MADBPM: MUSICAL AND AUDITORY DISPLAY FOR BIOLOGICAL PREDICTIVE MODELING

K. Michael Fox, Jeremy Stewart, Rob Hamilton

Rensselaer Polytechnic Institute
Department of the Arts
110 Eighth Street
Troy, New York 12180
{foxk, stewaj5, hamilr4}@rpi.edu

ABSTRACT

The modeling of biological data can be carried out using structured sound and musical process in conjunction with integrated visualizations. With a future goal of improving the speed and accuracy of techniques currently in use for the production of synthetic high value chemicals through the greater understanding of data sets, the *madBPM* project couples real-time audio synthesis and visual rendering with a highly flexible data-ingestion engine. Each component of the *madBPM* system is modular, allowing for customization of audio, visual and data-based processing.

1. INTRODUCTION

Data sonification is a rapidly evolving discipline that explores the use of data representing information from scientific processes, analytic models or real-time tracking as the drivers for sound generating systems. The goal of many sonification systems is the conveyance and perception of specific information related to source data as realized using sound as the principal component: an auditory display. In the same ways that computer-driven visualizations of datasets create visual analogues to the complex relationships between variables and data abstractions, sonifications represent data as audible constructs existing in time. For situations involving real-time multi-modal monitoring of continuous data streams, auditory displays working in conjunction with visual displays can provide additional attentional bandwidth for tasks ranging from the encoding of avatar motion and action in virtual space [1, 2] to correcting athletes' golf swings [3].

When sound is put to use as an expression of structured data derived from a functional process, a significant transformational and translational component is necessary to map parameters from this functional realm to parameters of sound that have the potential to express and expose specific relationships of import within the data. As audible sound itself is perceptually comprised of a great number of parameters ranging from amplitude to frequency to timbre to location in space, the organization of sounds and the organization of the parameter mappings that are associated with sound become crucial when attempting to create meaningful sonifications of complex data sets, themselves representative of complex multivariate processes. Such aesthetic and artistic exploration of data has great promise in driving new research paradigms as has been seen in recent projects begun as artistic initiatives such as Stanford University's "Brain Stethoscope" — a portable device that detects the onset of epileptic seizures through the real-time rhythmic and harmonic musical sonification of pre- and post-ictal neural activity [4].

In this light, musical sonification can be understood as the application of organizational techniques commonly attributed to musical composition and performance towards the grouping and structuring of sound generated from a data source. Musical characteristics such as pitch, rhythm, melody, harmony, timbre, tempo, spatialization, attack, decay and intensity can be mapped to raw parameters or calculated attributes of datasets, allowing for the creation of sonic output that we hear as musical in nature while still capable of conveying a great deal of information. Taken one step further, musical structures, such as the transformation of organized or generated note material, or the application of continuous changes to instrumental parameters responsible for shaping the arc of a musical phrase can also be driven by analyzed data.

Application of musical structures and techniques in this mapping process is experimental and compositional. Compositional process focuses at different levels of detail at different points in that process, ranging from low-level note-to-note attention to high-level structural attention. Similarly, sonification can range from mapping sound directly on discrete data points parsed over time to sonification of procedures in transformational algorithms. Bovermann et al. [6] further distinguish between the sonification of algorithms by juxtaposing merely inputs and outputs versus inputs, outputs, and all intra-algorithmic transformational steps between. They also propose a distinction for *operator based sonification*, where scientific models are directly embedded in sonic mapping functions.

The importance of these distinctions becomes particularly clear when comparing sonification of processed, structured data vs large sets of unstructured data. In the case of unstructured data, there may not be an inherent model with which to scaffold sonic or visual mappings and any data traversal method becomes an even more significant force in sonic signification. Significant questions around software platforms for sonification and visualization include whether the platforms are used for research-oriented exploration or post-research display of findings (presentation). Another question is that of whether the end-user is seeking aesthetic exploration (artistic) or empirical knowledge (scientific) or some mix of the two. An ideal software toolkit for data perceptualization would

https://doi.org/10.21785/icad2017.045

Figure 1: An example of visualization in *madBPM*.

allow for productive research and experiments by laboratories and artists, while also allowing that research to be presented live at professional meetings and artistic performances. The authors have produced a software toolkit and model for data perceptualization that emphasizes user-defined "behavioral abstractions" to improve sonification software flexibility and extensibility. This model has been implemented in the MadBPM software platform to create a unified research environment for both creative and analytical explorations of data through perceptualization.

