SONIFICATION WORKSTATION

Sean Phillips

Media Arts and Technology Department
University of California Santa Barbara
Goleta, CA USA
seanphillips@ucsb.edu

Andrés Cabrera

Media Arts and Technology Department
University of California Santa Barbara
Goleta, CA USA
andres@mat.ucsb.edu

ABSTRACT

Sonification Workstation is an open-source application for general sonification tasks, designed with ease-of-use and wide applicability in mind. Intended to foster adoption of sonification across disciplines, and increase experimentation with sonification by non-specialists, *Sonification Workstation* distills tasks useful in sonification and encapsulates them in a single software environment. The novel interface combines familiar modes of navigation from Digital Audio Workstations, with a highly simplified patcher interface for creating the sonification scheme. Further, the software associates methods of sonification with the data they sonify, in session files, which will make sharing and reproducing sonifications easier. It is posited that facilitating experimentation by non-specialists will increase the potential growth of sonification into fresh territory, encourage discussion of sonification techniques and uses, and create a larger pool of ideas to draw from in advancing the field of sonification. Source code is available at **https://github.com/Cherdyakov/sonification-workstation**. Binaries for macOS and Windows, as well as sample content, are available at **http://sonificationworkstation.org**.

1. INTRODUCTION

When referring to sonification applications we mean finished software programs targeting end-users, with a focus on creating sonifications. A broad definition of sonification is best for our purposes, and "the technique of rendering sound in response to data and interactions," which is found in section 1.1 of *The Sonification Handbook* is suitable [1]. This includes methods which convert data samples directly into amplitudes, known as *audification* and sometimes treated separately.

Comprehensive figures on the software used in sonification research are not readily available, but in 2012 Bearman and Brown reviewed 51 articles on sonification and found domain-specific programming languages to be the most common tools [2]. Their survey found *Supercollider* [3] and *Pure Data* [4] to be especially popular, with *Supercollider* leading the way amongst research published in The Proceedings of the International Conference on Auditory Display (ICAD) [5]. This is despite the existence of multiple sonification applications.

Sonification tools can generally be placed on a spectrum from most to least flexible, which usually correlates with the degree of complexity and therefore has an inverse relationship with accessibility to novices. On one extreme of this spectrum are the general-purpose programming languages. Domain-specific programming languages fall along the middle of this spectrum, while at the other extreme are found end-user applications dedicated to creating data sonifications.

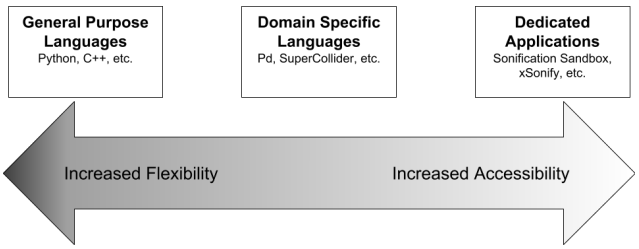

Figure 1: Sonification tools on a conceptual spectrum, illustrating a common trade-off between flexibility and ease-of-use.

End-user applications have been designed to sonify specific data types [6], but we are concerned here with applications designed for general sonification tasks. This implies applications which can load datasets of assorted sizes, sonify them in multiple, user-selected ways, and which place few restrictions on what the dataset represents. The given criteria still allow for a wide range of software types and a handful have been tested over the years, though few appear to be actively under development.

2. RELATED WORK

This section briefly describes some of the more significant, dedicated sonification applications that have been developed.

2.1. Sonification Sandbox (2003)

Sonification Sandbox was "motivated by the need for a multi-platform, multi-purpose toolkit for sonifying data" [7]. The program generates MIDI output, rather than audio. The graphical interface provides tabs for viewing the data, altering the parameter mappings, and adding context. In a sonification, context is non-signal information added to the output to help the listener interpret what they hear, analogous to the axes and trend lines on a visual graph [1]. In the *Sonification Sandbox* these context cues can include reference pitches for comparing to data values and click tracks to assist in interpreting time. The software is in beta and hasn't been updated for newer versions of Java, but it is still

available for download at the Georgia Tech School of Psychology website [8].

