ARTIFICIAL LIFE IN INTEGRATED INTERACTIVE SONIFICATION AND VISUALISATION: INITIAL EXPERIMENTS WITH A PYTHON-BASED WORKFLOW

Jack Armitage & Miguel Crozzoli

Intelligent Instruments Lab
University of Iceland
Reykjavík, Iceland
jack|miguel@hi.is

Daniel Jones

Jones/Bulley Studios
London, UK
daniel@jones.org.uk

ABSTRACT

Multimodal displays that combine interaction, sonification, visualisation and perhaps other modalities, are seeing increased interest from researchers seeking to take advantage of cross-modal perception, by increasing display bandwidth and expanding affordances. To support researchers and designers, many new tools are being proposed that aim to consolidate these broad feature sets into Python libraries, due to Python's extensive ecosystem that in particular encompasses the domain of artificial intelligence (AI). Artificial life (ALife) is a domain of AI that is seeing renewed interest, and in this work we share initial experiments exploring its potential in interactive sonification, through the combination of two new Python libraries, Tölvera and SignalFlow. Tölvera is a library for composing self-organising systems, with integrated open sound control, interactive machine learning, and computer vision, and SignalFlow is a sound synthesis framework that enables real-time interaction with an audio signal processing graph via standard Python syntax and data types. We demonstrate how these two tools integrate, and the first author reports on usage in creative coding and artistic performance. So far we have found it useful to consider ALife as affording synthetic behaviour as a display modality, making use of human perception of complex, collective and emergent dynamics. In addition, we think ALife also implies a broader perspective on interaction in multimodal display, blurring the lines between data, agent and observer. Based on our experiences, we offer possible future research directions for tool designers and researchers.

1. INTRODUCTION

In their recent state-of-the-art report on the integration of sonification and visualisation, Enge *et al.* conclude that:

> When it comes to the possibilities to design audiovisual display idioms, most researchers, as well as most domain experts, will not be able to successfully develop their own designs. Interdisciplinary knowledge bridging visualization, sonification, interactive design, and human perception is necessary to design effective, engaging, and re-usable audiovisual display idioms. Therefore, we cannot expect a domain expert to be able to quickly draft a prototype. [1]

While diverse knowledge is required to realise a professional audiovisual display, there are now more tools than ever attempting to make bringing them together easy, simple and customisable. Many of the sonification tools presented in their review are web-based GUIs that are indeed not meaningfully programmable or extensible. However, more recently, convergence has centred around the Python ecosystem, where a significant proportion of data science and research is currently taking place. In particular, multiple efforts have been made to bring the power of a dedicated audio digital signal processing (DSP) language to Python, including ways to embed SuperCollider in Python [2, 3], and a complete port of the SuperCollider class system [4]. However, while Python has many well-established frameworks for visualisation, most were not designed for the high-bandwidth interactivity that is now being demanded of them. Recent JavaScript-based visualisation frameworks such as Mosaic [5] increasingly do have these features, but Python programmers are being left behind. This is especially relevant in the case of integration between sonification and visualisation, where real-time performance becomes critical.

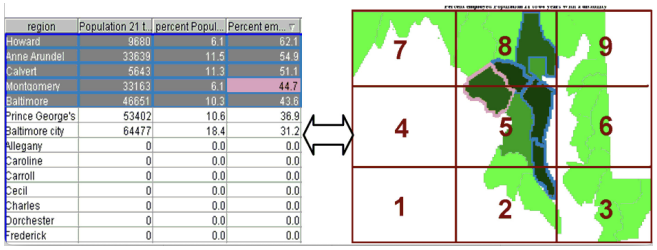
In this paper, we demonstrate the integration of two libraries, SignalFlow[1] [6] and Tölvera[2] [7], that aim to contribute to this exciting area of research. As the authors of these two libraries, we realised during the Timbre Tools Hackathon[3] in February 2024, that new affordances for interactive sonification might be possible by combining them. Although Tölvera was not designed as a visualisation library per se, and lacks scientific plotting features, its design for high-bandwidth real-time interaction, integration with common creative computing tools, and focus on composing together simulated behaviours, made it interesting to repurpose for sonification. Working with a hackathon team, we managed to demonstrate initial concepts that were then elaborated on over the following months[4], reaching a stage where the libraries could be used together in live performance.

