Sounds**: Envelope, Spectral Centroid, Spectral Rolloff, Spectral Entropy
- **Monophonic Pitch-Centric Sounds**: Pitch Corrected, Pitch Confidence, Pitch Salience, Chroma, MFCC
- **Polyphonic Pitch-Centric Sounds**: Chroma, MFCC, Spectral Centroid, Spectral Spread

Note that these are not intended to be mutually-exclusive categories. We have generated a small dataset of a little below 500 sounds by labeling each one as belonging to one or more of these four categories.

With the feature importance scores as inputs, and these four classes as labels, we can formulate a multi-label classification task. The classifier trained as part of this project is a collection of linear Support Vector Machines (SVMs), through scikit-learn's *MultiOutputClassifier* interface. Although this classifier is not currently used in downstream tasks, it sets the stage for future TEMP-based decision-

making in addition to the methods that already use importance scores to compute feature weights.

3.3.3 Data Model

The data model for *Sifting Sound* can be reduced to two categories: user data, and sound data. Implemented in PostgreSQL, the database schema is more complex in structure, but also more easily interpretable and efficiently queryable. The two predominant data object categories are distributed into tables and columns through normalization, relation, and vertical partitioning. The main database tables, with general descriptions, are listed here:

- **audio**: Main table for sounds.
- **audio_embeddings**: Table for 2D embeddings computed from feature summaries of sounds.
- **audio_features**: Table for storing computed audio features.
- **audio_collections**: Table for audio collections.
- **long_audio**: Per-user table for Sifter source audio and features.
- **users**: Table for user accounts.

3.3.4 Web API

Listed below are the web API endpoints, to which requests from a client application can be made, as well as brief descriptions of what they are used for.

- **/login**: User authentication (from form).
- **/sounds**: List of all sounds.
- **/upload**: Upload a new sound.
- **/sound/<int:id>**: Get sound by ID.
- **/upload/<int:id>**: Directly upload a sound.
- **/sifter/set**: Set Sifter source file for user.

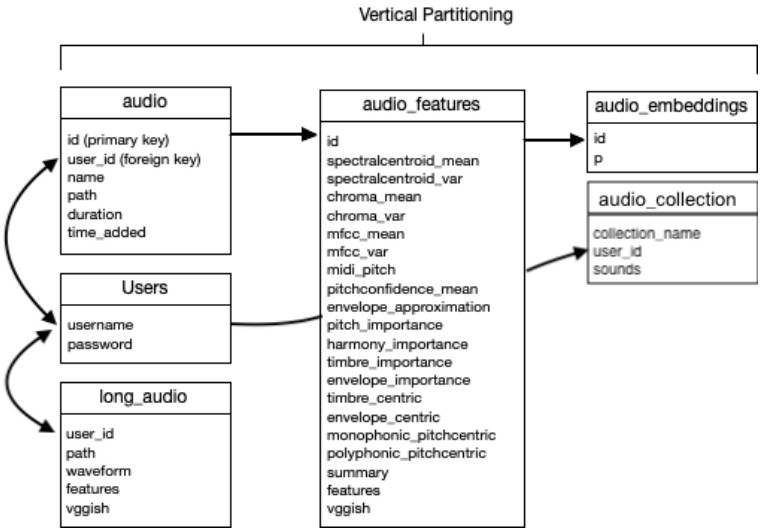


Figure 3.1: The main database tables with their main fields.

- **/sifter/check**: Check if Sifter source file is already the given file (by filename).
- **/audio/<string:filename>**: Get sound by filename, supporting partial content requests.
- **/waveform/<string:username>**: Get waveform binary data for user's Sifter source.
- **/sifter/query/<string:method>** Query Sifter using either envelope-based or feature-based method.
- **/sifter/auto/<string:method>** Automatically get Sifter results using either envelope-based or feature-based method.
- **/sifter/store**: Store a given segment of the Sifter source as an independent sound. Also adds it to the database.
- **/observatory/numsounds**: Get number of sounds.
- **/observatory/embeddings**: Get list of all sound embeddings (2D points).

- **/observatory/collect/<string:method>**: Collect sounds based on the given method and reference sound.
- **/observatory/store**: Store sound collection in the database
- **/sketchpad/approximate/<string:method>**: Evaluate a user's sketch and get the result.

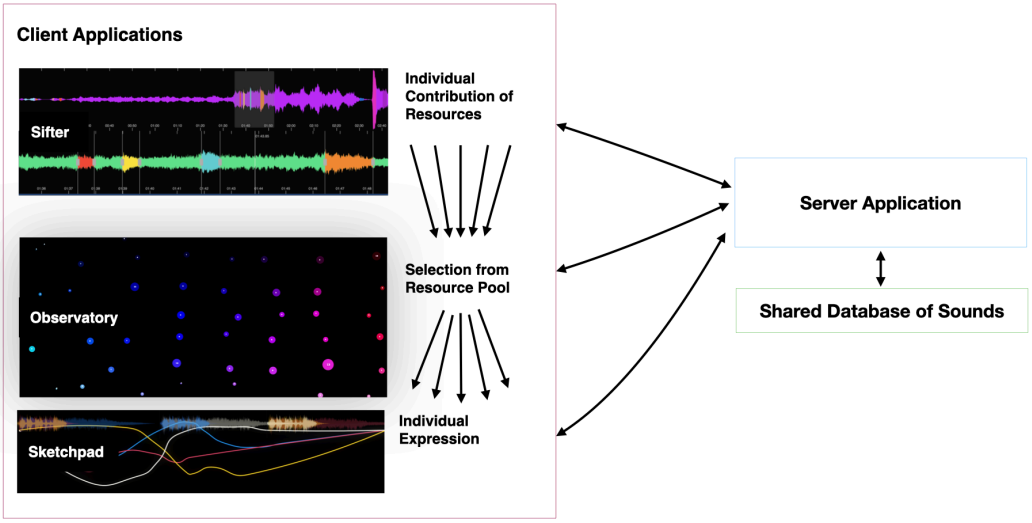


Figure 3.2: *Sifting Sound*'s client-server model.

3.4 The Sifter

The goal of the *Sifter* module is to search long audio files for interesting segments and retrieve these, in a way that supplements manual user exploration. It consists of some computational tools, as well as a user interface that allows interaction with the system.

The *Sifter*'s search methods are detailed later in this section, in two categories: the first deals with those methods that operate directly on the signal of the source, without needing a reference (automatic methods). The second deals with those that depend on the user supplying a query sound, which can be done either by a

user-located segment of the long audio file being searched (henceforth referred to as the source), or an external sound (query-based methods).

3.4.1 Program Structure

Each of the three project components (the *Sifter*, the *Observatory*, and the *Sketchpad*) include software elements both in the siftingsound package, as well as in the web application. Below is an overview of those that comprise the *Sifter* in both:

- siftingsound Package
  - **Sifter**: Class that handles all Sifter functionality. Analyzes sources and queries (with SoundAnalyzer), finds segments automatically or by query, and produces output that can be sent to the client, or directly used to retrieve segments.

- Web Application
  - **Sifter**: Component that wraps BBC's Peaks.js library[29] and handles the visualization and interaction for the main dual waveform display by adding custom event handlers, data store methods, and more.
  - **SifterBottom**: Component that holds the bottom part of the Sifter module, mainly the user controls.
  - **Waveform**: Component for drawing target audio waveforms.

3.4.2 Automatic Methods

There are two distinct techniques used for automatically finding salient segments, in the *Sifter*. The first relies on the audio signal envelope (computed as described earlier), and the second uses a combination of several other features.

Envelope

Here we review Spread Locally-Adjusted Maxima Search (SLAMS), a computationally simple and fast algorithm designed for this project, and modeled on manual waveform-based search. Intuitively, we wish to select the largest peaks in the signal envelope that are at least some distance apart, and then locally find change-points in either direction around these to form segments to retrieve.

The input parameters are:

- n: number of segments to find.
- s: the spread factor.
- w: the window length.
- d: the duration scale.

The first part of the algorithm deals with selecting n envelope indices. N.B. Δ below denotes the (first order) finite forward difference.

Algorithm 1 Spread Locally-Adjusted Maxima Search (Part 1)

Require: $n > 0$ and $0 \leq s \leq 1$
Ensure: $min(\Delta sorted(selection)) \geq mdist$
  $indices \leftarrow argsort(-envelope)$
  $selection \leftarrow \{indices_0\}$
  $mdist \leftarrow len(envelope) \div \frac{n}{s}$
  for all $idx \in \{indices_1, indices_2, \ldots\}$ **do**
    if $min\{|idx - s| \; \forall s \in selection\} \geq mdist$ **then**
      $selection \leftarrow selection \cup \{idx\}$
    end if
    if $|selection| \geq n$ **then**
      break
    end if
  end for

The second part deals with adjusting the indices in both directions, to local maxima of the envelope's difference given the w and d parameters. $audio$ below represents the original audio, from which the envelope is computed. Note that we write $X[a : b]$, for example, to denote the subset of values of X with indices $\{x \in \mathbb{Z} : a \leq x < b\}$. Default values for each are $a = 0, b = |X|$.

Algorithm 2 Spread Locally-Adjusted Maxima Search (Part 2)

$segments \leftarrow \{\}$
$r \leftarrow \dfrac{len(audio)}{len(envelope)}$
for all $s \in selection$ **do**
  $lb \leftarrow s - w$
  $rb \leftarrow s + \lfloor w * ds \rfloor$
  $sd \leftarrow diff(envelope[lb : rb])$
  $onset \leftarrow (argmax(sd[: w]) + lb) * r$
  $offset \leftarrow (argmax(sd[w :]) + w + lb) * r$
  $segments \leftarrow segments \cup \{audio[onset : offset]\}$
end for

This algorithm quickly produces a collection of segments that are expected to stand out, from different parts of the signal. Even if these sounds are not directly used, they can serve to seed ideas from the user, perform subsequent queries, direct attention to different parts of the signal, and offer a sampling of the kinds of content available. While it can be optimized further or made more approximate, in practice this is generally unnecessary for performance reasons due to its simplicity and the lower-resolution envelope signal that it works with.

Features

The algorithm that uses extracted audio features is considerably more complex, and looks for segments that are somehow anomalous in their pattern. To do this, we employ the Matrix Profile, along with a number of constraints and additional steps to adapt it to the domain of audio segment selection.

This process contains several steps and is somewhat implementation-specific in comparison with SLAMS, and so we review the major steps involved, including more detail where necessary.

1. Standardize (i.e. z-normalize) all the features extracted from the audio.
2. Reduce data to four dimensions with Principal Components Analysis (PCA).
3. Resample the signal envelope so that it is the same length as one of these components, and invert in the range $[0, 1]$.

4. Compute the approximate pan-MatrixProfile (by using the SKIMP algorithm)[55] with windows w for each of the four components, to get distance profiles and indices.

5. Sum the distance profiles across components for each window, discarding the indices.

6. Apply the inverted signal envelope computed in **3** as an annotation vector.[23] Intuitively, we can interpret this as favoring segments with a larger amplitude in our anomaly detection. Computationally, we realize this with distance profile d and annotation vector av as $d + (1 - av * max(d))$.

7. $argsort$ the inverse of each distance profile, discarding infinite values.

8. We then follow a similar search procedure to SLAMS, seeking the indices of the largest n values that preserve a minimum mutual difference $mdist = len(feature\_vector) \div \frac{n}{s}$.

9. The resulting indices are multidimensional indices (i, j), where i corresponds to a time-index, and $w_j \in w$ corresponds to a window-length.

10. The audio segment bounds for each result (i, j) are $i * R$ and $(i + w_j) * R$ where R is the hop size used in extracting features.

The goal of this procedure is to find maximally novel segments, with a few possible durations. It performs a much more comprehensive search, and so is slower; by default, the window lengths of 1, 1.75, 2.5, and 3.25 seconds are used. While maximal novelty may not necessarily predict utility for the user, it can again, in a different way, produce options to begin the process of query-based search and additional manual exploration. An important step is the application of the annotation vector, a technique for domain adaptation which, in this case, allows us to favor novel segments with larger amplitudes. This helps to avoid trivial novelty (for example, a signal for which silence may be unusual), and again serves as a simple correlate of salience that subtly guides the model.

3.4.3 Query-Based Methods

Similarly to the automatic methods, there are two query-based methods, and one each for working with the envelope or the extracted features.

Envelope

To retrieve segments related to a query sound, we use Mueen's Algorithm for Similarity Search (MASS)[62]. By default, this algorithm computes Euclidean distance under z-normalization, allowing us to retrieve segments with a similar shape very quickly, despite differences in amplitude. To retrieve n segments, we get the distance profile using MASS, and then get the top-n results (with an exclusion zone set to the length of the query segment).

The query may be a segment of the source audio, chosen manually or based on a suggestion from one of the two automatic methods. It may also be an external sound, in reference to which the source file is searched. To illustrate a simple example, let us imagine that we wish to collect percussive samples from a long audio recording. One way to immediately do this would be to supply a drum sample as a query sound, and review the results of the corresponding query. Since the envelope characterizes the sound of a drum well, this is likely to produce useful results. The earliest experiments with this technique were used for precisely such a purpose, with favorable results.

