

Sound Synthesis Parameters: Matching and Interpolation using Generative Neural Networks

Gwendal Le Vaillant

`gwendal.levaillant@umons.ac.be`

Friday 7th February, 2025

A dissertation submitted to the Faculty of Engineering
of the University of Mons, for the degree of Doctor of Philosophy in Engineering Science

Supervisor: Prof. Thierry Dutoit

This thesis was also supported by HE2B (Haute-École Bruxelles-Brabant), ISIB (Institut Supérieur Industriel de Bruxelles) and IRISIB (Institut de Recherche de l'ISIB).

Jury members

Prof. **Bernard Gosselin** - Université de Mons, Chairperson

Prof. **Thierry Dutoit** - Université de Mons, Supervisor

Prof. **Laurent Girin** - Grenoble-INP / GIPSA-lab

Prof. **Geoffroy Peeters** - Télécom Paris

Prof. **Xavier Siebert** - Université de Mons, Secretary

Ir. **Rudi Giot** - Institut Supérieur Industriel de Bruxelles

Abstract

THIS thesis addresses key challenges in manipulating sound synthesizers, which have shaped modern music production and have defined new music genres. Specifically, it focuses on the tasks of sound matching and sound morphing using conventional synthesizers.

Given the complexity of manipulating extensive parameter sets, called presets, the study introduces a scalable, synthesizer-agnostic approach to sound matching for creating presets from target sounds. The model is validated using a widespread frequency modulation synthesizer with a large set of interdependent parameters and a semi-modular architecture.

This thesis also introduces a novel variational auto-encoder model tailored for black-box synthesizer preset interpolation, which enables the intuitive generation of new presets from pre-existing ones. Leveraging multi-head self-attention networks, the model learns latent representations of synthesis parameters, aligning these with perceived timbre dimensions through attribute-based regularization. Objective evaluations of timbre linearity and smoothness show that the auto-encoder is able to gradually transition between diverse presets. The model’s performance is also demonstrated through subjective experiments. It can be adapted to any conventional non-differentiable synthesizer, and can perform other tasks such as extrapolations and automatic modulations of presets. These contributions advance sound design by offering automated, generative tools for sound matching and preset interpolation.

Acknowledgements



FIRST and foremost, I am grateful to my partner in life, Sílvia, who supported me in countless ways throughout these years. Without her, neither this thesis nor the related publications would have been possible. I also thank my brother, Nikolaz, for showing me the way to completing a PhD, and of course our parents for their support since the very beginning. I also think about my friends in Brussels, in Brittany and in Paris, who came to view my phrase, “Well, I should complete my PhD *soon*,” as a running joke spanning multiple years, but supported me anyway. I also thank my colleagues at ISIB and other institutions in Belgium, especially Rudi, Jean-Luc and Damien, for their advice and fruitful discussions.



I Am thankful to Prof. Thierry Dutoit for supervising my work over these years and for being an approachable director of ISIA-lab. I also would like to thank the members of the accompanying committee, Prof. Bernard Gosselin and Prof. Xavier Siebert, for their feedback and guidance. Special thanks to Rudi Giot for his insightful ideas, in particular during the early stages of this thesis, and for recommending me for a PhD program. Last but not least, I also thank Prof. Laurent Girin and Prof. Geoffroy Peeters for their interest in this work and for accepting to review it and be part of the jury.



Contents

Introduction	3
1 Sound synthesizers	7
1.1 Timbre, pitch and loudness	8
1.2 Conventional sound synthesizers	11
1.2.1 Additive and subtractive synthesis	12
1.2.2 Envelopes and modulations	15
1.2.3 Frequency modulation synthesis	17
1.2.4 Other methods	21
1.3 Presets and control interfaces	25
1.3.1 Classic user interfaces	25
1.3.2 Preset manipulation	26
1.3.3 FM synthesis presets	29
1.4 Differentiable synthesizers	30
1.4.1 Neural waveform synthesis	30
1.4.2 Neural spectral synthesis	34
1.4.3 Differentiable digital signal processing	35
1.4.4 Limitations	39
1.5 In brief	42
2 Automatic synthesizer programming	43
2.1 State of the art	45
2.1.1 Task definition	45
2.1.2 Heuristic methods	46

2.1.3	Data-driven techniques	48
2.1.4	Related fields and commercial applications	55
2.2	Dataset of DX7 presets	56
2.2.1	Data collection and filtering	56
2.2.2	Automatic labeling	58
2.3	PresetGenVAE model	62
2.3.1	Variational Autoencoder	62
2.3.2	Normalizing Flows	69
2.3.3	General model architecture	73
2.3.4	Output preset modeling	76
2.3.5	Implementation and training	78
2.4	Results and model analysis	80
2.4.1	Preset representation and decoder	80
2.4.2	Multi-channel spectrograms	83
2.4.3	Discussion	88
2.5	In brief	90
3	Sound morphing	91
3.1	Literature overview	93
3.1.1	Analysis and resynthesis	93
3.1.2	Traditional signal processing techniques	94
3.1.3	Data-driven methods	98
3.2	Method for objective evaluation	99
3.2.1	Previous works	99
3.2.2	Timbre features extraction	101
3.2.3	Sound morphing metrics: linearity and smoothness	106
3.3	Comparison of morphing techniques	108
3.3.1	Experiment description	108
3.3.2	Results	112
3.3.3	Discussion	118
3.4	In brief	119

4	Latent space interpolation of presets	121
4.1	Non-linear preset interpolation	123
4.2	Transformer-VAE Model	125
4.2.1	VAE for synthesizer presets	126
4.2.2	Transformer encoder and decoder	127
4.2.3	Additional spectrogram decoder	133
4.2.4	Output probability distributions	135
4.2.5	Regularization using timbre attributes	137
4.2.6	Training procedure	139
4.3	Results	140
4.3.1	Latent interpolation	140
4.3.2	Objective morphing performance	141
4.3.3	Reconstruction of start and end sounds	145
4.3.4	Computational requirements	147
4.3.5	Discussion	147
4.4	Other practical applications	149
4.4.1	Blending multiple presets	149
4.4.2	Extrapolation	150
4.4.3	Preset modulation	151
4.4.4	Other conventional synthesizers	153
4.5	In brief	154
5	SPINVAE model analysis	155
5.1	Subjective evaluation	156
5.1.1	Experimental protocol	156
5.1.2	Task difficulty	158
5.1.3	Comparison results	159
5.1.4	Smoothness and naturalness subjective ratings	159
5.1.5	Objective correlates of subjective ratings	161
5.2	Ablation study	163
5.2.1	Experiments	163

5.2.2	Auto-encoder architecture	165
5.2.3	Latent properties	167
5.2.4	Output probability distributions	169
5.2.5	Data augmentation strategies	171
5.3	Preset decoder analysis	172
5.3.1	Transformer attention maps	173
5.3.2	Output parameter trajectories	175
5.4	In brief	178
Conclusion		179
A Publications related to this thesis		183
A.1	Journal article	183
A.2	Papers in international conferences with peer review	183
A.2.1	Oral presentations	183
A.2.2	Poster presentations	184
B FM synthesis parameters		185
B.1	Yamaha DX7	185
B.2	Ableton Operator	187
C Preset creation using a conversational agent		189
C.1	Creation of a clarinet preset	189
C.2	Minimoog preset	190
C.3	DX7 FM preset	192
D Histograms of Dexed parameter values		195
E Correlations between timbre descriptors		199
F Detailed morphing linearity and smoothness results		203
G Decoded trajectories of synthesis parameters		211

Bibliography	219
List of Figures	249
List of Tables	255

List of acronyms

ACTM	AudioCommons Timbral Models	102
ADSR	Attack, Decay, Sustain, Release	15
AM	Amplitude Modulation	186
AR	Attributed-based Regularization	137
BN	Batch Normalization	75
CNN	Convolutional Neural Network	32
CPU	Central Processing Unit	33
CQT	Constant-Q Transform	34
DAW	Digital Audio Workstation	11
DLM	Discretized Logistic Mixture	135
DSP	Digital Signal Processing	30
DDSP	Differentiable Digital Signal Processing	35
EG	Envelope Generator	173
F_0	Fundamental Frequency	10
FFT	Fast Fourier Transform	35
FIR	Finite Impulse Response	35
FM	Frequency Modulation	5
GAN	Generative Adversarial Network	33
GPT	Generative Pre-trained Transformer	189

GPU	Graphics Processing Unit	32
GRU	Gated Recurrent Unit	51
HCI	Human-Computer Interaction	26
HpN	Harmonic plus Noise	35
HPSS	Harmonic-Percussive Source Separation	58
IIR	Infinite Impulse Response	40
IQR	Interquartile Range	102
LAT	Log Attack Time	102
LFO	Low Frequency Oscillator	16
LLM	Large Language Model	28
LSTM	Long Short-Term Memory	50
MAE	Mean Absolute Error	33
MAF	Masked Auto-regressive Flow	72
MDS	Multidimensional Scaling	9
MFCC	Mel-Frequency Cepstral Coefficient	36
MFCCD	MFCC Distance	50
MIDI	Musical Instrument Digital Interface	10
MIR	Musical Information Retrieval	101
MLP	Multi-Layer Perceptron	50
MOS	Mean Opinion Score	157
MSE	Mean Squared Error	31
MSS	Multi-Scale Spectral	35
MUSHRA	Multiple Stimuli with Hidden Reference and Anchor	39
NLL	Negative Log-Likelihood	32

NMF	Non-negative Matrix Factorization	97
NN	Neural Network	4
ODG	Objective Difference Grade.....	115
OSC	Open Sound Control	12
PCA	Principal Component Analysis	160
RealNVP	Real-valued Non-Volume Preserving flow	72
RMS	Root Mean Square	48
RNN	Recurrent Neural Network	50
SGD	Stochastic Gradient Descent.....	63
SMT	Sound Morphing Toolbox	95
SPINVAE	Synthesizer Preset Interpolation VAE	5
STFT	Short-Time Fourier Transform	34
std	standard deviation	10
TCN	Temporal Convolutional Network.....	32
TT	Timbre Toolbox.....	102
UI	User Interface.....	21
VAE	Variational Autoencoder	5
VST	Virtual Studio Technology	11

Introduction

Sound synthesizers can produce audio signals with timbres that vary from those of acoustic instruments to entirely unique sound textures. They have become a staple in contemporary music production and have even shaped new music genres. In the 1960s, the Moog modular synthesizer marked the first widely popularized musical instrument invention since Adolphe Sax’s creation of the saxophone in 1846 [1]. While synthesizers became fundamental to the electronic sound of the 1980s, their use in genres like Funk and early electronic music dates back to the 1970s.

The *Classic Synthesizers Behind 10 Iconic Tracks* from 1978 to 2020 is an on-line resource [2] for listening to the most famous synthesizers in pop culture. Today, they remain the primary instruments of dominant music genres such as Pop, Rap/Hip-hop and Electronic Dance Music. As of January 2024, synthesizers are easily recognizable in approximately 80% of the Billboard Global 200 songs¹. The music market itself is huge and keeps growing. For instance, the music recording industry alone has reported a 26.5 billion euros global revenue in 2023 [3], increasing by 10.2% since 2022. Synthesizers are also intensively used in movie, television and video games soundtracks and sound effects [4], by artistic choice or to reduce production budgets. They are used in research as well, e.g., in order to generate controllable and reproducible stimuli for psycho-acoustical experiments [5].

Sound synthesis is an essential tool for many practitioners, but creating a desired target sound often requires a tedious manipulation of numerous parameters [6]. The sets of parameters controlling the synthesis processes, called *presets*, are often extensive and complex. Hence, synthesizer manufacturers have to offer a wide range of presets, providing users with starting points to explore the synthesis sound space [7]. Many users rely on presets only and

¹The 200 most popular songs based on worldwide streaming and/or sales activity. <https://www.billboard.com/charts/billboard-global-200/>

do not investigate the full sonic space [8]. They tend to search extensive databases or libraries of sounds that fit specific needs, and that search is not always successful [9]. There is also a dedicated market for additional preset libraries [10].

Creating presets tailored for individual requirements is an interesting challenge. To perform this task, a class of techniques focus on searching for the set of parameters that allows for the best reproduction of a pre-recorded target sound. This field is referred to as automatic synthesizer programming, or sound matching. In particular, this thesis explores the use of generative Neural Networks (NNs) for the task of sound matching. These models learn to produce new data by identifying patterns in existing datasets. They can be used to create new presets by generating parameter configurations that mimic the characteristics of existing sounds while introducing novel variations. This allows for the easy and automated creation of unique presets in various ways.

Another field of research for creating and designing new sounds is sound morphing, which blends characteristics of two input waveforms to create new, intermediate timbres. Unlike simple mixing, morphing aims at producing intermediate sounds that are perceived as a hybridization of the two input waveforms, rather than a simple weighted superposition. Morphing can also be used for real-time sound transformation, provided the technique is fast enough, or in psycho-acoustical experiments [11]. Traditional morphing relies on analysis of the input waveforms, and on resynthesis using dedicated techniques [12].

The latter part of this thesis focuses on preset morphing for an arbitrary synthesizer seen as a black box, combining generative NNs with non-differentiable conventional synthesizers. It also explores objective metrics for evaluating sequences of morphed samples, such as their *linearity* or *smoothness*. In particular, generative models can be trained to encode presets into a latent space where linear interpolations can be computed. Intermediate latent embeddings can be decoded into presets, for creating new presets to be added into an existing library, or to perform real-time preset interpolations. This approach also enables other practical applications such as extrapolations or dynamic modulation of sounds.

Original contributions of this thesis

The main contributions of this thesis are the following:

1. The PresetGenVAE model, based on a Variational Autoencoder (VAE) [13], which was state-of-the-art for automatic Frequency Modulation (FM) synthesizer programming at the time of its publication;
2. An automatic, reproducible, and explainable method to compute objective evaluations of any sound morphing;
3. A review of sound morphing techniques, with a large-scale objective study that enables the identification of the current state-of-the-art from the literature;
4. The Synthesizer Preset Interpolation VAE (SPINVAE) model, which combines preset and audio modalities into a latent space and enables interpolations of presets for a black-box synthesizer;
5. The first successful and documented use of neural networks based on multi-head attention mechanisms [14] for modeling synthesizer presets.

Organization of this dissertation

Chapter 1 introduces conventional synthesizers and the underlying synthesis techniques, as well as differentiable synthesizers. It also details the challenges involved in managing and adjusting synthesizer settings.

Chapter 2 focuses on automatic synthesizer programming, with a particular emphasis on the challenges of sound matching for FM synthesis. It introduces a dataset and a multimodal generative neural network developed for this purpose.

Chapter 3 explores sound morphing and emphasizes its ability to create intermediate hybrid timbres. It reviews existing morphing techniques, proposes an objective evaluation method, and presents a quantified comparison of state-of-the-art approaches.

Chapter 4 introduces SPINVAE, a synthesizer-agnostic approach for latent interpolation of presets. It discusses the model's architecture, evaluates its interpolation performance using objective metrics, and explores

other practical applications such as extrapolation and dynamic preset modulation.

Chapter 5 further evaluates the performance of SPINVAE in preset interpolation through subjective and objective experiments. It also includes an empirical analysis of the preset decoder’s handling of sequences of synthesis parameters.

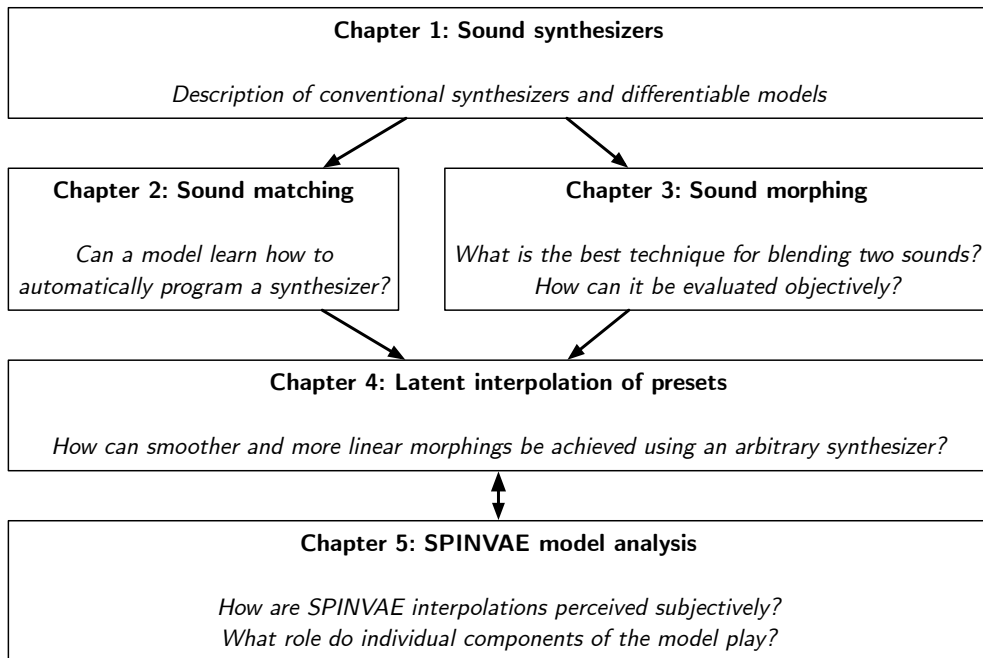


Figure A. Organization of the chapters in this dissertation.

Chapter 1

Sound synthesizers

Contents

1.1	Timbre, pitch and loudness	8
1.2	Conventional sound synthesizers	11
1.2.1	Additive and subtractive synthesis	12
1.2.2	Envelopes and modulations	15
1.2.3	Frequency modulation synthesis	17
1.2.4	Other methods	21
1.3	Presets and control interfaces	25
1.3.1	Classic user interfaces	25
1.3.2	Preset manipulation	26
1.3.3	FM synthesis presets	29
1.4	Differentiable synthesizers	30
1.4.1	Neural waveform synthesis	30
1.4.2	Neural spectral synthesis	34
1.4.3	Differentiable digital signal processing	35
1.4.4	Limitations	39
1.5	In brief	42

This chapter begins by exploring the realm of classic synthesizers, which have shaped the sound of music for decades. It delves into the main synthesis techniques, including additive, subtractive, and FM synthesis, each contributing to the diverse sonic capabilities available to musicians and sound designers. The concept of timbre is also briefly introduced. Classically defined as the quality of a sound that makes it distinct from other sounds at the same pitch and loudness, timbre is a fundamental aspect of auditory perception. It is crucial for distinguishing the unique character of sounds produced by different synthesizers.

Then, the challenges of managing and manipulating presets in synthesizers are discussed, highlighting the complexities involved in tailoring sounds to specific creative needs. In particular, FM synthesis presets are notoriously challenging to handle, and will be the main case study during the next chapters of this thesis.

The emerging field of differentiable synthesizers is finally introduced. These advanced models, developed from recent neural networks architectures, are able to learn and model sound characteristics directly from data. They offer new capabilities, such as timbre transfer. However, they also present some limitations, such as audio quality and computational demands, and are yet to be widely adopted in the industry.

1.1 Timbre, pitch and loudness

In order to describe and compare sound synthesis techniques, the notion of timbre must be introduced first. It is formally defined by the Acoustical Society of America [15] as:

“Timbre: multidimensional attribute of auditory sensation which enables a listener to judge that two non-identical sounds, similarly presented and having the same loudness, pitch, spatial location, and duration, are dissimilar. Timbre is related to sound quality, often specified by qualitative adjectives (e.g., bright or dull).

The timbre of a sound is strongly influenced by its time-varying characteristics, particularly during the initial portion (attack), and is also influenced by its ongoing spectral and temporal character-

istics. Timbre is an essential element in the identification of the source of a sound (e.g., particular musical instruments) and the manner of sound production.”

The qualities (or adjectives) of timbre are not specified in this definition, which has been criticized [16]. There is no clear consensus regarding an extensive list of orthogonal dimensions of timbre [17], or whether such orthogonal dimensions exist. Some timbre attributes may co-vary with pitch, loudness and even duration [18, 19], which goes against the aforementioned definition. For instance, one can consider that the timbre of a saxophone changes when it’s being played louder.

Nonetheless, several studies have tried to identify the most significant dimensions of timbre, based on subjective measurements of dissimilarities between pairs of samples. These samples were recordings of notes played using acoustic instruments or synthesizers, and they all had the same duration, pitch and loudness [20]. Perceptual dissimilarities, considered as distances between samples, were then provided to the Multidimensional Scaling (MDS) algorithm [21, 22] in order to map the samples into an N -dimensional space where original distances are preserved. N is typically small, e.g. $N = 2$ or $N = 3$, such that MDS is considered as a dimensionality reduction and visualization technique. However, the N dimensions do not hold intrinsic perceptual meaning. Therefore, an analysis is performed to identify the acoustic features (e.g. spectral, temporal) that best correlate to these dimensions. The N acoustic correlates can be interpreted as perceptually meaningful dimensions of timbre [17].

While results vary and depend on the experiment, some salient timbre features have been identified by several distinct works. The three best acoustic correlates of four studies [17, 18, 20, 23] are reported in Figure 1.1. It presents various spectral and/or temporal features which are properly defined in each corresponding study. Although the identified timbre dimensions are slightly different, they are generally well-correlated [17]. For instance, the attack, or rise time¹, has been highlighted in three out of four studies. Similar features such as spectral fluctuation, spectral flux and spectral variation can also be found across the cited works.

¹Usually defined as the time between a note’s onset and its peak level (see Subsection 1.2.2).

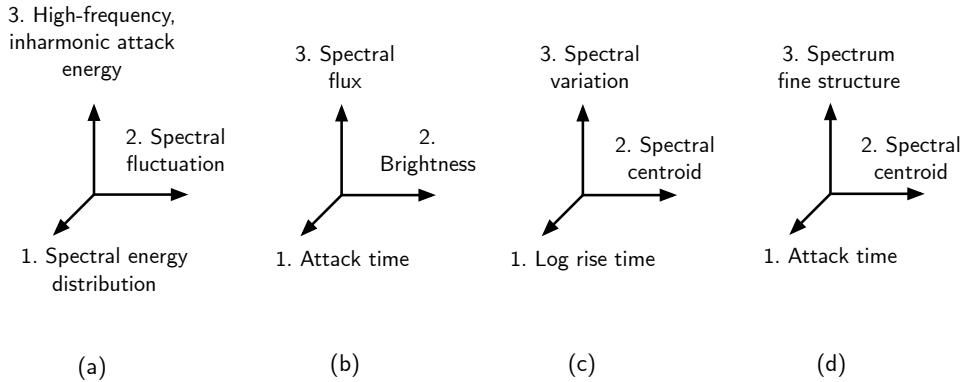


Figure 1.1. Acoustic correlates of MDS space axes, interpreted as salient dimensions of timbre, by four different studies. (a) Grey, 1975 [20]. (b) Krumhansl, 1989 [18]. (c) McAdams *et al.*, 1995 [17]. (d) Caclin *et al.*, 2005 [23].

Another work [24] has identified two main acoustic descriptors, one spectral and one temporal, while the third only accounts for a small amount of variance. Descriptors of spectral and temporal distributions, such as mean, standard deviation (*std*), skew and kurtosis, can also be used [25]. Recently, studies [25] have been describing timbre with semantic differential scales such as soft/hard, static/dynamic, small/big instrument, calm/explosive or dull/sharp. A more extensive discussion about timbre feature extractors is presented in Chapter 3.

In contrast to timbre, pitch and loudness are well-defined perceptual dimensions. Pitch is the auditory sensation which allows to order sounds from low to high on a melodic musical scale [26]. It is usually considered to have a monotonic and logarithmic relationship with Fundamental Frequency (F_0). E.g., the Musical Instrument Digital Interface (MIDI) tuning standard [27] expresses pitch as a function of frequency using a base 2 logarithm. This corresponds to the approximately logarithmic layout of frequencies in the human cochlea [28]. The human perception of sound pressure is also logarithmic [28]. Formally, loudness is the auditory sensation which allows to order samples on a scale from soft to loud [29], and it is mainly related to sound pressure. However, it is also dependent on frequency [30], e.g., a medium-frequency tone would be perceived louder than a high-frequency tone of the same amplitude. Loudness is dependent on duration as well, e.g., the human auditory system

has an acoustic reflex [28] that can attenuate sound transmission into the inner ear after a small delay.

1.2 Conventional sound synthesizers

Akin to [1], this section considers synthesizers that produce sound electronically, using any combination of analog circuitry, digital circuitry and/or software. This excludes electro-mechanical instruments such as the Hammond tonewheel organ, or acoustic instruments sounds processed through analog or digital audio effects. This section focuses on widespread synthesizers, most of them being commercial products which are extensively used by musicians, sound designers and producers.

Synthesizers come in various forms and configurations to cater to diverse artistic needs and production environments. From tactile hardware units that offer hands-on control to software synthesizers that provide convenience and portability, the range is extensive. Hardware synthesizers themselves come in different formats. Some provide integrated keyboards for immediate playability, such as the Yamaha DX7 in Figure 1.2(b) and Arturia Microfreak in Figure 1.2(d). Others, like the rack-mounted Minimoog Voyager in Figure 1.2(a), are compact units that can be played from an external keyboard. In the digital realm, synthesizers are available as standalone applications or as plugins, seamlessly integrating into Digital Audio Workstations (DAWs). For instance, the Diva software synthesizer shown in Figure 1.2(c) is provided as a Virtual Studio Technology (VST) plugin. The reader can refer to [1] to get an historical perspective on the development of synthesizers since the 1920s.

Controllers used to play notes into synthesizers are typically MIDI piano-like keyboards. Nonetheless, drum pads, string and wind instrument controllers are also commonly used [31]. Sequencers can also be used to autonomously play loops of notes. However, this subsection—and this thesis as a whole—will focus on sound synthesis processes and parameters rather than note controllers.

In practice, there are multiple ways to communicate notes to a synthesizer. Those which have an integrated keyboard may use internal proprietary protocols. For separate controllers and synthesizers, MIDI is ubiquitous but other



Figure 1.2. Four widespread commercial hardware and software synthesizers. (a) Minimoog Voyager: analog subtractive synthesizer. (b) Yamaha DX7: digital FM synthesizer. (c) U-he Diva: software modeling of famous analog synthesizers. (d) Arturia Microfreak: modern hybrid synthesizer with digital oscillators and analog filters.

protocols exist. In the digital realm, examples include Open Sound Control (OSC) [32] and the updated MIDI polyphonic expression [33] standard. Analog synthesizers often provide control voltage and gate inputs, also known as CV/gate [1]. Nonetheless, for the sake of simplicity and clarity of this thesis, any note being played into a synthesizer will be referred to as a MIDI note.

1.2.1 Additive and subtractive synthesis

Additive synthesis is a technique that constructs complex audio signals by summing individual sine waves [1]. Each sine wave has its own frequency, amplitude, and phase. When combined, they can create a wide range of sounds,

from simple tones to evolving textures. An example, based on the Fourier series decomposition of the periodic sawtooth signal, is given in Figure 1.3. In practice, additive synthesis allows for precise control over the harmonic structure of sounds. It is typically used to synthesize organ-like sounds, or to create atmospheric pads² and evolving sound textures. Pure additive synthesis can emulate acoustic instruments with rich non-harmonic spectra, but realistic results require a lot of analog oscillators or significant computing power for digital synthesis.

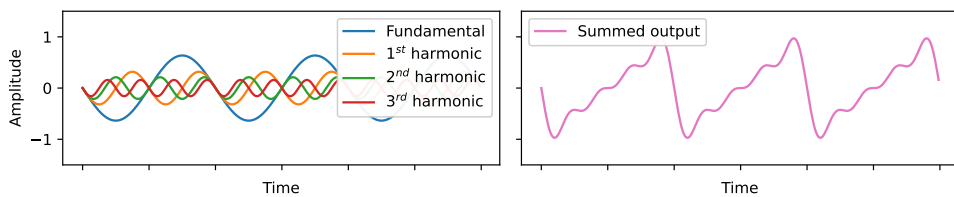


Figure 1.3. Additive synthesis example: sawtooth-like signal (right) obtained by summing signals from multiple sinusoidal oscillators (left).

Subtractive synthesis [1] relies on oscillators that generate complex waveforms with lots of harmonic content, and filters to attenuate or amplify specific harmonics. It is quite similar to source-filter modeling applied to human speech analysis and synthesis [34], although subtractive synthesizers typically provide basic sources such as square, sawtooth or triangle waveforms. As displayed in Figure 1.4, filters often feature resonance to give a characteristic color to the sound.

Subtractive synthesizers are known to be able to produce rich and deep bass sounds, e.g., when using sawtooth or square waves with a low cut-off frequency for the low-pass filter. Using resonant filters, they can also generate bright brass sounds or lead³ sounds that *cut through the mix* of a piece of music.

In practice, very few synthesizers are purely additive or subtractive. A notable example is the Minimoog from Figure 1.2(a) whose signal-routing architecture

²Pad refers to an ambient sustained synthesizer sound used to enrich the harmony of a piece of music.

³Sounds with sharp and distinct harmonic contents which belong to the musical foreground of a piece of music, i.e., they can be heard easily even if the music contains numerous instrumental layers.

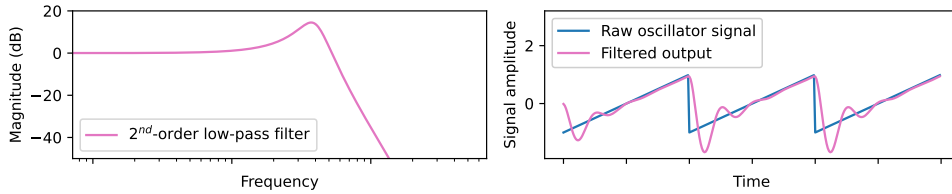


Figure 1.4. Subtractive synthesis example: sawtooth signal from an oscillator (right) processed through a low-pass filter (left).

has become a standard in the synthesizer industry [1]. It is considered subtractive although it contains a mixer that sums outputs from three oscillators and a noise source (Figure 1.5). Other components from this Figure 1.5, such as envelopes, will be detailed in the following subsection.

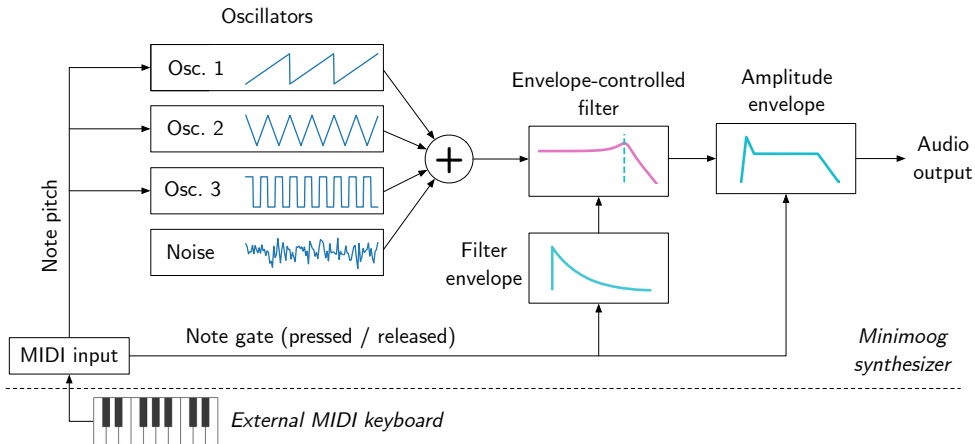


Figure 1.5. Simplified routing of signals inside the Minimoog synthesizer. Modulations are not represented.

The most common way to pilot synthesizers remains **MIDI** note messages, although the **MIDI** standard [27] dates from the 1980s. When a note is played, e.g. a key is pressed on a **MIDI** keyboard, the note's pitch and velocity⁴ are sent

⁴In the **MIDI** standard, note *velocity* designates the speed at which a key is pressed. A more appropriate and general term could be note *intensity*.

to the synthesizer. When the key is released, the same pitch and a null velocity are sent, which the synthesizer interprets as a “note off” message. Figure 1.5 shows that MIDI pitch controls oscillators’ parameters, in particular their F_0 .

1.2.2 Envelopes and modulations

Envelopes and modulation sources are fundamental components in the architecture of synthesizers, playing a crucial role in shaping the synthesized timbre. Envelopes are used to control various aspects of the sound over time, most commonly amplitude and filter cutoff frequency. The ubiquitous envelope form is named Attack, Decay, Sustain, Release (ADSR), and has been used since the early years of sound synthesis [35]. ADSR amplitude envelopes allow a musician to sculpt the temporal evolution of a note, from the initial onset to its eventual fadeout (Figure 1.6). When a note is played, Attack defines the time it takes for the sound to reach its maximum level, Decay sets the time to transition to the sustain level and Sustain specifies the level during the main sequence of the note. Release determines how long the sound takes to diminish after a note is released. Attack time, for example, has been identified multiple times as the most salient dimension of timbre (Section 1.1). Figure 1.6 displays a piecewise linear envelope, but many synthesizers also provide exponential and logarithmic curves. Moreover, synthesizers often use MIDI notes’ velocity to control the envelope itself, e.g. by amplifying the whole envelope. Therefore, velocity can influence the loudness of output sounds.

For instance, consider the modeling of a violin-like sound using a subtractive synthesizer, e.g. the one from Figure 1.5. String-and-bow interaction produces a movement that can be roughly approximated as a sawtooth [35]. Thus, the sawtooth waveform can be selected for Oscillator 1, e.g. by turning the appropriate knobs on the front panel displayed in Figure 1.2(a). The volume of other oscillators can be set to zero. Then, the amplitude ADSR knobs can be set to obtain an envelope similar to Figure 1.6. The resulting sound will somehow resemble a violin played using a bow, with a rather short attack time, a sustain mimicking the bow being continuously stroked along the string, and a release time corresponding to the string’s final oscillation with damping. If the sustain level was set to zero, the sound could be perceived as a pinched⁵ violin rather than a bowed one.

⁵*Pizzicato* playing technique, often called *plucked* strings in the synthesizers’ industry.

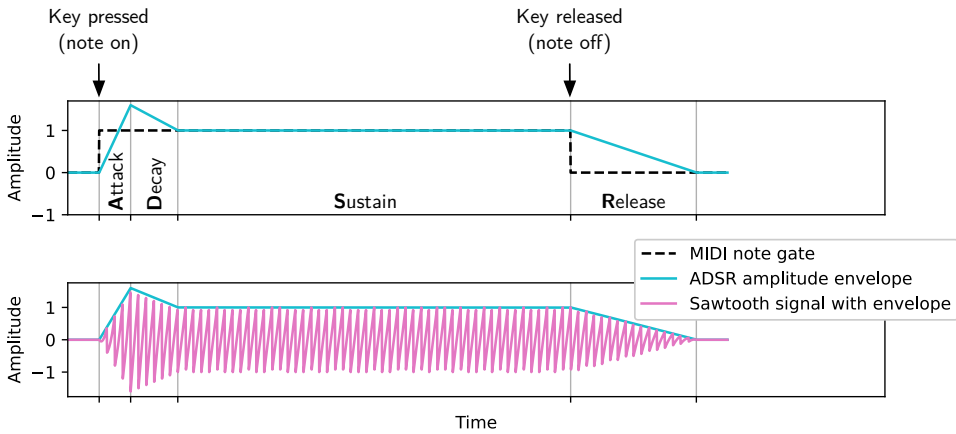


Figure 1.6. ADSR amplitude envelope applied to a sawtooth oscillator when a MIDI note is played (e.g. from a piano-like keyboard).

ADSR envelopes can also be used to control the evolution of filtering parameters such as cut-off frequency, as shown in Figure 1.5. The filter envelope displayed in this Figure 1.5 would produce a sweeping effect, with the resonance frequency going from the higher end to the lower end of the spectrum during the note’s sustain.

However, envelopes alone do not make sounds *alive*. Using deterministic oscillators (noise generators excluded), if the same note is played 1000 times using the same MIDI pitch and velocity, then the exact same waveform would be heard 1000 times. Moreover, the sustained part of the note would feel very static, i.e. its harmonic contents would remain exactly the same. This does not correspond to traditional acoustic instruments whose timbre is always slightly altered when the same note is played multiple times. Thus, modulation sources such as Low Frequency Oscillators (LFOs) introduce variability and movement into the sound. An LFO typically runs at a frequency below the 20–20,000 Hz audible range for humans, and its output can modulate synthesis parameters such as oscillators’ F_0 , filter cut-off frequency, or sustain level.

Some widespread LFO usages include the tremolo and vibrato effects. Tremolo is a modulation of the output waveform’s amplitude, which provides a rhythmic and dynamic modification of sound. An example of a strong tremolo,

based on a smoothed square LFO waveform, is given in Figure 1.7. Vibrato is a—usually small—modulation of the pitch of the note being played, which is extensively used by acoustic violin and guitar players. With subtractive synthesizers, it can be obtained by slightly modulating an oscillator’s F_0 from a sinusoidal LFO (Figure 1.8).

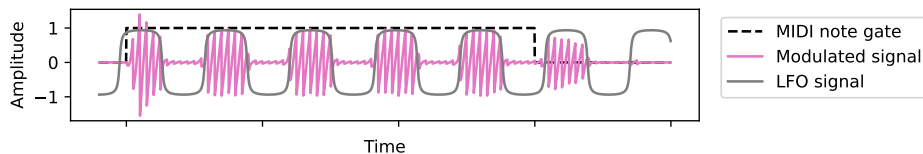


Figure 1.7. Tremolo effect obtained by using an LFO signal as amplitude modulation.

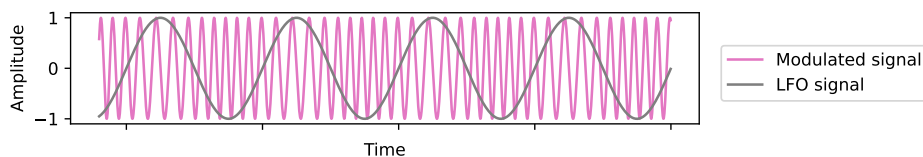


Figure 1.8. Vibrato effect obtained by using an LFO to modulate an oscillator’s F_0 . The amount of modulation is exaggerated for visualization purposes; in practice, vibrato is obtained by performing small F_0 variations.

1.2.3 Frequency modulation synthesis

FM synthesis is a powerful and complex method of sound generation, renowned for its ability to produce a wide array of timbres, from natural-sounding instruments to entirely unique sounds [1]. It involves the modulation of the frequency of one waveform, known as the carrier, by another waveform, called the modulator. The interaction between these waveforms creates intricate overtones and harmonic structures not achievable with subtractive synthesizers. Moreover, this type of synthesis offers a more efficient path to achieving complex spectra. Compared to additive and subtractive synthesis, it requires

fewer parameters, less memory, and reduced computation times [36]. It operates with a small set of oscillators, typically ranging from two to six, in contrast to the significantly higher computational demands of the additive approach [1].

In its simplest form, **FM** sound synthesis uses one sinusoidal carrier and one modulator. Disregarding initial phases, the modulated waveform's $y(t)$ can be simplified [37] as:

$$y(t) = A \sin(2\pi f_c t + I \sin(2\pi f_m t)) , \quad (1.1)$$

where A is the output peak amplitude, f_c and f_m are the carrier and modulator frequencies, and I is the modulation index. Equation 1.1 describes a phase modulation, but the **FM** and phase modulation denominations can be used interchangeably [38] because instantaneous frequency is the derivative of phase.

In the field of radio communications, **FM** typically uses $f_m \ll f_c$ where f_m holds the information being transmitted and f_c is a radio frequency. In both additive and subtractive synthesizers, a sinusoidal **LFO** can modulate the frequency of a sinusoidal oscillator. If their frequencies are denoted f_m and f_c , respectively, then the output signal can also be described by Equation 1.1 with $f_m \ll f_c$. However, the resulting effect would be perceived as a vibrato [36, p. 215], displayed in Figure 1.8, and is not considered true **FM** sound synthesis.

In contrast, **FM** synthesizers can use both f_m and f_c in the audible frequency range, where f_c defines the sound's pitch while f_m rather shapes the timbre. This allows the generation of harmonic and inharmonic sounds, as displayed in Figure 1.9 where the carrier frequency is set to $f_c = 100$ Hz. For instance, the first column with $f_m = f_c$ presents signals which contain odd and even harmonics. By setting the modulation index to a small $I = 0.6$ value, the result gets close to a sawtooth waveform. In the second column, where $f_m = 2f_c$, synthesized signals contain odd harmonics only. Using $I = 1$, the result resembles a low-pass filtered square waveform. The last column of Figure 1.9, where f_m is not a multiple of f_c , shows inharmonic signals only.

The first successful **FM** synthesizer is the Yamaha DX7, displayed in Figure 1.2(b). Commercialized in 1983, it provides a digital implementation

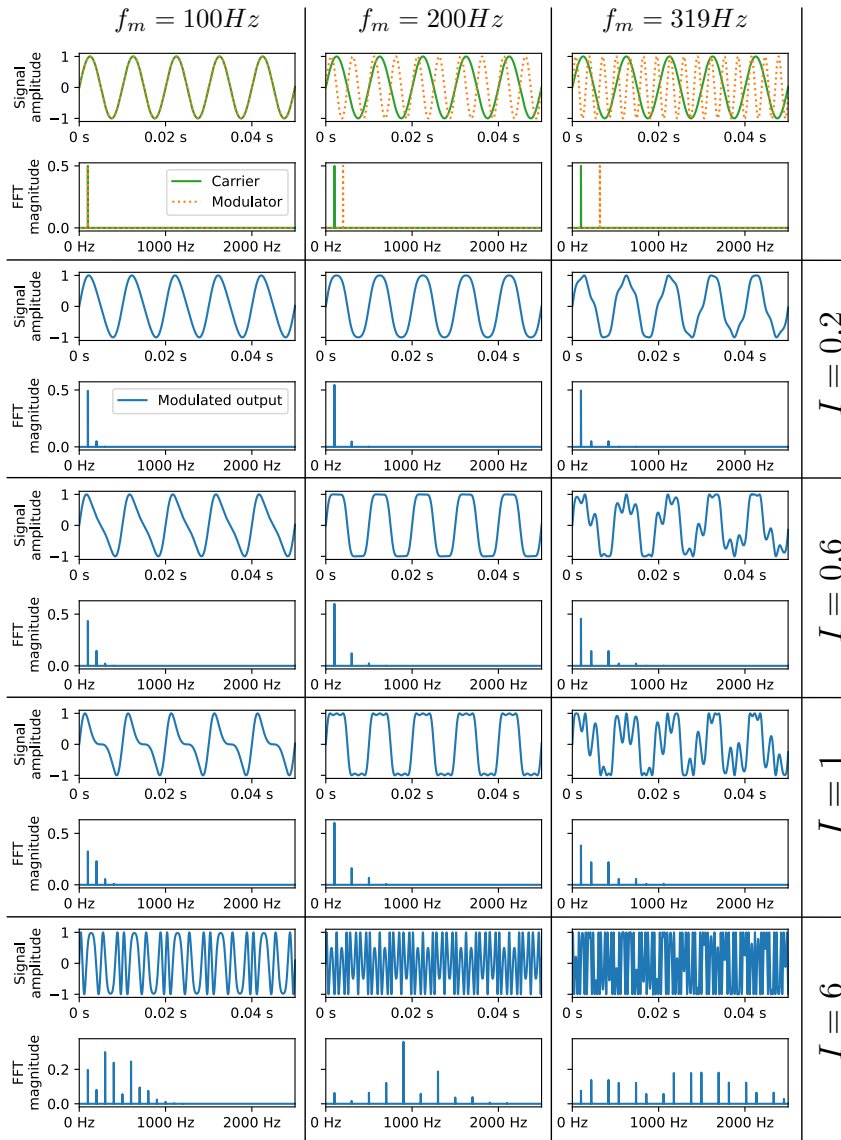


Figure 1.9. Basic FM signals synthesized using Equation 1.1 with a $f_c = 100\text{ Hz}$ carrier. Each column presents a different modulating signal frequency f_m , and each row presents a different modulation index I .

of FM synthesis with six oscillators, called *operators* [39]. While basic FM (Equation 1.1) considers sinusoidal modulators only, DX7 FM synthesis allows arbitrary modulator waveforms. The output signal $y_{op}(t)$ from a given operator can be written as:

$$y_{op}(t) = A_{op}(t) \sin(2\pi f_{op}t + x_{in}(t)) , \quad (1.2)$$

where f_{op} and $A_{op}(t)$ are the operator's carrier frequency and ADSR amplitude envelope, respectively, and $x_{in}(t)$ is an input modulation signal from another operator. The modulation index has disappeared from this equation because it is contained in the expression of $x_{in}(t)$ itself. This Equation 1.2 can be extracted from block diagrams of the original DX7 patent [40]. It can also be found in the C++ code⁶ of a software clone named Dexed [41].

The routing of audio-frequency signals in the FM architecture is called *algorithm* [39]. Three different algorithms are shown in Figure 1.10, and the whole set of 32 algorithms is presented in Figure B.1. In these figures, six rectangles represent the six operators, and each upper-level operator acts as a modulator for the operator beneath it. For instance, consider algorithm 1 from Figure 1.10. The sinusoidal output of operator 2 is routed to be $x_{in}(t)$ of operator 1. This algorithm also provides a deep modulation with three levels of FM (operators 6, 5, 4, 3). The number of levels may even be considered to be four, as the output of operator 6 is fed back to its input. Output signals $y_{op}(t)$ from operators 1 and 3 are finally summed and sent to the DX7 main audio output. Conversely, algorithm 32 sums the outputs of all operators, without using an operator to modulate another one. If the feedback of operator 6 is deactivated, then the DX7 turns into a purely additive synthesizer with six sinusoidal oscillators.

FM synthesis versatility made the DX7 one of the best-selling synthesizers of all time [2]. With hundreds of thousands of units sold, it made manufacturers realize that the synthesizer market was bigger than they thought [1]. It can be used as an additive synthesizer, or as a subtractive one using the embedded low-pass filter. Nevertheless, FM enables the generation of sounds that could

⁶https://github.com/asb2m10/dexed/blob/b6cf2e9190e216387ad96f977284ebbef765b447/Source/msfa/fm_op_kernel.cc, line 62, `FmOpKernel::compute` method

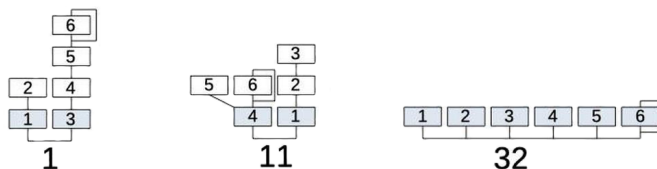


Figure 1.10. Three DX7 algorithms (out of a total of 32).

not be produced by previous methods. It is famous for its electric piano sounds, e.g. Fender Rhodes or Wurlitzer, such that some musicians stopped using their hardware counterparts [42]. The DX7’s sharp leads and deep basses were ubiquitous in 1980s pop music [2]. It also excels in synthesizing realistic percussion sounds, featuring inharmonic attacks and decays. In addition, it can generate sound effects, which can be defined as items that are not music or speech [43] Audio demonstrations of the DX7’s factory presets are available online [44].

Widespread modern FM synthesizers include Dexed [41] (a software clone of the original DX7), Arturia DX7 V, Korg Volca FM (a hardware clone), Native Instruments FM8, and Ableton Live Operator. Dexed’s User Interface (UI), presented in Figure 1.11, shows all DX7 parameters simultaneously, along with representations of the algorithm, ADSR envelopes, LFO waveform, etc. Figure 1.12 presents parameters related to one specific operator, and shows that each operator has its own frequency, envelope and modulation controls. The full set of DX7/Dexed synthesis parameters is described in Appendix B.1. Another FM synthesizer, named Ableton Operator, is briefly described in Appendix B.2. Although it has a different UI, the synthesis processes and parameters are very similar to Dexed’s.

1.2.4 Other methods

In the realm of sound synthesis, beyond the traditional additive, subtractive, and FM methods, several software techniques offer unique sonic possibilities and emulation capabilities. They can be divided into two categories: pure synthesizers, which generate audio samples from scratch using code, and sample-based synthesizers requiring pre-recorded sounds.



Figure 1.11. Dexed synthesizer in the VST plugin format, presenting all DX7 FM synthesis parameters.



Figure 1.12. Parameters of a single Dexed FM operator

On the one hand, physical and analog modeling approximate real-world instruments. Physical modeling synthesis simulates the physical properties and behaviors of acoustic instruments [1]. The Karplus-Strong method [45] is renowned for, and dedicated to, percussions and plucked strings synthesis. CORDIS [46] enables the synthesis of sound using mass-interaction simulations. Closely related, but not considered to be physical modeling, analog modeling aims at accurately replicating characteristics of classic analog synthesizers and filters. The resulting synthesizer is usually provided in a convenient software format, such as U-he Diva (Figure 1.2) or Arturia Analog Lab sold as VST plugins.

On the other hand, several methods rely on sample manipulation. Granular synthesis (Figure 1.13(a)) breaks an audio recording into grains, by applying smooth 1ms- to 50ms-long windowing functions [47]. Grains can be called *acoustical quanta* [47] that preserve some local frequency content from the original audio. They are eventually randomly selected and played based on parameters such as randomness and density. They produce easily distinguishable sounds, e.g. complex textures, evolving soundscapes or glitchy effects. Wavetable synthesis [1], another influential technique, uses pre-recorded periodic waveforms arranged in tables such as the ones displayed in Figure 1.13(b). The synthesizer can sweep through these waveforms, using LFOs and MIDI velocity to select different waveforms within a table. In order to obtain the desired F_0 , waveforms are stretched according to the pitch of MIDI notes. As they use waveforms from other synthesizers, wavetable sounds are usually difficult to distinguish as originating specifically from wavetable synthesis. Lastly, a sampler contains recordings of an acoustic instrument or hardware synthesizer, obtained for numerous pitches and velocities. It can alter the original samples by performing pitch and time adjustments, looping, or reverse playback. Practitioners typically view samplers and synthesizers as distinct types of devices [48, Section III]. However, some studies, e.g. [49], categorize *sampling synthesis* as an additional synthesis technique.

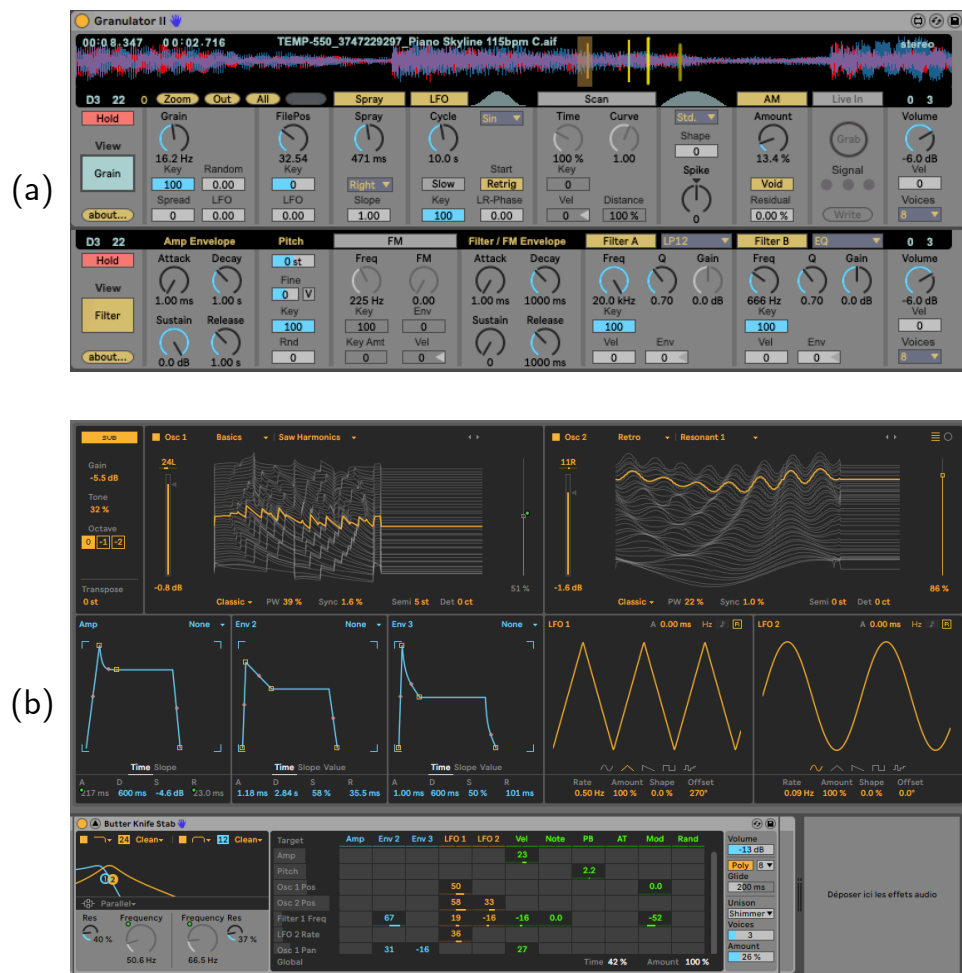


Figure 1.13. Two sample-based synthesizers from Ableton. (a) Granulator II. The input recording is displayed on the upper section of the user interface, with the grains being played represented by vertical yellow stripes. Granular synthesis and filtering parameters are available on the middle and lower sections, respectively. (b) Wavetable. The two parallel wave tables are on the upper part of the interface, LFOs on the middle right part, ADSR envelopes on the middle left part, and filters and modulations on the lower part.

1.3 Presets and control interfaces

1.3.1 Classic user interfaces

Section 1.2 has introduced the most common sound synthesis techniques, and some examples of successful commercial synthesizers implementing them have been presented in Figures 1.2 and 1.13. Nonetheless, a clear distinction should be made between sound synthesis processes (also known as engines) and synthesizers' UIs, as illustrated in Figure 1.14. The typical synthesizer UI is made of knobs⁷, sliders⁸, buttons and switches mapped to low-level synthesis parameters such as waveform type, cut-off frequency and attack time. Software designers purposefully reproduce UIs of hardware synthesizers [50, 51], such that software and hardware UIs tend to remain quite similar. Modern digital synthesizers nonetheless provide more visual feedback, e.g. filter frequency response and ADSR envelope plots, or oscilloscope views.

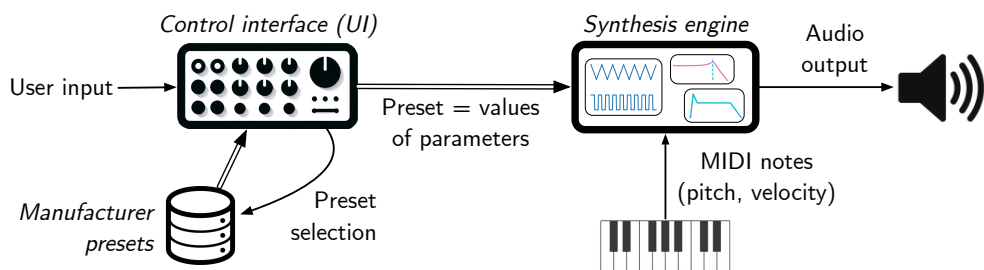


Figure 1.14. Schematic view of a synthesizer split into a synthesis engine and a UI.

Although each manufacturer features a unique UI signature, a consistent trait among all synthesizers is the extensive amount of controls offered to the user, ranging from dozens to hundreds. The large number of parameters can be observed in Figures 1.2, 1.11 and 1.13, e.g.:

- The Minimoog, arguably one of the simplest synthesizers, has 50 synthesis controls (excluding preset selection and MIDI configuration controls).

⁷Sometimes referred to as potentiometers, although the latter denotes an electric component.

⁸Also named faders.

- U-he Diva has 281 parameters, some of them being accessible from menus or from hidden elements of the modular interface.
- The Yamaha DX7 provides 155 synthesis parameters (detailed in Appendix B.1).

However, synthesis parameters may not always have a clear acoustical meaning, and they are often interdependent. Handling them individually can be a challenging and tedious task, potentially hindering inspiration and productivity [52]. Therefore, manufacturers provide numerous presets, i.e. sets of values of all parameters. The terms *voices* or *patches* are also sometimes employed to refer to presets. They can usually be selected directly from the UI, for the user to easily access the wide range of sounds enabled by the underlying synthesis engine.

1.3.2 Preset manipulation

A preset defines the timbre that a synthesizer generates. Once a given preset has been chosen by the user, it can be tweaked using individual UI controls in order to refine the sound. Although some users have to rely on *factory* presets only [2, 8], a recent study [6] with 122 participants has shown that subjects tend to modify or create presets more frequently than they utilize existing ones. However, configuring synthesizers using low-level numerical values was already considered old-fashioned in 1995 [52]. Almost 30 years later, this statement remains true and mainstream manipulation methods have not changed much.

Although Human-Computer Interaction (HCI) is not the main topic of this thesis, interactions between a synthesizer and a musician or producer can be briefly analyzed through the human action cycle [53] model. Figure 1.15 shows Norman's seven-step feedback cycle applied to a preset design task. The first step consists in formulating a goal to achieve. Steps 2–4 model the execution of an action or sequence of actions, where as steps 5–7 describe how a user evaluates the synthesizer's output. The final transition from step 7 to 1 may lead to a re-assessment of the goal itself. In an ideal situation, all steps would be very easy to perform.

