# SIREN: CREATIVE AND EXTENSIBLE SONIFICATION ON THE WEB

*Tristan Peng, Hongchan Choi, Jonathan Berger*

Center for Computer Research in Music and Acoustics
Stanford University
Stanford, CA, USA
{pengt, hongchan, brg}@ccrma.stanford.edu

## ABSTRACT

Parameter mapping sonification is a pervasive aspect of many everyday situations, from knocking on a watermelon to determine "goodness," to diagnostic listening to the pings and rattles of an engine. However, not all mappings are so intuitive. Sonification Interface for REmapping Nature (SIREN) is a multipurpose, powerful web application that aims at exploring sonification parameter mappings allowing for a wide range of flexibility in terms of sonic choices and mapping strategies. SIREN aspires to enable users to arrive at sophisticated sonification without getting bogged down in the details of programming.

## 1. INTRODUCTION

"In a good process of design, this conversation with the situation is reflective. In answer to the situation's back-talk, the designer reflects-in-action..." writes Donald Schön in his book *The Reflective Practitioner* [1]. A cyclical process of design, implementation, and evaluation is what Schön considers the essence of an expert designer's craft. In the following text, he narrates a story about Quist, the teacher, and Petra, the student. Experts like Quist instinctively engage in the cycle of reflection-in-action, but designers like Petra need additional assistance. Quist himself is not without tools: he uses plenty of sketching tools to facilitate his engagement with the process. Thus, it is important, from a meta-design perspective, to create links that enable not just Petra, but also Quist, to connect with the design process and spur faster and better designs. However, the status quo of designing sonifications lag behind what Schön envisions as the ideal design process. Overshadowed by the more ubiquitous visualization, sonification design is unwillingly trapped in a vicious cycle: a lack of accessible sonification tools prevents sonification from achieving a more important role in data display, and the same is true vice versa. Thus, sonification design is usually a very involved process, requiring not just patience in its implementation due to its longer development cycle, but more importantly the requisite skills in computer science and music needed to wrangle data, design mappings between data attributes and auditory parameters to maximize expressiveness, and evaluate the resulting sonification for its effectiveness. This makes the threshold for designing sonifications inaccessible for novices, and a reflective process difficult for experts.

This work is licensed under Creative Commons Attribution – Non Commercial 4.0 International License. The full terms of the License are available at http://creativecommons.org/licenses/by-nc/4.0/

SIREN[1] is a web based digital audio workstation (DAW) for data sonification developed as a design tool to address this need. As a web application, SIREN has near-universal support, obviating the need for specialized software or setup to use. The interface, heavily inspired by DAWs, provide a familiar user experience that gently introduces them to the data sonification pipeline. The workflow allows for easy customizability, extensibility, and shareability—the last reinforced by way of the web—while balancing rapid prototyping. The modularization of many of SIREN's features encourages reusing components between sonifications, enabling complex sonification projects easily through community-supported modules.

## 2. GOALS

SIREN's design philosophy is informed by two goals: first, the democratization of sonification as a data display medium; second, the convenience it offers to users to prototype, create, explore, and share sonifications. The visualization that SIREN offers reflects the intricacies of a sonification in a minimalist and digestible manner, allowing new users to confirm their creations in a more familiar setting and allowing instantaneous evaluation in exploring and prototyping all aspects of sonification.

### 2.1. Democratization

Sonification through code offers enormous capability for its design; SIREN aims to maintain the ceiling of sonification while simultaneously lowering the threshold. By providing a no-code, interactive, DAW-like interface, SIREN unlocks sonification design to users with little to no experience with programming and/or musical knowledge. Moreover, the modularization of code-enabled features and the general program state of SIREN allow users to easily share more abstruse parts of sonification design, bypassing the need for novice users to code in order to develop more advanced creations.

### 2.2. Exploration

Another important principle that guides SIREN's design is the affordance it can give users in exploring sonification without hassle. While many tools have been developed in visualization design to aid in exploratory data analysis—including commodity tools like Microsoft Excel and Google Sheets as well as more domain-specific systems like Polaris, SocialAction, Jigsaw, PivotGraph, and Lineup just to name a few examples in the vast visualization

[1] Code available at `https://github.com/Kizjkre/siren`.

https://doi.org/10.21785/icad2023.6167

literature—this abundance does not transfer over to sonification [2, 3, 4, 5, 6]. Thus, SIREN aspires to be an application that users can use to conduct exploratory data analysis through sonification, as well as providing a platform to prototype and share sonifications with others rapidly.