Features

For the query-based feature segment search, we adapt measures from both the query-based envelope method, and the automatic features-based method. Specifically, this process consists of the following steps (the first two steps are the same as the automatic features-based method):

1. Standardize (i.e. z-normalize) all the features extracted from the audio.
2. Reduce both source and query data to four dimensions with Principal Components Analysis (PCA).
3. Compute absolute Euclidean distance between the query and sliding query-sized windows of the source, for each principal component. We use the

MASS FFT trick to do this with $\mathcal{O}(kn \log n)$ time-complexity, where $k = 4$ is the number of principal components.

4. Sum these distance profiles along the time axis.
5. Retrieve the top n results using MASS's top-k algorithm with the exclusion zone parameter set to the length of the query signal.
6. Scale the indices by R, the hop size used for feature extraction.

This method performs a more broad search of the source content, by using multiple features with dimensionality reduction. This is expected to be especially helpful for cases in which timbre and pitch content are more useful for comparison than envelope. While the envelope-based method handles the E in the TEMP model, this features-based method handles the TMP. Using feature importance scores can help to weigh some features more or less based on the query.

3.4.4 User Interface

The *Sifter*'s user interface consists of a few key elements, most notably a dual waveform based on the Peaks.js library. Peaks.js is a web-based interface element for performing the type of manual editing described earlier, in the context of DAW applications. The top view, or overview, provides a visual representation of the entire file, while the bottom view allows more detail for a selected section of the audio, for manually marking onsets and offsets for segments. Additionally, the interface includes two custom waveform displays for displaying targets and results respectively, as well as a collection of buttons for performing search related actions.

The target and result waveform views also show the onset and offset times of identified segments. To add a target, the user selects the appropriate segment on the zoom view visual representation, and the segment's information is automatically added to the application's centralized state tree, which in turn causes the UI to update the number of targets (and the visual representation). A prototype can be seen, for reference, in fig. 3.3.

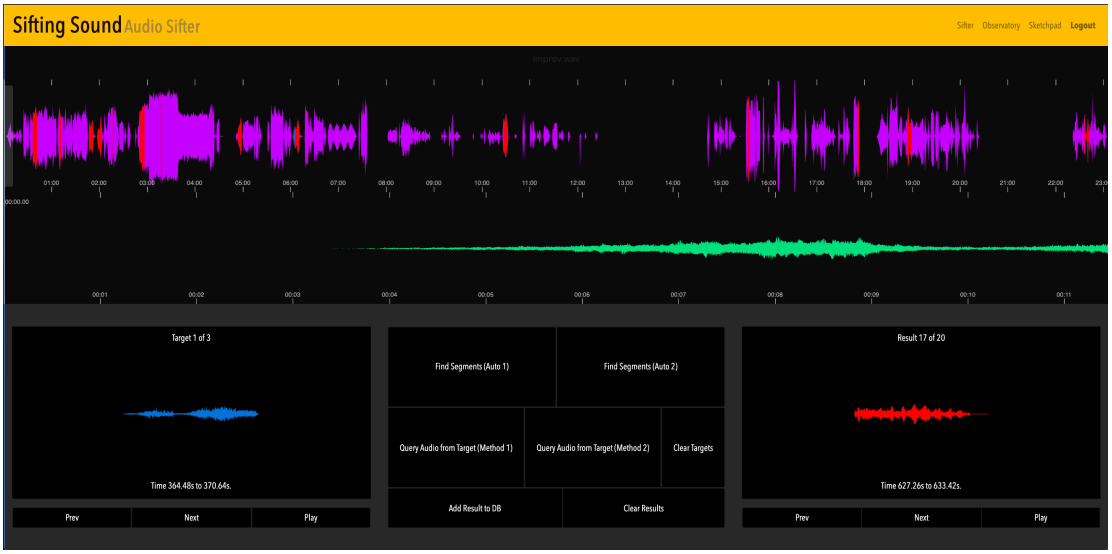


Figure 3.3: A prototype of the *Sifter* interface.

3.5 The Observatory

The *Observatory* is for exploring and forming subcollections from sounds in the shared database. Similarly to the *Sifter*, it provides tools for doing this both automatically and based on queries. The goal of the *Observatory* is to be able to assemble sound collections that are diverse, but whose components are still related.

3.5.1 Program Structure

- siftingsound Package
  - **Embedder**: Class that computes low-dimensional representations of audio features, and stores the object that performs these transformations for embedding new content later.
  - **Collector**: Class that looks for sounds in the database automatically or by query and collects these into groups.

- Web Application
  - **Scatterplot**: Main component that handles the data and controls.
  - **Points**: Draws the visual representation based on data from the Scatterplot.
  - **SoundPopover**: Adds an overlay describing a sound or group of sounds.

3.5.2 Sound Representation

The Embedder class uses UMAP to compute low-dimensional representations of sounds, from summarized feature statistic vectors. The SoundAnalyzer class, described earlier, computes the mean and the variance for each of 48 features, resulting in 96-dimensional vectors for sounds. These are transformed in a large batch into a two-dimensional representation, and the transformer is stored so that it can appropriately process new samples until a new batch embedding is performed.

UMAP is chosen here for its sensitivity to both global and local distances in the high-dimensional feature space. Storing the data transformer object allows the space of sounds to grow, and visual representations of them to account for such growth. Significantly, it does this so as to preserve the familiarity and consistency of the space. Periodically re-processing all data can help to recognize and present new overall structural relations and patterns. This can be done automatically, for example through the specification of thresholds beforehand, or by direct user action.

3.5.3 Building Collections

As noted earlier, the Collector class employs methods for both automatically selecting sounds, as well as grouping based on a query sound, or other query criteria as specified.

Query-Based Methods

- **Similar**: Get n closest sounds in the low-dimensional space.
- **Related**: Independently get $\frac{n}{D}$ closest sounds along each of the D embedding dimensions, and combine them together.
- **Pitch**: Get n closest sounds by predominant MIDI pitch (chromatic version).
- **PitchOct**: Get n closest sounds by predominant MIDI pitch, with the same pitch-class as well.
- **Duration**: Get n closest sounds by duration.
- **Nearest**: Multi-feature method that uses feature summaries to get the n closest sounds in a high-dimensional space. More intensive.
- **Shape**: Get n closest sounds by comparing resampled envelopes.
- **VGGish**: Get n closest sounds by comparing the first VGGish frame.

This variety of methods can be variously mixed and matched, to produce collections through a combination of manual search and selection in a visual space, and content-based search combined with user evaluation. As with the *Sifter*, the goal is to augment the user's manual collection effort while offering control over both the input and the outcomes as well. The goal is to accommodate both quick iteration and incremental development of sound collections.

Automatic Methods

For the automatic methods, we simply choose a sound already embedded at random, and then perform the corresponding query-based method from the chosen sound. The purpose of this is primarily to, as in the case of the *Sifter*'s automatic methods, offer suggestions to begin the process of collection. It offers an impulse, whose response is carried out through successive queries and manual search, as previously noted.

3.5.4 User Interface

This interface is considerably more complex, since it deals with representing a large number of data points, and providing ways to explore and collect subsets. The main portion of the interface is a scatterplot-like two-dimensional chart of sound groups, inspired by similar interfaces like *Constellation*, described in chapter 1. To accommodate a large number of sounds in the visualization, the interface uses the RBush[5] library which implements a version of the R-tree data structure, for spatial indexing, and D3.js[15] for transformation and visualization of data.

Building on this, a grid is automatically imposed on the screen, and the points in each grid box are collected together into a group. The size of the grid is easily user-adjustable, and clicking on a group opens a secondary view which is a scatterplot-like representation of only the points in that group. Since each point represents a sound, and the point positions are computed from sounds' audio features, this reflects a perceptually grouped map of the sonic space. Users can then go through and select sounds manually, through search functions, or a combination of both methods. Search and filtration methods are provided at the bottom of the interface. The current prototype is shown in fig. 3.4.

Figure 3.4: A prototype of the *Observatory* interface.

Each group is sized and labeled according to the number of sounds in it. A user can hover over a group to play a random sound from it, and click on it to expand its contents to the secondary view (shown in fig. 3.5). In this view, the user similarly hovers over sounds to play them, and clicks to both select them for querying and add them to a working collection, that can later be stored.

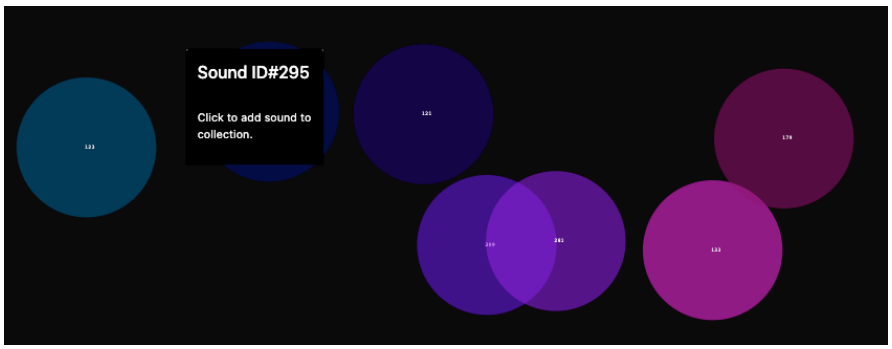


Figure 3.5: The secondary view showing sounds in a group.

As the user accumulates sounds into a collection through manual searching and selection, as well as through the provided automated tools, they accumulate in the application state tree. The current prototype implements a simple modal view that recalls the collection's contents, and allows them to be examined and stored to a collection in the system database. Fig. 3.6 shows this management view.

3.6 The Sketchpad

The *Sketchpad* combines two input modalities for gesturally creating expressive musical collages from sound. The first is aural; users can specify a sonic target, whether vocalized, performed on an instrument, or otherwise. This target specifies an initial "sound world", establishing a general context and trajectory for the collage. The second is graphical; users can draw patterns that take on roles as parameters, and inform both the trajectory of the collage, as well as that of audio processing applied to it to create its final output form.

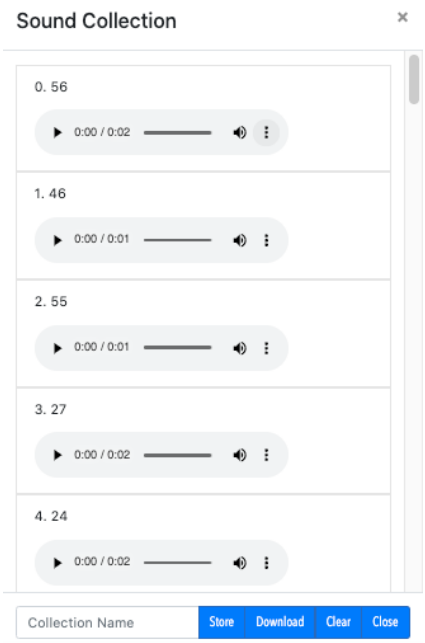


Figure 3.6: The collection management view.

3.6.1 Program Structure

- siftingsound Package
  - **Approximator**: Handles all aspects of the Sketchpad. Preprocesses and resamples parameter paths, analyzes audio (using SoundAnalyzer), retrieves candidate sounds from the database, performs the optimization to build the sound collage, and renders the audio.

- Web Application
  - **Sketchpad**: Main component that implements the frame and controls.
  - **Lines**: Component that renders parameter paths, based on data from Sketchpad.

3.6.2 Aural and Visual Sketching

The *Sketchpad* system relies on user-provided "sketches" to shape and combine individual sounds into composite collages. For our purposes, the term *sketch* is generalized to the multimodal setting, and functions somewhat differently than goal-directed freehand drawing.

We mean *sketch* to denote two distinct but associated structures. The first is the aural sketch, which is parameter-free and thereby similar in nature to freehand drawing sketches. The goal of the aural sketch is to supply a general morphological nucleus for the collage, around which sounds are assembled, compared, and combined.

The second is the visual sketch, which perhaps ironically is constrained and distributed into discrete entities. These entities are parameter paths, which operate essentially as arbitrary functions of time that can be assigned to audio selection metrics or processing routines.

These parameter paths are conceptually numerous, though the current implementation specifically provides for four:

- Energy
- Density
- Variance
- Weight

3.6.3 Forming Collages

The *Sketchpad*'s current implementation offers two different methods for producing a sound collage from user input. After a number of preprocessing steps, a sketch template and a set of options are available. The first method works to sparsely approximate the sketch template from the options, using a greedy algorithm inspired by matching pursuit. The second method solves a continuous optimization problem, minimizing the distance between an arrangement of available options and the template sketch.

Sound Selection, Option Enumeration

A preliminary preprocessing step is to retrieve a subset of available sounds to construct options from. In the case of collections supplied by the user, as assembled in the *Observatory*, this may be unnecessary. In other cases, we analyze the sketch and produce content-based database queries to retrieve related sounds. This is done using *Observatory* methods with input from the *Sketchpad*'s analysis. For example, the retrieve_sounds_by_pitch function produces a collection from database sound entries with a predominant MIDI pitch note number matching one computed from the sketch. In the case that this produces fewer than 10 results, a subsequent query by pitch class (mod 12) is performed. This reflects both the commonality of octave errors in pitch estimation algorithms, and the octave-invariant perceptual similarity of different tones. Similarly, we make available other methods to produce subcollections when not explicitly provided by the user.