2.2. SonART (2003)

The *SonART* toolkit used *The Synthesis ToolKit* (STK) [9] for synthesis and audio output, but added a scheduler and "parameter matrix engine," in an effort to provide cross-platform GUI tools for auditory display [10]. Akin to an audio matrix router in operation, the parameter matrix arrayed data parameters along the top of a 2D matrix and synthesis parameters down the right-hand side. This matrix arrangement allowed for arbitrary mapping of data to control parameters. The original paper proposes an ambitious long-term plan, with stated goals of "laying the foundation for an ongoing open-source collaborative effort," and "establishing and maintaining a well-documented and publicly accessible repository of sonification development tools." However, looking at internet archives, it appears the only download link for *SonART* dates to 2004 or earlier, not long after publication. Unfortunately, the downloads are no longer available [11]. The paper describes *SonART* as cross-platform and seems to contain screenshots from a Windows build, but it was not ultimately released for that platform.[1] *SonART* emphasized image sonification over general sonification tasks in the final release.

2.3. xSonify (2006)

NASA's *xSonify* was developed to sonify one-dimensional space physics data [12]. This makes it narrower in scope than other programs under consideration, but it still targets many datasets and has a history of practical application that makes it interesting. *xSonify* offers pitch, loudness, and rhythm modes, and some data pre-processing. *xSonify* includes text-to-speech facilities for menu navigation and a strong focus of the project has been accessibility for the visually-impaired. Co-author of the original *xSonify* paper Wanda L. Diaz Merced is blind and has used sonification in her physics research for many years [13, 14]. Merced also used a prototype of *xSonify* with visually-impaired students at the University of Puerto Rico [12]. *xSonify* is available at the Sonification Research page of NASA's website [15].

2.4. Sonifyer (2008)

Sonifyer is meant to be an easy-to-use sonification program, accessible even to amateurs. The authors became interested in such a project while sonifying EEG data with the Max/MSP [16] framework, writing that their Max sonification system became increasingly difficult to teach newcomers as it grew in complexity [17]. They also noted the steep learning curve of *Supercollider*, which they acknowledged as a popular sonification tool. *Sonifyer* is an effort to bring the user-friendliness of consumer software to the sonification space, including easy availability and installation, citing iTunes as a benchmark example. The original paper on *Sonifyer* also stressed the need for a more active community and easy sharing of sonifications. To address such needs the authors introduced a companion website alongside *Sonifyer*, which they hoped would provide a place to share audio samples and community knowledge. As of this writing the *Sonifyer* website appears to have very little content posted after 2009, and no samples posted after 2011 [18]. Curiously for a project aimed at wide adoption, *Sonifyer* will not function without obtaining a license from the makers via e-mail, and is available only for macOS. *Sonifyer* provides audification (which appears to be a strong suit) and limited FM parameter-mapping sonification.

2.5. Rotator (2016)

Rotator was created at MIT by Juliana Cherston [19]. It is a client-side web application, written in JavaScript, React, and Flux. It has a novel interface, which allows for visualization and sonification of multiple data streams at the same time. The software is aimed at "diversifying the way that users distribute data across their senses" [19, 20]. *Rotator* assumes the data has a geometric relationship and a user-provided schematic of the data origin-points is the key UI component. Users place bounding boxes around clusters of data streams on the schematic; one bounding box dictates the streams currently being visualized, another dictates the streams being sonified. The two boxes are fully independent and can overlap or delineate exclusive areas of the schematic. There are six synthesis possibilities, including audification. The *Rotator* project was largely for experimentation and is not under active development at the time of writing.[2]

3. SONIFICATION WORKSTATION

3.1. Motivation

The preceding overview of existing sonification software should help clarify motivations for the *Sonification Workstation* project. The state of the field suggests an opening for current work on sonification applications. Domain-specific programming languages such as *Supercollider* and *Pure Data* have contributed to numerous publications, and exhibit ongoing development [21, 22]. This contrasts with the more experimental nature and limited life-span of dedicated software, and invites additional efforts in the application space.

Sonification Workstation is an attempt to capture some of the utility of the domain-specific language solutions, while providing the simplified access to established sonification techniques and processes sought by prior dedicated software. Additionally, technical decisions were made to ease ongoing development and hopefully increase the project's longevity (see 4.2).

3.2. Application Overview

The *Sonification Workstation* interface consists of a single window, divided into two parts; the *data view* and the *patcher view*. The data view is the main user-interface for controlling playback of the data being sonified, and is analogous to the waveform view in a Digital Audio Workstation (DAW), such as *Pro Tools* or *Reaper*. The patcher view is where the user creates the synthesis tree that will determine the character of the sound. These two interfaces work together to allow data playback, parameter mapping, and synthesis design, without interrupting the flow of listening and evaluating.