## 3. RELATED WORK

SIREN has many predecessors, including MUSE, Sonification Sandbox, Rotator, xSonify, Interactive Sonification Toolkit, Sonification Workstation, SonART, MUSART, Sonifyer, MUSART, Highcharts Sonification Studio, and WebAudioXML and the WebAudioXML Sonification Toolkit, among others [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. SIREN also has a previous version of itself, but this new version is significantly upgraded and comes with a new set of guiding principles [18]. Each of the various sonification frameworks and applications will be discussed in relation to its adherence with SIREN's goals, will ultimately demonstrate the gap that SIREN fills. This section introduces existing sonification frameworks and applications and evaluates previous work using the criteria of democratization and exploration described in section 2.

### 3.1. MUSE

Developed in 1997, MUsical Sonification Environment (MUSE) is a parameter mapping sonification tool written in C++ that enables the musical sonification of scientific data. MUSE allows six auditory parameters to be mapped: timbre, rhythm, tempo, volume, pitch, and harmony. While this greatly simplifies the sonification creation workflow, the double-edged sword of lowering threshold also lowers the ceiling, as the user is limited to a subset in the entire palette of sonification techniques [19]. Thus, while it may allow for accessibility, exploratory use cases are limited [7].

### 3.2. Sonification Sandbox

Sonification Sandbox is another advancement in sonification tools. Designed as a Java-based cross-platform application, Sonification Sandbox gives users a myriad of features, including mappings and encodings. Encoding channels exist between the data attribute and four auditory parameters: pitch, timbre, gain, and pan. Sonification Sandbox's point of innovation is its emphasis on context, which enables customization of contextual sonification that remind listeners of the important metadata of the sonification during the sonification. An auditory graph is also presented to the user that shows the sonification. Despite the wealth of features, mapping functionality is limited to the widgets that are provided by Sonification Sandbox, and the restricted amount of auditory parameters limit the variety of sonifications that Sonification Sandbox supports [8].

### 3.3. Rotator

Like SIREN, Rotator is a web application for sonification. However, its main objective is to enable multisensory data exploration through distribution of data visually and aurally. Rotator is designed as an expert user interface with regards to experience in sonification with different goals, and thus targets a different user population [9].

### 3.4. xSonify

xSonify is a sonification framework developed to increase accessibility for the exploration of space physics data. Created in Java, xSonify targets users who are visually-impaired by allowing them to explore data through sound. The main limitation of xSonify is the restriction of unidimensional data, which undersells the potential of parameter mapping sonification as an expressive data display method [10].

### 3.5. Interactive Sonification Toolkit

Interactive Sonification Toolkit is a native application that encourages users to interact with various sonification algorithms through a GUI. With a multitude of supported sonification types and parameters, including audification, pitch, additive synthesis, filters, and many others, Interactive Sonification Toolkit abstracts much of the programming while still allowing for varied sonification techniques. However, the mapping functionality is limited to clamping, scaling, and transposing. Furthermore, as a native application, Interactive Sonification Toolkit loses out on accessibility due to the web's platform-agnostic nature and instantaneous setup [11].

### 3.6. Sonification Workstation

Sonification Workstation is a recent native application developed using the QT framework for sonification. Similar to SIREN, it adopts a DAW metaphor and aims to provide accessibility to sophisticated sonification techniques. Similarly, an argument against native applications outlined in 3.5 can be applied here. Moreover, the custom synthesis components limits experts' ability to use existing patches or modules and math-based mapping limits the data types that Sonification Workstation can support [12].

### 3.7. SonART

Sonification Application and Research Toolbox (SonART) is a C++ application built on the Synthesis Took Kit (STK) for sonification. The interface features a parameter engine matrix that allow users to define connections between the data and the parameters. Unlike many of the applications described in this section, SonART gives users the ability to extend its capabilities through the STK [13]

### 3.8. Sonifyer

Sonifyer is a OS X-only application that aims to provide accessibility in sonification for novice users. Similar to SIREN, the authors also propose a forum based on Sonifyer projects where sonifications can be shared. One of the biggest drawbacks in the inaccessibility of Sonifyer as a strict one-platform application. Thus, what is done to democratize sonification technology is made more difficult by its lack of support [14].