In practice, most elements of both hardware and software synthesizer interfaces are easily accessible and require almost no mechanical force to manipulate. Hence, step 4 will be easy to perform. If the synthesizer is suitably connected

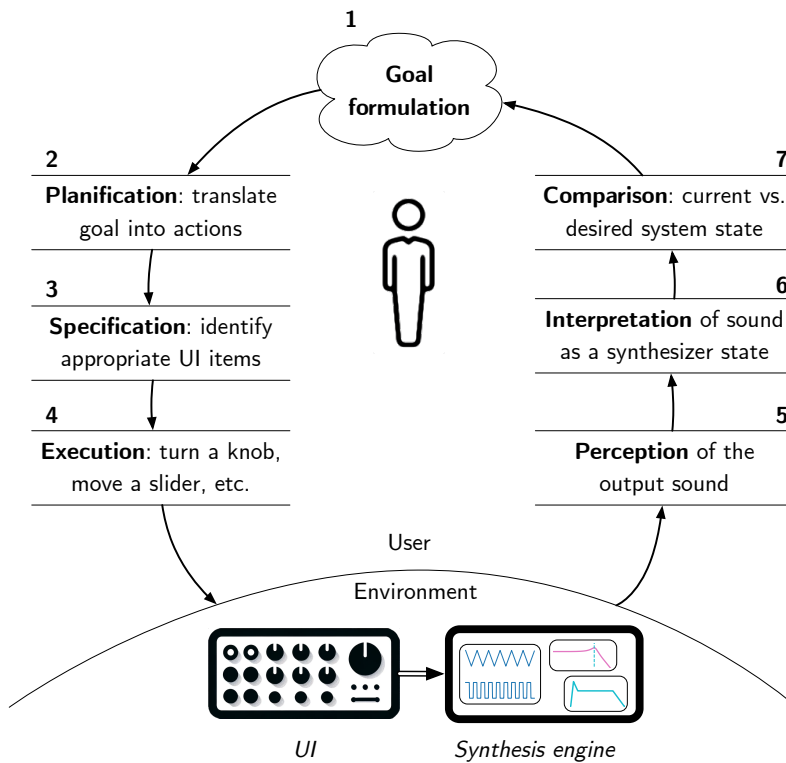


Figure 1.15. The human action cycle [53] used to model interactions between a user and a sound synthesizer.

to loudspeakers or headphones, then perception in step 5 should not be a problem either. Regarding the other steps, easiness is highly dependent on the goal, the user’s skills regarding sound synthesis, and the synthesizer itself. Two examples based on the Minimoog (Figure 1.5) are given hereafter.

First, consider the following goal: “Making the current preset sound brighter”. Formulating this goal requires the user to know what sonic possibilities are offered by the Minimoog. Then, the goal might be achieved by increasing the frequency of the low-pass filter (step 2), whose corresponding knob can be quickly identified on the front panel (step 3). After turning the knob (step 4), a different sound can be heard (step 5) and it seems to contain more high

frequencies (step 6). The goal seems to be achieved (step 7) so the cycle stops there.

Second, consider another, higher-level goal: “Making a preset that sounds like a clarinet”. Also consider that the current preset sounds like a bright brass instrument. An expert user might know that the clarinet’s spectrum is primarily made of odd harmonics (square waveform) with some lower-amplitude high-frequency even harmonics [54–56]. Therefore, setting one of the oscillators to use a square waveform may be appropriate (step 2). The waveform selection knob for oscillator 1 can be easily located (step 3) and is turned towards the square-signal shape (step 4). A slightly different timbre can be heard from the synthesizer (step 5), but it is almost identical to the previous one. This is not the synthesizer’s desired state (step 6). Possible explanations (step 7) could include: the volume of oscillator 1 being too low; the volume of other oscillators being too high; an unexpected modulation transforming the square waveform; the Minimoog’s square waveform not being appropriate to model a clarinet sound. The initial goal has not been reached, mainly because of the usually complex mappings between synthesis parameters and perceived output sound characteristics [6, 57, 58].

Considering the task difficulty, this second goal could be reformulated into an easier “Search for a clarinet preset from the Minimoog’s factory presets” (step 1) and a new cycle follows (steps 2–7). Several complete feedback cycles can occur until a proper preset is found.

Manipulating a synthesizer, even a simple subtractive one, requires expertise. As a small informal experiment, the ChatGPT Large Language Model (LLM) assistant has been used to try to reach that second goal: “Making a preset that sounds like a clarinet”. This famous model is able to pass history, biology, chemistry or statistics exams [59]. However, as reported in Appendix C, it is not yet able to help obtain a clarinet sound from a Minimoog synthesizer. While it offers some general guidelines, many suggestions regarding parameter values are vague and lack practical utility, with some incorrect assertions also present.

The given examples illustrate that presets are difficult to manipulate and to create from scratch. This challenge is widely acknowledged in the literature [6, 7, 10, 50, 51, 57, 58, 60–65]. Designing and adjusting presets requires not only a deep understanding of the synthesizer’s architecture but also a significant

amount of time and experimentation. This complexity underscores the need for more intuitive methods, interfaces and tools that can simplify the sound design process, making it more accessible to users at all skill levels.

1.3.3 FM synthesis presets

While FM synthesis is versatile and commercially successful (Subsection 1.2.3), it relies on a large number of parameters such as frequencies, envelopes, modulations and algorithms. For instance, the Yamaha DX7 has 155 parameters (Appendix B.1), Ableton Operator has 137 (Appendix B.2), and Native Instruments FM8 has more than 1000 [64]. Moreover, the intricate interdependencies of parameters for FM synthesis make it much harder to grasp [58, 60, 61] than additive, subtractive, or wavetable synthesis.

The FM synthesis space is also nonlinear [66], i.e., a linear increase of a given parameter's value is not likely to result in a perceptually linear modification of timbre dimensions. For instance, consider a knob used to control an oscillator's F_0 . Such a knob typically adjusts frequency on an exponential scale to accommodate the logarithmic perception of pitch (Section 1.1). Thus, if the oscillator is a carrier, then the relationship between knob position and perceived pitch modification is close to linear. Nonetheless, if the oscillator is a modulator, then rotating the knob entails complex changes in the overtones' amplitude and spectral location (e.g., Figure 1.9). The mapping between the position of this frequency knob and the perceived modifications of timbre attributes is non-linear. Any small change of a modulator's ADSR parameter will also have significant, non-linear effects on the output spectrum.

Regarding DX7 (introduced in Subsection 1.2.3) presets, the Musikinstrumenten-Museum, Berlin, Germany [2] states that:

“The Yamaha DX7 FM synthesizer was notoriously hard to program, and many artists chose to work instead with its built-in presets.”

Another small informal test was conducted by requesting ChatGPT to program a clarinet preset into a DX7 synthesizer. The answer, reported in Appendix C.3, holds almost no practical value. While the LLM assistant shows some general knowledge about DX7 parameters, it is completely unable to provide useful values.

These insights into FM synthesis, particularly with the DX7, highlight its complexity and intriguing challenges as a case study. Therefore, this thesis will focus on preset creation, interpolation and manipulation for this specific synthesizer. A model that performs well on the DX7 can be expected to be applicable to virtually any other commercial synthesizer.

1.4 Differentiable synthesizers

Advancements in neural networks in the last ten years have enabled the creation of *differentiable* synthesizers, whose internal synthesis processes are fully differentiable from input controls to output sound. They can generate audio by modeling raw waveforms, or by using differentiable oscillators and Digital Signal Processing (DSP) blocks. The internal weights of neural synthesizers are learned using gradient descent [67, Chapter 6], usually for the task of sound reproduction. Although these synthesizers have not been adopted by the industry, for various reasons discussed in the following subsections, they introduce interesting ideas and model architectures. This review of literature focuses on data-driven techniques for monophonic⁹ audio generation from musical notes (e.g. MIDI notes). Therefore, it excludes works related to multi-instrumental music generation, such as JukeBox [68], Riffusion [69] or Noise2Music [70] which produce music from text inputs.

1.4.1 Neural waveform synthesis

The first type of neural synthesis—sound synthesis from differentiable neural networks—involves generating raw audio waveforms directly, instead of using the traditional oscillators and filters. One of the first¹⁰ models has been NSynth [71] whose architecture is presented in Figure 1.16.

Its autoencoder structure is widely used for feature learning, dimensionality reduction or generative modeling [67, Chapter 14]. First, the input vector \mathbf{x} is encoded into a \mathbf{z} vector (Figure 1.17). This hidden \mathbf{z} , computed by a deep

⁹A *polyphonic* synthesizer typically relies on multiple identical *monophonic* synthesis processes running in parallel.

¹⁰While some previous synthesizers have claimed to be *neural* [1, p. 61], no documentation regarding their inner structure could be found.

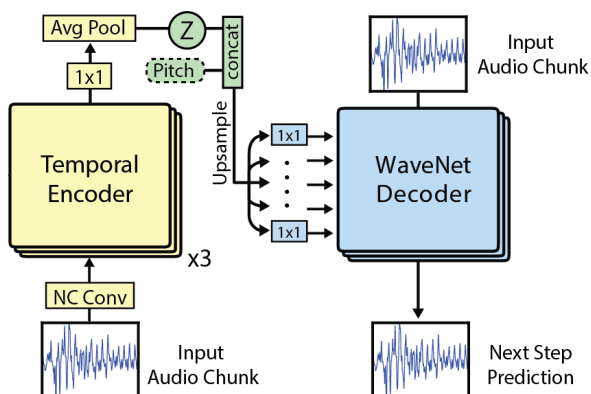


Figure 1.16. NSynth [71] model: autoencoder structure with Wavenet [72] autoregressive decoder. Figure from [71].

hidden layer in the network, can be called an embedding or a latent vector. Second, \mathbf{z} is provided to the decoder which tries to reconstruct \mathbf{x} . The learning process consists in minimizing the following loss function:

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) = \mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}), \quad (1.3)$$

where $\hat{\mathbf{x}}$ is the decoded output, f and g represent the encoder and decoder, respectively, and \mathcal{L} computes a dissimilarity. \mathcal{L} is usually called the reconstruction loss, and can be, e.g., a Mean Squared Error (MSE).

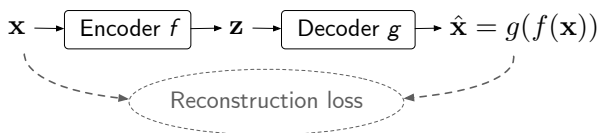


Figure 1.17. Vanilla autoencoder model

NSynth is trained on a dataset of 4s-long, 16kHz waveforms. This dataset—also named NSynth—provides 300k individual musical notes obtained from 1000 instruments [73] played using different MIDI pitches and velocities.

The encoder is a deep Convolutional Neural Network (CNN) [74] which computes convolutions with non-linear activations [67, Chapter 6] along the temporal axis of \mathbf{x} . Such a network can be referred to as a Temporal Convolutional Network (TCN). Its output, denoted Z in Figure 1.16, is shaped as a 125×16 matrix rather than a 2000-dimensional \mathbf{z} vector. Z can be interpreted as 125 vectors of dimension 16, where each vector encodes a 32ms-long time frame of \mathbf{x} . Such autoencoders, whose latent codes are smaller than the input, are called undercomplete [67, Chap. 14]. The equivalent data compression rate is 32:1. Ideally, Z would be a compact representation of timbre that is orthogonal to the input’s pitch and loudness.

The NSynth decoder, however, is not as simple as the one presented in Figure 1.17. It is built on Wavenet [72], originally used for speech modeling and synthesis. This model uses causal temporal convolutions, ensuring that a sample x_t only depends on the samples from the previous time steps. The decoder’s outputs are parameters of the joint probability distribution below:

$$p_{\text{decoder}}(\mathbf{x}) = \prod_{t=1}^T p_{\text{decoder}}(x_t | x_{t-1}, \dots, x_1, Z, \text{pitch}), \quad (1.4)$$

where $\mathbf{x} = \{x_1, x_2, \dots, x_T\}$. Instead of minimizing a straightforward reconstruction loss, e.g., a MSE, Wavenet is trained by maximizing the input’s likelihood described in Equation 1.4. In practice, this is equivalent to minimizing the input’s Negative Log-Likelihood (NLL), i.e., $-\log p_{\text{decoder}}(\mathbf{x})$, which is considered as a loss and can be computed as a numerically stable sum.

In order to estimate the probability distribution of a sample x_t , this decoder uses all previous samples $\{x_1, \dots, x_{t-1}\}$, as well as the note’s latent representation Z and pitch. Therefore, it belongs to the group of autoregressive networks which are considered to be powerful models [75] and generate high-quality waveforms [71, 72]. However, it requires x_t to be generated before moving to the next step and sampling x_{t+1} from $p_{\text{decoder}}(x_{t+1} | x_t, \dots, x_1, Z, \text{pitch})$. This iterative sampling prevents the model from using the fast parallel computations enabled by a Graphics Processing Unit (GPU). Therefore, the synthesis process is very slow, e.g., a 4s-long audio chunk is generated in approximately 15 to 20 minutes using a low 16kHz sampling frequency. A live demonstra-

tion [76] of NSynth’s playability has been provided by the authors. However, the sounds are not computed in real-time and have instead been pre-computed on a powerful GPU. The demonstrator itself is only a sampler.

With a Wavenet-based model, the audio resolution is reduced to 8 bits on a logarithmic scale [72]. In comparison, standard quality audio is encoded at 48kHz and 16 bits (linear) [77], and trained subjects can discriminate between standard- and higher-resolution contents. A synthesizer used as a DAW plugin typically uses a 32-bit floating-point representation.

Several works have been inspired by NSynth. For instance, it has been used as a part of the Neural Wavetable synthesizer [78]. However, the latter remains Wavenet-based and the associated very long computation times require audio waveforms to be pre-computed. Other studies have obtained faster generation using parallel sample computation. E.g., SING [79] uses a mostly convolutional, non-autoregressive autoencoder for 16kHz audio waveforms. Instead of maximizing the probability of each individual audio sample (Equation 1.4), it is trained using a spectral loss, which corresponds to the loss of Equation 1.3 but L is the Mean Absolute Error (MAE) and \mathbf{x} and $\hat{\mathbf{x}}$ are log power spectrograms rather than raw audio waveforms. Compared to NSynth, this enables faster audio generation using a GPU. Outputs values are also modeled as continuous, i.e., SING discards the 8-bit quantization. Multiple works also incorporate Generative Adversarial Networks (GANs) [80], which were initially successful at generating high-quality, realistic images. They can be repurposed in order to generate audio waveforms or spectrograms. MelGAN [81] can be used as a Wavenet replacement for raw speech and music synthesis. As opposed to the previously cited models, it does not require a GPU to run. It allows a $2\times$ faster than real-time audio inference on a Central Processing Unit (CPU), i.e., 1 second of audio is generated in 0.5 seconds. Another similar model is WaveGAN [9]. Nonetheless, both MelGAN and WaveGAN degrade the perceived audio quality. The RAVE [82] fully-convolutional synthesizer combines an autoencoder and a GAN. It enables both a fast inference ($20\times$ faster than real-time on a CPU), and a higher reconstruction quality compared to NSynth or SING. However, these performances were obtained by training independent RAVE models on different limited datasets, e.g., only violin, only darbuka drums, or only speech. A trained RAVE model corresponds more to a specific preset of a synthesizer than to a generic synthesizer. In contrast, a single NSynth or SING trained model can reproduce approximately 1000

different instruments from the NSynth dataset. Moreover, RAVE does not allow for playing **MIDI** notes at specific times, with controlled pitches and velocities. It is rather intended for audio manipulation through modifications in the latent space.

1.4.2 Neural spectral synthesis

Musical notes can indeed be generated through spectrogram synthesis and inversion, although the resulting audio quality is often lower than that obtained from raw audio synthesis. Several works [83, 84] model magnitude spectrograms of musical notes using an autoencoder, as displayed in Figure 1.18. The raw audio input is converted into a spectrogram using a Short-Time Fourier Transform (**STFT**) whose magnitude is provided to the encoder. Its output is decoded as a magnitude spectrogram, and the corresponding phase must be estimated before computing the inverse **STFT**. Similarly, [85] introduces an autoencoder for time frames of the Constant-Q Transform (**CQT**) [86] of input sounds. The separate processing of phase is quite commonly found [87, 88] in related works.

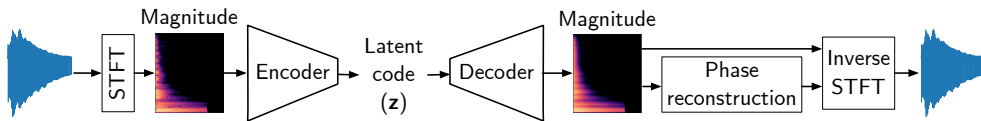


Figure 1.18. Magnitude spectrum autoencoder with output phase reconstruction.

Phase reconstruction is typically performed using the Griffin-Lim iterative algorithm [89], possibly taking input phase as initial value [84], and sometimes using other techniques such as phase gradient heap integration [90]. However, the subjective quality of audio remains subpar [9, 71], especially when compared to, e.g., Wavenet-based audio synthesis. Many artifacts, caused by erroneous phase reconstruction or approximate magnitude decoding, can be heard.

Leveraging the substantial amount of machine learning research on image generation, GANSynth [91] utilizes a **GAN** to generate the **STFT** magnitude and unwrapped phase simultaneously. It enables a higher audio quality and runs faster than Wavenet, but still requires a **GPU**. For the specific purpose of

percussive synthesis, DrumGAN [92] generates the real and imaginary parts of the **STFT** directly.

Nonetheless, they all suffer from the known drawbacks of GANs. First, a **GAN** does not provide an explicit and precise control over the timbre to be generated. In contrast, the encoder of an autoencoder (Figure 1.18) allows the computation of a latent code that corresponds to a given target timbre, whereas a **GAN** only provides a decoder—called generator. GANs can, however, be *conditioned* [93] to generate a specific class of output (e.g., a given family of instruments). Second, GANs are prone to mode collapse [94] where the generator learns to produce very realistic but always similar samples. Hence, there is no guarantee that a **GAN** model is able to reproduce the whole range of timbres from the training dataset.

1.4.3 Differentiable digital signal processing

Some models presented in the last subsections can synthesize high-quality audio, but they require either a slow auto-regressive decoder (part of an autoencoder architecture) or to train a **GAN**. In 2020, a new class of differentiable synthesizers, based on Differentiable Digital Signal Processing (**DDSP**) blocks, has emerged. They are trained to estimate the parameters of classical synthesis processes, e.g., oscillator frequency and amplitude, envelope parameters, or Finite Impulse Response (**FIR**) filter coefficients. Then, these differentiable processes can synthesize raw audio waveforms. In contrast, previous methods required to learn a very large number of **CNN** weights.

Differentiable additive synthesis

The first **DDSP** synthesizer [95] enables differentiable additive synthesis of harmonic components and filtered noise sources, also called Harmonic plus Noise (**HpN**) synthesis [96]. The model is trained as an autoencoder whose structure is presented in Fig. 1.19. The resynthesis objective is a Multi-Scale Spectral (**MSS**) loss based on differences between **STFT**s of the ground truth and resynthesized audio. Fast Fourier Transform (**FFT**) window sizes are 2048, 1024, 512, 256, 128 and 64, which cover different spectro-temporal resolutions.

The encoder is based on two sub-networks. One is the CREPE [97] pre-trained neural network for instantaneous F_0 estimation. The other estimates some time-varying $\mathbf{z}(t)$ values from the Mel-Frequency Cepstral Coefficients (MFCCs) [98] extracted from the target audio. The instantaneous loudness is computed in parallel. New time-varying pitch, loudness, and \mathbf{z} values are provided to the decoder every 4 milliseconds. Each $\mathbf{z}(t)$ is called a *residual embedding* and can be considered as an instantaneous representation of timbre. If the dataset only comprises a specific instrument timbre, e.g., only flute, violin or sitar, then $\mathbf{z}(t)$ can be discarded during training and inference.

The synthesizer itself is made of a decoder neural network, as well as differentiable oscillators, noise generation and reverberation modules. The decoder computes the synthesis parameters, i.e., the magnitude of 101 harmonics and 65 noise bands, for each time frame.

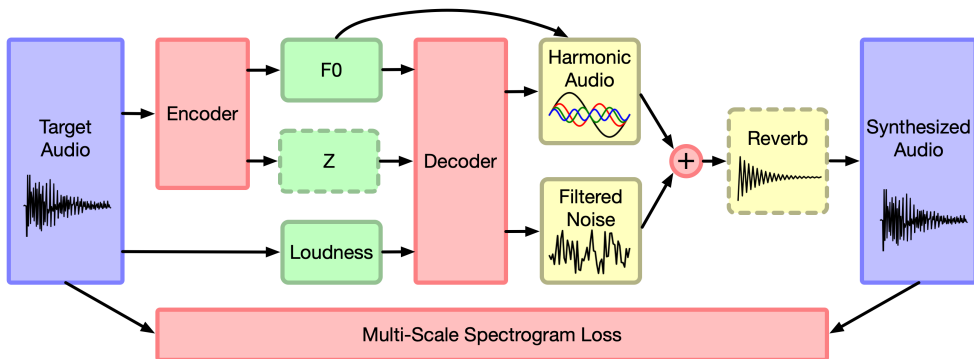


Figure 1.19. Autoencoder model used to learn a differentiable additive synthesizer (Figure from [95]).

Similar to RAVE, this model enables new impressive applications such as timbre transfer, also called tone transfer [99]. Oscillators are nonetheless restricted to a purely harmonic configuration, thus preventing the modeling of inharmonic instruments such as bells or percussions. Compared to NSynth, the audio quality of resynthesized samples from the NSynth dataset is improved [100] while the synthesis process is faster than real-time.

Inputs of the decoder from Figure 1.19 are not MIDI notes, which prevents the DDSP synthesizer from being played from a MIDI keyboard. The extended

MIDI-DDSP model [101] enables straightforward control of a DDSP decoder using extra encoders that take notes as inputs. These encoders are called *expression generator* and *synthesis generator*. For instance, they can transform the MIDI pitch and some vibrato modulation into a sequence of F_0 values, while the MIDI velocity and gate (Figure 1.6) can be used to generate a sequence of loudness values.

A real-time VST implementation [102] is provided by the Google Magenta team. It features eleven DDSP models where each model has been trained to reproduce a given instrument (e.g. flute, violin, sitar, etc.). However, for all instruments, output sounds often contain audible artifacts. The general issue of audio quality with neural synthesizers will be discussed more in Subsection 1.4.4.

Differentiable FM synthesis

Several works followed the first DDSP model and have tried to reproduce various classic synthesis processes using differentiable blocks. For instance, the recent DDX7 [103] study is a first step towards reproducing the DX7 FM synthesis using neural networks. The DDX7 overall architecture (Figure 1.20) is quite similar to the differentiable additive architecture (Figure 1.19). Besides pitch extraction using CREPE [97] and oscillators' frequency assignation, a TCN extracts time-varying envelope signals (i.e., volumes) for all oscillators. The model is trained to resynthesize either violin, flute or trumpet sounds from the University of Rochester Music Performance (URMP) dataset [104] by minimizing a MSS loss.

Using a small test set of violin, flute or trumpet sounds, the resynthesis performance of DDX7 is comparable to that of the HpN DDSP model. Nonetheless, DDX7 requires less synthesis parameters and computational resources. A subsequent work [105] improves target reconstruction and tone transfer using differentiable FM synthesizers.

However, numerous aspects of the original DX7 are missing, and are listed below.

- A given DDX7 model implements a single FM algorithm.

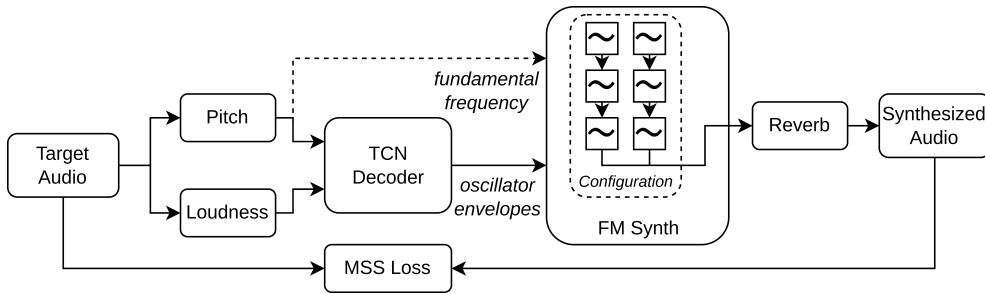


Figure 1.20. Autoencoder model for differentiable FM synthesis with a fixed configuration of FM oscillators (Figure from [103]).

- Given an FM algorithm, the ratios between frequencies of different oscillators are also fixed. They are hardcoded and depend on the timbre being synthesized (either violin, flute or trumpet). Modulation indices (denoted I in Equation 1.1) are also limited to a predefined value for each algorithm.
- Oscillator feedback cannot be implemented because it would prevent all time steps of a waveform from being generated in parallel. Non-parallel synthesis would lead to unreasonably long training and inference times.

Other DDSP synthesizers

Besides the HpN and FM neural synthesizers, several other attempts at reproducing classical synthesis techniques using DDSP modules have been published. Neural waveshaping [106] uses an harmonic excitation processed through non-linear distortion units. It does not perform better than the original HpN DDSP, but the underlying neural network possesses 20 times fewer parameters. Differentiable wavetable synthesis [107] relies on a learned dictionary of 20 periodic waveforms. The model is trained to compute a weight for each waveform, in order to approximately reproduce a given target sound.

Other recent studies focus on improving the differentiable resynthesis of the inharmonic or transient parts of sounds, which are not properly modeled by the previously cited DDSP synthesizers. For instance, DDSP-Piano [108] is based on HpN DDSP but is able to detune some partials in order to introduce

small inharmonicities. A differentiable guitar model [109] uses six parallel synthesizer modules with inharmonic oscillators and filtered noise. For unpitched percussive sounds, [110] uses a combination of sinusoidal resynthesis and TCN-based transients reconstruction. NoiseBandNet [43] improves the filtered noise reconstruction of transients from the HpN model (Figure 1.19). It separates white noise using very narrow (from 5 to 30Hz) filter banks, and the network is trained to control the audio level of each band. The reconstruction of sound effects (e.g., thunderstorm, metal or pottery impacts, footsteps, etc.) is more accurate and sounds convincing. However, the huge number of coefficients of the narrow FIR filters increases the network’s size and computational requirements.

1.4.4 Limitations

A large number of neural audio synthesis systems have been published since 2016. They have been widely adopted by the industry for applications such as text-to-speech¹¹. Yet, differentiable synthesizers are rarely used in music creation workflows [106]. This subsection explores several reasons for the limited adoption of neural synthesizers in general.

Audio quality and diversity

The main limitation of neural audio synthesizers applied to musical sound generation may be the output quality. Some limitations and audible artifacts regarding specific neural synthesis architectures have been mentioned in the previous subsections. A studio report [111] about DDSP usage states that artists can purposefully use the unwanted artifacts. New interesting sounds can be created. However, the degraded audio quality prevents a more general usage of neural synthesizers.

Recently, a large-scale study [100] has compared three types of models using the Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) evaluation protocol [112]. Participants had to rate five sounds at the same time, including a reference sound from the NSynth test dataset. The anchor sound was obtained by low-pass filtering at 1 kHz and reducing the quantization to

¹¹E.g., Amazon Polly, OpenAI GPT-4 Omni, etc.

8-bit. The three other sounds were reconstructions through NSynth waveform modeling and two **DDSP** models (**HpN**, and waveshaping).

MUSHRA ratings range from 0 to 100 where the 0–20, 20–40, 40–60, 60–80 and 80–100 intervals indicate a *bad*, *poor*, *fair*, *good* and *excellent* quality, respectively. The reference obtained an *excellent* median rating. The worst model, NSynth, has a *poor* median subjective audio quality. The two **DDSP** models are rated significantly higher than NSynth, although their quality is merely *fair*. This *fair* rating is also obtained by the anchor, which is a substantially degraded signal.

Moreover, a trade-off seems to exist between audio quality and timbre diversity. On one hand, Wavenet-based models are able to approximately generate arbitrary audio waveforms (for example, any instrumental family from the NSynth dataset), but the audio quality is lower and synthesis requires huge computational resources. On the other hand, models such as RAVE [82] or the **HpN DDSP** [95] reach better audio quality. A RAVE model—typically made of millions of weights—reaches optimal performance when trained to reproduce a narrow set of timbre. **HpN DDSP** models have strong assumptions about signal models and applying them to a wider range of non-harmonic sounds is very challenging [113]. Recent and more specific **DDSP** models such as the differentiable guitar [109] do not sound convincing yet.

Building and training **DDSP** modules

Models based on **DDSP**, which seem to be the best neural synthesis techniques, indeed rely on the *differentiable* characteristic of neural networks. This prevents them from modeling an arbitrary process. For instance, they have fixed signal flows and currently cannot integrate modularities such as a variable **FM** algorithm [103].

Moreover, **DDSP** models based on sinusoidal oscillators (e.g., additive or **FM**) cannot always be trained by straightforward gradient descent [113]. Constraints must be applied to the oscillators, enforcing a harmonic structure [95] for additive synthesis or a fixed routing [103] for **FM** synthesis.

Many common processing blocks involve feedback loops, which are challenging to integrate into a neural network. Auto-regressive processes, such as Infinite Impulse Response (**IIR**) filters or oscillator feedback, require samples to be

computed one-by-one. This prevents fast parallel training and inference of all samples in a given time segment. Training differentiable **IIR** filters can rely on manual gradient computations [113] or approximations [113, 114]. They nonetheless typically suffer from numerical instabilities [113, 114].

A more practical issue is the limited availability of some models and **DDSP** blocks, whose source code and weights are often not available [100]. Compared to their traditional counterparts, developing neural synthesizers entails high engineering costs [113, 115] and requires large training resources [100]. The limited availability of high-quality, monophonic musical training data may be another issue [113].

1.5 In brief

Summary of Chapter 1

- The main architectures (e.g. additive, subtractive, **FM**, wavetable, etc.) used in conventional synthesizers have been described.
- Manipulating presets in modern synthesizers is a complex task.
- **FM** synthesizers are particularly noteworthy due to their broad sonic palette and their reputation for being challenging to handle.
- Neural (differentiable) audio synthesizers have been introduced, opening up new applications. However, their current limitations have hindered widespread adoption among users and the industry.

Perspective for Chapter 1

- **FM** synthesis, specifically the DX7 **FM** architecture, has been selected as the primary case study for this thesis.
- The following chapters will focus on widespread conventional synthesizers rather than their differentiable approximations.
- Although neural sound synthesis is not the main topic of this thesis, it remains a very active research area and presents promising avenues for future works.
- Timbre has been only briefly introduced. A more extensive discussion about timbre description and feature extraction is available in Chapter 3.

Chapter 2

Automatic synthesizer programming

Contents

2.1	State of the art	45
2.1.1	Task definition	45
2.1.2	Heuristic methods	46
2.1.3	Data-driven techniques	48
2.1.4	Related fields and commercial applications	55
2.2	Dataset of DX7 presets	56
2.2.1	Data collection and filtering	56
2.2.2	Automatic labeling	58
2.3	PresetGenVAE model	62
2.3.1	Variational Autoencoder	62
2.3.2	Normalizing Flows	69
2.3.3	General model architecture	73
2.3.4	Output preset modeling	76
2.3.5	Implementation and training	78
2.4	Results and model analysis	80
2.4.1	Preset representation and decoder	80
2.4.2	Multi-channel spectrograms	83
2.4.3	Discussion	88
2.5	In brief	90

This chapter is partially based on the following publication:

- G. Le Vaillant, T. Dutoit and S. Dekeyser, "Improving Synthesizer Programming From Variational Autoencoders Latent Space," *24th International Conference on Digital Audio Effects (DAFx)*, Vienna, Austria, 2021, pp. 276-283, doi:[10.23919/DAFx51585.2021.9768218](https://doi.org/10.23919/DAFx51585.2021.9768218).

Automatic synthesizer programming, also called sound matching, automatic cloning of sounds, or timbre mimicking, consists in finding the preset that corresponds best to an input sound. While the task of finding an optimal preset can be fulfilled manually, it is often cumbersome and requires expert skills because many synthesizers provide dozens to hundreds of controls. It usually relies on trial and error [116], which is time-consuming and can significantly disrupt the creative workflow of composing and producing [10]. Numerous works have explored this topic, focusing on various synthesizers. In particular, sound matching for FM synthesis is a notoriously difficult problem and is the main topic of this chapter. Its four main contributions are the following:

- An extensive review of sound matching techniques published in the last three decades;
- A dataset of DX7 FM presets with automatic labeling based on traditional signal and text processing methods;
- The PresetGenVAE model, which was state-of-the-art for DX7 FM automatic programming at the time of its publication 2021;
- The first model that specifically considers synthesis parameters related to pitch and intensity, by processing multiple audio inputs rather than a single one.

2.1 State of the art

2.1.1 Task definition

An overview of the sound matching task is presented in Figure 2.1. Given an input target sound, that may have been generated by a synthesizer or recorded from an acoustic instrument, a sound matching model searches for the preset that enables the best reconstruction through a specific synthesizer. Most models described in the following subsections use input sounds that all have the same pitch (e.g., corresponding to the middle C3 MIDI note).

A sound matching algorithm can help users in the sound design process. For novice users, a matched preset can be used to retrieve a specific sound. For experts, it can serve as a starting point for further sound design [10]. Using a

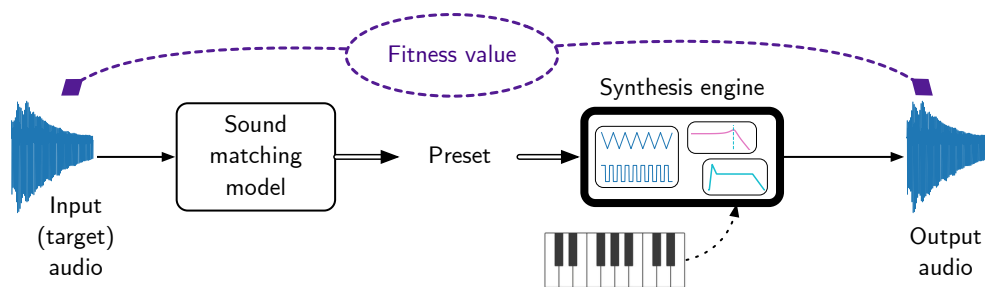


Figure 2.1. Overview of the sound matching task. The goal is to maximize fitness between input and output sounds.

synthesizer to recreate a sound allows for flexible adjustments to pitch, duration, and velocity. Samplers could also be used for these tasks, but they rely on audio time-stretching that can undesirably alter timbral qualities [8].

2.1.2 Heuristic methods

The first attempt at sound matching [117] has been published in 1982. Using a basic modulator-carrier FM synthesizer, it minimizes the spectral difference between the output and target sounds (Figure 2.1) using closed-form equations [118]. In this work, the synthesizer itself has only one variable parameter, which is a modulation index.

Subsequent works have considered more realistic synthesizer presets, which may contain a few dozens of parameters. The corresponding search space is so vast that heuristics, such as stochastic iterative search algorithms, became necessary. In particular, genetic algorithms [119] have been extensively used, as reported in Table 2.1. In the context of sound matching, the fitness function is often based on a combination of spectral and perceptual similarities between the output audio and the target (Figure 2.1). The chromosomes can be presets themselves, or intermediate representations. For sound matching using heuristic methods, one study [65] uses particle swarm optimization [120] rather than a genetic algorithm, where each particle is a preset of a subtractive virtual analog synthesizer.

Table 2.1. Sound matching studies based on genetic algorithms

Publication	Synthesis technique
Horner <i>et al.</i> , 1993 [60]	FM: single modulator and multiple carriers
Horner <i>et al.</i> , 1995 [121]	Additive
Horner, 1998 [61]	FM: nested modulators with feedback
Wun <i>et al.</i> , 2001 [122]	Wavetable
Garcia, 2002 [62]	Multiple topologies
Riionheimo <i>et al.</i> , 2003 [63]	Plucked strings
McDermott <i>et al.</i> , 2005 [123]	FM
Mitchell <i>et al.</i> , 2007 [57]; Mitchell, 2012 [58]	FM: one modulation level, three algorithms
Yee-King <i>et al.</i> , 2008, [64]	Subtractive and FM
Bozkurt <i>et al.</i> , 2011 [124]	SuperCollider [125], percussion and FM
Macret, 2013 [126]; Tatar <i>et al.</i> , 2016 [7]	OP-1 [127] multi-engine
Macret <i>et al.</i> , 2014 [128]	PureData [129]
Smith, 2017 [10]	Electric piano with distortion and phaser
Ye <i>et al.</i> , 2023 [130]	FM with arbitrary algorithm

As shown in Table 2.1, many different kinds of synthesis engines have been considered. Approximately 50% of the studies cited in this subsection focus on the challenging FM synthesizers. However, they have limitations in several

respects. First, all works but [7] consider rather small presets with less than 50 parameters [116, 131]. This is typically less than real-world synthesizers which often provide more than 100 different controls. Second, and more importantly, the huge computational requirements of these heuristics prevent their practical use [116, 131]. A study indicates search times that vary from seconds to hours [10] for simplified synthesizers with reduced sets of parameters, while another reports that the algorithm usually converges in a few minutes [64]. The most recent sound matching works [7, 65] deal with real-life VST synthesizers and have even larger requirements, such as clusters of computers. For instance, [7] is competitive with human sound designers but runs for 5 hours on a supercomputer with 50 CPU cores in order to match a 2s-long sound. Evolutionary algorithms cannot be sped up unless the synthesizer runs on a GPU [132], which is most often impossible with conventional synthesizers.

2.1.3 Data-driven techniques

Direct parameter estimation

In contrast to iterative search techniques listed in the previous subsection, works presented hereafter perform sound matching using models trained on datasets of presets. For instance, simple linear regressions can be used [133] to estimate the parameters of a VST synthesizer from acoustic features such as Root Mean Square (RMS) energy, spectral centroid or MFCCs. Experiments have been performed for sets of one to ten synthesis parameters, and the sound matching performance drops when more than five parameters are being considered.

Recent neural network architectures optimized by gradient descent can improve the results in both terms of speed and accuracy [131]. Unfortunately, conventional synthesizers are not differentiable and do not allow for gradient computation. Therefore, the direct optimization of an objective in the audio domain, such as illustrated in Fig. 2.1, is not possible. The typical setup for model-based sound matching is slightly different, and is presented in Fig. 2.2 where the following notations are used:

- \mathbf{u} denotes a preset from the dataset, which can be gathered from various sources or automatically generated;

- \mathbf{x} is the waveform synthesized using preset \mathbf{u} , or a representation of that waveform, e.g., a spectrogram, MFCCs and/or other acoustic features;
- $\hat{\mathbf{u}}$ and $\hat{\mathbf{x}}$ designate the matched preset and audio, respectively;
- $\mathcal{L}_{\text{preset}}(\hat{\mathbf{u}}, \mathbf{u})$ is a loss that quantifies the preset reconstruction error;
- $\mathcal{L}_{\text{audio}}(\hat{\mathbf{x}}, \mathbf{x})$ measures the audio reconstruction error.

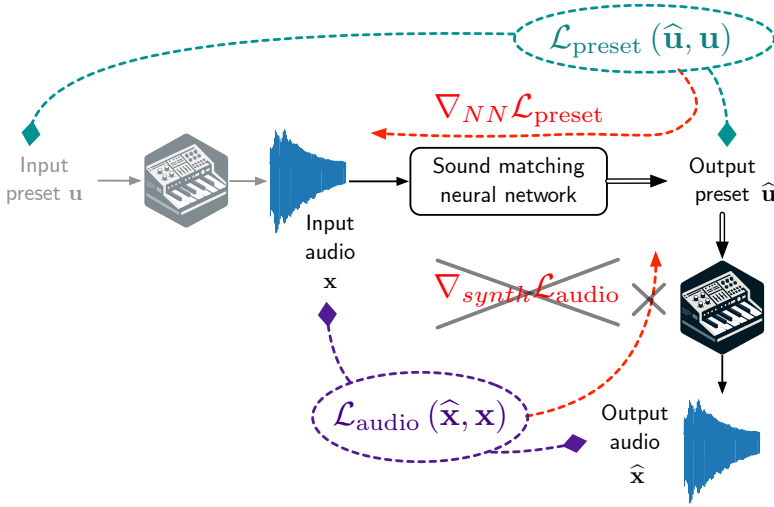


Figure 2.2. Model-based sound matching for a conventional, non-differentiable synthesizer. A preset loss can be backpropagated through the neural network, whereas an audio reconstruction loss cannot be backpropagated through the synthesizer.

Ideally, the sound matching objective should be the minimization of $\mathcal{L}_{\text{audio}}(\hat{\mathbf{x}}, \mathbf{x})$. However, the gradient of that loss with respect to the synthesis process, $\nabla_{\text{synth}} \mathcal{L}_{\text{audio}}$, cannot be computed for conventional synthesizers. Hence, models are only trained to minimize the synthesis parameter loss $\mathcal{L}_{\text{preset}}(\hat{\mathbf{u}}, \mathbf{u})$ by computing its gradient $\nabla_{NN} \mathcal{L}_{\text{preset}}$ with respect to the weights of the neural network. This simplified objective nonetheless poses a few issues [8, 134] which are discussed later in this subsection.

Three works [131, 135, 136] listed in the first section of Tab. 2.2 are trained using a $\mathcal{L}_{\text{preset}}$ loss only. They all focus on challenging FM synthesis processes and use different input modalities and neural network architectures.

For example, [131] processes time frames of 13-band input MFCCs through a Recurrent Neural Network (RNN) [67, Chapter 10]. In particular, a bi-directional Long Short-Term Memory (LSTM) [137] RNN navigates the sequence of MFCCs in both forward and reverse temporal directions. The hidden features are processed by a Multi-Layer Perceptron (MLP) [67, Chapter 6] which computes values for the 155 parameters of the DX7 synthesizer. The model has been evaluated by using the MFCC Distance (MFCCD), i.e., the RMS distance between target and reconstructed MFCCs, as \mathcal{L}_{audio} . It can be considered as a perceptual distance in timbre space [138]. Informal listening tests [131, 136] seem to indicate that the MFCCD threshold for perceived similarity is between 10 and 15. The model achieves an MFCCD smaller than 20 in 25% of the test scenarios for DX7 FM synthesizer programming, which slightly outperforms classical sound matching techniques based on heuristics. Nevertheless, computation durations are orders of magnitude faster.

The InverSynth [135] publication demonstrates that a model can be trained from a STFT using a 2D CNN. The synthesis process is based on custom fixed-architecture FM with four modulators and one carrier, and an output filter. The number of parameters is reduced to 23. InverSynth also shows that the model can be trained directly from raw audio using a 1D CNN pre-processing layer, although the performance is slightly impaired.

The Sound2Synth [136] pipeline leverages 32 models trained on 32 sub-datasets of DX7 FM synthesizer presets, where each dataset only contains presets with the same FM algorithm (Fig. 1.10). Each model is based on multiple parallel networks taking different inputs such as a spectrogram, a chromagram [139], MFCCs and other acoustic features. Each input modality is processed by an appropriate neural network, e.g., an LSTM for the time sequence of MFCCs, CNNs for spectrum-like inputs, or an MLP for acoustic features.

Some recent works, listed in the second section of Tab. 2.2, circumvent the optimization problem described in Fig. 2.2 by using differentiable synthesis processes. They are effectively able to optimize the \mathcal{L}_{preset} and \mathcal{L}_{audio} losses simultaneously using gradient descent. Models achieving the lowest \mathcal{L}_{preset} and the best parametric accuracy do not necessarily perform the best sound matching in terms of \mathcal{L}_{audio} loss [134, 140]. Therefore, using both losses for training is more appropriate.

For instance, [8] considers a differentiable 2-oscillator subtractive synthesizer with chorus¹ and reverb effects (19 parameters total). The sound matching network processes Mel-spectrogram inputs using a 1D-CNN on the frequency axis, then a Gated Recurrent Unit (GRU) [141] RNN on the time axis. The main limitation of [8] is nonetheless its much narrower scope, because it cannot be used with conventional non-differentiable synthesizers. Another model is [142] which focuses on two very specific differentiable synthesis techniques: an amplitude-modulated chirp generator, and a physical model of a rectangular membrane. The usual MSS loss has not been deemed appropriate for these synthesizers with highly non-stationary output signals. Instead, the audio loss relies on the joint time–frequency scattering transform [143], which better captures spectro-temporal modulations using wavelet convolutions and nonlinearities.

Other related works include SpiegeLib [144], a software library gathering various sound matching techniques based on genetic algorithms or neural networks. Some publications, e.g. [115, 116, 145], focus on parameter estimation for audio effects (e.g. distortion, compression, equalization, etc.) rather than synthesis.

Generative models

Sound matching models presented so far [8, 131, 133, 135, 136, 142] perform a unique task, i.e., the direct estimation of synthesis parameters from an input audio sample. Recently, neural network-based generative models have attracted significant attention for their ability to reconstruct, manipulate, and understand musical sounds. In particular, the VAE [146] framework, which will be formally introduced in Subsection 2.3.1, has been employed multiple times [134, 147, 148] for the sound matching task. The VAE seeks to reveal the underlying structure of data by learning an explicit latent space. It can be trained to model a small dataset of audio samples, for instance given as spectrograms. The latent space can be used for tasks such as generating new samples in the vicinity of an input, performing interpolations, or manipulating latent dimensions.

¹Based on summing the original signal and a slightly delayed copy of itself, which results in comb filtering.

Table 2.2. Data-driven sound matching models (non-generative)

Publication	Method	Target synth.
Itoyama and Okuno, 2014 [133]	Linear regression on acoustic features	Arbitrary VST synthesizers
Yee-King <i>et al.</i> , 2018 [131]	LSTM, MFCC inputs	DX7 FM
InverSynth: Barkan <i>et al.</i> , 2019 [135]	2D-CNN, STFT input	FM and subtractive
Sound2Synth: Chen <i>et al.</i> , 2022 [136]	Multiple features and spectra processed by parallel networks	DX7 FM
Masuda and Saito, 2023 [8]	1D-CNN and GRU, Mel-spectrogram input	Differentiable subtractive
Han <i>et al.</i> , 2024 [142]	2D-CNN, optimized perceptual loss [143]	Two specific diff. synths.

Encoded latent vectors serve as a high-level representation of the audio, which can also be decoded into presets in order to perform sound matching (Fig. 2.3). Compared to models which only optimize \mathcal{L}_{preset} (Fig. 2.2), those based on autoencoders (Fig. 2.3) seem to effectively capture the significance of synthesis parameters in the generated audio [134]. For FM synthesis in particular, it is known that a small adjustment in parameter space can lead to significant shifts in acoustic space [131, 140].

FlowSynth [134] has been the first sound matching work based on an audio VAE. Preset decoding is achieved using a class of invertible neural networks named *normalizing flows* [150] and described in Subsection 2.3.2. FlowSynth models subsets of continuous numerical parameters of Diva, an analog modeling synthesizer (Fig. 1.2(c)). It is mostly subtractive, with some FM capabilities. This study [134] compares some non-generative models to several VAE-based ones, and reports test measurements including errors on inferred

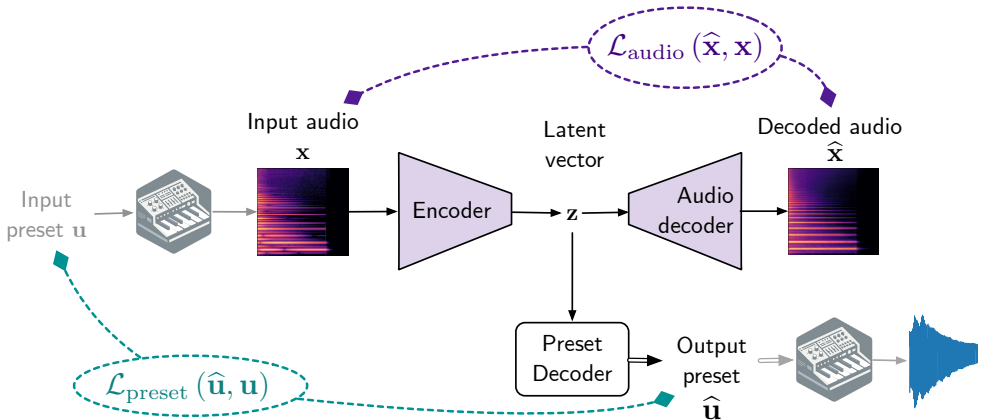


Figure 2.3. Audio autoencoder model with preset decoding from latent values

Table 2.3. Generative sound matching models

Publication	Model	Target synthesizer
FlowSynth: Esling <i>et al.</i> , 2020 [134]	Audio VAE and normalizing flows	Diva, 16, 32 or 64 parameters
PresetGenVAE: Le Vaillant <i>et al.</i> , 2021 [147]	Audio VAE and normalizing flows	DX7 FM, 144 parameters
MirrorNet: Siriwardena <i>et al.</i> , 2022 [149]	Audio autoencoder w/o preset decoder	Diva, 8 parameters
Faronbi <i>et al.</i> , 2023 [148]	Audio VAE, multiple preset decoders	Multiple: Serum, Diva, TyrellN6

presets \hat{u} and distances between target spectrograms \mathbf{x} and audio spectrograms synthesized from \hat{u} . First, VAE-based models improve the spectral error on synthesized audio, although error on inferred presets \hat{u} was smaller with non-generative baseline models. Second, comparative results about the different preset regression networks of VAE-based models are also presented. In terms of synthesized audio accuracy, results show that the usual MLP regression net-

works underperform compared to flow-based ones. Third, examples demonstrate how new presets can be generated from exploring the neighborhood of a latent vector.

However, there is room for improvement. For instance, the performance of FlowSynth decreases as the number of synthesizer parameters increases. Addressing this issue is crucial, as it limits the model’s ability to explore the full sonic range of a given synthesizer. PresetGenVAE [147] has a similar structure to FlowSynth, but is applied to DX7 FM synthesis and infers full presets without discarding some synthesis parameters. PresetGenVAE introduces separate modeling for numerical or categorical synthesis parameters, and take multiple input spectrograms in order to improve the matching. An extensive description and detailed results are provided in this chapter. A subsequent work [148] shows that a single audio VAE can be used to find presets for multiple synthesizers: Serum (wavetable), Diva and TyrellN6 (similar to, but simpler than Diva). Latent representations are shared by separate preset decoders, using one decoder for each synthesizer.

MirrorNet [149] belongs to a slightly different line of work, which is closer to melody reproduction and pitch tracking than to preset finding. It uses a vanilla autoencoder rather than a VAE, and latent vectors are used as presets, i.e., the preset decoder in Fig. 2.3 is the identity function. Each latent vector comprises eight Diva parameters (envelope, filter and vibrato controls), as well as note pitch and duration.

DX7 FM sound matching performance

All models introduced in sections 2.1.2 and 2.1.3 report sound matching performances for their own datasets and specific synthesizers. Models are also implemented in different languages, using different libraries, and some are closed-source or the source code is not available anymore. Therefore, it is very hard to objectively compare them all, and to indicate which work is the absolute best.

A significant amount of works have focused on the challenging DX7 FM synthesizer. The most recent of those, Sound2Synth [136], reports the performance in sound matching of several other works. Their results, reproduced in Table 2.4, were obtained from an undisclosed dataset of more than 100k

presets. Preset finding performance is objectively measured using the 13-band **MFCCD**, where the threshold for perceived similarity seems to be in the 10–15 range [131,136]. PresetGenVAE [147], the work that is discussed in the remainder of this chapter, represented the state of the art for DX7 **FM** synthesizer sound matching at the time of publication in 2021. It has been superseded by the Sound2Synth [136] proposal in 2022, which enables sound matching with an average **MFCCD** well below the similarity threshold.

Table 2.4. Performance of sound matching models for DX7 **FM** synthesis. **MFCCD** values extracted from [136], which considers that the threshold for perceived similarity is between 10 and 15.

Method	MFCCD (↓)	Generative
Hill Climbing [131]	22.0	✗
Genetic Algorithm [131]	31.3	✗
Bi-directional LSTM [131]	22.6	✗
PresetGenVAE [147]	14.7	✓
Sound2Synth [136]	5.4	✗

2.1.4 Related fields and commercial applications

As of 2024, very few commercial products integrate sound matching. Genopatch (published in October 2023) is part of the Synplant 2 **VST** synthesizer [151] which uses small set of 48 parameters. It is not open-source nor documented, but the sound matching procedure seems to rely on a genetic algorithm. The search can last for a few seconds or minutes, and convergence is not guaranteed. A commercial application [152] for the Vital **VST** wavetable synthesizer has been launched in June 2024. However, it claims to be "AI-powered" without further description, only works with a subset of Vital's synthesis parameters, and its performance is not reported.

Numerous *text-to-audio* solutions currently exist², capable of generating both sound effects and complete audio tracks corresponding from input prompts. However, the closely related field of *text-to-preset* remains relatively unexplored. Unlike direct audio generation, the presets generated through text-to-preset methods allow for more flexible tweaking. In particular, text-to-preset for non-differentiable synthesizers [153] has emerged as an active research area in 2024. Using pre-trained language and audio neural networks, and an evolutionary optimization procedure, [153] searches for the preset whose corresponding embedding best fits the embedding of an input text prompt. Experiments have been performed on a subtractive modular synthesizer with 78 parameters and running on a GPU, but the results are hard to evaluate [153]. While current audio results are not yet convincing, this approach shows potential for further exploration. Nevertheless, it lies beyond the scope of this thesis.

2.2 Dataset of DX7 presets

This section details the gathering of a dataset of DX7 presets required for training generative neural networks. It covers the processes involved in data collection, preset filtering, and automatic labeling. Dexed automatic programming has been studied in previous works [131, 144], which rely on randomly generated presets. The latter do not accurately represent musicians' use of a DX7 synthesizer, and have not been considered for building the PresetGenVAE model. A discussion of data augmentation strategies is nonetheless provided in Subsection 5.2.5 towards the end of this thesis.

2.2.1 Data collection and filtering

More than 12k DX7 public-domain *cartridges* have been gathered from various online sources³. Each cartridge, provided as MIDI SysEx (System Exclusive) data, holds 32 named presets, as well as its own name (Fig. 2.4). All cartridges were read using parts of Dexed source code, and their contents were saved to

²Such as <https://www.udio.com> or the open-source Stable Audio <https://stability.ai/stable-audio>

³For instance, sources include <https://yamahablackboxes.com/collection/yamaha-dx7-synthesizer/patches/> and https://bobbyblues.recup.ch/yamaha-dx7/dx7_patches.html (visited in 2024).

a temporary SQLite⁴ database. A large majority of the approximately 400k stored presets were duplicates, which were discarded. Some presets produced audio at a peak RMS volume lower than -40dB , and were also removed. The final dataset contains approximately 30k items.

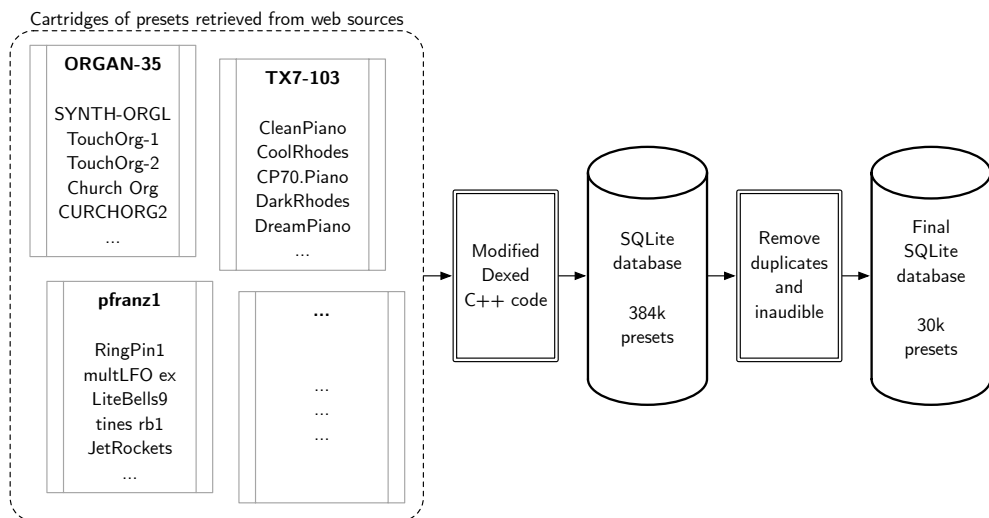


Figure 2.4. Presets gathering and filtering

Histograms of Dexed parameter values are reported in Appendix D. They show that the first four parameters, i.e., *Cutoff*, *Resonance*, *Output*, and *Master Tune Adjustment* are constant in the dataset. Similarly, the *OP Switch* controls of all operators are also always set to 1, i.e. all operators are always activated. In practice, any operator can also be deactivated by setting another parameter, *Output Level*, to zero. These ten fixed parameters correspond to features that were not available in the original DX7 but have been integrated into the Dexed VST, or to global synthesis parameters that were not stored in individual presets. They have been excluded for training. The *transpose* control has also been set to the middle C4 position. This further reduces the set of parameters to be modeled to 144 items.

⁴<https://www.sqlite.org>

2.2.2 Automatic labeling

This section introduces the methodology used for fast labeling of the dataset using simple algorithms that do not rely on models. The primary purpose is to provide an overview of the dataset’s composition, rather than to train classifiers or assess their accuracy.