[1] http://signalflow.dev

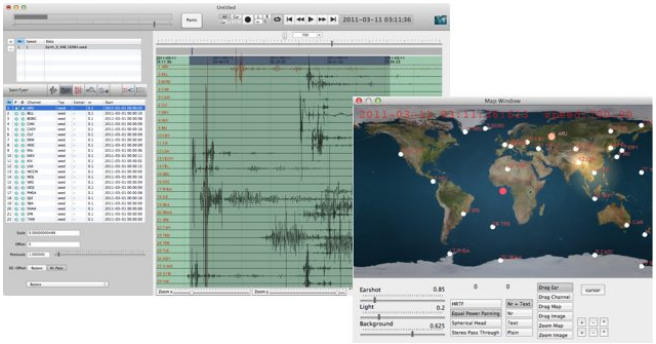
[2] http://tolvera.is

[3] https://comma.eecs.qmul.ac.k/timbre-tools-hackathon

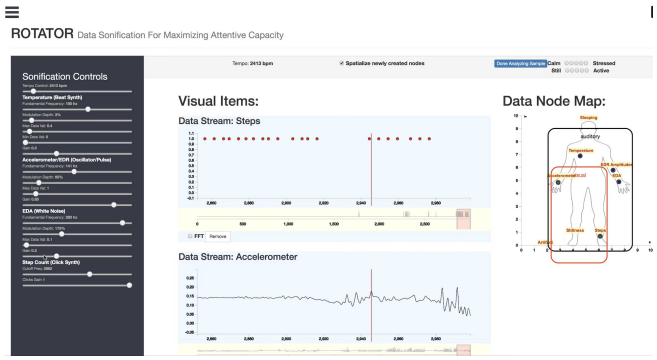
[4] Creative coding examples can be found at: https://github.com/Intelligent-Instruments-Lab/iil-examples/tree/main/tolvera/signalflow



(a) iSonic (2005) [8]

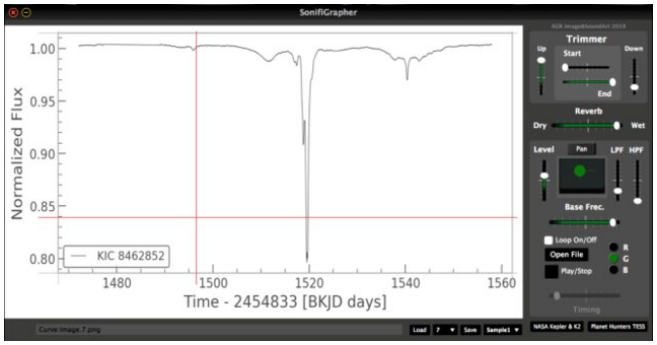


(b) sonifyer (2008) [9]

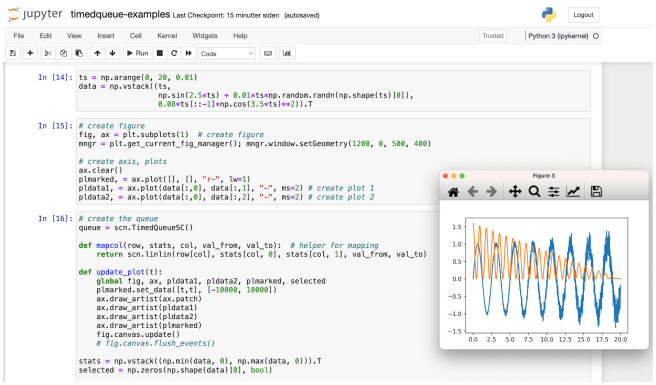


(c) Rotator (2017) [10]

(d) Sonification Workstation (2019) [11]

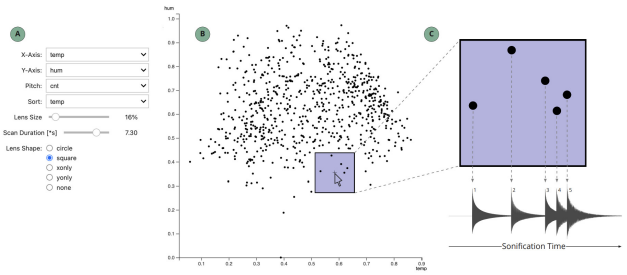


(e) Sonifigrapher (2019) [12]



(f) sc3nb (2021) [2]

(g) WebAudioXML (2021) [13]



(h) SoniScope (2022) [14]

Figure 1: Chronologically ordered screenshots of various sonification and visualisation tools reviewed in Section 2.1.

2. BACKGROUND

Multimodal displays that combine interaction, sonification, visualisation and perhaps other modalities, are seeing increased interest from researchers. Enge et al. [1] discuss that both visualisation and sonification, forms of data perceptualisation, offer various advantages and challenges. Visualisation provides a non-linear perception of data, easily revealing patterns, correlations, and trends. In contrast, sonification makes use of our excellent auditory system's capacity to understand temporal changes and patterns [1]. Combining these capabilities in a dynamic multi-display system would enrich the exploratory process and understanding of data.

2.1. Integrating Sonification & Visualisation

Various projects in the field of data display have attempted to create a workflow and integration between visualisation, sonification, and interaction (see also Figure 1). An extensive recent review can be found in [15]. Some of these projects relevant to the contextual placement of our system are:

iSonic [8] is an interactive sonification tool programmed in Java. It focuses on vision-impaired users to allow the exploration of geo-referenced numerical data. iSonic organises the data in two modes: a table mode organised by regions (rows) and attributes (columns), or a map mode showing the geographical distribution of statistical attributes. Users can interact with the data using a keyboard or touchpad and listen to the data mapped to pitch to reflect variations in the values, using changing timbres to represent spatial variation in the data.