Once we've obtained a computationally feasible number of sounds, we transform these into options. To do this we retrieve a feature matrix $\mathbf{F}_s$ for each sound, where $\mathbf{F}_s \in \mathbb{R}^{k \times n}$ (k = number of features, n = number of frames), and produce multiple versions of each where $len(sound) < len(sketch)$. Feature matrices are shifted in time, in increments of $\frac{len(sketch)}{10}$, and zero-padded on both sides so that they have the same number of frames as the sketch features. We maintain a list of sound sources and offsets with corresponding indices. While this often results in a large number of options, it offers two benefits: first, it allows us to deal with the sketch without segmenting it beforehand. Second, it replaces the inclusion of a time-offset variable (to be optimized) with time-shifted versions of sounds at a limited time-resolution, which allows multiple instances of a sound and simplifies the algorithms needed for the audio arrangement process.

Control Parameters

We specify and implement two forms of control parameters in the *Sifting Sound* project. The first of these is feature controls, which influence the template-matching process. These are implemented as a dictionary that maps the parameter name to a set containing one or more audio feature names. To apply these, we replace the associated audio feature vectors, in the template, with the provided control path interpolated and resampled appropriately. The second type is processor controls, which apply additional post-processing to the audio signal resulting from the template-matching. These are implemented as a dictionary mapping

a parameter name to a function that takes, as its arguments, the audio signal and the control parameter path, and returns the processed audio signal.

The current implementation includes four control parameters, as was noted earlier: Density, Variance, and Weight are feature controls, and Energy is a processor control. The names are chosen for their perceptual significance, but are ultimately arbitrary and can be renamed as needed. Specifically, the mappings are:

- **Density**: Spectral Spread, Dissonance
  - This acts as a control for the textural complexity of the output, with larger values favoring more timbrally diffuse and dissonant combinations of sounds.

- **Variance**: Spectral Flux
  - This parameter is subtle, but controls the amount of spectral change as a function of time. Ultimately, extracting more change-related features from the audio signal can allow for more dynamic parameter mappings.

- **Weight**: Spectral Centroid
  - The weight parameter controls the output spectrum's center of mass over time.

- **Energy** = $f(x, l) = x \cdot l^2$
  - This parameter applies a gain envelope to the assembled audio, giving it dynamic shape over time.

Feature Weights

By default, all audio features used in the template-matching process are given equal preference. However, just as we define ways to select subsets of the sound database deemed most relevant to the target sketch, we may also want to select a subset of features identified as most relevant. The dimensionality of the features doesn't cause either of the template-matching problems to be particularly complex, and so instead we compute weights for features that approximate their importance, using the *FeatureScorer* class discussed earlier in this thesis. This gives us importance scores for four categories of features, those related to each of timbre, envelope, pitch, and harmony.

The *Sketchpad* currently does not factor the computed envelope vector into its feature template. This is for multiple reasons, including most importantly that the envelope is chosen as a visually expressed parameter, but also that the envelope is computed at a different resolution from other features, that some of its characteristics are expected to be reflected in other features, and that it has less to do with the *Sketchpad*'s goals. Given the other three importance scores S, we apply the softmax function so that $\sum_{x \in S} = 1$, and then assign the resulting weights to features as shown below (features not listed here are assigned weight 0):

- **Timbre**: MFCC, Spectral Centroid, Spectral Rolloff, Spectral Flux, Spectral Entropy, Spectral Spread, Spectral Skewness, Spectral Kurtosis, Inharmonicity, Dissonance, Tristimulus
- **Pitch**: Pitch (Corrected), Pitch Confidence (MELODIA+YIN), Pitch Salience
- **Harmony**: Chroma

Forming Collages: Sparse Method

The first method, which results in collages with a relatively sparse texture and is inspired by matching pursuit, is a simple greedy algorithm. In each iteration, it seeks to find and add one sound option to the arrangement that most helps it more closely match the template. Or, we seek the option $\mathbf{X}$ that minimizes, with template $\mathbf{T}$, template-scaling constant c (defaults to 10), number of features n, arrangement $\mathbf{A}$, and weight-vector $\mathbf{w}$:

$$\sum_{i=0}^{n} ||c\mathbf{T}_i - (\mathbf{A} + \mathbf{X}_i)||_2 \mathbf{w}_i \tag{3.2}$$

Forming Collages: Dense Method

The second method results in sound collages with a much denser texture, with a relatively large number of sound instances. It is also generally slower, and should ideally be used with a small, carefully curated collection of sounds. For this method, we formulate and solve an optimization problem using simulated annealing. Simulated annealing is a probabilistic method for optimization, especially in the presence of many parameters (in our case, sounds). It draws from statistical mechanics, specifically the process of annealing (i.e. heating and then cooling a material to reduce defects) in solids, and adapts a Monte Carlo method to search by random sampling, which helps avoid local minima. This technique is helpful here because of the large numbers of options, uncertainty in the search space's characteristics, and its ability to provide a good solution. The *Sketchpad* currently uses a simple implementation provided by the mlrose[42] package. The initial version tried to select any number of sound options whose features, summed, are closest in distribution to the template's features, measured by Kullback-Leibler divergence[51].

The current implementation transforms this into a continuous optimization problem, in which the selection variable s actually reflects amplitude per sound $\in [0, 1]$. We also introduce another parameter, ρ or density. In assembling audio after optimization, sounds assigned an amplitude below $1 - \rho$ are not included, so this parameter acts as a kind of filter, controlling the textural complexity of the output as a whole (as opposed to the time-varying sketched density). This may also result in collages with more nuance, since sounds can be mixed at different possible amplitudes. For N options, n features, continuous selection variable $\mathbf{s} \subseteq [0, 1]$ indicates which options to combine. With option set L, template $\mathbf{T}$, constant c, and feature-weight vector $\mathbf{w}$, it tries to minimize:

$$\sum_{i=0}^{n} D_{KL} \left(\sum_{j=0}^{N} \mathbf{s}_j L_{ji} \parallel c\mathbf{T}_i \right) \mathbf{w}_i \tag{3.3}$$

Where L_{ji} denotes feature number i of option number j. By default, this method limits itself to a maximum of 200 iterations. Additionally, one optional preprocessing step is to initialize the selection vector with the sparse method's output selection vector, plus a small amount of noise, This can act as a kind of initial guide for the optimization process, and was found in practice to sometimes lead to quicker or better results, btu also has a different interpretation: the first, sparse, method is deterministic (i.e. given the same data, its behavior will be the same), and so the second method can in this case build around the outcome of the first method if the user finds the texture insufficiently rich or full.

Both methods estimate the features of mixtures by linearly combining the features of options. This method is efficient, but has its limitations when accuracy is desirable.[35] Some features are better estimated when combined linearly, while others rely on the initial sound selection and particular aspects (zero regions in the corrected pitch features, for example).

3.6.4 User Interface

The *Sketchpad*'s interface's primary function is to provide a visual input modality, in the form of drawing lines. User gestures are transformed into lines with some smoothing, and rendered with the Konva library. Most of the screen is given to representing the sketch and audio sketch waveform. Line selection and rendering functionality is given at the bottom, just as parameter selection and action initiators are in the other interfaces. A textual input allows the user to specify an audio collection to build with, by name (e.g. previously formed with the *Observatory*). This interface is the simplest one so far, designed to provide a comfortable, clear, and responsive sketching platform. A prototype version showing the lines, sketch waveform, and controls can be seen in fig. 3.7.

Figure 3.7: A prototype of the *Sketchpad* interface.

Similarly to the *Observatory*, a modal view handles additional functionality in this interface. In this case, that includes previewing and downloading the results of the collage-forming process, as well as downloading the user-supplied sketch audio to preserve it. This is seen in fig. 3.8.

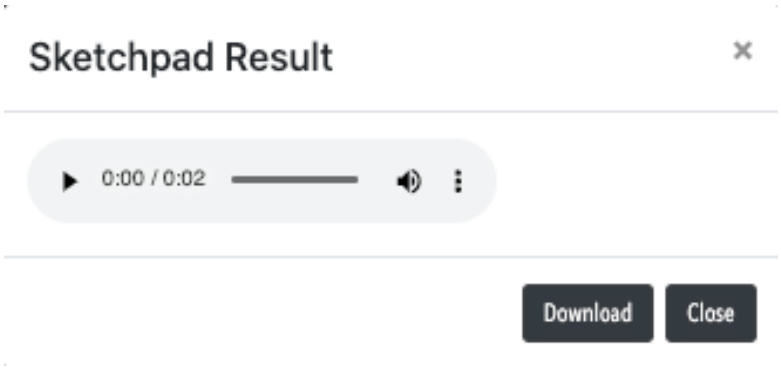


Figure 3.8: The *Sketchpad*'s modal view.

4

Discussion and Project Evaluation

Even within the prominently ambiguous and variable spaces of creative practice and artistic production, a great deal of work has been undertaken to study, evaluate, and define aspects of artifacts, tools, processes, and discourse produced in the field. While we may not expect to firmly rate this project's practical utility or quantitatively describe the quality of its results, this past work can, when combined with identifying and describing the project's goals, help to frame its methods and outcomes within a context of application, as well as design future evaluations that better reflect its efficacy in real-world scenarios.

4.1 Project Goals

Conceptually, *Sifting Sound* is a suite of connected tools that together establish an ecosystem for creative composition and production with large audio collections. While each module has a different domain and output form, they share a set of common goals with regard to how they support the creative process. Here, we investigate some of these goal areas in more detail, and analyze the project's strategies in each case.

4.1.1 Exploratory

Margaret Boden writes that exploratory creativity "involves the generation of novel ideas by the exploration of structured conceptual spaces."[11] Exploration is the primary interest of the *Sifter* and *Observatory* modules, which provide visual and computational representations of latent structure in sound entities (long audio files and large collections respectively), and a number of methods modeled on human search and perceptual novelty to facilitate new discoveries and relations, as well as user evaluation and incremental modification of these discovered patterns.

Considering these modules, in their simplest form, as search tools, we can draw comparisons to other forms of search used as creativity support. Bill Kules notes:

> An architect looking for "seed" ideas for a new project may search an architecture database. Novelists, journalists and artists may similarly search the web for new ideas. A historian will explore archival material for a research project.[50]

While multiple sound databases exist, along with methods for searching them, Kules also writes about exploratory search patterns:

> Searchers interactively explore the search results by browsing, filtering or other techniques. They issue multiple, successive queries as their information need evolves, possibly across multiple sessions. They exhibit "berry-picking" or "information foraging" behavior as they collect useful bits of information, follow promising threads and identify new information sources.

We can compare this to the process followed in the *voicecoil* project, described in chapter 2: an incremental, multi-session process whose requirements are ambiguous and changing. The *Sifter*'s methods facilitate specific and changing queries, queries built from query results or segments of interest identified by hand, and support persistent (i.e. multi-session) interaction. The *Observatory* similarly supports query, albeit as a method of imposing structure and forming collections from an organized space of possible sounds. It also supports persistent interaction by storing collections, and allowing them to be updated over time.

4.1.2 Generative

One way that the *Sifter* is a generative tool, though it does not generate sound material in the sense of synthesis, is that it uses search as a means to generate options. However, another way the *Sifter* generates creative options is without a query, by relying on models of human search processes. These creative options also double as query options which can be used to develop and retrieve more resources. As such, the search methods act as a kind of bridge between generative and exploratory modes of support.

The *Observatory* shares this generative character, although it begins with defined creative resources in a large shared pool. The outcomes are a little more defined as well: the goal is to produce collections that share some perceptual quality, and this can similarly be done with or without a query. The relational aspects of building collections that share meaningful attributes supports another goal area of this project, specifically one assumed by the *Sketchpad*.

4.1.3 Combinatorial

Combining objects or ideas into novel forms is sometimes referred to as combinatorial creativity[12]. The *Sketchpad* supports this kind of process by assembling sound collages from aural and visual gestural idea representations, and optionally sound compilations produced with the *Observatory*. It allows the exploration of recombinant expressive potential in a moldable and parametric way.

Earlier in this thesis, we established that the *Sketchpad* preserves the structure of each sound, and also that it performs its assembly by trying to minimize some kind of measure of difference between user input and the output. Treating sounds as atomic units is likely to place limitations on how low the difference measure can practically go, though we may consider this a feature. In addition to preserving individual sounds as they were contributed and curated by users, the difference measure can act as a measure of surprise, once a suitable minimum is found or a collage arrangement is constructed. Surprise is extensively documented as having a role in creative processes, and as a response to creative products[13][54]. It is also hoped that, here, it encourages continued engagement, contribution, and sharing, both with these tools and other ones.

4.1.4 Spontaneous

All three modules' interfaces are designed to support interactive, iterative workflows, and so an important corollary of this is that their computational techniques need to be able to deliver results quickly. This is reflected in the use of simple and approximate approaches throughout the system. We might expect that latency introduced in either search or assembly may deter exploratory user behavior, limiting the possibilities for spontaneous discovery and creativity.

Another layer of spontaneity is the shared database of sounds, to which the outputs of the *Sifter* flow. The *Observatory* supports exploring this shared database and discovering connections between sounds contributed by different users. Additionally, the *Sketchpad* allows these materials to flow into users' independent expressions, combining them and manifesting discovered connections.

4.2 System Performance

As discussed throughout this thesis, a significant goal of this project is to provide interactive methods. This requires that the processes designed and used perform quickly and scalably. Dan Olsen's goal of flexibility is another way of stating this:

> A UI tool is flexible if it is possible to make rapid design changes that can then be evaluated by users.[66]

In this section, we assess the performance of a few key techniques and algorithms developed in this work. Based on this analysis, we also discuss the roles different techniques can play in a user scenario, and limitations with regard to both quantity of content and latency in the system's response. Here we aren't evaluating the quality of the results, just how quickly they might be able to be returned in an interactive computing environment, on a consumer laptop.