3.2.1. Data View

The data view is populated whenever a new dataset is loaded via the *File* menu. Currently, data can be imported from CSV files. CSV columns are converted to tracks and plotted. Columns are more

[1] J. Berger, personal communication, September, 2017

[2] J. Cherston, personal communication, March, 2019

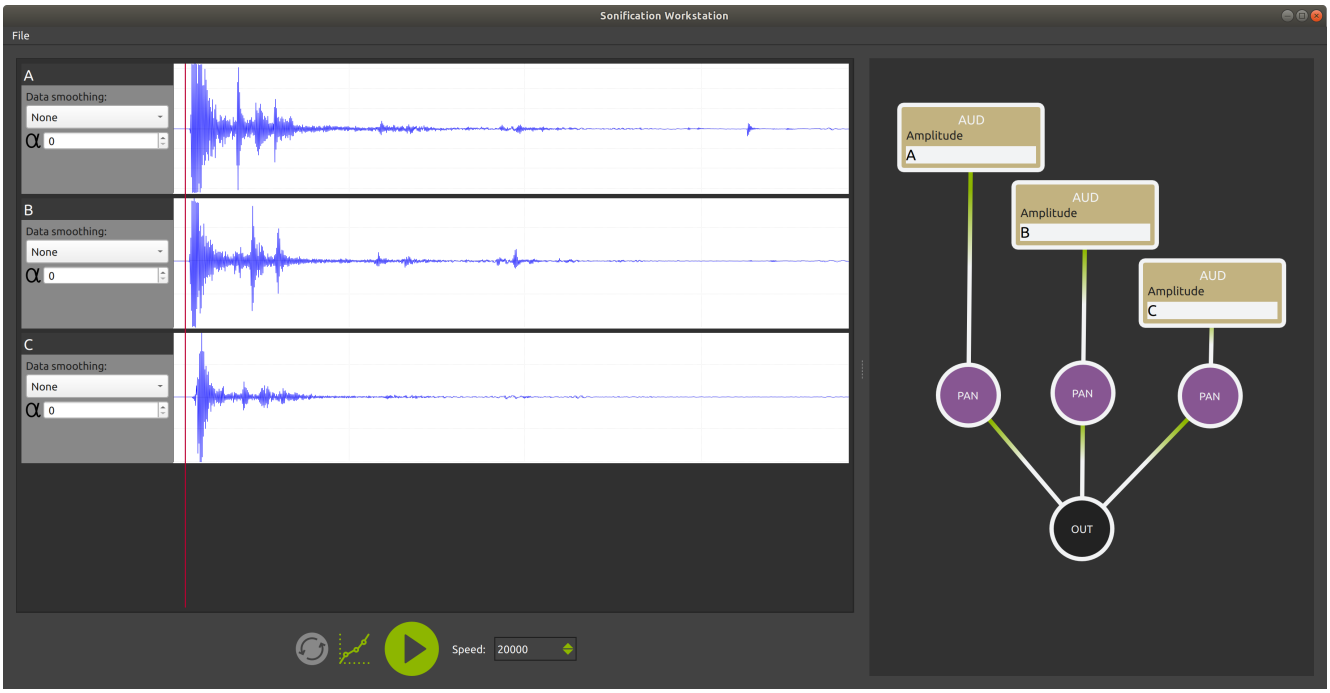

Figure 2: *Sonification Workstation* application window, configured for audification of a three channel seismic dataset at a playback rate of 20,000 data samples per second and using interpolation.

suited to represent individual data dimensions than rows, since common CSV editors can access only a limited number of columns in a single CSV file. Such programs will truncate a long series of values if it is entered in a row. Once loaded, data tracks are assigned variable names, for use in parameter mapping expressions (see 3.4).

Data is treated as a signal source, and the time domain can be quickly navigated by mouse. Clicking anywhere along the plotted data will move the playback cursor, right-clicking and dragging will create a bounded area for looping playback. Transport controls, seen directly below the data tracks, provide controls for play/pause, setting the playback speed, enabling looped playback, and enabling real-time interpolation between data points during playback. Playback speed is equivalent to the number of data points read every second and scales from zero to audio rate (48kHz). Playback is synchronized across all tracks, so that each sample of data playback represents a single point in the dataset, across all dimensions.