2. MADBPM

The *madBPM* software platform is built around a specific model that emphasizes actions and procedures. Sonfication and visualization in this platform is realized by end-users who define code-based objects that describe data-flow and logic in the traversal of data and mapping to sound or visuals. The sound synthesis is provided by the SuperCollider sound and music programming language [5], but the modular design allows for different backends capable of Open Sound Control messaging to be used. The software is written in C++ and relies on openFrameworks[1] for the visualization functionality.

We assume that musical structure and form can be utilized to not only represent characteristics of biological processes but more importantly also to aid researchers in discovering potentially interesting and important relationships previously hidden within complex data sets. From this assumption, we designed *madBPM*, a modular sonification and visualization platform that allows for the rapid prototyping and display of sonic and visual mappings. Initially developed for a project focused on the identification of key biological data points within the process of biosynthesis for high value chemicals, *madBPM* was designed as a modular toolkit, capable of interfacing with existing audio engines, visual coding languages and customized data ingestion modules. In its current state *madBPM* is linked to the *SuperCollider* [SC] sound and music programming language [5] and the *openFrameworks* visual programming language[2].

In the following sections, we describe some of the most important architectural features of the *madBPM* software environment and data perceptualization model. These features are described in the context of the original research project from which the software environment emerged.

2.1. Data Perceptualization in *madBPM*

Auditory and visual mappings from datasets are experimentally and contextually derived. These mappings are further impacted by the initial state of the data being mapped—for instance, whether the dataset contains errors or invalid data-points. Mappings may need to account for these, or the data may need to be pre-filtered before sonification and visualization. Software environments can position one or the other approach as always necessary by design, precluding the use of unstructured data sets. *madBPM* is designed to allow for flexibility in the kinds of data sets that might be processed in the environment by emphasizing its three layers of behavioral abstractions: 1) *program-level logic*, 2) *data traver-*

[1] `http://www.openframeworks.cc`

[2] `http://openframeworks.cc`

sal/parsing, and 3) *audio/visual mapping*. In the *madBPM* software environment researchers, artists, or other users generate results by defining transformational schemes at each of these three levels of abstraction. In the last layer, audio/visual mapping, data is transformed into parameters of sonic and visual events. The second layer defines schemata for data-sets to be algorithmically traversed. In the final and "top" layer, changes in the two lower layers of abstraction can be automated. Each of these layers is described more thoroughly in section 3.

2.2. Software Environment

madBPM makes extensive use of *openFrameworks* in its architecture. Users of the platform are initially presented with a lean graphical user interface [GUI] comprised of three key components (Fig. 2). The first of the components is a pane displaying the structured data files automatically loaded at startup, described in Section 2.3. Each entry in this pane represents a "tag" that describes a subsection of the data. Clicking on tags that appear in this list selects and highlights that tag, while leaving tags that exist within that subset selectable and making tags not represented in the subset unselectable (grey, non-clickable).

The second component, at center screen, is a GUI panel that displays the console output of the managed SC process. This panel allows users to get reference information from SC or debug unexpected behavior from within the platform during any development or testing.

A third component of the platform interface features a "utility bar" like structure near the bottom of the screen. The GUI is extensible from source code and the utility bar is a potential non-intrusive spot for buttons or short-hand reference information. Currently, the bar features a color indicator box representing the connection state with SC, a Frames Per Second meter, and a button "new collection". Once a series of tags have been selected from the first GUI component (the tag list), activating the "new collection" button triggers the data from those corresponding files to be combined into a collection (ordered set) for sonification and visualization. Visualization is drawn behind the GUI, which can be hidden by a hotkey.

2.3. Structure of Data

Data is currently read by the platform from partially pre-processed CSV files. These files provide both raw data in labeled columns, but also tags that identify the relationship between each file. From the research aims of the initial phases of the project, these files communicate analyzed information from lanes on an electrophoresis gel image. The labels that identify the gel image and each lane within that image are the tags that are applied to files and subsets of data within those files.

When *madBPM* reads these files, they are organized internally into "gel_Lane" objects holding the raw data and describing their tag relationships to the platform. After tag filters have been applied and a new collection is constructed, all internally stored gel_Lanes matching the query are aggregated into a "gel_Collection" object that is used by the platform for perceptualization. "gel_Collection" objects present themselves to the platform like a multidimensional iterator that are held by a Ranger object and parsed by Walker objects (Section 3). Once gel_Lane objects are stored in memory, each gel_Lane object merely wrap immutable data and are used to create user-defined gel_Collection objects.