Sonifyer [9] is an interactive sonification tool programmed for macOS that allows users to explore data features through audification or frequency modulation synthesis for parameter-based sonification. The project aims to create user-friendly software for both general and specialised users. The workflow includes importing various data formats, organising data into visual and spatial views, adjusting synthesis parameters, and using combined visual and auditory feedback to analyse the data.

Sonification Workstation [11] is an open-source application programmed in C++, QML, and JavaScript, using the Qt framework. It is an interactive tool that combines sound and graphic data representation by dividing the application into two windows: one for data visualisation and the other a patcher for mapping and creating sonification schemes. The project aims to provide a platform for data sonification, allowing people without sonification knowledge, to create data-to-sound representations easily.

Rotator [10] is a web-based multisensory analysis interface, programmed in JavaScript using React, D3.js for visualisation, and Web Audio API for audio synthesis. The project features a rotational display interface where users can interactively choose different data features by adjusting the contour of visualisation and sonification selection boxes. The platform offers six dedicated synthesisers based on filtering noise signals, mapping frequency to pitch in wave oscillators, applying ASDR envelopes to a sound signal, and varying the intervallic distance between two oscillators, where incoming data control all sound parameters. Additionally, the tool offers a data audification option for auditory display.

Sonifigrapher [12] is a synthesiser defined as a graph-to-sound converter. It uses light curves from NASA's publicly available exoplanet archive as a graphic source for sonification. Programmed in CSound, Sonifigrapher uses additive synthesis to generate controlled audio spectra for sonification.

`sc3nb` [2] is a Python package integrating the SuperCollider programming environment into Jupyter Notebook, enabling interactive control and sound synthesis for data-to-sound displays. The project includes TimedQueues, an event dispatcher that facilitates audiovisual displays by connecting sc3nb with Python libraries with visualisation functionalities such as matplotlib. The goal of the project is to provide an accessible and flexible coding system for auditory data science.

WebAudioXML Sonification ToolKit [13] is an open-source, web-based tool programmed in JavaScript using WebAudioXML for audio synthesis. It provides a highly accessible web platform for exploring data via visualisation of selected features and parameter-based sonification of those features. The goal of the project is to make data-to-sound display available to a broader audience, including non-experts. Users import data, visualise it, map data variables to audio parameters, and listen to the sonification directly in their web browsers.

SuperCollider's Class Library Port [4] is a fully functional port of the client side SuperCollider's class library from sclang to Python. It preserves the original architectural features of SuperCollider. The goal is to provide a flexible and comprehensive Python-based system for sound and music computing, harvesting SuperCollider's sound synthesis capabilities and taking advantage of the integration with other Python libraries.

SoniScope [14] is an interactive sonification tool using the Jupyter Notebook environment, D3.js for visualisation, and SuperCollider for audio synthesis, programmed in Python and JavaScript. It combines a scatterplot visualisation with parameter-based sonification for exploring multivariate data, displaying two dimensions on the visual representation - x and y axis - and sonifying two features as pitch and onset. The interface allows users to select features assigned to each visual and sound parameter.

2.2. Interactivity & Artificial Life

The integration of interactive sonification and visualisation with artificial life presents a fertile ground for a wide range of applications, for example in understanding the emergent properties of collective behaviors, such as flocking, schooling, or swarming, can be enhanced by sonification. This approach aids scientific inquiry and also serves as inspiration for new musical compositions and interactive art installations [16, 17], bridging the gap between informative sonification/visualisation and artistic expression. We have also explored conceptualising artificial life systems as a form of musical notation, we propose an innovative approach to interactive scores, where the behavior of artificial entities and their interactions serve as the basis for generating and manipulating notational material in real-time [18]. Finally, by sonifying the data, training process, outputs and behaviors of AI models, we can provide auditory cues that complement visual explanations, thus offering a more holistic understanding of AI's decision-making processes [19, 20].

3. IMPLEMENTATION

SignalFlow [6] is a sound synthesis framework that enables real-time interaction with an audio signal processing graph via standard Python syntax and data types, and Tölvera [7] is a library for composing self-organising systems, with integrated open sound control, interactive machine learning, and computer vision. In this section we provide an overview of both of these libraries, and then

demonstrate basic ways of integrating them. Later, in Section 5.1, we describe current limitations.

3.1. SignalFlow

SignalFlow [6] is a Python framework for audio signal processing, designed for intuitive expression and exploration of sonic ideas. Its core is implemented in cross-platform C++, with a Python API that allows DSP primitives (oscillators, operators, filters) to be created, connected and modulated in real-time. The inputs and outputs of the processing graph utilises native Python I/O and data types, meaning that it can interoperate seamlessly with existing packages for data science tasks such as sonification.