### 3.9. MUSART

MUSical Audio transfer function Realtime Toolkit (MUSART) is a C++ application for multivariate melodic sonifications through audio transfer functions. To create melodic sonifications, MUSART bases its sonifications on Western music theory, and is thus able to produce sonifications able to be recorded through Western music notation. However, Western music theory is only a small subset of all possible sounds and music traditions, and marginalizing

https://doi.org/10.21785/icad2023.6167

users unfamiliar with Western music traditions. Furthermore, sonification can produce a wide variety of sounds not classified under Western music traditions, which also limits the set of sonifications that can be produced with MUSART [15].

### 3.10. Highcharts Sonification Studio

Highcharts Sonification Studio (HSS) is a relatively new sonification application developed in collaboration between Highcharts and Georgia Institute of Technology's Sonification Lab. Developed as a web application, HSS. like SIREN, is able to take advantage of the benefits provided by a web application. It is important to note that SIREN and HSS have some common goals: as described in Cantrell et al.'s paper, HSS prioritized accessibility of the application towards a broad population encompassing students to researchers. Its layout is split into two main views: "Data" and "Chart." The "Data" view encapsulates a spreadsheet display of the data, and the "Chart" view shows a graph of the data. More importantly, the "Chart" view also contains the settings and parameters for the actual sonification, allowing users to control global audio settings and mappings for each data series via the right sidebar. However, HSS imposes some limitations on the sonification ability of the user. For example, timbres are limited to a subset of sounds, with only a strictly defined eight other auditory parameters to control. Within those auditory parameters, mapping is limited to a linear scaling of the data series that it is acting on [16].

### 3.11. WebAudioXML Sonification Toolkit

One of the newer applications among the list, the WebAudioXML Sonification Toolkit (WAST) is a web application that provides parameter mapping sonification using the WebAudioXML schema. It received positive evaluations when the application was used to introduce students to parameter mapping sonification, thus confirming its ability to provide accessible interface for sonification. Despite the benefits of WAST, the authors note areas of future research and development, including the integration of data wrangling tools within the interface as well as mapping functions and better visual guidance [17].

### 3.12. SIREN

SIREN as presented in the first paper is a much different application. The old guiding philosophy relied on tracks and channels as an analogy to tracks in DAWs, where a data attribute represented a track and a channel represented a group of tracks. As an upgraded version, the new SIREN focuses on tracks and regions, which are more robust and customizable than tracks and channels. The improvements in mapping design and the added ability to create custom synths add to the novelty of the new version of SIREN [18].

### 3.13. Comparisons to SIREN

All of these sonification applications have their own unique ways of lowering the difficulty of sonification design. Similarly, SIREN takes its own approach as well. While comparisons to SIREN will be made here, details of its implementation will be documented at length in section 4 below.

One of the major points of innovation for SIREN is its finer control of sonified data. Tracks and regions, explained in sections 4.1.1 and 4.1.2, allow users to define exactly when sonifications start, as well as the freedom to stack and layer the various data attributes over different times and auditory parameters, allowing different series controlling different auditory parameters to come together to create one sonification. Furthermore, builtin data wrangling tools give users the ability to highlight different aspects of the data attribute all within the same sonification. Finally, with the modularized synthesizer components, users are not limited to a developer-designated pool of timbres and sounds when creating sonifications.

## 4. METHODOLOGY

To unify the goals with the avenues of innovation, SIREN implements a design philosophy centered around tracks and regions. These concepts are then implemented as a web application. An evaluation is planned to test the efficacy of SIREN in achieving the goals set out in section 2.

### 4.1. Design

Inheriting the DAW metaphor, SIREN's design revolves around tracks and regions, analogous to the tracks and regions of DAWs. However, there are crucial differences between the two where SIREN adapts these concepts specifically for data sonification.

