ANTON BRUCKNER
PRIVATUNIVERSITÄT
OBERÖSTERREICH

Carlos Miguel González Bolaños

Matrikelnummer 61900925

FluCoMa: an approach to its ecosystem and convergence with the Wavesets Synthesis.

Masterarbeit

KMA Wissenschaftliche Arbeit

zur Erlangung des akademischen Grades

Master of Arts

des Studiums Komposition KMA

Studienkennzahl: RA 066 701

an der

Anton Bruckner Privatuniversität

Betreut durch: Ao. Univ. Prof. Andreas Weixler
Zweitleser: Prof. Dr. phil. Alberto de Campo (UdK Berlin)

Graz, 30.07.2023

Acknowledgment

I would like to express my sincere thanks to my supervisor Professor Andreas Weixler for all his knowledge and unconditional support during these years.

I would like to thank Professor Alberto de Campo for his kind willingness to cooperate in this research, which I hope will be the beginning of something bigger.

I would like to thank Rodrigo Constanzo for his time, knowledge, and advice on the FluCoMa project.

Finally, I would like to thank the University's Dean of artistic studies, Mr. Wilfried Brandstötter, for making collaboration with Prof. Alberto de Campo in this research possible.

Abstract

This paper proposes three possible intersections between Wavesets synthesis and FluCoMa's advanced listening and machine learning algorithms within the SuperCollider programming language. In the first part, the thesis introduces relevant topics such as SuperCollider, the FluCoMa project, and related topics, as well as the basic principles of Wavesets synthesis in SuperCollider. Subsequently, three FluCoMa code examples adapted by the author for possible integration with Wavesets synthesis are presented. Finally, the results are discussed, offering possible applications and implications for future research.

Table of contents

List of figures

List of charts

List of codes

I. Introduction

This research proposes an intersection of three broad fields of computer music that have been the object of interest in my musical practices and research in recent years: Wavesets synthesis, machine listening, and machine learning techniques oriented to musical practice.

The starting point of this research stems from investigating these fields within the SuperCollider programming language. SuperCollider offers numerous and flexible ways to perform Wavesets synthesis in real and deferred time, extending the possibilities originally proposed by the Computer Desktop Project environment. On the other hand, although SuperCollider has been one of the programming environments with more algorithms for the implementation of machine listening and machine learning techniques in a native way, it has not been until recent years, with the appearance of the FluCoMa project, that this field has undergone a real revolution.

This research introduces and explores the following question:

Is it possible to take advantage of the refined machine listening and machine learning algorithms proposed by the FluCoMa project to control and explore Wavesets synthesis in SuperCollider?

This paper is divided into two parts. The first part briefly introduces key aspects of the subject of this dissertation: what is SuperCollider? What are machine listening and machine learning? What is Wavesets synthesis? What is the FluCoMa project? The second part of the dissertation deals with the possible technical intersections between FluCoMa and Wavesets synthesis. For this purpose, three of the main example codes proposed by the FluCoMa project to exemplify three of its main applications (2D-corpus exploration, concatenative synthesis, and neural regression) are adapted.

Finally, the conclusions of this exploration and its possible proliferations are presented.

1. Introduction to SuperCollider

"[SuperCollider] represents the state of the art in audio programming technologies."[1]

SuperCollider has earned a reputation as one of the most popular software for real-time sound synthesis, algorithmic composition, and interactive data control.

It was designed in 1996 by James McCartney for the Macintosh platform, with two subsequent incarnations: SuperCollider 2 for MacOS9 and SuperCollider 3 for MacOSX. It is currently a cross-platform (Mac, Windows, and Linux) open-source software, supported mainly by its large and growing community of users.[2]

Its versatility as a digital audio working environment makes it ideal for users interested not only in music creation but also in scientific fields: algorithmic composition, multimedia creation, live coding, acoustic research, data sonification, interactive systems, interface design, interactive programming, WEB art or application design are some of its most popular applications.

The SuperCollider architecture consists of 3 main independent components that can be used together or separately: the audio server (scsynth/supernova), the programming language (sclang), and an IDE.[3]

- scsynth: considered the platform's core, it is an audio server programmed in C++[4] optimized for real-time audio synthesis.[5] The server uses the so-called UGens, the SuperCollider language representation of the basic blocks of signal calculations used to generate and process audio and control signals.[6] Like the server, the UGens are programmed in C++.[7]

---

[1] Andrea Valle, *Introduction to SuperCollider* (Berlin, Logos Berlin; Translation edition, 2016), 1.

[2] Ibid., 1-2.

[3] Ibid., 5.

[4] Fellipe Miranda Martins, "Estudo exploratório de processos de transformação sonora a partir de Trevor Wishart: reinvenção e tradução para o ambiente SuperCollider," Master's thesis, Universidade Federal de Minas Gerais, 2020, 56.

[5] Valle, *Introduction to SuperCollider*, 4.

[6] Martins, "Estudo exploratório", 56.

[7] Ibid.

In SuperCollider, it is possible to generate audio in non-real time, although with a more limited range of functionalities due to the client-server architecture of the program.[8]

- supernova is an alternative audio server that allows automatic multiprocessor parallelization[9]. It overcomes the inherent limitation of scsynth to operate on a single processor core by spreading the processing load across different computer cores.[10]

- sclang is the interpreted programming language of SuperCollider. It belongs to the category of object-oriented programming languages[11] and is a descendant of the Music-N family of languages.[12] Its design tries to strike a balance between the needs of real-time computing and the flexibility and simplicity of an abstract language[13] that allows different levels of abstraction (sclang allows low-level signal processing operations as well as intuitive and interactive ideas at a higher level of abstraction, closer to musical thinking).[14]

  The sclang has an interpreter in charge of "understanding" the SuperCollider language and syntax.[15] This interpreter also acts as a client communicating with the server through a network using OSC commands.[16] In the same way, the user can communicate with the audio server directly with OSC commands or through other programming languages that work as a client (Python, Processing, Haskell, Lua, and Ruby, among others).[17]

---

[8] Ibid., 58.

[9] Ibid., 56.

[10] Mads Kjedgaard, “Parallel processing in SuperCollider using SuperNova,” Mads Kjedgaard, last modified February 10, 2022, https://madskjeldgaard.dk/posts/supernova-intro/

[11] Valle, *Introduction to SuperCollider*, 3.

[12] James McCartney, “Forewords,” In *The SuperCollider Book*, ed. by Scott Wilson, David Cottle, and Nick Collins (Cambridge, Massachusetts London, England: The MIT Press, 2011), XI.

[13] “SuperCollider,” Wikipedia, last modified February 20, 2023, https://en.wikipedia.org/wiki/SuperCollider

[14] Valle, *Introduction to SuperCollider*, 4.

[15] Ibid., 3.

[16] Martins, “Estudo exploratório," 56-57.

[17] Ibid., 58.

- The IDE (Integrated Development Environment) is a cross-platform coding environment[18] with a powerful graphics engine.[19] It is in charge of building all the graphical aspects of SuperCollider, such as the GUIs (Graphical User Interfaces for visualization and interactive control of all types of data and signals), the text editor, the title, status, and document bars, the Help Browser, the Post Window, etc. but also allows the generation of vector graphics[20] and animations[21] in an algorithmic way. Users can replace SuperCollider's integrated IDE with another one of their choices, depending on the chosen platform (some available options are Atom, Vim, and Emacs, among others).[22]

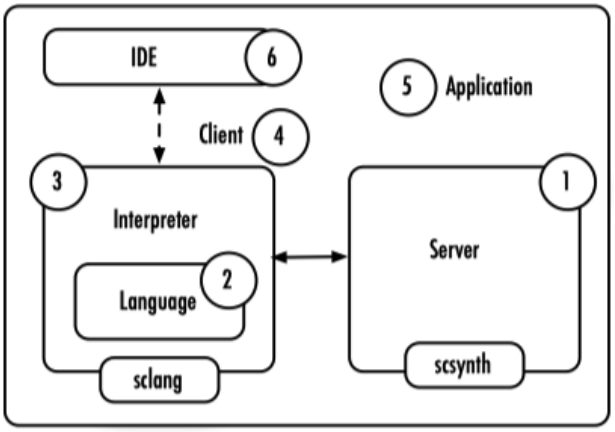

Figure 1 - SuperCollider architecture
Andrea Valle, *Introduction to SuperCollider* (Berlin, Logos Berlin; Translation edition, 2016), 5.

[18] "SuperCollider IDE," sccode, accessed February 19, 2023, https://doc.sccode.org/Guides/SCIde.html.
[19] Marinos Koutsomichalis, *Mapping and Visualization with SuperCollider: Create interactive and responsive audio-visual applications with SuperCollider* (Birmingham: Packt Publishing Ltd, 2013), 1.
[20] Ibid., 57-79.
[21] Ibid., 81-101.
[22] "SuperCollider," Wikipedia, last modified February 20, 2023, https://en.wikipedia.org/wiki/SuperCollider

This split architecture (scsynth, scland, IDE) has several advantages. On the one hand, it gives the SuperCollider environment high stability and efficiency because each component executes its task independently.[23] On the other hand, it favors modularity and extensibility, allowing a high degree of connectivity with, from, and to different environments, software, and hardware,[24] as well as extending its functionalities through third-party extensions.[25] This degree of openness of SuperCollider allows a customized approach that conveniently accommodates the needs and peculiarities of each user and project.

2. Introduction to topics related to FluCoMa

2.1. Machine Listening

Machine listening, also known as computer audition, is a subfield of artificial intelligence that deals with teaching computers how to interpret and understand audio data.[26] It is also defined as the ability of machines to simulate human hearing and musical skills.[27]

Machine listening combines disciplines such as signal processing, auditory modeling, music perception, pattern recognition, and machine learning, among others. It overlaps with fields such as music information retrieval, acoustic scene analysis, computational musicology, computer music, and machine musicianship.[28]

Its areas of study address various sub-problems such as signal representation, feature extraction, musical knowledge structures, sound similarity, sequence modeling, source separation, auditory cognition, and multi-modal analysis. Its most common applications are

---

[23] Martins, “Estudo exploratório," 58.

[24] Ibid.

[25] Nick Collins, “Extending SuperCollider,” composerprogrammer, accessed March 17, 2023, https://composerprogrammer.com/teaching/supercollider/sctutorial/Technicalities/10%20Extending%20SuperCollider.html

[26] A.I. For Anyone. “machine listening,” A.I. For Anyone, accessed April 22, 2023. https://www.aiforanyone.org/glossary/machine-listening

[27] Nick Collins, “Machine Listening in SuperCollider,” In *The SuperCollider Book*, ed. by Scott Wilson, David Cottle, and NickCollins (Cambridge, Massachusetts London, England: The MIT Press, 2011), 439.

[28] “Computer audition,” Wikipedia, last modified April 5, 2023, https://en.wikipedia.org/wiki/Computer_audition

speech recognition, sound classification, sound localization, speaker recognition, and music recognition.[29]

2.2. Music Information Retrieval

Music Information Retrieval (MIR) is an interdisciplinary field focused on retrieving information from music. It involves disciplines such as musicology, psychoacoustics, psychology, signal processing, informatics, and machine learning, among others.[30]

Applications of MIR include music classification, which involves categorizing music into genres, moods, artists, and instruments. It is also used in recommender systems to suggest music to users based on their listening history. MIR techniques are utilized for music source separation, identifying individual instruments within a complex audio signal. Another area of interest is automatic music transcription, converting audio recordings into symbolic notation or MIDI files. Additionally, MIR researchers aspire to create automated music generation systems, although success in this area still needs to be improved.[31]

The operation of an audio content analysis system consists mainly of two processing stages: extraction of descriptors (also known as audio features) employing audio descriptors and a second stage of inference (or interpretation) of the desired information using classification and regression algorithms, among others.[32]

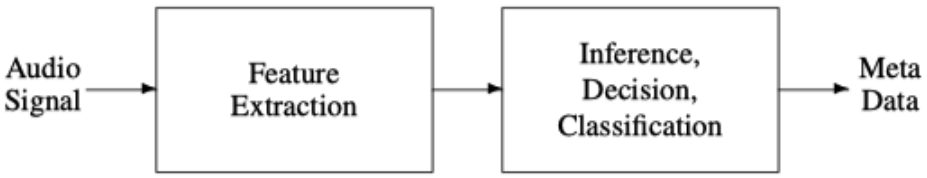

Figure 2 - MIR processing stages
Alexander Lerch, “Audio Content Analysis.” (Preprint, submitted on July 1, 2021), 3, https://arxiv.org/abs/2101.00132.

[29] A.I. For Anyone. “machine listening.”

[30] “Music information retrieval,” Wikipedia, last modified April 19, 2023, https://en.wikipedia.org/wiki/Music_information_retrieval

[31] Ibid.