Below is a minimal SignalFlow example. Here, we create and immediately start the audio processing graph, construct a stereo sine oscillator with a short envelope, connect the oscillator to the graph's output, and run the graph indefinitely.

```
from signalflow import *

graph = AudioGraph()
sine = SineOscillator([440, 880])
envelope = ASREnvelope(0.1, 0.1, 0.5)
output = sine * envelope
output.play()
```

This demo shows a few syntactical benefits that SignalFlow provides to make it easy to work with audio:

- The 2-item array of frequency values passed to `SineOscillator` is expanded to create a stereo, 2-channel output. Passing a 10-item array would result in a 10-channel output.
- Mathematical operators such as `*` can be used to multiply, add, subtract or divide the output of nodes, and creates a new output `Node` that corresponds to the output of the operation. This example uses an envelope to modulate the amplitude of an oscillator.
- Although the envelope is mono and the oscillator is stereo, SignalFlow automatically up-mixes the envelope's values to create a stereo output, so that the same envelope shape is applied to the *L* and *R* channels of the oscillator, before creating a stereo output.

For auditory display of offline datasets such as time series, SignalFlow recognises vector data types such as `numpy` arrays, which can be map For online mapping as presented in this paper, the parameters of the graph can be modulated in real-time, providing a live, audible portrayal of properties of a dynamical system.

3.2. Tölvera

Tölvera [7] is a Python library designed for composing together [23] and interacting with basal [24] agencies [25]. It provides creative coding-style APIs that allow users to combine and compose various built-in behaviours, such as flocking, slime mold growth, and swarming, and also author their own. With built-in support for Open Sound Control (OSC)[5] and interactive machine learning (IML)[6], Tölvera interfaces with existing music software and hardware, striving to be an accessible and powerful tool for exploring diverse intelligence [26] in artistic contexts.

A basic Tölvera program that displays a window, and a multi-species particle simulation exhibiting flocking behaviour, can be achieved with just a few lines of code:

```
from tolvera import Tolvera, run

def main(**kwargs):
    tv = Tolvera(**kwargs)

    @tv.render
    def _():
        tv.px.diffuse(0.99)
        tv.v.flock(tv.p)
        tv.px.particles(tv.p, tv.s.species())
        return tv.px

if __name__ == '__main__':
    run(main)
```

In Python, Tölvera is instanced as `tv`, and its main features are all available via the following sub-objects:

`tv.p`: Multi-species particle system, where each has a unique relationship with every other species, including itself.

`tv.s`: Declarative-style global dictionary of n-dimensional (`ndarray`) state structures that can be used by verur, including built-in OSC and IML creation.

`tv.v`: A collection of behaviours/models including Move, Flock, Slime and Swarm, with more being continuously added (Figure 2). *Verur* can be combined and composed in various ways.

`tv.px`: Drawing library including various shapes and blend modes, styled similarly to p5.js.

`tv.ti`: GPU simulation and rendering engine via Taichi[7] [27]. Can be run headless (without graphics).

`tv.osc`: Open Sound Control (OSC) via `iipyper`[8], including automated export of OSC schemas to JSON, XML, Pure Data (Pd), and Max/MSP.

`tv.iml`: Declarative-style global dictionary of interactive machine learning instances via `anguilla`[9].

`tv.cv`: computer vision integration based on OpenCV and Mediapipe[10].

3.3. SignalFlow-Tölvera Integration

We can combine the SignalFlow and Tölvera examples above, sonifying a Tölvera particle's position on the x-axis by mapping it to a sine oscillator's frequency in SignalFlow, with the two event loops `output.play()` and `@tv.render` not interferring with each other:

```
import signalflow as sf
from tolvera import Tolvera, run

def main(**kwargs):
    tv = Tolvera(**kwargs)
```

[5] https://github.com/Intelligent-Instruments-Lab/iipyper

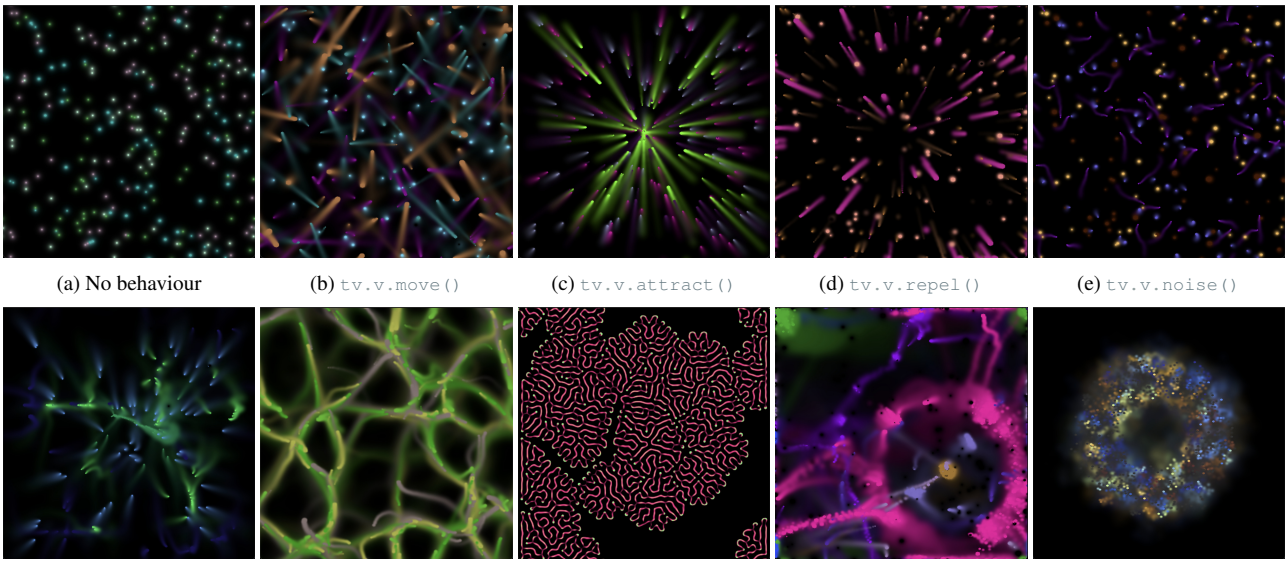
[6] https://github.com/Intelligent-Instruments-Lab/anguilla