Harmonic/percussive separation

For all presets, a single C4 MIDI note (pitch 60, velocity 100) has been played and held for 3.0s and audio has been recorded for 4.0s. A reduced 22050Hz sampling rate⁵ was used in order to lower CPU usage and the size of pre-rendered dataset files. A 100ms fadeout was applied at the end of audio files. In order to label presets, Harmonic-Percussive Source Separation (HPSS) [155] with a residual component [156] has been performed on all audio files. Samples with more than 40% harmonic or percussive spectral energy were labeled as harmonic or percussive, respectively. This 40% threshold seemed appropriate for most samples studied during an informal listening experiment. 27.4k presets were *harmonic* and 1.5k were *percussive*. 1.7k presets, which contained mostly HPSS residuals, had no descriptor at the end of this process. They were arbitrarily assigned a *sfx* label.

Three examples of HPSS decompositions are presented in Fig. 2.5. The first preset resembles an acoustic trumpet and shows a clear harmonic structure. The second is a hybrid conga/clave percussion. The third is a pulsating sound that is quite hard to describe with words, and is labeled *sfx*. Fig. 2.6 reports the histogram *harmonic*, *percussive*, and *sfx* labels. It shows that most sounds (over 90%) have a partially harmonic structure. Approximately 5% are labeled as *percussive* and 6% as *sfx*. Around 400 samples, such as tuned drums or bells, have both *harmonic* and *percussive* labels.

Some mislabeling can be observed in the dataset; for example, some hi-hat sounds are incorrectly labeled as *sfx*. The algorithm used is clearly not optimal. This subsection is intended to offer a rough idea of the dataset’s spectral contents, to better understand the sounds used for model training.

⁵This sampling rate is also the default with the *librosa* [154] Python package for music and audio analysis, which has been extensively used in this work.

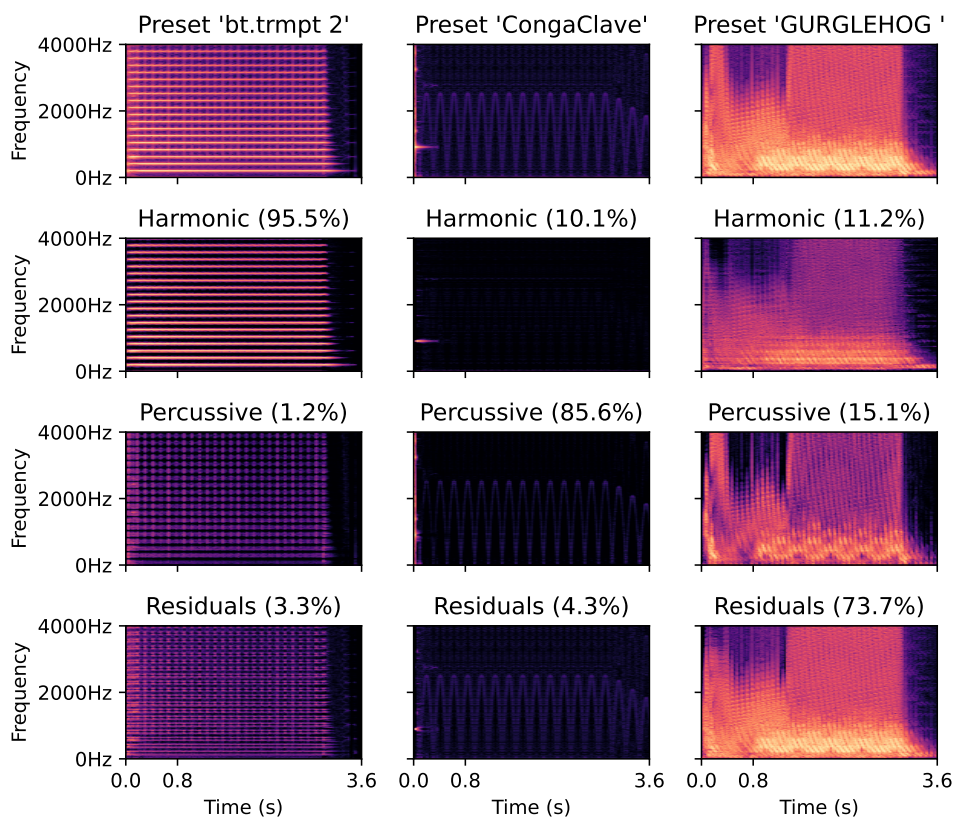


Figure 2.5. HPSS decomposition of three audio examples. Spectrograms have been cropped on both time and frequency axes.

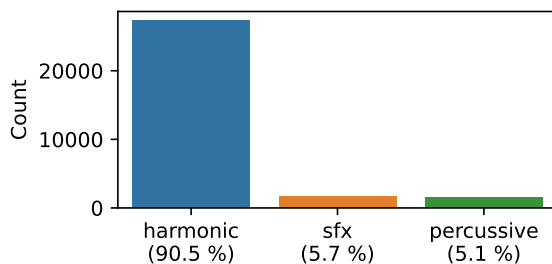


Figure 2.6. Histogram of *harmonic*, *percussive* and *sfx* labels across the dataset

Labels from character strings

Besides labels obtained from HPSS of audio files rendered from presets, another set of labels can be extracted from the presets' names, or the name of the original cartridge they belong to (Fig. 2.4). The list of candidate labels has been inspired by those of the NSynth dataset [73]. Additional labels, such as *harmonic_perc* and *sfx*, are used to better reflect the contents of this FM synthesis dataset. Automatic labeling can be performed by computing similarities between the names of the labels and those of the presets. For instance, the preset 'B/EPiano1Y' will be assigned the label *piano* because the 'piano' substring can be found. The same goes for the preset 'BRT RHODE1' because it was originally stored in a cartridge named 'PIANOS2'. The algorithm is case-insensitive, hence both are considered a perfect correspondence. In order to account for small variations, the edit distance [157] between labels and preset names can be computed. This distance, also called the Levenshtein distance, is the smallest number of single-character insertions, deletions, or substitutions needed to transform one word into another. If the label is strictly longer than four characters, it can match a preset name if the edit distance from a sub-string is 1 or less. For instance, 'ePian-26' will be labeled as *piano* because the edit distance from 'Pian-' to 'piano' is 1.

However, this method relies on a single string for each category and would miss a lot of correspondences. For instance, presets named 'violin' or 'cello' would not be classified into the *string* category, whereas they quite clearly should. A large majority of samples would actually remain unlabeled. Hence, most categories are described by multiple strings, and all of them are used to compute edit distances and perform matching with presets' names. These additional strings are the following:

- *piano*: steinway, wurlitzer, ...
- *string*: violin, cello, bowed, ...
- *brass*: horn, trumpet, saxophone, ...
- *sfx*: effect, drone, snd efx, chopper
- *harmonic_perc*: marimba, mallet, bell, xylo, ...
- *percussive*: perc, drum, snare, cymbal, hi hat, ...

- *organ*: hammond, leslie
- *guitar*: banjo, strat, ...
- *wind*: woodwind, clarinet, oboe, flute, ...
- *pluck*: pizz
- *vocal*: choir, female, male, human

The results are reported in Fig. 2.7. An informal check seems to indicate that the labeling is more often accurate than not, but errors can be found. First, the similarity measurement based on the Levenshtein edit distance is very basic; it can fail and raise false positives. Second, some presets or cartridges are clearly misnamed.

Unlabeled samples unfortunately represent 46.3% of the dataset and form the biggest category (not reported in Fig. 2.7). For instance, most presets from the 'pfranz1' cartridge in Fig. 2.4 cannot be matched with a label.

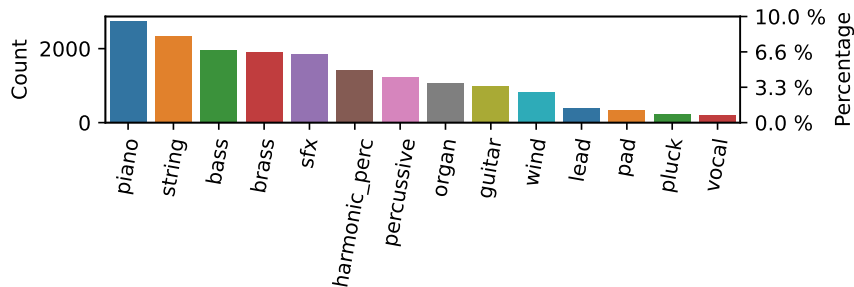


Figure 2.7. Histogram of labels based on presets' names

In general, these results seem coherent with those in Fig. 2.6. The dataset includes many primarily harmonic sounds, such as electric pianos, or string, brass and wind instruments. A significant amount of percussive presets, as well as special effects, can also be found. However, the labels are noisy and a large part of the dataset is unlabeled. Therefore, they should not be used for training classification models. The presets and labels are stored in a SQLite

database (26MB) available from the project’s Github repository⁶. Authors and sources of the original DX7 cartridges are also reported in this database.

2.3 PresetGenVAE model

2.3.1 Variational Autoencoder

The PresetGenVAE model introduced in this chapter is based on a generative autoencoder structure instead of a direct estimation of synthesis parameters from a target audio input. Autoencoders, in particular the VAE, are ubiquitous for creative applications and have been mentioned multiple times in this thesis. The VAE is formally introduced in this subsection.

Original formulation

VAEs [146] are deep probabilistic latent variable models that learn to map observed data $\mathbf{x} \in \mathbb{R}^{L_x}$ to a space of latent vectors $\mathbf{z} \in \mathbb{R}^{L_z}$ (Fig. 2.8). Observed items from the dataset are often vectors (e.g., 1D raw audio) or matrices (e.g., 2D spectrograms). In practice, the specific shape of the data is generally not crucial, and the vector notation \mathbf{x} is commonly employed irrespective of the actual shape.

Formally, a VAE aims at learning a joint probability distribution $p_\theta(\mathbf{x}, \mathbf{z})$ of the data \mathbf{x} and the underlying latent factors \mathbf{z} , where θ denotes the models’ parameters. This joint distribution can be expressed as $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$, and $p_\theta(\mathbf{x}|\mathbf{z})$ and $p_\theta(\mathbf{z})$ can be chosen arbitrarily. The latent prior $p_\theta(\mathbf{z})$ is typically constrained as $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}_{L_z})$ where \mathbf{I}_{L_z} is an identity covariance matrix in $\mathbb{R}^{L_z \times L_z}$.

On the decoder side (Fig. 2.8), the nature of distributions $p_\theta(\mathbf{x}|\mathbf{z})$ is chosen according to the type of data being modeled. For instance, for audio spectrograms modeling, $p_\theta(\mathbf{x}|\mathbf{z})$ is usually a free-mean, fixed-variance multivariate Gaussian distribution [87]. The distributions’ means, denoted $\mu_{\mathbf{x}}$, are com-

⁶First version: <https://github.com/gwendal-lv/preset-gen-vae/tree/main/synth>; updated version: <https://github.com/gwendal-lv/spinvae2/tree/main/synth>

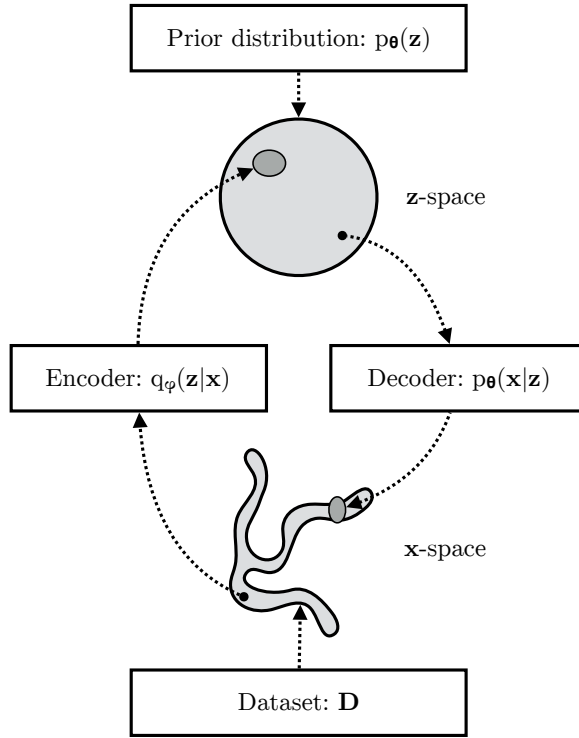


Figure 2.8. Overview of the VAE model. It learns to map an arbitrarily distributed dataset into a latent space with a predefined shape, and to decode latent samples \mathbf{z} back into the dataset’s space. Figure from [13].

puted by the decoder neural network. In practice, the typical decoder is quite straightforward and very similar to a deterministic autoencoder:

$$\begin{cases} \mu_{\mathbf{x}} = \text{DecoderNeuralNetwork}(\mathbf{z}; \theta) , \\ p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mu_{\mathbf{x}}, \mathbf{I}_{L_X}) . \end{cases} \quad (2.1)$$

Given a data point \mathbf{x} and a corresponding estimated latent code \mathbf{z} , the probability $p_{\theta}(\mathbf{x}|\mathbf{z})$ is easy to compute and the θ parameters can be optimized using Stochastic Gradient Descent (SGD).

On the encoder side, however, optimizing the model presents more complexity. The latent posterior density $p_\theta(\mathbf{z}|\mathbf{x})$ could be written as follows:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}, \quad (2.2)$$

where $p_\theta(\mathbf{x})$ is the marginal likelihood of the data point \mathbf{x} under our model parameterized by θ . Unfortunately, $p_\theta(\mathbf{x})$ requires a computationally intractable integration [13] over all possible latent values $\mathbf{z}' \in \mathbb{R}^{Lz}$:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z}')p_\theta(\mathbf{z}')d\mathbf{z}'. \quad (2.3)$$

Therefore, $p_\theta(\mathbf{z}|\mathbf{x})$ from Eq. 2.2 cannot be computed. Instead, the VAE formulation [146] suggests to use an approximate $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$, which is called a *variational* approximation. A common choice for $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian distribution with a diagonal covariance matrix:

$$\begin{cases} \mu, \sigma^2 = \text{EncoderNeuralNetwork}(\mathbf{x}; \phi), \\ q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2), \end{cases} \quad (2.4)$$

where $\mu, \sigma \in \mathbb{R}^{Lz}$, and ϕ designates the parameters (e.g., weights and biases) of the encoder neural network. σ denotes the diagonal coefficients are the normal distribution's covariance matrix. Given a data point \mathbf{x} , this approximate latent distribution $q_\phi(\mathbf{z}|\mathbf{x})$ has a closed-form expression and is easy to compute.

Ideally, the whole model should be trained to maximize the marginal likelihood $p_\theta(\mathbf{x})$, but it is intractable. Instead, the following objective (called the evidence lower bound) can be maximized [146]:

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) = \underbrace{\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction accuracy}} - \underbrace{D_{KL} [q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})]}_{\text{Regularization term}}, \quad (2.5)$$

with $\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x})$ [146]. Hence, maximizing $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ increases $p_\theta(\mathbf{x})$, i.e., the dataset is theoretically better modeled by the VAE. The first term in Eq. 2.5 can be considered as a reconstruction accuracy because it measures how log-probable an input \mathbf{x} is under the decoder output distribution. For training a VAE model in practice, this expectation is computed using a one-sample Monte-Carlo approximation [146], i.e., by drawing a single sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ from the encoder output distribution and providing that sample \mathbf{z} to the decoder.

The second term in Eq. 2.5 is a Kullback-Leibler divergence (D_{KL}), which measures how $q_\phi(\mathbf{z}|\mathbf{x})$ is different from the latent prior $p_\theta(\mathbf{z})$. It is non-negative and has a zero value if the two distributions are equal. Maximizing $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ should minimize the D_{KL} term, thus encouraging encoded probability distributions to remain close to the prior $p_\theta(\mathbf{z})$. The divergence is defined as follows:

$$D_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})] = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} d\mathbf{z} = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} \right]. \quad (2.6)$$

Although it may be difficult to compute in general, the particular Kullback-Leibler divergence between two multivariate normal distributions has a direct closed-form expression. Specifically, when $q_\phi(\mathbf{z}|\mathbf{x})$ has a diagonal covariance matrix and $p_\theta(\mathbf{z})$ is the standard normal distribution, the formula is:

$$D_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})] = \frac{1}{2} \left(\|\mu\|^2 + \sum_{i=1}^{L_Z} (\sigma_i^2 - \log \sigma_i^2) - L_Z \right), \quad (2.7)$$

where μ, σ parameterize the $q_\phi(\mathbf{z}|\mathbf{x})$ distribution (Eq. 2.4) and σ_i designates the i^{th} coordinate of the $\sigma \in \mathbb{R}^{L_Z}$ vector. This term has strong implications in shaping the latent space, and is further discussed in the following paragraphs.

Latent space properties

A VAE is trained to maximize the composite two-term objective from Eq. 2.5. When encoding inputs into a continuous latent space, the additional regularization term is typically computed as shown in Eq. 2.7 and should be minimized when optimizing the training objective. Hence, $\|\mu\|$ should also be minimal, i.e., $q_\phi(\mathbf{z}|\mathbf{x})$ tends to remain centered around the origin. Regarding the set of diagonal coefficients σ of the covariance matrix, the $\sigma_i \mapsto \sigma_i^2 - \log \sigma_i^2$ function reaches its minimal value 1 for $\sigma_i = 1$. Therefore, minimizing Eq. 2.7 forces the vector of standard deviations σ to remain close to 1_{L_Z} . In contrast, a standard autoencoder only learns to reconstruct some inputs. Its encoding process is deterministic, which would correspond to degenerate $q_\phi(\mathbf{z}|\mathbf{x})$ distributions of a VAE with $\sigma \rightarrow 0$. In the VAE framework, such Dirac-like probability densities are prevented by the D_{KL} term in Eq. 2.7.

When training a VAE model, a latent code \mathbf{z} must be sampled from the encoded distribution $q_\phi(\mathbf{z}|\mathbf{x})$ being provided to the decoder (reconstruction term

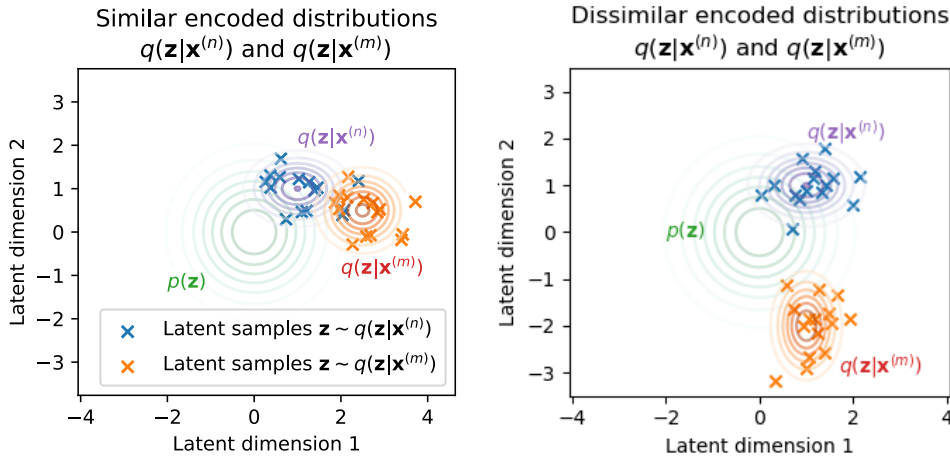


Figure 2.9. Visualizations of posterior distributions $q_\phi(\mathbf{z}|\mathbf{x})$ and latent prior $p_\theta(\mathbf{z})$ in a 2D latent space. Variances of $q_\phi(\mathbf{z}|\mathbf{x})$ densities are greatly exaggerated for visualization purposes and are often much lower in practice (e.g., < 0.1 or 0.01).

in Eq. 2.5). Two very dissimilar inputs $\mathbf{x}^{(n)}, \mathbf{x}^{(m)}$ are likely to be encoded into latent distributions $q_\phi(\mathbf{z}|\mathbf{x}^{(n)})$ and $q_\phi(\mathbf{z}|\mathbf{x}^{(m)})$ with significantly different means (Fig. 2.9, right), because the loss in Eq. 2.7 imposes large variances (close to 1) for these Gaussian distributions. Sampled vectors $\mathbf{z}^{(n)}, \mathbf{z}^{(m)}$ need to be distant enough for the decoder to easily differentiate between them and to approximately reconstruct the $\mathbf{x}^{(n)}, \mathbf{x}^{(m)}$ inputs.

If the two inputs $\mathbf{x}^{(n)}, \mathbf{x}^{(m)}$ are similar, an ideal VAE would learn to encode them into $q_\phi(\mathbf{z}|\mathbf{x}^{(n)}), q_\phi(\mathbf{z}|\mathbf{x}^{(m)})$ densities with similar joint support (Fig. 2.9, left). The decoder may not be able to discriminate between the two sampled latent codes $\mathbf{z}^{(n)}, \mathbf{z}^{(m)}$, and output distributions $p_\theta(\mathbf{x}|\mathbf{z}^{(n)}), p_\theta(\mathbf{x}|\mathbf{z}^{(m)})$ may be similar. Nevertheless, considering that $\mathbf{x}^{(n)}, \mathbf{x}^{(m)}$ are indeed similar, this would not lead to a sharp decrease in the training objective from Eq. 2.5 [158]. Therefore, the latent space should be continuous, i.e., small steps in the latent space should correspond to small variations in the original data space.

Ideally, the latent space is also compact near the origin, without *holes* which can be defined as low-density latent regions with $q_\phi(\mathbf{z}|\mathbf{x}) \approx 0$ for any \mathbf{x} in the dataset [159]. The aggregate (or marginal) posterior $q_\phi(\mathbf{z})$ can be defined and estimated as follows:

$$q_\phi(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim q(\mathbf{x})} [q_\phi(\mathbf{z}|\mathbf{x})] \approx \frac{1}{N} \sum_{n=1}^N q_\phi(\mathbf{z}|\mathbf{x}^{(n)}), \quad (2.8)$$

where $q(\mathbf{x})$ is the empirical distribution of the N data points in the training set, and the approximation is a Monte-Carlo estimate. This aggregate posterior $q_\phi(\mathbf{z})$, which represents the distribution of latent values provided to the decoder during training, should remain similar to the prior $p_\theta(\mathbf{z})$. The absence of holes guarantees that the decoder is trained to generate meaningful outputs for latent codes that are close to the origin.

Such an ideal latent space enables applications such as:

- sampling latent vectors from $q_\phi(\mathbf{z}|\mathbf{x})$ or $p_\theta(\mathbf{z})$ then generating new data using the decoder;
- interpolating between latent codes, then generating intermediate points in the \mathbf{x} space;

- searching for similar samples in a dataset by measuring distances between latent codes;
- using $q_\phi(\mathbf{z}|\mathbf{x})$ to identify outliers in the original data space;
- manipulating high-level attributes of \mathbf{x} , if the latent dimensions are *disentangled* [158].

β -VAE

The VAE objective from Eq. 2.5 has been proven to favor fitting the observations \mathbf{x} over performing correct \mathbf{z} inference [160]. An important issue is that the reconstruction term (Eq. 2.5) is often much larger than the regularization D_{KL} term. If the amount of regularization becomes negligible, it cannot prevent encoded probability densities from having a disjoint support. For instance, if $\sigma^2 \rightarrow 0_{L_Z}$, then the $q_\phi(\mathbf{z}|\mathbf{x})$ densities resemble Dirac distributions. This can lead to undesirable discontinuities and holes in the latent space. This phenomenon is further amplified by the usually very large difference between the number of dimensions of the \mathbf{x} and \mathbf{z} spaces, typically $L_X \gg L_Z$.

For instance, consider that \mathbf{x} is a 256×256 natural image or a spectrogram, and that \mathbf{z} is a 16-dimensional vector. Assuming Gaussian latent distributions, $D_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]$ is actually a sum of $L_Z = 16$ terms. If the pixels of \mathbf{x} are modeled by independent Gaussian distributions, i.e., $p_\theta(\mathbf{x}|\mathbf{z})$ has a diagonal covariance matrix and can be expressed as a product of 1D Gaussian densities, then $\log p_\theta(\mathbf{x}|\mathbf{z})$ is a sum of $L_X = 65536$ terms.

To prevent these issues, the β -VAE proposal [158] applies a $\beta > 1$ factor to the regularization term. The training objective, to be maximized, becomes:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \underbrace{\log p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Reconstruction}} - \beta \underbrace{D_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]}_{\text{Regularization}}. \quad (2.9)$$

The β hyper-parameter also leads to a lower entanglement of latent variables, at the expense of blurrier reconstructions [158]. It must nonetheless be tuned when training the encoder and decoder neural networks, which represents some extra work.

2.3.2 Normalizing Flows

Bijjective transformations of probability distributions

Normalizing Flows [150] are invertible neural networks designed primarily to transform probability distributions. For instance, they can map some samples from a dataset, whose probability distribution is unknown, to latent values whose probability can be easily estimated from a closed-form density. An example of a flow transformation, denoted T and applied to a toy dataset, is given in Fig. 2.10. In this figure, an arbitrary dataset of 2D points can be mapped to a Gaussian latent space.

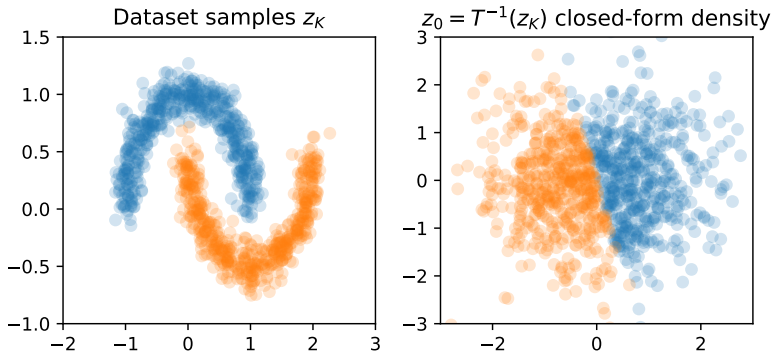


Figure 2.10. Invertible mapping of samples \mathbf{z}_K into latent values \mathbf{z}_0 with $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0; \mathbf{0}, \mathbf{I})$. \mathbf{z}_K points are artificial samples from a 2D dataset with two interleaving clusters (blue and orange).

Flow-based models are usually made of multiple layers of invertible transformations T_1, \dots, T_K . Considering a given layer T_k with $k \in [1, K]$, let $\mathbf{z}_{k-1}, \mathbf{z}_k$ denote random vectors and $T_k : \mathbf{z}_{k-1} \mapsto T_k(\mathbf{z}_{k-1}) = \mathbf{z}_k$ denote an invertible and differentiable transform whose inverse is also differentiable. Given $p(\mathbf{z}_{k-1})$, the probability density of \mathbf{z}_k can be obtained by a change of variables:

$$p(\mathbf{z}_k) = p(\mathbf{z}_{k-1}) |\det J_{T_k}(\mathbf{z}_{k-1})|^{-1}, \quad (2.10)$$

where J_{T_k} is the Jacobian matrix of the flow transform T_k , whose determinant should be easy and efficient to compute. In practice, most flow transforms [161–164] have lower- or upper-triangular Jacobian matrices, hence the determinant is simply the product of diagonal terms. Implementations ensure non-zero diagonal coefficients for numerical stability.

If $T = T_K \circ \dots \circ T_1$ is the full multi-layer flow model, then the log probability of a given sample \mathbf{z}_K can be directly computed as:

$$\log p(\mathbf{z}_K) = \log p(\mathbf{z}_0) - \sum_{k=1}^K \log |\det J_{T_k}(\mathbf{z}_{k-1})|, \quad (2.11)$$

where $p(\mathbf{z}_0)$ is the arbitrarily chosen *base* distribution of the flow, such as a standard normal distribution. After training, any data point \mathbf{z}_K can be mapped to a latent vector \mathbf{z}_0 and its probability can be computed using Eq. 2.11. Latent vectors \mathbf{z}_0 can also be sampled or manipulated in order to generate new data.

Normalizing flows are conceptually quite similar to the more recent diffusion models [165]. Both classes of models can progressively transform data into latent noise, or latent noise into data. Both require that dataset and latent samples have the same dimensionality, in contrast to VAEs or GANs, which often rely on a smaller latent space. Diffusion models are known to be able to generate highly realistic samples. However, in contrast to normalizing flows or VAEs, they don't provide an explicit tractable probability density function.

Latent space normalizing flows

Normalizing flows have been introduced for modeling a dataset, but they can also perform transformations on the latent vectors of a VAE (Fig. 2.11). The baseline VAE approximates the true intractable posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ with a simplified Gaussian density $q_\phi(\mathbf{z}|\mathbf{x})$. Although it can lead to good reconstruction results, this posterior approximation can be improved by using a more flexible model [150], e.g., the output of a sequence of normalizing flows T_1, \dots, T_K .

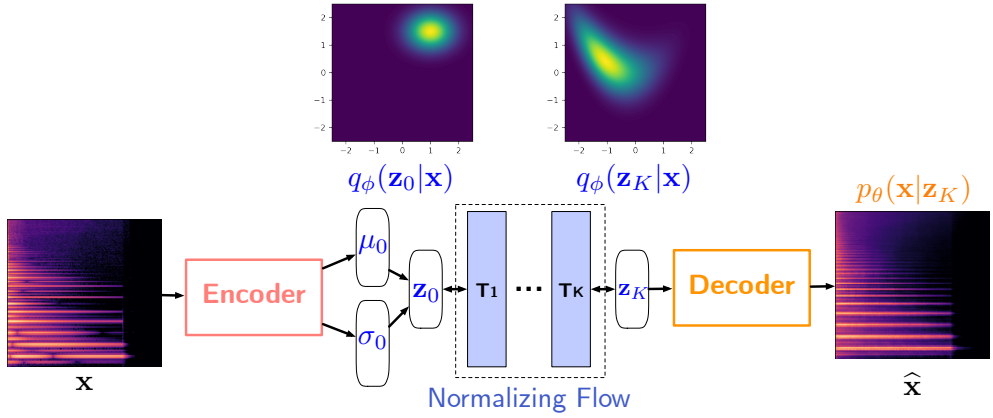


Figure 2.11. Audio spectrograms \mathbf{x} modeled by a VAE with a latent normalizing flow. In this example, the encoder output distribution $q_\phi(\mathbf{z}_0|\mathbf{x})$ is Gaussian whereas $q_\phi(\mathbf{z}_K|\mathbf{x})$ is more expressive but does not have a closed-form expression.

The encoder's output distribution remains Gaussian (as in Eq. 2.4) and is now denoted $q_\phi(\mathbf{z}_0|\mathbf{x})$. This $q_\phi(\mathbf{z}_0|\mathbf{x})$ becomes the flow's base distribution, whereas $q_\phi(\mathbf{z}_K|\mathbf{x})$ is now considered as the new latent posterior of the VAE. This $q_\phi(\mathbf{z}_K|\mathbf{x})$ is non-Gaussian, but it can be computed from Eq. 2.11 by using conditional probabilities $q_\phi(\bullet|\mathbf{x})$ instead of marginal probabilities $p(\bullet)$. However, when using the non-Gaussian $q_\phi(\mathbf{z}_K|\mathbf{x})$, the D_{KL} regularization term from the VAE objective (Eq. 2.5) no longer has a closed-form expression. This D_{KL} term can be estimated through a one-sample Monte-Carlo computation of the expectation from Eq. 2.6. Merging Equations 2.5, 2.6 and 2.11, a VAE with latent normalizing can be trained by maximizing the following objective:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_0 \sim q_\phi(\mathbf{z}_0|\mathbf{x})} \left[\underbrace{\log p_\theta(\mathbf{x}|\mathbf{z}_K)}_{=-\mathcal{L}_{Reconstruction}} + \beta \underbrace{\left(\log p_\theta(\mathbf{z}_K) - \log q_\phi(\mathbf{z}_0|\mathbf{x}) + \sum_{k=1}^K \log |\det J_{T_k}(\mathbf{z}_{k-1})| \right)}_{=-\mathcal{L}_{Latent}} \right], \quad (2.12)$$

where θ designates the neural network weights of both the encoder and normalizing flows. The remaining expectation is also typically simplified as a one-sample Monte-Carlo estimate. In the objective from Eq. 2.12, to be maximized, the corresponding training losses to be minimized are $\mathcal{L}_{Reconstruction}$ and \mathcal{L}_{Latent} .

Flow models

Two common flow models are the Masked Auto-regressive Flow (**MAF**) [163] and Real-valued Non-Volume Preserving flow (**RealNVP**) [164]. On one hand, a single **MAF** layer is fully auto-regressive, meaning that it introduces dependencies between all components of an output vector, and can be considered a universal approximator [150]. Thus, a multi-layer **MAF**-based model has a large modeling power. However, an **MAF** inverse computation is sequential and orders of magnitude slower than a forward computation.

On the other hand, a single **RealNVP** layer is a *coupling layer* displayed in Fig. 2.12. Half of the outputs (\mathbf{y}_1) are set equal to the first half of the inputs (\mathbf{x}_1), while an affine transformation is applied to the other half of inputs (\mathbf{x}_2). The scale and offset coefficients are calculated by functions f and g of \mathbf{x}_1 , typically implemented as neural networks. The transformation $\mathbf{y}_2 = \mathbf{s} \odot \mathbf{x}_2 + \mathbf{t}$ can be easily inverted knowing \mathbf{y} only, because \mathbf{s} and \mathbf{t} can be computed from \mathbf{y}_1 instead of \mathbf{x}_1 . Moreover, its Jacobian is a diagonal matrix, and its diagonal is made of ones and the $\mathbf{s} = f(\mathbf{x}_1)$ vector. Therefore, its determinant (required in Eq. 2.12) is simply the product of all elements of \mathbf{s} .

Even though this affine transformation is quite simple, a sequence of multiple **RealNVP** layers and permutations forms a fully auto-regressive model. Com-

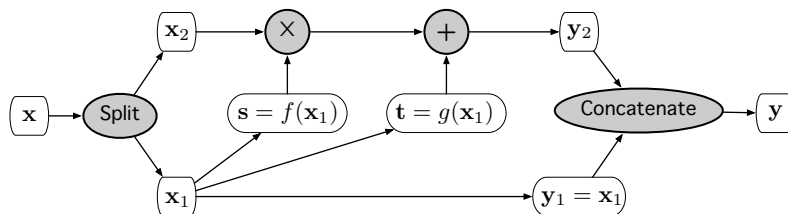


Figure 2.12. Computational graph for a forward [RealNVP](#) affine coupling layer. \mathbf{x} and \mathbf{y} are the input and output vectors, respectively, and sub-vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ all have the same size.

pared to fully auto-regressive flows such as [MAF](#), a [RealNVP](#)-based model provides more scalability but is less parameter-efficient. Moreover, the forward and inverse directions require the same amount of computation and are equally numerically stable [164].

2.3.3 General model architecture

Similar works

As previously indicated, the [PresetGenVAE](#) model (Fig. 2.13) is inspired by the [FlowSynth](#) proposal [134], based on a spectral [VAE](#) with a latent normalizing flow. Instead of using a single-path neural network (e.g., [MLP](#), [CNN](#)) from audio spectrograms \mathbf{x} to synthesizer presets \mathbf{u} , [FlowSynth](#) indirectly extracts presets from latent vectors of a [VAE](#). This approach assumes that the latent space holds enough meaningful audio information, which should be true if the [VAE](#) learns to compress spectrograms with tens of thousands of pixels into latent vectors with a few hundreds coordinates.

The [FlowSynth](#) study compares some feedforward models to several [VAE](#)-based models on limited subsets of numerical parameters (16, 32 and 64) of the [Diva VST](#) sound synthesizer. The related dataset contains 11k presets which were rendered into audio files to compute input spectrograms \mathbf{x} . Reported test measurements include errors on decoded presets $\hat{\mathbf{u}}$ and distances between target spectrograms \mathbf{x} and audio spectrograms synthesized from $\hat{\mathbf{u}}$. First, results demonstrate that their [VAE](#)-based models improve the spectral error on synthesized audio, although the error on decoded presets $\hat{\mathbf{u}}$ is smaller

with feedforward baseline models. Second, comparative results about various preset regression networks of VAE-based models are presented. In terms of synthesized audio accuracy, the usual MLP regression networks underperform compared to flow-based ones. Third, examples show how new presets can be generated from exploring the neighborhood of a latent vector. Moreover, thanks to the invertibility of the regression flow, any preset could be easily encoded into the audio latent space.

There is nevertheless room for improvement. As their authors admit, the performance of the FlowSynth model decreases as the number of synthesizer parameters increases. This issue must be addressed because it prevents the model from using all sonic possibilities of a given synthesizer. Regarding audio reconstruction and presets generation, reducing inference errors would generally improve the model. Moreover, the FlowSynth study focuses on a single mostly-subtractive synthesizer, so additional tests would be beneficial. Categorical parameters, which are excluded from the aforementioned experiments, constitute a large part of synthesizer controls thus must also be considered.

CNNs and normalizing flows

The autoencoder processes Mel-spectrograms \mathbf{x} and the convolutional encoder and decoder are symmetrical. Details about the number of channels and size of kernels are provided in Fig. 2.13. In order to reduce the high computational cost of convolutions, stride is (2, 2) for layers whose kernel is larger than one.

Each Mel-spectrogram \mathbf{x} is computed from an audio file corresponding to a preset \mathbf{u} . Audio files are 4s-long and sampled at 22050Hz, as described in Subsection 2.2.2. These files are transformed into 257-bins Mel-Spectrograms computed from the log-magnitude of a STFT (Hann window, width 1024, hop size 256, 347 time frames). Then, a -120 dB threshold is applied to these spectrograms, which are finally scaled into $[-1, 1]$ using the minimum and maximum amplitudes of the training dataset.

Considering the very promising results of FlowSynth [134] on subsets of up to 64 synthesis parameters, a flow transform is used to compute presets \mathbf{u} from latent vectors \mathbf{z} . However, this structure constraints the dimension L_Z of latent vectors $\mathbf{z}_0, \dots, \mathbf{z}_K$ to be that of the synthesizer parameters vector \mathbf{u} . FlowSynth [134] used a variant of MAF [162] as normalizing flows. However,

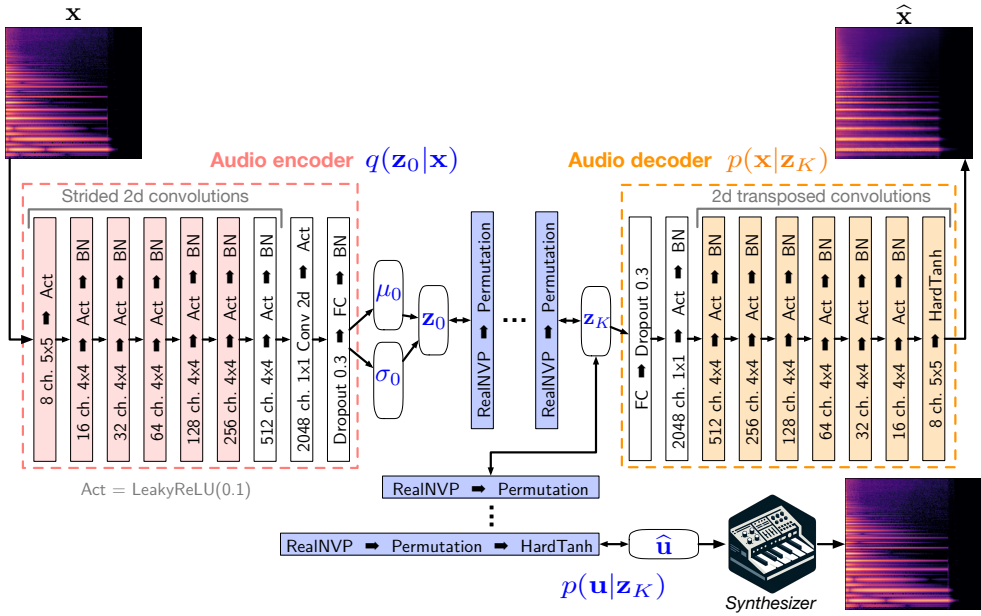


Figure 2.13. Block diagram of the single-channel spectrogram PresetGenVAE model (general architecture inspired from [134]). The number of channels of encoding (resp. decoding) convolutional layers denote the number of output (resp. input) channels.

MAF could not be used for complete DX7 presets with 144 parameters. When training PresetGenVAE models, **MAF** layers were not numerically stable and **SGD** often did not converge with large latent spaces with $L_Z > 100$. Thus, the more stable and scalable **RealNVP** layers have been preferred, with 6 latent layers and 6 layers for computing the preset. Internal scale and translation coefficients (Fig. 2.12) of each **RealNVP** layer are computed using a 2-layer **MLP** (300 hidden units) with residual connection, Batch Normalization (**BN**) [166] and 0.4 dropout probability. A *hardtanh* activation with $[0, 1]$ range is applied to the regression flow’s output. Permutations of vectors coordinates need to be performed in-between **RealNVP** layers, otherwise half of the latent coordinates would remain unchanged (\mathbf{x}_1 of Fig. 2.12 would hold the same values in all layers). These permutations are randomly initialized at model creation then kept constant.

2.3.4 Output preset modeling

Numerical and categorical parameters

Previous works have used different strategies to model various types of synthesis controls. On the one hand, some studies such as [131, 134] consider that all parameters are numerical and continuous. Whereas this assumption is appropriate for synthesis parameters such as a F_0 or a cut-off frequency, it is not for DX7 parameters such as the discrete waveform selector or the FM algorithm. However, [131] indiscriminately models all DX7 parameters as numerical, as displayed in Fig. 2.14(a).

When a model is trained on continuous numerical data, the corresponding loss for reconstruction is most often the MSE loss. Minimizing a MSE is equivalent to maximizing the log-probability of the data under a unit-variance multivariate Gaussian distribution parameterized by the model [67, Chap. 5]. On the one hand, training with this MSE loss only induces an inadequate ordering of categorical variables. It also assumes that proximity in value implies similarity in meaning, which is often incorrect. For instance, DX7 algorithms 18 and 19 represent entirely different routing of oscillators (Fig. B.1) and sound completely different.

On the other hand, some other studies [135, 136] could improve the sound matching performance by turning the parameter estimation problem into a classification task. This nonetheless requires a quantization of all numerical

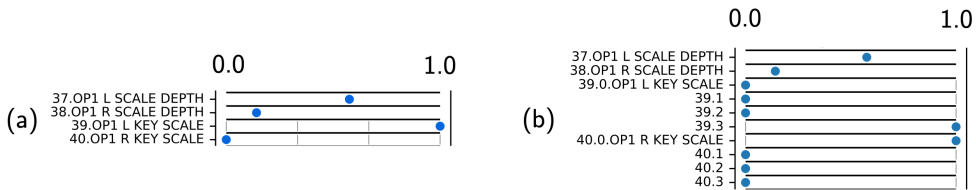


Figure 2.14. Two representations of a subset of four parameters from a preset. (a) VST representation: all parameters are stored as numerical although parameters 39 and 40 contain categorical data with four possible options. (b) More appropriate representation: parameters 37, 38 are left unchanged while 39, 40 are one-hot encoded.

parameters, where each value corresponds to a class. For training such models, each input value is transformed into a sequence of zeros and a single one value, as depicted in Fig. 2.14(b). This is often referred to as *one-hot encoding*. The corresponding loss is usually the categorical cross-entropy [67, Chap. 5] between the model’s output probabilities and the one-hot encoded targets. Unfortunately, this process ignores all ordering and interval information of numerical values.

While the one-hot encoding of categorical variables seems natural, it is not fully clear how the modeling of numerical parameters influence the automatic programming of a DX7 synthesizer. Experiments presented in the following pages will address this question.

Final training objective

The PresetGenVAE model (Fig. 2.13) is mostly based on a VAE with latent normalizing flow, whose loss is given in Eq. 2.12. The second normalizing flow is used as a powerful auto-regressive model to infer presets from latent codes. It is simply used as a second decoder in the autoencoder structure.

In contrast to conventional minimization of a loss function when training neural networks, the classical VAE formulation introduces an objective to be maximized. It can be converted to a cost function by applying a minus sign. Moreover, an extra loss term, $\mathcal{L}_{Preset}(\mathbf{u})$, encourages the model to retrieve presets \mathbf{u} from latent codes \mathbf{z}_K . The loss for training the PresetGenVAE model finally becomes:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = \mathcal{L}_{Audio}(\mathbf{x}) + \mathcal{L}_{Preset}(\mathbf{u}) + \beta \mathcal{L}_{Latent}(\mathbf{z}_0, \dots, \mathbf{z}_K), \quad (2.13)$$

where L_{Latent} is the regularization loss for a VAE with latent normalizing flow, described in Eq. 2.12, and model parameters θ, ϕ are omitted for the sake of readability. $\mathcal{L}_{Audio}(\mathbf{x})$ is the MSE between the spectral decoder output means and target \mathbf{x} spectrograms. Regarding $\mathcal{L}_{Preset}(\mathbf{u})$, a MSE loss is evaluated on each numerical parameter, while a categorical cross-entropy loss is computed on each categorical sub-vector. Nonetheless, early experiments have shown that categorical representations of discrete numerical parameters lead to higher

losses for a given accuracy. Thus, an empirical 0.2 factor has been applied to all categorical losses. The total preset loss is the sum of per-parameter losses.

In order to perform fair comparisons between variants of the models, all three losses from Eq. 2.13 are normalized, i.e., divided by the dimension of the corresponding data. This prevents the model from prioritizing, for instance, latent regularization over reconstruction accuracy when the latent size L_Z increases. Such a normalization implies a $\beta \approx L_X/L_Z$ regularization factor, where $L_X = 347 \times 257$ is the number of spectrogram pixels. In tests presented hereafter, this β ranges approximately from 150 to 1000. The normalized latent loss is multiplied by an arbitrary 0.2 factor in order to improve the tradeoff between amount of regularization and reconstruction accuracy. Resulting β values are similar to those of Higgins *et al.* [167] for 178×218 image modeling tasks. These authors also report that this hyper-parameter β is hard to tune, even on a logarithmic scale.

2.3.5 Implementation and training

The model described in Fig. 2.13 has been implemented and trained in Python using PyTorch [168], and using an external third-party library⁷ for normalizing flows. Regarding audio synthesis, VST plugins such as Dexed cannot be directly employed from Python code to render audio. The RenderMan⁸ wrapper addresses this issue. It is written in C++ and provides a Python interface to load a VST plugin, program a preset into it, play some MIDI notes and record the output audio.

Generating a 4s-long audio sample with RenderMan and the Dexed VST requires a few dozen milliseconds. Although rendering the dataset can be parallelized using multiple Python processes across multiple CPU cores, computing audio on the fly during model training is too slow and would significantly hinder the efficiency of the process. As a result, it is necessary to generate the audio files and Mel-spectrograms in advance and to store them (Fig. 2.15), allowing the model to access them directly during training. The optional one-hot encoding of parts of presets is also precomputed. It requires the description

⁷<https://github.com/bayesiains/nflows>

⁸<https://github.com/fedden/RenderMan>

of synthesis parameters and their nature (e.g., Table B.1 for Dexed) to be manually encoded.

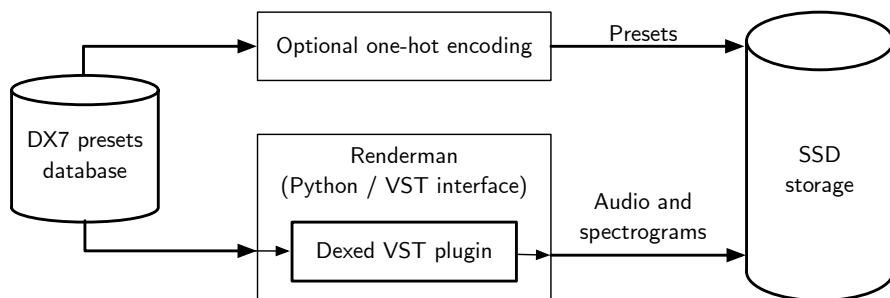


Figure 2.15. Precomputations of audio files, spectrograms and presets

The dataset (Section 2.2) has been randomly split into a held-out test set (20%) and a train/validation set (80%). Each model is trained five times in order to perform a k-fold cross-validation [67, Chap. 5] procedure (64% training set, 16% validation set). Results presented in the following subsections have been obtained from the test set only. However, performing 5-fold cross-validation places a significant demand on CPU and GPU resources, and requires substantial storage space (a few GB for each trained model) for validation and test audio outputs.

Models are trained during 400 epochs using the Adam optimizer [169] with a minibatch size of 160. In order to stabilize the training of normalizing flows, the learning rate increases linearly from 2×10^{-5} to 2×10^{-4} over the first 6 epochs. All losses are tracked to identify diverging values and automatically restart a training run if necessary. A scheduler divides the learning rate by 5 if validation losses did not improve during the last 12 epochs, and training could be stopped earlier if the learning rate falls below 10^{-7} . A linear β -warmup [170] from 50% to 100% is performed from epoch 0 to epoch 25. Early tests showed that β -warmup starting from 0% seems to significantly decrease validation performances. The training and validation procedure for a single model requires about 2.5 hours on two NVIDIA GTX 1070 GPUs.

2.4 Results and model analysis

At the time of its publication in 2021, the PresetGenVAE model was the state of the art in DX7 FM synthesizer sound matching, as indicated previously in Subsection 2.1.3. In the following sections, multiple variants of this model are explored in order to gain a deeper understanding of the key factors contributing to its performance.

2.4.1 Preset representation and decoder

Experimental results presented in this subsection compare the three following representations of presets:

- *Num only* considers all parameters as continuous numerical variables, including categorical parameters;
- *NumCat* indicates that numerical parameters are learned as continuous numerical variables and that categorical parameters are one-hot encoded (Figure 2.14);
- *NumCat++* indicates that categorical and low-cardinality (up to 32) numerical parameters are one-hot encoded.

Numerical and categorical synthesis parameters

The first experiment examines how the flow-based decoder network handles the three proposed representations of presets, and if the modifications introduced with PresetGenVAE enable the modeling of entire presets instead of subsets of parameters. All audio files and spectrograms were obtained by playing a MIDI note with a pitch of 65 (F4) and a medium intensity of 85 out of 127. Results are presented in sections (a) and (c) of Table 2.5, which reports the MAE of inferred numerical DX7 parameters and accuracy of categorical DX7 parameters. In order to measure the audio distance from the target input, audio files were rendered using decoded presets. Consistent with a similar study [134], the MAE between true and synthesized log-magnitude STFTs, as well as the 40-band MFCCs (86 time frames) MAE are presented.

Table 2.5. Comparison of flow- and MLP-based decoder models and different representations of presets. Reported values are averages of mean values from the five training folds, plus or minus one standard deviation. (\uparrow) indicates that higher values are better and (\downarrow) indicates that lower values are better.

Preset representation	L_Z	Parameters		Audio	
		Numerical MAE ($\times 10^{-1}$) (\downarrow)	Categorical Accuracy (%) (\uparrow)	Spectrogram MAE log (\downarrow)	MFCC MAE (\downarrow)
(a) 81 synthesis parameters (3 operators), flow-based decoder					
Num only	81	1.39 ± 0.03	58.1 ± 1.8	0.55 ± 0.02	17.9 ± 0.9
NumCat	140	1.19 ± 0.02	84.3 ± 1.0	0.51 ± 0.01	17.2 ± 0.6
NumCat++	340	0.92 ± 0.03	85.3 ± 1.0	0.47 ± 0.01	12.1 ± 0.5
(b) 81 synthesis parameters (3 operators), MLP-based decoder					
Num only	340	1.24 ± 0.02	66.6 ± 0.7	0.50 ± 0.01	16.1 ± 0.3
NumCat	340	1.19 ± 0.01	83.9 ± 0.9	0.55 ± 0.02	17.1 ± 0.4
NumCat++	340	1.03 ± 0.03	83.3 ± 0.7	0.47 ± 0.01	12.2 ± 0.4
(c) 144 synthesis parameters (6 operators), flow-based decoder					
Num only	144	1.47 ± 0.01	63.4 ± 2.3	0.74 ± 0.04	19.2 ± 0.6
NumCat	224	1.28 ± 0.01	85.1 ± 0.3	0.65 ± 0.01	18.6 ± 0.6
NumCat++	610	1.05 ± 0.01	86.0 ± 0.6	0.62 ± 0.01	14.7 ± 0.3
(d) 144 synthesis parameters (6 operators), MLP-based decoder					
Num only	610	1.36 ± 0.02	68.6 ± 0.8	0.70 ± 0.01	19.1 ± 0.1
NumCat	610	1.27 ± 0.00	84.4 ± 0.6	0.65 ± 0.01	18.6 ± 0.2
NumCat++	610	1.11 ± 0.02	84.4 ± 0.5	0.64 ± 0.01	15.0 ± 0.4

Out of the three representations, *NumCat++* is clearly and consistently the best. It seems to enable a finer modeling of the discrete numerical parameters learned using a one-hot categorical encoding. These parameters include the *Detune* and *Coarse Frequency* controls of each operator, which are main factors in shaping the FM spectrum. Hence, the matching is also improved in the spectral and MFCC domains. These findings are consistent with previous studies on smaller sets of synthesis parameters [135].

In terms of scalability, the performance of our model degrades only marginally when using six oscillators (144 parameters) instead of three (81 parameters). This is expected, as the three additional operators are operators 4, 5, and 6, which are the deepest ones in most DX7 FM architectures (Fig. B.1). They typically contribute to subtle modulations of higher harmonics, which are more challenging for the model to identify from Mel-spectrograms. Interestingly, the accuracy of categorical parameter inference improves as the number of learned parameters increases. These observations suggest that, unlike the related FlowSynth model [134], PresetGenVAE is scalable and should be applicable to any synthesizer with a larger set of controls.

MLP decoder with a fixed latent size

The *NumCat++* representation increases the size of the vector \mathbf{u} representing the preset. As a result, when using an invertible Flow-based decoder, the latent size L_Z also increases, which should facilitate the compression task of the spectral VAE. Hence, it is necessary to study whether the benefits of *NumCat++* models come from the representation itself, or simply from the L_Z increase.

For this second experiment, the three parameter representations are tested with a fixed large latent space dimension L_Z . However, the bijective flow decoder must be replaced by a non-invertible neural network, allowing for a different number of input and output dimensions. A standard 4-layer MLP with 1024 hidden units has been chosen. BN and dropout (0.4 probability) were appended to the two first MLP layers only. When using a full preset as target, the largest MLP decoder contains 3.0M parameters (5.9 MFLOP) whereas the largest flow decoder has 1.6M weights but requires more operations to process a single input (11.3 MFLOP).

Results are reported in sections (b) and (d) of Table 2.5. Using either three or six operators, *Num only* variants lead to the worst results. Compared to *NumCat*, the *NumCat++* models improve all metrics but categorical parameters accuracy, which is similar (for three operators) or slightly decreases (for six operators). Therefore, it is concluded that variations observed in sections (a) and (c) of Table 2.5 were not caused by the L_Z increase alone.

Moreover, these results allow to compare MLP- and flow-based decoder models. Consistent with [134], MLP networks underperform although they theoretically are universal approximators of vector-valued functions [171]. Despite having fewer neural network weights, normalizing flows achieve better performance.

Audio examples

Sound matching audio examples from the held-out test dataset are provided in Fig. 2.13. For many harmonic sounds, the reconstruction is very good and often imperceptible. In particular, harmonic electric piano sounds, such as the Rhodes, are usually properly translated into presets. These are typical DX7 sounds, accounting for almost 10% of the dataset (Fig. 2.7). On the spectral representation in Fig. 2.13, the matched violin preset may look quite dissimilar to the target. However, it sounds quite similar. For percussive samples, the performance is mixed, with some reconstructions being accurate and others less so. This may be due to the limitations of the spectrogram input, which is not ideal for describing and modeling transients. For sounds labeled as special effects (*sfx*), PresetGenVAE often performs poorly. It tends to find presets that have some general timbre similarity but sound obviously different from the target. For example, in Fig. 2.16, the decoded "R2.D2" preset has a lot of high-frequency content and properly selects a square-wave LFO, but it is clearly not the desired output. Additional audio examples are provided on the project's companion website⁹.

2.4.2 Multi-channel spectrograms

Audio projections of a preset

To the best of our knowledge, all previous and subsequent works related to preset finding consider that the sound to be matched corresponds to a unique note pitch and intensity. This includes the PresetGenVAE variants introduced in Subsection 2.4.1. However, these approaches entirely miss the role of synthesis parameters describing how timbre evolves with a note's pitch and intensity.

⁹<https://gwendal-lv.github.io/preset-gen-vae/>

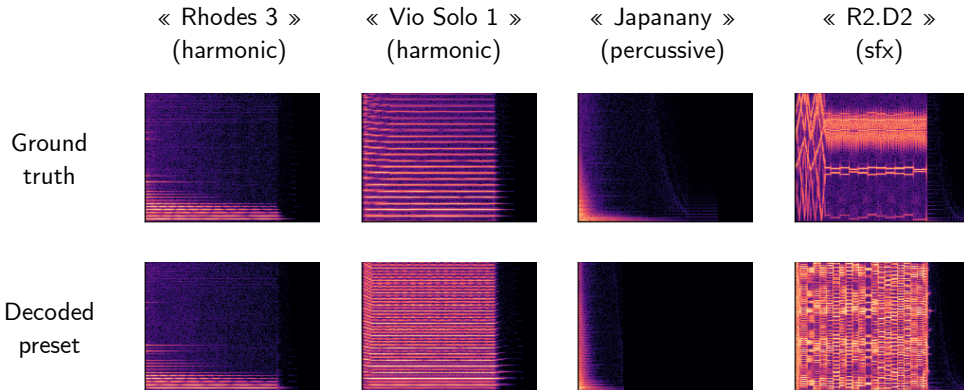


Figure 2.16. Sound matching examples using targets from the test dataset.