[32] Alexander Lerch, “Audio Content Analysis.” (Preprint, submitted in July 1, 2021), 3, https://arxiv.org/abs/2101.00132

2.3. Machine Learning

Machine learning is a subfield of artificial intelligence that enables software applications to enhance their predictive accuracy without explicit programming. Machine learning algorithms utilize historical data as input to predict new output values.[33] In the broader context of AI, machines mimic intelligent human behavior to execute complex tasks and problem-solving in a manner similar to humans.[34]

Machine learning encompasses three main categories based on the available "signal" or "feedback" to the learning system: Supervised, Unsupervised, and Reinforcement Learning:

- Supervised learning: “a type of machine learning that utilizes labeled data to train models. In labeled data, the output is already known [...] [, and] the model learns to map inputs to the respective outputs”.[35] Popular supervised learning algorithms include Linear Regression, Logistic Regression, Support Vector Machine, K Nearest Neighbor, Decision Tree, Random Forest, and Naive Bayes.[36]

- Unsupervised learning: “a type of machine learning that uses unlabeled data to train machines”.[37] Unlike supervised learning, unlabeled data lacks a fixed output variable. The model learns from the data, identifies patterns and features, and generates output accordingly. Common examples of unsupervised learning algorithms include K Means Clustering, Hierarchical Clustering, DBSCAN, and Principal Component Analysis.[38]

- Reinforcement Learning: a type of machine learning that “trains a machine to take suitable actions and maximize its rewards in a particular situation. It uses an agent and an environment to produce actions and rewards. The agent has a start and an end state. But, there might be different paths for reaching the end state, like a maze. In this

---

[33] Ed Burns, “machine learning”, TechTarget Editorial, last modified March, 2021, https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[34]Sara Brown, “Machine learning, explained”, MIT Sloan School of Management., April 21, 2021, https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[35] Menon Kartik, “An Introduction to the Types Of Machine Learning,” Simplilearn Solutions, last modified March 10, 2023, https://www.simplilearn.com/tutorials/machine-learning-tutorial/types-of-machine-learning#:~:text=There%20are%20primarily%20three%20types,machine%20learning%20one%20by%20one

[36] Ibid.

[37] Ibid.

[38] Ibid.

learning technique, there is no predefined target variable."[39] Standard reinforcement learning algorithms are Q-learning, Sarsa, Monte Carlo, and Deep Q network.[40]

Machine learning models are mathematical algorithms trained using data and intended to perform prediction and classification operations, among others.[41] These models allow machines to learn complex patterns and relationships, enabling them to generalize and perform tasks without being explicitly programmed for each situation.

Some of the most commonly used data processing and prediction models are Neural networks, Linear regression, Logistic regression, Clustering, Decision trees, and Random forests.[42]

2.4. Artificial Neural Networks

An artificial neural network is a computational model that mimics how the human brain processes information by interconnecting neuron-like processing units. These units are organized into layers, including an input layer, one or more hidden layers (where data is processed), and an output layer. The units are connected with varying weights.

The neural network learns through training, presenting examples with known outcomes and comparing the network's responses to actual outcomes. Through feedback, the weights are gradually adjusted to improve the accuracy of the predictions. Once trained, the network can be applied to new unknown cases to make predictions.[43]

---

[39] Ibid.

[40] Ibid.

[41] "Machine learning," Wikipedia, last Modified March 27, 2023, https://en.wikipedia.org/wiki/Machine_learning

[42] "What is machine learning?" IBM. accessed 22 March, 2023, https://www.ibm.com/topics/machine-learning

[43] "El modelo de redes neuronales," IBM, last modified August 17, 2021, https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model

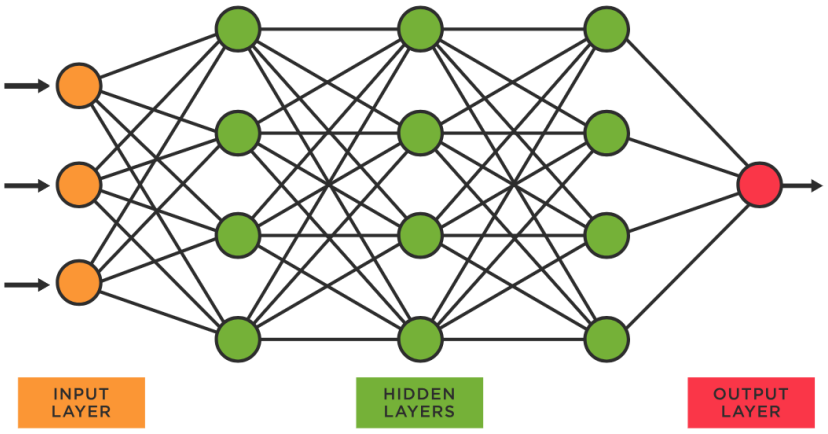

Figure 3 - Artificial neural network structure
TIBCO Software Inc., "What is a Neural Network ?" Tibco, accessed March 19, 2023, https://www.tibco.com/reference-center/what-is-a-neural-network

In an artificial neural network, each artificial neuron receives a stimulus as a numerical signal. The output of each neuron is calculated by a nonlinear function that considers the sum of its inputs.[44]

The connections between neurons are called "edges," and both neurons and edges have an associated weight. During the learning process, these weights are adjusted and changed to improve the neural network's performance.[45]

The weight of a connection can increase or decrease the strength of the signal transmitted between neurons. In addition, neurons can have a threshold, meaning they will only send a signal forward if the accumulated signal crosses that threshold.[46]

[44] TIBCO Software Inc., "What is a Neural Network ?" Tibco, accessed March 19, 2023, https://www.tibco.com/reference-center/what-is-a-neural-network
[45] Ibid.
[46] Ibid.

2.5. The symbiosis between Machine Listening and Machine Learning

Machine learning and machine listening are related because both rely on processing techniques and algorithms to extract useful information from data. Typically, applications involving machine listening benefit from machine learning by using machine learning algorithms and models to better process and understand audio signals. Machine learning often uses analysis and data provided by machine listening algorithms as part of the input data for model training. This interrelation is necessary to improve the accuracy and quality of machine learning models in various tasks, such as pattern recognition, classification, or prediction generation,[47] and to obtain the maximum potential of both technologies.

3. The Fluid Corpus Manipulation (FluCoMa) project

3.1. Introduction

> The FluCoMa Toolkit is much more than the actual code objects that use the "Fluid" prefix. FluCoMa is also a collection of learning resources, code examples, commissioned artworks, musicological articles, interviews, podcasts, a philosophy about interface design for creative coding, a conversation about the future of computer music, a curriculum of machine listening and machine learning topics, a community of users around the world, and more.[48]

The FluCoMa project is an artistic-pedagogical ecosystem focused on providing technological tools and resources related to machine listening and machine learning techniques for music creation and research with sound collections in Creative Code Environments[49] (CCEs).

---

[47] Such operations can be found in timbre recognition applications, where a neural network is recursively trained with sound files to make timbre predictions subsequently: https://learn.flucoma.org/learn/classification-neural-network/#a-brief-synopsis

[48] Ted Moore, James Bradbury, Pierre Alexandre Tremblay, and Owen Green, *FluCoMa for Pedagogues* (Centre for Research into New Music (CeReNeM), 2022), 3.

[49] Such as SuperCollider, Max/MSP, and PureData.

The project aims to enable researchers and technological musicians to discover and develop new workflows related to audio corpora with data-driven techniques.[50] To this end, the project seeks to provide a set of tools and resources flexible enough to allow divergent musical approaches,[51] and adapt to the different levels of *Techno-fluency* of its users.

> "We have tried to capture Techno-Fluency as being a matter of music-technical disposition that takes people's appetite for technical matters and implementation details to be a contextual, rather than genetic, affair: that is, we don't wish to make a priori assumptions that the 'technicity' of someone's practice is a matter of ability, so much as of preference".[52]

In the area of research and knowledge, the FluCoMa project seeks to provide a coherent and accessible framework for musicians and researchers that collects and synthesizes the scattered information around corporate audio practices and expertise. It aims to establish points of connection between this knowledge, technology, and musical practice that enable a deeper understanding of this field.[53] On the other hand, the project seeks to promote further musical research related to this field[54] and make a methodological contribution by analyzing current interdisciplinary collaborative practices between researchers and musicians[55] to clarify and enable new exchanges of ideas between the two communities.

The FluCoMa project became public around 2017 and was (and still is) developed at the Centre for Research in New Music (CeReNeM) of the Department of Music and Music Technology at the University of Huddersfield. Its development took about five years, with the

---

[50] Pierre Alexandre Tremblay, Gerard Roma, Owen Green, and Alex Harker, "From Collections to Corpora: Exploring Sounds through Fluid Decomposition," *Proceedings of the International Computer Music Conference 2019* (September 2019): 295. https://pure.hud.ac.uk/en/publications/from-collections-to-corpora-exploring-sounds-through-fluid-decomp

[51] Owen Green, Pierre Alexandre Tremblay, and Gerard Roma, "Interdisciplinary Research as Musical Experimentation: A case study in musicianly approaches to sound corpora". *Proceedings of the Electroacoustic Music Studies Network Conference 2018* (January 2019): 7. http://www.ems-network.org/spip.php?article471

[52] Ibid., 6.

[53] Moore et al. *FluCoMa for Pedagogues*, 3.

[54] Pierre Alexandre Tremblay, Gerard Roma, and Owen Green, "Digging it: Programmatic Data Mining as Musicking," *Proceedings of the International Computer Music Conference 2021* (July 2021): 296. https://pure.hud.ac.uk/en/publications/digging-it-programmatic-data-mining-as-musicking

[55] Tremblay, Roma, Green, and Harker, "From Collections to Corpora: Exploring Sounds through Fluid Decomposition" 1.

prerelease for all CCEs and source code for version 1.0.0 in May 2020.[56] Its development was made possible through numerous beta and alpha releases, workshops, and feedback from multiple early users.[57]

The main developers of the source code and pedagogical ecosystem were Owen Green, Gerard Roma, Pierre Alexandre Tremblay, James Bradbury, Ted Moore, Jacob Hart, Alex Harker, with Rodrigo Constanzo, Richard Devine, Alice Eldridge, Daniele Ghisi, Leafcutter John, Lauren Hayes, Olivier Pasquet, Sam Pluta and Hans Tutschku as alpha users.[58]

3.2. The FluCoMa resource ecosystem

FluCoMa's ecosystem of resources is divided into three main categories:

- Creative Coding Environment Materials
- Web resources
- Foro de discusión

Firstly, there are the integrated materials of each ECC environment. These provide specific information about the operation of the objects natively within each software.[59] Then there are the resources on the FluCoMa website,[60] that complement the information integrated into the CCEs. These allow for deepening the knowledge surrounding the tools through web references to the learning objects, articles on specific objects, and valuable scientific knowledge.[61] On the other hand, they also aim to encourage artistic creativity and the enrichment of ideas through additional materials such as articles on works of art, interviews with users, musicological and analytical articles, and examples of patches made with FluCoMa.[62] Finally, there is the online discussion forum called Discourse,[63] where, in addition to the usage questions thread, there are threads for sharing code, learning resources, and links of interest.[64]

---

[56] https://github.com/flucoma/flucoma-max/releases/tag/1.0.0-RC1
[57] Moore et al. *FluCoMa for Pedagogues*, 3.
[58] Tremblay, Roma, and Green, “Digging it,” 300.
[59] Moore et al. *FluCoMa for Pedagogues*, 4.
[60] https://learn.flucoma.org/
[61] Moore et al. *FluCoMa for Pedagogues*, 5-6.
[62] Moore et al. *FluCoMa for Pedagogues*, 6.
[63] https://discourse.flucoma.org/
[64] Moore et al. *FluCoMa for Pedagogues*, 6.

All these resources combine to offer users a wide range of information and support in using FluCoMa in a tiered manner[65] that can meet the *Techno-Fluency* of each artist at any project and stage of the creative process.[66]

3.3. The FluCoMa Toolkit

The FluCoMa Toolkit has been designed to integrate natively within the CCEs. They are aimed at people with an intermediate level in the CCE of their choice[67] to provide them with a coherent and self-sufficient general framework.[68] They have both real-time and non real-time processing and functionalities,[69] and have been developed and organized as closely as possible among the CCEs[70] to facilitate communication and exchange among users. They are intended to be flexible and configurable, designed to enable progressive and divergent artistic exploration.[71]

The tools were released through two iterations. The first iteration focused primarily on musical approaches to signal decomposition and description. This included tools for temporal segmentation of signals, spectral separation of sound layers, spectral resynthesis, and audio descriptors. Several utilities for buffer operations were also included.[72]

The second iteration focuses on exploring, interacting, and manipulating audio corpora and sonic data. For this purpose, tools for creating and manipulating datasets, similarity queries, supervised and unsupervised machine learning, scaling, and normalization processing for analysis are proposed. New utilities are also included to facilitate recursive procedures, more buffer operations, and real-time queries.[73]

---

[65] Ibid., 3-4.

[66] Ibid., 3.

[67] Tremblay et al.,"From Collections," 2.

[68] Pierre Alexandre Tremblay, Gerard Roma, and Owen Green, "Digging it: Programmatic Data Mining as Musicking," 296.

[69] Tremblay et al., "From Collections," 2.

[70] Tremblay, Roma, and Owen Green, "Digging it," 296.

[71] Tremblay et al., "From Collections," 2.

[72] Ibid., 4-5.

[73] Tremblay, Roma, and Green, "Digging it," 297-298.