4.2.1 Sifter

The *Sifter* has four primary methods, described in chapter 3 in detail. In this section, we'll group them by the signals they work with, rather than whether or not they make use of queries. For the envelope-based methods, we assess the time taken to produce 10 results from audio signals with durations ranging from 10

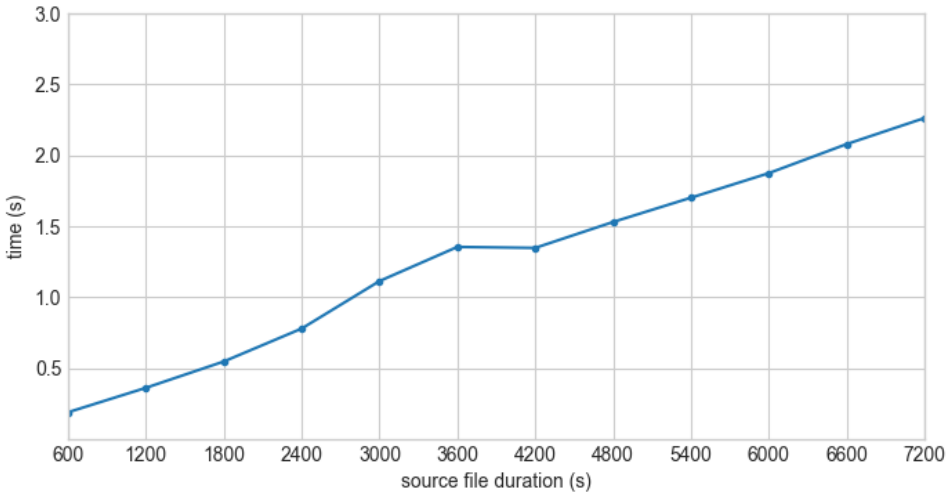


Figure 4.1: Performance of the query-based envelope search for segments.

minutes to 2 hours, in increments of 10 minutes. To begin with, let's look at results from the query-based process with a 1-second long query (fig. 4.1).

Even with a 2-hour audio file, it takes less than 2.5 seconds to retrieve 10 results. This doesn't include the initial time needed to load and analyze the audio file, which can take a few seconds but only generally needs to be done once per file. Queries depend on the data either being stored in memory, or being retrieved from the *Sifting Sound* database.

To connect these measurements to the question at hand, of whether they reflect interactivity support, we can compare them to perceptually-established time constants that establish response time limits. Card et al. describe 1 second as the perceptual processing time constant[18], which Nielsen describes as a limit for uninterrupted flow of thought, for the user[64]. In many cases, the expected response time for this process will be below this particular threshold.

A different time limit established is the unit task time constant, which is 10 seconds. Nielsen notes that for longer response times, users will want to perform other tasks while waiting for results. This is perhaps a more effective threshold to assume in the case of the *Sifter*'s methods; each request can be treated as a task, the user waits for results and then performs subsequent requests as needed.

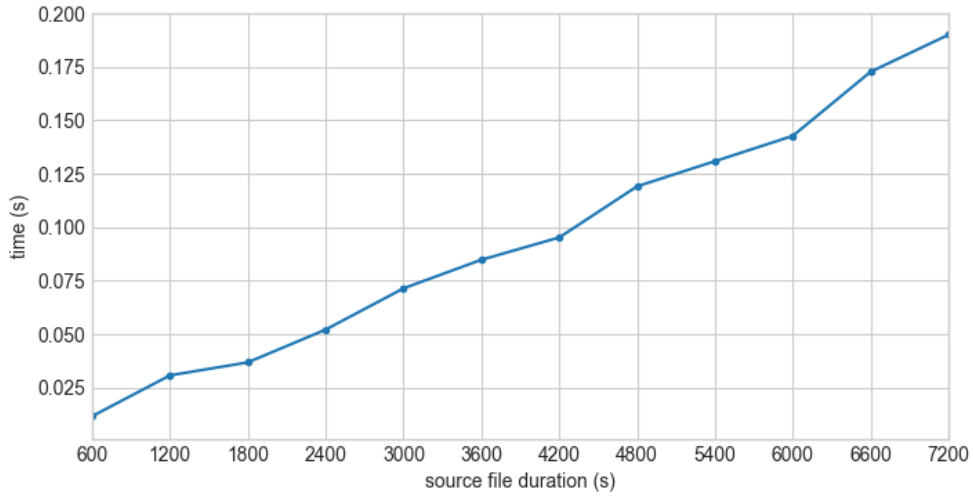
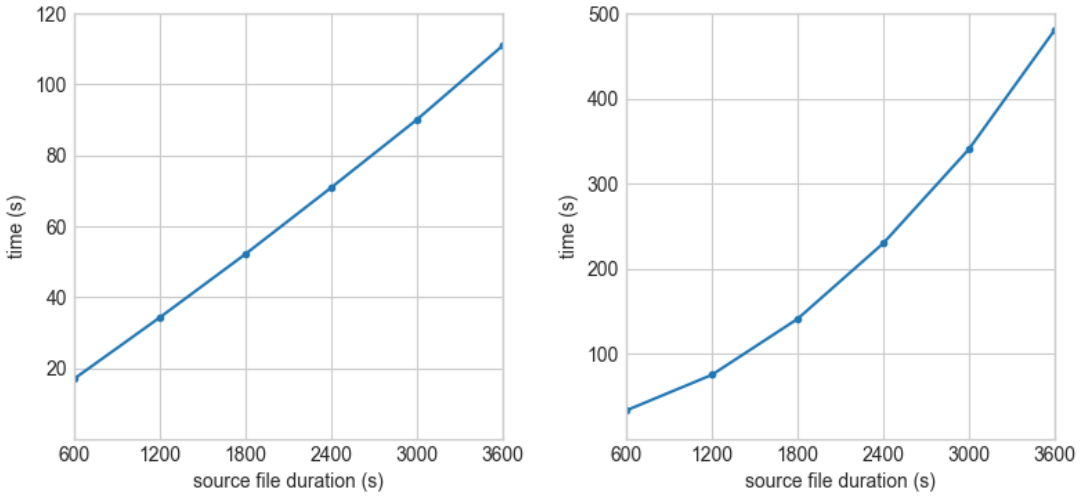


Figure 4.2: Performance of the automatic envelope search for segments.

Second, we can take a look at the performance of the automatic method based on the signal envelope (fig. 4.2). In this case, response times are even quicker, with the 2-hour audio source resulting in a response time under one fifth of a second. Similarly, we might expect that these times alone support interactive use well. These measurements do not factor in other times, such as those involved in loading and analyzing the audio files, or in retrieving data from the database, since these vary situationally and may not always be applicable (for example, when data is kept in memory after a previous request). In some cases, requests will need extra time to accomplish tasks.

The query-based and automatic methods based on multidimensional audio features take considerably longer, both because of the dimensionality of the feature space and the complexity of the models, despite preprocessing steps and approximate search methods used. This makes them less suitable for interactive use as described previously (see fig. 4.3). Placing constraints on these two techniques, especially the automatic method (e.g. number of windows) can considerably improve performance.

The time taken is expected to still be, however, much lower than manual search methods. We might expect that users will want to perform other tasks while waiting for results to be computed. The user interface's use of asynchronous requests and centralized application state supports this well, so that the user is free



(a) Query-based method.

(b) Automatic method.

Figure 4.3: Performance of both the feature-based approaches.

to perform other tasks, even in other modules of the interface, while awaiting results from a *Sifter* request. When results are returned, they are added to the application's state tree and the UI is updated to display them as needed. Furthermore, with server-deployed versions, more powerful computing infrastructure may yield faster results as well.

4.2.2 Sketchpad

The *Sketchpad* employs *Observatory*-like selection methods to produce a feasible pool of sound options from which to choose and assemble a collage with. Part of the reason to do this is the sensitivity of the chosen computational approaches to the amount of data they need to process. Earlier we discussed the interactive style of sketching-as-a-design-process, and as such it is important to facilitate such a back-and-forth, iterative workflow. While the number of sounds is only one performance variable (the duration of the sketch, the number of sound options generated from the sounds, and algorithm-specific parameters are other significant such variables), it may help to give us a general sense of the *Sketchpad*'s response times.

First, let's take a look at the sparse, greedy algorithm, with its default parametriza-

tion (between 10 and 40 sound selections) with a 6-second sketch, and between 50 and 500 unique sounds to transform into incrementally time-shifted options, and select from (fig. 4.4).

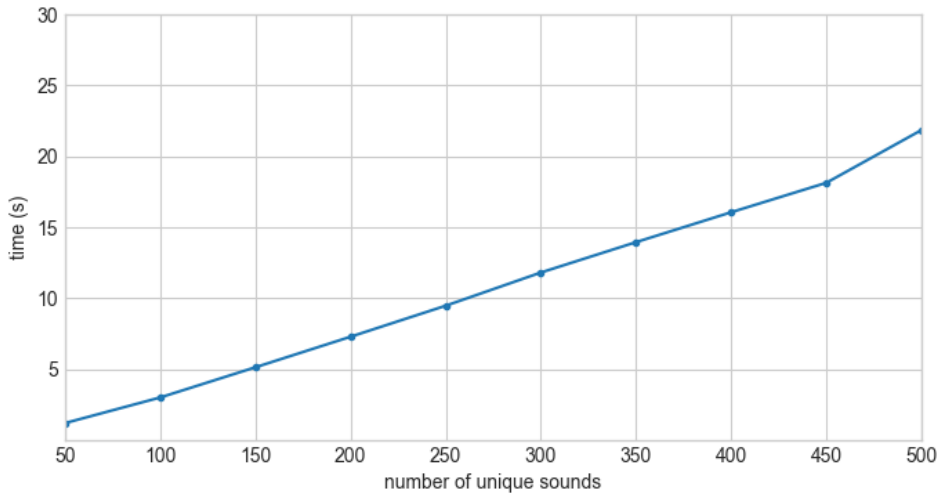


Figure 4.4: *Sketchpad* performance: method 1.

By default, the *Sketchpad* is expected to pre-select ≤ 100 unique sounds to work with, which, in this instance, takes much less time to assemble than the 10 seconds noted earlier. This also includes the time taken to synthesize the output audio, by loading all the files and combining them together, as well as normalizing the mix. The second, dense method, based on simulated annealing, performs much slower and its performance is more difficult to characterize, given how variant it can be. Generally, as it is parameterized by default, it expects between a fourth and a tenth of the amount of data as the first method, for roughly equivalent performance. Consequently, it might either be used with small collections for interactive sketching, or with larger collections for detailed "offline" rendering.

While, as mentioned, it is possible to assemble collages from more sounds, and require the user to wait and/or perform other tasks while waiting, the low response times also do support interactive, iterative sketching to develop the user's compositional ideas in context. Further experiments are needed to determine the extent to which this conclusion can be supported in practical user scenarios, given the additional latency introduced by other parts of the system and interac-

tion.

4.3 Planned Studies

To characterize the practical efficacy of the software environment's techniques and interfaces in supporting its stated goals, it may be helpful to collect some data. In this section, we identify two types of data collection expected to yield useful information. The first involves delivering a hosted prototype to a small number of users, and prescribing some very general task directions along with a survey that addresses how the specific designs and methods of the system relate to both the task areas, and the intended goals. The other type deals with collecting anonymized data from practical use of the system as deployed to a larger number of users, with a few areas of relevance specifically noted.

4.3.1 User Testing

A major challenge of applying standard user testing strategies in this ambiguous problem domain is the lack of standardized tasks. Additionally, this project is both positioned in a specialized domain, and also intended to be accessible to a wide range of users. Finally, it is expected that a user will spend some time with the tools, learning their behaviors and thereby learning to use them expressively, which is difficult to contain within standard user testing situations. Olsen identifies these as assumptions often made in usability testing, and remarks that they are not often met.

Instead, we can prescribe general, albeit module-specific, patterns of use, and offer some demonstrative materials to each user. Materials might involve long audio files stored on a server, that can be automatically loaded for each user, in the case of the *Sifter*. They might also include example collections and sketches in the case of the *Observatory* and *Sketchpad*. The suggested patterns of applying these materials might be in the form of statements. For example:

- Find 20 interesting segments in this audio file.
- Make a collection of 40 sounds with the pitch Eb.
- Create a 10-second long collage of unpitched sounds that gradually gets louder.

- Make a 12-second long “drone” collage, centered around middle C.

These instructions require some background in audio and/or music to interpret, but leave room for participants with a basic level of experience, at an intermediate level, and those highly proficient with audio production tools and who have strong musical backgrounds. An effective study design might include a balanced sample of participants with different levels of self-reported domain expertise.

To collect data that reflects users’ experiences with the system, we can ask a range of questions that address different aspects of their experiences. One area of interest is aesthetic value, for which we might ask respondents to rate the quality of the *Sifter*’s results as compared with the source’s background audio, or to describe the artistic quality of the *Sketchpad*’s rendering of their sketches.

Related to this is expressive leverage: the ways in which user intent is transformed, and a comparatively small amount of effort from the user can yield results that typically require more work. For this, we might ask users to rate how related the *Sketchpad*’s outputs are to their sketch materials, over the course of a number of sketches. We might also ask how well the *Observatory* collects sounds that are related to their queries, or are mutually related in the case of query-less collecting, in the user’s estimation. Other elements of interest include system performance and response times, interface design and layout, visualization strategies, and usefulness across various source materials.