These relationships are complex. E.g., linearly increasing the intensity should not result in a linear amplitude augmentation, and playing a higher pitch often does not simply shift the spectrum towards higher frequencies. These relations between [MIDI](#) notes and loudness, pitch and timbre are described by numerous parameters, such as the *L/R Key Scale*, *L/R Scale Depth*, *Rate Scaling*, *Key Velocity* or *Break Point* controls of each DX7 operator.

Hence, an input corresponding to a single [MIDI](#) note is not enough to retrieve all synthesis parameters. This would be somewhat equivalent to trying to reconstruct a 3D scene from a single 2D projection (a single camera shot). Although this approach may work for simple scenes taken from the right angle, it is in general not adequate. Multiple points of view are required for a neural network to be able to build proper representations of the scene, such as shown in [Fig. 2.17](#) from the *generative query networks* [172] article. In this context, a 3D scene is described by the location, size, orientation and color of some 3D shapes, and corresponds to a preset. A high-level representation of a 3D scene corresponds to a \mathbf{z}_K latent vector. A 2D rendering of the scene (from a specific position and angle) corresponds to a particular audio rendering of a preset (using a given [MIDI](#) note).

Inspired by the generative query networks, multi-channel images are provided at the [VAE](#)'s input ([Fig. 2.18](#)), where each channel corresponds to a single note. The following experiment processes six-channel spectrograms with associated

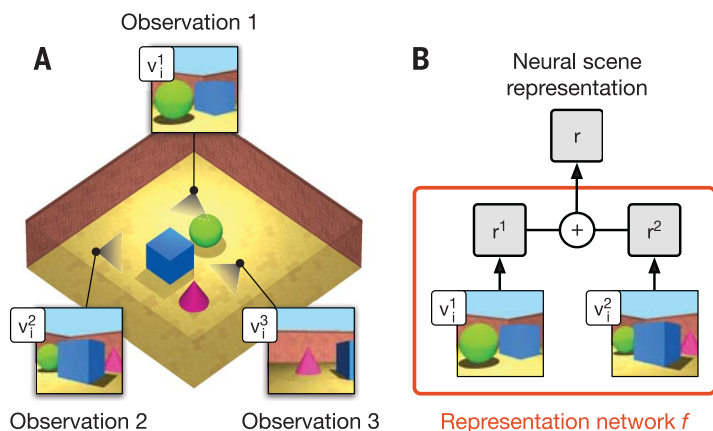


Figure 2.17. Neural representation of a 3D scene using multiple 2D camera shots of the scene. Figure extracted from [172].

MIDI (pitch, intensity) values of (40, 85), (50, 85), (60, 42), (60, 85), (60, 127), and (70, 85). This architecture utilizes a single-spectrogram encoder neural network, which is applied sequentially to each input channel. The resulting outputs are compact 256-channel feature maps that encapsulate high-level information about each spectrogram. These feature maps are subsequently stacked and combined using another **CNN**. The decoder operates on a similar principle. Compared to the single-channel model (Fig. 2.13), this new **VAE** processes six times more data, thus requires more parameters. Nevertheless, the multi-channel neural network contains less than two times more parameters (42.8M vs. 27.3M). Hence, it can be considered parameter efficient.

Improved parameter inference

To compare single- and multi-channel architectures in a fair manner, the single-channel model in Fig. 2.13 is slightly modified. First, the 512-channel encoder and decoder layers are enlarged to 1800 channels, so that the total number of weights is 43.0M. Second, the two first elements of μ_0 are set to the **MIDI** pitch and intensity (normalized into $[-1, 1]$) of the corresponding input spectrogram. The two first elements of σ_0 are set to $2/127$. Both models are

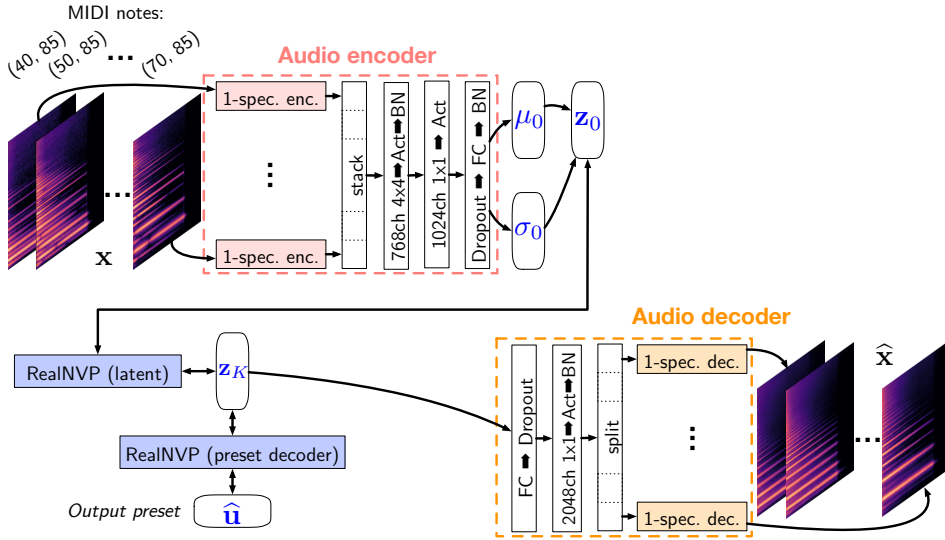


Figure 2.18. PresetGenVAE model with multi-channel input and output spectrograms obtained from a single preset played using different MIDI notes. *1-spec* blocks partially encode or decode a single spectrogram, and are applied sequentially to each input channel. They contain the blocks indicated by the same color on Fig. 2.13.

Table 2.6. Comparison of the single-channel and multi-channel models

Input ch.	All parameters		Pitch and intensity parameters		Audio	
	Numerical MAE $\times 10^{-1}$ (\downarrow)	Categorical Accuracy % (\uparrow)	Numerical MAE $\times 10^{-1}$ (\downarrow)	Categorical Accuracy % (\uparrow)	Spectral MAE log (\downarrow)	MFCC MAE (\downarrow)
81 synthesis parameters (3 FM operators), $L_Z = 340$						
1	0.97 ± 0.01	83.9 ± 0.4	0.240 ± 0.003	85.6 ± 0.3	0.47 ± 0.01	12.1 ± 0.3
6	0.86 ± 0.01	86.3 ± 0.4	0.185 ± 0.006	87.3 ± 0.3	0.46 ± 0.01	11.1 ± 0.3
144 synthesis parameters (6 FM operators), $L_Z = 610$						
1	1.14 ± 0.02	84.0 ± 0.3	0.308 ± 0.008	84.3 ± 0.3	0.64 ± 0.01	15.2 ± 0.1
6	0.99 ± 0.02	86.7 ± 0.4	0.240 ± 0.008	86.4 ± 0.3	0.60 ± 0.01	13.9 ± 0.3

trained as described in the previous section, with training duration increasing to approximately seven hours.

The results are presented in Table 2.6 and reports both audio synthesis accuracy and parameter inference accuracy. However, unlike Table 2.5, which focused on a single note, audio accuracy here is evaluated using the six training MIDI notes. Table 2.6 separately shows results regarding parameters specifically related to the pitch and intensity of the played notes.

Models utilizing multi-channel input spectrograms show a notable improvement across all criteria. Example spectrograms are provided in Fig. 2.19, where the single-spectrogram version does not properly estimate synthesis parameters influencing the magnitude. Learning all synthesizer parameters using just a single MIDI note was likely an ill-defined problem, as some parameters require multiple pitches and/or intensities for accurate estimation. Therefore, it is argued that all sound matching frameworks should incorporate multi-channel convolutional structures similar to the one introduced in this section.

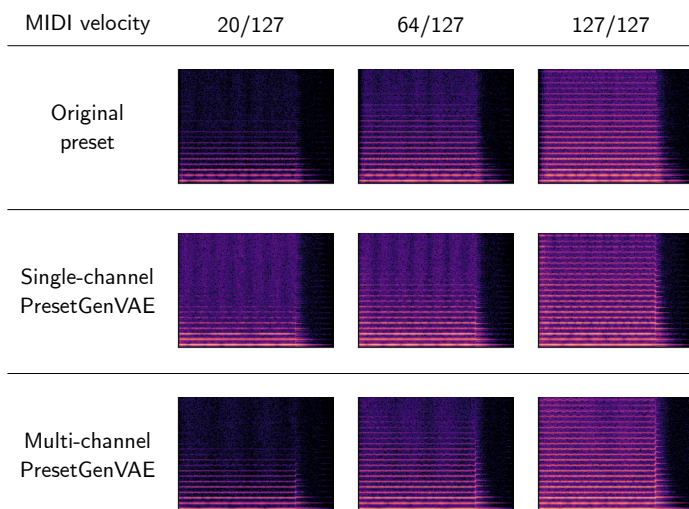


Figure 2.19. Comparison of output presets from the single- and multi-channel PresetGenVAE models, rendered to audio using multiple MIDI notes.

2.4.3 Discussion

Sound matching performance

The key question that arises is whether PresetGenVAE achieves a sufficient level of accuracy in sound matching. Unfortunately, there does not exist a unified framework for evaluating synthesisers in general [113]. Regarding audio similarity, commonly used metrics are typically based on log-magnitude spectra (e.g. MAE, MSE or MSS errors) or distances in MFCC domain.

As reported in Table 2.4, another study [136] confirms that PresetGenVAE performs better than previous sound matching works for DX7 FM sound matching, including those using heuristics or neural networks. However, Table 2.4 also indicates that a difference between matched and target presets, albeit small, seems to be on average perceptible. Informal listening tests suggest that these differences can indeed be heard. Formal subjective listening tests, while potentially insightful, have not been conducted because they are not a viable option when comparing numerous models.

For DX7 sound matching, PresetGenVAE was considered state of the art until the introduction of the Sound2Synth model in 2022 [136]. Sound2Synth is nonetheless non-generative, and it relies on 32 distinct models operating in parallel, each tailored to a specific FM algorithm.

Interfacing synthesizer plugins

VST plugins are ubiquitous, yet difficult to use directly from Python code, requiring a significant amount of engineering work to create efficient and functional data pipelines. Without proper optimization (e.g., Fig. 2.15), rendering audio files could take several days. In this thesis, plugins are handled using the now outdated RenderMan wrapper. Alternative and newer technical solutions could be explored, such as DawDreamer¹⁰. In order to keep computational resources for training models, it would be interesting to develop a software framework for rendering audio files on a dedicated server within the local network. These considerations are left for future work.

¹⁰<https://github.com/DBraun/DawDreamer/>

Latent interpolation of presets

Based on a VAE, the PresetGenVAE model possesses generative capabilities, including interpolations in the latent space. Ideally, when interpolating, the decoded presets should gradually transition from one timbre to another. However, in practice, many unexpected intermediate sounds are produced. Due to poor interpolation quality, further study with PresetGenVAE was deemed unnecessary. A dedicated model appears necessary to achieve smooth latent space interpolations of presets, a topic which is further discussed in Chapters 4 and 5.

2.5 In brief

Summary of Chapter 2

- This chapter has provided an extensive review of automatic synthesizer programming techniques, suggesting several categories such as heuristic methods, and generative and non-generative data-driven models.
- In order to train a generative model, a dataset of DX7 FM synthesis presets has been gathered, filtered and labeled.
- The PresetGenVAE model has been introduced and applied to sound matching for 144-parameter DX7 FM presets, including the FM *algorithm* which modifies the internal routing of signals. It was state-of-the-art until a non-generative model enabled better performance in 2022.
- Multi-channel input spectrograms can be utilized to improve the estimation of synthesis parameters, in particular those related to the pitch and intensity of MIDI notes.

Perspective for Chapter 2

- PresetGenVAE is a generative model which theoretically enables interpolation. Unfortunately, the quality of latent space interpolations of presets seems poor and another model may be necessary.
- This work provides valuable insights into building and training scalable VAEs. It also offers results regarding the modeling of numerical and categorical synthesis parameters separately.
- Controlling synthesizers with text prompts is a potential research avenue, but is beyond the scope of this thesis.

Chapter 3

Sound morphing

Contents

3.1 Literature overview	93
3.1.1 Analysis and resynthesis	93
3.1.2 Traditional signal processing techniques	94
3.1.3 Data-driven methods	98
3.2 Method for objective evaluation	99
3.2.1 Previous works	99
3.2.2 Timbre features extraction	101
3.2.3 Sound morphing metrics: linearity and smoothness	106
3.3 Comparison of morphing techniques	108
3.3.1 Experiment description	108
3.3.2 Results	112
3.3.3 Discussion	118
3.4 In brief	119

Sound morphing involves blending multiple inputs to generate a new sound with an intermediate timbre. It is sometimes called timbre morphing or timbre interpolation [49], and it produces intermediate, hybrid sounds that incorporate elements from each original sound. In contrast, waveform cross-fading, which can also be referred to as vector synthesis [1, Chap. 1], or simply mixing, is usually not considered as a morphing technique.

An example is provided in Fig. 3.1, where the intermediate cross-fade samples clearly appear as a superposition of the spectra from the two original samples. The two original components (the start and end samples) can be easily and distinctively heard in each intermediate mix. In contrast, when morphing, each intermediate sample has its own timbre. Each morphed sample is perceived as a unique acoustic source rather than a summation of two distinct sources. Morphing also involves a gradual evolution of spectro-temporal characteristics. For example, in the morphing presented in Fig. 3.1, the attack time is smoothly reduced from the start (long attack duration) to the end (very short attack). The number of harmonics, related to timbre features such as *brightness* or *spectral spread*, is also reduced gradually at each step of the sequence.

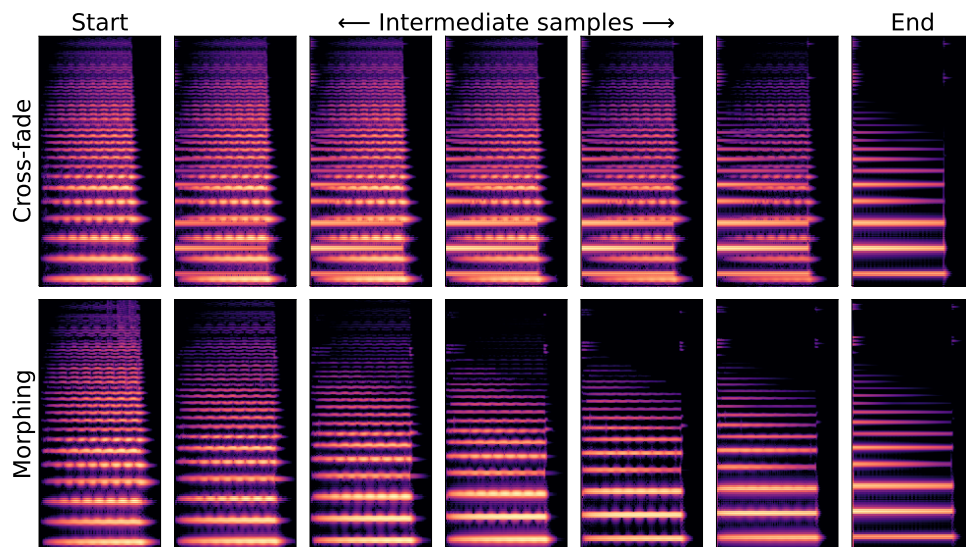


Figure 3.1. Comparison of cross-fading and morphing using the same start and end (also called *source* and *target*) samples.

Section 3.1 provides an overview of past and current morphing techniques, structured around different methods of analysis and resynthesis of audio signals. Section 3.2 focuses on the evaluation of sound morphing, which in most studies relies on a qualitative analysis of a few selected examples. This section also introduces an objective method based on timbre features and computations of metrics. Finally, Section 3.3 provides a quantified comparison of state-of-the-art techniques for sound morphing, and an analysis of the results.

The main contributions of this chapter are as follows:

- An updated review of existing morphing techniques, including those based on recently published differentiable (neural) synthesis models;
- A toolbox for automatic, reproducible and white-box evaluation of any sound morphing, inspired by previous works and extended using new criteria;
- The first large-scale evaluation of multiple morphing techniques, based on objective criteria rather than qualitative analysis of examples.

3.1 Literature overview

3.1.1 Analysis and resynthesis

Analysis–resynthesis systems enable creative and flexible sound transformations by representing waveforms through a set of musically relevant features [173]. These features are extracted from the time-domain signal during the analysis phase (Fig. 3.2), and most often use different kinds of time–frequency representations. The representation can undergo several transformation steps and should be suitable for audio resynthesis.

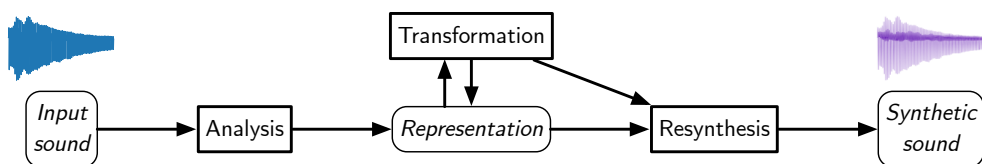


Figure 3.2. Overview of an analysis-resynthesis system.

All morphing algorithms are based on some kind of analysis-resynthesis process, by performing transformations on audio representations rather than simply mixing time-domain waveforms. Various signal processing techniques can be used to perform the analysis and resynthesis operations, and the most common are briefly described in the following subsections. In the context of deep neural networks for modeling audio, analysis and resynthesis are usually named encoding and decoding, respectively. The representation (Fig. 3.2) then corresponds to a latent code.

3.1.2 Traditional signal processing techniques

Sinusoidal modeling

For morphing purposes, it is common to describe musical instrument sounds as a set of sine waves with amplitudes, frequencies, and phases that vary over time. These elementary waves are also called sinusoids or partials. Regarding sinusoidal analysis, multiple techniques have been developed to estimate the location (frequency) and amplitude of the partials [12], and to track their evolution over time. Resynthesis is relatively straightforward and typically involves using an additive synthesizer (Subsection 1.2.1). Extra noise sources with time-varying spectral properties can be used alongside the main sinusoidal components [12]. They can help model breath noises for wind instruments, scraping sounds, etc.

A notable example is the Lemur representation [174] for timbre morphing, published in 1995. In this representation, partials can be created when their magnitude rises above a given threshold, or deleted when their magnitude becomes too low. The total number of partials in a waveform can change throughout its duration.

The analysis and representation of transients pose a significant challenge in additive synthesis [49]. For instance, accurately capturing the onset of a sound, which is crucial for perception, is difficult due to the need for precise time resolution. Hence, sound morphing algorithms have tried to tackle this issue. E.g., in 2003, Loris [175] introduced the *reassigned bandwidth-enhanced additive* sound model. This representation can merge sinusoidal and noise energy into a single partial, characterized by time-varying parameters for frequency, amplitude, and noisiness (or bandwidth). A more recent study [176], based on

sinusoidal modeling, estimates a noise source and filter using linear predictive coding [177].

Another series of works [178–180] introduced the explicit control of morphing using dimensions of timbre, such as *brightness*, *softness* and *warmth*. They also rely on sinusoidal plus noise modeling, named *additive/residual*. For instance, a *brightness* morphing can be performed by shifting the spectral centroid of an harmonic sound, through amplification or attenuation of its sinusoidal components [178].

Sinusoidal analysis-resynthesis in general is known for enabling high-quality sound morphing, although a formal comparison with other techniques could not be found in the literature. This will be discussed in Sections 3.2 and 3.3 of this chapter.

This type of modeling nonetheless suffers from a few technical drawbacks. It requires dozens or hundreds of partials, which increases the computational cost of both analysis and resynthesis. For instance, a low-pitch piano note typically requires 100 partials for proper reconstruction [181]. Some morphing libraries provide implementations for dedicated and powerful DSP hardware. E.g., Loris [175] can run inside the Kyma [182] environment using an external proprietary DSP. Others, such as the recently published Sound Morphing Toolbox (SMT) [183] for Matlab, can run on a consumer-grade CPU but usually remain CPU-intensive.

Manipulations and transformations of such representations, by practitioners or algorithms, are tedious [181]. In the context of sound morphing, the partials of the two input sounds need to be matched. I.e., a correspondence between fundamental frequencies, harmonics and other non-harmonic partials of the two representations must be established. This poses a difficult problem [184]. A perfect one-to-one correspondence is highly improbable [176], except in cases where both input sounds are monophonic and quasi-harmonic musical instrument sounds with the same pitch [184]. Some methods also require to manually align some temporal events (e.g., attack, decay, release, ...) of the input signals [175], and to manually label all partials for establishing correspondences. This prevents a fully automated morphing procedure. Recent works such as SMT [183] automatically perform partial matching as well as temporal event identification. Such an analysis is, however, extremely CPU-intensive, as demonstrated in Subsection 3.3.2 at the end of this chapter.

Source-filter modeling

Source-filter models assume that signals are obtained by processing an input signal (also called the excitation) through a filter or resonator. They have been widely applied in speech analysis, processing, and synthesis [181]. A famous example is linear predictive coding [177] for modeling and compressing speech signals.

The use of source-filter models can be extended to musical sound manipulations, including morphing. The concept of filter is closely related to that of spectral envelope [181], which designates a scalar function that connects the dominant peaks of a sound's spectrum (Fig. 3.3). It changes over time, e.g., different envelopes can correspond to different vowels within a speech signal.

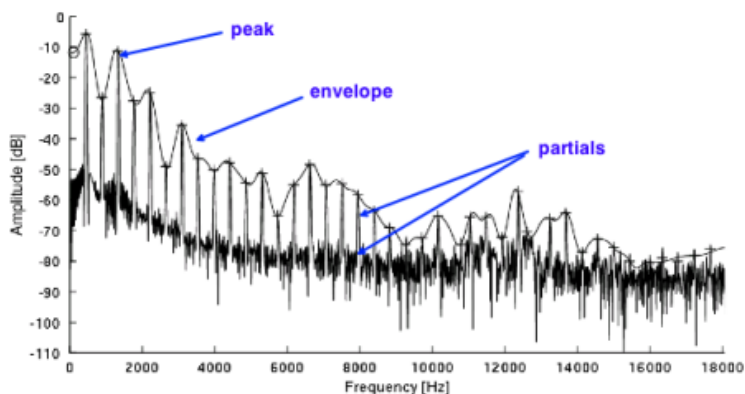


Figure 3.3. Example of a waveform decomposed into partials and a spectral envelope. Figure from IRCAM AudioSculpt's online manual [185].

Analysis-resynthesis models based on spectral envelopes enable transformations such as modifying the envelope while keeping the partials at the same frequency location, or shifting the harmonic contents with a constant envelope. Compared to sinusoidal models with hundreds of individual sine waves, the control is simpler and more direct [181]. For an extensive discussion about the various techniques for estimation and manipulation of spectral envelopes, the reader can refer to Rodet and Schwarz [181].

For morphing purposes, sinusoidal representations can be extracted first and then transformed them into partial locations and spectral envelopes [184, 186,

187]. Residuals (analyzed signals without the spectral peaks) can be modeled as white noise and filtered through other spectral envelopes.

In another line of work, STRAIGHT representations [188] are based on estimates of the instantaneous fundamental frequency and spectral envelope. They are also referred to as smooth pitch-adaptive time-frequency representations. Primarily intended for speech transformations, they can perform any sound morphing by piecewise bilinear interpolation on the time and frequency axes. Nevertheless, this requires anchor points in the smooth time-frequency representations.

Other techniques

Theoretically, sinusoidal plus noise and source-filter models can reproduce an arbitrary input waveform. However, in practice, there is often a perceptible difference between original and reconstructed signals [186], even for quasi-harmonic orchestral instruments. In particular, filtered white noise is often an inaccurate model of the true residuals of a sinusoidal decomposition [186].

Hence, some specific techniques dedicated to the morphing of noisy, non-harmonic transients have been developed. For example, [189] employs discrete wavelet transforms. In the more specific field of video game sound design, impact sounds [190] and granular textures [191] can be interpolated using inverse cumulative amplitude spectra. In order to morph between two similar percussive sounds, e.g. two snare or two hi-hat samples, [192] suggests using basic cross-fading. The two inputs must be temporally aligned using time stretching. More generally, cross-fading may be a proper morphing technique when input sounds are very similar in timbre, pitch, and duration, though this is a rare case.

Other alternative spectral techniques allow for sound morphing without explicitly relying on a signal model. For example, interpolation can be carried out in a perceptual spectral domain derived from MFCC and inverse MFCC calculations [193]. In [194], the morphing is performed with the help of 2D convolutions applied to the input spectrograms. Non-negative Matrix Factorization (NMF) can also be used to decompose the two inputs, provided as magnitude spectrograms, into bases and activations. The morphing is achieved by interpolating the spectral bases while using the activation of the first input

only [195]. This process can be applied separately to the quasi-harmonic and transient components, obtained through HPSS [155].

3.1.3 Data-driven methods

Recent research demonstrates that morphing algorithms can incorporate data-driven, differentiable models. For example, [196] generates Mel-spectrograms conditioned on the embeddings of various musical instruments, while a WaveNet [72] vocoder reconstructs audio waveforms from Mel-spectrograms. The *latent timbre synthesis* [85] approach presents a VAE with CQT for both inputs and outputs. Additionally, GANs have been applied to the specific task of audio texture morphing [197].

Moreover, many differentiable synthesizers introduced in Section 1.4 are trained as autoencoders and can be employed for morphing. The encoder performs some sort of analysis of the input sound to be reconstructed, while the decoder drives a resynthesis process. Morphing can be performed through interpolation of internal representations, e.g., linear interpolation between latent codes corresponding to two different sounds.

For instance, a DDSP HpN model [95] can be used for timbre morphing [198]. The underlying analysis-resynthesis process is conceptually similar to that of sinusoidal models with additive noise (introduced in Subsection 3.1.2). In fact, most DDSP-based synthesizers, such as those using wavetables [107], can morph by interpolating latent representations. Autoencoder models for direct waveform generation, such as WaveNet [71] or RAVE [82], can be easily used for latent-based sound morphing as well.

Here, the analysis-resynthesis is data-driven rather than based on signal models or known time-frequency transforms. However, compared to traditional techniques such as sinusoidal modeling, latent representations do not have a clear meaning and cannot be easily used to perform fine sound manipulations.

3.2 Method for objective evaluation

3.2.1 Previous works

Task complexity

Evaluating interpolation quality is straightforward for simple artificial shapes like 2D lines [199]. Clear and objective features, such as length and orientation, can be measured for each step. For instance, the sequences in Fig. 3.4 show a diverse set of interpolations for synthetic 2D lines. Sequence (a) resembles a fade-out followed by a fade-in rather than a proper morphing. Sequence (b) is also not a good example of morphing, as a sudden change in orientation can be observed. It is not *smooth*. Sequence (c) is better, although the angle does not evolve consistently, and sequence (d) appears ideal with a *linear* trajectory of both angle and length features.

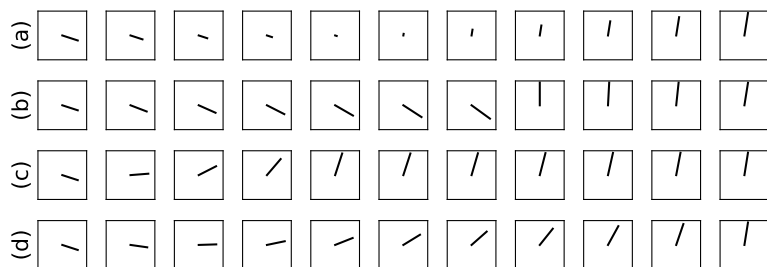


Figure 3.4. Four sequences of interpolated artificial 2D lines.

In contrast, determining what defines a 'good' interpolation in the domain of audio is more challenging. Formal perceptual evaluations are time-consuming [187] and cannot realistically be used for comparing numerous models using large sets of test sounds. Studies about sound morphing have relied on different methods described in the following paragraphs.

Qualitative analysis of examples

A large majority of works related to sound morphing provide a qualitative analysis based on a few selected examples. Several works [178, 179, 188, 190, 193, 194, 198, 200] discuss the morphing quality of a single sequence, e.g., a morphing from a bassoon to a clarinet, or from an impact on wood to an impact on porcelain. Other studies [174, 176, 180, 189, 191, 195] focus on a few more examples (two to five), up to 24 source timbres [175]

Such qualitative analyses are useful for users to know how well a method can be expected to perform in a specific domain. However, it is also a strong limitation and does not indicate whether the method is usable outside that domain. A proper and larger test dataset would be appropriate.

Subjective tests

A few morphing algorithms have been supported by subjective tests. For instance, ten subjects were asked to rate the *naturalness* of STRAIGHT [188] morphings. Regarding percussive sounds, a subjective *linearity* can be estimated by asking subjects whether a morphed step is perceived as closer to the start or to the end sample [192].

Another subjective experiment [187], involving eleven subjects, compared source-filter models to sinusoidal modeling. Morphings obtained from a pair of harmonic sounds were presented, and subjects were asked which model was *smoother*. Twenty-six pairs of sounds were available. The results were inconclusive, and neither technique appeared to outperform the other.

Objective evaluations

A theoretical framework for the formal evaluation of sound morphings [201] introduces three main criteria: *correspondence*, *intermediateness* and *smoothness*. *Correspondence* designates the mapping from salient features of the source to those of the target, which should remain consistent during morphing. *Intermediateness* refers to the perception of intermediate samples as resulting from the hybridization of the two inputs. For instance, the middle item in Fig. 3.4(c) is not intermediate, because it is very similar to the

end sample. *Smoothness* should be a measure of how gradual the transitions are. However, this work [201] only provides general guidelines without explicit methods or formulas for computing the suggested criteria.

The first fully-defined objective evaluation [187] computes the quadratic error between measured and ideal values of five features. The feature set consists of four spectral features (centroid, spread, skewness, and kurtosis) and the temporal centroid, the computation of which is further discussed in Subsection 3.2.2. Feature values are computed for 26 different morphings with their own pair of starting and ending sounds, and for linearly spaced morphing steps. The ideal feature trajectory is linear, hence the measured error represents a non-linearity. This evaluation [187] has inspired the method developed for this thesis, which is presented in Subsections 3.2.2 and 3.2.3.

A more specialized model [197], which generates and morphs audio textures (e.g. water, wind) using GANs, evaluates the linearity of internal GAN features as a proxy for morphing linearity. However, this method is specific to a class of models, and it is not supported by further objective or subjective studies.

3.2.2 Timbre features extraction

Descriptors

Computational acoustic modeling is a major and active research topic in the domain of timbre [16]. Two main approaches [19] can be found in the literature: the music psychology approach, and the Musical Information Retrieval (MIR) approach. Psychologists often try to identify and study the most salient attributes of timbre, such as the log-attack time or the spectral variation. As discussed in Section 1.1, a common method relies on MDS to create timbre dimensions and then identifies the tractable spectro-temporal attributes that correlate best. In contrast, MIR tasks do not rely on such limited sets of features with only a few timbre attributes. For instance, for instrument or genre classification, it is beneficial for algorithms to have a complete representation of the input audio, even though some attributes may be correlated.

In the context of sound morphing, all aspects of timbre should evolve gradually. Hence, the “MIR approach” has been considered the most appropriate. Timbre attributes, also named timbre descriptors [202], can be extracted from various time-frequency representations. Some descriptors, such as a note’s release duration, are computed only once per sound and qualified as *global*. Other descriptors, such as the spectral centroid, vary over the duration of a sound and are typically computed for each time frame. They are called *local* descriptors and typically provided as time series [202].

Two solutions have been used for computing attributes: Timbre Toolbox (TT) [203] and AudioCommons Timbral Models (ACTM) [204]. TT computes dozens of spectral characteristics, such as *inharmonic*ity or *spectral roll-off*, as well as temporal features such as *Log Attack Time (LAT)* or *decay*. These audio features serve as a substitute for how humans perceive timbre [202]. Temporal features are global, whereas spectral descriptors are local. For a given sound analyzed by TT, only the median value and Interquartile Range (IQR) of local features, rather than the entire time series, are kept [203]. Moreover, the *noisiness* feature has been discarded because it was almost always equal to its maximal 1.0 value, in particular for the dataset of FM-synthesized sounds introduced in Section 2.2. The *attack* feature has also been excluded, in order to focus on the more perceptually salient LAT. Finally, 45 TT features are considered. TT features such as F_0 or RMS volume, although classically defined as orthogonal to the dimensions of timbre [15], are purposefully kept for evaluating morphings. A log function has been applied to features homogeneous to an amplitude or a frequency, for features to lie on a scale that should be closer to perception. This also helps reduce the number of outlier values. The log distortion has also been applied to the *odd to even harmonic ratio* feature, which originally seemed to lie on an exponential scale and presented numerous outliers.

In addition to TT features, ACTM [204] offers seven global semantic descriptors like *boominess*, *warmth*, and *depth*. The values of ACTM attributes are obtained through regression models trained on audio samples labeled with semantic terms. These regression models use spectral and temporal features as input variables.

Feature selection

Values of features from **TT** and **ACTM** span very different ranges. Hence, they are zero-mean, unit-variance normalized using statistics of the training set of audio files described in Section 2.2.

As indicated by Peeters *et al.* [203], multiple pairs of **TT** descriptors may provide redundant information. The overlap with **ACTM** features is also unknown. Thus, an unsupervised feature selection process was applied to the combined **ACTM** and **TT** feature set. Spearman rank correlation coefficients were computed using descriptor values from the entire training dataset (Section 2.2), and features with a high correlation (above 0.9) to others were excluded. As a result, the number of **ACTM** and **TT** descriptors was reduced to 6 and 31, respectively. Given an audio input \mathbf{x} , the corresponding set of attributes is denoted $\mathbf{a} \in \mathbb{R}^{37}$.

The correlations between these 37 remaining features are presented in Fig. 3.5. Several pairs of descriptors are moderately to highly (from 0.6 to 0.9) correlated, but have been purposefully kept because they represent clearly different aspects of timbre. For instance, the **IQR** of harmonic energy ($tt_HarmErg_IQR$) and the **IQR** of the **RMS** envelope (tt_RMSEnv_IQR) have a 0.81 correlation coefficient. This could be expected because most samples from the dataset are mostly harmonic (Subsection 2.2.2). Hence, for such harmonic sounds, larger waveform amplitude variations imply larger changes in total harmonic energy from one time frame to another. Nonetheless, these two descriptors do not represent the same acoustic property, especially for non-pitched sounds. More details about the full set of descriptors, without feature selection, are available in Appendix E.

The distributions of normalized values for the 37 remaining descriptors are provided in Fig. 3.6. Many features, such as **ACTM** brightness, **ACTM** roughness or **TT** spectral roll-off, are nicely distributed with very few outliers. Others, such as the **TT** decay and decay slope, contain outliers with reasonable absolute values (maximum 15).

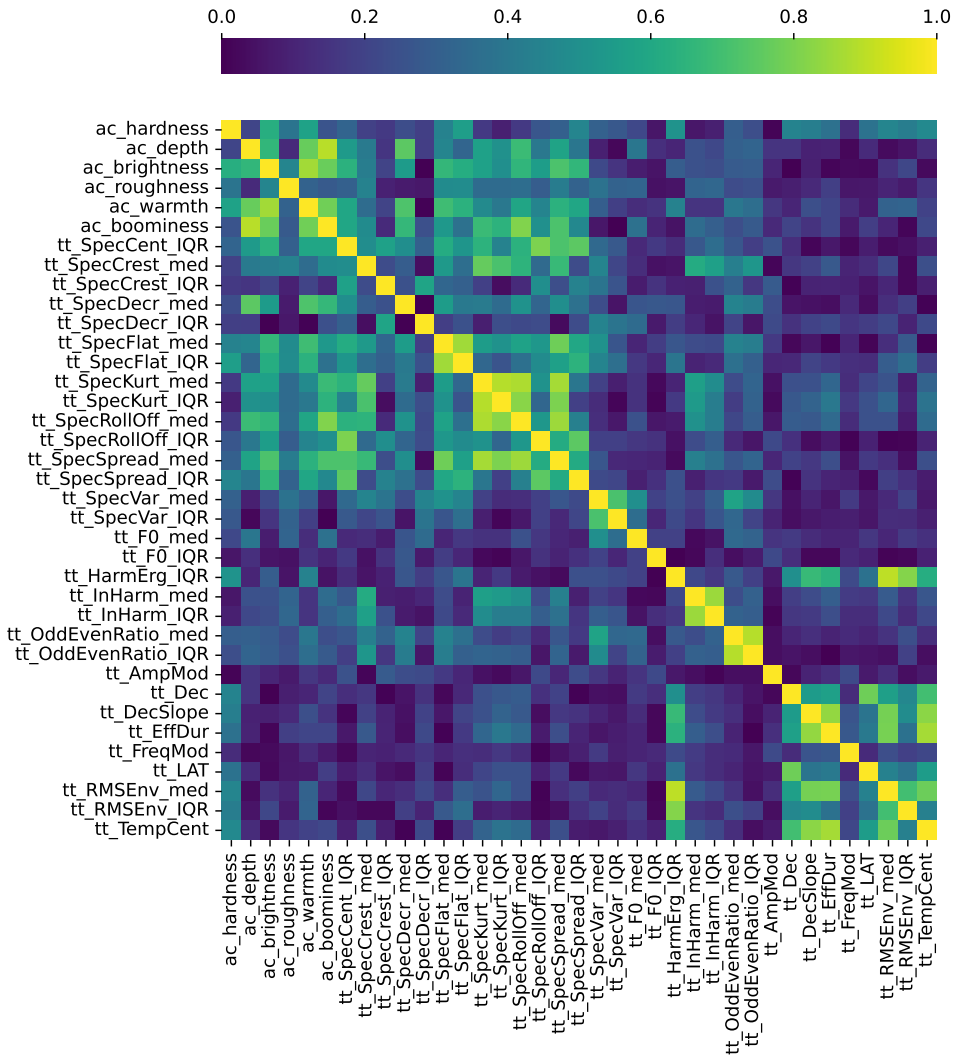


Figure 3.5. Rank correlations between the 37 retained timbre descriptors. The 'ac' and 'tt' prefixes indicate ACTM and TT features, respectively.

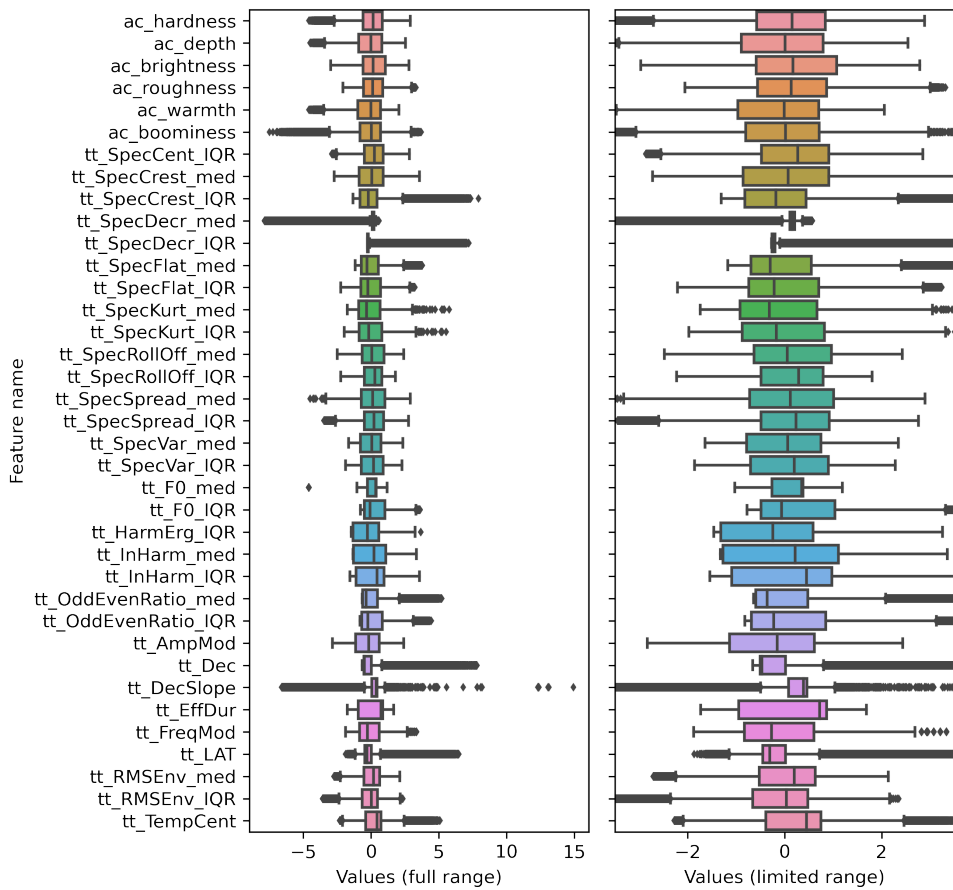


Figure 3.6. Distribution (boxplots) of normalized values for the 37 selected descriptors of timbre.

Implementation

The official **ACTM** implementation¹ is written in Python, whereas **TT** is available as a Matlab toolbox². The latter can be invoked from Python code by

¹https://github.com/AudioCommons/timbral_models

²<https://github.com/VincentPerreault0/timbretoolbox>

using Python sub-processes and running Matlab scripts from the command line.

However, both **ACTM** and **TT** are highly CPU-intensive, resulting in very long analysis times. Processing a dataset of thousands of samples required more than a week. While both libraries are demanding in terms of computational resources, **ACTM** is particularly inefficient despite providing fewer features. Upon exploring the code, it was found that several internal features, such as a 6th-order 20Hz high-pass filtered signal, or standardized multi-band loudness values for each time frame, were redundantly computed for multiple output descriptors. This issue was resolved by computing these features only once.

Additionally, computation times can be significantly reduced through parallelization. Features are computed sequentially (**TT** then **ACTM**), but each subset of features uses parallel processing, with 9 Matlab processes for **TT** and 12 Python subprocesses for **ACTM**. As a result, analyzing the dataset from Section 2.2 takes approximately one day on a high-performance 24-core CPU (AMD Ryzen Threadripper 3960X).

Such powerful **CPUs** are not always readily accessible to researchers. Therefore, **GPU** implementations written in PyTorch would be highly beneficial but are currently unavailable. Even better, a differentiable implementation would allow for integration of these descriptors into deep learning processes. Unfortunately, such alternatives are currently unavailable.

3.2.3 Sound morphing metrics: linearity and smoothness

Recent studies [199, 205–207] on model-based interpolation of various input modalities (images, sound, 3D motion, etc.) have primarily focused on the criterion of smoothness. Given an objective feature, its smoothness can be defined using the **RMS** of the second-order derivative of this feature’s values across the interpolation sequence [199]. In the context of morphing, the smoothness of the j^{th} timbre feature \mathbf{a}_j is:

$$\text{Smoothness}(\mathbf{a}_j) = -\sqrt{\frac{1}{T-2} \sum_{t=2}^{T-1} \left(\frac{\mathbf{a}_j[t-1] - 2\mathbf{a}_j[t] + \mathbf{a}_j[t+1]}{(T-1)^2} \right)^2}, \quad (3.1)$$

where T denotes the number of morphing steps. T is limited to relatively small values, such as 7, 9 or 11, in order to reduce the costs of morphing in terms of computation and storage. The minus sign ensures that the highest smoothness values (near zero) represent optimal smoothness, while increasingly negative values reflect a deterioration in smoothness. The smoothness of an audio morphing can be defined as the average smoothness of timbre features.

However, relying on smoothness alone can sometimes be misleading. For example, a degenerate interpolation consisting of zero-volume sounds could yield a medium smoothness score. In general terms, smoothness does not capture whether feature values deviate from an ideal linear path in feature space. This is illustrated by the *low linearity* curve from Fig. 3.7, which is moderately smooth.

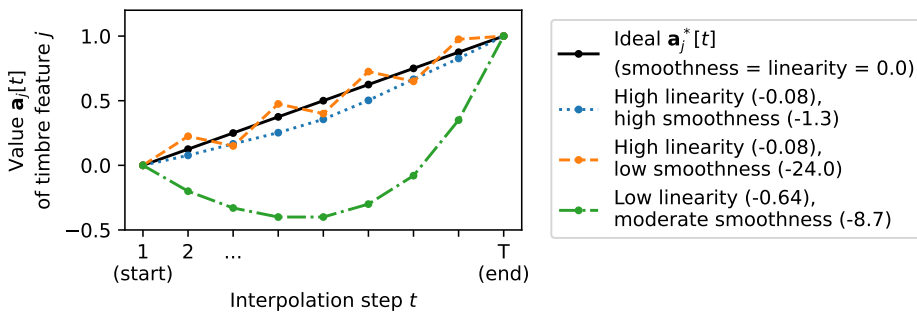


Figure 3.7. Examples of morphing linearity and smoothness for different trajectories of an artificial timbre feature.

Thus, the linearity of morphings is also evaluated. This is consistent with previous research on sound morphing [187], which considers that an interpolation is perceptually linear when timbre feature values change linearly. In this thesis, the feature-wise linearity is defined using the RMS distance between feature values $\mathbf{a}_j[t]$ and a linear regression from starting to ending values of the feature (the *ideal* curve from Fig. 3.7) denoted as $\mathbf{a}_j^*[t]$. It is formally computed as follows:

$$\text{Linearity}(\mathbf{a}_j) = -\sqrt{\frac{1}{T} \sum_{t=1}^T (\mathbf{a}_j[t] - \mathbf{a}_j^*[t])^2}. \quad (3.2)$$

Like smoothness, linearity is negative, with a low value indicating poor-quality morphing. The highest and optimal value is zero.

However, the linearity metric complements but does not replace the smoothness metric. Good smoothness ensures that the morphing avoids small yet sharp feature variations. For example, the *low smoothness* curve in Fig. 3.7 has the same linearity as the *good* curve, but its smoothness is roughly 20 times worse. The results presented in this thesis utilize both metrics: linearity and smoothness of interpolations. The method is available as an open-source toolbox³ and can be applied to any sound morphing. An example with four features is provided in Fig. 3.8.

3.3 Comparison of morphing techniques

3.3.1 Experiment description

Selected models

Numerous techniques are dedicated to morphing or can perform it as an auxiliary feature, and this experiment focuses on the most relevant and most cited ones. Works focusing on very specific types of sounds (impact sounds [190], audio textures [191], etc.) or specific instruments (e.g., DDSP models for piano [108] or guitar [109] modeling) have not been considered.

Methods without a usable open-source implementation were not included either. For instance, the Kyma implementation of Loris [175] requires proprietary external DSP hardware. Its C++ implementation⁴ is outdated and could not be compiled. Moreover, Loris requires an extensive manual labeling of test sounds, which prevents its usage with an automatic evaluation pro-

³<https://github.com/gwendal-lv/sound-morphing-metrics>

⁴<https://sourceforge.net/projects/loris/>

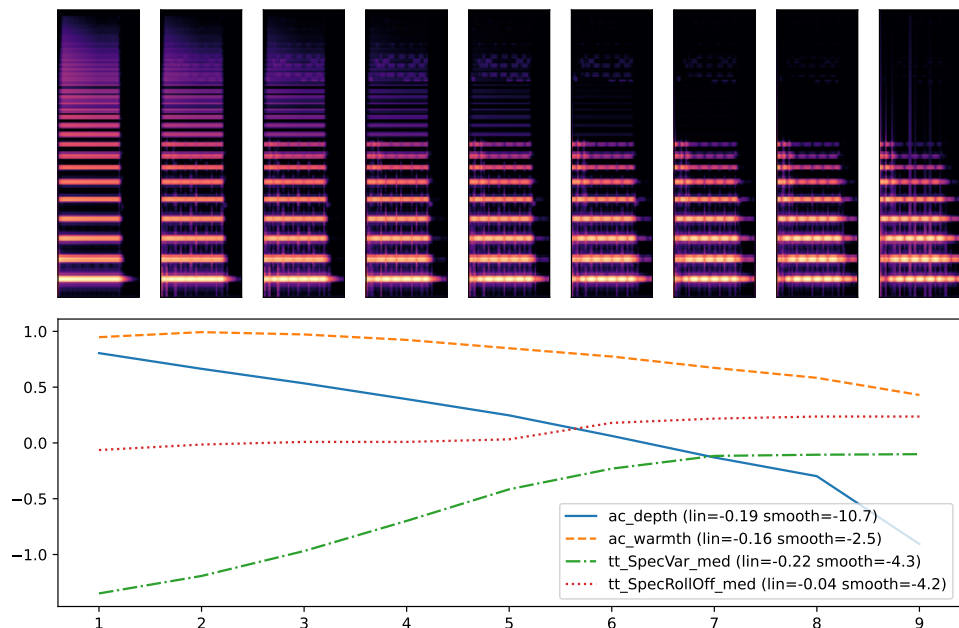


Figure 3.8. Computation of four timbre features and the associated linearity and smoothness metrics. The linearity and smoothness values are designated by *lin* and *smooth*, respectively. Morphing performed by SMT [183].

cedure. The Zynaptic Morph⁵ commercial solution has also been excluded for several reasons. In addition to being closed-source and proprietary, the generation of morphings is not easily automated, and no proper technical documentation is available. The Zynaptic company holds a patent [208] related to a smooth time-frequency transform, which may be a building block for their Morph plugin. This transform seems somewhat similar to STRAIGHT [188] representations, which are included in the following results.

Models with prohibitively long synthesis times, such as WaveNet [71] (Subsection 1.4), have also been excluded. Similar faster models, such as RAVE [82] for direct raw waveform generation, have been favored.

⁵<https://www.zynaptiq.com/morph/>

The final list of models considered in this subsection contains seven models, categorized into traditional signal processing techniques and data-driven approaches. The first category includes Spectral NMF [195], which decomposes sounds into spectral bases and activations; STRAIGHT [188], which computes pitch-adaptive smooth spectral envelopes; and SMT [183], based on additive analysis-resynthesis. The second category includes CQT-VAE [85], a variational autoencoder trained to reconstruct CQT time frames; RAVE [82], a convolutional autoencoder for raw audio synthesis; DDSP-HpN [95], a differentiable additive synthesizer with harmonic plus noise modeling; and DDSP-wavetable [107], which resynthesizes sounds using learned wavetables.

Test dataset

Most studies from Section 3.1 evaluate the proposed algorithms using between one and five examples (Subsection 3.2.1). The most extensive analysis [187] is based on 26 pairs of quasi-harmonic sounds from three families of musical instruments: woodwind, brass and string.

In order to compare multiple techniques from Section 3.1, a larger dataset with a broader range of timbre is more appropriate. The dataset introduced in Section 2.2, containing thousands of sounds generated from a FM synthesizer, is well-suited for this task. Conveniently, this dataset also contains the preset corresponding to each sound, which will be useful in the following chapters.

In Chapter 2, 20% of the whole dataset was reserved for testing. From this point onwards in this thesis, the portion of test samples is reduced to 10%, in order to lower both the computational cost and storage space required for evaluating morphings. Using consecutive pairs of presets from this shuffled test set with 3030 audio files, 1515 morphings can be generated. The number of interpolation steps has been arbitrarily set to $T = 9$. This value seems to provide a good tradeoff between resolution (number of steps T) and computational resource usage for evaluating linearity and smoothness.

Implementations and hyper-parameters

Most morphing algorithms use a 48kHz sampling rate and outputs were stored as 32-bit floating point WAV files, requiring approximately 10.5GB of disk

space for 1515 sequences with $T = 9$ steps and 4s-long audio files. The only exception is **SMT**, which had to run at 44.1kHz in order to properly reconstruct the pitch of input sounds.

The morphed outputs were then downsampled to 16kHz, which reduces the duration of timbre features evaluation for a single model to approximately 2.5 hours on a 24-core (48 threads) AMD 3960X **CPU**. Outputs were finally re-encoded as 16-bit WAV files, reducing disk space usage to approximately 3.5GB per model.

Several open-source implementations worked out of the box without further configuration. For instance, a recent **STRAIGHT** implementation⁶ automatically detects anchor points in the smoothed spectrograms. Conversely, **SMT** had enormous computational requirements and was not immediately usable. The *spectral oversampling* parameter had to be reduced from 4 to 2; otherwise, the analysis needed more than the 32 GB of RAM available on the testing machine. All input and output audio waveforms have also been constrained to 4s in length, which enabled an acceleration of the analysis process by modifying parts of the Matlab code. With these optimizations, the 1515 **SMT** sequences could be computed in approximately three days.

Multiple differentiable models (Subsection 3.1.3) are suitable for morphing by performing linear interpolations in the latent space. Some PyTorch implementations of neural synthesizers or spectral autoencoders were ready to use. Others required modifications or custom developments, as well as some hyperparameter tuning, for this experiment. The set of Python scripts and modules for training the models and computing interpolations is provided as a GitHub repository⁷.

Neural synthesizers training

The dataset is divided into 80% for training (24k sounds), 10% for validation and 10% for test (3k sounds each). Models were trained on an NVIDIA RTX 3090 **GPU**. Training a single **DDSP HpN** model [95] was relatively fast, taking eight hours. Training a differentiable wavetable synthesizer [107] or a **VAE** for

⁶https://github.com/HidekiKawahara/legacy_STRAIGHT

⁷<https://github.com/gwendal-lv/neural-synths-morph>

CQT reconstruction [85] required around one day, while RAVE [82] took nine days.

3.3.2 Results

Morphing performance

Tab. 3.1 and Tab. 3.2 report the performance of seven tested morphing techniques, in terms of linearity and smoothness. For techniques with variants or tunable hyper-parameters, only the best configuration’s performance is included in the results. For instance, NMF-based morphing [195] is supposed to be computed in parallel for the harmonic and percussive spectra obtained by HPSS. However, the straightforward variant without HPSS performed better. Morphing using RAVE [82] is better when not using the adversarial fine-tuning. For the differentiable wavetable synthesizer auto-encoder [107], 10 wavetables (rather than 20) enables the best overall morphing performance.

The values of these objective linearity and smoothness metrics depend heavily on the timbre feature, making absolute values challenging to visualize, compare and interpret. Therefore, all linearity and smoothness results in Tab. 3.1 and Tab. 3.2 are expressed as variations relative to SMT [183], which has been arbitrarily chosen as the baseline. A positive variation indicates an improvement in interpolation quality, while a negative variation indicates degradation. The best possible variation is +100%, which corresponds to linearity and smoothness metrics reaching an optimal value of zero. A -100% variation indicates metric values that are double those of the SMT baseline (degraded by a factor of two).

All techniques, including SMT, occasionally fail. For instance, pitch estimation issues, numerical instabilities and algorithmic errors have been observed when conducting this experiment, and previously by other authors [197]. In particular, SMT failed for 63 out of 1515 test sequences (4.2%). For these sequences, the output is null samples and the corresponding smoothness is a perfect 0.0 which cannot be used as baseline value. These failed sequences were excluded from the smoothness results (Tab. 3.2) but retained in the linearity results (Tab. 3.1).

Table 3.1. Linearity statistics for multiple morphing techniques applied to the test dataset. Mean results include the `std` across the 37 timbre features. The number of timbre features with significantly improved or degraded linearity (Wilcoxon signed-rank test) is also reported.

Model	Mean (\pm <code>std</code>) % (\uparrow)	Median % (\uparrow)	Improved features (\uparrow)	Degraded features (\downarrow)
Spectral NMF [195]	-27.5 (\pm 54.7)	-13.5	9	25
STRAIGHT [188]	+7.9 (\pm 30.7)	+10.2	13	14
SMT [183] (baseline)	0	0	0	0
CQT-VAE [85]	-1.2 (\pm 41.4)	+8.7	13	19
RAVE [82]	+0.9 (\pm 22.7)	+5.0	5	24
DDSP HpN [95]	+2.8 (\pm 23.8)	+1.7	8	27
DDSP wavetable [107]	+0.5 (\pm 24.8)	+1.4	7	28

Table 3.2. Smoothness statistics for multiple morphing techniques.

Model	Mean (\pm <code>std</code>) % (\uparrow)	Median % (\uparrow)	Improved features (\uparrow)	Degraded features (\downarrow)
Spectral NMF [195]	-45.8 (\pm 166.2)	+16.1	19	14
STRAIGHT [188]	+9.2 (\pm 28.5)	+15.6	18	15
SMT [183] (baseline)	0	0	0	0
CQT-VAE [85]	-26.6 (\pm 61.7)	-13.9	12	22
RAVE [82]	-1.1 (\pm 39.6)	+10.9	15	18
DDSP HpN [95]	+15.8 (\pm 20.6)	+20.2	25	9
DDSP wavetable [107]	+13.8 (\pm 18.9)	+12.8	23	8

Results from Tab. 3.1 and Tab. 3.2 indicate that CQT-VAE [85] and spectral NMF [195] clearly underperform compared to SMT. RAVE morphing is on average similar to SMT in terms of linearity and smoothness, while the median linearity and smoothness are slightly better. However, the linearity and smoothness of most individual timbre features are actually degraded compared to SMT (Wilcoxon signed-rank tests, p-value < 0.05). STRAIGHT morphing enables the best mean and median linearity among all tested techniques, and an improved smoothness compared to SMT. The two DDSF synthesizers perform similarly and clearly provide the best smoothness. Their mean linearity is relatively similar to that of SMT, although most individual features evolve significantly less linearly.

Detailed morphing linearity and smoothness for all timbre features are presented in Appendix F. However, they remain quite difficult to interpret. They do not clearly indicate that specific subsets of features (e.g., those related to harmonicity or temporal events) are consistently improved or degraded by different models.

Relying on results from Tab. 3.1 and Tab. 3.2 only, the best technique could be either STRAIGHT or the HpN DDSF additive synthesizer. However, other factors can be taken into account and are discussed in the following paragraphs.

Reconstruction of start and end sounds

As indicated in Subsection 3.2.1, a large majority of published morphing techniques do not provide objective evaluations of the results. Most of these works also don't indicate whether the start and end (source and target) sounds can be properly reconstructed by the underlying analysis and resynthesis system. The following paragraphs argue that this is an important criterion which should not be neglected.