Currently, the FluCoMa Toolkits provided for the SuperCollider software are divided into the following categories:

- Slice Audio
- Analyse Audio
- Decompose Audio
- Transform Audio
- Analyse Data
    - Containers
    - Analyse data
    - Preprocessing
    - Searching and Querying
    - Supervised Machine Learning
    - Unsupervised Machine Learning
- Helpers
    - Buffer Utilities
    - Viewers
    - Corpus Building

The following chart shows a detailed list of all FluCoMa tools in SuperCollider:

Slice Audio		
on signals	**on buffers**	**description**
FluidAmpGate	FluidBufAmpGate	Events from amplitude envelope
FluidAmpSlice	FluidBufAmpSlice	Onsets from amplitude envelope
FluidOnsetSlice	FluidBufOnsetSlice	Spectral onset detector
FluidTransientSlice	FluidBufTransientSlice	Transient model onset detector
FluidNoveltySlice	FluidBufNoveltySlice	Novelty based onset detection on a choice of descriptors
Analyse Audio		
on signal	**on buffers**	**description**

FluidPitch	FluidBufPitch	Pitch descriptors
FluidLoudness	FluidBufLoudness	Loudness Descriptor
FluidMelBands	FluidBufMelBands	Energy in Mel Bands
FluidMFCC	FluidBufMFCC	Timbral Descriptor with Mel Frequency Cepstral Coefficients
FluidSpectralShape	FluidBufSpectralShape	Seven Spectral Shape Descriptors
FluidChroma	FluidBufChroma	Pitch Classes Descriptor
FluidNMFMatch	-	Real-time activation of NMF bases
-	FluidBufNMFSeed	Quick starting estimates for NMF components using Singular Value Decomposition
-	FluidBufSTFT	Perform STFT / ISTFT on Buffers
FluidAmpFeature	FluidBufAmpFeature	Detrending Amplitude Envelope Descriptor
FluidNoveltyFeature	FluidBufNoveltyFeature	Novelty descriptor based on a choice of analysis descriptors
FluidOnsetFeature	FluidBufOnsetFeature	Descriptor comparing spectral frames using a choice of comparisons
FluidSineFeature	FluidBufSineFeature	Sinusoidal peak extraction

Decompose Audio

on signals	**on buffers**	**description**
FluidSines	FluidBufSines	Decompose into sines + residual
FluidTransients	FluidBufTransients	Decompose into transients + residual
FluidHPSS	FluidBufHPSS	Decompose into 'harmonic' and 'percussive' layers
-	FluidBufNMF	Use Nonnegative Matrix Factorisation to explore and decompose sounds

Transform Audio

on signals	**on buffers**	**description**
FluidAudioTransport	FluidBufAudioTransport	Interpolate between sounds using Optimal Transport
FluidNMFFilter	-	Filter sound using NMF bases

FluidNMFMorph	-	Morph between sounds using NMF components
-	FluidBufNMFCross	Cross synthesise buffers using NMF components

Analyse Data

Containers

FluidDataSet	Container that associates data points with identifiers
FluidLabelSet	Container of labels associated with IDs

Analyse Data

on signals	**on buffers**	**description**
FluidStats	FluidBufStats	Compute statistics

Preprocessing

FluidNormalize	Normalize FluidDataSets and Buffers
FluidStandardize	Standardize FluidDataSets and Buffers
FluidRobustScale	Scale FluidDataSets and Buffers using order statistics

Searching and Querying

FluidKDTree	Nearest Neighbour queries on FluidDataSet
FluidDataSetQuery	Construct custom queries on FluidDataSet

Supervised Machine Learning

FluidKNNRegressor	Regression by Nearest Neighbour modelling
FluidKNNClassifier	Classification by Nearest Neighbour modelling
FluidMLPRegressor	Regression using Multilayer Perceptron model
FluidMLPClassifier	Classification by Nearest Neighbour modelling

Unsupervised Machine Learning

FluidPCA	Principal Component Analysis for preprocessing and dimension reduction
FluidMDS	Multidimensional Scaling for dimension reduction
FluidKMeans	K-Means clustering
FluidSKMeans	Spherical K-Means clustering
FluidUMAP	Dimension reduction with UMAP algorithm
FluidGrid	Transform a data set of two dimensional points into a two dimensional grid using the Munkres Algorithm.
Helpers	
Buffer Utilities	
FluidBufCompose	A utility for manipulating the contents of buffers.
FluidBufScale	Remap range of values
FluidBufThresh	Zero elements below threshold
FluidBufSelect	Select range (frame-wise or channel wise)
FluidBufSelectEvery	Select every N elements (frame-wise or channel wise)
FluidBufFlatten	Flatten multichannel data into single channel 'point'
FluidBufToKr	Read data from a buffer into a Kr stream
FluidKrToBuf	Write data into a buffer from a Kr Stream
Viewers	
FluidPlotter	View a FluidDataSet in a plotter window
FluidWaveform	View an audio buffer with overlays, such as slices from a FluCoMa slicer
Corpus Building	
FluidLoadFolder	Load a folder of sounds into a Buffer
FluidSliceCorpus	Batch-slice a corpus
FluidProcessSlices	Batch-analyse slices

Chart 1 - FluCoMa tools in SuperCollider
Accessed from FluCoMa's native SuperCollider help file (FluCoMa version 1.0.5.)

4. Wavesets Synthesis

4.1. Introduction

Composer Trevor Wishart introduced the waveset concept in his book *Audible Design*.[74] [75] He implemented this concept as audio processing within the Computer Desktop Project (CDP) software.[76]

These transformations fall into the Microsound processing and synthesis category[77] and are considered a variant of the grain concept and Granular synthesis,[78] [79] where the chosen grains or segments are now pseudo-wavecycles called wavesets.

A pseudo-wavecycle or waveset is a signal segment between two consecutive zero-crossing pairs. They are called "pseudo" because these waveform segments do not necessarily correspond to real wavecycles of the signal.[80]

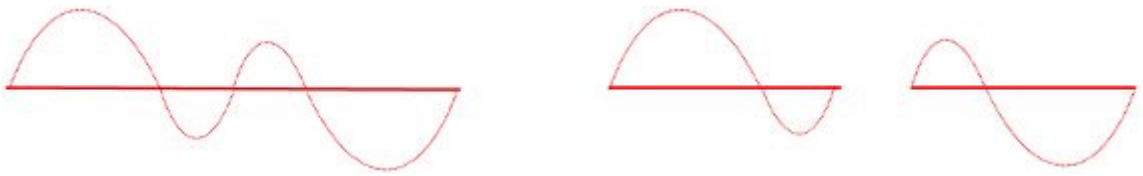

Figure 4 - Real wavecycle (left) consisting of two pseudo-wavecycles (right)
Adapted from: Archer Endric, "CDP 'Wavecycle' DISTORT Functions," ensemble-software, last updated October 30, 2021, http://www.ensemble-software.net/CDPDocs/html/cdistort.htm#DISTORTLIST

---

[74] http://www.trevorwishart.co.uk/AUDIBLE_DESIGN.pdf

[75] Alberto de Campo, "Microsound," In *The SuperCollider Book*, edited by Wilson, Scott, David Cottle, and NickCollins (Cambridge, Massachusetts London, England: The MIT Press, 2011), 491.

[76] http://www.composersdesktop.com/index.html

[77] Curtis Roads, *Microsound* (Cambridge, Massachusetts London, England: The MIT Press, 2001), 205.

[78] Endric, "CDP 'Wavecycle' DISTORT Functions".

[79] "Granular synthesis is a sound synthesis method that operates on the microsound time scale.
It is based on the same principle as sampling. However, the samples are split into small pieces of around 1 to 100 ms in duration. These small pieces are called grains. Multiple grains may be layered on top of each other, and may play at different speeds, phases, volume, and frequency, among other parameters."
extracted from "Granular synthesis," Wikipedia, last modified January 16, 2023.
https://en.wikipedia.org/wiki/Granular_synthesis

[80] Ibid.

The CDP waveset distortion functions are based on manipulating these small audio fragments individually or in groups by repeating, skipping, replacing, transposing, superimposing, reversing, and other methods.[81]

The results of this processing are highly dependent on the input file and the processing applied. If the file tends to be periodic (like a sine wave), the result will be similar to the input file. Conversely, if the input file tends to be aperiodic (noisy or inharmonic sounds), the result will tend to be more unpredictable as a general rule.[82]

Average	Average the waveshape over *N* wavesets
Cyclecnt	Count wavesets in soundfile
Delete	Time-contract soundfile by deleting wavesets
Divide	Distortion by dividing wavesets frequency
Envel	Impose envelope over each group of wavesets
Filter	Time-contract a sound by filtering out waveets
Fractal	Superimpose miniature copies of source wavesets onto themselves
Harmonic	Harmonic distortion by superimposing 'harmonics' onto wavesets
Interact	Time-domain interaction of two sounds
Interpolate	Time-stretch file by repeating and interpolating between them
Multiply	Distortion by multiplying waveset frequency
Omit	Omit *A* out of every *B* wavesets, replacing them with silence
Pitch	Pitchwarp wavesets of sound
Pulsed	Distort a sound by imposing a series of impulses on the source, or on a specific waveset segment of the source.
Reform	This process reads each waveset and replaces it with a different waveform of the same length.
Repeat	Timestretch soundfile by repeating wavesets
Repeat2	Repeat wavesets without time-stretching
Replace	The strongest waveset in a group replaces the others

[81] Ibid.

[82] Ibid.

Replim	Timestretch by repeating wavesets (below a specified frequency)
Reverse	Cycle-reversal distortion in which the wavesets are reversed in groups
Shuffle	Distortion by shuffling wavesets
Telescope	Time-contract sound by telescoping *N* wavesets into 1

Chart 2 - List of CDP Wavesets transformations.
Archer Endric, "CDP 'Wavecycle' DISTORT Functions," ensemble-software, last updated October 30, 2021, http://www.ensemble-software.net/CDPDocs/html/cdistort.htm#DISTORTLIST

4.2. Wavesets Synthesis in SuperCollider

There are several objects to perform wavesets analysis and synthesis in SuperCollider. Their internal implementations and functionalities reveal that they are all based on the class *Wavesets*,[83] originally implemented by Alberto de Campo. The objects perform the following functions in deferred time:

- Wavesets start frame location
- Amplitude analysis and other statistical data on wavesets
- Storage of analysis information for further manipulation

Wavesets	Analyses the wavesets of a monophonic file stored on disk and store its information. Additionally, it loads the analyzed file into a buffer
Wavesets2	Similar to the *Wavesets* class but modified to perform the analysis on a buffered file within SuperCollider
WavesetsEvent *WavesetsMultiEvent*	Optimized for wavesets playback via Events and multi-channel file channel loading. They are based on the *Wavesets2* class

Chart 3 - Standard objects for waveset analysis in SuperCollider

[83] https://github.com/supercollider-quarks/Wavesets

Daniel Mayer,[84] Fabian Seid,[85] and Olaf Hochherz[86] have also implemented objects for real-time and non real-time Wavesets synthesis in SuperCollider.

4.3. Basic implementation

The following codes have been extracted or adapted from SuperCollider's internal help file for the *Wavesets* class.[87]

4.3.1. Waveset analysis and query

The *Wavesets* class performs wavesets analysis on a monophonic file hosted on disk.

```
w = Wavesets.from(String.scDir +/+ "sounds/a11wlk01.wav");
```

Code 1 - Wavesets analysis on a file hosted on disk

Once the file has been analyzed, information can be obtained in a general or specific way by evaluating different methods, some of which are listed in the chart below.

.dump	Contains mainly analysis data
.signal	The audio signal that was analyzed
.name	The wavesets name in the global dictionary
.buffer	A buffer on the server that was created from the same soundfile
.numFrames	The number of frames of the soundfile, the buffer and the Wavesets
.sampleRate	The sample rate of the signal/buffer. default is s.sampleRate
.xings	All integer indices of the zero crossings found
.numXings	Total number of zero crossings found
.lengths	Lengths of all wavesets

[84] Daniel Mayer, "Software," Accessed March 11, 2023, https://www.daniel-mayer.at/software_en.htm

[85] Fabian Seidl, *Granularsynthese mit Wavesets für Live-Anwendungen,* Master's thesis, Technische Universität Berlin, 2016.

[86] Olaf Hochherz, "SPList, a Waveset synthesis library and its usage in the composition "draussen"," *Proceedings of Linux Audio Conference 2008.* (2008): 1-6. https://lac.linuxaudio.org/2008/download/papers/19.pdf

[87] https://github.com/supercollider-quarks/Wavesets/blob/master/HelpSource/Wavesets.schelp

.amps	Peak amplitude of every waveset
.maxima	Indices of positive maximum value in every waveset
.minima	Indices of negative minimum value in every waveset
.fracXings	The calculated fractional zerocrossing points.
.fracLengths	Fractional lengths - in effect, this is 1/wavesetFreq.
.minSet	Shorted waveset
.maxSet	Longest waveset
.avgLength	Average length of all wavesets
.sqrAvgLength	Weighted average length
.minAmp	Softest waveset amplitude
.maxAmp	Loudest waveset amplitude
.avgAmp	Average amplitude of the entire waveset
.sqrAvgAmp	Weighted average of (squared) amplitude of the entire waveset
.plot	Plot a section of <length> Wavesets from <startWs>

Chart 4 - Methods for waveset information retrieval

4.3.2. Basic waveset playback engine

```
(
{
    var startFr, endFr, dur;
    startFr = w.xings[800];              //return frame for waveset 800
    endFr = w.xings[820];                //return frame for Waveset 820
    dur = endFr - startFr / w.buffer.sampleRate;      //frames to dur
    BufRd.ar(1,w.buffer,Line.ar(startFr,endFr,dur,doneAction: 2)) //frames reader
}.play;
)
```

Code 2 - Basic waveset playback engine

The instrument consists of a buffer content reader (*BufRd*), which reads the audio fragment corresponding to the selected waveset or group of wavesets from the analyzed file (now contained in a buffer).

The reading engine for the *BufRd* object is the *Line* object. It reads frames linearly with an initial value (start frame of the waveset), an end value (end frame of the waveset), and a time duration (duration of the waveset).

The evaluation of the code instantiates a Synth that will be extinguished from the server once the linear ramp of the *Line* object is finished (due to the doneAction:2 action), thus avoiding the accumulation of objects and overloading of the audio server.