[7] https://taichi-lang.org/

[8] https://github.com/Intelligent-Instruments-Lab/iipyper

[9] https://github.com/Intelligent-Instruments-Lab/anguilla

[10] https://developers.google.com/mediapipe



(a) No behaviour (b) `tv.v.move()` (c) `tv.v.attract()` (d) `tv.v.repel()` (e) `tv.v.noise()`

(f) `tv.v.flock()` boids [21] (g) `tv.v.slime()` physarum (h) `tv.v.rd()` reaction-diffusion (i) `tv.v.plife()` particle life (j) `tv.v.swarm()` swarmalators [22]

Figure 2: Examples of behaviours and models available via `tv.v`. Top row: stateless `tv.v`. Bottom row: stateful `tv.v`.

```
    graph = sf.AudioGraph()
    sine = sf.SineOscillator(440)
    stereo = sf.StereoPanner(sine, 0.0)
    output = sine * 0.1
    output.play()

    def update():
        pos = tv.p.field[0].pos
        sine.frequency = 100+(pos[0]*(1000-100))/tv.x

    @tv.render
    def _():
        update()
        tv.px.diffuse(0.99)
        tv.v.flock(tv.p)
        tv.px.particles(tv.p, tv.s.species())
        return tv.px

if __name__ == '__main__':
    run(main)
```

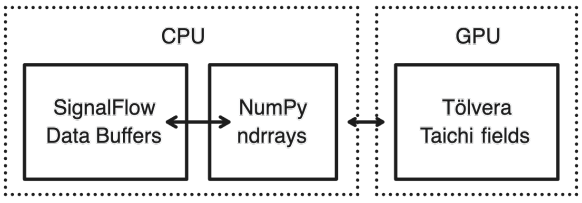


Figure 3: Diagram of data interoperation between SignalFlow and Tölvera: SignalFlow processes data inside its `AudioGraph` on the CPU, and Tölvera processes Taichi fields on the GPU, with NumPy `ndarray`s being the bridge between them.

Data interoperation (Figure 3) is possible without modifying the underlying C++ engines of SignalFlow and Tölvera, via NumPy's N-dimensional array (`ndarray`):

From SignalFlow to Tölvera:

```
ti_buf = ti.ndarray(dtype=ti.f32,
↪   shape=output.output_buffer.shape)
ti_buf.from_numpy(output.output_buffer)
```

And from Tölvera to SignalFlow:

```
buf = Buffer(2, 1024)
ti_buf = ti.ndarray(dtype=ti.f32, shape=buf.data.shape)
ti_buf.fill(1)
buf.data = ti_buf.to_numpy()
```

Composition of behaviours can also be achieved, for example by modulating Tölvera models' `weight` parameter via SignalFlow LFOs:

```
graph = AudioGraph()
lfo = SineLFO(0.5, 0, 10)
lfo.play()

@tv.render
def _():
    tv.px.diffuse(0.99)
    tv.v.move(tv.p, lfo.output_buffer[0][0])
    tv.px.particles(tv.p, tv.s.species())
    return tv.px
```

More elaborate examples are discussed in the next section.

4. EARLY CASE STUDIES

Initially, we have prioritised creative and artistic exploration of SignalFlow-Tölvera integration, rather than problem-oriented analytical and scientific tasks, though we eventually seek to explore the middle ground between these two. Following Buxton[11] [28],

[11]"[...] in the grand scheme of things, there are three levels of design: standard spec., military spec., and artist spec. Most signif-

we believe that creative contexts probe the possibility space in diverse and demanding ways, that are useful in early-stage performance testing. Rather than defining a rigid workflow, we instead sought to explore the broad compositional space through sketching and artistic practice (Figure 4).