In the literature, morphing is often referred to as hybridization. However, if an analysis-resynthesis technique cannot reproduce the timbre characteristics of the two input sounds, then it is highly unlikely to generate proper hybrid intermediate timbres. Nonetheless, a morphing sequence using inaccurately reconstructed start and end samples could still be perceived as very smooth, with objective smoothness values close to an ideal 0.0. The linearity metric could be moderately good if timbre features are not too far from the ideal

affine trajectories (Fig. 3.7). Linearity and smoothness metrics alone appear insufficient to fully quantify the quality of a given morphing technique

Hence, the reconstruction accuracy of all morphing models should be considered. Fig. 3.9 reports the 13-band MFCCD between ground truth sounds from the test dataset and their reconstruction by each model, assuming that the threshold for perceived similarity is between 10 and 15 (Subsection 2.1.3). Fig. 3.9 also presents PEMO-Q [209] evaluations of the perceived quality difference between reconstructions and original sounds. In particular, the PEMO-Q Objective Difference Grade (ODG) quantifies a perceptual quality impairment from 0 (*imperceptible*) to -4 (*very annoying*).

Regarding MFCCD, considered as a general measure of timbre dissimilarity [138], only SMT has a median value below the 10.0 threshold. HpN DDSP is the second best performer with a median MFCCD slightly above the 15.0 threshold. Notably, among the seven techniques considered, these two are the only ones that rely on additive synthesis. The MFCCD values for other techniques are primarily distributed above the 10–15 range. However, PEMO-Q evaluations indicate that SMT is much more accurate than HpN DDSP. PEMO-Q is based on a psycho-acoustical model of auditory perception which analyzes signals using multiple filtering steps. This leads to finer representations than MFCCs, and enables PEMO-Q distances to take smaller differences into account.

In Fig. 3.9, considering only sounds that are mostly *harmonic*, the median and average ODG for HpN DDSP are -2.94 and -2.82 , respectively. These are close to an *annoying* signal degradation. For SMT, the average and median ODG are -1.35 and -0.76 , respectively, i.e. *perceptible but not annoying*. This large difference between the two models is consistent with MFCCD results. It may come from the location of partials on the frequency axis. They are constrained to a purely harmonic structure with HpN DDSP, whereas SMT can handle subtle frequency deviations of the harmonics [183]. Audible artifacts are also often present in sounds generated by DDSP models, which are likely to substantially reduce the PEMO-Q ODG results.

Regarding the other *non-harmonic* sounds in Fig. 3.9, the MFCCD is generally similar to that of the entire dataset. A notable exception is SMT, which is the best model for harmonic sounds but is also clearly the worst for percussions or sound effects. Sinusoidal analysis-resynthesis appears to struggle with

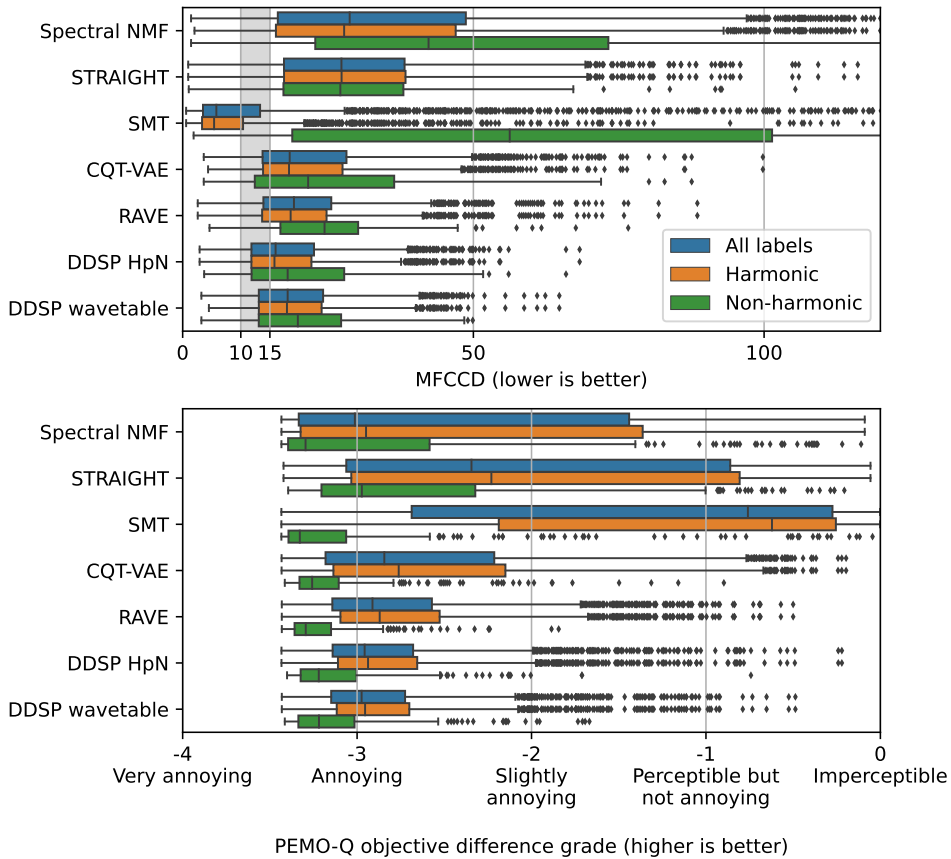


Figure 3.9. Reconstruction accuracy of test dataset samples (3030 sounds). *All labels* corresponds to the entire test dataset. *Harmonic* only includes the 2746 sounds (90.6%) identified as such after HPSS analysis (Subsection 2.2.2), while *Non-harmonic* include others.

reconstructing sharp transients and partials with quickly varying frequencies. Moreover, the analysis relies on pitch estimation, although a F_0 cannot always be identified with unpitched sounds. PEMO-Q ODG scores, however, show that all models provide worse reconstructions of *non-harmonic* sounds compared to *harmonic* sounds.

Computational requirements

When computing morphing with different methods, large discrepancies in computation times were observed and are reported in Tab. 3.3. RAVE, which relies solely on CNNs, is the fastest, as claimed in the original paper [82]. DDSP-based models allow for fast synthesis, but the analysis is quite intensive, requiring approximately 4 seconds for two 4-second input sounds. The computational bottleneck lies in the slow F_0 estimation performed by the CREPE [97] CNN. Spectral techniques (NMF and CQT-VAE) are moderately fast for both analysis and resynthesis, whereas SMT and STRAIGHT are substantially slower. SMT analysis, particularly the detection and matching of partials, incurs very high computational costs. However, resynthesis is faster than real-time.

Table 3.3. Computational requirements for analysis and resynthesis of 4 seconds of 48kHz audio for all models, except SMT which had to run at 44.1kHz, using an Intel Core i7-10700KF (8-core, 3.80GHz) CPU. Analysis durations include the processing of the two input sounds, while resynthesis durations correspond to a single output sound.

Model	Analysis	Interpolation and synthesis
Spectral NMF	1.23 s (± 0.64)	2.03 s (± 0.137)
STRAIGHT	13.3 s (± 2.4)	1.11 s (± 0.18)
SMT	155.5 s (± 19.5)	0.194 s (± 0.130)
CQT-VAE	0.403 s (± 0.140)	1.78 s (± 0.09)
RAVE	0.099 s (± 0.005)	0.063 s (± 0.002)
DDSP HpN	3.92 s (± 0.53)	0.072 s (± 0.001)
DDSP wavetable	3.92 s (± 0.53)	0.046 s (± 0.001)

3.3.3 Discussion

Determining the best technique among these seven is challenging. **NMF** and **CQT-VAE** clearly underperform compared to the others, struggling with test sound reconstruction and producing less smooth and linear morphings. While **STRAIGHT** achieves the most linear morphings, it also exhibits one of the highest **MFCCD** reconstruction errors. Thus, **STRAIGHT** cannot be considered as the best technique.

DDSP-based morphing is both smoother and more linear than **SMT**-based morphing. It is also orders of magnitude faster, for both analysis and resynthesis. Wavetable synthesizers are usually considered appropriate for morphing, by interpolating intermediate wavetables [107]. However, the differentiable wavetable model does not objectively perform better than the **HpN** counterpart. **HpN DDSP** could be considered as the best morphing technique, in terms of objective linearity and smoothness. However, it also fails to accurately reconstruct sounds from the test dataset.

Finally, **SMT** seems to provide the best overall compromise, although the computational resources required for analysis are substantial. The second best compromise could be the differentiable **HpN** synthesizer. However, the reconstruction accuracy is subpar. It could be improved with better models and training dataset (Subsection 1.4.4), but this is left for future work. Remarkably, these two best models are based on additive resynthesis. Unfortunately, neither can properly reproduce non-harmonic sounds, which represent approximately 10% of the test dataset.

3.4 In brief

Summary of Chapter 3

- A review of the literature on traditional sound morphing algorithms based on signal processing, as well as morphing using differentiable synthesizers, has been presented.
- The lack of objective evaluations in the literature has been highlighted. Based on relevant studies about timbre and interpolation in general, a method has been introduced and enables the computation of the linearity and smoothness of morphings.
- Identifying the best technique is difficult. However, the best trade-off between morphing linearity, morphing smoothness and reconstruction accuracy is [SMT](#), based on additive analysis-resynthesis.

Perspective for Chapter 3

- [SMT](#) will be used as a baseline for experiments presented in the following chapters.
- The objective evaluation method is provided as a Python toolbox but remains computationally intensive and could be optimized, a task left for future work.
- Correlations between subjective evaluations of morphings and the objective smoothness and linearity introduced in Section 3.2 have not yet been presented. They will be reported and discussed in Chap. 5.

Chapter 4

Latent space interpolation of presets

Contents

4.1	Non-linear preset interpolation	123
4.2	Transformer-VAE Model	125
4.2.1	VAE for synthesizer presets	126
4.2.2	Transformer encoder and decoder	127
4.2.3	Additional spectrogram decoder	133
4.2.4	Output probability distributions	135
4.2.5	Regularization using timbre attributes	137
4.2.6	Training procedure	139
4.3	Results	140
4.3.1	Latent interpolation	140
4.3.2	Objective morphing performance	141
4.3.3	Reconstruction of start and end sounds	145
4.3.4	Computational requirements	147
4.3.5	Discussion	147
4.4	Other practical applications	149
4.4.1	Blending multiple presets	149
4.4.2	Extrapolation	150
4.4.3	Preset modulation	151
4.4.4	Other conventional synthesizers	153
4.5	In brief	154

This chapter is based on the following publications:

- G. Le Vaillant and T. Dutoit, "Synthesizer Preset Interpolation Using Transformer Auto-Encoders," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greece, 2023, doi:[10.1109/ICASSP49357.2023.10096397](https://doi.org/10.1109/ICASSP49357.2023.10096397)
- G. Le Vaillant and T. Dutoit, "Latent Space Interpolation of Synthesizer Parameters Using Timbre-Regularized Auto-Encoders," *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 32, pp. 3379-3392, 2024, doi:[10.1109/TASLP.2024.3426987](https://doi.org/10.1109/TASLP.2024.3426987)

This chapter presents a new approach for the latent interpolation of synthesizer presets, using a model called **SPINVAE**. Traditional sound morphing techniques are tied to specific synthesis methods and produce suboptimal results when applied to diverse acoustic sources. In contrast, **SPINVAE** allows for flexible, model-agnostic interpolation of synthesizer presets, treating synthesizers as black-box systems.

The chapter begins with a discussion on the limitations of existing morphing techniques and the motivation behind developing **SPINVAE**. It then introduces the Transformer-based [14] **VAE** architecture, highlighting its key components and advantages for modeling synthesizer presets. An evaluation of the model’s interpolation performance is also presented, using objective timbre linearity and smoothness metrics, as well as a comparison with state-of-the-art waveform morphing introduced in Chapter 3. The chapter concludes by exploring additional practical applications of **SPINVAE**, including blending multiple presets, extrapolations, and dynamic modulation of sounds.

The chapter’s two main contributions are:

- The **SPINVAE** model, combining the preset and audio modalities into a latent space which enables interpolations of presets;
- The first successful use of multi-head attention neural networks for modeling synthesizer presets.

4.1 Non-linear preset interpolation

The primary limitation of traditional sound morphing techniques introduced in Subsection 3.1.2 is that each is tied to a particular synthesis method. This restricts their use by users who select synthesizers based on distinct audio characteristics. Additionally, specialized sound models often produce subpar results when applied to different acoustic sources (Subsection 3.3.2, and [183]). Sound morphing algorithms also often depend on computationally intensive analysis and resynthesis, as noted in the original publications [190, 191] or observed in practice (Tab. 3.3). Furthermore, many sound morphing methods are not easily adaptable for playing an intermediate sound using any **MIDI** note [189–192, 194, 195], despite this being one of the most fundamental features of any synthesizer. Morphed waveforms could be played using samplers,

although they would be challenging to modify in musical terms or to control expressively in real time [65]. A notable exception is the Kyma [182] implementation of Loris [175], which addresses both issues. However, it still requires tedious manual annotations of all dataset samples and relies on proprietary DSP hardware.

The synthesis techniques mentioned in Subsection 3.1.2 were chosen for morphing due to their ability to produce smooth transformations of sounds. For example, [188] performs resynthesis using smooth time-frequency representations, while [186, 187] interpolate between smoothed spectral envelopes. Similarly, morphing based on additive resynthesis [183] using matched partials is inherently smooth. Conversely, conventional synthesizers provide some non-differentiable and non-linear synthesis controls. These include categorical waveform selectors, discrete routing options for low- or audio-frequency signals, and non-linear distortion parameters, among others.

In this chapter, the aim is to develop a model that allows for interpolation of presets for any synthesizer (Figure 4.1). To create a generalizable model, synthesizers are treated as non-differentiable black boxes, with presets comprising both numerical and categorical parameters. The current default approach involves computing a linear interpolation for each synthesis parameter, where a *macro-control* guides the transition from the initial preset to the target preset. This method can produce smooth transitions if synthesis controls are aligned with perceptual scales, such as logarithmic scales for frequencies and amplitudes. It can be well-suited for additive and subtractive synthesizers composed of independent oscillators and filters, provided these oscillators and filters serve a similar function in both presets. However, in practice, a perfect correspondence between presets is rare [176], and interpolation in parameter space does not necessarily result in a perceptually smooth morph [202].

For FM synthesis in particular, automatic programming is a notoriously difficult problem (Chap. 2), and preset interpolation is unlikely to be any easier. A specific example of morphing with FM synthesis has been found in the literature [210]: using a basic single-carrier and single-modulator FM, a brass sound can be morphed to a percussion using simple linear parametric interpolation. However, others examples, quantitative evaluations or a general model for FM preset interpolation are not presented. Native Instruments' FM8 synthesizer claims that it allows "morphing between the timbral characteristics of four FM8 sounds" [211]. However, the morphing is limited to "new FM8 sounds"

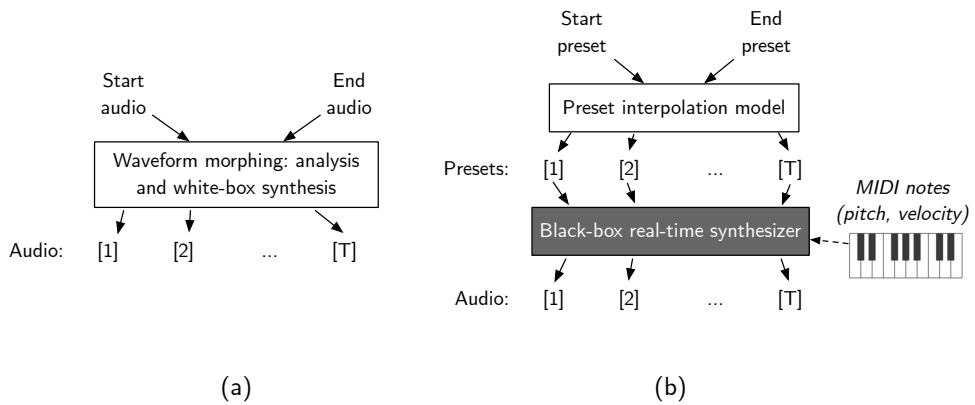


Figure 4.1. Block diagrams of classic sound morphing techniques with dedicated synthesis engines (a) and preset interpolation for a black-box conventional synthesizer (b). *Start* and *End* items are also sometimes referred to as *Source* and *Target* in the literature.

and excludes synthesis parameters such as modulations and envelopes. Additionally, neither the interpolation method nor quantified morphing quality measurements provided. For DX7 FM parameters, the linear interpolation occasionally works, but it fails more often than not, as shown in Subsection 5.1 in the next chapter.

More generally, a non-linear method, applicable to any black-box synthesizer, would be useful. Modern neural networks could learn how to perform this task. VAEs [167, 212] or adversarial auto-encoders [199] can be trained to interpolate between reconstructed audio outputs. However, these approaches rely on differentiable decoders to produce sound, while this chapter focuses on preset interpolation for an external sound synthesizer.

4.2 Transformer-VAE Model

This section presents building blocks of the SPINVAE model for black-box synthesizer preset interpolation (Fig. 4.1(b)). Its components have been selected based on both theoretical benefits and measured performance on the

validation dataset of DX7 FM sounds. An ablation study will be presented in Section 5.2 and will formally validate the effectiveness of SPINVAE components introduced in this section.

4.2.1 VAE for synthesizer presets

VAEs exhibit useful latent space properties, as described in Subsection 2.3.1, which make them particularly well-suited for data interpolation tasks. By interpolating between latent codes, VAEs can generate smooth transitions between data points, enabling effective and meaningful transformations. The structure of a basic VAE for presets (denoted \mathbf{u}) is provided in Fig. 4.2. The corresponding training loss, to be minimized, is:

$$\mathcal{L}(\mathbf{u}) = \underbrace{-\log p(\mathbf{u}|\mathbf{z})}_{=\mathcal{L}_{\text{Preset}}} + \beta \underbrace{D_{KL}[q(\mathbf{z}|\mathbf{u})||p(\mathbf{z})]}_{\mathcal{L}_{\text{DKL}}}, \quad (4.1)$$

where $p(z) = \mathcal{N}(0, I_{L_Z})$ and $q(\mathbf{z}|\mathbf{u}) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2)$, as discussed in Subsection 2.3.1. The choice of $p(\mathbf{u}|\mathbf{z})$, however, is not straightforward and strongly influences both the reconstruction loss and the overall model’s performance. Appropriate decoder output distributions dedicated to preset modeling will be introduced in Subsection 4.2.4.

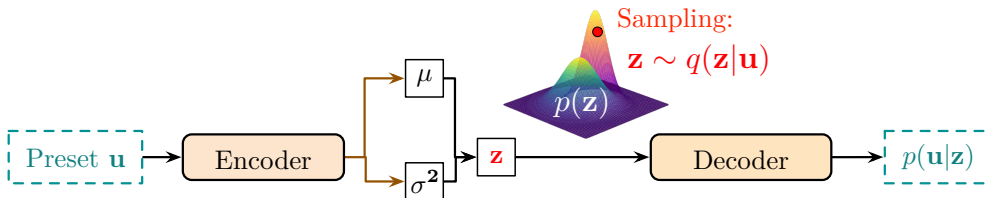


Figure 4.2. Basic VAE for synthesizer presets.

After training, only the decoder is required for interpolation, and sampling a preset $\mathbf{u} \sim p(\mathbf{u}|\mathbf{z})$ from the decoder’s output distribution is always straightforward. In practice, presets are often obtained by selecting the most probable

value, such as the mean of a Gaussian distribution or the maximal logit of a softmax distribution, rather than through actual sampling.

An important question that arises here is the number of latent space dimensions, and its relationship to the number of synthesis parameters. Embeddings dedicated to data compression may not be suitable when the main focus is on altering and transforming the audio signal [213]. Hence, the size L_Z of latent codes \mathbf{z} is not necessarily smaller than or equal to the size of the input preset. For instance, when modeling DX7 presets with 144 parameters, the model introduced in this chapter relies on a latent size of $L_Z = 512$. Such a VAEs is called overcomplete [67, Chap. 14], and this choice will be extensively discussed in Chap. 5.

4.2.2 Transformer encoder and decoder

Limitations of normalizing flows

Some generative models for sound matching, such as [134, 147] introduced in Chap. 2, use *continuous* normalizing flows [150] for decoding presets. These models can handle continuous numerical synthesis parameters such as *frequency* or *amplitude*. For categorical parameters such as *algorithm* or *waveform type*, one-hot encoding can be used, although it substantially increases the latent size. Categorical flows [214] should be used instead to model these parameters, but these flows cannot handle numerical and categorical outputs simultaneously. Another key limitation of normalizing flows is that the data dimensionality must match the number of latent dimensions. This constraint was considered too restrictive for the experiments conducted in this thesis.

Moreover, numerical instabilities were repeatedly observed during the training of these auto-regressive neural networks. Furthermore, normalizing flows can be replaced by more recent and powerful models, such as Transformers [14], which are also inherently better suited for modeling sequences of synthesis parameters.

The Transformer model

Transformers [14] excel at modeling sequences of values (called *tokens*) due to their self-attention mechanism, which allows them to capture dependencies between tokens across the entire sequence, regardless of the distance. They are the fundamental block of most modern LLMs but can be applied to various types of sequence data such as time series, square patches of an input image [215] or of an audio spectrogram [216], or even protein sequences [217].

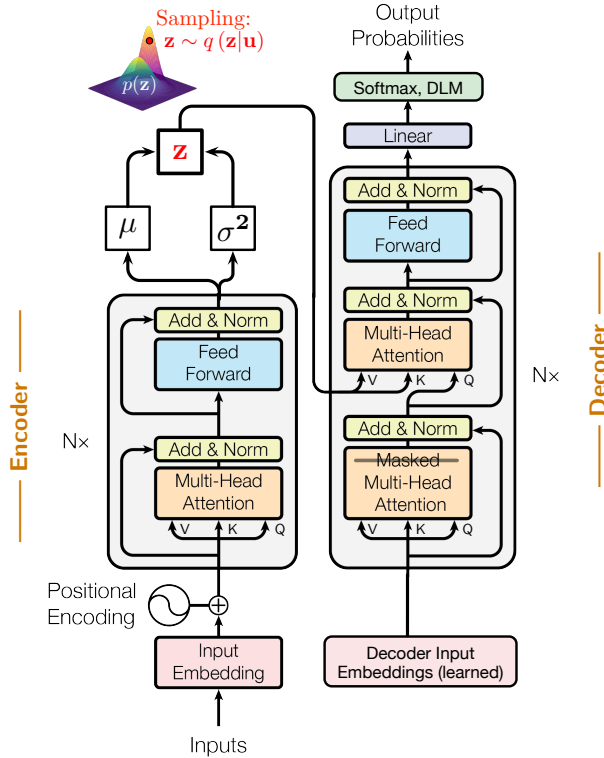


Figure 4.3. Transformer-VAE where inputs are presets \mathbf{u} and outputs are probability distributions $p(\mathbf{u}|\mathbf{z})$. V, K and Q designate the *Values*, *Keys* and *Queries*, respectively. Original figure from [14], modified.

Unlike MLPs where each layer processes inputs in isolation, a single Transformer layer can model the relationships between parameters more effectively, allowing for better handling of temporal or ordered data. Transformers also avoid the vanishing (or exploding) gradient problem [218] that affects RNNs, e.g., LSTM or GRU neural networks, because they do not rely on sequential processing. This ability to attend to every part of a sequence simultaneously makes Transformers particularly well-suited for tasks where parameter interactions and order are crucial, such as in sound synthesis. For instance, the values of tokens corresponding to an FM operator can be processed differently depending on the tokens corresponding to the FM algorithm (which changes the routing of signals) or any other operator.

In the attention mechanism, each token in a sequence is transformed into three vectors: Key, Query, and Value. Query represents the current token asking for relevant information from the others, while Key represents the content of each token that the query will compare against. The attention mechanism computes a similarity score between the Query of a token and the Keys of all tokens. Value holds the actual information of each token that gets passed through. The attention scores are then used to weight the Values of all tokens, determining how much attention a given token should give to others in the sequence. More details about the attention computation can be found in the original paper [14].

As displayed in Fig. 4.3, the original Transformer model can be adapted into a Transformer-VAE by using some output tokens of the encoder to parameterize the posterior distributions $q(\mathbf{z}|\mathbf{u})$. Transformer-based VAEs have already been applied to the somewhat related tasks of symbolic music [219] or 3D motion [205] generation.

Preset encoder

The preset encoder architecture is presented in Fig. 4.4, starting with an embedding layer which generates tokens with a size arbitrarily set to 256. Numerical parameters are processed through a linear projection from \mathbb{R} to \mathbb{R}^{256} . Categorical parameters use a distinct dictionary of embeddings for each synthesizer parameter, e.g., *Algorithm* and *OP1 Mode* have separate dictionaries. Blending numerical and categorical tokens has been done in other fields, such as *chemical language modeling* [217] for generating or clustering molecules.

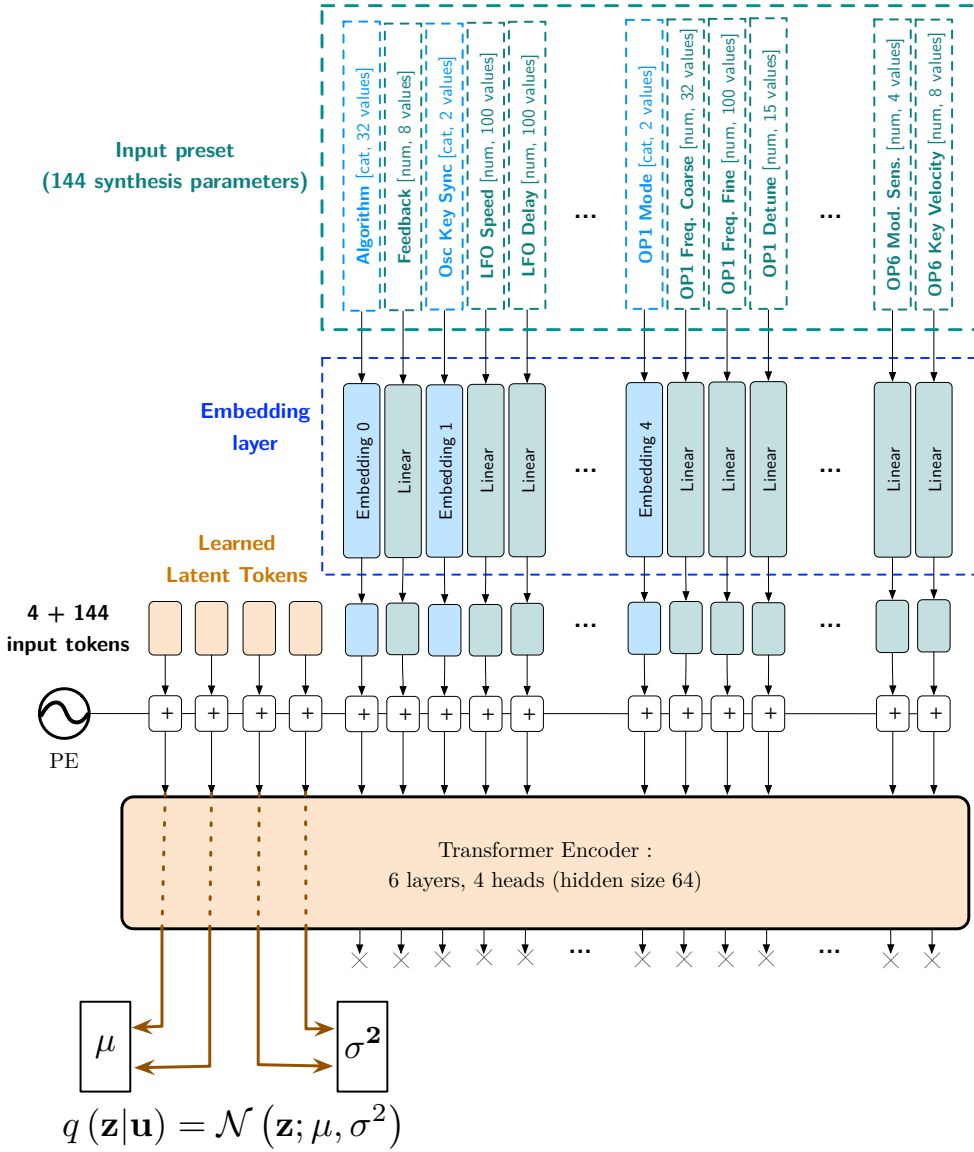


Figure 4.4. Proposed Transformer encoder architecture for modeling synthesizer presets. Hidden tokens have a 256 size, while the latent size is $L_Z = 512$.

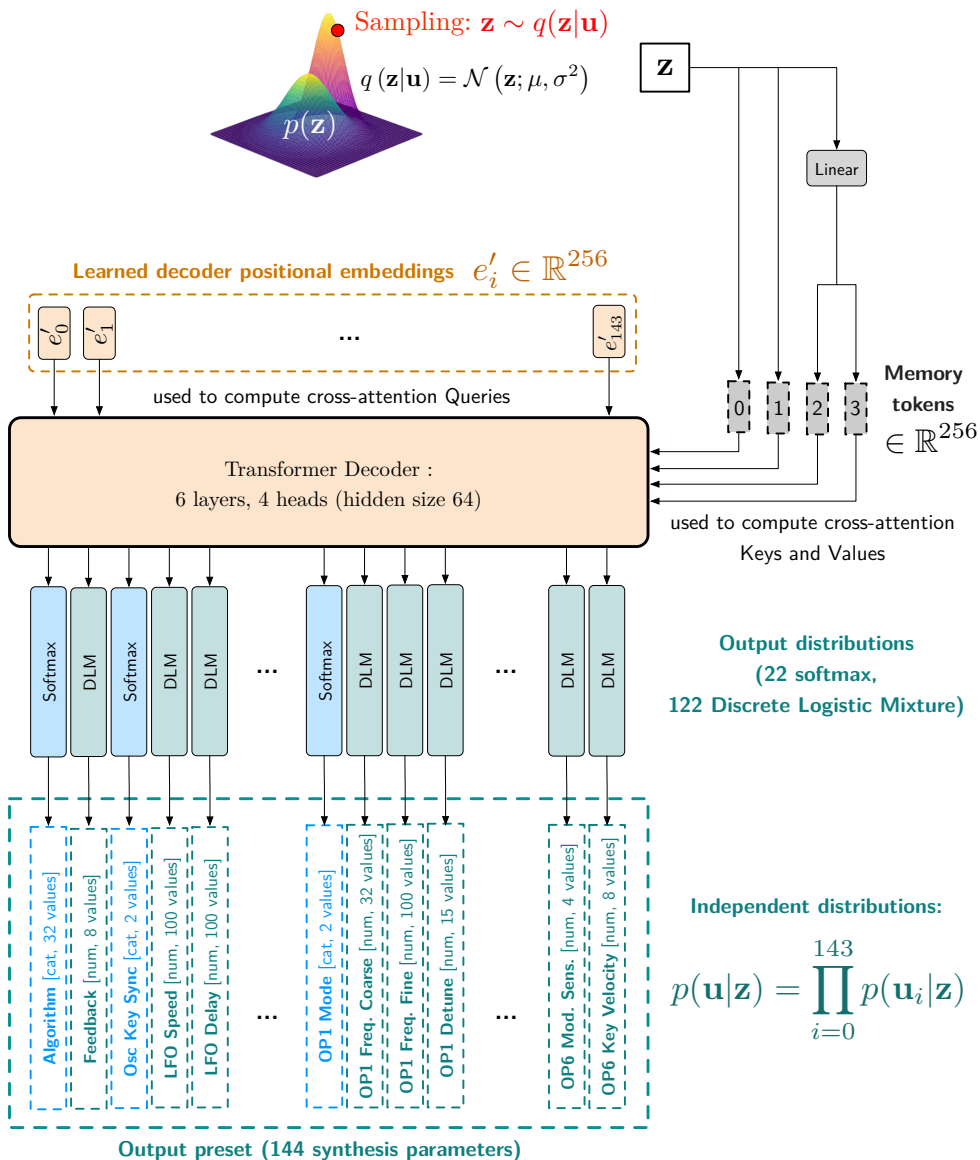


Figure 4.5. Proposed Transformer decoder architecture for synthesizer presets.

This results in a sequence of 144 tokens of size 256. To incorporate positional information, sinusoidal positional encodings are summed as in [14], allowing the attention mechanism to account for the position of each parameter in the sequence. Additionally, a set of latent tokens is appended at the beginning of the sequence. The input latent tokens are learnable vectors of size 256, and their value remains constant after training. They are conceptually related to special tokens used with LLMs, such as <BOS> or <EOS> (beginning of sentence, end of sentence) [220], although their role is different. Output tokens at the same position as these latent tokens are retained, while all other outputs are discarded.

The number of latent tokens depends on the configurable latent size L_Z . For instance, if $L_Z = 512$ as displayed in Fig. 4.4, then 4 latent tokens (size 256) are learned during training and processed by the encoder. The first two encoder output tokens are concatenated to build the $\mu \in \mathbb{R}^{512}$ vector, and the next two are used for $\sigma \in \mathbb{R}^{512}$.

Preset decoder

Details about the preset decoder are provided in Fig. 4.5. Its input is a sequence of learned positional embeddings that are progressively transformed into tokens corresponding to each synthesis parameter. These final tokens are then used to parameterize the probability distributions for each parameter. The input positional embeddings, $\mathbf{e}'_i \in \mathbb{R}^{256}$ in Fig. 4.5, change during the learning process. However, after training, their value remains constant and does not depend on the preset being decoded.

Information about the preset is available through *memory tokens*, comprising the raw latent code \mathbf{z} and a linearly modified version of \mathbf{z} , both reshaped into a set of 256-dimensional vectors. The number of memory tokens is determined by the latent size L_Z , and the same memory tokens are provided to each transformer layer. Meanwhile, the hidden token size remains fixed at 256 across all decoder layers.

In the original Transformer formulation, in the context of natural language processing, triangular decoder attention masks are used for tokens to attend previous tokens only (previous words in a text). This enables a sequential (word-by-word) decoding of text, using techniques such as top-K sampling

[221] or beam search [222]. Sequential decoding is required for tasks such as machine translation or novel text generation, where the length of the output sequence is not known in advance.

For preset modeling, the sequence length remains fixed. Thus, as indicated in Fig. 4.3, decoder attention masks can be discarded. Then, each hidden token can attend to tokens at any position. This allows for a fast decoding of all preset distributions in parallel. Training with triangular attention masks and sequentially decoding synthesis parameters was attempted, but consistently resulted in worse preset reconstructions.

4.2.3 Additional spectrogram decoder

A simple Preset-VAE (Fig. 4.2), composed solely of the Transformer encoder and decoder, aims to minimize the $\mathcal{L}_{\text{Preset}}$ and $\beta\mathcal{L}_{\text{DKL}}$ terms from Eq. 4.1. However, such a model would not have information about how a preset sounds, so this information is unlikely to be encoded in the latent space. Hence, an additional $\mathcal{L}_{\text{Audio}}$ loss encourages a CNN to learn how to generate Mel-spectrograms $\hat{\mathbf{x}}$ from the latent representation \mathbf{z} of a preset (Fig. 4.6). The training loss becomes:

$$\mathcal{L}(\mathbf{u}, \mathbf{x}) = \underbrace{-\log p(\mathbf{u}|\mathbf{z})}_{=\mathcal{L}_{\text{Preset}}} - \underbrace{\log p(\mathbf{x}|\mathbf{z})}_{=\mathcal{L}_{\text{Audio}}} + \beta \underbrace{D_{KL}[q(\mathbf{z}|\mathbf{u})||p(\mathbf{z})]}_{\mathcal{L}_{\text{DKL}}}. \quad (4.2)$$

The audio decoder CNN, associated with the preset encoder, can be considered as a proxy for the real synthesizer. They approximate non-differentiable processes with a differentiable model. Neural proxies have been used previously for modeling audio effects [115, 223] or their gradients [145]. The audio decoder in Fig. 4.6 is a nine-layer CNN with residual connections. Its output distribution $p(\mathbf{x}|\mathbf{z})$ models each spectrogram pixel as a unit-variance Gaussian distribution, which is equivalent to using a per-pixel MSE loss. The audio decoder is only used during training.

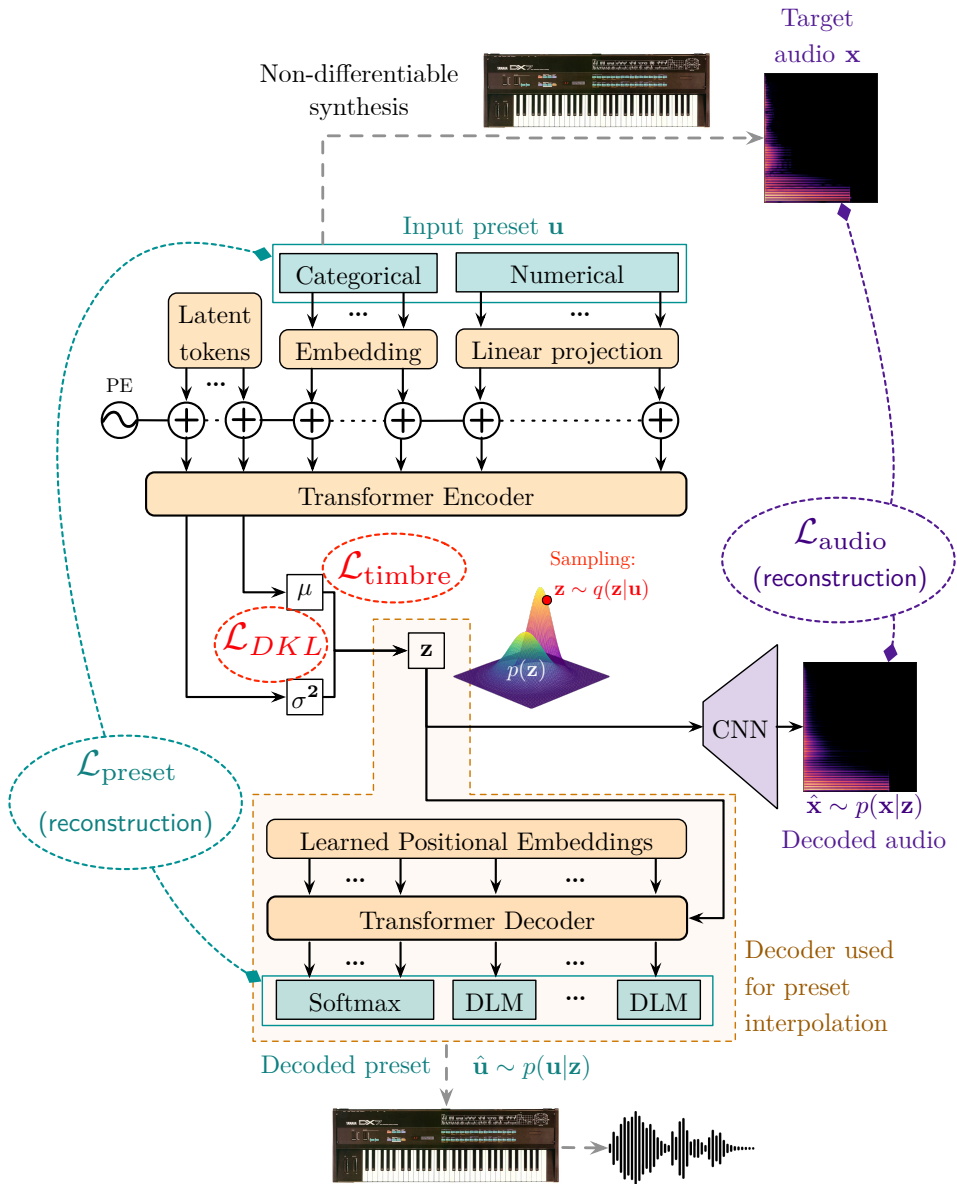


Figure 4.6. Architecture of the SPINVAE model and training losses \mathcal{L} .

4.2.4 Output probability distributions

Discretized Logistic Mixture distributions

Chap. 2 has demonstrated that improvements could be obtained by modeling numerical and categorical parameters separately, for the task of automatic synthesizer programming. Here, in contrast to previous related studies [131, 134, 135, 147], the preset decoder $p(\mathbf{u}|\mathbf{z})$ employs appropriate distinct probability distributions for categorical and numerical synthesis parameters (Fig. 4.5). Categorical outputs are obtained by applying the usual softmax function on each output token. Numerical parameters, however, are often discrete with widespread synthesizers. For instance, a DX7 *amplitude* parameter has 100 possible values whereas *detune* provides a discrete range of 15 semitones (Appendix B.1). The usual Gaussian densities, for which minimizing the NLL is equivalent to minimizing an MSE loss, are not completely appropriate.

Hence, normal distributions for numerical parameters have been replaced by Discretized Logistic Mixture (DLM) distributions, originally published as part of the PixelCNN++ [224] model for integer pixel values in [0..255]. Probabilities are computed using discrete bins, and more probability is typically assigned to the lowest and highest bins. This corresponds to the observed marginal distributions of synthesis parameters' values in the dataset (Appendix D). Three mixture components are used [225], as illustrated in Fig. 4.7. For a value $u_i \in [0..N_i - 1]$ of the i^{th} synthesis parameter, where N_i denotes the number of possible values, the probability mass function $p(u_i|\mathbf{z})$ is:

$$p(u_i|\mathbf{z}; \pi_i, \mathbf{m}_i, \mathbf{s}_i) = \sum_{k=1}^3 \pi_{i,k} \left(S \left(\frac{u_i + 0.5 - m_{i,k}}{s_{i,k}} \right) - S \left(\frac{u_i - 0.5 - m_{i,k}}{s_{i,k}} \right) \right), \quad (4.3)$$

where $\pi_i, \mathbf{m}_i, \mathbf{s}_i \in \mathbb{R}^3$ are values from the i^{th} output token and S is the logistic sigmoid function. For edge cases $u_i = 0$ or $u_i = N_i - 1$, replace $u_i - 0.5$ by $-\infty$ or $u_i + 0.5$ by $+\infty$, respectively. In contrast to the original DLM implemen-

tation [224], the one developed for SPINVAE¹ allows a different N_i for each u_i . I.e., it handles presets whose parameters have different discrete ranges. Log-probability computations remain, nevertheless, parallel and numerically stable.

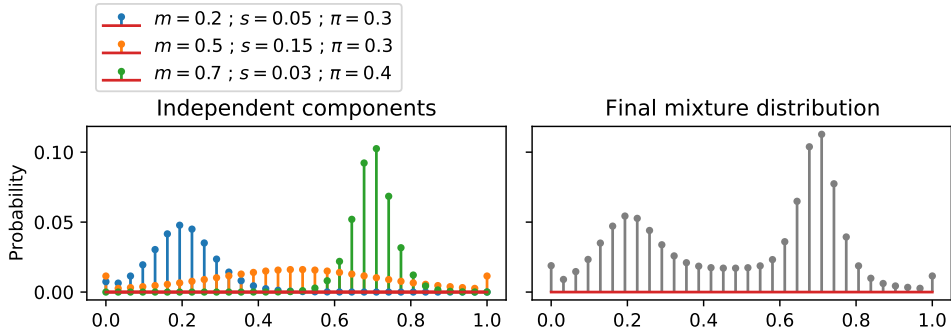


Figure 4.7. DLM with three components for 32 possible discrete values normalized into $[0.0, 1.0]$.

Independent distributions

The parallel decoding of the distribution of each synthesis parameter, without attention masks in the decoder (Subsection 4.2.2), implies that these output distributions are independent. That is, as indicated in Fig. 4.5, $p(\mathbf{u}|\mathbf{z}) = \prod_i p(\mathbf{u}_i|\mathbf{z})$ where \mathbf{u}_i is the i^{th} synthesis parameter.

Alternatively, the decoding could be sequential and based on conditional probabilities, e.g., the algorithm could be decoded first, then parameters for each FM operator could be decoded depending on the most probable algorithm. However, this requires the use of attention masks during training, which hinders the accuracy of preset reconstruction (Subsection 4.2.2). Moreover, the Transformer structure and self-attention mechanism already introduce a high degree of interdependence between hidden tokens. Hence, conditional decoder distributions have not been further considered.

¹<https://github.com/gwendal-lv/spinvae2/blob/main/utils/probability.py>

4.2.5 Regularization using timbre attributes

The loss from Eq. 4.2 encourages the model to learn compact latent representations (through \mathcal{L}_{DKL}) which holds enough information about the preset (through $\mathcal{L}_{\text{Preset}}$) and the corresponding audio spectrogram (through $\mathcal{L}_{\text{Audio}}$) to reconstruct those. Ideally, latent codes \mathbf{z} would contain information about the preset’s timbre. Unfortunately, latent coefficients do not easily relate to perception [84, 206].

However, our goal is to achieve an interpolation that impacts timbre smoothly and continuously. To this end, an additional loss term, $\mathcal{L}_{\text{Timbre}}$ (Eq. 4.4), is introduced during the training of SPINVAE:

$$\mathcal{L}_{\text{SPINVAE}} = \mathcal{L}_{\text{Preset}} + \mathcal{L}_{\text{Audio}} + \beta\mathcal{L}_{\text{DKL}} + \gamma\mathcal{L}_{\text{Timbre}}. \quad (4.4)$$

This term encourages the VAE to align ACTM and TT features², denoted as $\mathbf{a} \in \mathbb{R}^{38}$, with specific dimensions of the latent space. Similar to β , a hyperparameter $\gamma \in \mathbb{R}^+$ controls the amount of regularization. By incorporating this loss, the learning process shifts from purely unsupervised to self-supervised, where attributes are automatically computed instead of being based on human annotation. Still, this does not qualify as fully-supervised learning, since the model is not directly trained for the task of interpolation.

This idea has been implemented in some related works [84, 206, 226, 227] in different ways. For instance, Roche *et al.* [84] use a straightforward MSE loss to align some latent coefficients with human ratings scaled in $[-1; +1]$. Esling *et al.* [206] minimize the Kullback-Leibler divergence between two distributions: the distribution of distances between latent codes, and the distribution of distances in an MDS timbre space obtained from perceptual ratings. Pati and Lerch [226] introduce an Attributed-based Regularization (AR) loss to enforce monotonic relationships between timbre attributes and some latent coefficients. In contrast to other regularization methods [84, 206, 228], AR does not enforce latent values during training. It seems to enable a slightly better overall performance and has been arbitrarily chosen over the first two.

²For regularization, timbre features considered are the 37 features from Subsection 3.2.2 plus the *linear attack time*.

Given a minibatch of M presets, the **AR** loss for the j^{th} latent dimensions relies on two distance matrices $D_{\mathbf{a},j}, D_{\mu,j} \in \mathbb{R}^{M \times M}$. The $D_{\mathbf{a},j}$ matrix contains the differences between values of the j^{th} feature, \mathbf{a}_j , for all items from the minibatch:

$$D_{\mathbf{a},j}(n, m) = \mathbf{a}_j^{(n)} - \mathbf{a}_j^{(m)}, \quad (4.5)$$

with $j \in [0, 37]$ and where $n, m \in [0, M - 1]$ are indices of examples in a minibatch. Similarly, $D_{\mu,j} \in \mathbb{R}^{M \times M}$ denotes the matrix of differences between μ_j from the encoder's output (Fig. 4.6):

$$D_{\mu,j}(n, m) = \mu_j^{(n)} - \mu_j^{(m)}. \quad (4.6)$$

Considering the j^{th} latent dimension only, the loss becomes:

$$\mathcal{L}_{\text{AR}}(\mu_j, \mathbf{a}_j) = \text{MAE}(\tanh(\delta D_{\mu,j}) - \text{sign}(D_{\mathbf{a},j})), \quad (4.7)$$

where δ is a hyperparameter influencing the spread of means μ of the $q(\mathbf{z}|\mathbf{u}; \mu, \sigma^2)$ distributions. Finally, $\mathcal{L}_{\text{timbre}}$ is the average of \mathcal{L}_{AR} terms computed for all regularized dimensions, which might be fewer than the total number of latent dimensions. If the latent size is $L_Z \geq 152$, each timbre feature is used to regularize four dimensions rather than one, which enables a stronger regularization. This factor four corresponds to the 2×2 shape of feature maps provided to the **CNN** decoder's input. The total number of regularized latent coefficients becomes $38 \times 4 = 152$. If $L_Z < 152$, then only 38 latent dimensions are regularized.

4.2.6 Training procedure

Before training the SPINVAE model, a general bi-modal (spectrograms and presets) VAE has been pre-trained. An extra CNN encoder is plugged into the model, and its μ_{CNN} , σ_{CNN} outputs are summed with the μ and σ vectors from Fig. 4.6. First, only the CNN encoder and decoder (forming a spectral VAE) are trained, from a merged dataset of Mel-spectrograms from NSynth notes [73] (714 different instruments), 2.2k Surge synthesizer³ patches and 24k Dexed presets (80% of the dataset from Chap. 2). Second, the bi-modal VAE is trained using the Dexed presets and corresponding sounds. Even though it presents poor performance, it is able to approximately reconstruct presets and spectrograms. The weights of embedding, encoder and decoder layers are used as initial weights for the SPINVAE model and all variants presented in the next chapter. Third, fine-tuning is performed without the spectrogram CNN encoder, as depicted in Fig. 4.6. Fine-tuning is relatively fast (approximately 2.5 hours on an NVIDIA RTX 3090 GPU), although the final evaluation of timbre features interpolation quality on the validation dataset requires approximately the same amount of time (2.5 hours on a 24-core AMD 3960X CPU).

Each training step uses β -warmup [229] with the β factor for the Kullback-Leibler divergence in Eq. 4.4 linearly increasing from 10% to 100% of its maximal value over the first 20 epochs. The learning rate is gradually increasing from 5% initially to its maximal 100% value at epoch 20. It is then scheduled to decrease exponentially from 100% at epoch 20 to 10% at the end epoch 220. The maximal learning rate is 4×10^{-4} for transformer blocks, and 2×10^{-4} for CNN blocks. Batch size is set to 128. Other training and implementation details are available on the SPINVAE Github repository⁴.

It is not asserted that the pre-training procedure improves the results. Nonetheless, it clearly reduces the durations required for fine-tuning models, which allows for conducting larger hyper-parameter sweeps. Moreover, splitting the training into several steps introduces a cyclical modification of both the learning rate and β , which may be beneficial. Cyclical learning rate annealing has been empirically shown to improve the performance of differ-

³<https://surge-synthesizer.github.io>

⁴The latest version of the model can be found at: <https://github.com/gwendal-lv/spinvae2>

ent types of models [230]. β annealing can improve the overall performance of VAEs, reducing the posterior Kullback-Leibler divergence (improved latent space regularization) while simultaneously increasing the reconstruction accuracy [229].

4.3 Results

4.3.1 Latent interpolation

In order to compute an interpolation between two presets $\mathbf{u}^{(n)}$ and $\mathbf{u}^{(m)}$, they are first encoded as latent vectors $\mathbf{z}^{(n)} = \mu^{(n)}$ and $\mathbf{z}^{(m)} = \mu^{(m)}$, respectively. This encoding can be considered as an *analysis* of the preset, although no audio is directly provided to the model. Then, a linear latent interpolation produces a series of $\{\mathbf{z}[t], t \in [1, T]\}$ embeddings, where $\mathbf{z}[1] = \mathbf{z}^{(n)}$ and $\mathbf{z}[T] = \mathbf{z}^{(m)}$. Each vector $\mathbf{z}[t]$ is decoded into a preset $\mathbf{u}[t]$, which is eventually programmed into the synthesizer for rendering audio (Fig. 4.6). For results presented in this chapter and the next one, interpolations are made of $T = 9$ steps. The test dataset of 1515 pairs of Dexed FM presets is the same as used in Chap. 3, which will enable a comparison with traditional sound morphing models. As previously mentioned in this thesis, FM synthesis is particularly challenging to handle. Hence, a model that performs well on the Dexed VST plugin can be expected to have broad applicability across conventional synthesizers in general.

Spherical interpolation of latent vectors was considered as a replacement for linear interpolation. The former has been empirically shown to enable better interpolations with generative models such as GANs, which are unlikely to randomly draw samples close to the origin [231]. In contrast to linear interpolations, spherical-interpolated intermediate vectors do not come close to the origin. However, with VAEs, some latent codes can be sampled very close to 0_{LZ} . In practice, spherical interpolation did not seem to show benefits and has not been further considered for experiments.

4.3.2 Objective morphing performance

Objective linearity

As discussed in Section 4.1, preset interpolation with SPINVAE is not directly comparable to sound morphing techniques from Section 3.1. Nonetheless, their audio outputs can be compared using the objective metrics from Section 3.2. Results presented in Tab. 4.1 include three techniques: SMT morphing based on additive analysis-resynthesis, considered in Chap. 3 as the current state-of-the-art technique for sound morphing; the *de facto* linear preset interpolation (Section 4.1); and non-linear preset interpolation using SPINVAE.

Compared to the SMT baseline, the linearity of SPINVAE interpolations is similar. The average timbre feature linearity is 2.2% worse while the median linearity is 0.8% better. This indicates that the *intermediateness* [201] of generated presets is approximately as good as the state-of-the-art SMT, and

Table 4.1. Linearity and smoothness metrics for two preset interpolation techniques and SMT sound morphing applied to the test dataset. Mean results include the `std` across the 37 timbre features. The number of timbre features with significantly improved linearity (Wilcoxon signed-rank test) is also reported. Higher values are better.

Model	Linearity improvement			Smoothness improvement		
	Mean %	Med. %	Improved features	Mean %	Med. %	Improved features
<i>Using SMT as baseline</i>						
SMT	0	0	—	0	0	—
Linear interp.	-34.0 ± 42.2	-29.2	2	-265 ± 198	-242	0
SPINVAE	-2.2 ± 29.6	+0.8	4	-195 ± 150	-165	0
<i>Using the linear preset interpolation as baseline</i>						
SMT	$+18.9 \pm 24.2$	+22.6	35	$+67.0 \pm 17.5$	+72.3	37
Linear interp.	0	0	—	0	0	—
SPINVAE	$+22.4 \pm 10.6$	+22.9	37	$+16.1 \pm 11.2$	+17.9	33

that **SPINVAE** is an adequate method for intuitive creation of hybrid presets. When using the linear interpolation of presets as baseline (lower part of Tab. 4.1), the timbre linearity is 22.4% better on average, and is significantly improved for all considered 37 timbre features.

Detailed per-feature linearity results are presented in Fig. F.5 from Appendix F, but it is quite difficult to precisely explain why some features are more linear with either **SMT** or **SPINVAE**. The only clearly remarkable features are those related to harmonicity, extracted by **TT** [203]: median and **IQR** for F_0 , harmonic energy, inharmonicity, and odd-to-even ratio. These features are consistently more linear with **SMT** than **SPINVAE** used with an **FM** synthesizer. This probably comes from the fine modeling of the partials with **SMT** additive analysis-resynthesis, which enables a precise transformation of harmonic features.

Objective smoothness

Tab. 4.1 also reports the smoothness of the three aforementioned methods. It clearly indicates that preset interpolation is far less smooth than sound morphing. The average feature smoothness score of **SPINVAE** is approximately $3\times$ that of **SMT** (-195% smoothness variation), while the median smoothness is approximately $2.5\times$ worse (-165%). Detailed results are reported in Fig. F.6 from Appendix F.

This outcome was expected, as synthesis engines such as **SMT** are specifically designed for morphing and exhibit very smooth relationships between synthesis parameters and the resulting sound modifications. In contrast, when performing interpolations with a conventional synthesizer like the DX7, the **SPINVAE** decoder may occasionally change a non-differentiable synthesis parameter, such as the algorithm, envelope curve, or waveform type. Therefore, achieving optimal smoothness is likely unattainable.

The lower section of Tab. 4.1 allows for a comparison of the smoothness of the two preset interpolation methods. **SPINVAE** outperforms the default linear preset interpolation, with a 16.1% average and 17.9% median improvements.

Examples

An example in Fig. 4.8 shows the results of two preset interpolation methods (linear and SPINVAE) alongside SMT sound morphing applied to the same start and end presets from the test dataset. The sequence of spectrograms, as well as trajectories of four of the most relevant (Section 5.1) timbre features, are presented. This serves as a good illustration of what is typically observed in the test dataset sequences, highlighting common patterns and behaviors across different techniques. Multiple audio examples are available on an interactive companion website⁵. Additional and more detailed audio examples, presented alongside the evolution of timbre features and the MFCCD and PEMO-Q reconstruction scores, are provided on a separate webpage⁶.

The linear preset interpolation demonstrates an obvious non-smoothness of features, as well as a noticeable deviation from an ideal linear trajectory. For instance, step 2 is completely irrelevant because its spectro-temporal characteristics are quite unrelated to those of the start and end spectrograms. Both linearity and smoothness measure this sudden and undesirable variation.

SMT is clearly much smoother than both preset interpolation methods, although not all features are perfectly linear. E.g., the ACTM *warmth* and TT spectral rolloff deviate from the ideal linear trajectory, and the accumulating errors (for all interpolation steps) reduce the linearity (Eq. 3.2). As measured and explained previously, SPINVAE is also substantially less smooth than SMT. This may be unavoidable when performing interpolations of presets for an inherently non-smooth synthesis process.