#### *4.1.1. Tracks*

Figure 4 illustrates what tracks are in SIREN. Similar to those on DAWs, a track occupies a horizontal length that spans the length of the entire sonification. The box on the left is the track header, which contains information about the track as well as interactive elements that change various attributes of the track. Each track represents one instance of a synth with mappable auditory parameters. The track displays one auditory parameter at a time to minimize clutter, though all auditory parameters are active within a track. Regions (described in section 4.1.2) from any attached dataset can be dragged into a track and positioned within the track at any time. A track may have any number of regions in any view (auditory parameter) as long as regions do not overlap.

#### *4.1.2. Regions*

Regions are akin to regions in DAW, however in SIREN they are defined by the cells of a data attribute in a dataset. Regions only exist within a track, and can be placed anywhere in the track temporally. Within a track, a region determines the value of the auditory parameter that it is placed on. Regions, and its corresponding track parameter, must be of the same data type as defined by Stanley Stevens: nominal, ordinal, or quantitative [20]. The user can define custom mappings (described in section 4.1.4) to convert between data types, but SIREN also provides automated conversion via the "Default" mapping function seen at the bottom of the sidebar in figure 1. As seen in figure 4, regions in SIREN are placed under the timeline within tracks denoted by the gray background that represent the horizontal (temporal) extent of the region.

#### *4.1.3. Synths*

Synths are the vehicle that converts data to sound in the SIREN workflow. Users can create or use community-made synths as a part of their SIREN project. Synths are made in JavaScript

https://doi.org/10.21785/icad2023.6167

using the Web Audio API, and with minimal processing, can be used in SIREN. Figure 2 shows a synthesizer programmed in JavaScript that can connect to SIREN. The modularization of synths allow them to be easily developed with basic familiarity in JavaScript and the Web Audio API as well as allowing it to be shareable among projects and users. Users can also implement filtering through synthesizers, such as using the Web Audio API's `BiquadFilterNode` in the synth design.

#### *4.1.4. Mappings*

Mapping functions are a crucial part of sonification design. To this end, SIREN also incorporates this important step in its workflow. With SIREN's mapping capabilities, users can filter and map data as well as transform between data types. Figure 5 shows SIREN's interface with the mapping window open. Functions are written in JavaScript with the definition `(x: any, i: Number, array: any[]) => any`. `x` is the datum that the function is operating on, `i` is the index of the datum, and `array` the array of items of the region. Returning `undefined` indicates exclusion and is thus filtered out by SIREN. Mappings, like synths, are modularized within SIREN's code, and can thus be easily shared across users and projects.

#### *4.1.5. Visual Design*

Visual design is paramount to the accessibility of an application. Thus, SIREN is meticulously designed to be familiar, clean, and easy to use. Drawing on the DAW metaphor, SIREN's overall layout is spaced much like that of a DAW: figure 1 shows the various components of SIREN, including the toolbar at the top, the sidebar on the side, the main body in the middle, and the controls on the bottom. The design also mimics a native application, adding to the immersion of its usage, rectifying one of the main drawbacks of a web-based application. The visualizations of tracks and regions also aligns with the resulting sonification, with mappings visually affecting the positions of the datum in the region, which lead to faster development time for creating sonifications as well as bridging a gap between the vernacular visualizations and less-familiar sonifications for new users. To really emphasize the visual-aural connection, a red seeker seen in figure 1 allow users to see their sonification in action. The result is a WYSIWYG interface: the user is able to visualize exactly the progression of the sonification throughout time, and track it as the sonification is playing.

### 4.2. Implementation

As a web application, SIREN uses many modern web frameworks, libraries, and APIs. First and foremost, SIREN is a full-stack application built on MongoDB, Express.js, Solid.js, and Node.js. Other libraries used include Tailwind CSS for styling, d3.js for data wrangling, and Ace Editor to power in the in-app synth and mapping creation tools. Icons were sourced from Heroicons and Tabler Icons.[2] Most importantly, the Web Audio API is used extensively to create the actual sonification.

### 4.3. Evaluation

Evaluation of SIREN is a crucial step in empirically determining the efficacy of SIREN's design in achieving the goals listed in section 2. To this end, we plan to conduct two user studies, each focusing on one of the two goals. Since the two goals target vastly different populations, we have designed the evaluation differently for both.