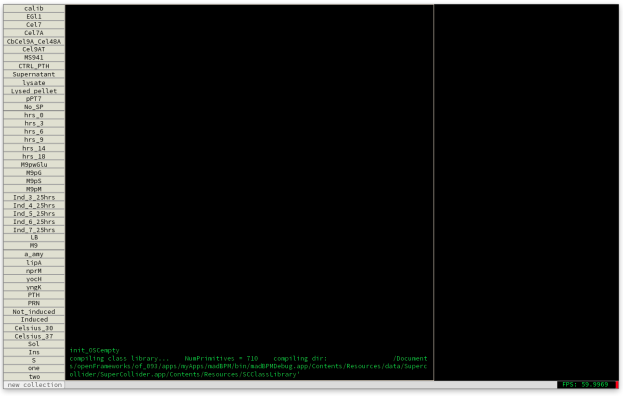

Figure 2: *madBPM* GUI at startup. At left, the tagged data filtering pane. At center is the console output of *SuperCollider*. At the bottom is the "utility bar", featuring a button to build an active collection from the selected data tags, a Frames Per Second meter, and, at far right, a color status icon indicating whether the platform has successfully initialized *SuperCollider*.

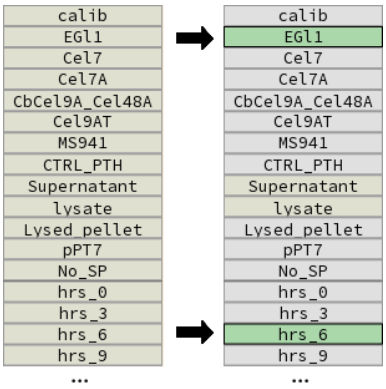

Figure 3: Detail of the tag list selection and filtering. Selecting tags filters out data that do not exist in all selected categories. Grey tags do not appear in the specified subset, while yellow tags may be selected to further constrain the set.

2.4. Sound Backend API

While the structure of visualization mappings are completely internal to the C++ application core, sonification mappings are represented within *madBPM* as objects in two ways: Sounder objects in the platform's C++ source and corresponding sibling Sounder objects in SC source. Sounder objects are encapsulations of specific algorithmic behaviors for the transformation of data into sonic or musical material. To facilitate the communication between the platform's internal representation and the SC backend representation, all data is transferred via Open Sound Control UDP messages and conforms to a strict API. Messages are directed to an address, which may be on the same computer or on another machine across a network, and are constructed as a command string followed by corresponding arguments. All arguments after the command specifier are tagged by preceding the argument with a string descriptor beginning with a ':' character.

The application level commands sent from the core platform to SC are comprised of:

- `loadSynthDefs`
  Load the platform's SC synthesis definitions.
- `create` {`:cls :id`}
  Create a new Sounder object of type `:cls` with class-specific unique `:id`.
- `updateParams` {`:cls :id :vol :rf :rv :mw`}
  Update the Sounder object of type `:cls` and `:id` with the values `:vol`, `:rf`, `:rv`, and `:mw`.
- `remove` {`:cls :id`}
  Remove the Sounder object of class `:cls` and with `:id`.
- `shutdown`
  Stop all active sounders and shutdown the process.
- `funcDefinition` {`:cls :sel`}
  Reply to *madBPM* front-end with a description of `:cls`'s function named `:sel`.

3. BEHAVIOR AND STRUCTURE IN *MADBPM* PROGRAMS

The most important design feature of the *madBPM* platform is that of *behaviors*. These are emphasized in three key levels of abstraction: program-level logic, data-parsing behavior, and data perceptualization algorithms or mappings. Each of these levels of behavior are represented within the platform as objects which describe their function over time (Fig. 5). From the beginning, *madBPM* was intended to aid in lab research data-oriented artistic inquiry, but also to enable both real-time professional presentation and artistic performance. Each of these levels of behavioral abstraction aim to address high and low level structural concerns for any of these contexts. Users define the collections, subsets of the data based on selected tags, to be sonified and visualized, and these collections are passed to the lower levels of the behavioral object hierarchy. By asking users to explicitly define the data parsing and meta structure (program-level logic), the platform is flexible enough to allow work with both un- or pre-processed datasets, structured or unstructured data, or multiple forms of data segmentation and tagging. At the current state of the project, these objects are still defined in C++ source code and compiled into the platform.