Another remarkable observation in Fig. 4.8 is the gradual evolution of F_0 for the SMT morphing between these two pitched sounds. On the one hand, as long as SMT is able to perform F_0 estimation, the F_0 interpolation is very linear and smooth (Figures F.5 and F.6 from Appendix F). On the other hand, both the linear and SPINVAE preset interpolations introduce sudden changes in F_0 . However, these variations are a multiplication of the F_0 by a factor 2 or 0.5, i.e., the musical note remains the same, give or take an octave. This is illustrated in Fig. 3.8 and by examples on the detailed companion website. For instance, in Fig. 3.8, SPINVAE interpolation keeps the same F_0 for steps 1 and

⁵<https://gwendal-lv.github.io/spinvae2>

⁶<https://gwendal-lv.github.io/spinvae2/detailed>

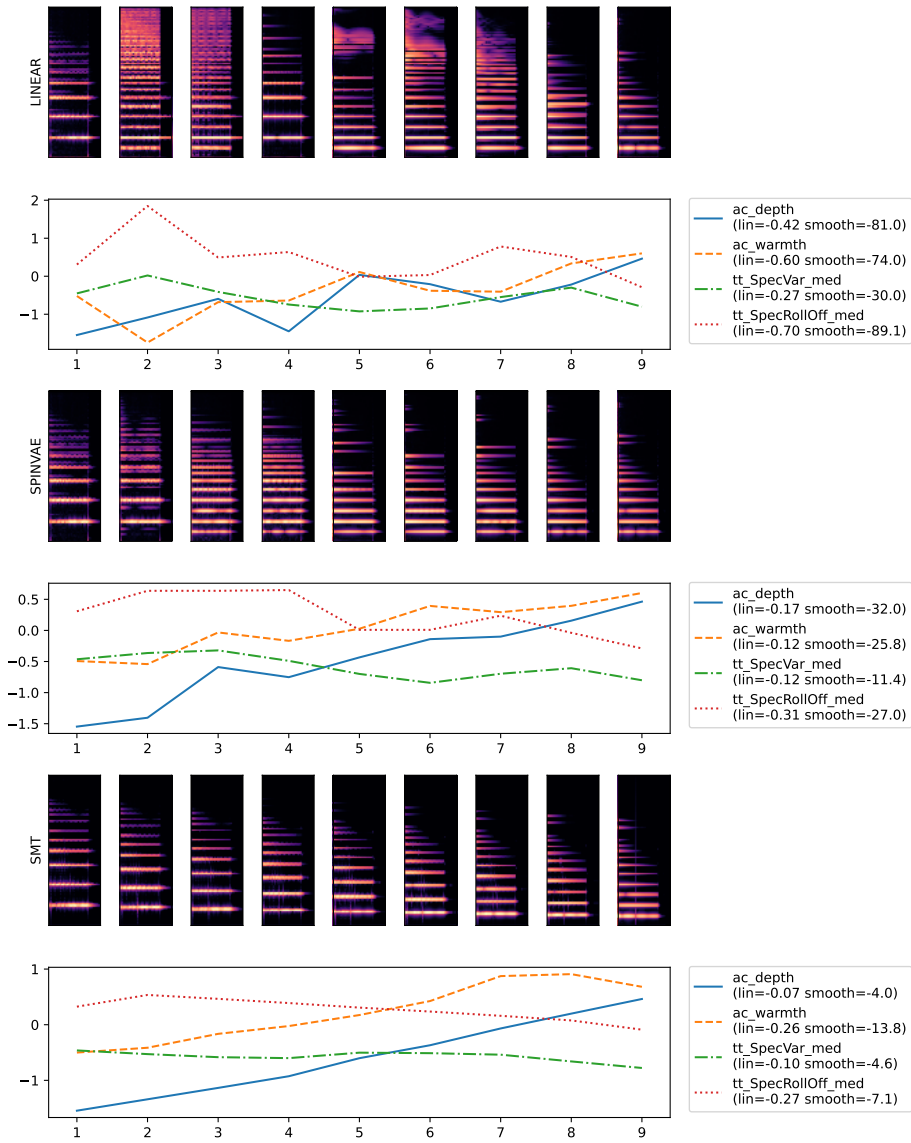


Figure 4.8. Two preset interpolation methods (linear and SPINVAE) and SMT sound morphing applied to the same start and end preset/sound.

2, then halves the F_0 in step 3 (corresponding to moving to the lower octave), and finally keeps the F_0 constant for steps 3 to 9. Regarding the F_0 feature, for creative applications such as live preset interpolation or intermediate preset creation, it is argued that SPINVAE is much more appropriate. When playing a preset from a MIDI interface, the perceived musical note remains the same up to an octave.

Fig. 4.8 also shows that neither SPINVAE nor SMT perfectly reconstruct the start and end sounds, in contrast to the linear preset interpolation which does not rely on analysis-resynthesis (or encoding-decoding). For SPINVAE, the difference is nearly indistinguishable. For SMT, however, clear differences in the middle- to high-frequency spectrum can be observed. Detailed results on reconstruction errors are presented in the following subsection.

4.3.3 Reconstruction of start and end sounds

Similar to the study of sound morphing techniques in Chapter 3, it is crucial to know whether SPINVAE is able to accurately reproduce the start and end presets or not. Fig. 4.9 provides the MFCCD and PEMO-Q ODG results of SMT and SPINVAE for the test dataset. The linear preset interpolation is not reported because it uses ground truth presets as starting and ending points (ideal MFCCD and PEMO-Q ODG values of zero).

Regarding MFCCD for the entire test dataset (all labels), SPINVAE reconstruction is very good for most samples with a MFCCD well below the 10–15 threshold. The median MFCCD is 2.5, while the mean (not reported in Fig. 4.9) increases to 5.9 ± 10.2 because of outliers. Most of SMT reconstructions are also below the 10–15 threshold, although the median MFCCD rises to 5.8. A significant number of samples are very badly reconstructed by SMT, which corresponds to the numerous outliers in Fig. 4.9. Some of these bad reconstructions correspond to the occasional failures observed with most sound morphing algorithms, as described in Subsection 3.3.2. As a consequence, the average MFCCD for SMT is 19.1 ± 36.5 .

According to detailed per-label MFCCD boxplots, SMT also fails to reconstruct most of the non-harmonic sounds. The mean MFCCD for *percussive* and *SFX* sounds are 45.5 ± 46.2 and 79.0 ± 58.1 , respectively, i.e. well above the

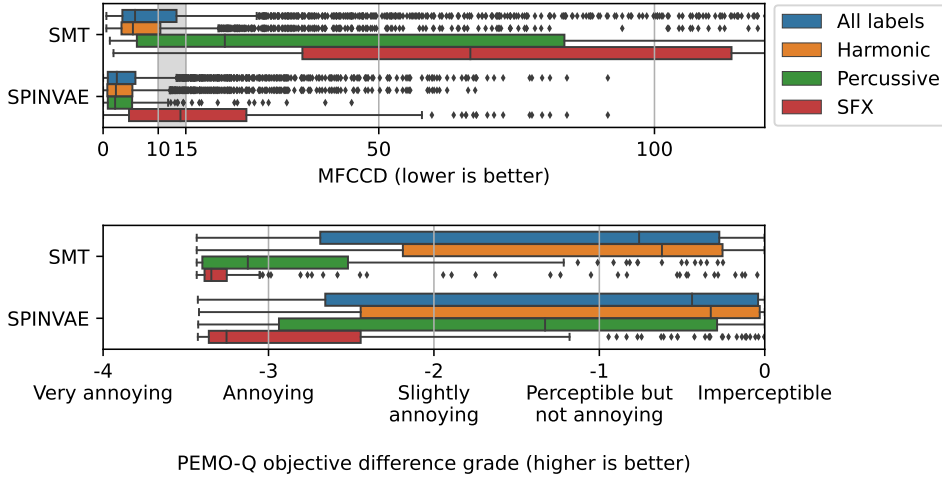


Figure 4.9. Reconstruction accuracy of test dataset samples (3030 sounds) for baseline **SMT** sound morphing and **SPINVAE** preset interpolation.

10–15 threshold. This is likely due to the difficulty in modeling the transients of these sounds using additive plus noise analysis-resynthesis.

In comparison, **SPINVAE** fares much better with an average **MFCCD** for *percussive* and *SFX* items of 5.2 ± 8.0 and 20.4 ± 21.4 , respectively. **SPINVAE** directly encodes some synthesis controls such as attack times and envelope levels. This enables a fine modeling of transients for *percussive* sounds which all share some common spectro-temporal characteristics. *SFX* presets, however, are very diverse and each one is quite unique, such that most *SFX* test presets do not have similar examples in the training dataset. Moreover, they represent about only 5% of the dataset. Hence, **SPINVAE** struggles to properly encode and decode all of these *SFX* presets. The reconstructions of *SFX* sounds are, nonetheless, much better than those of **SMT**.

As for the PEMO-Q results, for **SMT**, the median **ODG** for all labels is between -0.5 and -1.0 , i.e., close to *perceptible but not annoying*. The median **ODG** is in the same range for *harmonic* samples only. For **SPINVAE** and *harmonic* sounds, **ODG** results are higher than -0.5 , i.e., close to *imperceptible*. However, for both methods, the mean **ODG** is approximately -1.0 (*perceptible*

but not annoying). Similar to MFCCD results, the ODG of *percussive* and *SFX* sounds is worse than that of *harmonic* sounds. Using SPINVAE, the median distortion of *percussive* sounds is *perceptible but not annoying*, while the degradation of *SFX* sounds is *annoying*. Using SMT, the distortion is worse and is *annoying* for both *percussive* and *SFX* items.

Regarding FM synthesis parameter values decoded by SPINVAE, not reported in Fig. 4.9, the mean and median accuracy for categorical parameters are 99.86% and 100%, respectively. The mean and median L1 error for numerical parameters (values normalized in [0.0; 1.0]) are 2.33×10^{-3} and 0, respectively. As discussed in Chapter 1, FM synthesis can generate rich spectra but subtle variations in parameter values can lead to large perceived changes in output audio. Therefore, a precise reconstruction of FM presets is needed. These last results and those from Fig. 4.9 indicate that SPINVAE achieves the required level of precision for most presets.

4.3.4 Computational requirements

Tab. 4.2 reports the computational requirements for analysis and resynthesis using different models. Compared to waveform morphing, SPINVAE analysis (preset encoding) and resynthesis (preset decoding and waveform synthesis) are both lightweight. For example, the analysis is roughly 200 times faster than differentiable HpN and 10,000 times faster than state-of-the-art SMT morphing. It is efficient because the dimensionality of a preset is much lower than that of 48 kHz audio. Encoding a preset, performing a linear latent interpolation and decoding presets are rather cheap operations, even though Transformer blocks are used and the model has not been optimized for computational efficiency. Synthesis is also fast because conventional synthesizers are optimized by developers and manufacturers for real-time polyphonic usage. The 12ms synthesis time for Dexed is actually due, for the most part, to the overhead introduced by the slow Python wrapper for VST plugins.

4.3.5 Discussion

Compared to waveform morphing techniques from Chapter 3, the primary advantage of SPINVAE modeling is that it does not rely on a specific synthesis

Table 4.2. Computational requirements for analysis and resynthesis using an Intel Core i7-10700KF (8-core, 3.80GHz) CPU. Analysis durations include the processing of the two input sounds or presets, while resynthesis durations correspond to a single output waveform. Audio sampling rate is 48kHz except for SMT which had to run at 44.1kHz.

Model	Analysis	Interpolation and synthesis
SMT	155.5 s (± 19.5)	0.194 s (± 0.130)
DDSP HpN	3.92 s (± 0.53)	0.072 s (± 0.001)
Linear preset interp.	0	0.012 s (± 0.001)
SPINVAE	0.018 s (± 0.000)	0.050 s (± 0.000)

engine (Fig. 4.1). The model has learned to interpolate presets without being explicitly told that the synthesizer in use is a DX7. As a result, it is widely applicable and has a much wider scope than sound morphing techniques.

When using state-of-the-art SMT morphing as baseline, SPINVAE achieves similar timbre linearity. However, due to the non-smooth and non-differentiable nature of relationships between synthesis controls and output sound, the smoothness is substantially degraded. Nonetheless, for creative applications such as generating an intermediate preset (a single preset in-between two others), the lack of smoothness is not a relevant issue.

In comparison to the linear preset interpolation, which is the only other technique for manipulating presets for a black-box synthesizer, SPINVAE offers greater linearity and smoother transitions. SPINVAE does not always perfectly reconstruct the start and end presets, but the error is most often imperceptible.

4.4 Other practical applications

4.4.1 Blending multiple presets

Results from Section 4.3 demonstrate that the SPINVAE model can perform interpolations between two presets, and its performance for this task has been discussed. Nevertheless, the creation of intermediate presets is not restricted to a blending between one *source* and one *target* preset. Multiple presets can be encoded into the latent space, and a weighted arithmetic mean can be computed on latent codes. This average \mathbf{z} embedding can be decoded into a blended preset.

Graphical interpolation interfaces have been specially developed for linear interpolation between several vectors. The most famous example is the *Nodes* object for Max⁷, shown in Fig. 4.10(a), which can be used to associate one latent vector to each disk on a x/y control surface. The weighted arithmetic average is controlled by the distance between a cursor on the x/y control surface, and the center of each shape. Other interfaces, such as the one displayed in Fig. 4.10(b), follow a similar principle while allowing arbitrary overlapping geometric shapes [232, 233].

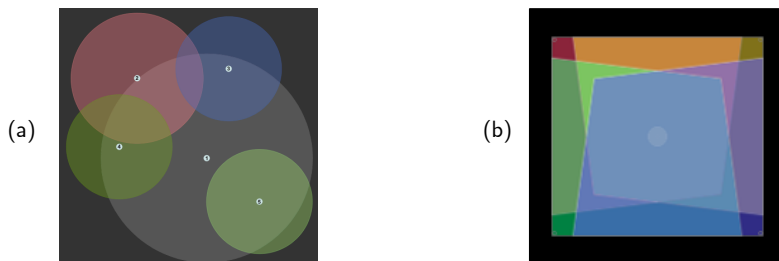


Figure 4.10. Examples of control interfaces for blending multiple vectors (e.g., presets, sounds, latent codes, etc.). Each geometric shape assigns a weight to a vector, depending on the position of a cursor.

In contrast, most sound morphing techniques from Chapter 3 need to establish *correspondences* between the analyzed representations of exactly two inputs sounds. This includes the state-of-the-art SMT, which requires matching

⁷<https://cyclimg74.com/products/max>

partials of the *source* sound with partials of the *target* sound using a dedicated algorithm. However, some morphing methods from Chapter 3, such as those based on differentiable synthesizers, can embed inputs into a latent space where straightforward linear operations can be performed. Similar to SPINVAE, they allow the computation of a weighted sum for an arbitrary number of latent embeddings.

4.4.2 Extrapolation

In addition to interpolation between presets, SPINVAE also enables extrapolation beyond the test presets. Interpolation uses the means $\mu^{(n)}, \mu^{(m)}$ from the preset encoder output (Fig. 4.4), corresponding to some inputs presets $\mathbf{u}^{(n)}, \mathbf{u}^{(m)}$. As displayed in Fig. 4.11, the linear blending formula $\mathbf{z}(\alpha) = (1 - \alpha)\mu^{(n)} + \alpha\mu^{(m)}$ enables extrapolation with $\alpha < 0.0$ or $\alpha > 1.0$, and any given $\mathbf{z}(\alpha)$ can be decoded into an extrapolated preset. An illustrative sequence of spectrograms is provided in Fig. 4.12, and audio examples are available online⁸.

The example in Fig. 4.12 shows that extrapolated presets exacerbate some characteristics of the starting or the ending preset. For instance, towards the left of Fig. 4.12, spectrograms show less and less high frequency contents. Towards the right of Fig. 4.12, however, spectrograms are richer and the low-frequency modulation of the main partials seems to get stronger. However, informal tests seem to show that extrapolation cannot be extended too far in general. Output presets tend to diverge into incoherent sounds; e.g., harmonic presets occasionally turn into unrelated or noisy presets. This is probably inherent to the linear latent extrapolation, which generates samples that get more and more distant from the origin. The likelihood of these extrapolated latent embeddings is very low under the prior distribution $p(\mathbf{z})$ (Fig. 4.11). Hence, these embeddings get decoded into unlikely and unexpected presets. Spherical extrapolation [231] could help mitigate this issue, but it has not been formally tested yet. An in-depth study of extrapolation is left for future works.

⁸<https://gwendal-lv.github.io/spinvae2/#spinvae-2-extrapolations>

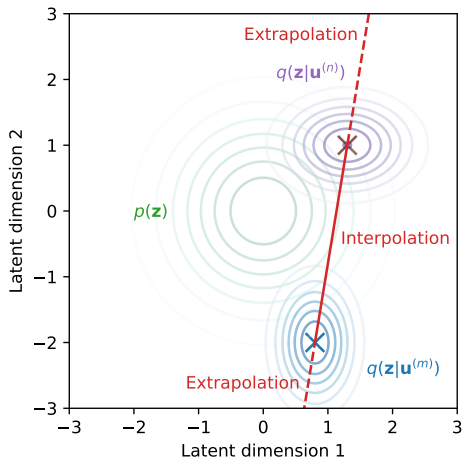


Figure 4.11. Interpolation and extrapolation in an artificial 2D VAE latent space. The prior distribution is $p(z) = \mathcal{N}(0, I)$. The $q(\mathbf{z}|\mathbf{u}^{(n)}) = \mathcal{N}(\mu^{(n)}, \sigma^{(n)})$ and $q(\mathbf{z}|\mathbf{u}^{(m)}) = \mathcal{N}(\mu^{(m)}, \sigma^{(m)})$ distributions result from the encoding of two distinct presets $\mathbf{u}^{(n)}, \mathbf{u}^{(m)}$. The distributions’ means $\mu^{(n)}, \mu^{(m)}$ are represented by crosses.

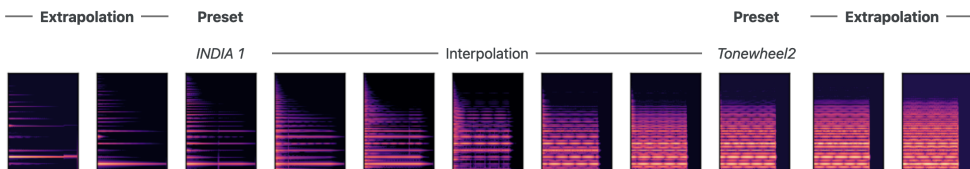


Figure 4.12. Interpolation and extrapolation example using SPINVAE.

4.4.3 Preset modulation

New modulation techniques are also emerging from the SPINVAE model. For example, coherent variations of a preset \mathbf{u} can be achieved by sampling latent codes from the encoded posterior distributions $q(\mathbf{z}|\mathbf{u}) = \mathcal{N}(\mathbf{z}; \mu, \sigma)$ and decoding them into new presets. As illustrated in Fig. 4.13, this allows a syn-

thesizer preset to evolve over time, resulting in a more *dynamic* sound. Audio examples are available online⁹.

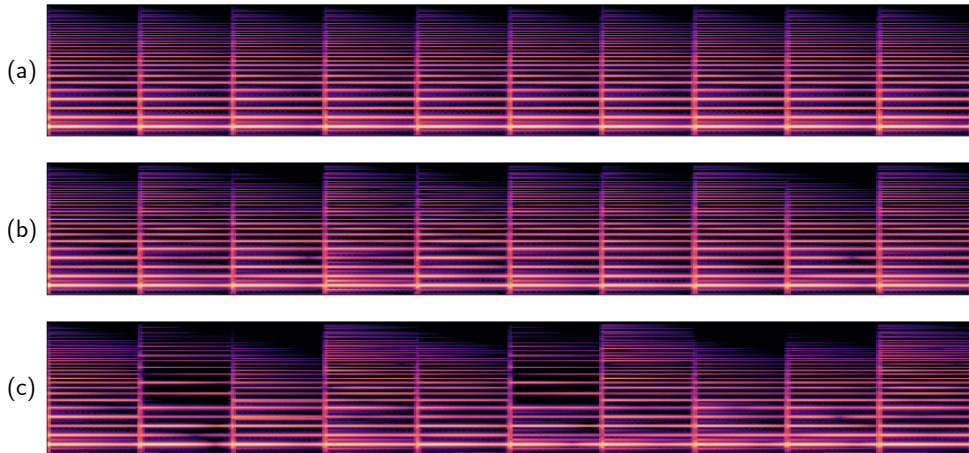


Figure 4.13. Modulations of a preset \mathbf{u} using SPINVAE, obtained by playing the same MIDI note 10 times using a randomly sampled preset $\mathbf{z} \sim q(\mathbf{z}|\mathbf{u}) = \mathcal{N}(\mathbf{z}; \mu, \sigma)$ for each note. (a) The same preset played 10 times, equivalent to sampling with $\sigma = 0_{L_z}$. (b) Sampling with $\sigma = 2\sigma_{\text{encoder}}$. (c) Sampling with $\sigma = 3\sigma_{\text{encoder}}$.

When sampling a VAE, the σ vector of a posterior latent distribution $q(\mathbf{z}|\mathbf{u}) = \mathcal{N}(\mathbf{z}; \mu, \sigma)$ can be artificially increased. Similar to raising the temperature in temperature scaling [220, Chap. 5] with a LLM, this adjustment flattens the probability densities. This allows for greater diversity and creativity, enabling the generation of more varied and novel outputs.

Similar modulations are classically obtained by manually routing some LFOs to amplitudes or cut-off frequencies (Subsection 1.2.2). This prevents the output sound from being perceived as *static*, but requires dedicated effort to tune the modulations for each individual preset. Automatic generation of modulations has been previously achieved using genetic algorithms for synthesizing dynamic drum timbres [234]. Compared to genetic algorithms, the main benefit of using a VAE is the explicit probability distributions. Another advantage

⁹<https://gwendal-lv.github.io/spinvae2/#presets-modulation>

is the direct sampling from these distributions and the feedforward decoding of a preset, whereas an iterative genetic algorithm does not immediately provide a result.

4.4.4 Other conventional synthesizers

This chapter's results are based on [SPINVAE](#) controlling the DX7 [FM](#) architecture, which was primarily chosen because it is notoriously hard to program. Preset databases and an open-source emulation, which are necessary for training and evaluation, are also available. Training and testing the model with other conventional synthesizers would require obtaining presets from the manufacturers or third parties, and to build new pipelines to automatically program presets and render audio. In the context of this thesis, these tasks have been left for future works. However, they should be easier for manufacturers themselves. They already have access to databases of presets, and own the source code of software synths or have developed software emulations of hardware synthesizers (such as the software clone of the OP-1 synthesizer [7]).

4.5 In brief

Summary of Chapter 4

- Traditional sound morphing techniques are limited by their dependence on specific synthesis methods, and the best ones are computationally intensive.
- The **SPINVAE** model was developed to address the challenge of interpolating synthesizer presets in a model-agnostic manner, treating synthesizers as black-box systems. For testing, it was coupled with the Dexed **FM** synthesizer. The model simultaneously handles all 144 **FM** parameters, including the *algorithm* that controls signal routing, and employs distinct output probability distributions for numerical and categorical parameters.
- The **VAE** architecture combined with Transformers enables interpolations between latent embeddings of synthesizer presets. The objective timbre linearity of transitions is similar to that of the **SMT** baseline for waveform morphing.

Perspective for Chapter 4

- The combination of **SPINVAE** and a **FM** synthesizer is not as smooth as state-of-the-art waveform techniques. Further improving the smoothness could be challenging due to the non-differentiable synthesis parameters.
- The quality of interpolations performed by **SPINVAE** has been quantified through objective metrics from the previous chapter. The next chapter will provide a subjective evaluation of the interpolation.
- Architecture choices for building **SPINVAE** have been justified by theoretical foundations. The next chapter will present an ablation study to formally study the effect of elements such as the multi-head attention encoder and decoder, latent dimension L_Z , etc.

Chapter 5

SPINVAE model analysis

Contents

5.1	Subjective evaluation	156
5.1.1	Experimental protocol	156
5.1.2	Task difficulty	158
5.1.3	Comparison results	159
5.1.4	Smoothness and naturalness subjective ratings	159
5.1.5	Objective correlates of subjective ratings	161
5.2	Ablation study	163
5.2.1	Experiments	163
5.2.2	Auto-encoder architecture	165
5.2.3	Latent properties	167
5.2.4	Output probability distributions	169
5.2.5	Data augmentation strategies	171
5.3	Preset decoder analysis	172
5.3.1	Transformer attention maps	173
5.3.2	Output parameter trajectories	175
5.4	In brief	178

This chapter is partially based on the following publication:

- G. Le Vaillant and T. Dutoit, "Latent Space Interpolation of Synthesizer Parameters Using Timbre-Regularized Auto-Encoders," *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 32, pp. 3379-3392, 2024, doi:[10.1109/TASLP.2024.3426987](https://doi.org/10.1109/TASLP.2024.3426987)

This chapter explores various aspects of the SPINVAE model’s performance in preset interpolation, combining both subjective and objective evaluations. It begins with a perceptual evaluation to compare SPINVAE’s outputs against a baseline. Then, a systematic ablation study assesses the impact of specific model components based on objective metrics. Finally, a closer empirical examination of the preset decoder is conducted, highlighting how it manages parameter trajectories and utilizes Transformer attention mechanisms to produce coherent outputs.

5.1 Subjective evaluation

Objective metrics from Subsection 4.3.2, and Table 4.1 in particular, demonstrate that SPINVAE preset interpolation is better than the linear parametric interpolation, referred to as the *reference* interpolation in this section. To complement these previous results, a perceptual evaluation of the SPINVAE model was conducted. This evaluation allows comparison with the linear reference, and an assessment of the relationship between subjective measurements and objective metrics.

Results reported in Table 4.1 also show that SPINVAE preset interpolation performs less smoothly than state-of-the-art waveform morphing techniques such as SMT [183]. However, as discussed in Subsection 4.3.5, these techniques have a narrower scope as they cannot be used with arbitrary black-box synthesizers. Thus, a direct comparison with preset interpolation is not entirely fair and has not been further considered in this section. Regardless, the objectively measured degradation in smoothness is clearly perceptible in informal listening tests, and it has not been examined through a formal subjective experiment.

5.1.1 Experimental protocol

The experiment has been implemented as a web application, available given a webpage address and corresponding password. It lasted approximately 15 minutes, and 25 subjects participated. It consisted of an anonymous interactive online survey, and participants were informed that only their answers

were recorded, and that they could take a break or stop the experiment at any time.

After an introductory survey (discussed later in this section), the main part of the experiment consisted in presenting sequences of sounds. Participants were able to play individual sounds, or to play the whole sequence automatically (Fig. 5.1). They could replay sounds as many times as they needed.

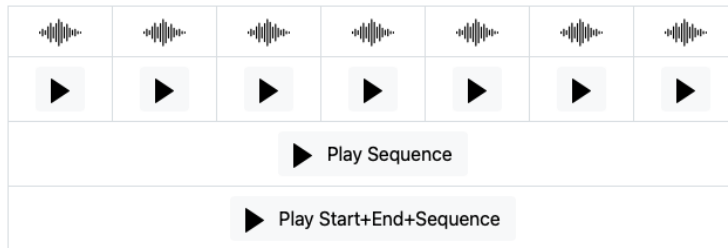


Figure 5.1. Screen capture of the interactive web application used for playing a sequence of sounds. When a sound is played, the corresponding waveform picture is highlighted while others are hidden.

All presented sequences were paired, i.e., if a reference interpolation between presets $\mathbf{u}^{(n)}$ and $\mathbf{u}^{(m)}$ was presented to a participant, then the corresponding SPINVAE interpolation between $\mathbf{u}^{(n)}$ and $\mathbf{u}^{(m)}$ was also presented at some point to this participant. A total of 292 reference sequences and 292 corresponding SPINVAE sequences, each made of $T = 7$ steps, have been evaluated. They were randomly chosen from the test dataset containing 1515 interpolations, for each user at the beginning of the experiment. The main part of the experiment was divided into two phases, which are described in the following paragraphs.

Subjective ratings

During the first phase, subjects were provided a single sequence at a time, and were asked how *smooth*, and how *natural* it was. Ratings could be given on 1 (*not smooth/natural at all*) to 5 (*very smooth/natural*) integer scales, similar to Mean Opinion Score (MOS) scales. The following details were given:

“*Very smooth* means that two consecutive samples always sound quite similar. *Not smooth at all* means that two consecutive samples always sound very different.”

“*Very natural* means that the progression of sounds, from the start to end samples, is performed as you could have expected. *Not natural at all* means that you mostly heard unexpected samples between the start and end samples.”

Comparative ratings

During the second phase, two sequences named A and B were presented simultaneously, using two instances of the playback interface from Fig. 5.1 on the same web page. They corresponded to a pair of reference and SPINVAE interpolations randomly labeled as either A/B or B/A. Participants had to identify the sequence that sounded the smoothest and the most natural to them. Five answers were allowed: A is definitely better; A is marginally better; they are equally good; B is marginally better; B is definitely better.

5.1.2 Task difficulty

A recent sound morphing study [187] considered a similar evaluation task to be very difficult for subjects. Indeed, it requires them to store multiple auditory objects in working memory. Non-verbal information is retained for a few seconds but deteriorates significantly after twelve seconds [235]. Hence, in order to reduce the cognitive load, sequences presented during this experiment are limited to $T = 7$ steps and 2.5s-long sounds. The interface for playing sounds is also minimalist (Fig. 5.1), in order to reduce potential distractions. This interface was iteratively improved based on feedback from alpha testers (subjects not included in the final cohort).

As a result, this experiment has not been considered particularly difficult by the participants. As part of a final survey, subjects were asked how difficult the ratings and comparisons were, on a Likert scale from 1 (*very difficult*) to 5 (*very easy*). The average answer is 3.2 for ratings (close to *neutral*), and 3.7 for comparisons (close to *easy*).

An initial survey also asked participants to evaluate their own skills regarding musical practice, music theory and sound synthesis, based on short text descriptions. The cohort could be split into two groups, with 14 subjects in the *skilled* group and 11 subjects in the other. The evaluation of sequences of sounds is expected to be more accurate for musicians, whose auditory working memory is enhanced compared to non-musicians [236].

5.1.3 Comparison results

The results of subjective comparisons are reported in Fig. 5.2(a) and highlight the superiority of model-based preset interpolation compared to the reference (linear parametric interpolation). This is consistent with the objective metrics from Chapter 4 (lower part of Table 4.1). Moreover, Fig. 5.2(b) shows that the most skilled participants express an even stronger preference for SPINVAE interpolations. Because these subjects possess cognitive advantages for the task [236], results in Fig. 5.2(b) are considered a more accurate assessment of the model’s subjective improvements over the reference method.

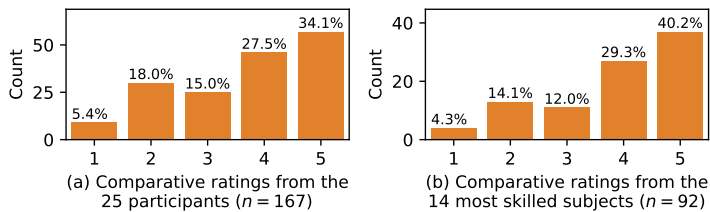


Figure 5.2. Subjective comparisons between paired interpolations. “5” means “SPINVAE is definitely better than the reference” and “1” means “the reference is definitely better than SPINVAE”.

5.1.4 Smoothness and naturalness subjective ratings

Paired subjective ratings are reported in Fig. 5.3. The average *smoothness* is 3.4 for SPINVAE and 3.0 for the reference, while the average *naturalness* is 3.3 and 2.8 for SPINVAE and the reference, respectively. The distributions of SPINVAE *smooth* and *natural* scores are significantly higher than those of

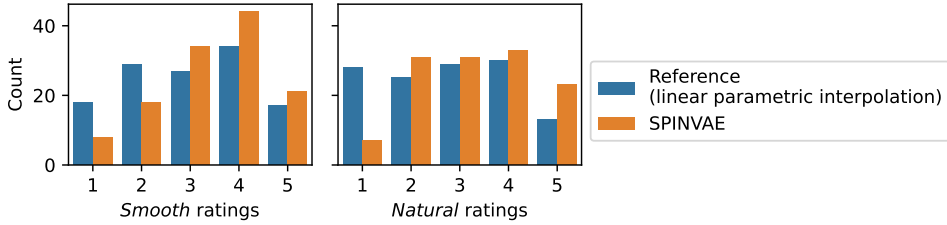


Figure 5.3. Subjective evaluations of smoothness and naturalness. $n = 125$ for each model and each type of rating (500 measurements total).

the reference (p-values from the Wilcoxon signed-rank tests are 1.4×10^{-3} and 1.9×10^{-4} , respectively). This further demonstrates the enhanced quality of model-based interpolations.

Another observation is that the reference method generates a lot of interpolations which are not very smooth or natural, because of the non-linear and non-differentiable relationships between synthesis parameters. Nevertheless, Fig. 5.3 demonstrates that the reference can also perform well. A substantial amount of reference sequences have been rated 4/5 or 5/5 on the smoothness and naturalness scales. If presets $\mathbf{u}^{(n)}$ and $\mathbf{u}^{(m)}$ employ the same FM algorithm and have similar configurations of oscillators, a straightforward linear parametric interpolation can indeed be smooth and natural.

A Principal Component Analysis (PCA) has also been performed on the set of paired *smooth* and *natural* ratings, without considering the technique (reference or SPINVAE) related to these measurements. Integer values were zero-mean, unit-variance normalized before performing the analysis, and the results are presented in Fig. 5.4 (displaying non-normalized integer ratings). The first eigenvector unexpectedly lies almost perfectly on the "smooth = natural" diagonal direction, and it explains 87.9% of the variance. Therefore, it seems that the dimensionality of measurements could be reduced from 2D to 1D. Based on these findings, future experiments could ask subjects to evaluate how "smooth and natural" each sequence is, combining both criteria.

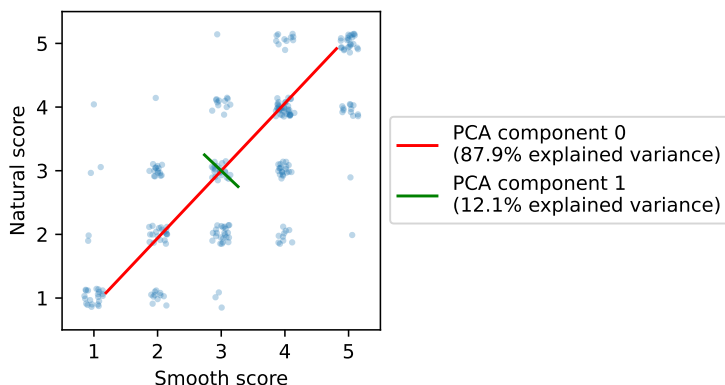


Figure 5.4. PCA on the set of *smooth* and *natural* subjective ratings. An artificial 0.15 jitter has been added to integer values for visualization purposes.

5.1.5 Objective correlates of subjective ratings

The objective linearity and smoothness metrics proposed in Section 3.2 are based on relevant studies about timbre and sound morphing in particular, as well as model-based interpolation in general. For the test sequences randomly drawn for this experiment, the correlations between objective metrics and subjective ratings were also computed. For each pair of objective metric (e.g., linearity of the ACTM depth) and subjective rating (e.g., *naturalness*), 250 data points are available. The Pearson correlations for all possible pairs are presented in Fig. 5.5.

Nearly all proposed metrics show significant correlations with subjective measurements, further supporting the validity of our objective evaluation method from Section 3.2. Out of 74 objective metrics, only two are not significantly correlated to both subjective ratings. The amplitude modulation feature exhibits the lowest correlations (for both the objective linearity and smoothness) and could be excluded from future objective evaluations.

The most correlated features are semantic descriptors of timbre, such as depth, warmth and boominess computed by ACTM [204]. Regarding TT features, spectral variation, spectral roll-off, and inharmonicity provide some of the best objective correlates. This is consistent with previous research on timbre [17,

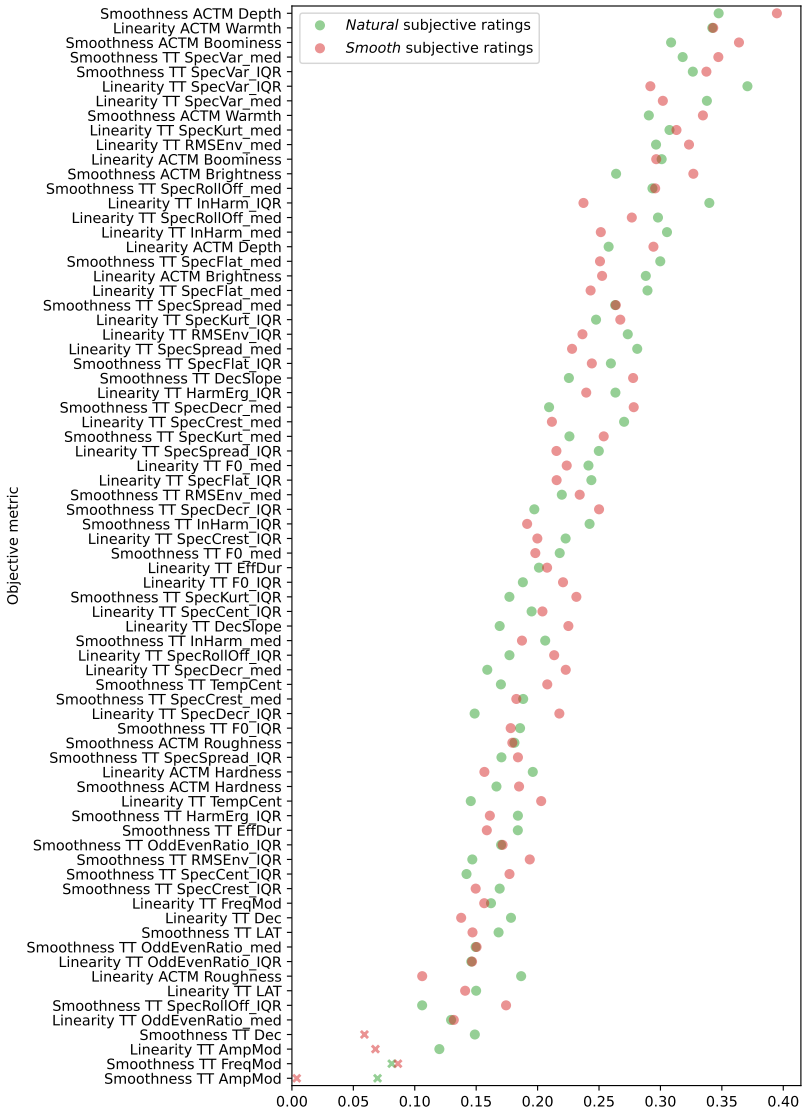


Figure 5.5. Pearson correlations between objective interpolation quality metrics and *natural/smooth* subjective ratings. Significant correlations (p-value < 0.05, n = 250) are represented by dots, otherwise crosses are used.

20]. However, the relatively low correlation of the LAT and spectral centroid metrics is unexpected and contrasts with their typical relevance [17, 20].

The largest correlation values are around 0.35 to 0.40, which is moderate. Nevertheless, it is argued the correlation levels are reduced by noise in the subjective measurements. Among the 292 uniformly sampled sequences, 29 have been presented to at least two subjects. Among these 29 sequences, for a given interpolation from preset $\mathbf{u}^{(n)}$ to $\mathbf{u}^{(m)}$ and method (reference or SPINVAE), the average rating difference is 1.1 (27.5% of the rating scale) for *smooth* and 1.4 (35%) for *natural*. The maximum difference between two participants is 3 for both criteria. This shows that different participants tend to rate the same sequence quite differently.

The computation of audio features is also a source of noise. Occasional numerical instabilities were observed when computing TT and ACTM timbre descriptors. Moreover, some feature estimators may generate inaccurate values [237]. These errors can propagate into the computations of objective metric values, potentially lowering their correlations with subjective measurements.

Overall, neither the smoothness nor the linearity objective metric consistently outperforms the other. For example, the smoothness of the ACTM depth shows stronger correlations with subjective ratings than its linearity, while the reverse is true for the TT RMS envelope. This further supports the appropriateness of using both types of objective metrics, i.e. linearity and smoothness, as discussed in sub-section 3.2.

5.2 Ablation study

5.2.1 Experiments

This section presents systematic evaluations of the impact of specific components introduced in Section 4.2. By removing these elements one by one, their individual contribution to the overall performance of the model can be assessed, providing deeper insights into their roles within the system.

For all variations of the SPINVAE model, such as training without the latent timbre regularization, or replacing Transformer blocks by MLPs, a multi-dimensional hyper-parameter sweep has been performed. Results reported in

each row of Table 5.1 are obtained using the best set of hyper-parameters, selected according to performance on the validation dataset. Values from Table 5.1 are those from final evaluations on the test dataset.

For each model variant, a minimum of 12 sets of β, γ hyper-parameters (Eq. 4.4) have been used for training and validation, as they seem to have a strong influence on interpolation quality. The amount of VAE regularization¹ β could vary in $\{5 \times 10^{-4}; 5 \times 10^{-5}; 5 \times 10^{-6}\}$. The γ hyper-parameter, controlling the amount of timbre AR in the latent space, could vary in $\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$. For models with different types of neural networks, for instance when using MLPs or LSTM networks instead of Transformer blocks, multiple learning rates were also tried. Values could be smaller or greater than the SPINVAE default learning rate (Subsection 4.2.6). The range of hyper-parameters was occasionally extended, depending on observed validation performance. Moreover, other tests in this section rely on additional sweeps of the L_Z hyper-parameter.

As a result, hundreds of models were trained and validated in order to obtain results reported in this section. This was possible thanks to the relatively fast training (approximately 2.5 hours) of those rather lightweight models, although the objective evaluation of the interpolation quality is CPU-intensive, adding an extra 2.5 hours for the validation of each variant.

Similar to Section 5.1, the linear parametric interpolation of presets is used as a baseline for reporting the relative improvements or degradation of the linearity and smoothness metrics. As previously discussed, SMT waveform morphing has a narrower scope compared to linear or SPINVAE interpolations of presets, which can both handle an arbitrary black-box synthesizer. Regardless, using results with SMT as baseline (not reported in Table 5.1) does not change the relative performance of models variants, thus the conclusions of this ablation study would remain the same.

¹These β values are $\ll 1$ because the spectral loss is normalized for numerical stability, i.e., it is divided by the number of output pixels, while the latent loss is not normalized and is approximately proportional to the number of latent dimensions. If the audio reconstruction loss was not normalized, the equivalent β values would be > 1 , as announced in Subsection 2.3.1.

5.2.2 Auto-encoder architecture

Input and output modalities

The first section of Table 5.1 presents the following auto-encoder architectures:

- **SPINVAE**: a preset auto-encoder with an extra spectrogram decoder acting as a differentiable proxy for the synthesizer (Fig. 4.6);
- **Preset-only VAE**: does not encode or decode audio or spectrogram data;
- **Sound matching**: similar to the PresetGenVAE model (Fig. 2.13), it is a spectral **CNN** auto-encoder with a Transformer preset decoder;
- **Bi-modal VAE**: encodes and decodes both a preset and its spectral representation.

The preset-only VAE (with timbre **AR**) performs almost as well as **SPINVAE** (mean difference: linearity -0.8% , smoothness -1.1%). This indicates that the extra spectrogram decoder (Fig. 4.6) is not the most critical element of the

Table 5.1. Interpolation performance of variants of the **SPINVAE** model, using the linear parametric interpolation as baseline. Mean results also report the **std** across the 37 timbre features.

Model	Linearity		Smoothness		MFCCD (↓)
	Mean % (↑)	Med. % (↑)	Mean % (↑)	Med. % (↑)	
SPINVAE	$+22.4 \pm 10.6$	$+22.9$	$+16.1 \pm 11.2$	$+17.9$	4.1
— without timbre AR	$+19.2 \pm 10.1$	$+18.0$	$+12.1 \pm 10.8$	$+13.9$	2.2
Preset-only VAE	$+21.6 \pm 10.4$	$+21.3$	$+15.0 \pm 11.0$	$+17.1$	4.2
— without timbre AR	$+11.1 \pm 11.0$	$+9.2$	$+11.1 \pm 10.3$	$+12.4$	7.4
Sound matching	$+1.1 \pm 15.6$	-0.3	-9.8 ± 20.2	-11.7	13.2
Bi-modal VAE , additive	$+18.9 \pm 9.8$	$+17.7$	$+12.2 \pm 10.1$	$+13.1$	2.4
Bi-modal VAE , contrastive	$+21.7 \pm 10.4$	$+21.1$	$+14.7 \pm 10.9$	$+15.9$	4.1
MLP	$+2.9 \pm 13.1$	$+2.5$	-3.0 ± 18.8	-1.6	13.6
LSTM	-36.0 ± 30.7	-30.6	-36.5 ± 28.5	-32.7	22.3

SPINVAE architecture. The reason for this is probably the timbre AR during training, which already enforces the extraction of some general characteristics of the spectrograms, and their encoding in the latent space.

The sound matching setup discards the preset encoder, which substantially hinders preset reconstruction. As a result, the MFCCD—the perceptual distance between audio rendered using the ground truth preset, and audio obtained from the reconstructed preset—is more than three times that of SPINVAE. This failure to properly reconstruct presets indicates a discrepancy between spectral and preset representations. This is unlikely to produce good preset interpolations, even if the spectral interpolation using the model’s CNN decoder (not evaluated in Table 5.1) was nearly perfect

Two bi-modal VAEs, both trained with timbre AR, were also tested. The *additive* version simply sums the outputs from the preset and audio encoders. This additive bi-modal VAE deteriorates the linearity (−2.5%) and smoothness (−3.9%) of interpolations. This is remarkable because an extra encoder should lead to improved latent representations. However, this basic fusion of modalities, i.e. summing a preset embedding and an audio embedding, does not enforce the learning of fully-shared representations. Some latent coordinates may encode values of preset parameters while other may encode the corresponding spectrogram. Hence, an interpolation of presets is not guaranteed to be linear and smooth in the timbre domain.

The second bi-modal VAE implements contrastive alignment of the preset and audio embeddings, which is a more advanced fusion of modalities. It can be described as two VAEs (a VAE for presets, and another for spectrograms) running in parallel, akin to [238]. Nonetheless, a contrastive loss [239] encourages the two VAEs to learn shared representations. During training, this contrastive loss maximizes the dot product of latent embeddings for paired data (e.g., a preset and the corresponding Mel-spectrogram) while minimizing the dot product of unrelated latent codes in a minibatch. Contrastive learning is a powerful technique for pre-training models, making them adaptable for a range of downstream tasks or zero-shot inference scenarios [240], including the automatic labeling of audio samples. Despite its versatility, this approach does not appear to significantly improve performance in the specialized task of smooth preset decoding. The linearity and smoothness of the bi-modal contrastive VAE are similar to those of SPINVAE (−0.7% and −1.4%, respec-

tively), although the former requires an additional CNN encoder and involves a more complex training procedure.

Preset encoder and decoder neural networks

The second section of Table 5.1 reports the interpolation performance of SPINVAE when the preset encoder and decoder are replaced by either an MLP or a LSTM RNN. The MLPs are made of six layers of 2048 units with batch normalization between hidden layers. Each LSTM network is bidirectional, i.e., it processes tokens in both forward and backward directions. LSTM networks are also made of six layers, with hidden and cell states (tokens) of size 256.

Although they are universal approximators in theory [171], MLPs generally struggle with sequence data because they lack the capability to model dependencies between tokens. Unlike dedicated sequence models Transformers, which are designed to capture patterns across different tokens at known positions, MLPs do not take into account the order of the data. MLPs also cannot build explicit internal attention mechanisms. Consequently, they struggle to model presets in a VAE framework, and both the reconstruction accuracy and interpolation quality are low.

The observed failure of the LSTM variant could likely be attributed to the vanishing or exploding gradient problem [218]. When modeling Dexed presets, sequences comprise 144 tokens and can be considered as quite long. LSTM networks were harder to train and required more effort in hyper-parameter tuning to achieve decent reconstructions of presets. The reconstruction MFCCD remains the largest in Table 5.1 and interpolation performance is one of the worst.

5.2.3 Latent properties

Latent timbre regularization

The first section of Table 5.1 also reports the performance of SPINVAE and the Preset-only VAE trained without the AR loss (latent timbre regularization). For both variants, the linearity and smoothness are reduced, which confirms

the usefulness of the AR loss (Eq. 4.4 and Eq. 4.7). In particular, the Preset-only VAE without AR suffers from a significant drop in performance. This could be explained by the complete absence of audio information in the latent embeddings, because this model is neither provided spectrograms nor timbre attributes.

Another observation is that the Preset-only VAE with timbre AR performs better than SPINVAE without timbre AR. Timbre regularization alone appears to influence the latent representations more than the addition of the spectral CNN decoder. This extra decoder seems to be the least important element of the SPINVAE architecture in Fig. 4.6, although it does provide a small performance improvement.

Number of latent dimensions

Fig. 5.6 reports changes in linearity and smoothness for a few different L_Z values ranging from $L_Z = 128$ to $L_Z = 1024$. A general observation is that the model's performance is quite robust to changes in L_Z . As described in Subsection 4.2.5, the timbre AR loss is applied to 38 latent coordinates for $L_Z = 128$, or 152 coordinates otherwise. This reduced amount of regularization could explain the slightly lower linearity with $L_Z = 128$. With $L_Z \geq 192$, however, the effect of L_Z on interpolation performance is not clear.

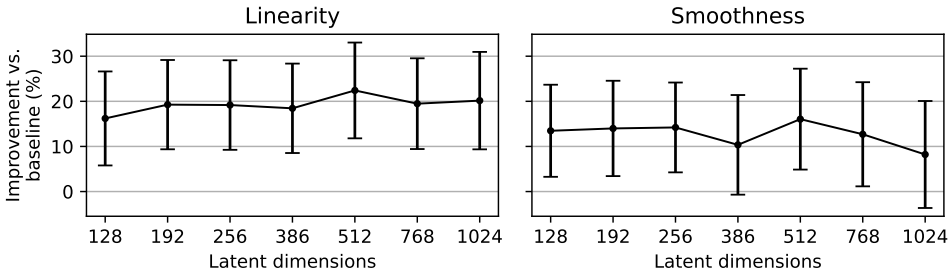


Figure 5.6. Evolution of interpolation linearity and smoothness with the latent size L_Z (mean and std across the 37 timbre features.)

Several factors come into play. On one hand, latent vectors need to be sufficiently large to ensure a good reconstruction of the data [84]. On the other

hand, increasing the latent size can lead to latent collapse, where some latent coordinates become uninformative and are ignored by the decoders. This issue is common when using a powerful non-Gaussian decoder [241], such as $p(\mathbf{u}|\mathbf{z})$ whose parameters are the outputs of Transformer neural network (Fig. 4.6). The collapse can also be selective [241], i.e., it allows for the identification of unnecessary latent dimensions. However, this could collapse some timbre-regularized dimensions, which would be ignored by the decoder. For modeling DX7 presets, a latent size of $L_z = 512$ appears to offer a good balance between regularized and unregularized dimensions, while providing enough latent units to accurately reconstruct presets (as displayed in Table 5.1). This optimal size may vary when controlling a different synthesizer.

5.2.4 Output probability distributions

Table 5.2 reports the effect of using different probability distributions for modeling numerical synthesis parameters, which are all discrete with Dexed. Categorical parameters are always modeled using softmax distributions, thus are not included in these results.

Table 5.2. Interpolation performance when modeling numerical synthesis parameters with different probability distributions. Improvements are relative to the linear parametric interpolation baseline. Mean results also report the `std` across the 37 timbre features.

Model	Linearity		Smoothness		MFCCD (↓)
	Mean % (↑)	Med. % (↑)	Mean % (↑)	Med. % (↑)	
Gaussian (unit variance)	-8.1 ± 15.6	-9.5	+16.2 ± 14.0	+16.3	20.8
Gaussian (decoded var.)	+15.2 ± 8.7	+16.5	+18.0 ± 8.5	+19.8	7.1
DLM 2	+20.8 ± 10.3	+20.0	+15.7 ± 10.8	+17.7	3.5
DLM 3 (SPINVAE)	+22.4 ± 10.6	+22.9	+16.1 ± 11.2	+17.9	4.1
DLM 4	+19.0 ± 10.5	+18.8	+12.2 ± 11.4	+14.5	3.6
Softmax	+19.7 ± 10.8	+18.2	+6.1 ± 14.3	+7.7	4.3

Modeling discrete numerical parameters with normal distributions is quite common. However, using unit-variance Gaussian densities, where only the distributions' means are computed by the preset decoder, leads to the worse results in Table 5.2. Nonetheless, normal distributions whose variance computed by the decoder perform much better. When variance is a decoder output, it tends to get lower in order to minimize the $-\log p(\mathbf{u}|\mathbf{z})$ loss. Therefore, any mistake (a wrongly inferred mean) by the decoder would be heavily penalized, which tends to improve the data reconstruction accuracy. In contrast, Gaussians with fixed variance do not penalize bad reconstructions as much.

DLM distributions, introduced in Subsection 4.2.4, enable both the lowest reconstructed **MFCCD** and the highest mean and median interpolation linearity. Compared to Gaussian distributions, the discrete synthesis parameters are properly modeled as discrete, rather than modeled as continuous. This lowers the output error for these parameters. Moreover, **DLM** distributions are more likely to assign the highest probability to the minimal and maximal numerical values, which corresponds to observations on the training dataset (Appendix D). Using a mixture of three components seems to offer the best linearity and smoothness for preset interpolations. However, the **MFCCD** reconstruction error can be slightly improved by using either two or four components. Although three components seem to be an optimum for modeling numerical **FM** parameters, another count may be more appropriate when using a different synthesizer.

The two Gaussian densities remain, however, the smoothest. If the preset decoder is *smooth*, e.g. if the relationship between input embeddings and output values is Lipschitz continuous, then the Gaussian means cannot change abruptly. Decoder smoothness is indeed a characteristic that can be expected from continuous VAEs (Subsection 2.3.1). Hence, with normal distributions, values of numerical parameters change gradually during an interpolation, which results in an improved objective timbre smoothness.

In contrast, with **DLM** distributions, the most probable value can change abruptly. For instance, two components can have very different means (m in Fig. 4.7) but very similar weights π . The components' weights in the mixture are also outputs of the decoder; then, if the decoder is smooth, the evolution of means and weights will be smooth as well. However, a small change in weights can lead to a sudden modification of the most probable value.

The last distribution reported in Table 5.2 is softmax, which consists of discrete unordered logits corresponding to each possible discrete value. Therefore, the concept of proximity between values is lost, which leads to a steep decline of the interpolation smoothness. Moreover, the linearity and MFCCD values are not significantly better than those of DLM distributions, which overall seem to be the most appropriate.

5.2.5 Data augmentation strategies

Thanks to the preset encoding and decoding method, the SPINVAE architecture can be trained using data from any synthesizer. However, not all synthesizers offer as many presets as the DX7. This subsection introduces two data augmentation techniques to expand the size of the training dataset.

The first technique introduces small variations of synthesis parameters based on basic knowledge of the DX7 synthesizer. For instance, the FM algorithm can be replaced by a similar one, e.g., algorithm 1 can be switched with 2, 3 with 4, 24 with 25, etc. (see Appendix B.1). Small noises can also be added to numerical parameters. This first technique is referred to as “4× data augmentation” in Table 5.3, because four variants of each training preset were generated. It was used to train models previously mentioned in this chapter.

Table 5.3. Influence of data augmentation techniques on interpolation linearity and smoothness, and on MFCCD reconstruction error.

Model	Linearity		Smoothness		MFCCD
	Mean	Med.	Mean	Med.	
	% (↑)	% (↑)	% (↑)	% (↑)	(↓)
No data augmentation	+21.0 ± 10.4	+19.7	+11.5 ± 10.2	+12.5	4.5
4× data aug. (SPINVAE)	+22.4 ± 10.6	+22.9	+16.1 ± 11.2	+17.9	4.1
+5k random presets	+20.9 ± 9.5	+20.6	+14.9 ± 9.6	+16.7	3.2
+10k random presets	+19.9 ± 9.2	+20.0	+15.2 ± 9.3	+17.1	2.8
+24k random presets	+19.1 ± 8.0	+19.6	+15.4 ± 9.1	+17.4	1.8

The second technique relies on marginal distributions $p(u_i)$ of synthesis parameters u_i to generate new random presets. For DX7 synthesis, all parameters are discrete, thus these marginals $p(u_i)$ are probability mass functions extracted from the observed distributions of values in the training dataset (Appendix D). Then, novel presets are generated by independently sampling each $u_i \sim p(u_i)$. A potential other technique would involve sampling some u_i from uniform distributions. However, this approach tends to produce a large number of noise-like or inaudible DX7 presets, and was therefore not retained.

Table 5.3 demonstrates that the proposed “4× data augmentation” achieves the best interpolation performance. In particular, the smoothness is substantially enhanced compared to the basic training without data augmentation. The augmented examples, obtained from handcrafted variations, are similar to but also slightly different from the original presets. They seem to make the model able to encode inputs into a smoother latent space with fewer holes (Subsection 2.3.1)

Regarding the perceptual reconstruction of test presets, all variants present an average MFCCD well below the 10–15 threshold. Nonetheless, the use of extra random presets, appended to the training set, helps reduce the MFCCD. Unfortunately, it also slightly lowers the interpolation linear and smoothness, which seems to indicate that the VAE learns worse representations of the data. Better augmented training samples could be obtained with the help of GANs [80] instead of naively sampling from discrete marginals $p(u_i)$. GANs are trained not to produce unrealistic samples. Therefore, they could create presets with a likelihood more closely matching that of human-made presets from the original dataset.