3.2.2. Patcher View

One of the important contributions of *Sonification Workstation* is providing a flexible way to construct a sonification, without the need for domain-specific knowledge, by consolidating the main techniques into a very simple patching interface. With a small set of synthesis components and the available data mapping and scaling features (see following sections), *Sonification Workstation* offers tools for additive synthesis, subtractive synthesis, frequency and amplitude (AM and FM) modulation, and audification. Context for values can be created via fixed frequency oscillators or noise beds. Context for time can be added with short AD envelopes, triggered at data playback rate, or synchronizing modulation and playback rates.

Figure 2 shows a three-track seismic dataset. The patcher interface has been populated with synthesis components to audify and pan the three channels. The Audification (AUD) synthesis components have been maximized (see section 3.4), showing tracks A, B, and C, have each been mapped to a single AUD component.

3.3. High-Level Synthesis Components

The high-level synthesis components in *Sonification Workstation* encapsulate the data mapping and audio settings. The patcher interface is inspired by domain-specific patcher languages such as Pure Data and Max, but is quite simple in comparison. There are fewer than a dozen instantiable types and no differentiation between mono, stereo, or control signals. A brief description of the existing high-level synthesis components follows.

The OSC Component

A sinusoidal oscillator. Accepts arbitrary functions mapped to frequency and optionally scales the frequency value within a user-selected range.

The AM Component

An amplitude modulator. Will modulate the amplitude of any parent synthesis component. The frequency accepts arbitrary mappings and the value can be scaled.

The FM Component

A frequency modulator. Will modulate the frequency of parent OSC, AM, and FM components. Will also modulate the pan position of the PAN object. Frequency and depth parameters accept arbitrary mappings and can be scaled.

The AUD Component

Turns the results of data mappings directly into amplitudes

for audification. Values are always scaled within the range [-1.0,1.0] to prevent clipping and maximize gain.

The PAN Component

Pans the output of connected components in the stereo field. Pan position accepts arbitrary mappings and can be scaled, but values are clipped to the range [-1.0,1.0].

The ENV Component

Applies an AD envelope to connected components. Attack and Decay values accept arbitrary mappings and the values can be scaled.

The VOL Component

Scales the output of connected components. Accepts arbitrary mappings and can be scaled. The gain value will be clipped to the range [-1.0,1.0]. Negative values allow VOL to be used for phase inversion.

The NSE Component

Generates white, pink, or Brownian noise.

The EQ Component

Biquad filter with mappable resonance and frequency. Switchable high-pass, low-pass, band-pass, or notch. Combines with the NSE Component for subtractive synthesis.

The OUT Component

The root of the synthesis tree, connecting to the OUT Component will pass a component's signal to the audio callback for output to the computer sound card.

3.4. Parameter Mapping

Sonification Workstation synthesis components accept parameter mappings in the form of a mathematical expression. Data tracks are assigned names on import, these names can be used in expressions as variables and multiple data tracks can be included in the same expression. Valid mappings are constants (e.g. setting an oscillator to a fixed frequency of 440Hz), data tracks, or an arbitrary expression including both. Expressions are evaluated in real-time.

In their minimized state, each high-level synthesis component is a colored circle with a text identifier. Double-clicking maximizes the component, revealing controls for mapping and audio settings. Figure 3 shows four examples of maximized synthesis components and their settings, clock-wise from top-left these are:

1. A noise generator (NSE), set to generate white noise.
2. An amplitude modulator (AM), with frequency mapped to the square root of data track A minus data track B. It is shown modulating the amplitude of the connected oscillator directly below.
3. An oscillator (OSC), with frequency mapped to the values of data track C. The oscillator also has scaling enabled, which will scale the incoming values to fit, in this case, between 100Hz and 800Hz.
4. A pan control (PAN), set to full left.

While not implemented in the current build, the authors are also interested in adding parameter mapping to the transport. An earlier version of *Sonification Workstation* allowed parameters to control the rate of playback, essentially making time a mappable parameter.

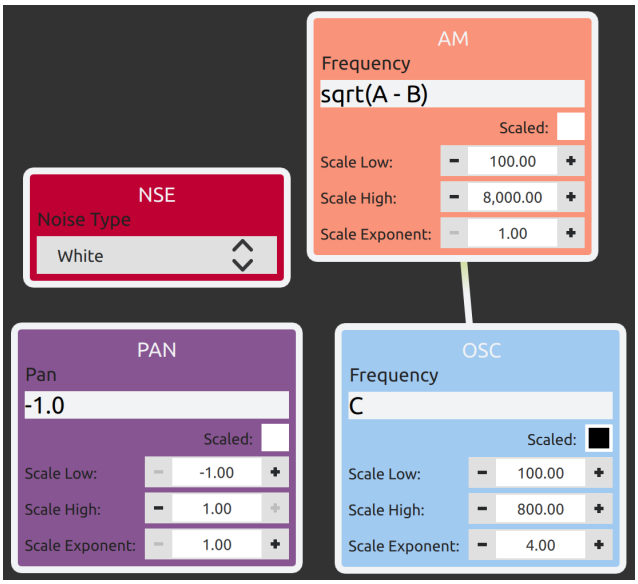

Figure 3: Maximized synthesis components, showing their data mapping and audio settings.