Together, these answers along with coding and analysis can help to determine which aspects of the system perform desirably given the contexts of use, the user’s expertise, and the materials the system works with. This can help to further refine the design and implementation of the modules, and prepare them for deployment to a larger population of users, leading us to another form of data collection that uses a larger sample size.

4.3.2 Logging and Automated Collection

With a larger number of users, we may be able to observe overall trends in how and when different elements of the system are used, and perhaps even how they prompt or discourage various forms of engagement with the system. This may help to determine how well the tools are supporting user creative production. Some background information may still be helpful to collect, such as user domain expertise and goals, and these can be collected when setting up user ac-

counts. Beyond this, no biographical or personal identifying information is necessary.

Example types of user actions that can easily be logged might include:

- How often each module's different methods are selected.
- How long searches take to complete.
- The durations of material processed in the *Sifter* and *Sketchpad*.
- The proportions of pitched vs. unpitched sounds chosen and used by users.
- Audio feature summaries of content used and made by users.
- The time spent and number of requests made per module.

4.4 Application Scenarios

In this section, we consider two possible scenarios in which the *Sifting Sound* system might realistically be deployed.

4.4.1 Personal Library

A common pattern for building a sound sample library is to record long-form performances, improvisations, explorations, or other acoustic phenomena, and then edit these long recordings to produce sample collections. These collections often have some consistency of character, or even an underlying theme in some cases. It is also common to assemble samples into reusable composites, by sequencing and layering.

The *Sifting Sound* project provides infrastructure to construct an arbitrary number of such collections by augmenting the search patterns, automating the collection patterns in part, and allowing quick and iterative sound compositing. It also maintains a database of sounds, collections, sources, and analysis data, making new samples, libraries, and combinations trivial to produce, reuse, and modify as necessary. It bridges these tasks, manages the data organization automatically, and provides visual interfaces and computational methods to manage the scale

of these resources as they grow and change over time. Since the database system also includes user accounts, it is possible to host it on a server, and manage access to collaborators as well.

4.4.2 Collaborative Project

The *City Symphonies*[Cit 86] projects, developed by composer Tod Machover, are characterized by community participation and large amounts of sound. These projects involve a large number of audio recordings, both made by a core team, as well as submitted by community members, and also often involve sound-based interfaces through which participants play with and shape sounds, both for their own expression and as part of the ultimate symphony, in which these sounds are united with music written for acoustic instruments (typically orchestras).

Sounds recorded for these projects often are initially long field recordings, ranging from a few minutes to hours long. These usually need to be extensively edited to produce a collection of sound resources, and these are often categorized by source or destination. These sounds are then compiled into instruments, transformed into materials for the final piece by combination and processing, and also often added to applications for public interaction and participant expression. *Sifting Sound* is designed to partially automate these processes, scale them to a much greater amount of content, provide a flexible foundation on which to build custom versions for projects, and involve public participation at every stage: from recording, finding interesting and salient sounds of a place, forming interesting collections of these, and transforming them expressively through gesture and experimentation by participants.

4.5 Practical Demonstration

In this section, we walk through the process of extracting a number of audio samples from long audio files, combining these with pre-prepared samples, forming sound collections both manually and with help from automation, and finally producing sketches that combine these into new structures. This serves as a demonstration of the processes involved, and how they might work in a real-world setting.

4.5.1 Materials

We begin with two types of sonic materials. The first is a collection of field recordings made in Chennai, India in January 2020. The selected recordings vary in duration but total a little under 2 hours. The second is an exploratory improvisation, largely consisting of pitched material with a range of different timbres from synthesis tools. This is a little over 20 minutes long, and so while the total amount of content isn't massive, it serves as a good source for an initial demonstration.

To augment these, a number of sounds that are manually pre-segmented, including some collections described earlier in this thesis, are introduced into the database to act as a kind of working collection on which these new sounds build, in a way that one might expect within a personal scenario, as outlined previously. Subsets and combinations of these will be applied to forming collections and producing sketches.

4.5.2 Sifting

As a first step, we wish to extract interesting segments from the long, unstructured audio sources. Beginning with the field recordings, this was done through a combination of methods. First, the automatic envelope-based method was used to quickly identify a number of candidate segments. Through manual review, many of these were found to be immediately useful. After this, brief percussive excerpts were searched for with a snare drum sample as a query by envelope, again producing a number of useful samples. Finally, the automatic features-based method was tried, to see if it produced any segments markedly different from those seen before. A few of these were selected, and the resulting collection, in total, consists of 468 samples. These samples have a median duration of about 2 seconds, and a total duration of 14 minutes and 30 seconds. They range in duration from 0.4s to 3.7s.

With the second source, a similar process was followed but substituting a features-based query-driven search for the envelope-based version, with a synth chord sample as the query. The output collection, after some filtering by hand, consists of 144 samples. The total duration in this case is 4 minutes and 10 seconds, with a median sample duration of 1.5s, and individual samples ranging between 0.15s and 3.6s. More significantly, the character of this collection is very different from the first: most segments contain some pitched or tonal material, and have

a very synthetic quality given their original source. These samples also needed a little more filtering than the first collection, possibly because of the relatively large amount of silence. Even so, a majority of produced options were ultimately selected, and the process took only a few minutes.

4.5.3 Collection

All 612 samples were placed into a new *Sifting Sound* database instance. As noted earlier, a number of additional sounds were then added to replicate a situation in which the database already contained sounds (for example, from previous sifting sessions). A batch embedding is performed to produce scalable 2D coordinates for sounds, and then we begin the process of collection.

The following four collections were produced. Two of these, each with 150 sounds, were made by querying the database with an example sound and retrieving results through a combination of available methods (by pitch, duration, etc.). The other two were produced manually, in the sense that one was pre-formed, and the other was formed as part of one of the *Sifter* sessions mentioned before. The four collections are:

Name	Count	Brief Description
kr1	84	Collection of string instrument samples, taken from the Kronos Quartet recording (see chapter 2).
improv1	144	Collection of tonal synthesizer-based improvisation segments, excerpted with the *Sifter*.
snarebased1	150	Collection of samples found by various query methods from a snare drum sample sound.
chordbased1	150	Similarly, a collection formed from a synth chord sample sound.

4.5.4 Sketching

Forming these collections helps to identify smaller groups of sounds that both share some meaningful identity for composition, and also provide a feasible number of options to attempt the *Sketchpad*'s methods with. A total of 12 different collages were produced, from 3 different sketches. One of these is a simple pink noise signal, while the other two are vocalizations: one produced by whistling, and the other is a voiceless palato-alveolar fricative (ʃ). The parameter paths are

held constant at 1 for these sketches, to allow the differences in the methods and other parameters to come through more clearly.

In the next section, we investigate the outputs of the *Sketchpad* given inputs and parameters, and analyze both how the collages present and how they change. This also offers some insight into important aspects of the system's behavior, and provides ideas for further refinement and development.

4.6 Audio Examples

4.6.1 Sketch 1 and 2

See Appendix A for media links.

METHOD1_STRINGSTOSKETCH1.WAV

This file was made by using the *Sketchpad*'s first (sparse) method for forming collages, given the kr1 (strings) sample collection, with sketch1.aif. The output is unpitched, and consists of discrete events that together approximate the noise-like timbre of the sketch. The collage largely represents the general qualities of the sketch, not taking into account precise envelope characteristics.

METHOD2_STRINGSTOSKETCH1.WAV

This is a version of the previous sketch, produced with the second (dense) method. Texturally, this is more dense because it has a larger number of sounds mixed together at various amplitudes. A consequence of this is a greater sense of surface smoothness, despite the many discrete sound events heard. Additionally, there is a greater diversity of content introduced in this collage.

METHOD1_STRINGSTOSKETCH2.WAV

Sketch 2 contains a pitched whistle, and this collage reflects that in its prominently pitched elements. These don't all have the same pitch as the sketch, but do reflect a greater degree of proximity than choosing unpitched samples would, or those further away in pitch or variation of pitch. Generally, the collage represents

the drone-like quality of the sketch, introducing some additional textural complexity and layering, resulting in a more full, spectrally broadband result.

METHOD2_STRINGSTOSKETCH2.WAV

This second method is again even more full in texture, accommodating many diverse elements but focusing them around the pitched, whistle-like central character of the original sketch. Notably, the second method here begins with the first method's selections and alters them.

4.6.2 Pink Noise

These examples use a 2 second pink noise sample as a sketch, providing a way to characterize the output and examine how it varies. See Appendix B for media links.

METHOD1_SNAREBASEDTONOISE.WAV

This example uses samples from the snarebased1 collection, many of which have a brief, percussive character. Given the limited resolution of approximation, as well as the greedy approach, the result is a number of discrete events of spectrally similar character to the sketch. This offers a useful idea for future use: the offset increment (i.e. 1/10th of the sketch duration by default) can be a parameter, with smaller increments causing sketches to take longer to produce but allowing cases such as this one to produce smoother results as desired. This is likely a particularly useful parameter for method 1, since it performs relatively quickly. Another approach, however, is to simply use method 2's density to fill this result out more, and better reflect the texturally dense sketch audio.

METHOD2_SNAREBASEDTONOISE.WAV

This version again begins from the output of the first, but forms a denser representation of the sketch by combining more layers together, with more flexibility in proportion. The result has more of the surface qualities of the noise sketch, but still maintains some of the specific characteristics of the first version.

METHOD1_TONALFRAGMENTSTONOISE.WAV

This collage uses the same pink noise signal as a sketch, but uses the improv1 collection of sounds. As noted before, these sounds are brief tonal fragments, produced with synthesizers. At first listen, this sounds quite different from pink noise, given the character of the samples. Many instances of brief melodic and percussive sound are heard, constituting a similar problem to the previous sketch example with method 1. In previous instances, method 2 partially mitigated this effect by beginning from method 1 and adding more elements, as well as modifying selected ones. We can observe how method 2 performs this time in more detail, by examining its output both with and without using method 1 to begin with, and with different values for ρ (i.e. the density parameter).

METHOD2_TONALFRAGMENTSTONOISE

The two files with the _with_initialstate tag were made beginning with the output of method 1 as an initial collage state to modify. One of these, with ρ set to 0.1, sounds largely the same as the output of method 1. However, when ρ is set to 0.2, the collage begins to take on the character we've heard before in comparing the methods: more elements, with a smoother textural surface as a result. Notably, the density was set to 0.4 by default for the others.

For comparison, examples of method 2's output without the initial selection state, at various values of ρ, are also provided. With $\rho = 0.01$, independent elements are still heard to some degree, but the texture is more broadband and noise-like than with the initial selection state. With $\rho = 0.4$, the texture is even more dense and broadband, and with $\rho = 1$, it sounds as though many instances are combined to the effect of obscuring their individual character. This effect may be desirable in some cases, but generally, the difference from using method 1 as an initial state suggests that in some cases, it may be preferable not to do so, and that this can as well be transformed into a parameter so that the user can make this decision, based on creative context.

4.7 Audiovisual Example

Here we review a simple example of a sketch with its audio source and drawn parameters, connect these inputs to the output, and then modify a parameter, comparing the output by observation and analysis. The goal is to illustrate the

interactive properties and processes of working with the *Sketchpad* interface, and hint at its expressive possibilities. See Appendix C for audio links.

4.7.1 Initial Sketch

The inputs for the initial sketch were sketch_av.wav, recorded in the *Sketchpad* interface running in the browser, and the parameter paths shown in fig. 4.5, subsequently drawn over the audio waveform.

The audio sketch in this case is a vocalization, beginning with a velar ejective and unvoiced fricative (k'x) and transitioning to near silence, followed by an unvoiced palato-alveolar fricative (ʃ). The unvoiced consonants mean that, like Sketch 1 in the previous section, there are no identifiable pitched components.

Figure 4.5: Parameters for initial sketch.

Listening to sketchoutput_av1.wav, we can hear a representation of this unpitched character in the sounds assembled, as well as general features of the sketch's trajectory: an initial accumulation of sound, followed by a slight reduction in density and even amplitude, and then more activity again but with a different spectral character.

What we are actually hearing is a combination of the audio sketch approximation with the visual parameter paths: the envelope is ultimately controlled by the blue

Energy line, and so the amplitude variation of the original sketch is only preserved to an extent. Especially with regard to the finer variations, i.e. those with a smaller time-scale, the broad and slow change in the drawn Energy line largely overrides these fluctuations, and the result is a compromise between aspects of each. This parameter path offers an example of the purpose of audiovisual sketching: the audio sketch provides aesthetic context and structure, and the parameters offer control for iterative design.

4.7.2 Variation 1 and Comparison

The control offered by the parameter paths is apparent with regard to the envelope, since we can make a comparison to the envelope of the sketch audio source. The other parameters have more complex mappings and abstract interpretations, and so here we will modify one in particular, Weight, and then observe the change in the result.