5.3 Preset decoder analysis

This section provides a further analysis of the Transformer-based preset decoder based on an empirical examination of examples. Unlike the previous sections in this chapter, which focused on systematic studies with objective results, this approach allows for a more detailed and illustrative exploration of how the decoder functions in practice.

5.3.1 Transformer attention maps

Transformer attention maps (e.g., Figures 5.7 and 5.8) are visual representations of the attention mechanism within a Transformer model. They show how the model focuses on different parts of the input sequence when processing each token. Specifically, for a given layer and attention head, the attention mechanism assigns weights to each input token (on the vertical axis of Figures 5.7 and 5.8), indicating the importance of other tokens (on the horizontal axis) relative to it. These maps allow for a better understanding of how the model captures dependencies and relationships within the data.

Multiple types of maps can be displayed, for each layer and each attention head: self-attention in the encoder, self-attention in the decoder, and cross-attention in the decoder. Self-attention maps from the last decoder layers are often the most interesting because they reflect the final decisions made by the model when generating an output. Tokens in these last layers will not be processed much more before being used as parameters of the decoder's output probability distributions.

In particular, layers 5 and 6 of the 6-layer decoder demonstrate remarkable and partially explainable characteristics. Figures 5.7 and 5.8 display self-attention weights averaged over the four attention heads and over all presets from the test dataset. Averaging attention weights from all samples would not make sense in the context of natural language processing, where all sequences have different lengths. However, when processing presets using the method proposed in Chap. 4, tokens always have the same position and functionality.

Layer 5 (Fig. 5.7) displays a recurring, almost periodic attention pattern where tokens of each FM operator (oscillator) attend to tokens describing other operators. In particular, tokens corresponding to Envelope Generator (EG) rate and level controls, describing the amplitude envelope (similar to Fig. 1.6) of a given operator, tend to attend to the same controls from other operators. This enables the decoder to generate coherent presets, ensuring, for example, that two carrier operators do not have the exact same frequency, and that carriers of upper harmonics have a lower amplitude than the F_0 carrier. In particular, Fig. 5.7 shows that the attention between operators 1, 2 and 3 is the strongest. These operators are more often used as carriers than the others (Appendix B.1). Hence, they are responsible for producing the primary

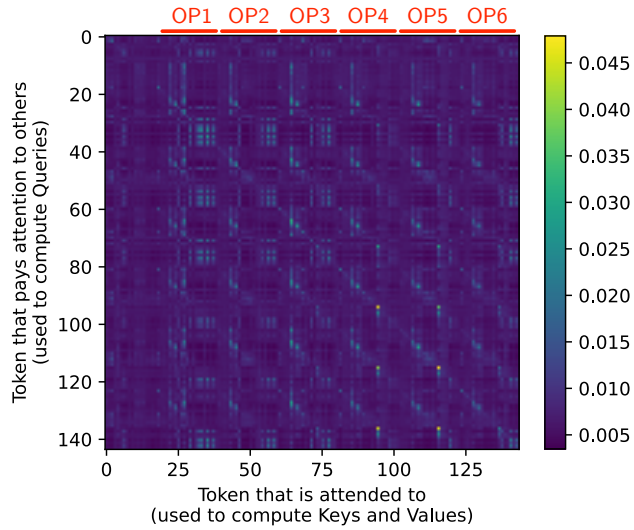


Figure 5.7. Average self-attention map for decoder layer 5/6.

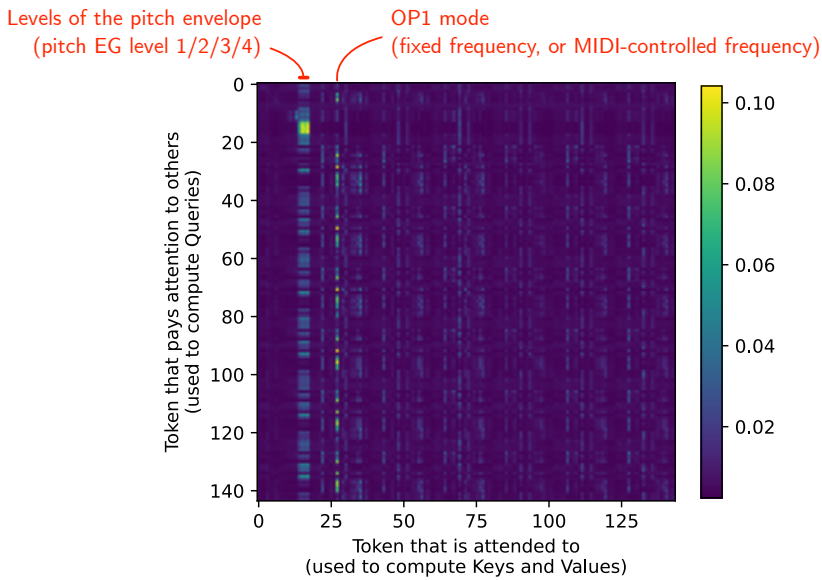


Figure 5.8. Average self-attention map for decoder layer 6/6.

audible frequencies, whose values are highly inter-dependent when building a proper harmonic structure.

The first tokens correspond to the main synthesis controls, including algorithm, main pitch envelope or LFO settings, as shown in the lower part of Fig. 1.12. In decoder layer 5, the main controls (token indices 0 to 17) effectively attend to some tokens related to individual operators. However, the latter do not attend to these main controls (the left-hand side of Fig. 5.7 mostly shows low attention weights).

In decoder layer 6, the opposite can be observed. The inter-operator attention is very weak, while most tokens only attend to some of these main controls, as visible with the vertical stronger attention weights on the left-hand side of Fig. 5.7. In particular, all operators of the FM synthesizer attend the main pitch envelope. This can be explained by the need for all operators, and carrier operators in particular, to synchronize their own frequency with the main pitch of the synthesizer.

5.3.2 Output parameter trajectories

This section presents the evolution of a few parameters during a SPINVAE preset interpolation. Fully understanding and analyzing these parameter trajectories is extremely challenging and not the goal here. It is well known that FM synthesis is particularly difficult to handle, which is precisely why the SPINVAE approach is valuable. The aim is to make a few observations on examples that effectively illustrate general behaviors observed in these interpolations.

For instance, a few synthesis parameters are displayed in Fig. 5.9. These parameters follow a non-linear trajectory, even though SPINVAE interpolation is linear in the latent space. In particular, the *EG level 3* parameters of all operators, corresponding to a sustain level, evolve on quasi-logarithmic scales. The SPINVAE model seems to have learned that such trajectories produce more linear results in the timbre space.

Another observation is that SPINVAE chooses to change the FM algorithm abruptly from 5 to 12, and that this modification happens near the middle of the sequence. Any change in the FM algorithm is inherently unsmooth, as it alters the routing of signals between operators. Interestingly, the change of

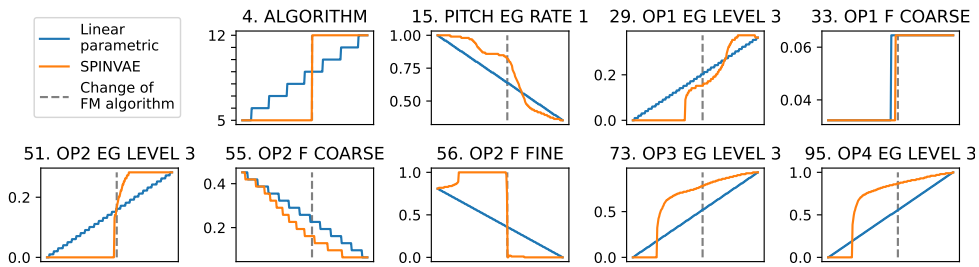


Figure 5.9. Trajectories of a few parameters during an interpolation from the "ELEC.PNO A" preset (electric piano) to the "WindEns2Ed" preset (wind ensemble).

FM algorithm is synchronized with sudden variations of the *OP1 coarse frequency*, *OP2 EG level 3* and *OP2 fine frequency*. **SPINVAE** may be adjusting other synthesis parameters to offset the abrupt change in algorithm, eventually improving the objective timbre linearity and smoothness of the interpolation.

Fig. 5.10 reports the modifications in output sound observed when the **SPINVAE** model changes the **FM** algorithm, for the interpolation presented in Fig. 5.9. The modifications are visible on the Mel-spectrograms and are clearly audible as well, but the two sounds are not entirely different. The **MFCCD** is 21.5, which is approximately double the 10–15 threshold for timbre similarity.

Fig. 5.11 presents similar data but for the linear parametric interpolation instead. When the algorithm is changed, other **FM** parameters continue evolving on affine curves without taking that crucial modification into account. As a result, the variation in timbre is large, with a **MFCCD** of 55.3 between the sounds rendered just before and just after the change. In contrast to the naive preset interpolation, **SPINVAE** (Fig. 5.10) appears to compensate somewhat for the modification of this routing of signals between operators, although the compensation is not perfect.

Appendix G provides two other examples, reporting the trajectories of the entire set of synthesis parameters. One example uses two similar starting and ending **FM** algorithm, whereas the other uses very different signal routings. In both cases, similar observations to those of Fig. 5.9 can be made.

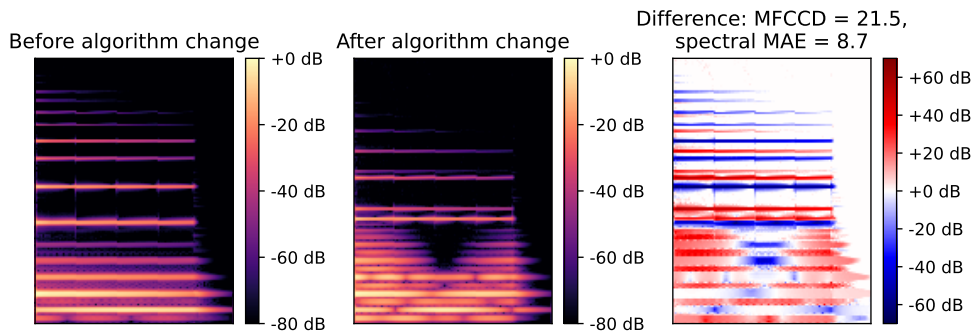


Figure 5.10. Mel-spectrogram modifications occurring at the moment of the FM algorithm change during the preset interpolation computed by SPINVAE in Fig. 5.9. Transition from algorithm 5 to algorithm 12.

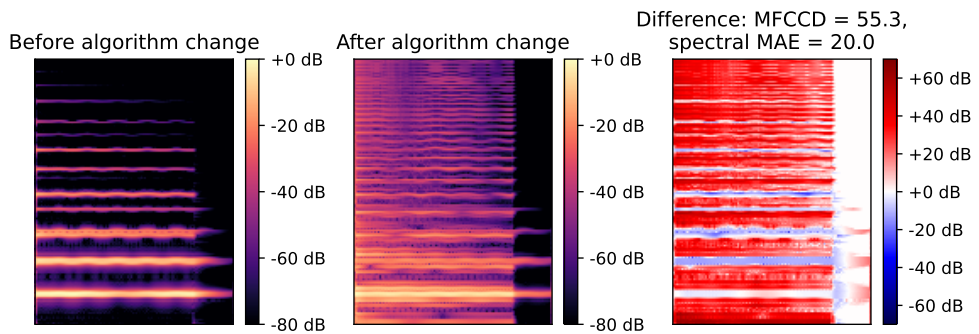


Figure 5.11. Mel-spectrogram modifications occurring at the moment of an FM algorithm change during the linear parametric interpolation in Fig. 5.9. Transition from algorithm 9 to algorithm 10.

5.4 In brief

Summary of Chapter 5

- The subjective evaluation demonstrated that **SPINVAE** performs interpolations that sound smoother and more natural, compared to the reference linear parametric interpolation.
- The ablation study identified key components that contribute to model performance. In particular, timbre regularization applied to latent embeddings, and the use of Transformer encoder and decoder for presets are the most important elements.
- **SPINVAE** interpolations show non-linear trajectories for individual synthesis parameters, with synchronized adjustments that help maintain smooth transitions.

Perspective for Chapter 5

- Data augmentation for datasets of synthesizer presets was a key component in improving latent representations, and it is a topic worth pursuing.
- Experiments and results were based on the combination of **SPINVAE** with an **FM** synthesizer treated as a black box. Future work could focus on interfacing the **SPINVAE** model with other synthesizers, followed by training and evaluation.

Conclusion

This thesis has explored key aspects of sound matching and morphing using conventional synthesizers, focusing primarily on the challenges of preset manipulation and the development of generative models to perform these tasks. Significant contributions, published in international conferences and in a journal, have advanced the state of the art in these fields.

Starting with a review of conventional synthesis techniques and the complexities of handling **FM** synthesis, the work set the foundation for investigating methods for automatic synthesizer programming, also called sound matching. The introduction of the PresetGenVAE model represented a significant improvement in this area, providing a scalable, generative approach to sound matching. While being synthesizer-agnostic, the model was tested on a 144-parameter **FM** synthesizer whose manipulation requires expert skills. PresetGenVAE also handles the challenging **FM algorithm** that controls the routing of signals between oscillators in the semi-modular **FM** architecture. This first research also offered valuable insights into building and training models for effective automatic synthesizer programming, and provided a curated dataset of approximately 30,000 presets and used for other works in this thesis.

Then, a review of the literature on sound morphing techniques revealed a wide range of existing methods, from traditional signal processing approaches to more recent models based on neural synthesizers. However, it also highlighted a lack of standard methods for objectively evaluating the quality of a sequence of morphed waveforms, making it difficult to compare different techniques directly. Inspired by relevant studies on timbre and interpolation, the linearity and smoothness metrics were proposed to provide quantified measurements of the quality of a morphing. For each morphing model and each sequence of the test dataset, the two metrics were evaluated on a set of 37 timbre features extracted using two different tools, providing spectro-temporal as well as semantic descriptors of sound samples. Through these objective metrics, the

best-performing method, based on additive plus noise analysis and resynthesis, was identified. It offered the best trade-off between linearity and smoothness of transitions, and accurate reconstruction of sound characteristics.

Waveform morphing relies on specific synthesis techniques that ensure smooth results. When using an arbitrary conventional synthesizer with no available morphing method, presets could be directly interpolated to try to create intermediate presets. However, the complexity and intricate relationships between synthesis parameters can make the linear parametric interpolation fail in practice. The **SPINVAE** model was introduced to tackle this challenge, treating synthesizers as non-differentiable, black-box systems while utilizing a multi-modal **VAE** architecture. Presets were considered as ordered sequences of synthesis parameters which were encoded and decoded by Transformer neural networks providing inter-parameter attention mechanisms. Specific adjustments were made to embed synthesis parameters into individual tokens, and to use appropriate decoder output probability distributions. An additional training loss enforced the explicit encoding of timbre attributes in the latent space, which was used to perform linear latent interpolations. Presets could be decoded from latent embeddings and then rendered to audio, and the resulting sequences were approximately as linear in timbre space as the best waveform morphing technique. **SPINVAE** was proved to be an appropriate method for generating intermediate presets, and in contrast to state-of-the-art waveform morphing, it is not computationally intensive and generates interpolations in real-time. Subjective evaluations further demonstrated the effectiveness of **SPINVAE**, as subjects rated the model both smoother and more natural than reference linear preset interpolation technique.

Finally, ablation studies identified the critical role of latent timbre regularization as well as Transformer components. In particular, multi-head attention mechanisms allowed the Transformer decoder to capture dependencies between all parameters of presets being generated. This enabled the computation of coherent outputs. The model also appeared to produce synchronized, non-linear adjustments across synthesis parameters, which helped in maintaining smooth transitions, a key advantage over the simpler linear parametric interpolation methods. These findings underline the potential for generative neural networks to reshape how presets are created and manipulated, offering new flexibility for sound designers.

Future research could build on this foundation by optimizing computational aspects, in particular for the sound morphing evaluation procedure. The rendering of sounds from external software synthesizers, as well as the evaluation of timbre features, represented significant computational bottlenecks. Regarding sound morphing, modern differentiable synthesizers demonstrated promising results although they do not outperform the best current techniques. They struggle to reconstruct sounds from a dataset with a wide range of timbre, which hinders their overall performance. Further developments using such synthesizers could establish a new state-of-the-art.

Another aspect is the smoothness of [SPINVAE](#) interpolations, which is not ideal and could benefit from further improvements. The applicability of [SPINVAE](#) to various types of synthesizers could also be explored, provided that a software interface for the synthesizer is implemented, and that sufficient datasets of presets are available. In general, for both tasks of sound matching and preset interpolation, more data could be beneficial for training neural networks. Some solutions for data augmentation were explored in this thesis, but a more extensive study could be conducted. Preset generation using dedicated techniques and models could be considered as well. Additionally, integrating text-based controls for synthesizers would further enhance the scope and impact of both tasks of automatic synthesizer programming and preset interpolation. A multi-modal integration of language models with databases of presets and sounds could significantly enhance the usability and functionality of current synthesizers. The works presented in this thesis provide a robust starting point for these explorations, bridging gaps between traditional signal processing and data-driven approaches to sound design with synthesizers.

Appendix A

Publications related to this thesis

A.1 Journal article

- G. Le Vaillant and T. Dutoit, "Latent Space Interpolation of Synthesizer Parameters Using Timbre-Regularized Auto-Encoders," *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 32, pp. 3379-3392, 2024, doi:[10.1109/TASLP.2024.3426987](https://doi.org/10.1109/TASLP.2024.3426987)

A.2 Papers in international conferences with peer review

A.2.1 Oral presentations

1. G. Le Vaillant and T. Dutoit, "Synthesizer Preset Interpolation Using Transformer Auto-Encoders," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greece, 2023, doi:[10.1109/ICASSP49357.2023.10096397](https://doi.org/10.1109/ICASSP49357.2023.10096397)
2. G. Le Vaillant, T. Dutoit and S. Dekeyser, "Improving Synthesizer Programming From Variational Autoencoders Latent Space," *24th International Conference on Digital Audio Effects (DAFx)*, Vienna, Austria, 2021, pp. 276-283, doi:[10.23919/DAFx51585.2021.9768218](https://doi.org/10.23919/DAFx51585.2021.9768218)
3. G. Le Vaillant, T. Dutoit, and R. Giot, "Analytic vs. holistic approaches for the live search of sound presets using graphical interpolation", in

Proceedings of the International Conference on New Interfaces for Musical Expression (NIME), Birmingham, UK, July 2020, pp. 227–232, doi:[10.5281/zenodo.4813330](https://doi.org/10.5281/zenodo.4813330)

A.2.2 Poster presentations

1. G. Villée and G. Le Vaillant, “Music making mobile application using geometry”, in *Proceedings of the International Computer Music Conference*, 2018, pp. 276–283. [Online]. Available: <https://quod.lib.umich.edu/i/icmc/bbp2372.2018.054/1>
2. G. Le Vaillant, G. Villée, and T. Dutoit, “Portable C++ framework for low-latency musical touch interaction with geometrical shapes”, in *Proceedings of the International Computer Music Conference*, 2017, pp. 372–377. [Online]. Available: <https://quod.lib.umich.edu/i/icmc/bbp2372.2017.063/1>

Appendix B

FM synthesis parameters

B.1 Yamaha DX7

The set of 32 DX7 algorithms is reported in Figure B.1 below. Some provide deep modulations (e.g. 1, 2, 18) while others turn the DX7 into a mostly additive synthesizer (e.g. 31, 32).

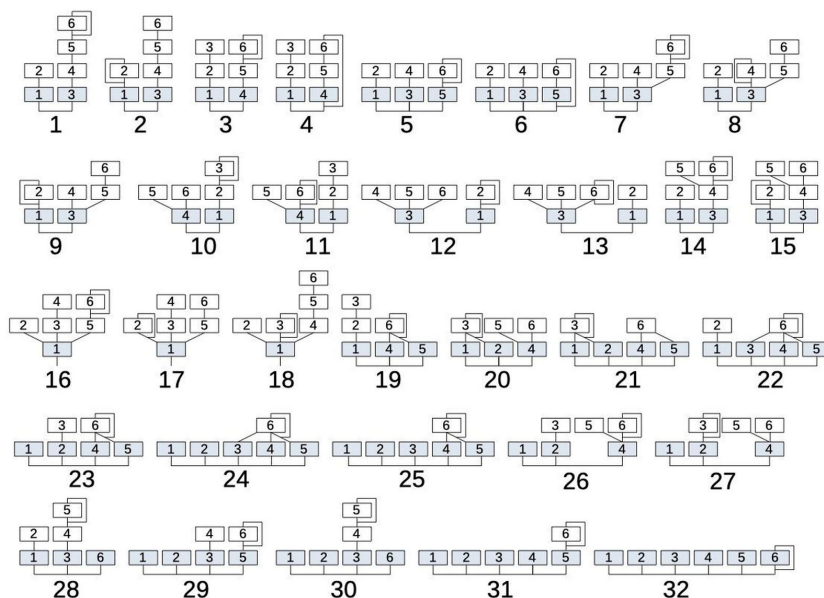


Figure B.1. Algorithms available with the Yamaha DX7 FM synthesizer.

Table B.1 presents the full list of FM synthesis parameters provided by Dexed [41], a VST software clone of the Yamaha DX7 synthesizer. In this Table, parameters' names use acronyms such as Amplitude Modulation (AM), EG, LFO, and abbreviations such as Mod. (Modulation), Op. (Operator) and Osc. (Oscillator).

A parameter's type can be either categorical (e.g., the FM algorithm or LFO waveform) or numerical (e.g. an envelope level or oscillator frequency). Cardinality is the size of the set of possible values of a given parameter. All numerical parameters are discrete.

DX7 synthesis is controlled by a total of 155 parameters, comprising 127 numerical and 28 categorical ones. Out of these 155 parameters, the first 23 are general controls, while the remaining 132 are specifically dedicated to the six individual operators. Each operator has the same set of 22 parameters.

Table B.1. DX7 synthesis parameters

Index	Name	Type	Cardinality
0	Cutoff	Numerical	100
1	Resonance	Numerical	100
2	Output Volume	Numerical	100
3	Master Tune Adjust	Numerical	100
4	Algorithm	Categorical	32
5	Feedback	Numerical	8
6	Osc. Key Synchronize	Categorical	2
7	LFO Speed	Numerical	100
8	LFO Delay	Numerical	100
9	LFO Pitch Mod. Depth	Numerical	100
10	LFO AM Depth	Numerical	100
11	LFO Key Synchronize	Categorical	2
12	LFO Wave	Categorical	6
13	Transpose	Numerical	49
14	Pitch Mod. Sensitivity	Numerical	8
15	Pitch EG Rate 1	Numerical	100
16	Pitch EG Rate 2	Numerical	100
17	Pitch EG Rate 3	Numerical	100
18	Pitch EG Rate 4	Numerical	100

19	Pitch EG Level 1	Numerical	100
20	Pitch EG Level 2	Numerical	100
21	Pitch EG Level 3	Numerical	100
22	Pitch EG Level 4	Numerical	100
23, 45, 67, 89, 111, 133	Op(1-6) EG Rate 1	Numerical	100
24, 46, 68, 90, 112, 134	Op(1-6) EG Rate 2	Numerical	100
25, 47, 69, 91, 113, 135	Op(1-6) EG Rate 3	Numerical	100
26, 48, 70, 92, 114, 136	Op(1-6) EG Rate 4	Numerical	100
27, 49, 71, 93, 115, 137	Op(1-6) EG Level 1	Numerical	100
28, 50, 72, 94, 116, 138	Op(1-6) EG Level 2	Numerical	100
29, 51, 73, 95, 117, 139	Op(1-6) EG Level 3	Numerical	100
30, 52, 74, 96, 118, 140	Op(1-6) EG Level 4	Numerical	100
31, 53, 75, 97, 119, 141	Op(1-6) Output Level	Numerical	100
32, 54, 76, 98, 120, 142	Op(1-6) Mode	Categorical	2
33, 55, 77, 99, 121, 143	Op(1-6) Frequency Coarse	Numerical	32
34, 56, 78, 100, 122, 144	Op(1-6) Frequency Fine	Numerical	100
35, 57, 79, 101, 123, 145	Op(1-6) Osc. Detune	Numerical	15
36, 58, 80, 102, 124, 146	Op(1-6) Break Point	Numerical	100
37, 59, 81, 103, 125, 147	Op(1-6) L Scale Depth	Numerical	100
38, 60, 82, 104, 126, 148	Op(1-6) R Scale Depth	Numerical	100
39, 61, 83, 105, 127, 149	Op(1-6) L Key Scale	Categorical	4
40, 62, 84, 106, 128, 150	Op(1-6) R Key Scale	Categorical	4
41, 63, 85, 107, 129, 151	Op(1-6) Rate Scaling	Numerical	8
42, 64, 86, 108, 130, 152	Op(1-6) AM Sensitivity	Numerical	4
43, 65, 87, 109, 131, 153	Op(1-6) Key Velocity	Numerical	8
44, 66, 88, 110, 132, 154	Op(1-6) Switch	Categorical	2

B.2 Ableton Operator

Ableton Operator is a hybrid FM/additive/subtractive synthesizer. It features four oscillators that can be combined using various algorithms. Each oscillator can generate an arbitrary waveform by adjusting the amplitude of its first 64 harmonics. This characteristic qualifies Ableton Operator as an additive synthesizer. Oscillators can also use ideal waveforms, such as a pure sawtooth containing an infinity of overtones, and filters. Thus, this synthesizer also

meets the criteria for subtractive synthesis. Nonetheless, the eleven available algorithms enable true **FM** synthesis, yielding richer timbres.

Each of the four oscillators has 22 parameters. There are also 49 controls related to the algorithm, pitch envelope, and modulations. Therefore, excluding parameters specific to additive and subtractive synthesis, the total count for **FM** synthesis parameters stands at 137. This number remains high, especially when considering that the Ableton company is known to build user-friendly software and hardware.

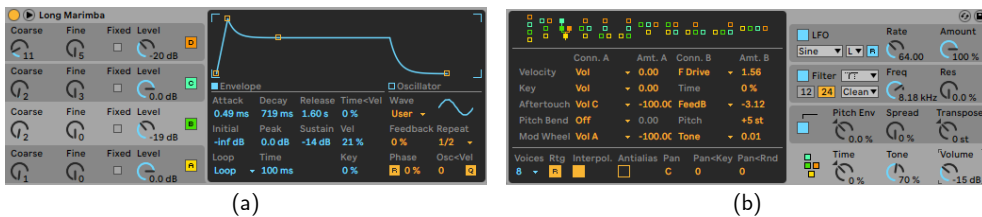


Figure B.2. UI of the Ableton Operator synthesizer. (a) Oscillator view, showing amplitude envelope and waveform selection parameters. (b) Algorithm and modulations view.

Appendix C

Preset creation using a conversational agent

C.1 Creation of a clarinet preset

This Appendix presents conversations obtained using ChatGPT¹ (based on the Generative Pre-trained Transformer (GPT)-4 model [59]) in February 2024. The first prompt was "Can you generate synthesizer presets?", to which ChatGPT answered positively using a very general description of synthesizers' presets. This answer does not hold much value and is not reported in this Appendix. Then, two more prompts were entered, requesting the GPT to generate presets for a Minimoog and a DX7 FM synthesizer. Answers are reported and briefly analyzed in the two following sections.

¹<https://chat.openai.com>

C.2 Minimoog preset

ChatGPT 4

User

I'd like a clarinet preset for a minimoog synthesizer

ChatGPT

Creating a clarinet preset on a Minimoog (or any similar monophonic synthesizer) involves mimicking the harmonic content and response of a clarinet using analog synthesis techniques. The clarinet is a woodwind instrument with a distinct sound, characterized by its rich, warm tones in the lower registers and brighter, more piercing tones in the higher registers. Its sound is primarily even harmonics due to its cylindrical bore.

Since the Minimoog is a subtractive synthesizer, we'll focus on approximating this timbre through oscillator settings, filter manipulation, and envelope shaping. Here's a basic guideline to create a clarinet-like sound on a Minimoog or a similar synthesizer:

- Oscillator Settings
 - Oscillator 1 (OSC 1): Set to a square wave. This will be the primary waveform to mimic the clarinet's harmonic content.
 - Oscillator 2 (OSC 2): Tune it an octave or a fifth above OSC 1, depending on the desired harmonic complexity, and set it to a square wave as well. This adds richness to the sound.
 - Oscillator 3 (OSC 3): You might use this oscillator for additional texture or keep it off for a simpler sound. If used, set it to a square or triangle wave, slightly detuned for a fuller sound.
- Filter Settings
 - Cutoff Frequency: Adjust the cutoff frequency of the filter to control the brightness of the sound. For a warmer, lower register clarinet sound, lower the cutoff frequency. For higher notes, you might slightly increase it.
 - Emphasis (Resonance): Increase slightly to emphasize the harmonics near the cutoff frequency, but not too much, as it can make the sound too sharp or synthetic.
- Envelope Settings
 - Attack (A): Set to a relatively fast but not instant setting to mimic the slight delay in sound production of a clarinet.
 - Decay (D): Adjust for a quick descent to the sustain level, simulating the natural decay of a wind instrument.

- Sustain (S): Set to a medium or high level, as a clarinet note can be sustained by the player’s breath.
- Release (R): A moderate release time will mimic the sound tailing off as the player stops blowing.
- Additional Tips
 - Glide (Portamento): A slight amount of glide can add to the legato feel of a woodwind performance.
 - Modulation: Use the LFO moderately to introduce a slight vibrato effect, similar to a clarinetist’s breath vibrato, by modulating the pitch or the filter cutoff frequency.
- Example Settings
 - OSC 1: Square Wave, 8’ (footage)
 - OSC 2: Square Wave, 8’ or 4’, slightly detuned
 - Filter Cutoff: Adjust to taste, depending on the register
 - Filter Emphasis: Slightly increased
 - Attack: 10-20ms
 - Decay: Short
 - Sustain: High
 - Release: 200-300ms
 - Glide: Minimal
 - LFO for Vibrato: Rate and amount to taste, very subtle

Remember, these settings are a starting point. The exact character of a clarinet can be challenging to replicate perfectly on a Minimoog due to its unique acoustic properties, but with careful tweaking, you can get close to that warm, expressive sound. Experiment with the oscillator mix, envelope settings, and modulation to find the best approximation.

These instructions are clearly written, well-structured, and use the appropriate terms to describe synthesis parameters of the Minimoog. Therefore, they have been tested on a physical Minimoog Voyager RME. Unfortunately, multiple errors, approximations and unclear instructions make the guidelines quite difficult to apply in a practical situation.

For instance, the answer indicates that the clarinet’s sound primarily consists of even harmonics, which is incorrect [54, 55]. Strong even harmonics only appear when playing the highest notes [56]. Nonetheless, the second part of the answer tells to select a square wave oscillator (with only odd harmonics), which should actually help obtaining a clarinet-like sound. Another inconsistency

appears in the advice regarding the glide parameter: initially, it recommends using "a slight amount of glide," only to later assert that glide should be minimal.

Many instructions are also very vague, e.g. suggesting filter or **ADSR** parameters be set to "short" or "high", or the numbers given are wrong. The suggested attack time is definitely too short, and while the decay time is not specified precisely, it should be longer than the attack time. The second square oscillator does not help improving the timbre.

Ultimately, the instructions did not allow to obtain a convincing preset. Nevertheless, achieving a clarinet-like sound on a Minimoog is indeed possible².

C.3 DX7 FM preset

ChatGPT 4

User

How to build a clarinet preset using an FM synthesizer such as the DX7?

ChatGPT

Creating a clarinet preset on an FM synthesizer like the Yamaha DX7 involves a different approach than on an analog synthesizer like the Minimoog. FM synthesis uses the modulation of one waveform by another to create complex harmonic content, which is well-suited for mimicking the rich overtone structure of acoustic instruments like the clarinet.

The clarinet produces sound with a strong fundamental tone and a series of overtones that are primarily even harmonics, due to its cylindrical bore. In the case of FM synthesis, we can use operators (the FM equivalent of oscillators) to modulate each other and create a timbre similar to a clarinet.

Here's a basic approach to designing a clarinet sound on the DX7 or similar FM synthesizers:

1. Choose Your Algorithm

Select an algorithm that allows for both carrier and modulator operators to be utilized effectively. For a clarinet sound, an algorithm with at least one carrier and multiple modulators in a series or parallel configuration

²https://youtu.be/S_FKXtwqJoQ?si=jANetlh5W7RWSBPL&t=330

would be ideal. This setup allows you to create the fundamental tone while also sculpting the harmonic content.

2. Set the Operators

- **Carrier Operator (Fundamental Tone):** This operator produces the audible sound. Set it to a sine wave, which will be modulated by other operators to create the clarinet's timbre.
- **Modulator Operators (Harmonic Content):** Use these to modulate the carrier operator. For a clarinet, you'll primarily want to enhance even harmonics. Start with sine waves and adjust the modulation index (the depth of the modulation) to create the desired harmonic complexity.

3. Fine-tune the Envelopes

Each operator in an FM synth has its own envelope. For a clarinet sound:

- **Carrier Envelope:** Mimic the way a clarinet's sound evolves. Use a moderate attack, a quick decay to a sustain level that matches the note's holding phase, and then a release that simulates the note ending.
- **Modulator Envelopes:** These shape the harmonic content over time. Start with settings similar to the carrier but experiment with shorter decay or sustain levels to vary the harmonic texture as the note sustains.

4. Adjust the Modulation Index

The modulation index controls the intensity of the modulation and, therefore, the harmonic content. Adjusting this for each modulator will change the timbre, allowing you to dial in the characteristic clarinet sound—rich and warm in the lower register, brighter and more piercing in the upper register.

5. Implement Vibrato

A key aspect of woodwind sounds, including the clarinet, is vibrato. Use an LFO to modulate the pitch of the carrier slightly to add this effect. Set the rate to a realistic vibrato speed, and keep the depth subtle.

6. Experiment with Feedback

Some FM synths, like the DX7, allow for operator feedback, which can introduce additional harmonics. A small amount of feedback on a modulator can add richness or edge to the sound, helping to simulate the reediness of a clarinet.

Example Settings for a DX7 Clarinet Preset

- Algorithm: Choose one with a clear carrier and multiple modulators in a configuration that suits harmonic development.
- Carrier Operator: Sine wave, with an envelope that has a moderate attack, full sustain, and medium release.
- Modulator Operators: Sine waves, modulating the carrier to emphasize even harmonics, with varied modulation indices and envelopes that decay faster than the carrier.
- Vibrato: Use an LFO to modulate the carrier's pitch at a slow to moderate rate, with a low depth.
- Feedback: Experiment with minimal feedback on one modulator to enhance the timbre.

Remember, creating realistic instrument sounds on an FM synthesizer requires a lot of experimentation and fine-tuning. The exact settings will depend on your specific synthesizer model and the sound you're aiming for. Start with these guidelines and adjust parameters as needed to capture the essence of the clarinet.

Similar to the Minimoog-related answer, this one about the DX7 is very well-structured and uses appropriate vocabulary and definitions.

However, it remains very generic and lacks specific guidelines on how to program a clarinet-like preset into the DX7. For instance, the description of the algorithm as having "at least one carrier and multiple modulators in a series or parallel configuration" applies to 30 out of 32 DX7 algorithms, making it too broad. Neither the amounts of modulation nor any other parameter are quantified, leaving the user to interpret what 'low', 'varied', or 'faster' means. Moreover, the LLM repeats the same mistake regarding the clarinet's spectrum by incorrectly asserting that even harmonics should be emphasized. Nonetheless, it's worth noting that the final piece of advice, which is "[...] an FM synthesizer requires a lot of experimentation and fine-tuning", may indeed be the most practical and useful guideline..

Appendix D

Histograms of Dexed parameter values

This appendix reports the distributions of synthesis parameters' values for the dataset introduced in Section 2.2. The histograms for all individual parameters are presented in Figures D.1, D.2, and D.3. In these figures, counts on the vertical axis are not informative and have been removed to improve readability. Ticks on the horizontal axis indicate the location of possible values for each parameter (e.g. 32 possible values for *algorithm*, 100 for *LFO speed*, etc.). The cardinality of each parameter is reported in Appendix B. Grey histograms indicate parameters which are not considered during training. The blue color designates main synthesis parameters, and other colors represent parameters of independent operators.

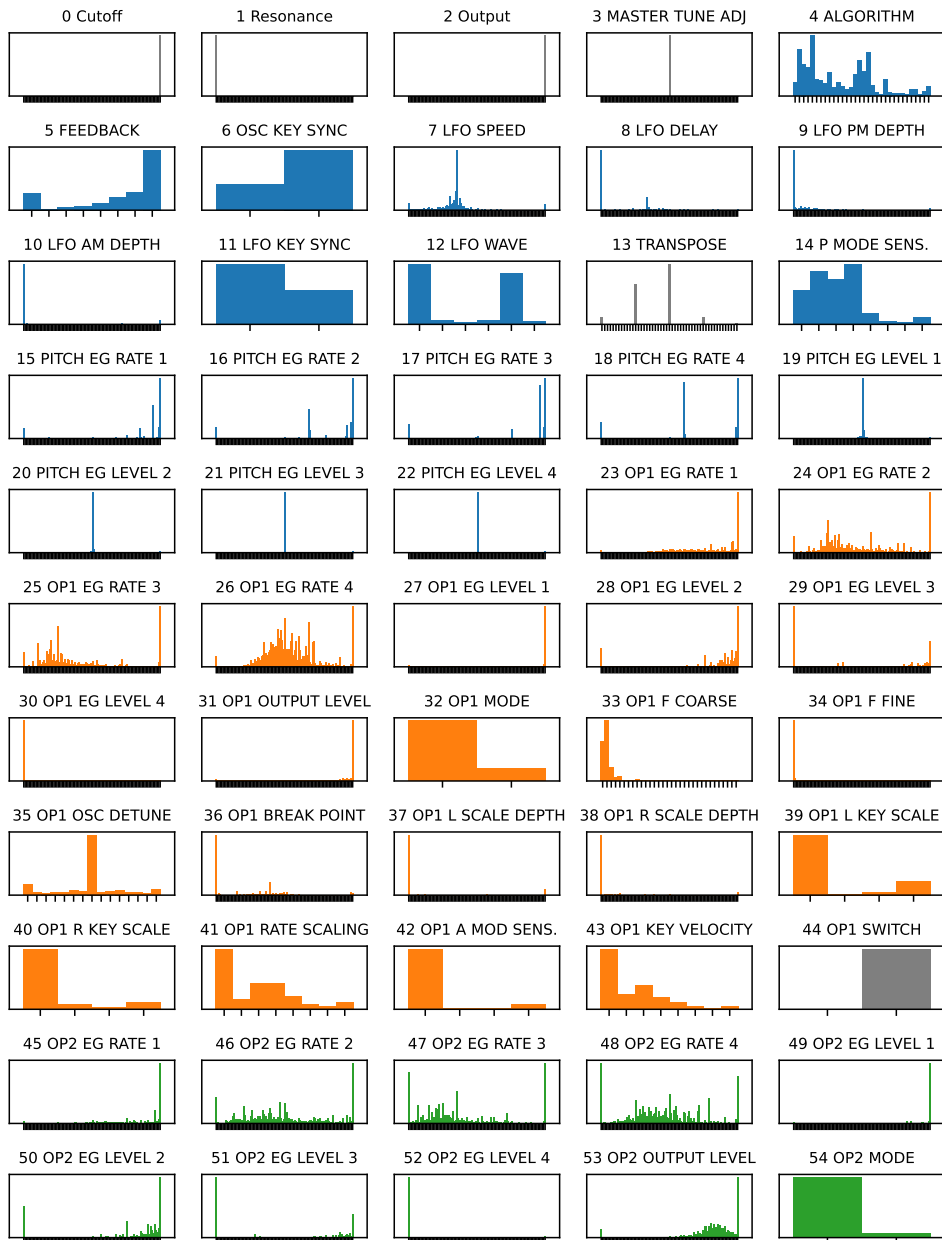


Figure D.1. Histograms of Dexed synthesis parameters (Part 1/3)

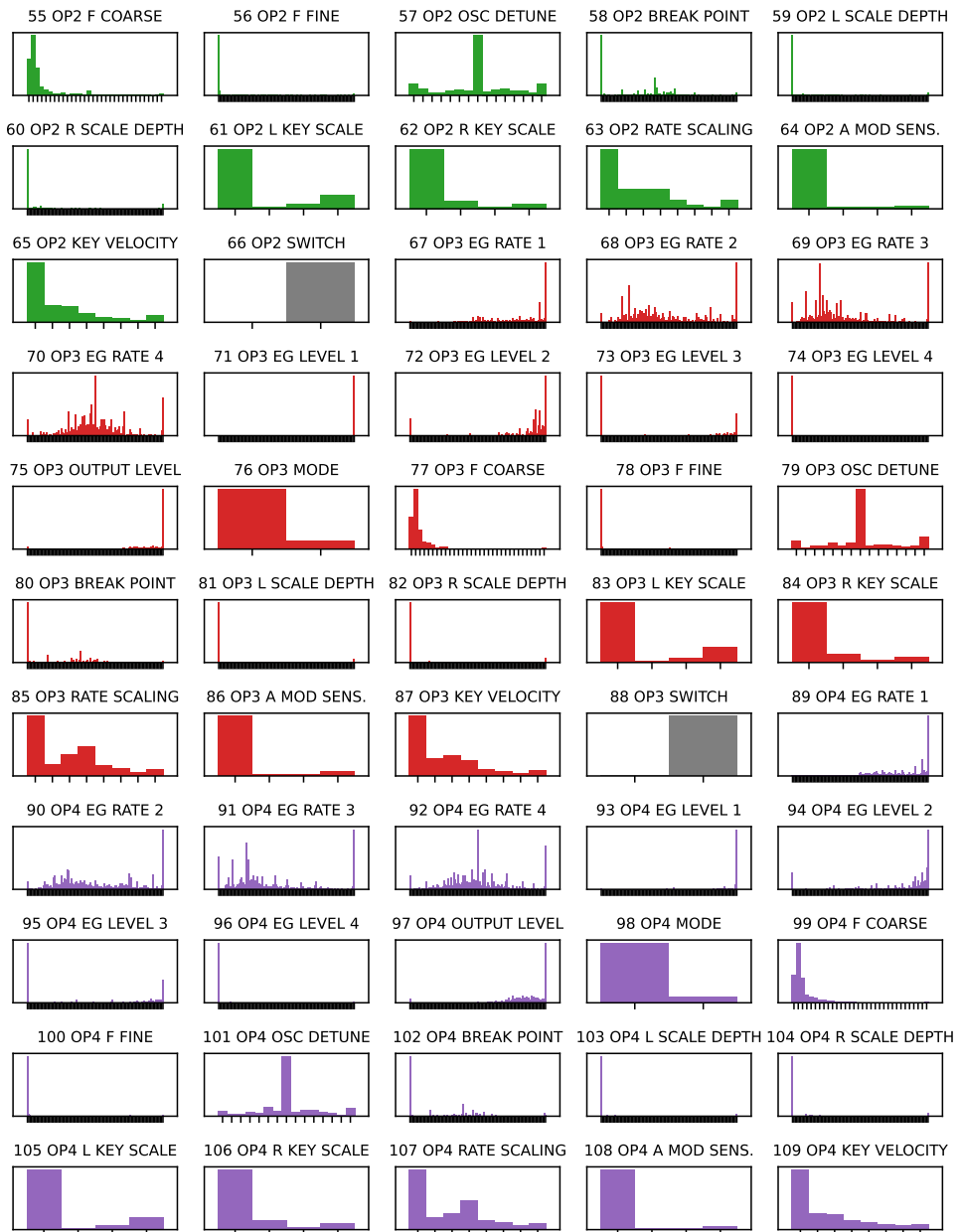


Figure D.2. Histograms of Dexed synthesis parameters (Part 2/3)

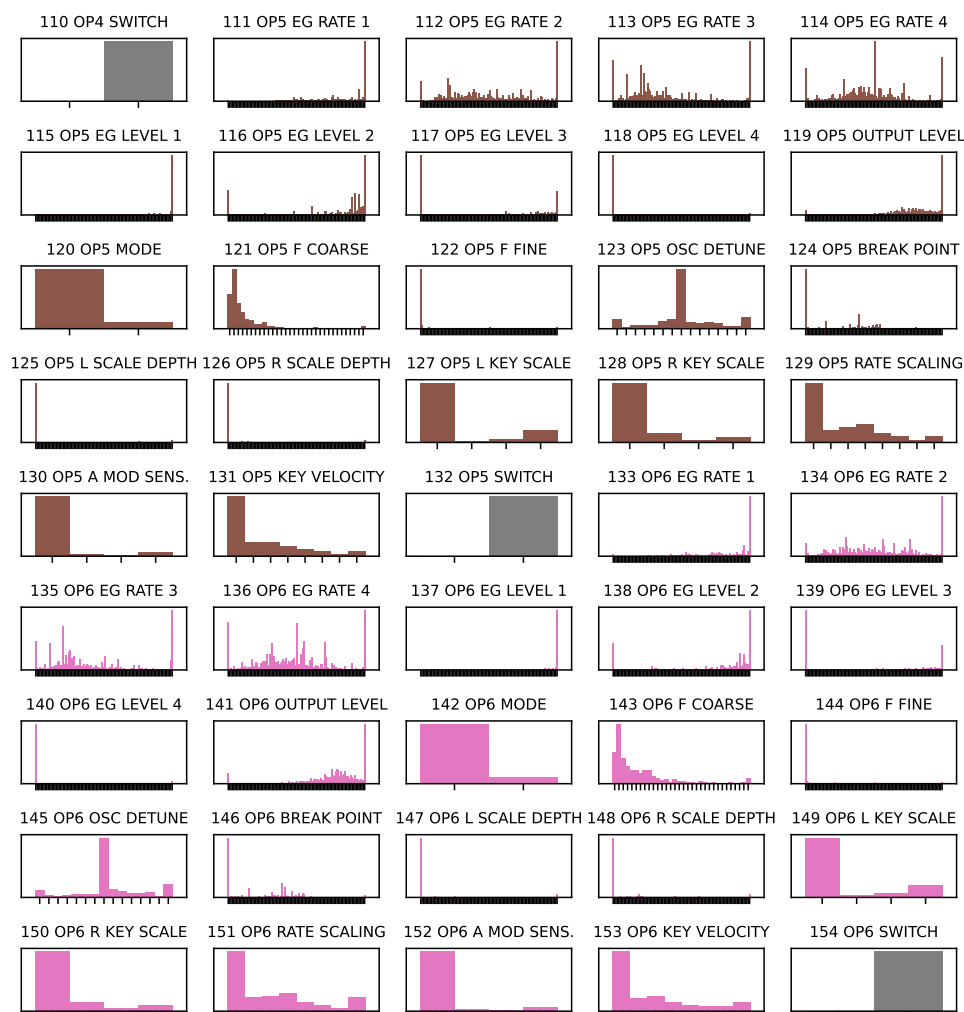


Figure D.3. Histograms of Dexed synthesis parameters (Part 3/3)

Appendix E

Correlations between timbre descriptors

This appendix presents the correlations between all available [TT](#) [203] and [ACTM](#) [204] descriptors (before the feature selection process). [Fig. E.1](#) plots the Spearman rank correlation coefficients obtained for descriptors values from the training dataset (Section 2.2). [Fig. E.2](#) highlights the highly correlated features.

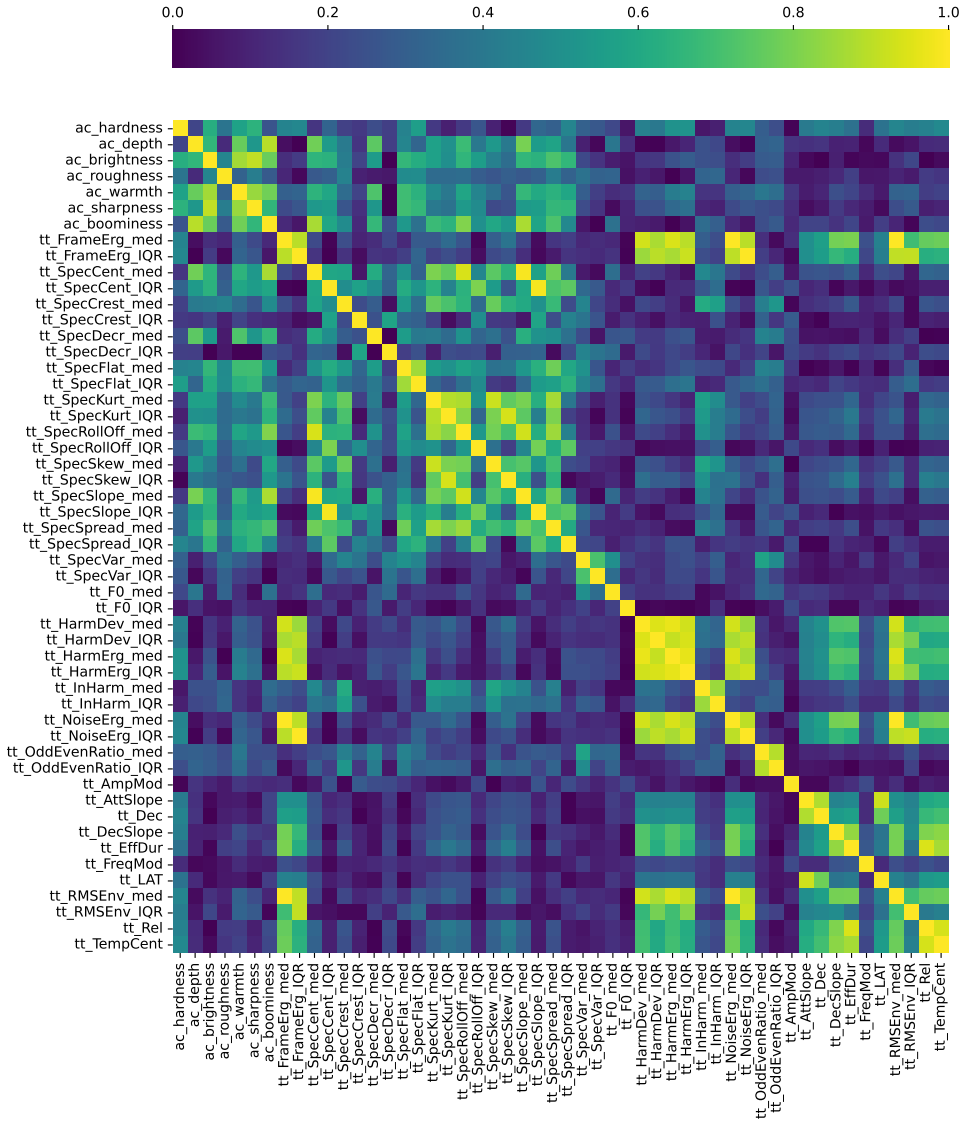


Figure E.1. Spearman rank correlations between all available TT and ACTM timbre descriptors.

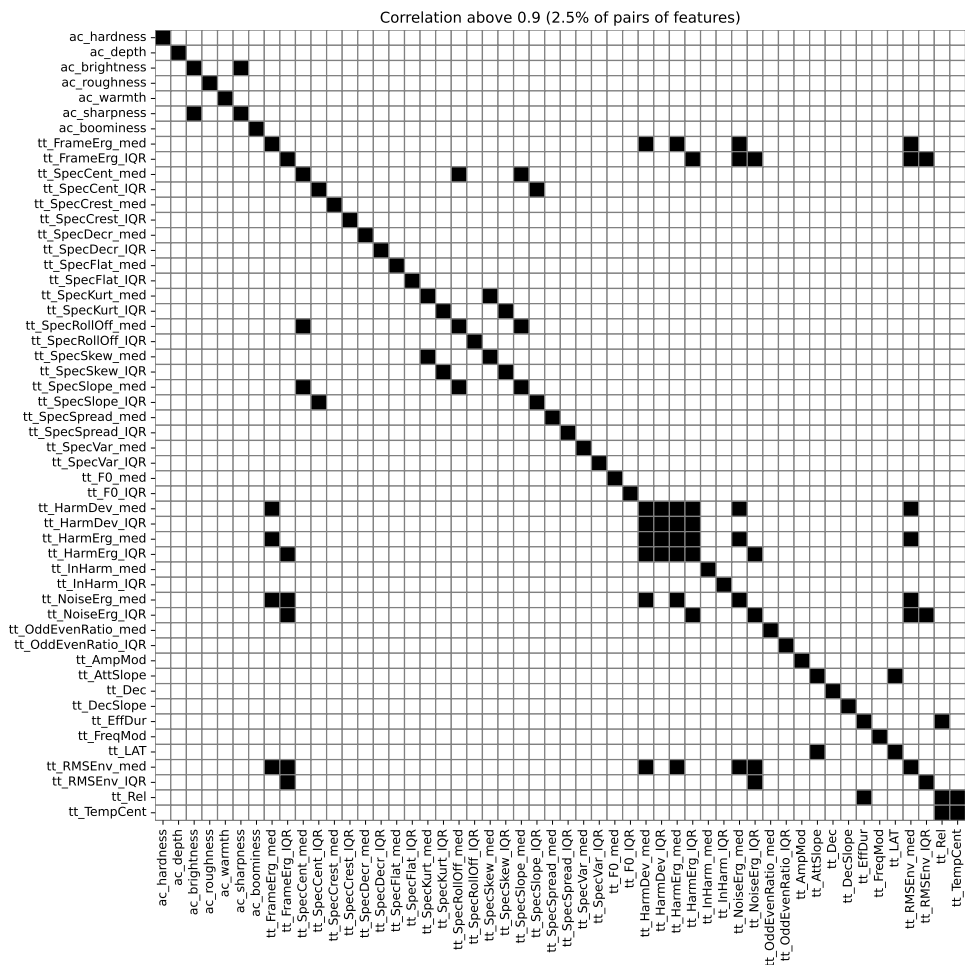


Figure E.2. Highly correlated descriptors of timbre.

Appendix F

Detailed morphing linearity and smoothness results

This appendix presents detailed boxplots of the linearity and smoothness for multiple models of sound morphing or preset interpolation, covering each timbre feature considered for evaluation (Sec. 3.2). Each boxplot represents 1515 data points corresponding to the 1515 sequences of morphed samples.

For each timbre feature, metric values (smoothness or linearity) are scaled using the SMT baseline mean value. A zero value is optimal, while increasingly negative values indicate a degraded linearity or smoothness. The display of negative values has been truncated for readability.

The following results are presented in this appendix, for two sets of models:

- Models from Chap. 3: SMT, spectral NMF, STRAIGHT, CQT-VAE, RAVE, DDSP HpN, DDSP wavetable
 - Linearity in Fig. F.1 and Fig. F.2
 - Smoothness in Fig. F.3 and Fig. F.4
- Models from Chap. 4: SMT, linear parametric interpolation, SPINVAE
 - Linearity in Fig. F.5
 - Smoothness in Fig. F.6

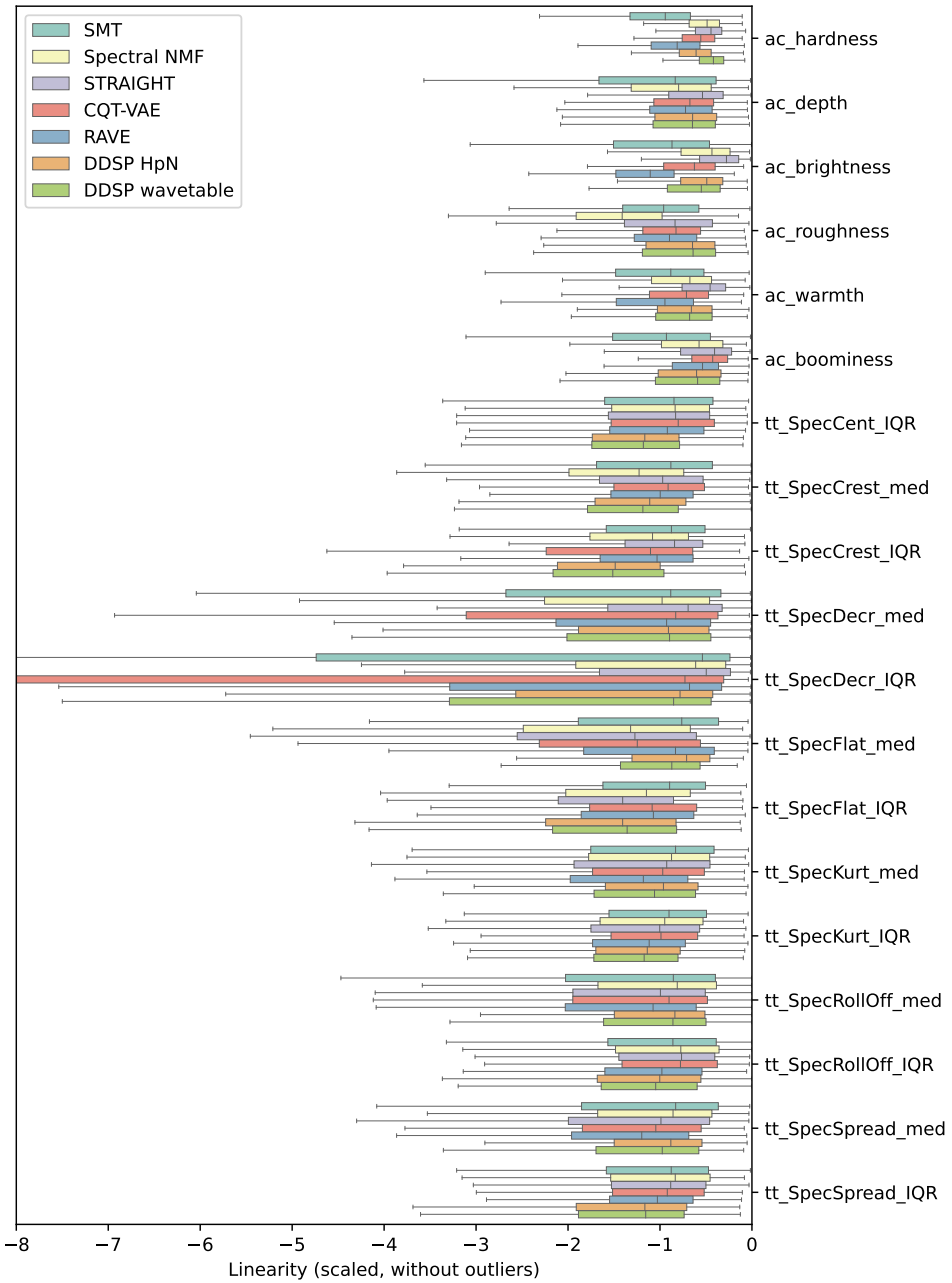


Figure F.1. Linearity of seven morphing techniques (part 1/2).