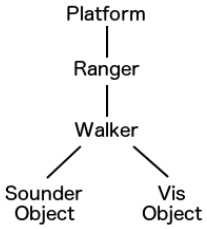

Figure 5: The behavioral object hierarchy. At the top level, the platform references a Ranger object, which defines a "program-like logic" executing over time. Ranger objects own at least one Walker object, each parsing data collections. Walkers communicate the data they parse to the Sounder objects and Vis objects they own.

3.1. Sounder Objects and Vis Objects

At the lowest level of the behavioral hierarchy are Sounder objects and Vis objects. These objects define specific mappings and algorithms for the transformation of data into visual elements on the screen or sound through speakers. These objects do not traverse data, nor do they define the rate at which data is accessed. These objects receive data from parent (Walker) objects and respond to them according to their defined behavior. All Sounder objects are polymorphic relatives of a base Sounder class, while Vis objects are similarly related to a VisObject base class. Both classes receive data from and interface with parents in identical ways: Sounders and Vis objects respond to their parent Walker object's call to an update function that accepts all relevant perceptualization data. Since Sounder and Vis objects encapsulate self-contained visualization and sonification algorithms, these objects may range from simple one-to-one mappings to much more complex real-time statistical models.

3.2. Walker Objects

Walker objects are specifically encapsulated defined behaviors for iteratively parsing gel_Collections they reference. Example behaviors might include: forward sequential traversal, visiting each gel_lane in sequence and every value in the lane; reverse sequential traversal, opposite of forward sequential; minimum to maximum traversal, visiting elements across each or all lanes from lowest value to highest; or, selective traversal, visiting every lane in the collection and updates Sounders and Vis objects only for certain values.

In the hierarchy of behavioral objects, Walkers communicate the values they visit in the data with the Sounder and Vis objects they have references to.

3.3. Ranger Objects

Ranger objects encapsulate "program-like logical structure", and interface directly with the platform and Walker objects they own. Ranger objects are analogous to the role of a musical ensemble conductor. Typically, only one Ranger class would be active at any given time, and these classes define meta-level structures and sequences during a professional or artistic presentation of the data