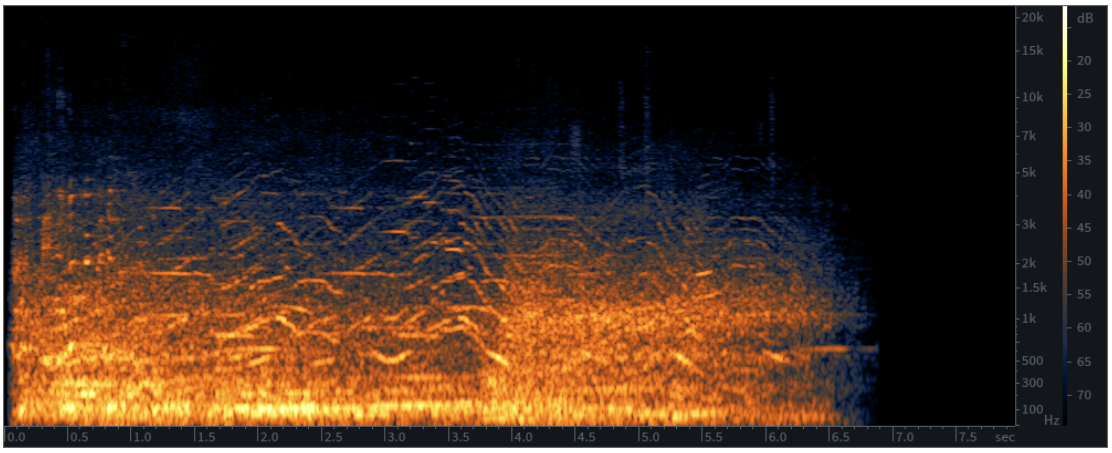


Figure 4.6: Mel spectrogram for sketchoutput_av1.wav.

To start with, we can review the spectrum of the initial sketch's result, sketchoutput_av1.wav, shown in fig. 4.6. We can see that the general distribution of energy in the spectrogram is similar to that of the audio sketch. The envelope parameter path is sloped downward here, but perhaps too gently to see clearly in the spectral visualization. A waveform display (see fig. 4.7) better reflects this trend in the amplitude.

A variation on this sketch is found in sketchoutput_av2.wav. To produce a variation, we simply return to the *Sketchpad* interface after listening to the output,

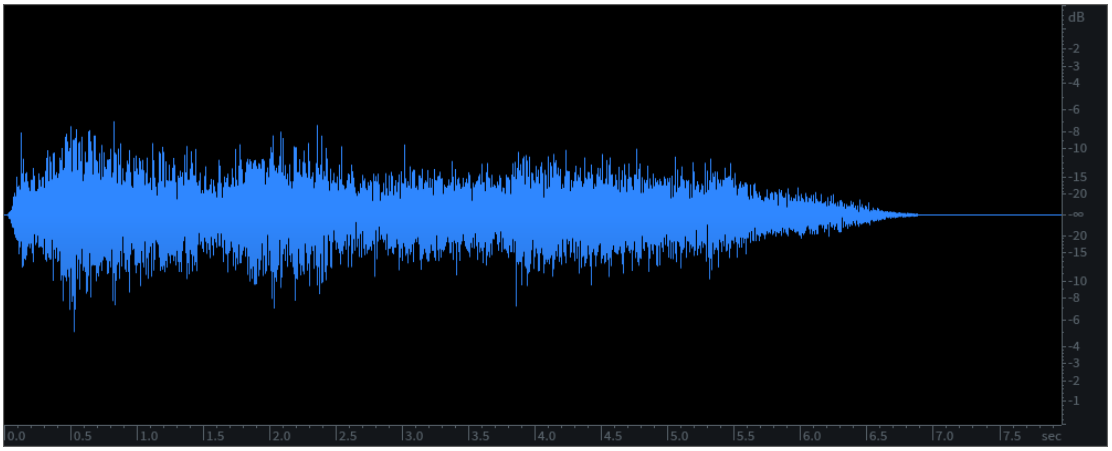


Figure 4.7: Audio waveform for sketchoutput_av1.wav.

and select and alter any parameters as desired. Let's vary the two parameters discussed so far, i.e. Energy and Weight. Fig. 4.8 shows this new version.

Since we've already observed the action of the Energy parameter, a relatively small variation is introduced here to demonstrate its sensitivity in an interactive context. Broadly, a more significant decrease in the amplitude of the output is prescribed, and we can both hear in the rendering and see in the waveform representation (in fig. 4.9) that a corresponding change appears in the output.

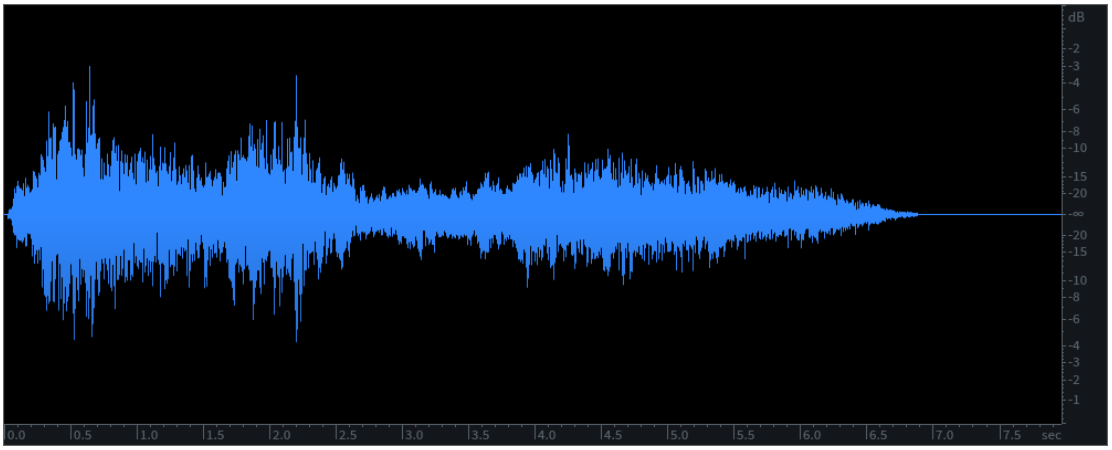


Figure 4.9: Audio waveform for sketchoutput_av2.wav.

Conversely, the Weight parameter is significantly modified. The general trajectory

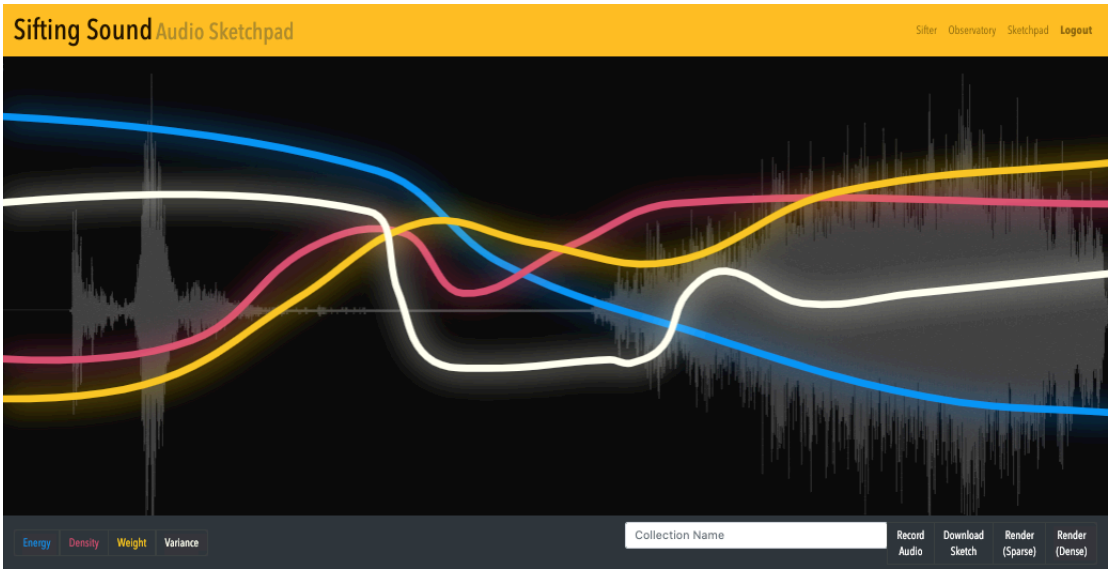


Figure 4.8: Parameters for sketch variation.

is roughly inverted, with a peak followed by a dip in the middle. It's worth remembering before we look at the spectrum that the Weight parameter is mapped to the spectral centroid feature, and a lower vertical position corresponds to more weight. This is done to provide a natural mapping from an interface perspective, given real world associations with weights of objects.

Listening to the new version, we can observe the similarity in content, but difference in character. While prominent aspects of the "what" are held constant, the "how" has changed, and we hear a much lighter middle section, and ending section as well. A glance at the spectrogram (see fig. 4.10) confirms this; we can see less low-frequency energy in both the middle and ending sections of the output.

For the middle, the low signal level in the audio sketch combined with the peak in the drawn Weight parameter explains the significant lack of low frequency energy. Additionally, corresponding to what we noted about the Weight parameter's path, we observe a small return of some low frequency energy immediately following this.

Most significantly, upon listening, this version of this sketch sounds like a variation of the first. This is a significant goal of these parameters. They offer the user a design-oriented sense of control over the qualities of the composition, without

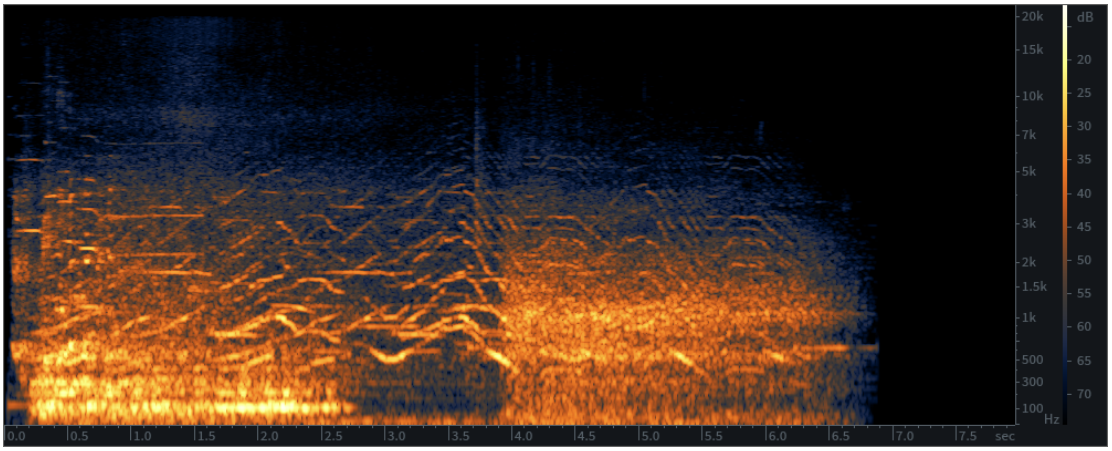


Figure 4.10: Mel spectrogram for sketchoutput_av2.wav.

replacing the audio input's role of being a reference according to which the composition is constructed.

4.8 Interface Walkthrough

This section details the contents of an interface walkthrough video, in which we explore the interactive properties of the system, and demonstrate their use. It supplements the previous examples by showing how each of the interfaces work, as well as how content flows through them and is connected to the user experience.

For the full video, under 2 minutes in duration, see Appendix D. In the video, several frames are annotated to reflect the actions being undertaken at specific moments. We review a selection of these below, with time-stamps representing positions in the video they are taken from, for reference. This serves to provide further commentary and context supporting each action demonstrated.

Figure 4.11: 00:08.7: Target selection.

In fig. 4.11, we see the basic structure of the *Sifter* interface, with a target selection in blue. These selections are adjustable, and are made by clicking on the zoomed waveform view, and dragging handles attached to the ends of the segment. The buttons at the bottom center of the interface provide methods to query based on the currently-selected segment.

Figure 4.12: 00:16.5: Result review.

Next, we see the results of running an initial envelope-based query (i.e. method

1). The results are shown in red (fig. 4.12), and the currently-selected result can be added to the database, if desired, using the bottom-left button.

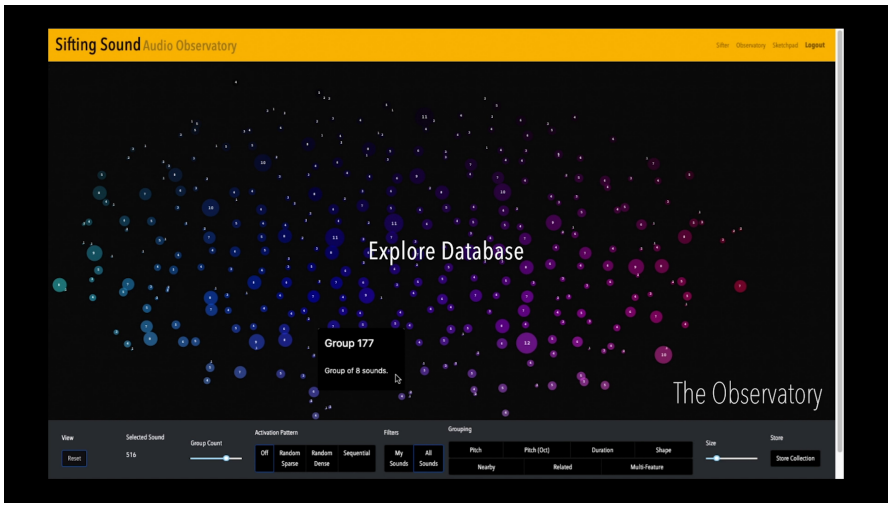


Figure 4.13: 00:35.4: Database exploration.

For the *Observatory*, the default view is a grouped exploration of the sound database, shown in fig. 4.13. Hovering over groups provides a basic, stochastic navigation of the space of sounds, which can be used to find areas of interest.