3.5. Data Scaling

Synthesis components provide for the scaling of data mappings to usable values. In Figure 3 the scaling controls can be seen at the bottom of the maximized AM, OSC, and NSE interfaces. Parameter scaling is controlled with the following values:

- Scaled: enables scaling. If disabled, the data values are used as-is. There is no protection against aliasing.
- Scale Low: the lowest output value desired, corresponding to the lowest data value in an assigned row, or the lowest value of the assigned expression, given the current dataset.
- Scale High: the highest output value desired, corresponding to the highest data value in an assigned row, or the highest value of the assigned expression, given the current dataset.
- Scale Exponent: controls the shape of the curve the data values are mapped to.

The formula for scaling is taken from the *scale* object in the Max programming language [23]. Inverted mappings are also possible, by setting *Scale Low* to the top of the desired scale range and *Scale High* to the bottom of the range. Inverted mappings have been identified as useful in cases where increasing data values have an intuitively inverse relationship to the data, e.g. when size is mapped to pitch [1].

3.6. Session Files

The state of a session, including the synthesis tree and the path to the currently loaded dataset, can be saved to a session file. Session files are .json files, with the dataset path and objects representing the state of each synthesis component stored in human readable form. Sharing a session file, together with the dataset, affords complete reproduction of the sonification and enables collaboration, modification and experimentation.

3.7. Color Scheme

The color scheme for UI elements in *Sonification Workstation* draws heavily from the so-called "Kelly colors." This is a set of 22 colors, meant to provide maximum contrast for color coding tasks, published by Kenneth L. Kelly for the Inter-Society Color Council in 1965 [24]. According to Green-Armytage, the ISCC has worked recently to bring Kelly's list up to date, but improving on it was difficult [25]. Kelly's list is designed so that the second color provides maximum contrast with the first, the third color will contrast maximally with colors one and two, etc. Kelly chose the first nine colors for their differentiability by individuals with red-green color blindness.

Major UI elements of *Sonification Workstation* utilize the first thirteen colors of the list. While using *Sonification Workstation* is not a color coding task per se, the data track view requires color coding and differentiable colors were desired for all of the synthesis components. It was not possible to keep to the first nine Kelly colors, while providing unique colors for each of the synthesis components. Additionally, multiple neutral shades are used to provide some organization to the main application window, where Kelly only provides values for white, black and a single grey. To further aid users less-sensitive to color differences, all synthesis components are surrounded by a white ring, providing very high contrast against the dark background of the patcher view. They are also labelled with a text-based code that indicates their function.

4. TECHNICAL NOTES

4.1. Processing Rates

The software operates at three different processing rates.

4.1.1. Audio Rate

Audio rate processing happens at the sample rate of the audio system, which is set to 48,000Hz. This is the rate at which all synthesis classes generate audio, regardless of the playback rate or the rate of change in any associated data parameters. If mapped data parameters are changing more slowly than the audio rate (a typical case), the synthesis components will generate multiple audio samples from a single data point.

4.1.2. Command Processing Rate

User commands to sonification components and the transport are buffered in a lock-free ring buffer, for consumption by the audio callback, which happens at *command process rate*. This allows the user to issue changes to synthesis parameters and data playback settings without interrupting audio processing. The command processing rate is set to the audio block rate, consequently user commands are processed once for every time the audio buffer is filled.

4.1.3. Step Rate

Step rate is the rate at which values from the dataset are read and passed to all synthesis blocks in the patcher view. This is dependent upon the data sample playback rate, which is set in samples per second. Therefore, at the default speed of "1", the step function is called on every synthesis component once every second. The step function provides a way for synthesis blocks to take special actions when new data points are reached. The ENV class, for example, can re-trigger envelope generation at the step rate.