4.3.3 Specific instruments for waveset playback

A more efficient way to play wavesets is by implementing a specific instrument or SynthDef that allows the instantiation of Synths and a more flexible manipulation of wavesets.

The following SynthDefs come by default inside the *Wavesets* class and can be loaded into the server directly by evaluating the following code:

```
Wavesets.prepareSynthDefs;
```

Code 3 - Load specific wavesets SynthDefs on the server

4.3.3.1. SynthDef(\wvst0)

```
(
SynthDef( \wvst0, {
    arg out = 0, buf = 0, start = 0, length = 441,
    playRate = 1, sustain = 1, amp=0.2, pan;

    var phasor = Phasor.ar(0,BufRateScale.ir(buf) * playRate, 0, length) + start;
    var env = EnvGen.ar(Env([amp, amp, 0], [sustain, 0]), doneAction: 2);
    var snd = BufRd.ar(1, buf, phasor) * env;
    OffsetOut.ar(out, Pan2.ar(snd, pan));
}, \ir.dup(8)).add;
)
```

Code 4 - wvst0 SynthDef

This instrument works similarly to the function in section 4.3.2. The main difference lies in the replacement of the *Line* object by the *Phasor* object. *Phasor* allows loop playback of the

chosen fragment or waveset (necessary for CDP's Repetition transformation, among others). It also enables implementing operations with the playback Rate (necessary for transformations such as Transposition, Reversal, and Harmonic distortion).

On the other hand, *Phasor* lacks the doneAction:2 function, which conveniently removes the Synth from the server when the *Line* ramp reaches its destination. To overcome this, a hard envelope (*EnvGen*) is added to the algorithm, whose only function is to remove the Synth from the server once the corresponding waveset transformation has been performed.

Another difference of this algorithm concerning the one proposed in section 4.3.2. is that the *Phasor* object locates the frames of the chosen waveset in the analyzed file given an offset (start argument) and a number of frames (duration of the chosen waveset in frames). It is the Rate argument that sets the playback speed of these frames.

The sustain variable contains all the necessary calculations to obtain the exact duration that the envelope should last, depending on the type of processing.

sustain = (file duration in seconds * number of repetitions) / PlayRate

- If the waveset lasts 1 second, has 1 repetition, and PlayRate = 1, then:
  sustain = (1 sec * 1 rep) / 1 = 1 sec.
- If the waveset lasts 1 second, has 2 repetitions, and PlayRate = 1, then:
  sustain = (1 sec * 2 rep) /1 = 2 sec.
- If the waveset lasts 1 second, has 2 repetitions, and PlayRate = 2, then:
  sustain = (1 sec * 2 rep) / 2 = 1 sec.
- If the waveset lasts 1 second, has 4 repetitions, and PlayRate = 0.5, then:
  sustain = (1 sec * 4 rep) / 0.5 = 8 sec.

Chart 5 - Basic calculation and examples for sustain variable

The Phasor values are then entered into the *BufRd* object to play the chosen audio fragment.

The manual instantiation of the instrument would be as follows:

```
(
var startWs = 100, length = 6, rep = 10, playRate = 0.5;
var startframe, endframe, sustain;

startframe = w.xings[startWs];
endframe = w.xings[startWs + length];

//duration in seconds
sustain = (endframe - startframe) * rep / playRate / w.sampleRate;

Synth("wvst0", [
    \bufnum, w.buffer,
    \start, startframe,
    \length, endframe - startframe,
    \playRate, playRate,
    \sustain, sustain,
    \amp, 1
]);
)
```

Code 5 - Manual instantiation of wvst0

4.3.3.2. SynthDef(\wvst1gl)

```
(
SynthDef(\wvst1gl, {
    arg out = 0, buf = 0, start = 0, length = 441, playRate = 1, playRate2 = 1,
    sustain = 1, amp = 0.2, pan;

    var playRateEnv = Line.ar(playRate, playRate2, sustain);
    var phasor = Phasor.ar(0,BufRateScale.ir(buf) * playRateEnv,0,length) +
start;
    var env = EnvGen.ar(Env([amp, amp, 0], [sustain, 0]), doneAction: 2);
    var snd = BufRd.ar(1,buf,phasor) * env;

    OffsetOut.ar(out, Pan2.ar(snd, pan));
}, \ir.dup(8)).add;
)
```

Code 6 - wvst1gl SynthDef

The wvst1gl instrument is similar to the wvst0 instrument with the particularity that it implements a modulation for the Rate argument of the *Phasor* object. It uses a *Line* object that implements the interpolation between two values at a specific duration set by the sustain variable. In this way, all calculations for the envelope duration are still scaled and synchronized with the *Phasor* object.

On the other hand, the PlayRate value is also eliminated for calculating the duration of the sustain variable.

sustain = (duration of the file in seconds * number of repetitions)

The manual instantiation of the Synth would be as follows:

```
(
var startWs = 100, length = 100, rep = 100, playRate = 1, playRate2 = 2;
var startframe, endframe, sustain;

startframe = w.xings[startWs];
endframe = w.xings[startWs + length];
sustain = (endframe - startframe) * rep / w.sampleRate;  //duration in seconds

sustain.postln;

Synth("wvst1gl", [
    \bufnum, w.buffer,
    \start, startframe,
    \length, endframe - startframe,
    \playRate, playRate,
    \playRate2, playRate2,
    \sustain, sustain,
    \amp, 1
]);
)
```

Code 7 - Manual instantiation of wvst1gl

4.3.4. Brief note on the instantiation of wavesets transformations in SuperCollider

There are many ways to instantiate Synths for wavesets playback in SuperCollider. Many of these methods are described in detail in the internal help file of the SuperCollider *Wavesets* object and in the "Microsound" chapter of *The SuperCollider Book*.[88] In turn, most of the wavesets transformations proposed in the CDP environment have been implemented by de Campo[89] and Martins.[90]

[88] de Campo, “Microsound,” 491-503.
[89] Ibid.
[90] Martins, “Estudo exploratório", 87-124.

II. Implementations

5.1. 2D Waveset Corpus Explorer

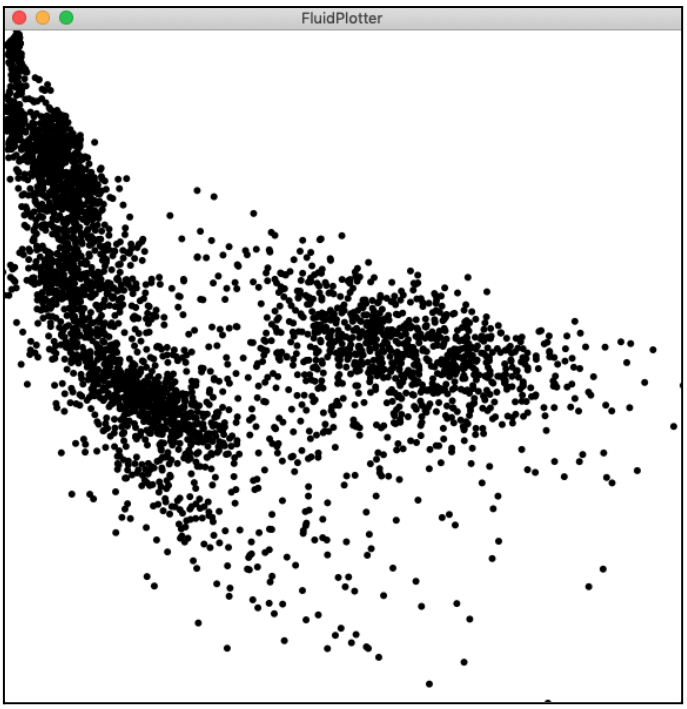

Figure 5 - 2-dimensional wavesets plotter
Elaborated by the author

5.1.1. Introduction

The instrument consists of a 2-dimensional plotter and a waveset corpus player.[91] Each waveset is represented by a point on the 2-dimensional plotter, organized according to its

[91]An explanatory video as well as the code of the instrument can be found at the following link: https://drive.google.com/drive/folders/1YuIMaLMCr15J_xMqu1KxeAi_D1MgZiSa?usp=drive_link

analytical spectral centroid energy[92] values (horizontal axis) and loudness[93] values (vertical axis). A player enables independent listening of wavesets.

The instrument is useful for the interactive exploration and analysis of wavesets. The correspondence between visual and auditory information provides intuitive and direct knowledge of the corpus and relevant general information (trends and distributions of wavesets according to their spectral centroid energy and loudness).

This instrument is an adaptation of the "plotter-3" code.[94] [95]

5.1.2. Implementation

A wavesets analysis is performed on a monophonic audio file contained in a buffer using the *Wavesets2* class.

```
~source = Buffer.read(s, Platform.resourceDir +/+ "sounds/a11wlk01.wav");
~waveSet = Wavesets2.fromBuffer( ~source );
```

Code 8 - Wavesets analysis on a file hosted on a buffer

The start frame information for each waveset is stored in a separate buffer.

```
~xings = ~waveSet.xings;
~bufferIndex = Buffer.loadCollection(s, ~xings);
~bufferIndex.loadToFloatArray( action: { arg indexWs; indexWs.postln } );
```

Code 9 - Storage of wavesets indexes in a buffer

---

[92] "The spectral centroid is the center of mass of a magnitude spectrum, i.e., the frequency at which the spectral magnitudes can be separated into two equal parts. That means that low frequency signals will have a low centroid while substantial high frequency components and noise will increase the centroid. Despite its technical definition, the spectral centroid has been shown to be strongly correlated to the perceptual sound attribute brightness." extracted from Lerch, "Audio Content Analysis," 20.

[93] "Our perception of how "loud" a sound is can be complex to understand and measure. Human hearing is not constant or linear across the range of frequencies that we are sensitive to. It is possible to have sounds that register as loud when measured that nonetheless are perceived as weak (and vice versa) depending on their frequency content. Loudness is an audio descriptor that attempts to model such characteristics of human hearing according to the EBU R128 specification." Extracted from FluCoMa, "Loudness," FluCoMa Learn, accessed March 24, 2023, https://learn.flucoma.org/reference/loudness/

[94] https://learn.flucoma.org/learn/2d-corpus-explorer/

[95] https://www.youtube.com/watch?v=qom6x1u4_6A

With the *FluidWaveform* object, we can visualize the wavesets on the waveform. This object is intended to visualize the events detected by FluCoMa time segmentation objects such as *FluidBufAmpGate*, *FluidBufOnsetSlice*, etc., whose volume of data to visualize is significantly smaller.

```
FluidWaveform( ~source, ~bufferIndex);
```

Code 10 - Display of all analyzed wavesets

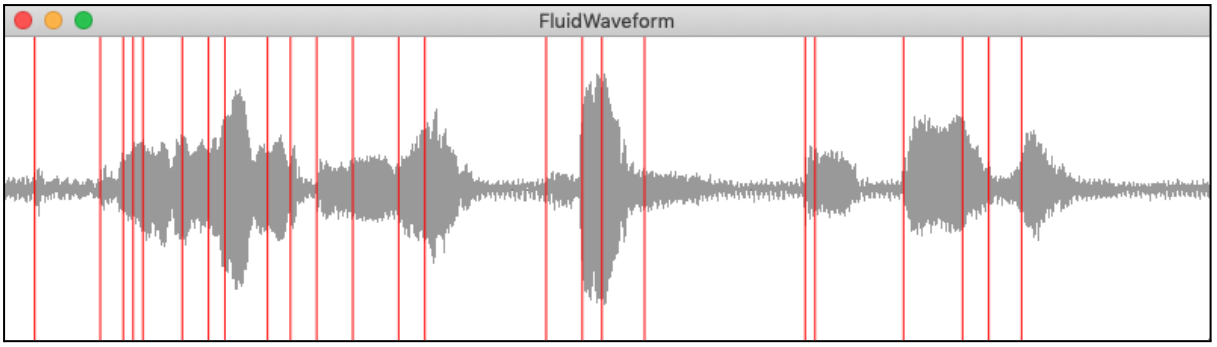

Figure 6 - Display of events retrieved by the *FluidBufOnsetSlice* object
Elaborated by the author

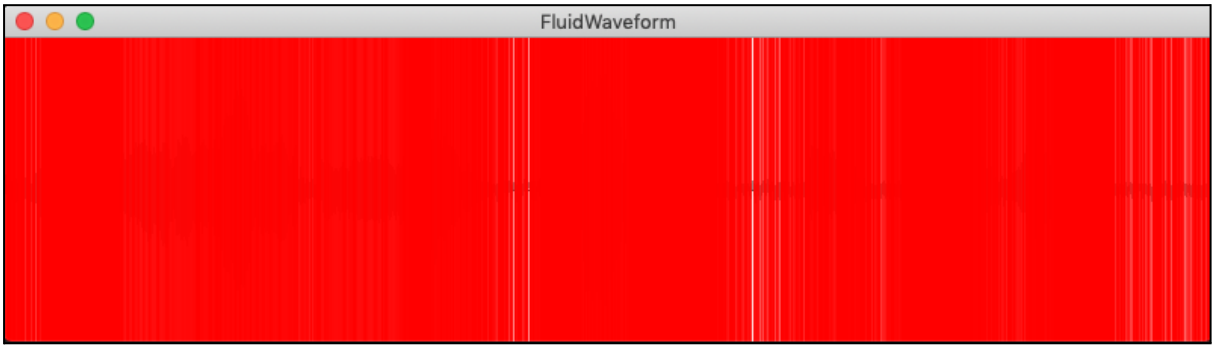

Figure 7 - Display of events retrieved by the *Wavesets* object
Elaborated by the author

Each audio segment corresponding to each waveset is analyzed with descriptors. In this case, two types of descriptors have been used: a spectral shape descriptor to obtain centroid values (*FluidBufSpectralShape*) and a loudness descriptor to obtain loudness values (*FluidBufLoudness*). Because these descriptors perform analyses on each frame of each waveset and these have different temporal durations (different number of frames), it is necessary to reanalyze this analytical data to obtain a single meaningful value for each waveset. This is done using the *FluidBufStats* object, which returns a single value, the mean in this case, for each of the analyses.