4.1. Creative Coding Examples

In the `iil-examples` footnote provided in Section 1 we offer a variety of creative coding sketches that explore different possibilities for integrating ALife with interactive sonification. These examples are organised into category folders, including at the time of writing *see*, *listen* and *behave*. The folder *see* focuses on different ways of accessing and drawing SignalFlow nodes and buffers, *listen* focuses on sonification of Tölvera models and states, and *behave* explores combinations of the two. We welcome readers to try these examples and share feedback on their experiences.

4.2. Corpus-Based Granular Synthesis

A more complex example in the same repository demonstrates a scenario where `librosa` [29] and `sklearn` [30] are used to perform principle component anlaysis (PCA) on Mel-frequency cepstral coefficients (MFCCs). The results of this analysis allow an audio file to be decomposed into grains which can be triggered by SignalFlow's `SegmentedGranulator` class. These grains can then be triggered to play using various approaches in Tölvera. For example, a moving particle agent can represent a virtual microphone, allowing the user to listen to the data by moving around the points.

4.3. Live Coding Performance

Live coding of SignalFlow and Tölvera is possible via Sardine[12] [31], a recently proposed Python-based live coding environment. Sardine enables live coding of Tölvera down to individual Taichi GPU kernels, increasing interactivity of the programming experience. The main mechanism of Sardine is called a `@swim` function:

```
@swim
def number(p=0.5, i=0):
    print(P('1 1+1 1*2 1/3 1%4 1+(2+(5/2))', i))
    again(number, p=0.5, i=i+1)
```

`@swim`s are temporally recursive functions, featuring iterators (`i=i+1`) that can be used in a pattern language `P("...")`, thus the above example will print the values of the pattern `P` based on the iterator value `i` every time Sardine schedules the function `number`. A performance titled *Gagnavera* ("data being", Figure 4b) was made at Mengi in Reykjavík, Iceland in March 2024 based around three `@swim` functions:

- `render_loop`: a live version of Tölvera's `@tv.render` method.
- `flock_loop`: a SignalFlow saw oscillator bank mapped to Tölvera particles.
- `active_loop`: Sardine `P` patterns controlling Tölvera particle activity amounts.

Simply by writing patterns in the `active_loop` function, a great deal of variation was achievable, as activating/deactivating particles or species would reshape the emergent dynamics. Usually in musical live coding the musician is directly editing sound patterns, whereas this approach focused on perturbation of ecological dynamics and their impact on sound. Listening is often highlighted as the most important activity in the live coding perception-action loop, and the additional complexity of the loop in this case necessitated a different kind of listening.

4.4. Gestural Performance with Computer Vision

As part of a CAMP[13] workshop, a performance titled *Sveimivera* ("hover being", Figure 4c) was made. In this performance, five pairs of saw oscillators were mapped to particle XY positions to imitate insect buzzing sounds and panned accordingly. These oscillators were also visualised as a Lissajous curve that tracked the particle positions. Mediapipe hand tracking was then used to attract the five oscillator pairs towards the performer's fingertips, with their sounds also modulating based on their proximity to their target fingertip. This setup allowed the performer to create sonic variations based on different hand gestures and kinds of movement, as well as hiding the hand from the tracker and allowing the oscillators to flock amongst themselves. Similarly to the live coding example, controlling sound indirectly creates an interesting set of complex dependencies that the performer is encouraged to explore and to navigate.

5. DISCUSSION

This preliminary investigation has established the feasibility of combining SignalFlow and Tölvera in order to pave the way for deeper studies into the potential of combining ALife with real-time interactive sonification. Both more breadth and depth of work are required to fully realise this potential. Although we have partly chosen Python as our target language for its AI ecosystem, we have yet to combine our approach with frameworks like PyTorch or JAX, though this is already possible in Taichi[14]. This is high on our list of priorities, along with collaborations with professional data analysts and scientists. Surveying our work so far, in this section we describe the technical limitations at our current stage of development. Then, we attempt to address what our early experiences point towards as directions for ongoing research, and implications for multimodal display.

5.1. Technical Limitations

The integration between SignalFlow and Tölvera is at an early stage and can be improved in a number of different ways. Mainly, the way data is accessed and memory is shared between the two can be improved, with lower-level integrations on both sides. In SignalFlow, a dedicated Tölvera Node could provide a simple API for passing in Tölvera state and arbitrary Taichi data containers into the SignalFlow context. In Tölvera, dedicated utility functions would provide the inverse, allowing SignalFlow audio buffers and other data to be easily manipulated inside Taichi's GPU scope.

Another area that can be improved is temporal synchronisation. Although SignalFlow can provide a clock signal based on its Impulse Node, it lacks a dedicated scheduler, and Tölvera lacks a stable framerate. Currently, it is not currently possible for Tölvera

icantly, I learned that the third was the hardest (and most important), but if you could nail it, then everything else was easy."