4.2. Qt Framework

The *Sonification Workstation* is written in C++, QML, and JavaScript, using the Qt framework. The graphical front end is where the QML and JavaScript code resides, while core classes for synthesis and playback are written in C++. The current build targets Qt LTS version 5.12 and C++17. Source code is available at the project's GitHub repository (source: https://github.com/Cherdyakov/sonification-workstation). It is hoped that building with the Qt framework will help ongoing support and development of *Sonification Workstation*, since the framework itself is well-established and receives regular updates. Additionally, a number of existing sonification applications are only available for a single platform (see section 2. Related Work), limiting their potential audience. Targeting the Qt framework and using cross-platform libraries for synthesis and audio i/o means that *Sonification Workstation* can be built for macOS, Linux, and Windows.

4.3. Gamma Synthesis Library

Most of the high-level patcher objects contain lower-level synthesis classes, commonly referred to as *unit generators*. OSC contains an Oscillator unit generator, ENV contains an envelope unit generator, and so on. Many unit generators contained in the *Sonfication Workstation* synth objects are from the Gamma C++ synthesis library (source: https://github.com/LancePutnam/Gamma), written by Lance Putnam [26].

4.4. Mathematical Expression Toolkit

Evaluation of parameter mapping expressions is handled by the C++ Mathematical Expression Toolkit, written by Arash Partow (source: https://github.com/ArashPartow/exprtk).

5. TAXONOMIC EVALUATION AND FUTURE WORK

In examining the fitness of the completed project for its purpose, some criteria must be chosen. Several experts have attempted to generate sonification taxonomies. It stands to reason that a successful general purpose sonification application would address one or more well-considered taxonomies. This section evaluates the merits and shortcomings of the *Sonification Workstation* through the lens of example taxonomies, then notes significant improvements indicated by this evaluation.

5.1. Functional Taxonomy

Summarizing available research, the Sonification Handbook [1] describes the following functional categories:

1. Alarms, alerts, and warnings
2. Status, process, and monitoring messages
3. Data exploration
4. Art, entertainment, sports, and exercise

At this point in time, the software addresses category three most fully, with additional application to the artistic element of category four.