The new analysis values are stored in a dataset with a unique identifier using the *FluidDataSet* object.

```
(
~dataSetWS = FluidDataSet(s);

~bufferIndex.loadToFloatArray(action:{
arg wavesets;

var spec = Buffer(s);
var loudness = Buffer(s);

var stats = Buffer(s);
var stats2 = Buffer(s);

var point = Buffer(s);

wavesets.doAdjacentPairs {    //adjacent pair of frames analysis (Waveset
duration)
arg start, end, i;            //first value-second value-iteration
var num = end - start;        //waveset length in frames

//analyzes centroid information and store them in "spec" buffer
FluidBufSpectralShape.processBlocking(s,~source,start,num,features:spec,select:[\
centroid]);
//get centroid statistical mean value and store it in "stats" buffer
FluidBufStats.processBlocking(s,spec,stats:stats,select:[\mean]);


//analyzes loudness information and store them in "loudness" buffer
FluidBufLoudness.processBlocking(s,~source,start,num,features:loudness,select:[\l
oudness]);
//get loudness statistical mead value and store it "stats2" buffer
FluidBufStats.processBlocking(s,loudness,stats:stats2,select:[\mean]);


//store centroid statistical mean values in "Point" buffer
FluidBufCompose.processBlocking(s,stats,destination:point,destStartFrame:0);

//store statistical loudness mean values in "Point" buffer
FluidBufCompose.processBlocking(s,stats2,destination:point,destStartFrame:1);
```

```
//load "point" buffer data into a database with unique id
~dataSetWS.addPoint(i,point);

"waveset % / %".format(i,wavesets.size).postln;

if((i%100) == 99){s.sync};
};

s.sync;

~dataSetWS.print;   //print dataset content
});
)
```

Code 11 - Analysis of wavesets with descriptors and storage in dataset

For a correct performance of the KDTree functions and accurate representation of points in the 2-dimensional space, it is necessary to scale the centroid and loudness values stored in the dataset. For this purpose, they are normalized to a standard range (0-1.0) with the *FluidNormalize* object and stored in a new dataset.

```
(
~normedDataSetWS = FluidDataSet(s);
FluidNormalize(s).fitTransform( ~dataSetWS, ~normedDataSetWS);
)

~normedDataSetWS.print;
```

Code 12 - Standardization and storage of analysis values in dataset

Non-normalized data	Normalized data
rows: 2180 cols: 2	rows: 2180 cols: 2
0 6460.6 -84.098	0 0.54991 0.16404
1 9087.7 -83.848	1 0.77795 0.16715
2 6754.4 -69.223	2 0.57541 0.34906
...	...
2177 134.8 -66.837	2177 0.00082556 0.37874
2178 137 -67.751	2178 0.0010162 0.36738
2179 133.6 -68.614	2179 0.00072165 0.35664

Chart 6 - examples of standardized and non-standardized values

A KDTree[96] is fitted to the normalized analysis dataset employing the *FluidKDTree* object. It enables the search for nearest neighbor points under the mouse pointer in 2-dimensional space.

```
~tree = FluidKDTree(s).fit(~normedDataSetWS);
```

Code 13 - Fit normalized values in KDTree

The normalized dataset information is loaded into the 2-dimensional space viewer employing the FluidPlotter object. This object enables the implementation of custom functions for the mouse behavior. The following procedure is implemented:

1. Post nearest current values (index, centroid, and loudness)
2. Highlight the selected point in the 2-dimensional space
3. Instances the player with the selected index
4. Set current point as “previous” for a new comparison

```
(
~normedDataSetWS.dump({   //get the information of the “~normedDataSetWS” back to the
lenguaje
arg dict;
var point = Buffer.alloc( s, 2);     //mouse position buffer for KDTree
var previous = nil;

dict.postln;

defer{
FluidPlotter(
dict: dict,                     //load Dataset information into the Plotter

mouseMoveAction:{
arg view, x, y;
[x,y].postln;                   //print mouse position values
point.setn(0,[x,y]);            //store mouse position values in the "point" buffer

~tree.kNearest(point,1,{        //search for nearest neighbor points at the mouse position
arg nearest;
if(nearest != previous){        //if the chosen point is not equal to the previous point
then:
```

[96] “In computer science, a k-d tree (short for k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space. k-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key (e.g. range searches and nearest neighbor searches) and creating point clouds. k-d trees are a special case of binary space partitioning trees.” extracted from “k-d tree,” Wikipedia, last modified February 28, 2023, https://en.wikipedia.org/wiki/K-d_tree

```
nearest.postln;              //1 - post nearest current values (index, centroid and
loudness)
view.highlight_(nearest);    //2 - highlight the selected point
~wavesetPlayer.(nearest.asInteger);    // 3 - instance the player with the selected index

previous = nearest;           //4 - set current point as previous for a new comparison
}
});
});
};
});
)
```

Code 14 - Custom function for mouse behavior

The instrument in charge of reproducing the wavesets is similar to the SynthDef(\wvst0) instrument; the main difference is that the start and end frames are provided as control signals by index.

```
(
~wavesetPlayer = {
arg index;             //input ~wavesetPlayer.(nearest.asInteger) value
{
var startsamp = Index.kr( ~bufferIndex, index);
var stopsamp = Index.kr( ~bufferIndex, index+1);

var phs = Phasor.ar( 0, BufRateScale.ir( ~source), startsamp, stopsamp);
var sig = BufRd.ar( 1, ~source, phs);
var dursecs = (stopsamp - startsamp) / BufSampleRate.ir( ~source);
var  env  =  EnvGen.kr(  Env(  [  0,  1,  1,  0], [  0.03,  dursecs  -  0.06,  0.03] ),
doneAction: 2);

sig.dup * env
}.play;
};
)
```

Code 15 - 2D wavesets playback engine

5.2. Waveset Corpus Concatenation

5.2.1. Introduction

The instrument[97] implements Concatenative synthesis[98] from two corpora (source and target) or sets of wavesets extracted from two different audio files (one for each corpus). An MFCC[99] analysis is performed on each waveset and stored with a unique identifier in datasets (one for each corpus). A KDTree searches for the most similar wavesets between the two corpora. If, for the waveset chosen in the target corpus, the algorithm finds a similar waveset in the source corpus, then this new waveset is played instead.

This instrument is an adaptation of the "audio_query_with_scaler" code.[100]

5.2.2. Implementation

Waveset analysis is performed on two monophonic files (one for the source corpus and one for the target corpus) using the *Wavesets2* class.

```
(
~source_buf = Buffer.read(s,
"/Users/carlosgonzalezbolanos/Desktop/audiosParaPruebas/_quickSounds/voz_2_mono.w
av" );
~target_buf = Buffer.read(s,
"/Users/carlosgonzalezbolanos/Desktop/audiosParaPruebas/_quickSounds/voz_1_mono.w
av" );
);
```

[97] An explanatory video as well as the code of the instrument can be found at the following link: https://drive.google.com/drive/folders/1FYHArG1GgLfXIlK2RDfSNCdE3LLEnBq-?usp=sharing

[98] "Concatenative synthesis is a technique for synthesizing sounds by concatenating short samples of recorded sound (called units). The duration of the units is not strictly defined and may vary according to the implementation, roughly in the range of 10 milliseconds up to 1 second. It is used in speech synthesis and music sound synthesis to generate user-specified sequences of sound from a database (often called a corpus) built from recordings of other sequences.
In contrast to granular synthesis, concatenative synthesis is driven by an analysis of the source sound, in order to identify the units that best match the specified criterion." extracted from "Concatenative synthesis," Wikipedia, last modified November 5, 2022, https://en.wikipedia.org/wiki/Concatenative_synthesis

[99] "MFCC stands for Mel-Frequency Cepstral Coefficients [...]. This analysis is often used for timbral description and timbral comparison. It compresses the overall spectrum into a smaller number of coefficients that, when taken together, describe the general contour of the spectrum." extracted from FluCoMa, "MFCC," FluCoMa Learn, accessed March 24, 2023, https://learn.flucoma.org/reference/mfcc/.

[100] https://discourse.flucoma.org/t/audio-query-in-supercollider-demo/1149

```
(
~waveset_source = Wavesets2.fromBuffer( ~source_buf);
~waveset_target = Wavesets2.fromBuffer( ~target_buf);
);

(
~xings_source = ~waveset_source.xings;
~xings_target = ~waveset_target.xings;
);

(
~source_index_buf = Buffer.loadCollection(s, ~xings_source);
~target_index_buf = Buffer.loadCollection(s, ~xings_target);

);
```

Code 16 - Wavesets analysis and storage of indexes in buffers

A function is implemented to perform MFCC analysis with 13 coefficients. The function analyzes each of the wavesets of the selected corpus using the *FluidBufMfCC* object. Because each waveset has a different number of frames, and *FluidBufMfCC* returns an analysis for each frame of the waveset, it is necessary to reanalyze each analysis to obtain a single meaningful summary statistic per waveset (this is done using the *FluidBufStats* object). Using the *FluidBufFlatten* object, the mean value of these analyses is extracted. Subsequently, they are added to a dataset with a unique identifier.

```
(
~analyze_to_dataset = {
arg audio_buffer, wavesets_buffer, action; //the audio buffer to analyze, a buffer with
//the slice points, and an action to execute when done

var features_buf = Buffer(s); //a buffer for writing the MFCC analyses into
var stats_buf = Buffer(s);    //a buffer for writing the statistical summary of the MFCC
//analyses into
var flat_buf = Buffer(s); // a buffer for writing only he mean MFCC values into
var dataset = FluidDataSet(s); // the dataset that all of these analyses will be stored in
~nmfccs = 13;

//get the indices from the server loaded down to the language
wavesets_buffer.loadToFloatArray(action:{ arg wavesets_array;

fork{
//iterate over each index in this array, paired with this next //neighbor so that we know
//where to start and stop the analysis
wavesets_array.doAdjacentPairs{
arg start_frame, end_frame, waveset_index;
var num_frames = end_frame - start_frame;

"analyzing waveset: % / %".format(waveset_index + 1,wavesets_array.size - 1).postln;
```

```
//mfcc analysis, hop over that 0th coefficient because it relates to loudness and here we
//want to focus on timbre
FluidBufMFCC.processBlocking(s,audio_buffer,start_frame,num_frames,features:features_buf,st
artCoeff:1,numCoeffs: ~nmfccs);

//get a statistical summary of the MFCC analysis for this waveset
FluidBufStats.processBlocking(s,features_buf,stats:stats_buf,select:[\mean]);

//extract and flatten just the 0th frame (numFrames:1) of the statistical summary (because
//that is the mean)

FluidBufFlatten.processBlocking(s,stats_buf,destination:flat_buf);

//now that the means are extracted and flattened, we can add this datapoint to the
//dataset:
dataset.addPoint("waveset-%".format(waveset_index),flat_buf);

if((waveset_index % 100) == 99){s.sync};

};

s.sync;

action.value(dataset); // execute the function and pass in the dataset that was created!
};
});
};
)
```

Code 17 - Custom function for MFCC analysis on a specific corpus
and storage of analyzed values in datasets

Execute source and target corpus analysis:

```
(
~analyze_to_dataset.( ~source_buf, ~source_index_buf,{
arg ds;
~source_dataset = ds;
~source_dataset.print;
});
)

(
~analyze_to_dataset.(~target_buf,~target_index_buf,{
arg ds;
~target_dataset = ds;
~target_dataset.print;
});
)
```

Code 18 - Perform the analysis of the selected corpus

The mean values of the source dataset are normalized (range 0-1.0) with the *FluidNormalize* object and stored in a new dataset. They are then fitted to a KDTree utilizing the *FluidKDTree* object.

```
~scaled_dataset = FluidDataSet(s);      //a dataset for scaled values
~scaler = FluidNormalize(s);            //a function to normalize values

~scaler.fitTransform(~source_dataset, ~scaled_dataset);

~kdtree = FluidKDTree(s);               //create a KDTree
~kdtree.fit(~scaled_dataset,{"kdtree fit".postln;});
```

Code 19 - Standardization function and KDTree

The search for similar wavesets between the two corpora is performed by a function that implements the following actions:

1. Select by index a waveset and its analysis from the target corpus, and store it in a buffer (query_buf)
2. Normalize the found point and store it in a buffer
3. Search for the most similar waveset according to analytical values in the source corpus stored in the KDTree
4. Instantiate the player and play the new waveset for the duration of the chosen target waveset

```
(
fork{
var query_buf = Buffer.alloc(s,~nmfccs);   //a buffer for doing the neighbor lookup with
var scaled_buf = Buffer.alloc(s,~nmfccs);

~target_index_buf.loadToFloatArray(action:{
arg target_index_array;

//prepend 0 (the start  of the file) to the indices array
target_index_array = [0] ++ target_index_array;

//append the total number of frames to know how long to play the last slice for
target_index_array = target_index_array ++ [~target_buf.numFrames];


inf.do{
arg i;

//get the index to play by modulo one less than the number of slices
var index = i % (target_index_array.size - 1);
```

```
// nb. that the minus one is so that the waveset from the beginning of the file to the
//first //index is call "-1"
// this is because that waveset didn't actually get analyzed
var waveset_id = index - 1;
var start_frame = target_index_array[index];
var dur_frames = target_index_array[index + 1] - start_frame;

// this will be used to space out the source wavesets according to the target timings
var dur_secs = dur_frames / ~target_buf.sampleRate;

"target waveset: %".format(waveset_id).postln;


// as long as this slice is not the one that starts at the beginning of the file (-1) and
// not the slice at the end of the file (because neither of these have analyses), let's
// do the lookup

if((waveset_id >= 0) && (waveset_id < (target_index_array.size - 3)),{

//1 - use the waveset id to (re)create the waveset identifier and load the data point into
//"query_buf"
~target_dataset.getPoint("waveset-%".format(waveset_id.asInteger), query_buf,{
//2 - once it's loaded, scale it using the scaler
~scaler.transformPoint( query_buf, scaled_buf,{
//3 - once it's neighbour data point in the kdtree of source waveset
~kdtree.kNearest( scaled_buf ,1,{
arg nearest;
//4 - peel off just the integer part of the waveset to use in the ~wavesetPlayerConca
//instrument
var nearest_index = nearest.asString.split($-)[1].asInteger;
nearest_index.postln;
~wavesetPlayerConca.(nearest_index,dur_secs);
});
});
});
});
//dur_secs.wait;    //not optimized
0.1.wait;
};
});
};
)
```

Code 20 - Customized neighbor search function

The wavesets player is as follows:

```
(
~wavesetPlayerConca = {
arg index, src_dur;
{
//lookup the start frame with the index *one the server* using Index.kr
var start_frame = Index.kr(~source_index_buf,index);
//same for the end frame
var end_frame = Index.kr(~source_index_buf,index+1);
var num_frames = end_frame - start_frame;
var dur_secs = min(num_frames / SampleRate.ir(~source_buf),src_dur);
```

```
var sig =
PlayBuf.ar(1,~source_buf,BufRateScale.ir(~source_buf),0,start_frame,0,2);
var env = EnvGen.kr(Env([0,1,1,0],[0.03,dur_secs-0.06,0.03]),doneAction:2);

//include this env if you like, but keep the line above because it will free the
//synth after the waveset
sig = sig * env;
OffsetOut.ar(0, sig.dup);
}.play;
};
)
```

Code 21 - Concatenative wavesets playback engine

5.3. Wavesets Regressor

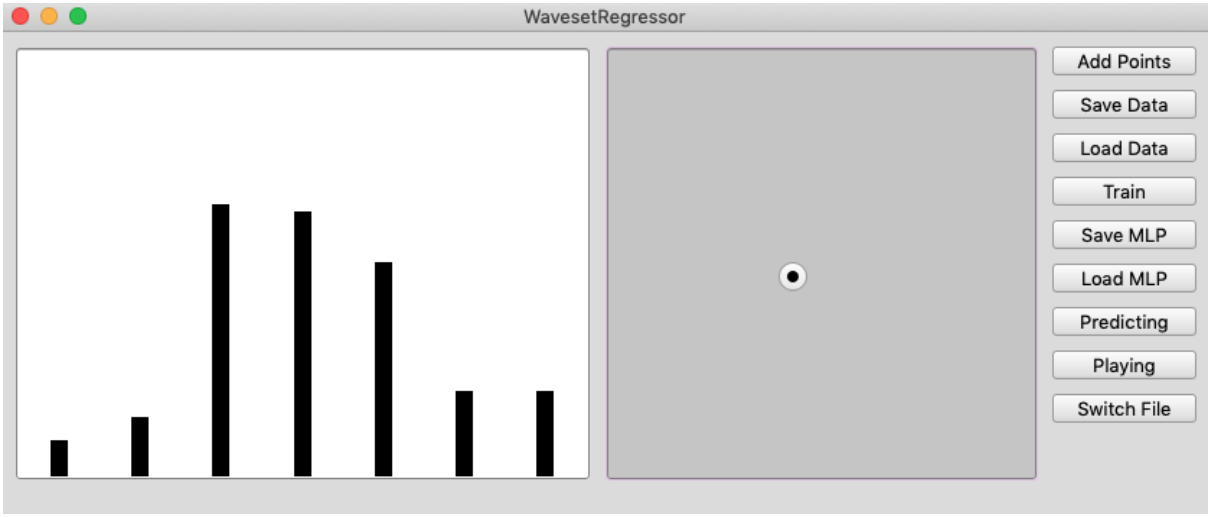

Figure 8 - *Wavesets Regressor* instrument GUI
Elaborated by the author

5.3.1. Introduction

The instrument[101] consists of a wavesets player whose parameters are controlled by a neural network (Multi-Layer Perceptron)[102] using the regression strategy: the neural network predicts new output values from a set of input values. For this task, input and output examples associated with a unique identifier must be introduced to the neural network.

In this instrument, input and output values correspond to arbitrary points stored in datasets with an identifier that associates a specific XY coordinate in a 2-dimensional space to seven output values that are subsequently assigned to instrument parameters. These points are fed into the neural network, which will try to predict new points based on them. The more we train the neural network, the better mathematical predictions it will make (which does not mean that they are musically or expressively superior).

The instrument is especially useful for the interactive exploration of musical gestures as it favors the exploration of focused and generalized musical spaces. The predictions made by the neural network often produce unexpected values of high musical value. On the other hand, the instrument allows the general multiparametric control of the instrument in a highly intuitive way.

The instrument is based on the code "regressor-video-complete-server code".[103]

5.3.2. Implementation

5.3.2.1. The wavesets player

A wavesets analysis is performed on a monophonic file using the *WavesetsEvent* class, and the SynthDef(\wvst1gl) is loaded on the server.

---

[101] An explanatory video as well as the code of the instrument can be found at the following link: https://drive.google.com/drive/folders/1TmXip2mZY9aDiUNuV9h1KsItBpYSjgrg?usp=sharing

[102] "The MLPRegressor is a neural network that can be used to perform regression. In machine learning, regression can be thought of as a mapping from one space to another where each space can be any number of dimensions. [...] By providing input and output data [...] the neural network is trained using *supervised learning* to predict output data points based on input data points." extracted from FluCoMa, "MLPRegressor," FluCoMa Learn, accessed March 25, 2023, https://learn.flucoma.org/reference/mlpregressor/

[103] https://learn.flucoma.org/learn/regression-neural-network/

```
~wavesets = WavesetsEvent.read(Platform.resourceDir +/+
"sounds/a11wlk01-44_1.aiff");


WavesetsEvent.prepareSynthDefs;
```

Code 22 - Wavesets analysis and SynthDef loading

The instantiation of wavesets is done by means of an iterator (.do function) inside a Task Definition (*Tdef*). This combination of objects creates independent events for each waveset or group of wavesets automatically. For each event, values for each instrument parameter are entered, either manually configured in the multislider or generated through regression by the neural network. These values are scaled within useful ranges for each Wavesets Synthesis parameter.

```
Tdef(\waveset, {
inf.do{ |i|
var event = ~wavesets.asEvent((

//from which waveset to start + waveset shuffle
start: ((i % (~wavesets.size - shufMax)) + ~shuf.linlin(0,1,0,shufMax) ).round,
//how many wavesets to play
num: ~numWs.linlin(0, 1, 1, 50).round,
//how many times to repeat the selected wavesets
repeats: ~repeats.linlin(0, 1, 1, 20).round,
//playback speed of the audio file
rate: ~playRate.linlin(0, 1, 0.2, 10),
//end playback speed of the audio file (will create a linear glisson sweep)
rate2: ~playRate2.linlin(0, 1, 0.2, 10),
pan: 0 + ~pan.rand.round(0.1) * [1.0, -1.0].choose,
//scales the duration, so that wavesets will overlap or have gaps between them.
legato: ~legato.linlin(0, 1, 0.1, 2),
));
event.play;
event[\dur].wait;
}
});
```

Code 23 - Wavesets playback engine

5.3.2.2. Neural Network

The neural network is implemented by using the *FluidMLPRegressor* object. A single hidden layer neural network with seven neurons is configured to receive and produce input and output values greater than 0 and smaller than 1. The neural network is configured to use each input-output example pair 1000 times in each training before returning the percentage error.

```
var mlp = FluidMLPRegressor(s,
[7],
activation:FluidMLPRegressor.sigmoid,
outputActivation:FluidMLPRegressor.sigmoid,
maxIter: 1000,
learnRate:0.1,
batchSize:1,
validation:0
);
```

Code 24 - Neural network parameters

5.3.2.3. Graphical User Interface

The graphical interface consists of three main elements:

- A 2-dimensional display with interactive control of XY values (*Slider2D*)
- An interactive multiband display with 7 bands (*MultiSliderView*)
- Set of buttons (*Button*)

The *Slider2D* produces values between 0 and 1, which is ideal since this is the range the neural network can accept. These values are continuously stored in a buffer (xybuf). A function allows or disallows the routing of these values to the neural network. If the function allows routing, they will be used as input values for active prediction in the neural network. The seven new values predicted by the neural network are stored in a buffer (paramsbuf) and assigned to the synthesis parameters.

```
xyslider = Slider2D(win,Rect(420,10,300,300)).action_{
arg view;                                    //the Slider2D itself

xybuf.setn(0, [view.x, view.y]);             //store XY values in buffer "xybuf"


if( predicting) {   //if true, then:         //prediction function

//"make a prediction"
mlp.predictPoint( xybuf, paramsbuf, {    //store predicted values in paramsbuf

//assigns predicted values to parameters
paramsbuf.getn(0, 7, {
arg prediction;
prediction.postln;

~numWs = prediction[0];
~repeats = prediction[1];
~playRate = prediction[2];
~playRate2 = prediction[3];
~pan = prediction[4];
~shuf = prediction[5];
~legato = prediction[6];

//update values in multislider
defer{
multislider.value = prediction;
};
});
});
};
};
```

Code 25 - Customized *Slider2D* function

Similar to the *Slider2D* object, the *MultiSliderView* object produces and receives values between 0 and 1. It has two main functions: the display of neural network values (if the neural network is predicting) and manual modification of the synthesis parameters. For the latter case, these seven values are also stored in the parambuf buffer and assigned to each synthesis parameter individually.

```
//Multislider for synthesis parameter
multislider = MultiSliderView(win, Rect(10, 10, 400, 300))
.elasticMode_(1)
.isFilled_(1)
.action_{
arg ms;                      //ms - multislider values
paramsbuf.setn(0, ms.value); //store multislider values in buffer "parambuf"
ms.value.postln;
```

```
//assigns multislider values to synthesis parameters manually
~numWs = ms.value[0];
~repeats = ms.value[1];
~playRate = ms.value[2];
~playRate2 = ms.value[3];
~pan = ms.value[4];
~shuf = ms.value[5];
~legato = ms.value[6];
}.value_(0.25.dup(7));
```

Code 26 - Customized *MultiSliderView* function

Additionally, a series of buttons are included in the interface that enable the following functions:

1. Associated storage on datasets of *Slider2D* and *MultiSliderView* values as points with unique identifiers
2. To export these datasets as JSON files. The JSON format enables easy interconnectivity as they can be opened in any CCE
3. To load point datasets from JSON files
4. To train the neural network
5. To export the neural network as JSON file
6. To load a neural network from a JSON file
7. To enable or disable neural network prediction
8. To enable the instantiation of new wavesets (play/stop functions)
9. To load a new audio file and perform its wavesets analysis

```
//1 - add point to Dataset
Button(win, Rect( 730, 10, 100, 20))
.states_([["Add Points"]])
.action_{
var id = "point-%".format(counter);

//add datapoint to Dataset with unique ID
xydata.addPoint( id, xybuf);
paramsdata.addPoint( id, paramsbuf);

counter = counter + 1;

xydata.print;
paramsdata.print;
};
```

```
//2 - save Dataset as json file
Button(win, Rect( 730, 40, 100, 20))
.states_([["Save Data"]])
.action_{
xydata.write("/Users/carlosgonzalezbolanos/Desktop/xydata.json");
xydata.write("/Users/carlosgonzalezbolanos/Desktop/paramsdata.json");
};

//3 - load a Dasaset from a json file
Button(win, Rect( 730, 70, 100, 20))
.states_([["Load Data"]])
.action_{
xydata.read("/Users/carlosgonzalezbolanos/Desktop/xydata.json");
xydata.read("/Users/carlosgonzalezbolanos/Desktop/paramsdata.json");
};

//4 - train the neural network
Button(win, Rect( 730, 100, 100, 20))
.states_([["Train"]])
.action_{

//input data and output data example pairs
mlp.fit( xydata, paramsdata, {
arg loss;
loss.postln;
});
};

//5 - saves the neural network as a json file
Button(win, Rect( 730, 130, 100, 20))
.states_([["Save MLP"]])
.action_{
mlp.write( "/Users/carlosgonzalezbolanos/Desktop/mlp.json");
"mlp saved".postln;
};

//6 - load a neural network from a json file
Button(win, Rect( 730, 160, 100, 20))
.states_([["Load MLP"]])
.action_{
mlp.read( "/Users/carlosgonzalezbolanos/Desktop/mlp.json");
"mlp loaded".postln;
};

//7 - boolean, allows the neural network to predict
Button(win, Rect( 730, 190, 100, 20))
.states_([["Not Predicting"], ["Predicting" ]])
.action_{
arg but;
predicting = but.value.asBoolean;
};

//8 - plays or stops wavesets instantiation
Button(win, Rect( 730, 220, 100, 20))
.states_([["Stopped"], ["Playing" ]])
.action_{
arg but;
```

```
but.value.postln;
if (but.value == 0, {Tdef(\waveset).stop}, { Tdef(\waveset).play } );
};

//9 - load a new sound and run waveset analysis
Button(win, Rect( 730, 250, 100, 20))
.states_(
[["Switch File", Color.black]]
)
.action_({|obj|
Dialog.openPanel({ | path |
var ws = WavesetsEvent.new;
ws.readChannel(path, onComplete: { ~wavesets = ws; ~path = path })
});
});
```

Code 27 - Custom buttons

III. Conclusion

The emergence of the FluCoMa project has been a revolution in computer music focused on machine listening and machine learning techniques with sound collections.