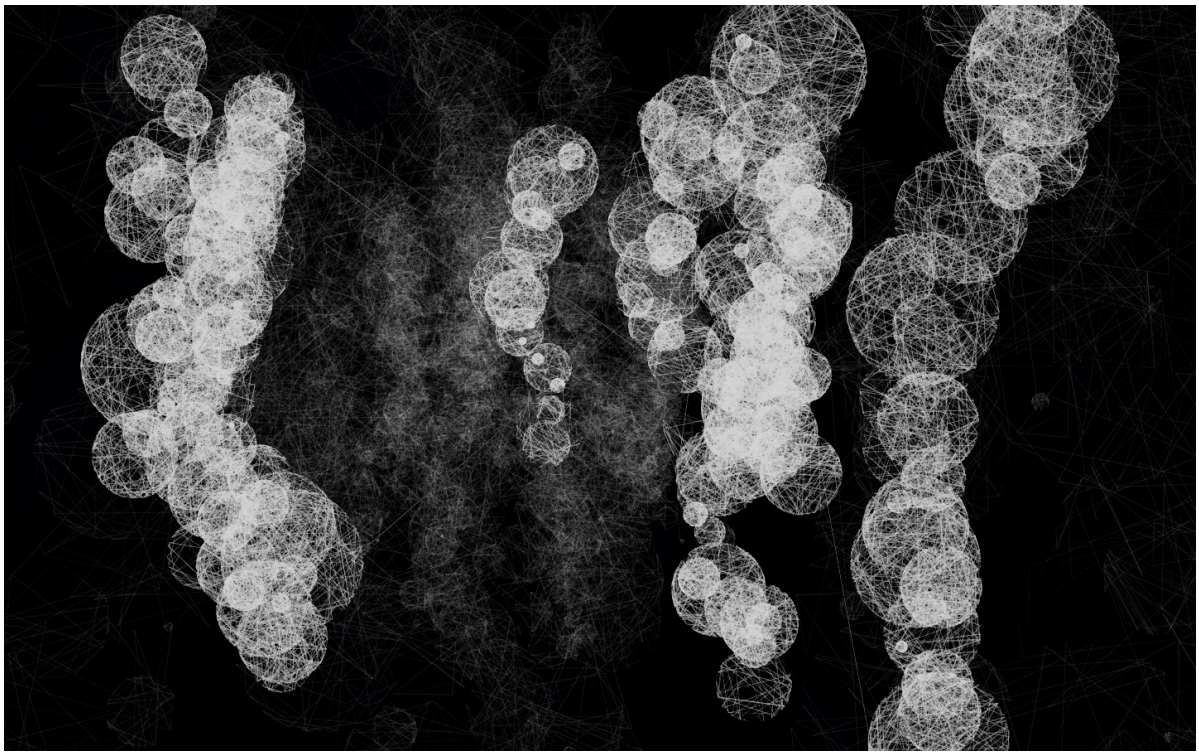

Figure 4: A closeup of the the team's "Norris" Vis_object. Data from electrophoresis gels are used to transform and extrude 3D meshes in spaces, representing density and skew as signifiers of specific trials and the distribution of their molecular weights.

perceptualization. A Ranger object, for example, might begin its operation by defining three different concurrent Walker objects and after some conditions have been met replace two of them with Walker objects of different behavior types.

These objects create the Walker objects that traverse the data, and choose which perceptualization objects those Walker objects should report to — creating, removing, or altering the relationships between these when necessary.

4. AN EXAMPLE PROGRAM

This section will describe an example hierarchy of behavioral abstractions that would define a specific operating program in the *madBPM* platform. As described in the previous section, Ranger objects encapsulate the storage and lifespan of objects that parse data streams, or the program logic of the presentation. For this example program, the Ranger object might begin operation by automatically creating two collections of data from different subsets of tags. The collections (A and B) might consist of all of the gel_Lanes tagged with {"supernatant", "M9pG", "hrs6"}, while the second consists of gel_Lanes tagged with {"Cel9AT", "lipA", "Induced"}, respectively. Next, the Ranger must associate each collection with a parsing object. Collection A could be assigned a *ForwardWalker* and collection B a *LocalMaxWalker*.

The *ForwardWalker* and *LocalMaxWalker* objects are predefined built-in Walker objects for traversing data collections they are associated with. *ForwardWalker*'s parse the each of the gel_Lanes in their associated collection in the order they are defined, and within each gel_Lane this walker visits each datum in the order defined. *LocalMaxWalker* visits each gel_Lane in its associate collection in the order defined in the collection similar to the *ForwardWalker*. However, for each gel_Lane visited, the *LocalMaxWalker* will only visit the largest local data value. The rate at which Walkers traverse the collection they are associated with is also specified by the Ranger object that defined them. The parent Ranger also specifies the action that Walker objects should take when they have reached the end of their collections. By default, Walkers that reach the end of their collection return to the start and continue parsing again. But Walkers can also be set to stop all parsing upon completion and to tell their parent Ranger that they have finished.

Instead of discontinuing parsing, the example Ranger will allow the default looping behavior and keep track of its own timing clock. Now that the program consists of subsets of data and behavioral abstractions that define how to parse them, the Ranger must associate sonification and visualization mappings for the Walkers. Perceptualization algorithms are encapsulated in Sounder and VisObjects. The example Ranger will create a *ScaleSounder* for both Walker objects, but it could provide *ForwardWalker* with a *GelBars_VisObject* (Fig. 1) and a *LocalMaxWalker* with a *NorrisMesh_VisObject* (Fig. 4). The drawn output of both visualization objects are overlaid on the same screen space. Based on each Walker object's specified sample timing, the objects poll their next data point and deliver that data to their connected mapping objects, both Sounders and VisObjects.

In this example program, the Ranger might use an internal timer to sequence changes in data collections, data parsing algorithms, and perceptualization mappings. For example, after 5 minutes have elapsed the Ranger could fade out the *ScaleSounder*

attached to the *LocalMaxWalker* and replace it with a *TimbreShapeSounder*[3]. After a few more moments, the example Ranger might remove *ForwardWalker* from the program, replacing it with a slightly altered clone of the *LocalMaxWalker*.

It is also be possible for the program flow in the Ranger to change based on conditional logic. An example mentioned above involved possibly removing a Walker once it had completely parsed a data collection instead of looping again over the data. Another possibility, however, is that Walkers that encounter data values within specific ranges could instigate structural changes in their parent Ranger. For example, if a Walker iterating over a collection encounters a value that is near a given molecular weight and has a localized intensity above a given threshold, the Ranger could respond by removing some Walkers and generating other new collections, parsing algorithms, and data mappings. The resulting change affects both the aural and visual data mappings, but also the logical structure of the analysis program. This flexibility could possibly provide a means of exploring perceptual feature optimization automatically through behavioral objects.