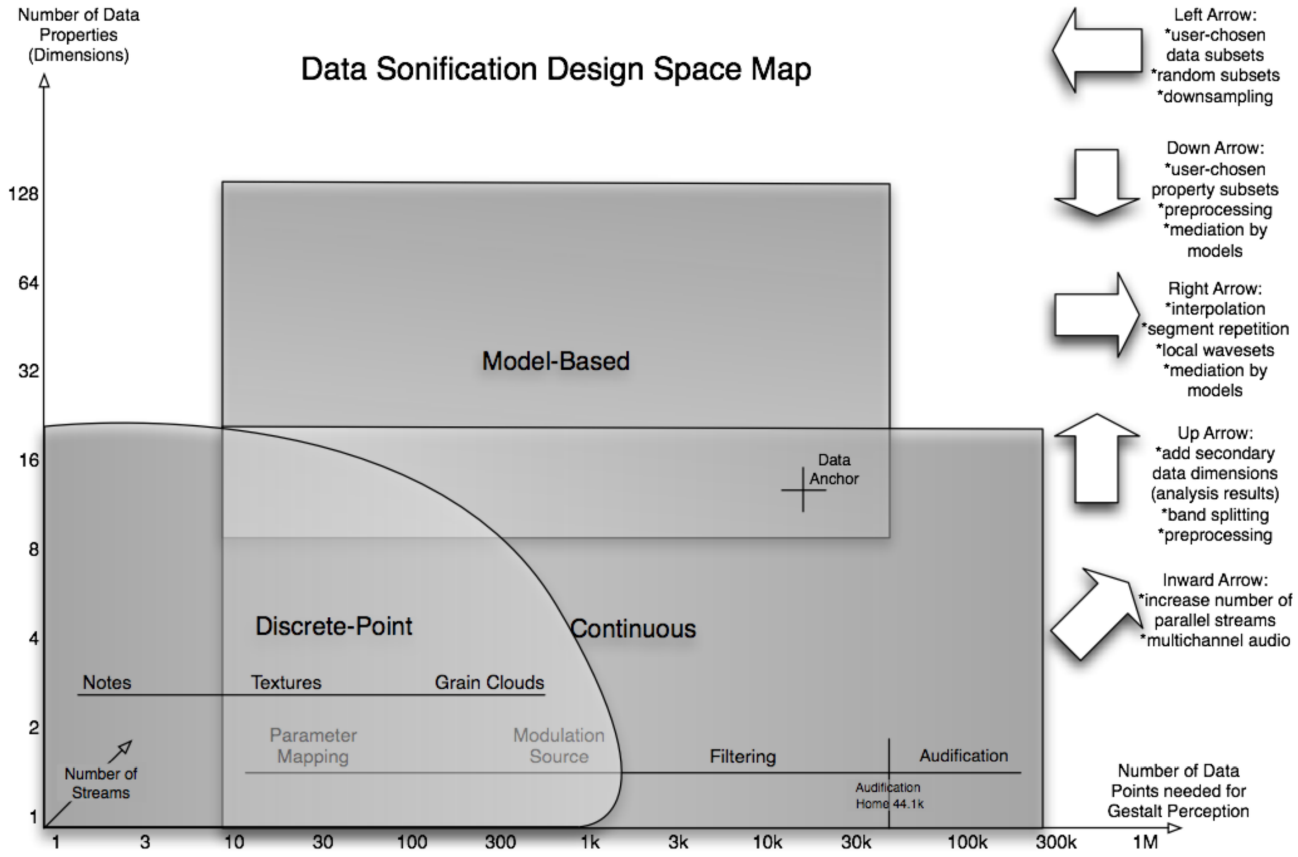

Figure 4: De Campo's Data Sonification Design Space Map, taken from the original paper. *"The overlapping zones are fuzzy areas where different sonification approaches apply; the arrows on the right refer to movements on the map, which correspond to design iterations."* — de Campo

Category one, *alarms, alerts, and warnings*, refers to sounds which "indicate something has occurred, or is about to occur" or "require immediate response or attention." Category two likewise refers to monitoring things "current or ongoing." These can be prototyped in *Sonification Workstation* using recorded data, but live streaming of input data is not currently implemented. Real-time input would be highly desirable for users working on these sorts of tasks, and sports and exercise sonification could also benefit from real-time sonification. The project should be expanded in the future, to incorporate real-time process monitoring and event based sonification via network messages.

5.2. De Campo's Sonification Design Space Map

In a 2007 paper [27], de Campo began by classifying sonification strategies into three categories:

1. Continuous Data Representation
2. Discrete Point Data Representation
3. Model-Based Data Representation

De Campo employed these categories in creating the *Data Sonification Design Space Map*. The map is intended as a guide for choosing sonification strategies, based upon the number of data dimensions, the number of simultaneous streams, and the number of data points required to comprise a single "gestalt." Briefly, number of streams refers to the number of data dimensions that are sonified in parallel, either through spatialization or using sonically distinct frequency ranges per stream, etc., while a gestalt is a perceivable pattern or recognizable audio structure that is of interest.

De Campo describes the use-case for the Design Space Map this way: "The Design Space Map enables a designer or researcher to engage in systematic reasoning about applying different sonification strategies to his/her task or problem, based on data dimensionality and perceptual concepts."

De Campo's map posits a space in which the sonification designer can move freely between many different techniques and dataset types. This view of sonification design is highly compatible with the idea of a generalized sonification application. De Campo has perhaps provided a map not only for sonification designers, but for the developers of sonification software as well. The prospect of a design aid, such as the Design Space Map, combined with a software tool meant to realize the methods it describes, is a potentially exciting development in sonification software design.

Sonification Workstation already incorporates the ability to transition along the various axes of the Design Space Map. This is not a coincidence, de Campo's paper had a strong influence on the current project. Realizing the concept of freedom of movement along these axes is only a starting point however, and specific areas of the

map are covered thinly or left unaddressed. Currently, *Sonification Workstation* offers parameter mapping, modulation source, filtering, and audification. Incorporating physical modeling is an intriguing possibility and it would help cover a large area on the design map. The challenge would be in incorporating physical modeling that is applicable to general sonification tasks. Many model-based sonifications reflect the real properties of an object or physical system under study and can be difficult to generalize. Some general model-based sonification strategies have been offered; Lee, Sell, and Berger proposed methods based on digital waveguide meshes [28], while Hermann and Ritter describe a system based on a crystal growth model [29]. Such a method is an excellent candidate for inclusion in *Sonification Workstation*, even if experimentally. *Sonification Workstation* was designed to be extensible through the addition of new synthesis components such as these.

6. SUMMARY

The authors have presented the *Sonification Workstation*, software uniquely suited to generalized sonification work, with a low barrier to entry. Prior work in this space was presented for context and the present work was examined in relation to existing sonification taxonomies, illuminating the strengths and weaknesses of the approach, and providing future direction for the project.