Its entire ecosystem, based on technological tools and an exhaustive and tiered system of educational resources by levels, has facilitated, approached, and made these techniques attractive to musicians and researchers of all levels.

Although I had tried these techniques in various environments with good artistic results, it was not until the appearance of FluCoMa that I felt a native integration that would allow me to think about artistic projects in an integrative way within my creative code environment of choice, SuperCollider.

On the other hand, these techniques have made possible another point of view different from my other great interest during my research: Wavesets synthesis. Through the detailed study of its algorithms, it has been possible not only to understand its technical operation (closely related to FluCoMa's slice reproduction algorithms by index) but also to obtain truly unique and unexpected sound results.

Alberto de Campo states: "The possibilities of microsound as a resource for both sound material and structural ideas are nowhere near being exhausted."[104] This statement is likely to remain valid for several decades to come.

This paper has proposed three adaptations of educational codes proposed by the FluCoMa project, modified to enable the intersection with Wavesets synthesis. The challenges of this work have been: (1) in the case of the *2D Waveset Corpus Explorer* and *Waveset Corpus Concatenation* instruments, to find a way to technically integrate the analyses returned by the *Wavesets2* class and (2) to implement the Wavesets synthesis engine in the *Wavesets Regressor* instrument.

Although these adaptations involve a simple modification of the main algorithms, their implementation has been slightly more challenging than expected (due to the highly buffer-based native operation of FluCoMa objects).

---

[104] de Campo, “Microsound,” 500.

Regarding the instruments, the *2D Waveset Corpus Explorer* and *Wavesets Regressor* instruments have turned out to work efficiently. In contrast, the *Waveset Corpus Concatenation* instrument has turned out to have performance problems (probably due to the high and fast processing rate) on my computer.[105] On an artistic level, the *Wavesets Regressor* instrument provided the most interesting results because it exploits the potential of Wavesets synthesis in a more complete and performative way. The concatenation of wavesets, proposed by the *Waveset Corpus Concatenation* instrument, envisions possible adaptations of real-time analysis and resynthesis for performative uses in the future.

From this point on, both technical and creative questions arise: How to improve these instruments to use them in a creative context? What other intersections between FluCoMa and Wavesets synthesis are possible? What other utilities does FluCoMa offer that can be useful and attractive for my musical practice, apart from the intersection with Wavesets synthesis? How do other users use the FluCoMa toolkits, and how could these practices inspire new artistic research? What type of creative works do Wavesets synthesis, machine listening, and machine learning techniques enable?

There is no doubt that this exciting exploration has only just begun and that only through more research and real artistic practice will I be able to visualize the true potential of this fascinating technology.

[105] MacBook Pro 9.1, i7, 2.3 GHz, 16 ram

Bibliography

A.I. For Anyone. “machine listening.” A.I. For Anyone, accessed April 22, 2023. https://www.aiforanyone.org/glossary/machine-listening.

Brown, Sara. “Machine learning, explained.” MIT Sloan School of Management. Last modified April 21, 2021. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

Burns, Ed. “machine learning”. TechTarget Editorial. Last modified March, 2021. https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML.

Collins, Nick. “Extending SuperCollider.” composerprogrammer. Accessed March 17, 2023. https://composerprogrammer.com/teaching/supercollider/sctutorial/Technicalities/10%20Extending%20SuperCollider.html

Collins, Nick. “Machine Listening in SuperCollider.” In The SuperCollider Book, edited by Wilson, Scott, David Cottle, and NickCollins, 439-22. Cambridge, Massachusetts London, England: The MIT Press, 2011.

“Computer audition.” Wikipedia. Last modified April 5, 2023. https://en.wikipedia.org/wiki/Computer_audition.

“Concatenative synthesis.” Wikipedia. Last Modified November 5, 2022. https://en.wikipedia.org/wiki/Concatenative_synthesis.

de Campo, Alberto. “Microsound.” In The SuperCollider Book, edited by Wilson, Scott, David Cottle, and NickCollins, 463-41. Cambridge, Massachusetts London, England: The MIT Press, 2011.

Endric, Archer, “CDP 'Wavecycle' DISTORT Functions.” ensemble-software. Last updated October 30, 2021. http://www.ensemble-software.net/CDPDocs/html/cdistort.htm#DISTORTLIST.

“El modelo de redes neuronales.” IBM. Last modified August 17, 2021. https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model.

“Granular Synthesis.” Wikipedia. Last modified January 16, 2023. https://en.wikipedia.org/wiki/Granular_synthesis.

Green, Owen, Pierre Alexandre Tremblay, Gerard Roma. “Interdisciplinary Research as Musical Experimentation: A case study in musicianly approaches to sound corpora”. *Proceedings of the Electroacoustic Music Studies Network Conference 2018* (January 2019): 1-12. http://www.ems-network.org/spip.php?article471.

Hochherz, Olaf. “SPList, a Waveset synthesis library and its usage in the composition “draussen”.” *Proceedings of Linux Audio Conference 2008* (2008): 1-6. https://lac.linuxaudio.org/2008/download/papers/19.pdf

Kartik, Menon. “An Introduction to the Types Of Machine Learning.” Simplilearn Solutions. Last modified March 10, 2023. https://www.simplilearn.com/tutorials/machine-learning-tutorial/types-of-machine-learning#:~:text=There%20are%20primarily%20three%20types,machine%20learning%20one%20by%20one.

“k-d tree.” Wikipedia. Last Modified February 28, 2023. https://en.wikipedia.org/wiki/K-d_tree.

Kjedgaard, Mads. “Parallel processing in SuperCollider using SuperNova.” Mads Kjedgaard. Last modified February 10, 2022. https://madskjeldgaard.dk/posts/supernova-intro/

Koutsomichalis, Marinos. “Vector Graphics.” In *Mapping and Visualization with SuperCollider: Create interactive and responsive audio-visual applications with SuperCollider.* Birmingham: Packt Publishing Ltd, 2013.

Lerch, Alexander. “Audio Content Analysis.” Preprint, submitted on July 1, 2021. https://arxiv.org/abs/2101.00132.

“Loudness.” FluCoMa Learn, Accessed March 24, 2023. https://learn.flucoma.org/.

“Machine learning”. Wikipedia. Last Modified March 27, 2023. https://en.wikipedia.org/wiki/Machine_learning.

Martins, Fellipe Miranda. “Estudo exploratório de processos de transformação sonora a partir de Trevor Wishart: reinvenção e tradução para o ambiente SuperCollider.” Master 's thesis, Universidade Federal de Minas Gerais. 2020.

Mayer, Daniel. “Software.” Accessed March 11, 2023. https://www.daniel-mayer.at/software_en.htm.

McCartney, James. “Forewords.” In *The SuperCollider Book*, edited by Wilson, Scott, David Cottle, and NickCollins, IX-3. Cambridge, Massachusetts London, England: The MIT Press, 2011.

“MFCC.” FluCoMa Learn. Accessed March 24, 2023. https://learn.flucoma.org/reference/mfcc/.

“MLPRegressor.” FluCoMa Learn. Accessed March 25, 2023. https://learn.flucoma.org/reference/mlpregressor/.

Moore, Ted, James Bradbury, Pierre Alexandre Trembly, and Owen Green. *FluCoMa for Pedagogues.* Centre for Research into New Music (CeReNeM), 2022.

"Music information retrieval." Wikipedia. Last modified April 19, 2023. https://en.wikipedia.org/wiki/Music_information_retrieval.

Roads, Curtis. *The Computer Music Tutorial*. Edition unstated edition. Cambridge, Massachusetts: The MIT Press, 1996.

Roads, Curtis. *Microsound*. Cambridge, Massachusetts London, England: The MIT Press, 2001.

Seidl, Fabian. *Granularsynthese mit Wavesets für Live-Anwendungen.* Master's thesis, Technische Universität Berlin. 2016.

"SuperCollider." Wikipedia. Last modified February 20, 2023. https://en.wikipedia.org/wiki/SuperCollider.

"SuperCollider IDE." sccode. Accessed February 19, 2023. https://doc.sccode.org/Guides/SCIde.html.

TIBCO Software Inc. "What is a Neural Network ?" Tibco. Accessed March 19, 2023. https://www.tibco.com/reference-center/what-is-a-neural-network.

Tremblay, Pierre Alexandre, Gerard Roma, and Owen Green. "Digging it: Programmatic Data Mining as Musicking." *Proceedings of the International Computer Music Conference 2021* (July 2021): 295-6.

https://pure.hud.ac.uk/en/publications/digging-it-programmatic-data-mining-as-musicking.
Tremblay, Pierre Alexandre, Gerard Roma, Owen Green, and Alex Harker. "From Collections to Corpora: Exploring Sounds through Fluid Decomposition." *Proceedings of the International Computer Music Conference 2019* (September 2019): 223-6. https://pure.hud.ac.uk/en/publications/from-collections-to-corpora-exploring-sounds-through-fluid-decomp.

Valle, Andrea. *Introduction to SuperCollider*. Berlin: Logos Berlin; Translation edition, 2016.

"What is machine learning?" IBM. Accessed 22 March, 2023. https://www.ibm.com/topics/machine-learning.

Wishart, Trevor. *Audible Design: A Plain and Easy Introduction to Sound Composition*. York: Orpheus The Pantomime Ltd., 1994.

Wishart, Trevor. *On Sonic Art*. York: Routledge, 1996.

Appendix 1 - *2D Waveset Corpus Explorer* code

```
~source = Buffer.read(s, Platform.resourceDir +/+ "sounds/a11wlk01.wav");
~waveSet = Wavesets2.fromBuffer( ~source );
~xings = ~waveSet.xings;
~bufferIndex = Buffer.loadCollection(s, ~xings);
~bufferIndex.loadToFloatArray( action: { arg indexWs; indexWs.postln } );


FluidWaveform( ~source, ~bufferIndex);



(
~dataSetWS = FluidDataSet(s);
~bufferIndex.loadToFloatArray(action:{
arg wavesets;
var spec = Buffer(s);
var loudness = Buffer(s);
var stats = Buffer(s);
var stats2 = Buffer(s);
var point = Buffer(s);
wavesets.doAdjacentPairs {    //adjacent pair of frames analysis (Waveset
duration)
arg start, end, i;            //first value-second value-iteration
var num = end - start;        //waveset length in frames
//analyzes centroid information and store them in "spec" buffer
FluidBufSpectralShape.processBlocking(s,~source,start,num,features:spec,select:[\
centroid]);
//get centroid statistical mean value and store it in "stats" buffer
FluidBufStats.processBlocking(s,spec,stats:stats,select:[\mean]);
//analyzes loudness information and store them in "loudness" buffer
FluidBufLoudness.processBlocking(s,~source,start,num,features:loudness,select:[\l
oudness]);
//get loudness statistical mead value and store it "stats2" buffer
FluidBufStats.processBlocking(s,loudness,stats:stats2,select:[\mean]);
//store centroid statistical mean values in "Point" buffer
FluidBufCompose.processBlocking(s,stats,destination:point,destStartFrame:0);

//store statistical loudness mean values in "Point" buffer
FluidBufCompose.processBlocking(s,stats2,destination:point,destStartFrame:1);
//load "point" buffer data into a database with unique id
~dataSetWS.addPoint(i,point);
"waveset % / %".format(i,wavesets.size).postln;
if((i%100) == 99){s.sync};
};
s.sync;
~dataSetWS.print;   //print dataset content
});
```

```
)


(
~normedDataSetWS = FluidDataSet(s);
FluidNormalize(s).fitTransform( ~dataSetWS, ~normedDataSetWS);
)


~normedDataSetWS.print;
~tree = FluidKDTree(s).fit(~normedDataSetWS);


(
~wavesetPlayer = {
arg index;            //input ~wavesetPlayer.(nearest.asInteger) value
{
var startsamp = Index.kr( ~bufferIndex, index);
var stopsamp = Index.kr( ~bufferIndex, index+1);
var phs = Phasor.ar( 0, BufRateScale.ir( ~source), startsamp, stopsamp);
var sig = BufRd.ar( 1, ~source, phs);
var dursecs = (stopsamp - startsamp) / BufSampleRate.ir( ~source);
var  env  =  EnvGen.kr(  Env(  [  0,  1,  1,  0], [  0.03,  dursecs  -  0.06,  0.03] ),
doneAction: 2);
sig.dup * env
}.play;
};
)


(
~normedDataSetWS.dump({   //get the information of the "~normedDataSetWS" back to the
lenguaje
arg dict;
var point = Buffer.alloc( s, 2);     //mouse position buffer for KDTree
var previous = nil;
dict.postln;
defer{
FluidPlotter(
dict: dict,                     //load Dataset information into the Plotter
mouseMoveAction:{
arg view, x, y;
[x,y].postln;                   //print mouse position values
point.setn(0,[x,y]);            //store mouse position values in the "point" buffer
~tree.kNearest(point,1,{        //search for nearest neighbor points at the mouse position
arg nearest;
if(nearest != previous){        //if the chosen point is not equal to the previous point
then:
nearest.postln;              //1 - post nearest current values (index, centroid and
loudness)
view.highlight_(nearest);    //2 - highlight the selected point
~wavesetPlayer.(nearest.asInteger);    // 3 - instance the player with the selected index
previous = nearest;           //4 - set current point as previous for a new comparison
```

```
}
});
});
};
});
)
```