[12] `https://sardine.raphaelforment.fr/`

[13] `http://campfr.com`

[14] `https://docs.taichi-lang.org/docs/external`

(a) Corpus-Based Granular Synthesis example (b) Gagnavera (March 2024) (c) Sveimivera (May 2024)

Figure 4: Examples discussed in Section 4.

to know which sample in a SignalFlow buffer is currently being played (in the LFO example, Tölvera reads the first sample in the buffer `lfo.output_buffer[0][0]`), and SignalFlow cannot trigger Tölvera functions via callbacks. Aside from implementing a scheduler natively in SignalFlow, one possibility is to instead borrow one, potentially from Sardine [31].

5.2. Synthetic Behaviour as Display Modality

What does ALife offer to multimodal display? The term *ALife* at this point carries a lot of connotations based on its >30 year history. To circumnavigate this, we have found it useful to consider ALife as specifically affording *synthetic behaviour* as an interactive display modality. In the same way that visualisation and sonification engage our perception to convey information and feelings, synthetic behaviour makes use of human perception of complex, collective and emergent dynamics to communicate. In display, we commonly think of auditory, visual, and audiovisual *icons*, and with ALife we might also consider *behavioural icons* that vibrate, cluster, flock, swarm, spin, spiral and gesticulate. It follows that *behavioural grammars* could conceivably be designed based on research about human perception of motion and agency. Indeed, appropriate taxonomies and grammars of behaviour likely already exist in other fields, waiting to be tested in this context.

5.3. Spectrums of Interaction

In addition, we think ALife also implies a broader perspective on interaction in multimodal display. As the term itself suggests, in human-computer interaction (HCI) it is common to rigidly assume a bidirectional interaction between two entities: human and computer. In interactive sonification, this assumption is often carried over: a user interacts with data and perceives it as sound, in a perception-action cycle. In ALife however, interactions are happening at multiple scales of granularity and between entities of varying levels of sophistication, which are usually a combination of human and non-human agents. ALife can give individual data points extended agency beyond visual and auditory display, allowing them to move, transform and respond to their environment and other inputs. Thus the lines between data, agent and observer become blurred. This theme has been prevalent throughout HCI in calls for more-than-human accounts and interfaces that start from ecological and relational perspectives, inspired particularly by the work of Barad on agential realism [32]. We discuss these themes more in our upcoming NIME 2024 paper [7].

6. CONCLUSION

We presented a Python-based workflow for creation of integrated interactive sonification and visualisation projects that make use of artificial life, based on the combination of two libraries, SignalFlow and Tölvera. SignalFlow enables real-time interaction with audio signal processing, harnessing the familiar syntax and data types of Python for intuitive sound synthesis. Tölvera's capabilities in visualising and simulating artificial life systems, coupled with features such as open sound control, interactive machine learning, and computer vision, show potential for creative and insightful explorations of self-organising systems. We highlighted a variety of application areas ripe for exploration through this integration, including the development of interactive audiovisual interfaces, the audiovisual display of collective behavior, and the augmentation of traditional sonification and visualisation practices with artificial life techniques. The next steps in our work involve extending and refining the integration between SignalFlow and Tölvera, ensuring smoother workflows and broader applicability in future work.

7. ACKNOWLEDGEMENTS

Jack Armitage would like to thank the Timbre Garden hackthon team – Eloi Marín Gratacós, Ivan Meresman Higgs, Farida Yusuf and Cláudio Lemos – and the Taichi developers and community. Jack Armitage is the creator of Tölvera and led both the Tölvera-SignalFlow integration and the writing and editing of this paper. Daniel Jones is the creator of SignalFlow and contributed to the Tölvera-SignalFlow integration and writing of this paper. Miguel Crozzoli contributed to the literature review and editing of this paper. The Intelligent Instruments project (INTENT) is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001848).

8. REFERENCES

[1] K. Enge, E. Elmquist, V. Caiola, N. Rönnberg, A. Rind, M. Iber, S. Lenzi, F. Lan, R. Höldrich, and W. Aigner, "Open Your Ears to Take a Look: A State-of-the-Art Report on the Integration of Sonification and Visualization," no. arXiv:2402.16558, Feb. 2024.

[2] T. Hermann and D. Reinsch, "Sc3nb: A Python-SuperCollider Interface for Auditory Data Science," in *Audio Mostly 2021*, Trento, Italy, September 2021, pp. 208–215.

[3] D. Jones, "Python client for the supercollider audio synthesis server," https://github.com/ideoforms/python-supercollider/, 2019.

[4] L. Samaruga and P. Riera, "A port of the SuperCollider's class library to Python," in *Proceedings of the 17th International Audio Mostly Conference*, ser. AM '22, New York, NY, USA, Oct. 2022, pp. 137–142.

[5] J. Heer and D. Moritz, "Mosaic: An architecture for scalable & interoperable data views," *IEEE Trans. Visualization & Comp. Graphics (Proc. VIS)*, 2024. [Online]. Available: http://idl.cs.washington.edu/papers/mosaic

[6] D. Jones, "Signalflow," https://signalflow.dev/, 2023.