#### *4.3.1. Evaluating Adherence to Democratization*

First, we plan to evaluate SIREN's ability to lower the threshold of sonification creation and design. The participants will be recruited through an introductory seminar course at Stanford University titled "MUSIC 15N: The Aesthetics of Data." Through this course, students learn about sonification and various sonification methods through a seminar style lecture and hands-on projects. By introducing SIREN as a sonification tool, we hope that students who use the application for early projects will be able to focus more on the quality of the sonification rather than dealing with the minutiae of debugging. To this end, we will have students engage in a pre-evaluation questionnaire as well as a post-evaluation reflection. The questionnaire will deal with their existing knowledge of sonification, programming, and music. Then, students will use SIREN as part of their homework and create sonifications using the application. After the evaluation is completed, students will then be asked to fill out a reflection that asks about the threshold and ceiling of the application as they used it. Specifically, we plan to target difficulties that students experienced when working with SIREN as well as features that they wished they had to further simplify their process of creating sonifications. The reflection will also include a section regarding the role of SIREN in assisting and facilitating their learning.

#### *4.3.2. Evaluating Adherence to Exploration*

On the other hand, SIREN's other goal is to ensure that the applications is still robust enough to support more advanced sonification creation. For this evaluation, we plan to recruit musicians and those familiar with sonification from the community, and invite them, with compensation, to a session where they can test out SIREN. For these users, the paramount question is whether or not users were able to create a desired sonification using SIREN, and if it was done in a way that would be faster than manual data wrangling and programming. Similar to the evaluation outline in section 4.3.1, there will be an initial survey to help contextualize evaluators' experience and familiarity with sonification, as well as a final reflection where they can share their opinions on how well they think SIREN was able to help them speed up the sonification design process. Throughout the evaluation, we will encourage pariticpants to think out loud as they navigate the application, which will provide invaluable insight into their thought process and give crucial feedback to the usability of the system.

## 5. USAGE

This section describes the workflow for a generic sonification project using SIREN. After working through the steps, the user can listen to the sonification using the play, pause, and stop controls at the bottom. To save the sonification and/or the project, the user can click on the "File" menu in the toolbar shown in figure 1, and click "Save" and/or "Export .wav."

[2] Specific implementation details can be found in the project's GitHub repository: `https://github.com/Kizjkre/siren/`.

https://doi.org/10.21785/icad2023.6167

### 5.1. Adding Datasets

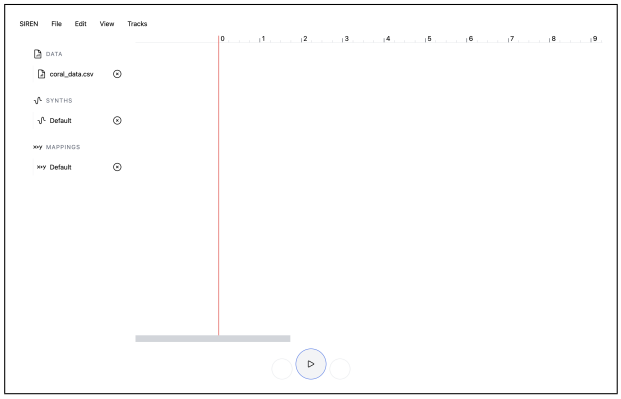



Figure 1: SIREN's home view.

SIREN comes prebuilt with a sample dataset, but users can choose to include their own dataset using the "File" menu shown in figure 1 and then clicking on the "Import CSV" item.