Appendix 2 - *Waveset Corpus Concatenation* code

```
(
~source_buf = Buffer.read(s,
"/Users/carlosgonzalezbolanos/Desktop/audiosParaPruebas/_quickSounds/voz_2_mono.w
av" );
~target_buf = Buffer.read(s,
"/Users/carlosgonzalezbolanos/Desktop/audiosParaPruebas/_quickSounds/voz_1_mono.w
av" );
);

(
~waveset_source = Wavesets2.fromBuffer( ~source_buf);
~waveset_target = Wavesets2.fromBuffer( ~target_buf);
);

(
~xings_source = ~waveset_source.xings;
~xings_target = ~waveset_target.xings;
);

(
~source_index_buf = Buffer.loadCollection(s, ~xings_source);
~target_index_buf = Buffer.loadCollection(s, ~xings_target);
);



(
~analyze_to_dataset = {
arg audio_buffer, wavesets_buffer, action; //the audio buffer to analyze, a
buffer with //the slice points, and an action to execute when done
var features_buf = Buffer(s); //a buffer for writing the MFCC analyses into
var stats_buf = Buffer(s);    //a buffer for writing the statistical summary of
the MFCC //analyses into
var flat_buf = Buffer(s); // a buffer for writing only he mean MFCC values into
var dataset = FluidDataSet(s); // the dataset that all of these analyses will be
stored in
~nmfccs = 13;
//get the indices from the server loaded down to the language
wavesets_buffer.loadToFloatArray(action:{ arg wavesets_array;
fork{
//iterate over each index in this array, paired with this next //neighbor so that
we know //where to start and stop the analysis
wavesets_array.doAdjacentPairs{
arg start_frame, end_frame, waveset_index;
var num_frames = end_frame - start_frame;
"analyzing waveset: % / %".format(waveset_index + 1,wavesets_array.size -
```

```
1).postln;
//mfcc analysis, hop over that 0th coefficient because it relates to loudness and
here we //want to focus on timbre
FluidBufMFCC.processBlocking(s,audio_buffer,start_frame,num_frames,features:featu
res_buf,startCoeff:1,numCoeffs: ~nmfccs);
//get a statistical summary of the MFCC analysis for this waveset
FluidBufStats.processBlocking(s,features_buf,stats:stats_buf,select:[\mean]);
//extract and flatten just the 0th frame (numFrames:1) of the statistical summary
(because //that is the mean)
FluidBufFlatten.processBlocking(s,stats_buf,destination:flat_buf);
//now that the means are extracted and flattened, we can add this datapoint to
the //dataset:
dataset.addPoint("waveset-%".format(waveset_index),flat_buf);
if((waveset_index % 100) == 99){s.sync};
};
s.sync;
action.value(dataset);        // execute the function and pass in the dataset
that was created!
};
});
};
);


(
~analyze_to_dataset.( ~source_buf, ~source_index_buf,{
arg ds;
~source_dataset = ds;
~source_dataset.print;
});
);

(
~analyze_to_dataset.(~target_buf,~target_index_buf,{
arg ds;
~target_dataset = ds;
~target_dataset.print;
});
);



~scaled_dataset = FluidDataSet(s);      //a dataset for scaled values
~scaler = FluidNormalize(s);               //a function to normalize values
~scaler.fitTransform(~source_dataset, ~scaled_dataset);

~kdtree = FluidKDTree(s);               //create a KDTree
~kdtree.fit(~scaled_dataset,{"kdtree fit".postln;});
```

```
(
~wavesetPlayerConca = {
arg index, src_dur;
{
//lookup the start frame with the index *one the server* using Index.kr
var start_frame = Index.kr(~source_index_buf,index);
//same for the end frame
var end_frame = Index.kr(~source_index_buf,index+1);
var num_frames = end_frame - start_frame;
var dur_secs = min(num_frames / SampleRate.ir(~source_buf),src_dur);
var sig =
PlayBuf.ar(1,~source_buf,BufRateScale.ir(~source_buf),0,start_frame,0,2);
var env = EnvGen.kr(Env([0,1,1,0],[0.03,dur_secs-0.06,0.03]),doneAction:2);
//include this env if you like, but keep the line above because it will free the
//synth after the waveset
sig = sig * env;
OffsetOut.ar(0, sig.dup);
}.play;
};
)



(
fork{
var query_buf = Buffer.alloc(s,~nmfccs);   //a buffer for doing the neighbor
lookup with
var scaled_buf = Buffer.alloc(s,~nmfccs);
~target_index_buf.loadToFloatArray(action:{
arg target_index_array;
//prepend 0 (the start  of the file) to the indices array
target_index_array = [0] ++ target_index_array;
//append the total number of frames to know how long to play the last slice for
target_index_array = target_index_array ++ [~target_buf.numFrames];
inf.do{
arg i;
//get the index to play by modulo one less than the number of slices

var index = i % (target_index_array.size - 1);
// nb. that the minus one is so that the waveset from the beginning of the file
//to the //first //index is call "-1"
// this is because that waveset didn't actually get analyzed
var waveset_id = index - 1;
var start_frame = target_index_array[index];
var dur_frames = target_index_array[index + 1] - start_frame;
// this will be used to space out the source wavesets according to the target
timings
var dur_secs = dur_frames / ~target_buf.sampleRate;
"target waveset: %".format(waveset_id).postln;
// as long as this slice is not the one that starts at the beginning of the file
```

```
//(-1) and not the slice at the end of the file (because neither of these have
//analyses), let's do the lookup
if((waveset_id >= 0) && (waveset_id < (target_index_array.size - 3)),{
//1 - use the waveset id to (re)create the waveset identifier and load the data
point into //"query_buf"
~target_dataset.getPoint("waveset-%".format(waveset_id.asInteger), query_buf,{
//2 - once it's loaded, scale it using the scaler
~scaler.transformPoint( query_buf, scaled_buf,{
//3 - once it's neighbour data point in the kdtree of source waveset
~kdtree.kNearest( scaled_buf ,1,{
arg nearest;
//4 - peel off just the integer part of the waveset to use in the
~wavesetPlayerConca //instrument
var nearest_index = nearest.asString.split($-)[1].asInteger;
nearest_index.postln;
~wavesetPlayerConca.(nearest_index,dur_secs);
});
});
});
});
//dur_secs.wait;    //not optimized
0.1.wait;
};
});
};
)
```

Appendix 3 - *Wavesets Regressor* code

```
(
Window.closeAll;
s.waitForBoot{
var synth, multislider, win, xyslider;

var xybuf = Buffer.alloc(s, 2);      //2 frames, XY values
var paramsbuf = Buffer.alloc(s, 7);  //7 frames, for each synthesis parameter

var xydata = FluidDataSet(s);              //dataset for XY values
var paramsdata = FluidDataSet(s);          //dataset for synthesis parameter

var counter = 0;              //counter for point identifiers in datasets

var predicting = false;

var initPlay = 0;

var shufMax = 50;



//Neural network
var mlp = FluidMLPRegressor(s,
[7],
activation:FluidMLPRegressor.sigmoid,
outputActivation:FluidMLPRegressor.sigmoid,
maxIter: 1000,
learnRate:0.1,
batchSize:1,
validation:0
);

win = Window("WavesetRegressor", Rect(10, 10, 840, 335)).front;



//Multislider for synthesis parameter
multislider = MultiSliderView(win, Rect(10, 10, 400, 300))
.elasticMode_(1)
.isFilled_(1)
.action_{
arg ms;                         //ms - multislider values
paramsbuf.setn(0, ms.value);    //store multislider values in buffer "parambuf"
ms.value.postln;

//association of multislider values to synthesis parameters
~numWs = ms.value[0];
~repeats = ms.value[1];
~playRate = ms.value[2];
~playRate2 = ms.value[3];
~pan = ms.value[4];
~shuf = ms.value[5];
~legato = ms.value[6];
```

```
}.value_(0.25.dup(7));
xyslider = Slider2D(win,Rect(420,10,300,300))
.action_{
arg view;                                   //the Slider2D itself

xybuf.setn(0, [view.x, view.y]);            //store XY values in buffer "xybuf"

if( predicting) {                           //if true, then:

//"make a prediction" given the following input and output values
mlp.predictPoint( xybuf, paramsbuf, {

//assigns predicted values to parameters
paramsbuf.getn(0, 7, {
arg prediction;
prediction.postln;
~numWs = prediction[0];
~repeats = prediction[1];
~playRate = prediction[2];
~playRate2 = prediction[3];
~pan = prediction[4];
~shuf = prediction[5];
~legato = prediction[6];

//update values in multislider
defer{
multislider.value = prediction;
};
});
});
};
};




//1-add point to Dataset
Button(win, Rect( 730, 10, 100, 20))
.states_([["Add Points"]])
.action_{
var id = "point-%".format(counter);

//add datapoint to Dataset with unique ID
xydata.addPoint( id, xybuf);
paramsdata.addPoint( id, paramsbuf);

counter = counter + 1;

xydata.print;
paramsdata.print;
};


//2-save Dataset as json file
Button(win, Rect( 730, 40, 100, 20))
.states_([["Save Data"]])
```

```
.action_{
xydata.write("/Users/carlosgonzalezbolanos/Desktop/xydata.json");
xydata.write("/Users/carlosgonzalezbolanos/Desktop/paramsdata.json");
};


//3-load a Dasaset from a json file
Button(win, Rect( 730, 70, 100, 20))
.states_([["Load Data"]])
.action_{
xydata.read("/Users/carlosgonzalezbolanos/Desktop/xydata.json");
xydata.read("/Users/carlosgonzalezbolanos/Desktop/paramsdata.json");
};


//4-train the neural network
Button(win, Rect( 730, 100, 100, 20))
.states_([["Train"]])
.action_{

mlp.fit( xydata, paramsdata, {    //input data and output data example pairs
arg loss;
loss.postln;

});
};


//5-saves the neural network as a json file
Button(win, Rect( 730, 130, 100, 20))
.states_([["Save MLP"]])
.action_{
mlp.write( "/Users/carlosgonzalezbolanos/Desktop/mlp.json");
"mlp saved".postln;
};


//6-load a neural network from a json file
Button(win, Rect( 730, 160, 100, 20))
.states_([["Load MLP"]])
.action_{
mlp.read( "/Users/carlosgonzalezbolanos/Desktop/mlp.json");
"mlp loaded".postln;
};


//7-boolean, allows the neural network to predict
Button(win, Rect( 730, 190, 100, 20))
.states_([["Not Predicting"], ["Predicting" ]])
.action_{
arg but;
predicting = but.value.asBoolean;
};


//8-plays or stops wavesets instantiation
Button(win, Rect( 730, 220, 100, 20))
```

```
.states_([["Stopped"], ["Playing" ]])
.action_{
arg but;
but.value.postln;
if (but.value == 0, {Tdef(\waveset).stop}, { Tdef(\waveset).play } );
};


//9-load a new sound and run waveset analysis
Button(win, Rect( 730, 250, 100, 20))
.states_(
[["Switch File", Color.black]]
)
.action_({|obj|
Dialog.openPanel({ | path |
var ws = WavesetsEvent.new;
ws.readChannel(path, onComplete: { ~wavesets = ws; ~path = path })
});
});

s.sync;



~wavesets = WavesetsEvent.read(Platform.resourceDir +/+
"sounds/a11wlk01-44_1.aiff");



WavesetsEvent.prepareSynthDefs;



s.sync;



Tdef(\waveset, {
inf.do{ |i|
var event = ~wavesets.asEvent((

//from which waveset to start + waveset shuffle
start: (( i % (~wavesets.size - shufMax)) + ~shuf.linlin(0,1,0,shufMax) ).round,
num: ~numWs.linlin(0, 1, 1, 50).round,           //how many wavesets to play

//how many times to repeat the //selected wavesets
repeats: ~repeats.linlin(0, 1, 1, 20).round,

rate: ~playRate.linlin(0, 1, 0.2, 10),     //playback speed of the audio file

//end playback speed of the audio file //(will create a linear glisson sweep)
rate2: ~playRate2.linlin(0, 1, 0.2, 10),

pan: 0 + ~pan.rand.round(0.1) * [1.0, -1.0].choose,

//scales the duration, so that //wavesets will overlap or have gaps between them.
legato: ~legato.linlin(0, 1, 0.1, 2),
```

```
//scale the amplitude of the original //sound + random waveset omission from 0 -
100%
//amp: if(~prob.coin, {~volume}, {0}),
));
event.play;
event[\dur].wait;
}
});
};
)
```

"Hiermit erkläre ich eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe. Alle Stellen oder Passagen der vorliegenden Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für die Reproduktion von Noten, grafischen Darstellungen und anderen analogen oder digitalen Materialien.

Ich räume der Anton Bruckner Privatuniversität das Recht ein, ein von mir verfasstes Abstract meiner Arbeit sowie den Volltext auf der Homepage der ABPU zur Einsichtnahme zur Verfügung zu stellen."

_____

Carlos Miguel González Bolaños

30.07.2023