[7] J. Armitage, V. Shepardson, and T. Magnusson, "Tölvera: Composing With Basal Agencies," in *Accepted for Proc. New Interfaces for Musical Expression*, Utrecht, Netherlands, Sep 2024.

[8] H. Zhao, C. Plaisant, and B. Shneiderman, "iSonic: Interactive sonification for non-visual data exploration," in *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility*, Baltimore MD USA, Oct. 2005, pp. 194–195.

[9] F. Dombois, "Sonifyer a concept, a software, a platform," in *Proceedings of the 14th International Conference on Auditory Display (ICAD 2008)*, Paris, France, June 2008, pp. 1–4.

[10] J. Cherston and J. A. Paradiso, "Rotator: Flexible distribution of data across sensory channels," in *Proc. 23rd International Conference on Auditory Display (ICAD 2017)*, Pennsylvania, USA, June 2017.

[11] S. Phillips and A. Cabrera, "Sonification workstation," in *The 25th International Conference on Auditory Display*, Newcastle-upon-Tyne, UK, June 2019, pp. 184–190.

[12] A. G. Riber, "Sonifigrapher. sonified light curve synthesizer," in *Proc. 25th International Conference on Auditory Display (ICAD 2019)*, Newcastle-upon-Tyne, UK, June 2019, pp. 62–66.

[13] H. Lindetorp and K. Falkenberg, "Sonification for everyone everywhere: Evaluating the webaudioxml sonification toolkit for browsers," in *The 26th International Conference on Auditory Display (ICAD 2021)*, Online, June 2021, pp. 15–21.

[14] K. Enge, A. Rind, M. Iber, R. Höldrich, and W. Aigner, "Towards Multimodal Exploratory Data Analysis: SoniScope as a Prototypical Implementation." in *EuroVis (Short Papers)*, Rome, Italy, June 2022, pp. 67–71.

[15] H. Kim, Y.-S. Kim, and J. Hullman, "Erie: A Declarative Grammar for Data Sonification," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '24, New York, NY, USA, May 2024, pp. 1–19.

[16] T. F. Tavares, T. R. P. Pessanha, G. Nishihara, and G. Z. L. Avila, "Alloy Sounds: Non-Repeating Sound Textures with Probabilistic Cellular Automata," in *2021 24th International Conference on Digital Audio Effects (DAFx)*, Surrey, UK, September 2021, pp. 245–252.

[17] D. Jones, "AtomSwarm: A Framework for Swarm Improvisation," in *Workshops on Applications of Evolutionary Computation*, vol. 4974, Berlin, Heidelberg, 2008, pp. 423–432.

[18] J. Armitage and T. Magnusson, "Agential Scores: Exploring Emergent, Self-Organising and Entangled Music Notation," in *Proceedings of the 8th International Conference on Technologies for Music Notation and Representation*, Northeastern University, Boston, Massachusetts, USA, 2023.

[19] B. W. Schuller, T. Virtanen, M. Riveiro, G. Rizos, J. Han, A. Mesaros, and K. Drossos, "Towards Sonification in Multimodal and User-friendlyExplainable Artificial Intelligence," in *Proceedings of the 2021 International Conference on Multimodal Interaction*, Montréal QC Canada, Oct. 2021, pp. 788–792.

[20] A. Akman and B. W. Schuller, "Audio Explainable Artificial Intelligence: A Review," *Intelligent Computing*, vol. 3, p. 0074, Jan. 2024.

[21] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Aug 1987, pp. 25–34.

[22] K. P. O'Keeffe, H. Hong, and S. H. Strogatz, "Oscillators that sync and swarm," *Nature Communications*, vol. 8, no. 1, p. 1504, Nov. 2017.

[23] J. Horowitz and J. Heer, "Live, Rich, and Composable: Qualities for Programming Beyond Static Text," no. arXiv:2303.06777, Mar. 2023.

[24] P. Lyon, F. Keijzer, D. Arendt, and M. Levin, "Reframing cognition: Getting down to biological basics," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 376, no. 1820, p. 20190750, Jan. 2021.

[25] J. Davies and M. Levin, "Synthetic morphology with agential materials," *Nature Reviews Bioengineering*, vol. 1, no. 1, pp. 46–59, Jan. 2023.

[26] M. Levin, "Technological Approach to Mind Everywhere: An Experimentally-Grounded Framework for Understanding Diverse Bodies and Minds," *Frontiers in Systems Neuroscience*, vol. 16, p. 768201, 2022.

[27] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Transactions on Graphics*, vol. 38, no. 6, pp. 1–16, Dec. 2019.

[28] B. Buxton, "Artists and the art of the luthier," *ACM SIGGRAPH Computer Graphics*, vol. 31, no. 1, pp. 10–11, 1997.

[29] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the 14th Python in Science Conference (SciPy 2015)*, Austin, TX, July 2015, pp. 18–24.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[31] R. Forment and J. Armitage, "Sardine: A Modular Python Live Coding Environment," in *International Conference on Live Coding*, Utrecht, Netherlands, April 2023.

[32] K. Barad, *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*, 2007.