### 5.2. Creating Synths

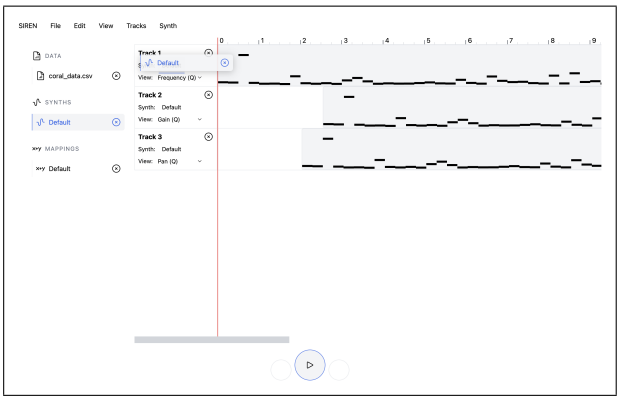
```
1  export const parameters = Object.freeze({
2    time: ['Time'],
3    timbral: ['Frequency', 'Gain', 'Pan']
4  });
5
6  export default () => {
7    const context = new AudioContext();
8    const osc = new OscillatorNode(context);
9    const gain = new GainNode(context);
10   const pan = new StereoPannerNode(context);
11
12   gain.connect(context.destination);
13   pan.connect(gain);
14   osc.connect(pan);
15
16   return ({
17     context,
18     updates: new Map()
19       .set(
20         ['Frequency', 'Time'],
21         (f = 440, t = 0) => osc.frequency.setValueAtTime(f, t)
22       )
23       .set(
24         ['Gain', 'Time'],
25         (g = 1, t = 0) => gain.gain.setValueAtTime(g, t)
26       )
27       .set(
28         ['Pan', 'Time'],
29         (p = 0, t = 0) => pan.pan.setValueAtTime(p, t)
30       ),
31     start: () => osc.start(),
32     stop: () => osc.stop()
33   });
34 };
```

Figure 2: Full code snippet of a synthesizer.

Figure 2 shows a code snippet of a synthesizer with parameters "Frequency," "Gain," "Pan," and "Time." Lines 7–14 create the actual synthesizer, and lines 16–33 play the synthesizer according to the value of the parameters. To repurpose a Web Audio synthesizer to work with SIREN, all the user would need to do is to export an object `parameters` that has the two keys `timbral` and `time`, each an array of the names of the parameters according to their type of being a time-based or non-time-based parameter, `export default` a function around the synthesizer, and return an object containing the keys `context`, `updates`, `start`, `pause`, and `stop`. The details of each entry is described below:

- `context`: the `AudioContext` that the synthesizer is built on.
- `updates`: a JavaScript `Map` object that contains the names of the changed parameters as the key, and a function containing the parameters in the order presented in the key as arguments with default values that updates the synthesizer's state as the value. The default value is used in cases where there is no region at the time automating the parameter during the sonification.
- `start`: a function `() => void` that starts the synthesizer.
- `pause`: a function `() => void` that pauses the synthesizer.
- `stop`: a function `() => void` that stops the synthesizer.

Synths can then be connected to tracks via a drag-and-drop interface: by dragging the synth from the sidebar to the synth name in the track header, the user can change the synths that the track instantiates. This will also change the parameters that can be shown on a track, as every synth are likely to have its own set of parameters.




(a) Dragging and dropping a synth to a track.

(b) Dragging and dropping a mapping to a region.

Figure 3: SIREN's drag-and-drop interface.

https://doi.org/10.21785/icad2023.6167

### 5.3. Creating Tracks and Regions

Figure 4: SIREN's workstation populated with tracks and regions on the tracks.

The next step in SIREN's workflow is creating a track. Tracks can be added using the "Track" menu from the toolbar, from where a new track will appear under the timeline in the main body of SIREN's interface. Users can then add regions to the track. To do so, users click on a dataset from the sidebar, which unfolds all of the attributes from the dataset in the sidebar shown in figure 4 with the dataset `coral_data.csv`. Users can then drag attributes to the track body, which will place the attribute and all data associated it as a region, shown as the three gray regions in figure 4. Meanwhile, users can rename the track by clicking and editing the boldfaced name in the track header, as well as changing the synth and the view. Views are associated with the various parameters of the synthesizer connected to the track, and may vary depending on usage.

### 5.4. Creating and Applying Mappings

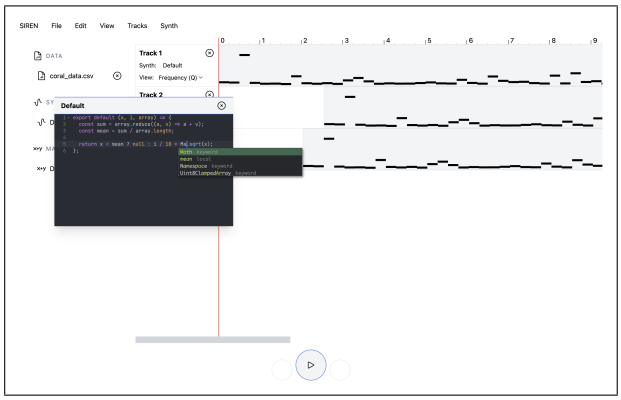



Figure 5: The mapping window where users can define mappings and change data attribute types.