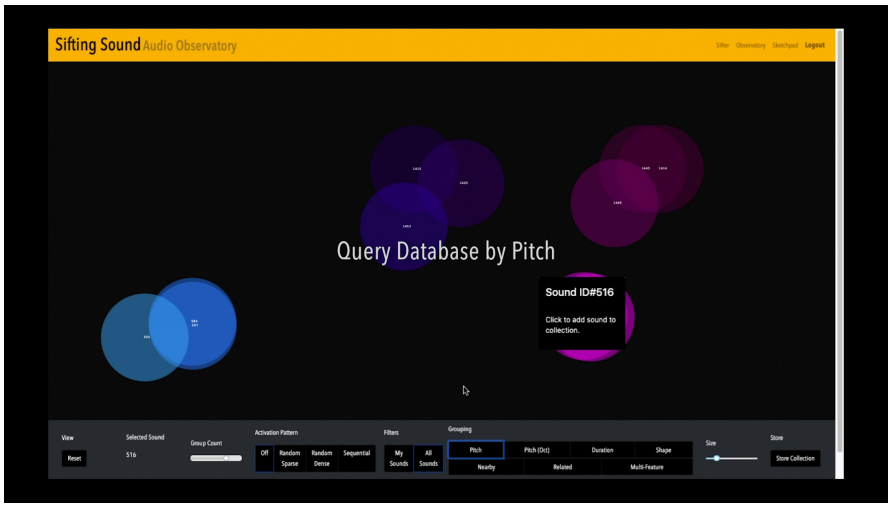


Figure 4.14: 00:42.6: Database query from a selected sound.

We expand a group by clicking, select a sound from within it, and then query the database for sounds nearest in pitch to the selected sound (fig. 4.14).

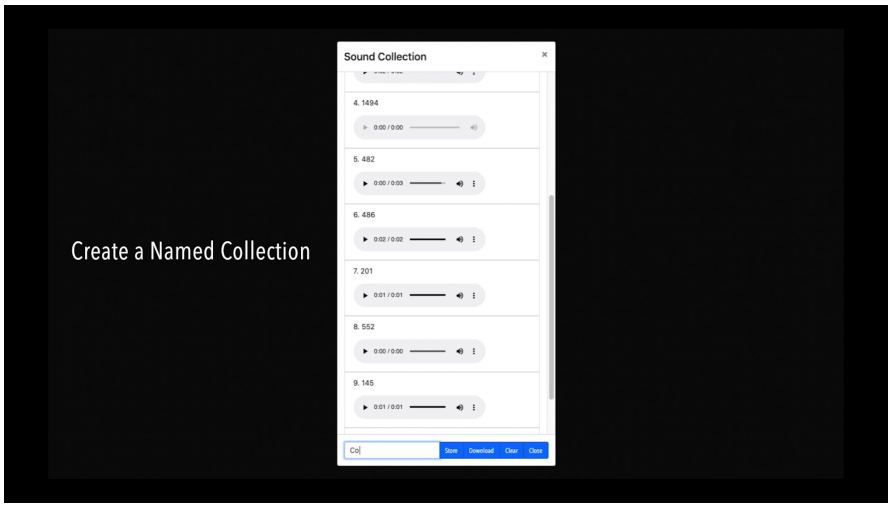


Figure 4.15: 00:59.4: Collection formation.

The modal view in fig. 4.15 allows us to review query results and, if desired, add them to a named collection stored persistently in the database. We can aggregate several queries and manual selections this way before making a collection from them.

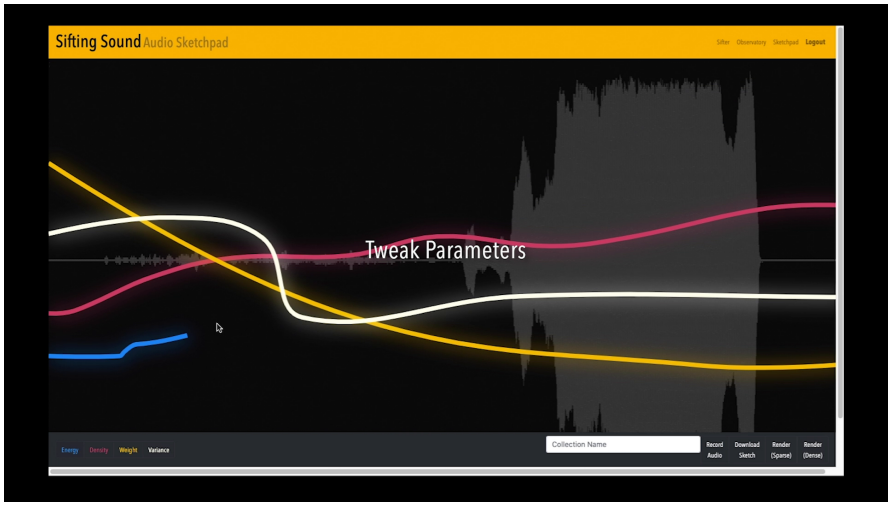


Figure 4.16: 01:24.9: Parameter-path drawing.

Next, we turn to the *Sketchpad* to make a composition from these sounds. We record an audio sketch for reference, design parameter paths as shown in fig. 4.16, and optionally enter a collection name into the box at the bottom, to specify a set of sounds to compose with.

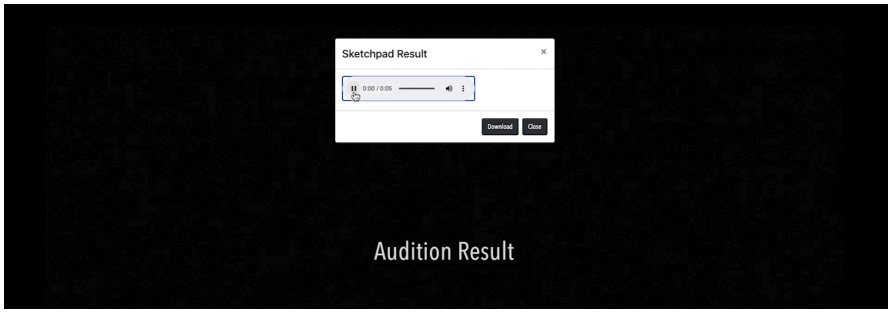


Figure 4.17: 01:43.9: Composition preview.

Finally, we can audition the result and optionally download it (fig. 4.17). Auditioning this result also acts as part of an interactive process: the sketch parameters are preserved so that they can be quickly modified after listening, and a new composition can be attempted.

5
Conclusion

This thesis has presented the design and development of an interactive software environment for creative audio composition and production with large and diverse sound collections. As part of this, we examined algorithms, processes, software systems, and user interfaces that together compose such an expressive platform, and explored how the design of these elements has contributed to the project's goals.

The work thus far has also laid the foundation for future research and development. This includes studies to characterize the contextual effectiveness of the tools and designs, and a number of planned improvements, modifications, and optimizations. Additionally, compositional systems that extend user creative effort and artistic judgment while providing accessible interfaces are a rich and exciting area for future development.

5.1 Future Work

Methods, Features, and Parameters

Currently, some of the algorithms run quite efficiently even with large amounts of data. This is also true of the query methods. However, in the interest of scaling to larger collections and maintaining interactivity, we can introduce additional con-

straints and approximations at determined data thresholds, to allow the methods to adapt to the scale they are attempted at.

Additional methods that use other features, such as the VGGish features, are already in the process of being implemented and tested. These can help leverage their descriptive power, while making them interpretable to the extent needed by the application. Additionally, new methods for the *Sketchpad*'s collage building may be of interest; for example, introducing and optimizing other mix parameters for collages (variable amplitudes and audio processing for individual sounds, for instance).

Another area for improvement is control of already existing parameters. Currently, there isn't an easy way to externally control many of the parameters involved in the system's processes. It may be helpful to identify some of these that have expressive meaning, as has already been done in some cases, and design ways to control them that don't require detailed technical information and are accessible.

Finally, the features chosen in this thesis represent a range of potentially useful measurements. Presently, they are selected through a combination of some experimentation and experience. Detailed experimentation can help to produce a more optimal set of features for the tasks outlined in this thesis. Additionally, engineering new purpose-designed features can help to keep the dimensionality low.

User Interfaces

At present, the UIs are designed to be simple and allow control of the most salient functionality in the project. This also means that they lack some of the detail offered by the underlying software package. Partially redesigning their layouts to add expansions that offer more advanced controls can help to maintain the simplicity while allowing more experienced users more comprehensive command as necessary.

Deployment Preparation

A significant amount of work is needed to make this project production-ready, so that it can be deployed as part of a large-scale collaborative venture. This ranges from refining existing implementations to implementing additional inter-

face elements, software infrastructure such as database access layers and API endpoints, and connecting things together while practically testing them.

New Elements

This work provides a base on which to build creative sound-oriented applications. It is hoped that the concepts and tools detailed in this thesis can support a range of new client applications, designed for different purposes, users, and user communities. Curation, exploration, and combination each admit a wide range of potential applications that enhance and build on human creative actions.

Far from discrete, isolated regions of sound-based creative activity, these tasks naturally complement and relate to one another and are part of a fluid environment of sound production and music composition. *Sifting Sound* does not aim to enumerate or comprehensively address all such important areas of musical composition with sounds. Future work might take a similar integrative approach with other related practices, such as sound synthesis, long-form composition, soundscape construction, etc. and build tools for them that share information.

Another dimension for new tool development is level of expertise or familiarity. This can manifest in interface design, but also in parameter complexity and presentation, and even the basic framing of the tools: what they do, and the kinds of outcomes they are intended for. *Sifting Sound* offers some flexibility in this regard, and the intended audience can be specified in terms of the presentation of parameters, and complexity of controls. Future work can build on this platform to develop tools intended for specific audiences.

Finally, another important factor in the user communities is style. Style is a highly complex phenomenon, and models of stylistic context can help to produce diversely interesting and artistic outcomes. The TEMP model presented in this work helps to address this partially, but a great variability of sounds in each category encourages further effort. Future applications may also blend sound-based music composition and note-based music composition, as is the foundation of not only much modern music in the classical tradition, but also a great deal of popular music. This rich and expanding universe of sound invites much more effort in creatively examining and expressively transforming its diverse contents.

New Music

Artistic forms and narratives built on recombinant media are diverse and dominant in our age, and constantly growing. Even more broadly, perhaps everything is, if informally, a remix; existing media is filtered through individual and community perspectives and tastes, and transformed into new work, as we discussed at the very beginning of this thesis.

A long-term goal is to drive forward such combinatorial expressions, expanding the space of musical possibilities. Just as new communications technologies helped to give rise to our sound-based, remix-oriented musical cultures, we can imagine vibrant new musics building on our age of information technologies, mediated through powerful, creatively useful tools. *Sifting Sound* takes a step in this direction, toward a new hyper-recombinant aesthetic concept, craft, and creative practice.

A

Audio Examples: Sketch 1 and 2

Description

These files represent two audio sketches, as well as two renderings of each with the two *Sketchpad* methods.

Files

1. **sketch1.aif** 836K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/sketch1.aif

2. **method1_stringstosketch1.wav** 643K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method1_stringstosketch1.wav

3. **method2_stringstosketch1.wav** 643K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_stringstosketch1.wav

4. **sketch2.aif** 1.5M
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/sketch2.aif

5. **method1_stringstosketch2.wav** 1.1M
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method1_stringstosketch2.wav

6. **method2_stringstosketch2.wav** 1.1M
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_stringstosketch2.wav

QR Code

B

Audio Examples: Pink Noise

Description

These files represent the output of using pink noise as an audio sketch, varying some parameters.

Files

1. **method1_snarebasedtonoise.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method1_snarebasedtonoise.wav

2. **method1_tonalfragmentstonoise.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method1_tonalfragmentstonoise.wav

3. **method2_snarebasedtonoise.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_snarebasedtonoise.wav

4. **method2_tonalfragmentstonoise_density_0.01.wav**
   274K http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_tonalfragmentstonoise_density_0.01.wav

5. **method2_tonalfragmentstonoise_density_0.1_with_initialstate.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_tonalfragmentstonoise_density_0.1_with_initialstate.wav

6. **method2_tonalfragmentstonoise_density_0.2_with_initialstate.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_tonalfragmentstonoise_density_0.2_with_initialstate.wav

7. **method2_tonalfragmentstonoise_density_0.4.wav** 274K

http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_tonalfragmentstonoise_density_0.4.wav

8. **method2_tonalfragmentstonoise_density_1.wav** 274K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/method2_tonalfragmentstonoise_density_1.wav

QR Code

C

Audiovisual Examples

Description

These files are the audio input and outputs for the audiovisual example.

Files

1. **sketch_av.wav** 69K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/sketch_av.wav

2. **sketchoutput_av1.wav** 686K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/sketchoutput_av1.wav

3. **sketchoutput_av2.wav** 686K
   http://web.media.mit.edu/~nsingh1/files/audio_examples/sketchpad/sketchoutput_av2.wav

QR Code

D

Interface Walkthrough

Description

This is a video file containing a walkthrough of *Sifting Sound*'s interface.

Image

Figure D.1: Title frame from SiftingSoundThesisDemo.mp4.

File

SiftingSoundThesisDemo.mp4 15M

http://web.media.mit.edu/~nsingh1/files/video_examples/SiftingSoundThesisDemo.mp4

QR Code

References

[50f] 50 for the Future. [Online]. Available from: https://50ftf.kronosquartet.org/.