7. REFERENCES

[1] Various, *The Sonification Handbook*, J. G. N. Thomas Hermann, Andy Hunt, Ed. Berlin, DE: Logos Verlag Berlin, 2011.

[2] N. Bearman and E. Brown, "Who's sonifying data and how are they doing it? a comparison of icad and other venues since 2009," in *Proc. of the 18th Int. Conf. on Auditory Display*, 2012, pp. 231–232.

[3] J. McCartney, "Supercollider: A new real time synthesis language," in *Proc.of the Int. Computer Music Conference*, 1996, pp. 257–258.

[4] M. S. Puckette, "Pure data," in *Proc.of the Int. Computer Music Conference*, 1996, pp. 224–227.

[5] http://www.icad.org, accessed: 2017-10-17.

[6] R. A. Khan, R. K. Avvari, K. Wiykovics, P. Ranay, and M. Jeon, "Lifemusic: Reflection of life memories by data sonification," in *Proc. of the 22nd Int. Conf. on Auditory Display*, 2016, pp. 90–92.

[7] B. Walker and J. Cothran, "Sonification sandbox: A graphical toolkit for auditory graphs," in *Proc. of the 9th Int. Conf. on Auditory Display*, 2003, pp. 231–232.

[8] http://sonify.psych.gatech.edu/research/sonification_sandbox/, accessed: 2019-03-29.

[9] P. R. Cook and G. P. Scavone, "The Synthesis ToolKit (STK)," in *Proc. of the Int. Computer Music Conference*, 1999, pp. 164–166.

[10] ——, "Sonart: The sonification application research toolbox," in *Proc. of the 8th Int. Conf. on Auditory Display*, 2002.

[11] https://ccrma.stanford.edu/~woony/software/sonart/, accessed: 2019-03-29.

[12] R. M. Candy, A. M. Schertenleib, and W. L. D. Merced, "xSonify sonification tool for space physics," in *Proc. of the 12th Int. Conf. on Auditory Display*, 2006.

[13] http://www.npr.org/2017/01/20/510612425/how-can-we-hear-the-stars, accessed: 2019-03-29.

[14] https://www.cfa.harvard.edu/sed/projects/star_songs/pages/xraytosound.html, accessed: 2017-09-02.

[15] https://spdf.sci.gsfc.nasa.gov/research/sonification/, accessed: 2019-03-29.

[16] M. S. Puckette, "The patcher," in *Proc.of the Int. Computer Music Conference*, 1988, pp. 420–429.

[17] F. Dombois, "Sonifyer: A concept, a software, a platform," in *Proc. of the 14th Int. Conf. on Auditory Display*, 2008, pp. 1–4.

[18] http://www.sonifyer.org/?lang=e, accessed: 2019-03-29.

[19] J. M. Cherston, "Auditory display for maximizing engagement and attentive capacity," MIT, 2016.

[20] J. Cherston and J. A. Paradiso, "Rotator: Flexible distribution of data across sensory channels," in *Proc. of the 23rd Int. Conf. on Auditory Display*, 2017, pp. 86–93.

[21] https://puredata.info/downloads/pure-data, accessed: 2019-05-27.

[22] https://supercollider.github.io/archive, accessed: 2019-05-27.

[23] https://docs.cycling74.com/max7/maxobject/scale, accessed: 2019-03-29.

[24] K. L. Kelly, "Twenty-two colors of maximum contrast," *Color Engineering*, vol. 110, no. 3, pp. 26–27, 1965.

[25] P. Green-Armytag, "A colour alphabet and the limits of colour coding," *Colour: Design & Creativity*, vol. 5, no. 5, pp. 1–23, 2010.

[26] L. Putnam, "Gamma: A C++ sound synthesis library further abstracting the unit generator," in *Proc.of the Int. Computer Music Conference*, 2014, pp. 1382–1388.

[27] A. de Campo, "Toward a data sonification design space map," in *Proc. of the 13th Int. Conf. on Auditory Display*, 2007, pp. 342–347.

[28] G. S. K. Lee and J. Berger, "Sonification using digital waveguides and 2- and 3-dimensional digital waveguide mesh," in *Proc. of the 11th Int. Conf. on Auditory Display*, 2005, pp. 140–145.

[29] T. Hermann and H. J. Ritter, "Crystallization sonification of high-dimensional datasets," in *Proc. of the 8th Int. Conf. on Auditory Display*, 2002, pp. 1–6.