Mappings can then be created from the window, where users can write a JavaScript function that specifies the filtering and mapping operations. These mapping functions are reusable and can be applied to any region across any SIREN project. After the user writes the mapping function, they can drag it from the sidebar onto the region that they want to apply the mapping to. The mapping should return one of the three data types (nominal, ordinal, quantitative), and for best practices, should be applied to regions whose parameters are of the same type.

## 6. USE CASES

SIREN has a variety of purposes, from exploratory to educational. Related to the goals of SIREN, democratization of sonification techniques can be used to serve a pedagogical role, where SIREN is used to introduce students to sonification without the burden or focus on learning programming languages or libraries. This would liberate student's attention to the intricacies of the medium and instead enable them to explore the flexibility of sonification. Moreover, SIREN's clear workflow allow novice users to develop a clear mindset about the nature of designing sonifications. The other important goal of SIREN is to optimize the process of designing sonifications for prototyping, exploration, or creating. Uses cases in this domain are clear: improving the user's ability to create sonifications lead to faster iteration of quantity over quality, which has beneficial impacts for creating overall better sonifications [21, 22].

## 7. CONCLUSION AND FUTURE WORK

SIREN is a powerful sonification tool; nonetheless, there are aspects of SIREN that can be improved to further provide accessibility and lower the threshold. This section describes future work for SIREN, as well as closing thoughts of SIREN as a platform that advances the ability of sonification creation through its accessibility.

### 7.1. Future Work

Though SIREN already incorporates many features, there are still improvements that can be made. First, SIREN can better take advantage of server-side capabilities that allow users to more easily share projects, datasets, synths, mappings, and sonifications. This would entail designing a more robust backend that can deliver the various components of a SIREN project as well as user authentication. The frontend would also have to adapt by adding interactions that allow users to choose community-created components when adding synths, datasets, or mappings to a project. Moreover, the ChucK language for music programming recently released a web-based version called WebChucK [23]. WebChucK integration as a synth creation module can also be a powerful tool for creating sonifications. This can be further bolstered by the implementation of a mini synth IDE that includes features tailored to Web Audio and WebChucK, such as with visualizable audio graphs. Furthermore, user studies should be conducted to verify SIREN's design in achieving its goals. SIREN would be evaluated on two groups of users—novices and experts—that better inform SIREN as an application that lowers the threshold of parameter mapping sonification design (as planned in section 4.3).

### 7.2. Conclusion

This paper presented SIREN, a flexible, extensible, and shareable interface for creating sonifications. Through the various design

https://doi.org/10.21785/icad2023.6167

choices, SIREN aims to mitigate the gap in accessibility with designing sonifications while providing a platform for intricate designs that experts can feel comfortable experimenting within. Hermann, Hunt, and Neuhoff describe a future where they "envision established and standardized sonification techniques, optimized for certain analysis tasks, being available as naturally as today's mouse and keyboard interface" [24, p. 4]. SIREN as a tool for the democratization and facilitation of parameter mapping sonification works towards that vision of ubiquity.

## 8. REFERENCES


[1] D. A. Schön, *The reflective practitioner: how professionals think in action*, paperback ed ed. Aldershot: Avebury, 1991.

[2] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: a system for query, analysis, and visualization of multidimensional relational databases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, p. 52–65, Jan 2002.

[3] M. Wattenberg, "Visual exploration of multivariate graphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06. New York, NY, USA: Association for Computing Machinery, Apr 2006, p. 811–819. [Online]. Available: https://doi.org/10.1145/1124772.1124891

[4] A. Perer and B. Shneiderman, "Balancing systematic and flexible exploration of social networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, p. 693–700, Sep 2006.