5. CONCLUSIONS AND FUTURE WORK

The artistic nature of musical sonification is a key element in the future plans for the *madBPM* platform, allowing our team to pursue both artistic and diagnostic goals using the project. Working in conjunction with members of the Biological Sciences department at Rensselaer Polytechnic Institute our team is investigating methods of multi-modal sonification and visualization using *madBPM* to allow researchers to better understand relationships between proteins used in the synthesis of high-value chemicals. *madBPM* allows both scientific researchers and artists to process and map data parameters from recent experiments quickly and efficiently to parameters of sound ranging from low-level synthesis techniques to higher-level organizational or compositional parameters. In this manner we envision a series of sonification and visualization experiments that analyze data sets from multiple viewpoints, allowing for fresh new looks into the data itself.

At the same time, the use of biological data as the progenitor of data-driven artworks is central to the *madBPM* project, allowing composers and visual artists to experiment with biological data in the creation of multi-modal artworks. An exhibition of such works is currently being planned at Rensselaer Polytechnic Institute's Collaborative-Research Augemented Immersive Virtual Environment[CRAIVE] Laboratory to showcase how biological data can inspire art, as well as how art can inspire research using biological data.

Within *madBPM*'s technical implementation, the project has clearly defined future milestones including:

- Implementation of a Domain Specific Language (DSL) for real-time scripting and definition of Rangers, Walkers, and Sounder/VisObjects
- Support for real-time data streams and ad-hoc data models
- Expanding the existing support for running the platform as a networked application

The creation of a DSL for real-time scripting of the *madBPM* platform will allow for more rapid research prototyping and expressive aesthetic data explorations. Currently, the team has proposed a DSL model that would focus on defining the high level behavioral abstractions, allowing users to customize data mapping and program logic at the application's runtime. The scripting language specification would require a balance between low level access to data and functions that can reach the level of composition — creating complex behaviors from underlying behavioral components.

madBPM uses a data model that is derived from electrophoresis gels in biosynthetic chemical research. In the course of implementing a data importing framework, the team found explored many useful ways of tagging, storing, and passing data around the platform. Another proposal for the future development of *madBPM* is the generalization of these methods for different types of data models and streams. In particular, the team would like to implement a pipeline for real-time data streams with data models that can be defined or redefined on the fly (in coordination with the DSL milestone).

The final proposed milestone for future development is expansion of the *madBPM*'s networking capabilities. As the size of data archives and complexity of algorithms grow, it often becomes necessary to distribute computational workloads amongst networked computer nodes. This is especially true for professional and aesthetic presentation of data perceptualization, where timing and reliability can be crucial. *madBPM*'s use of Open Sound Control already leverages a balance in speed and reliability and the possibility of communicating across a network between front and back ends. Proposed future development would break processes down into smaller units for concurrent distributed computation, allowing visual and audio display to be broken up amongst several monitors and speakers.

[3] *TimbreShapeSounder*s use filters to reshape the spectral characteristics of a sonic drone texture

6. ACKNOWLEDGEMENTS

The work described here was made possible through a grant from the Rensselaer Polytechnic Institute's Knowledge and Innovation Program (KIP).

7. REFERENCES

[1] R. Hamilton, "Musical Sonification of Avatar Physiologies, Virtual Flight and Gesture," *Lecture Notes in Computer Science: Computer Music Multidisciplinary Research (CMMR) Journal*, Springer-Verlag, Heidelberg, Germany, 2014.

[2] R. Hamilton, *Perceptually Coherent Mapping Schemata for Virtual Space and Musical Method*, Ph.D. Thesis, Stanford University, 2014.

[3] M. Kleiman-Weiner and J. Berger. 2006. "The Sound Of One Arm Swinging: A Model For Multidimensional Auditory Display Of Physical Motion." *International Conference on Auditory Display*, London, UK. http://www.mit.edu/ maxkw/pdf/kleiman2006sound.pdf

[4] G. Slack, "Hearing a Seizure's Song", *Discover Magazine*, online: http://discovermagazine.com/seizure, May, 2014.

[5] T. Bovermann, J. Rohrhuber, and A. de Campo, "Laboratory Methods for Experimental Sonification," in T. Hermann, A. Hunt, and J.G. Neuhoff, editors, *The Sonification Handbook*, Logos Publishing House, Berlin, Germany, pp. 237–272.

[6] J. McCartney, “Rethinking the Computer Music Language: SuperCollider”. *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, Feb. 2002.