[Cit] City symphonies. [Online]. Available from: https://citysymphonies.media.mit.edu/.

[Spo] Spoons. [Online]. Available from: http://www.folkmusic.ru/logka.php.

[4] (2019). The Kronos Quartet Presents Tod Machover's GAMMIFIED. [Online]. Available from: https://www.media.mit.edu/events/the-kronos-quartet-presents-tod-machover-s-gammified/.

[5] Agafonkin, V. (2017). RBush: A high-performance JavaScript R-tree-based 2D spatial index for points and rectangles. [Online]. Available from: https://github.com/mourner/rbush.

[6] Aldoshina, I. & Davidenkova, E. (2016). The history of electro-musical instruments in Russia in the first half of the twentieth century.

[7] Bartsch, M. A. & Wakefield, G. H. (2001). To catch a chorus: using chroma-based representations for audio thumbnailing. In *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)* (pp. 15–18).

[8] Belkin, M. & Niyogi, P. (2003). Laplacian Eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6), 1373–1396.

[9] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828.

[10] Bernardes, G., Guedes, C., & Pennycook, B. (2013). EarGram: An application for interactive exploration of concatenative sound synthesis in Pure Data. In M. Aramaki, M. Barthet, R. Kronland-Martinet, & S. Ystad (Eds.), *From Sounds to Music and Emotions* (pp. 110–129).

[11] Boden, M. A. (1998). Creativity and artificial intelligence. *Artificial Intelligence*, 103(1), 347 – 356.

[12] Boden, M. A. (2009). *Creativity: How Does It Work?*, chapter 13, (pp. 235 – 250).

[13] Boden, M. A. (2010). *Creativity and Art: Three Roads to Surprise*. Oxford University Press.

[14] Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, J. R., & Serra, X. (2013). Essentia: An audio analysis library for music information retrieval. In *ISMIR*.

[15] Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2301–2309.

[16] Brownlee, J. (2014). Discover feature engineering, how to engineer features and how to get good at it. [Online]. Available from: https://machinelearningmastery.

com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/.

[17] Cannam, C., Landone, C., Sandler, M., & Bello, J. (2006). The Sonic Visualiser: A visualisation platform for semantic descriptors from musical signals. *ISMIR 2006 - 7th International Conference on Music Information Retrieval*, (pp. 324–327).

[18] Card, S. K., Robertson, G. G., & Mackinlay, J. D. (1991). The Information Visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91 (pp. 181–186). New York, NY, USA: Association for Computing Machinery.

[19] Carpentier, G., Assayag, G., & Saint-James, E. (2010). Solving the musical orchestration problem using multiobjective constrained optimization with a genetic local search approach. *Journal of Heuristics*, 16(5), 681–714.

[20] Cassidy, A. (2013). Constraint schemata, multi-axis movement modeling, and unified, multi-parametric notation for strings and voices. *Search: Journal for New Music and Culture*, 10.

[21] Coblentz, H. (2019). Glass music of the twentieth and twenty-first centuries. *Tempo*, 73(289), 30–41.

[22] Colyer, C. (1986). *Studio Report*. Technical report, Centre d'Etudes de Mathematique et Automatique Musicales, France.

[23] Dau, H. A. & Keogh, E. (2017). Matrix Profile V: A generic technique to incorporate domain knowledge into motif discovery. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17 (pp. 125–134). New York, NY, USA: Association for Computing Machinery.

[24] de Cheveigné, A. & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4), 1917–1930.

[25] de Reydellet, J. (1996). Pierre schaeffer, 1910-1995: The founder of "musique concrète". *Computer Music Journal*, 20(2), 10–11.

[26] Delmotte, V. D. (2012). *Computational Auditory Saliency*. PhD thesis.

[27] Dobson, K., Whitman, B., & Ellis, D. P. W. (2005). Learning auditory models of machine voices. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.* (pp. 339–342).

[28] Farbood, M. M., Pasztor, E., & Jennings, K. (2004). Hyperscore: A graphical sketchpad for novice composers. *IEEE Comput. Graph. Appl.*, 24(1), 50–54.

[29] Finch, C., Parisot, T., & Needham, C. (2017). Peaks.js: Audio waveform rendering in the browser. *BBC R&D*.

[30] Font, F., Roma, G., & Serra, X. (2013). Freesound technical demo. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13 (pp. 411–412). New York, NY, USA: Association for Computing Machinery.

[31] Ganchev, T., Fakotakis, N., & George, K. (2005). Comparative evaluation of various MFCC implementations on the speaker verification task. *Proceedings of the SPECOM*, 1.

[32] Gemmeke, J. F., Ellis, D. P. W., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., & Ritter, M. (2017). Audio Set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 776–780).

[33] Giannakopoulos, T. & Pikrakis, A. (2014). Audio features. In *Introduction to Audio Analysis* chapter 4, (pp. 59 – 103). Academic Press.

[34] Gillebaart, M., Förster, J., Rotteveel, M., & Jehle, A. C. M. (2013). Unraveling effects of novelty on creativity. *Creativity Research Journal*, 25(3), 280–285.

[35] Gillick, J., Cella, C. E., & Bamman, D. (2019). Estimating unobserved audio features for target-based orchestration. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR, November 4-8, 2019* (pp. 192–199).

[36] Goldschmidt, G. (1991). The dialectics of sketching. *Creativity Research Journal*, 4(2), 123–143.

[37] Gomes, J. M., Chambel, T., & Langlois, T. (2013). SoundsLike: Movies soundtrack browsing and labeling based on relevance feedback and gamification. In *Proceedings of the 11th European Conference on Interactive TV and Video* (pp. 59–62).

[38] Guennebaud, G., Jacob, B., et al. (2014). Eigen: A C++ linear algebra library. *URL http://eigen.tuxfamily.org*.

[39] Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3, 1157–1182.

[40] Hackbarth, B., Schnell, N., & Schwarz, D. (2010). AudioGuide: A framework for creative exploration of concatenative sound synthesis.

[41] Harkins, P. (2016). Microsampling: From Akufen's microhouse to Todd Edwards and the sound of UK Garage. In *Musical Rhythm in the Age of Digital Reproduction* (pp. 177–194).

[42] Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose.

[43] Helen, M. & Lahti, T. (2006). Query by example methods for audio signals. In *Proceedings of the 7th Nordic Signal Processing Symposium - NORSIG 2006* (pp. 302–305).

[44] Huang, N. & Elhilali, M. (2017). Auditory salience using natural soundscapes. *The Journal of the Acoustical Society of America*, 141(3), 2163–2176.

[45] jehan, T. (2005). *Creating Music By Listening*. PhD thesis.

[46] Kim, B. & Pardo, B. (2017). I-SED: An interactive sound event detector. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, IUI '17 (pp. 553–557). New York, NY, USA: Association for Computing Machinery.

[47] Kim, K., Lin, K.-H., Walther, D. B., Hasegawa-Johnson, M. A., & Huang, T. S. (2014). Automatic detection of auditory salience with optimized linear filters derived from human annotation. *Pattern Recognition Letters*, 38, 78 – 85.

[48] Kobel, M. (2019). The drum machine's ear: XLN Audio's drum sequencer XO and algorithmic listening. *Sound Studies*, 5(2), 201–204.

[49] Kruskal, J. & Wish, M. (1978). *Multidimensional Scaling*. Sage Publications.

[50] Kules, B. (2005). Supporting creativity with search tools. *Creativity Support Tools*, (pp. 53–64).

[51] Kullback, S. & Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, 22(1), 79–86.

[52] Lansky, P. (1992). Table's Clear.

[53] Livingstone, S. R. & Russo, F. A. (2018). The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. *PLOS ONE*, 13(5), 1–35.

[54] Macedo, L. & Cardoso, A. (2001). Creativity and surprise. In *Proceedings of the AISB'01 Symposium on Creativity in Arts and Science. SSAISB* (pp. 84–92).

[55] Madrid, F., Imani, S., Mercer, R., Zimmerman, Z., Shakibay, N., & Keogh, E. (2019). Matrix Profile XX: Finding and visualizing time series motifs of all lengths using the Matrix Profile. In *2019 IEEE International Conference on Big Knowledge (ICBK)* (pp. 175–182).

[56] Manning, P. (2012). The Oramics machine: From vision to reality. *Organised Sound*, 17(2), 137–147.

[57] McDonald, K., Tan, M., & Mann, Y. (2017). The Infinite Drum Machine. [Online]. Available from: https://experiments.withgoogle.com/drum-machine.

[58] McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction.

[59] McLeod, K. (2002). How copyright law changed hip hop: An interview with Public Enemy's Chuck D and Hank Shocklee. *Stay Free Magazine*, 20.

[60] Mesaros, A., Heittola, T., Eronen, A., & Virtanen, T. (2010). Acoustic event detection in real life recordings. In *2010 18th European Signal Processing Conference* (pp. 1267–1271).

[61] Miller, P. D. (2004). Algorithms: Erasures and the art of memory. *Audio culture: Readings in modern music*, (pp. 348–54).

[62] Mueen, A., Zhu, Y., Yeh, M., Kamgar, K., Viswanathan, K., Gupta, C., & Keogh, E. (2017). The fastest similarity search algorithm for time series subsequences under euclidean distance. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[63] Müller, M. (2007). Dynamic time warping. In *Information Retrieval for Music and Motion* (pp. 69–84). Springer Berlin Heidelberg.

[64] Nielsen, J. (1994). *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[65] Nilsson, P. A. (2011). *A Field of Possibilities: Designing and Playing Digital Musical Instruments*. PhD thesis.

[66] Olsen, D. R. (2007). Evaluating user interface systems research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07 (pp. 251–258). New York, NY, USA: Association for Computing Machinery.

[67] Oswald, J. (1985). Plunderphonics, or audio piracy as a compositional prerogative. In *Wired Society Electro-Acoustic Conference, Toronto*.

[68] Peeters, G. (2004). A large set of audio features for sound description (similarity and classification) in the CUIDADO project.

[69] Ramsay, D., Ananthabhotla, I., & Paradiso, J. (2019). The intrinsic memorability of everyday sounds. In *Audio Engineering Society Conference: 2019 AES International Conference on Immersive and Interactive Audio*.

[70] Roads, C. (2005). The art of articulation: The electroacoustic music of Horacio Vaggione. *Contemporary Music Review*, 24(4-5), 295–309.

[71] Robertson, E. (2010). "It Looks Like Sound!": Drawing a history of "animated music" in the early twentieth century. Master's thesis, University of Maryland, College Park.

[72] Rocchesso, D., Lemaitre, G., Susini, P., Ternström, S., & Boussard, P. (2015). Sketching sound with voice and gesture. *Interactions*, 22(1), 38–41.

[73] Russolo, L. (1913). *The Art of Noise*. Something Else Press.

[74] Salamon, J. & Gomez, E. (2012). Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6), 1759–1770.

[75] Schnell, N., Roebel, A., Schwarz, D., Peeters, G., & Borghesi, R. (2009). MuBu & friends - assembling tools for content based real-time interactive processing in Max/MSP. *Proceedings of the International Computer Music Conference (ICMC 2009)*.

[76] Schwarz, D. (2007). Corpus-based concatenative synthesis. *IEEE Signal Processing Magazine*, 24(2), 92–104.

[77] Schwarz, D., Beller, G., Verbrugghe, B., & Britton, S. (2006). Real-time corpus-based concatenative synthesis with CataRT. In *9th International Conference on Digital Audio Effects (DAFx)* (pp. 279–282).

[78] Smith, L. I. (2002). *A tutorial on principal components analysis*. Technical report, Cornell University, USA.

[79] Tordini, F., Bregman, A. S., & Cooperstock, J. R. (2015). The loud bird doesn't (always) get the worm: Why computational salience also needs brightness and tempo.

[80] Torpey, P. A. (2013). *Media Scores: A framework for composing the modern-day Gesamtkunstwerk*. PhD thesis.

[81] Tremblay, P., Green, O., Roma, G., & Harker, A. (2019). From collections to corpora: Exploring sounds through fluid decomposition. In *International Computer Music Conference Proceedings 2019*, International Computer Music Conference Proceedings.

[82] Tzanetakis, G. (2011). Audio feature extraction. In T. Li, M. Ogihara, & G. Tzanetakis (Eds.), *Music Data Mining*. USA: CRC Press, Inc., 1st edition.

[83] Ussachevsky, V. (1960). Wireless Fantasy.

[84] van Troyer, A. (2013). Constellation: A tool for creative dialog between audience and composer. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research* (pp. 534–541).

[85] van Troyer, A. (2017a). Mm-rt: a tabletop musical instrument for musical wonderers. In *NIME* (pp. 186–191).

[86] van Troyer, A. (2017b). *Score instruments: a new paradigm of musical instruments to guide musical wonderers*. PhD thesis, Massachusetts Institute of Technology.

[87] Wang, J., Zhang, K., Madani, K., & Sabourin, C. (2015). Salient environmental sound detection framework for machine awareness. *Neurocomputing*, 152, 444 – 454.

[88] Westerkamp, H. (2002). Linking soundscape composition and acoustic ecology. *Organised Sound*, 7(1), 51–56.

[89] Wright, M. C., Freed, A., Rodet, X., Virolle, D., & Woehrmann, R. (1998). New applications of the Sound Description Interchange Format. In *ICMC: International Computer Music Conference* Ann Arbor, United States.

[90] Yeh, C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., & Keogh, E. (2016). Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th International Conference on Data Mining (ICDM)* (pp. 1317–1322).