[5] J. Stasko, C. Görg, and Z. Liu, "Jigsaw: Supporting investigative analysis through interactive visualization," *Information Visualization*, vol. 7, no. 2, p. 118–132, Jun 2008. [Online]. Available: http://journals.sagepub.com/doi/10.1057/palgrave.ivs.9500180

[6] S. Gratzl, A. Lex, N. Gehlenborg, H. Pfister, and M. Streit, "Lineup: Visual analysis of multi-attribute rankings," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, p. 2277–2286, Dec 2013.

[7] S. K. Lodha, J. Beahan, T. Heppe, A. Joseph, and B. Zane-Ulman, "Muse: A musical data sonification toolkit," Nov 1997. [Online]. Available: https://smartech.gatech.edu/handle/1853/50750

[8] B. N. Walker and J. T. Cothran, "Sonification sandbox: A graphical toolkit for auditory graphs," Jul 2003. [Online]. Available: https://smartech.gatech.edu/handle/1853/50490

[9] J. Cherston and J. A. Paradiso, "Rotator: Flexible distribution of data across sensory channels," Jun 2017. [Online]. Available: https://smartech.gatech.edu/handle/1853/58351

[10] R. M. Candey, A. M. Schertenleib, and W. L. Diaz Merced, "Xsonify sonification tool for space physics," Jun 2006. [Online]. Available: https://smartech.gatech.edu/handle/1853/50697

[11] S. Pauletto and A. Hunt, "A toolkit for interactive sonification," Jul 2004. [Online]. Available: https://smartech.gatech.edu/handle/1853/50827

[12] S. Phillips and A. Cabrera, "Sonification workstation," Jun 2019. [Online]. Available: https://smartech.gatech.edu/handle/1853/61529

[13] O. Ben-Tal, J. Berger, B. Cook, M. Daniels, and G. Scavone, "Sonart: The sonification application research toolbox," Jul 2002. [Online]. Available: https://smartech.gatech.edu/handle/1853/51376

[14] F. Dombois, "Sonifyer a concept, a software, a platform," Jun 2008. [Online]. Available: https://smartech.gatech.edu/handle/1853/49949

[15] A. J. Joseph and S. K. Lodha, "Musart: Musical audio transfer function real-time toolkit," Jul 2002. [Online]. Available: https://smartech.gatech.edu/handle/1853/51366

[16] S. J. Cantrell, B. N. Walker, and Ø. Moseng, "Highcharts sonification studio: An online, open-source, extensible, and accessible data sonification tool," in *Proceedings of the 26th International Conference on Auditory Display (ICAD 2021)*. Virtual Conference: International Community for Auditory Display, Jun 2021, p. 210–216. [Online]. Available: http://hdl.handle.net/1853/66348

[17] H. Lindetorp and K. Falkenberg, "Sonification for everyone everywhere: Evaluating the webaudioxml sonification toolkit for browsers," Jun 2021. [Online]. Available: https://smartech.gatech.edu/handle/1853/66351

[18] T. Peng and H. Choi, "Siren: A case study in web audio based sonification," Jun 2021. [Online]. Available: https://smartech.gatech.edu/handle/1853/66345

[19] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, p. 3–28, Mar 2000. [Online]. Available: https://dl.acm.org/doi/10.1145/344949.344959

[20] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, p. 677–680, Jun 1946. [Online]. Available: https://www.science.org/doi/10.1126/science.103.2684.677

[21] D. Bayles and T. Orland, *Art & fear: observations on the perils (and rewards) of artmaking*. Santa Barbara, CA: Capra, 1993.

[22] S. P. Dow, A. Glassco, J. Kass, M. Schwarz, D. L. Schwartz, and S. R. Klemmer, "Parallel prototyping leads to better design results, more divergence, and increased self-efficacy," *ACM Transactions on Computer-Human Interaction*, vol. 17, no. 4, pp. 18:1–18:24, Dec 2011. [Online]. Available: https://doi.org/10.1145/1879831.1879836

[23] G. Wang and P. Cook, "Chuck: a programming language for on-the-fly, real-time audio synthesis and multimedia," in *Proceedings of the 12th annual ACM international conference on Multimedia*, ser. MULTIMEDIA '04. New York, NY, USA: Association for Computing Machinery, Oct 2004, p. 812–815. [Online]. Available: https://doi.org/10.1145/1027527.1027716

[24] *The sonification handbook*. Berlin: Logos Verlag, 2011.