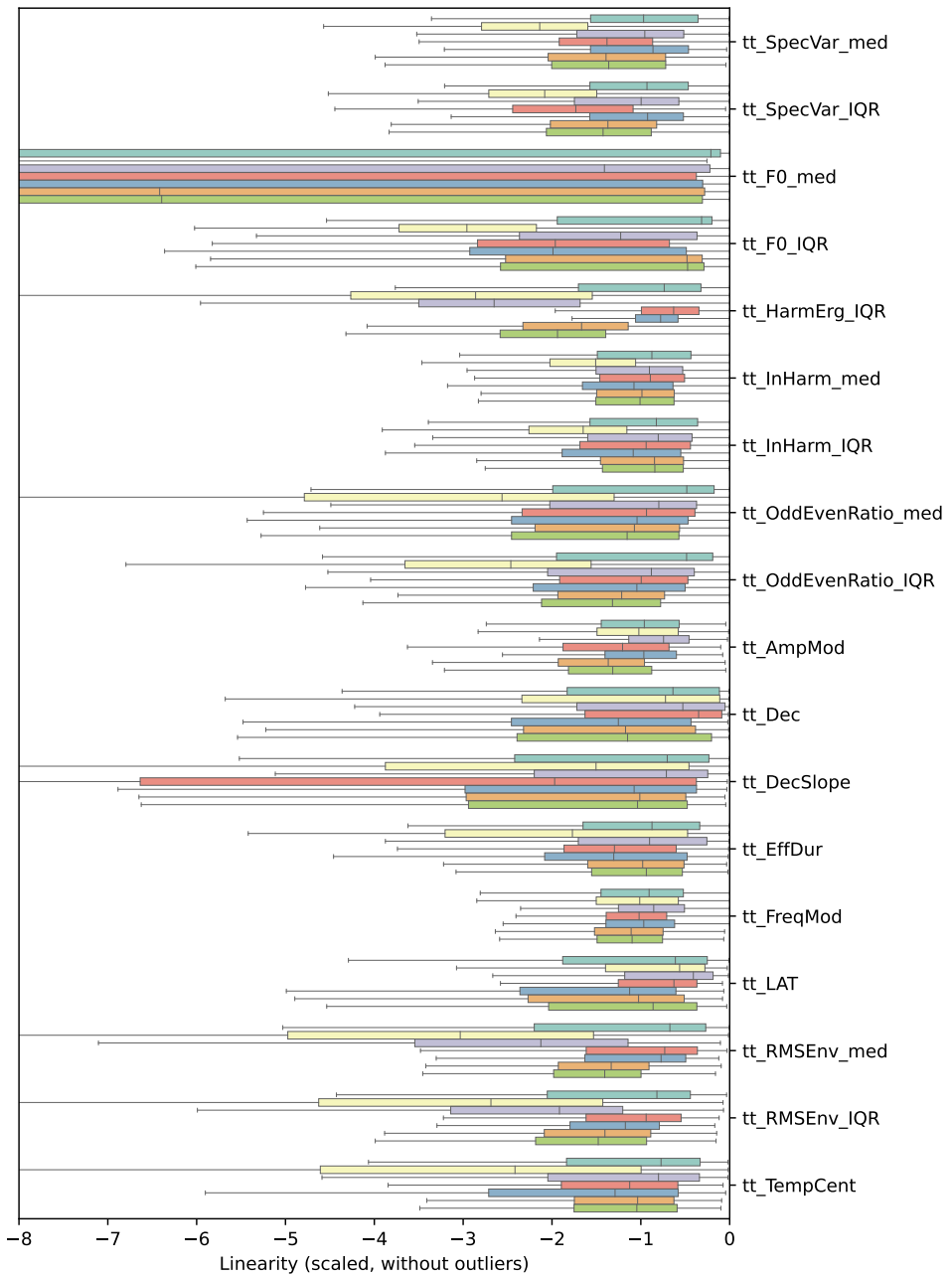


Figure F.2. Linearity of seven morphing techniques (part 2/2).

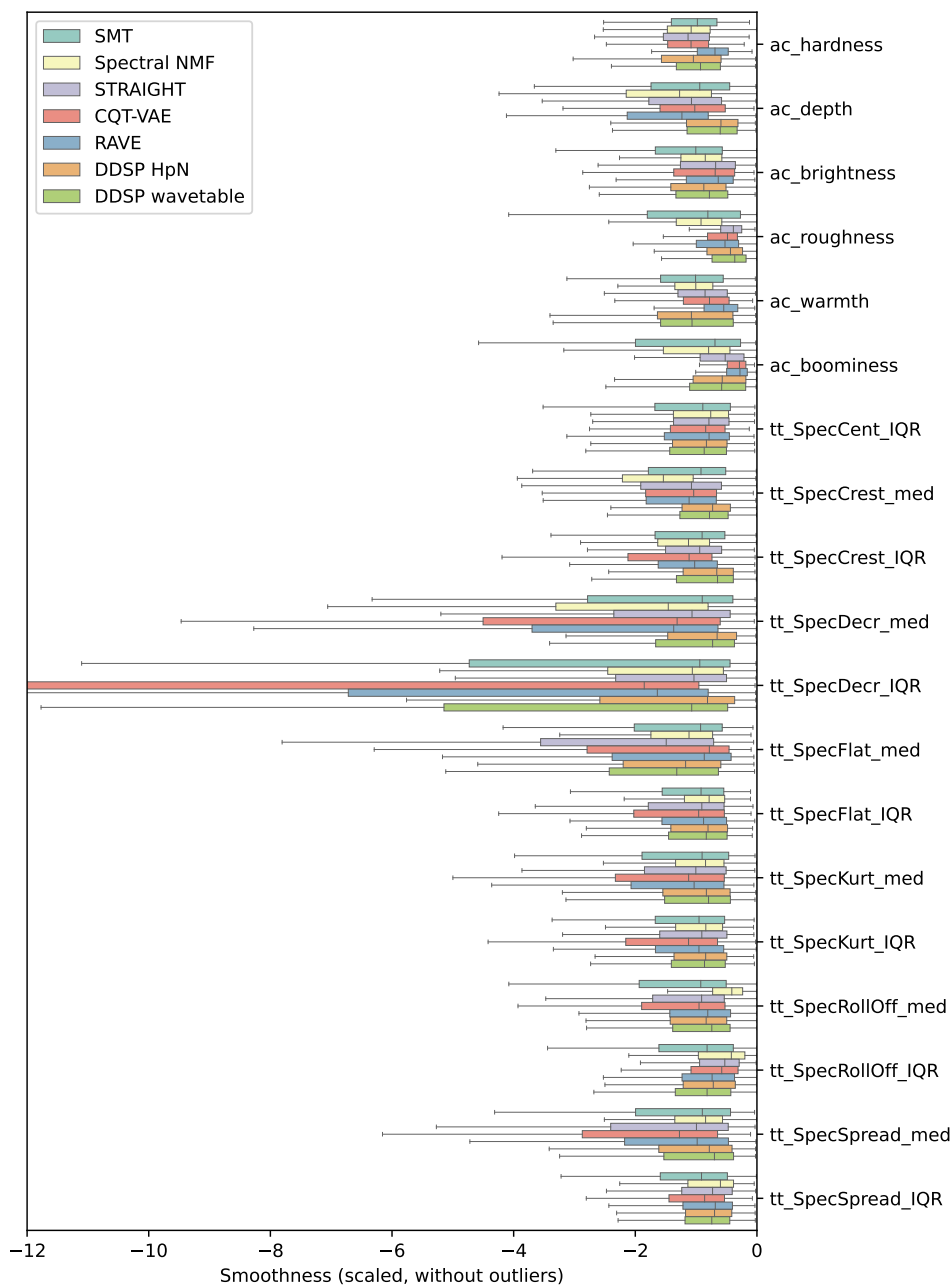


Figure F.3. Smoothness of seven morphing techniques (part 1/2).

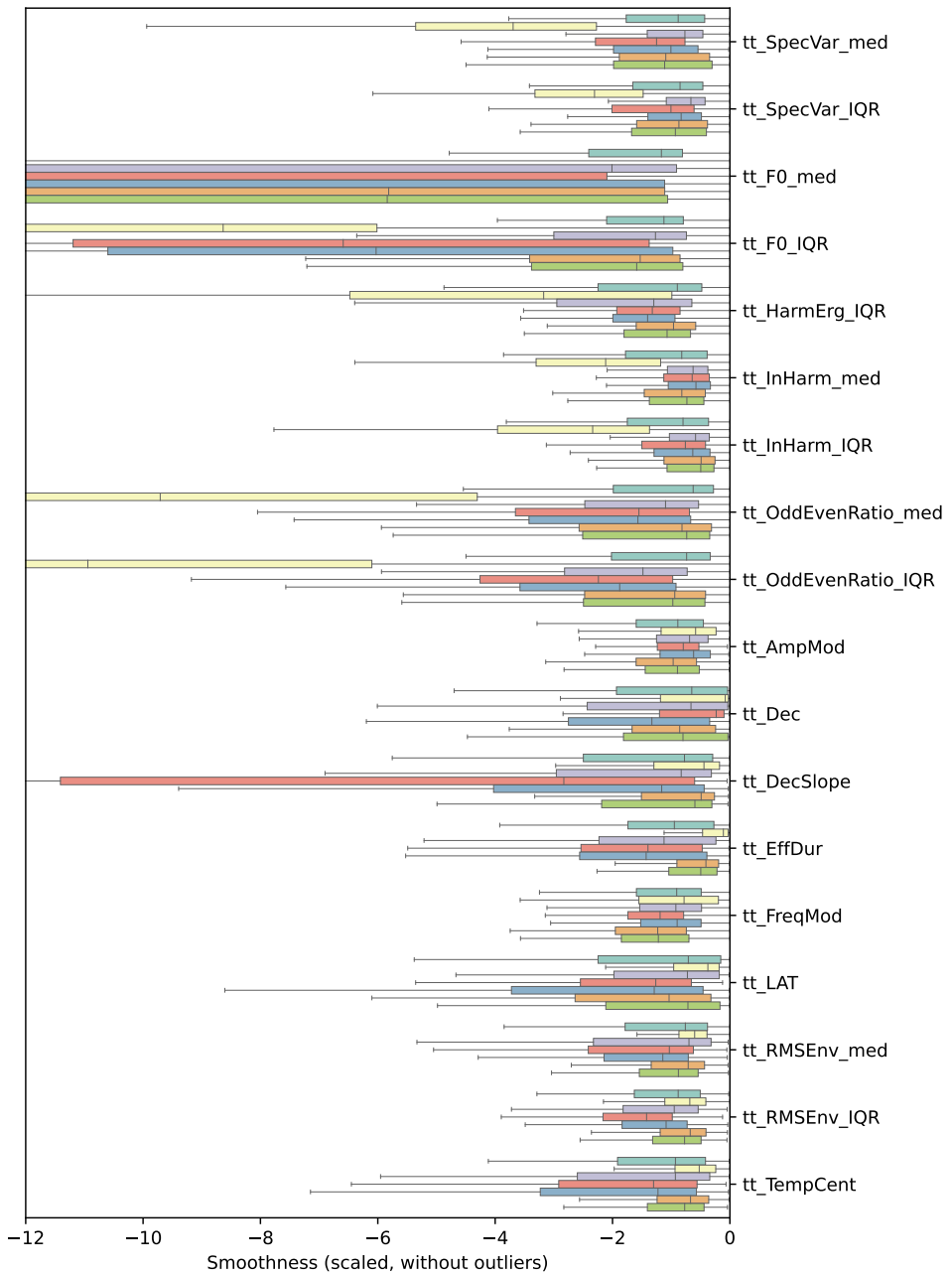


Figure F.4. Smoothness of seven morphing techniques (part 2/2).

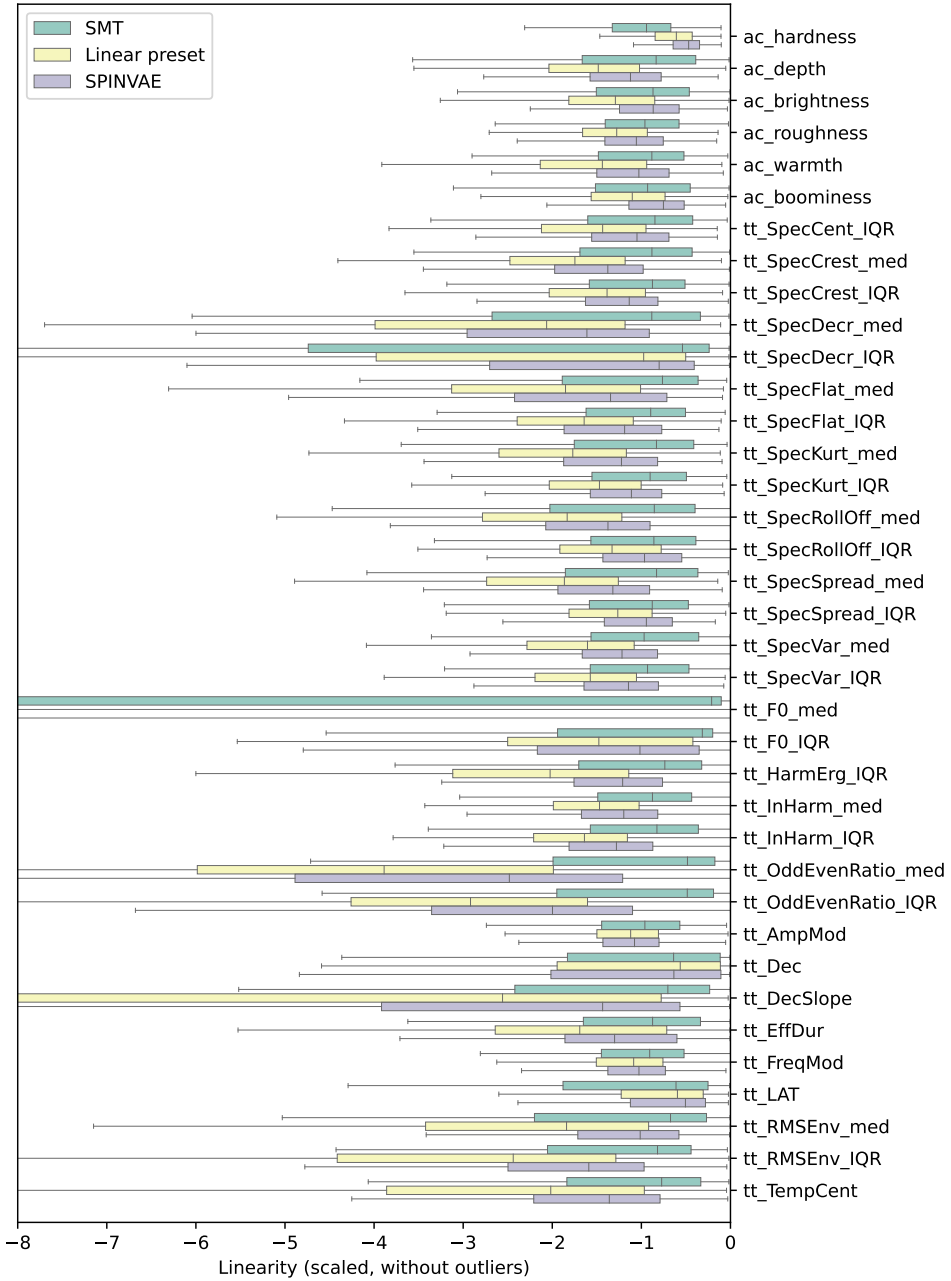


Figure F.5. Linearity of two preset interpolation techniques compared to the SMT baseline

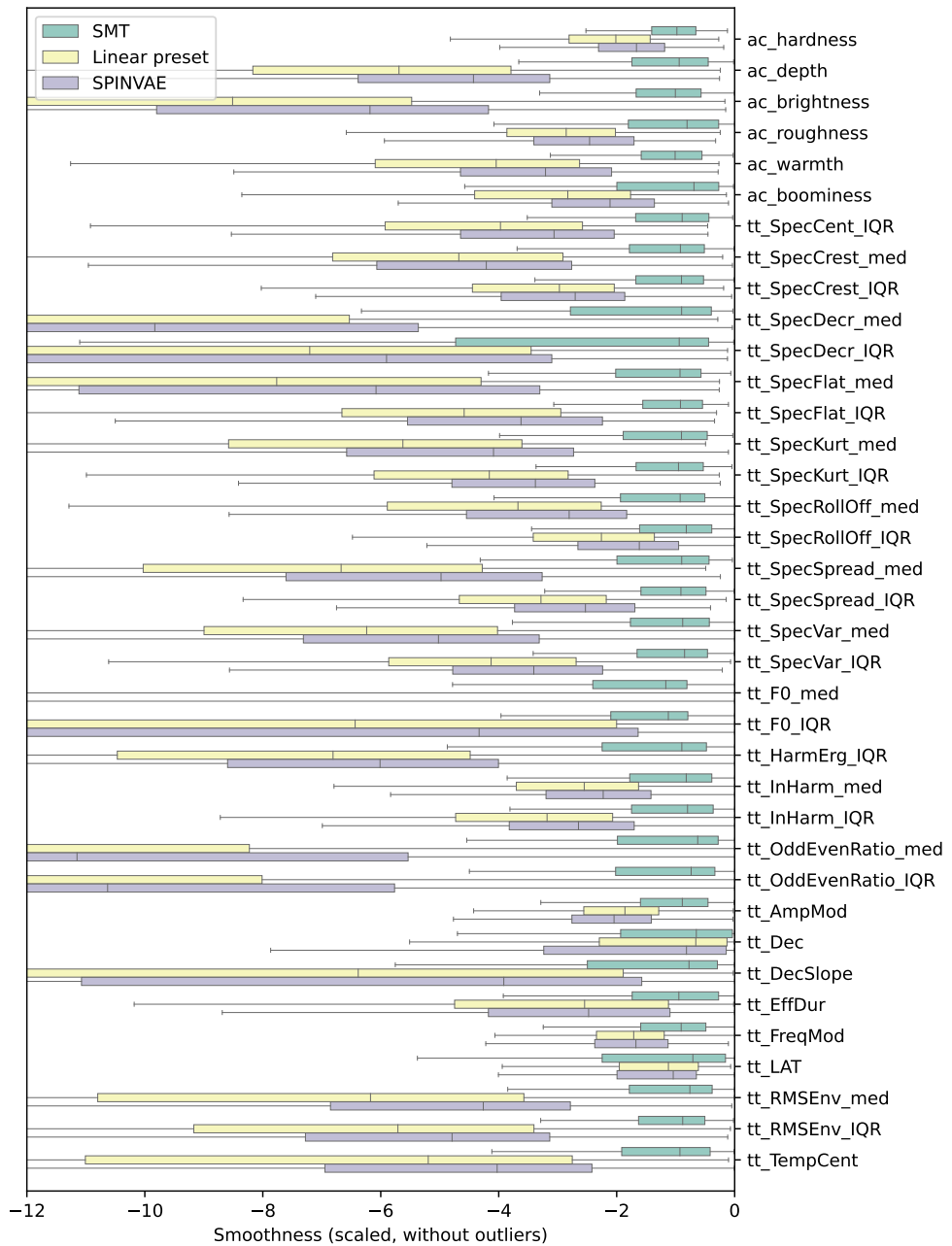


Figure F.6. Smoothness of two preset interpolation techniques compared to the SMT baseline

Appendix G

Decoded trajectories of synthesis parameters

This appendix reports two examples of trajectories of synthesis parameters obtained by [SPINVAE](#). For both examples, some synthesis parameters remained constant during the interpolation and are omitted in the following figures.

The first example is reported in Figures [G.1](#) and [G.2](#). It is an interpolation between two relatively similar presets whose original names were "E.Piano 23" and "B3 Organ 3". They use a relatively similar routing of [FM](#) oscillators (algorithms 6 and 8).

The second example is reported in Figures [G.3](#) and [G.4](#). The corresponding presets are named "AnlgSyn.45" and "ClinkieBel", one being a classic synthesizer sound, while the other is a harmonic percussion. They have very different timbres and use entirely different [FM](#) algorithms. When computing a linear interpolation in the latent space, the corresponding presets decoded by [SPINVAE](#) use different intermediate algorithms rather than switching abruptly from the initial to the final algorithm.

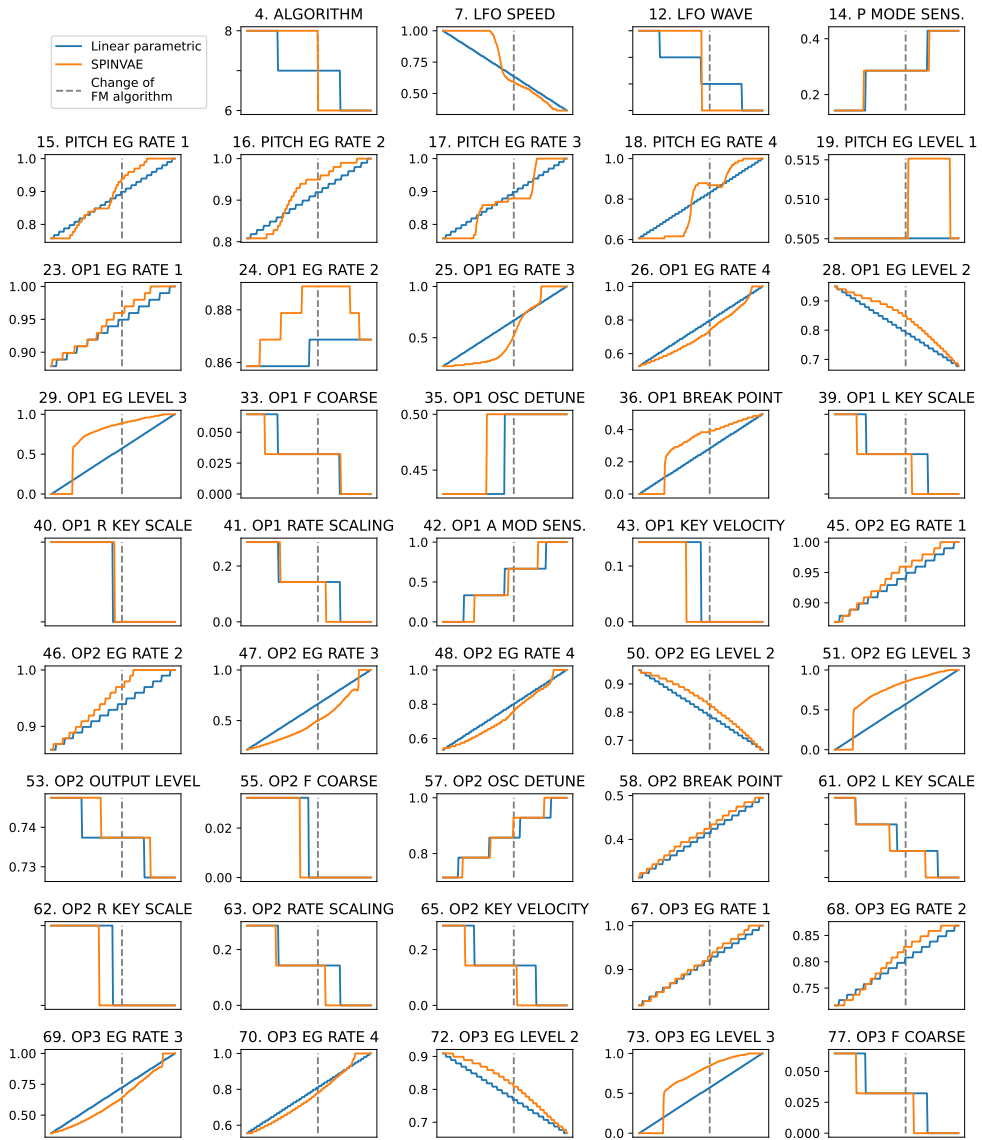


Figure G.1. Trajectories of parameters during a preset interpolation (part 1/2).

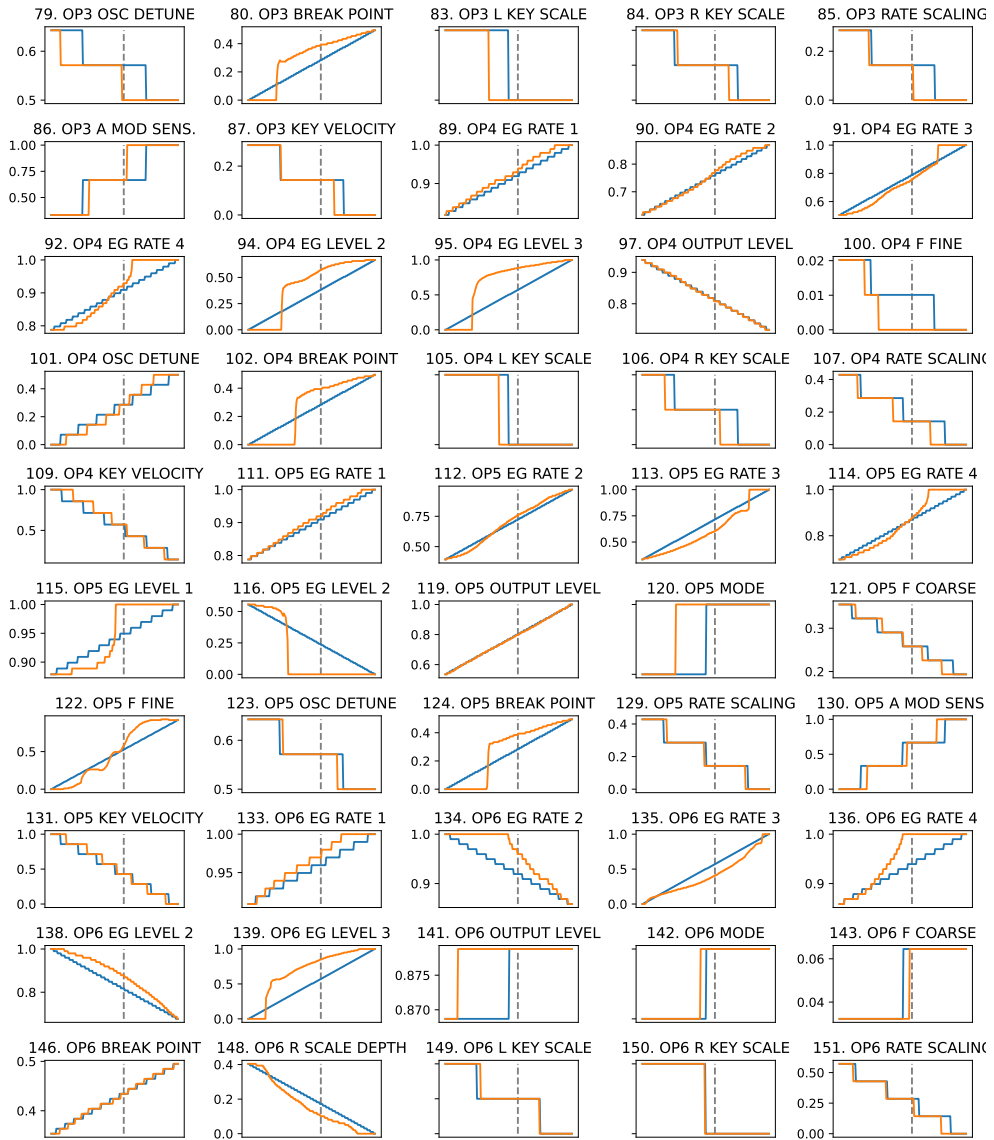


Figure G.2. Trajectories of parameters during a preset interpolation (part 2/2).

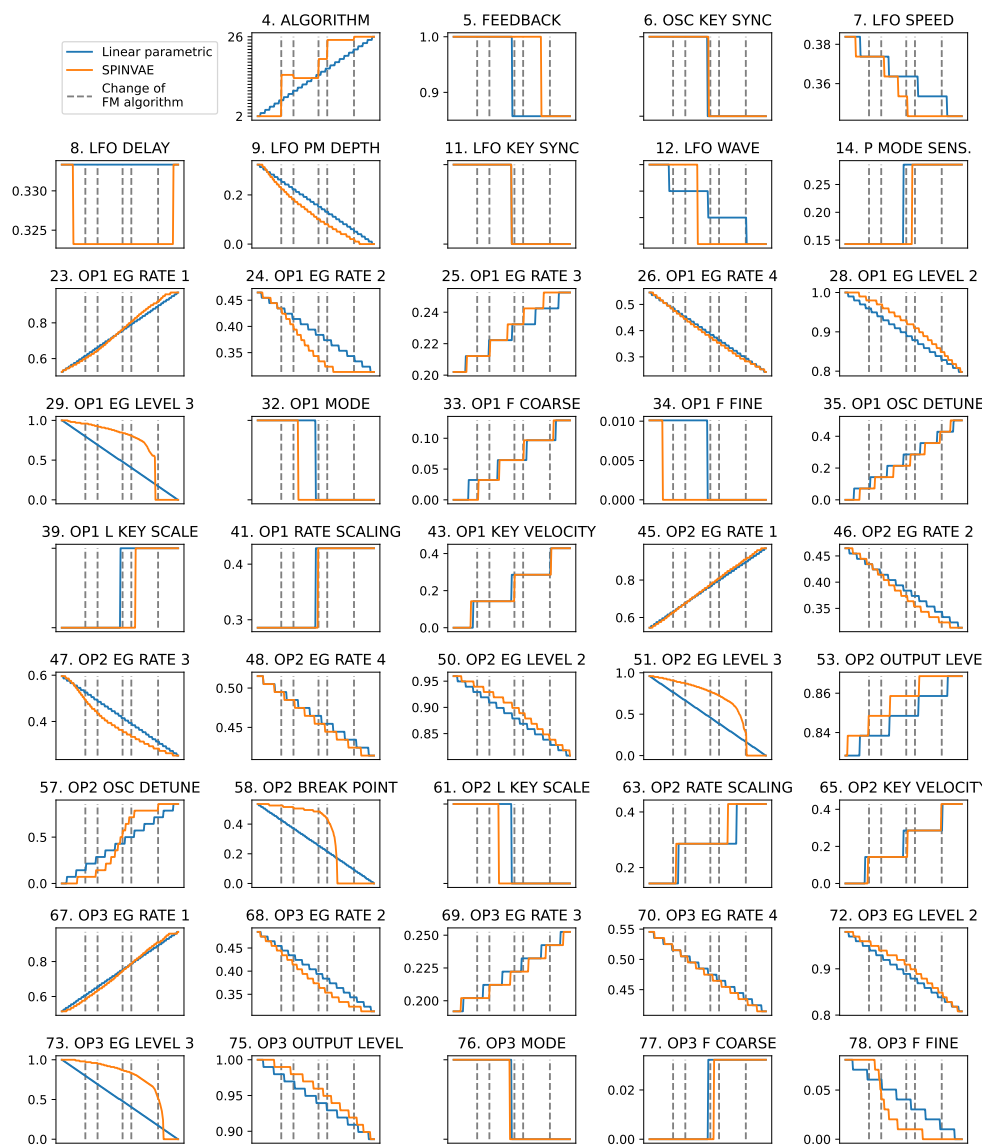


Figure G.3. Trajectories of parameters during a preset interpolation (part 1/2).

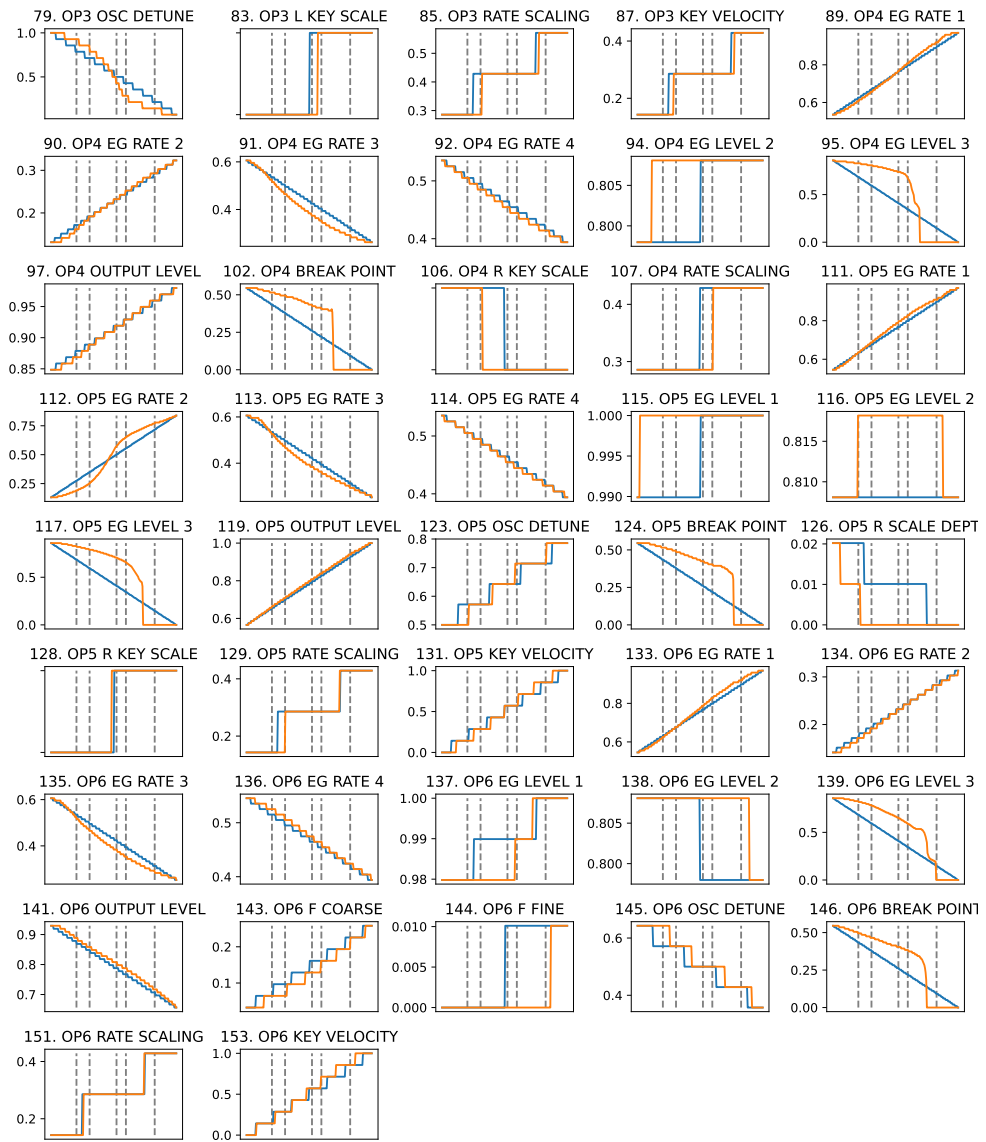


Figure G.4. Trajectories of parameters during a preset interpolation (part 2/2).

Bibliography

- [1] M. Vail, *The synthesizer: a comprehensive guide to understanding, programming, playing, and recording the ultimate electronic music instrument*. Oxford University Press, 2014.
- [2] What's that sound? the classic synthesizers behind 10 iconic tracks. Musikinstrumenten-Museum, Berlin, Germany. (Accessed January 2024). [Online]. Available: <https://artsandculture.google.com/story/bAXhk5CyN9w2KA>
- [3] International Federation of the Phonographic Industry, “Global music report 2024 — state of the industry”. [Online]. Available: <https://globalmusicreport.ifpi.org>
- [4] M. Jenkins, *Analog synthesizers: understanding, performing, buying: from the legacy of Moog to software synthesis*, 2nd ed. Routledge, 2020.
- [5] S. Le Groux and P. Verschure, “Emotional responses to the perceptual dimensions of timbre: A pilot study using physically informed sound synthesis”, in *Proceedings of the 7th International Symposium on Computer Music Modeling*, 2010.
- [6] G. Krekovic, “Insights in habits and attitudes regarding programming sound synthesizers: A quantitative study”, in *Proceedings of the 16th Sound and Music Computing Conference*, 2019.
- [7] K. Tatar, M. Macret, and P. Pasquier, “Automatic synthesizer preset generation with presetgen”, *Journal of New Music Research*,

- vol. 45, no. 2, pp. 124–144, 2016. [Online]. Available: <https://doi.org/10.1080/09298215.2016.1175481>
- [8] N. Masuda and D. Saito, “Improving semi-supervised differentiable synthesizer sound matching for practical applications”, *IEEE/ACM Trans. Audio, Speech, Lang. Process. (TASLP)*, vol. 31, pp. 863–875, 2023.
- [9] C. Donahue, J. McAuley, and M. Puckette, “Adversarial audio synthesis”, in *International Conference on Learning Representations (ICLR)*, 2019.
- [10] B. D. Smith, “Play it again: Evolved audio effects and synthesizer programming”, in *Computational Intelligence in Music, Sound, Art and Design: 6th International Conference, EvoMUSART*, 2017, pp. 275–288.
- [11] S. Carral, “Determining the just noticeable difference in timbre through spectral morphing: A trombone example”, *Acta Acustica united with Acustica*, vol. 97, no. 3, pp. 466–476, 2011.
- [12] J. W. Beauchamp, “Analysis and synthesis of musical instrument sounds”, in *Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music*, J. W. Beauchamp, Ed. Springer New York, 2007, pp. 1–89. [Online]. Available: https://doi.org/10.1007/978-0-387-32576-7_1
- [13] D. P. Kingma and M. Welling, “An introduction to variational autoencoders”, *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019. [Online]. Available: <http://dx.doi.org/10.1561/22000000056>
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.
- [15] “Timbre”, Acoustical Society of America, Acoustical Terminology Database 11.09, 2024. [Online]. Available: <https://asastandards.org/terms/timbre>

- [16] K. Siedenburg, C. Saitis, and S. McAdams, “The present, past, and future of timbre research”, in *Timbre: Acoustics, Perception, and Cognition*, K. Siedenburg, C. Saitis, S. McAdams, A. N. Popper, and R. R. Fay, Eds. Springer International Publishing, 2019, pp. 1–19.
- [17] S. McAdams, S. Winsberg, S. Donnadieu, G. De Soete, and J. Krimphoff, “Perceptual scaling of synthesized musical timbres: Common dimensions, specificities, and latent subject classes”, *Psychological Research*, vol. 58, no. 3, pp. 177–192, 1995.
- [18] C. L. Krumhansl, “Why is musical timbre so hard to understand”, in *Structure and perception of electroacoustic sound and music*, 1989, pp. 43–53.
- [19] K. Siedenburg, I. Fujinaga, and S. McAdams, “A comparison of approaches to timbre descriptors in music information retrieval and music psychology”, *Journal of New Music Research*, vol. 45, no. 1, pp. 27–41, 2016.
- [20] J. M. Grey, “Multidimensional perceptual scaling of musical timbres”, *Journal of the Acoustical Society of America*, vol. 61, no. 5, pp. 1270–1277, 1977.
- [21] J. B. Kruskal, “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis”, *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.
- [22] T. F. Cox and M. A. A. Cox, *Multidimensional scaling*, 2nd ed. Chapman and Hall / CRC press, 2000.
- [23] A. Caclin, S. McAdams, B. K. Smith, and S. Winsberg, “Acoustic correlates of timbre space dimensions: A confirmatory study using synthetic tones”, *The Journal of the Acoustical Society of America*, vol. 118, no. 1, pp. 471–482, 2005.
- [24] S. Samson, R. J. Zatorre, and J. O. Ramsay, “Multidimensional scaling of synthetic musical timbre: Perception of spectral and temporal

- characteristics.” *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, vol. 51, no. 4, p. 307, 1997.
- [25] T. M. Elliott, L. S. Hamilton, and F. E. Theunissen, “Acoustic structure of the five perceptual dimensions of timbre in orchestral instrument tones”, *The Journal of the Acoustical Society of America*, vol. 133, no. 1, pp. 389–404, 01 2013.
- [26] “Pitch”, Acoustical Society of America, Acoustical Terminology Database 11.01, 2024. [Online]. Available: <https://asastandards.org/terms/pitch/>
- [27] “Official midi specifications”, The MIDI Association. [Online]. Available: <https://www.midi.org/specifications>
- [28] E. Javel, “Auditory system, peripheral”, in *Encyclopedia of the Neurological Sciences*, M. J. Aminoff and R. B. Daroff, Eds. New York: Academic Press, 2003, pp. 305–311.
- [29] “Loudness”, Acoustical Society of America, Acoustical Terminology Database 11.03, 2024. [Online]. Available: <https://asastandards.org/terms/loudness/>
- [30] “Algorithms to measure audio programme loudness and true-peak audio level”, International Telecommunication Union, Standard ITU-R BS.1770-5, Nov. 2023.
- [31] T. Todoroff, “Control of digital audio effects”, in *DAFX*, U. Zölzer, Ed. John Wiley and Sons, Ltd, 2002, ch. 12, pp. 465–497.
- [32] M. Wright, “OpenSoundControl Specification 1.0”, Center for Computer Research in Music and Acoustics (CCRMA), 2002. [Online]. Available: <https://ccrma.stanford.edu/groups/osc/spec-1.0.html>
- [33] “M1-100-UM MIDI Polyphonic Expression v1.1”, The MIDI Association. [Online]. Available: <https://midi.org/>

mpe-midi-polyphonic-expression

- [34] I. R. Titze, “Nonlinear source–filter coupling in phonation: Theory”, *The Journal of the Acoustical Society of America*, vol. 123, no. 5, pp. 2733–2749, 05 2008.
- [35] J.-C. Risset and M. V. Mathews, “Analysis of musical-instrument tones”, *Physics today*, vol. 22, no. 2, pp. 23–30, 1969.
- [36] C. Roads, *The computer music tutorial*. MIT press, 1996.
- [37] J. M. Chowning, “The synthesis of complex audio spectra by means of frequency modulation”, *Journal of the Audio Engineering Society*, vol. 21, no. 7, pp. 526–534, 1973.
- [38] K. Nielsen, “Practical linear and exponential frequency modulation for digital music synthesis”, in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx-20)*, Vienna, Austria, 2020, pp. 132–139.
- [39] Yamaha, *DX7 Operating Manual*. [Online]. Available: https://fr.yamaha.com/files/download/other_assets/9/333979/DX7E1.pdf
- [40] T. Nishimoto, “Electronic musical instrument capable of varying a tone synthesis operation algorithm”, Patent US4 554 857A, 1985, Yamaha Corporation.
- [41] Dexed - fm plugin synth. (Accessed January 2024). [Online]. Available: <https://asb2m10.github.io/dexed/>
- [42] M. Lavengood, “What makes it sound ’80s?: The yamaha dx7 electric piano sound”, *Journal of Popular Music Studies*, vol. 31, no. 3, pp. 73–94, 09 2019.
- [43] A. Barahona-Ríos and T. Collins, “NoiseBandNet: Controllable time-varying neural synthesis of sound effects using filterbanks”, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp.

- 1573–1585, 2024.
- [44] Yamaha DX7 audio demos. (Accessed in May 2024). [Online]. Available: <https://www.synthmania.com/dx7.htm>
- [45] K. Karplus and A. Strong, “Digital synthesis of plucked-string and drum timbres”, *Computer Music Journal*, vol. 7, no. 2, pp. 43–55, 1983.
- [46] C. Cadoz, A. Luciani, and J. L. Florens, “Cordis-anima: a modeling and simulation system for sound and image synthesis: the general formalism”, *Computer Music Journal*, vol. 17, no. 1, pp. 19–29, 1993.
- [47] C. Roads, “Introduction to granular synthesis”, *Computer Music Journal*, vol. 12, no. 2, pp. 11–13, 1988.
- [48] D. Hosken, *An introduction to music technology*. Routledge, 2014.
- [49] L. Haken, K. Fitz, and P. Christensen, “Beyond traditional sampling synthesis: Real-time timbre morphing using additive synthesis”, in *Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music*, J. W. Beauchamp, Ed. Springer New York, 2007, pp. 122–144. [Online]. Available: https://doi.org/10.1007/978-0-387-32576-7_3
- [50] A. Seago, S. Holland, and P. Mulholland, “A critical analysis of synthesizer user interfaces for timbre”, *Proceedings of the XVIII British HCI Group Annual Conference HCI 2004*, vol. 2, pp. 105–108, 2004.
- [51] C. Rasmussen, “Evaluating the usability of software synthesizers: An analysis and first approach”, University of Guelph, Ontario, Canada, MSc Thesis, 2018.
- [52] E. R. Miranda, “An artificial intelligence approach to sound design”, *Computer Music Journal*, vol. 19, no. 2, pp. 59–75, 1995.
- [53] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.
- [54] S. E. Stewart and W. J. Strong, “Functional model of a simplified clar-

- inet”, *The Journal of the Acoustical Society of America*, vol. 68, no. 1, pp. 109–120, 1980.
- [55] M. Barthet, P. Guillemain, R. Kronland-Martinet, and S. Ystad, “On the relative influence of even and odd harmonics in clarinet timbre”, in *International Computer Music Conference*, 2005.
- [56] K. M. Kay, “Harmonic analysis of the soprano clarinet”, College of William and Mary in Virginia, BSc Thesis, 2017.
- [57] T. J. Mitchell and D. P. Creasey, “Evolutionary sound matching: A test methodology and comparative study”, in *Sixth International Conference on Machine Learning and Applications (ICMLA)*, 2007, pp. 229–234. [Online]. Available: <https://doi.org/10.1109/ICMLA.2007.34>
- [58] T. Mitchell, “Automated evolutionary synthesis matching”, *Soft Computing*, vol. 16, no. 12, pp. 2057–2070, 2012. [Online]. Available: <https://doi.org/10.1007/s00500-012-0873-x>
- [59] OpenAI, “Gpt-4 technical report”, 2023. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [60] A. Horner, J. Beauchamp, and L. Haken, “Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis”, *Computer Music Journal*, vol. 17, no. 4, pp. 17–29, 1993.
- [61] A. Horner, “Nested modulator and feedback fm matching of instrument tones”, *IEEE Transactions on Speech and Audio Processing*, vol. 6, no. 4, pp. 398–409, 1998. [Online]. Available: <https://doi.org/10.1109/89.701371>
- [62] R. A. Garcia, “Automatic design of sound synthesis techniques by means of genetic programming”, in *Audio Engineering Society Convention 113*, Oct 2002. [Online]. Available: <http://www.aes.org/e-lib/browse.cfm?elib=11250>

- [63] J. Riionheimo and V. Välimäki, “Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation”, *EURASIP Journal on Advances in Signal Processing*, vol. 8, pp. 791–805, 2003.
- [64] M. Yee-King and M. Roth, “Synthbot: An unsupervised software synthesizer programmer”, in *Proceedings of the International Computer Music Conference*, 2008.
- [65] S. Heise, M. Hlatky, and J. Loviscach, “Automatic cloning of recorded sounds by software synthesizers”, in *Audio Engineering Society Convention 127*, Oct 2009. [Online]. Available: <http://www.aes.org/e-lib/browse.cfm?elib=15053>
- [66] R. D. Ashley, “A knowledge-based approach to assistance in timbral design”, in *12th International Computer Music Conference*, 1986.
- [67] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [68] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, “Jukebox: A generative model for music”, 2020, arXiv:2005.00341.
- [69] S. Forsgren and H. Martiros. (2022) Riffusion - Stable diffusion for real-time music generation. [Online]. Available: <https://riffusion.com/about>
- [70] Q. Huang, D. S. Park, T. Wang, T. I. Denk, A. Ly, N. Chen, Z. Zhang, Z. Zhang, J. Yu, C. Frank, J. Engel, Q. V. Le, W. Chan, Z. Chen, and W. Han, “Noise2music: Text-conditioned music generation with diffusion models”, 2023, arXiv:2302.03917v2.
- [71] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, “Neural audio synthesis of musical notes with wavenet autoencoders”, in *International Conference on Machine Learning (ICML)*, 2017, pp. 1068–1077.

- [72] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio”, 2016, arXiv:1609.03499v2.
- [73] The NSynth Dataset. Google Magenta. (Accessed in May 2024). [Online]. Available: <https://magenta.tensorflow.org/datasets/nsynth>
- [74] Y. LeCun, Y. Bengio *et al.*, “Convolutional networks for images, speech, and time series”, *The handbook of brain theory and neural networks*, 1995.
- [75] A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy, “Fixing a broken elbo”, in *International conference on machine learning*. PMLR, 2018, pp. 159–168.
- [76] Open NSynth Super. Google Magenta. (Accessed May 2024). [Online]. Available: <https://github.com/googlecreativelab/open-nsynth-super>
- [77] J. D. Reiss, “A meta-analysis of high resolution audio perceptual evaluation”, *Journal of the Audio Engineering Society*, vol. 64, no. 6, pp. 364–379, 2016.
- [78] L. Hantrakul and L.-C. Yang, “Neural wavetable: a playable wavetable synthesizer using neural networks”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.
- [79] A. Défossez, N. Zeghidour, N. Usunier, L. Bottou, and F. Bach, “Sing: Symbol-to-instrument neural generator”, in *32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018.
- [80] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets”, *Advances in neural information processing systems*, vol. 27, 2014.
- [81] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. C. Courville, “MelGAN: Generative

- adversarial networks for conditional waveform synthesis”, in *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [82] A. Caillon and P. Esling, “Rave: A variational autoencoder for fast and high-quality neural audio synthesis”, 2021, arXiv:2111.05011v2.
- [83] J. Colonel, C. Curro, and S. Keene, “Autoencoding neural networks as musical audio synthesizers”, in *International Conference on Digital Audio Effects (DAFx)*, 2018.
- [84] F. Roche, T. Hueber, M. Garnier, S. Limier, and L. Girin, “Make that sound more metallic: Towards a perceptually relevant control of the timbre of synthesizer sounds using a variational autoencoder”, *Transactions of the International Society for Music Information Retrieval (TISMIR)*, vol. 4, pp. 52–66, 2021.
- [85] K. Tatar, D. Bisig, and P. Pasquier, “Latent timbre synthesis: Audio-based variational auto-encoders for music composition and sound design applications”, *Neural Computing and Applications*, vol. 33, no. 1, p. 67–84, jan 2021. [Online]. Available: <https://doi.org/10.1007/s00521-020-05424-2>
- [86] J. C. Brown, “Calculation of a constant Q spectral transform”, *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.
- [87] L. Girin, F. Roche, T. Hueber, and S. Leglaive, “Notes on the use of variational autoencoders for speech and audio spectrogram modeling”, in *International Conference on Digital Audio Effects (DAFx)*, 2019.
- [88] A. Barahona-Ríos and T. Collins, “SpecSinGAN: Sound Effect Variation Synthesis Using Single-Image GANs”, in *Sound and Music Computing Conference*, 2022.
- [89] D. Griffin and J. Lim, “Signal estimation from modified short-time fourier transform”, *IEEE Transactions on acoustics, speech, and signal*

- processing*, vol. 32, no. 2, pp. 236–243, 1984.
- [90] Z. Pruša and P. L. Søndergaard, “Real-time spectrogram inversion using phase gradient heap integration”, in *Proceedings of the International Conference on Digital Audio Effects (DAFx-16)*, 2016.
- [91] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, “GANSynth: Adversarial neural audio synthesis”, in *International Conference on Learning Representations (ICLR)*, 2019.
- [92] J. Nistal, S. Lattner, and G. Richard, “Drumgan: Synthesis of drum sounds with timbral feature conditioning using generative adversarial networks”, in *Proceedings of the International Society for Music Information Retrieval (ISMIR) Conference*, 2020. [Online]. Available: <https://archives.ismir.net/ismir2020/paper/000255.pdf>
- [93] M. Mirza and S. Osindero, “Conditional generative adversarial nets”, 2014. [Online]. Available: <https://arxiv.org/abs/1411.1784>
- [94] D. Bau, J.-Y. Zhu, J. Wulff, W. Peebles, H. Strobel, B. Zhou, and A. Torralba, “Seeing what a gan cannot generate”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.
- [95] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, “Ddsp: Differentiable digital signal processing”, in *International Conference on Learning Representations (ICLR)*, 2020.
- [96] X. Serra and J. Smith, “Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition”, *Computer Music Journal*, vol. 14, no. 4, pp. 12–24, 1990.
- [97] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, “CREPE: A convolutional representation for pitch estimation”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 161–165.

- [98] B. Logan, “Mel frequency cepstral coefficients for music modeling.” in *International Society for Music Information Retrieval (ISMIR) Conference*, 2000.
- [99] M. Carney, C. Li, E. Toh, N. Zada, P. Yu, and J. H. Engel, “Tone transfer: In-browser interactive neural audio synthesis.” in *Joint Proceedings of the ACM IUI Workshops*, 2021.
- [100] A. Vinay and A. Lerch, “Evaluating generative audio systems and their metrics”, *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pp. 858–865, 2022.
- [101] Y. Wu, E. Manilow, Y. Deng, R. Swavely, K. Kastner, T. Cooijmans, A. Courville, C.-Z. A. Huang, and J. Engel, “MIDI-DDSP: Detailed control of musical performance via hierarchical modeling”, in *International Conference on Learning Representations*, 2022.
- [102] DDSP-VST: Neural Synthesis in your DAW. Google Magenta. (Accessed May 2024). [Online]. Available: <https://magenta.tensorflow.org/ddsp-vst>
- [103] F. Caspe, A. McPherson, and M. Sandler, “DDX7: Differentiable FM Synthesis of Musical Instrument Sounds”, *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pp. 608–616, 2022.
- [104] B. Li, X. Liu, K. Dinesh, Z. Duan, and G. Sharma, “Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications”, *IEEE Transactions on Multimedia*, vol. 21, no. 2, p. 522–535, feb 2019. [Online]. Available: <https://doi.org/10.1109/TMM.2018.2856090>
- [105] F. Caspe, A. McPherson, and M. Sandler, “FM Tone Transfer with Envelope Learning”, *Proceedings of Audio Mostly*, 2023.
- [106] B. Hayes, C. Saitis, and G. Fazekas, “Neural waveshaping synthesis”,

- Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2021.
- [107] S. Shan, L. Hantrakul, J. Chen, M. Avent, and D. Trevelyan, “Differentiable wavetable synthesis”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 4598–4602.
- [108] L. Renault, R. Mignot, and A. Roebel, “DDSP-Piano: a neural sound synthesizer informed by instrument knowledge”, *Journal of the Audio Engineering Society*, vol. 71, pp. 552–565, september 2023.
- [109] A. Wiggins and Y. Kim, “A differentiable acoustic guitar model for string-specific polyphonic synthesis”, in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2023.
- [110] J. Shier, F. Caspe, A. Robertson, M. Sandler, C. Saitis, and A. McPherson, “Differentiable modelling of percussive audio with transient and spectral synthesis”, in *Forum Acusticum - 10th Convention of the European Acoustics Association*, 2023.
- [111] M. Yee-King and L. McCallum, “Studio report: sound synthesis with DDSP and network bending techniques”, in *2nd Conference on AI Music Creativity*, 2021.
- [112] “Method for the subjective assessment of intermediate quality level of audio systems”, International Telecommunication Union, Recommendation ITU-R BS.1534-3, Oct. 2015.
- [113] B. Hayes, J. Shier, G. Fazekas, A. McPherson, and C. Saitis, “A review of differentiable digital signal processing for music and speech synthesis”, *Frontiers in Signal Processing*, vol. 3, 2024. [Online]. Available: <https://doi.org/10.3389/frsip.2023.1284100>
- [114] C.-Y. Yu, C. Mitcheltree, A. Carson, S. Bilbao, J. D. Reiss, and G. Fazekas, “Differentiable all-pole filters for time-varying audio systems”, 2024. [Online]. Available: <https://arxiv.org/abs/2404>.

07970v2

- [115] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, “Style transfer of audio effects with differentiable signal processing”, *Journal of the Audio Engineering Society*, vol. 70, no. 9, pp. 708–721, 2022. [Online]. Available: <http://www.aes.org/e-lib/browse.cfm?elib=21883>
- [116] C. Mitcheltree and H. Koike, “Serumrnn: Step by step audio vst effect programming”, in *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, 2021, pp. 218–234.
- [117] J. W. Beauchamp, “Synthesis by amplitude and brightness matching of analyzed musical instrument tones”, in *Audio Engineering Society Convention 69*, 1981. [Online]. Available: <https://www.aes.org/e-lib/browse.cfm?elib=3837>
- [118] J. Beauchamp, “Practical sound synthesis using a nonlinear processor (waveshaper) and a high-pass filter”, *Computer Music Journal*, vol. 3, no. 3, pp. 42–49, 1979.
- [119] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992.
- [120] J. Kennedy and R. Eberhart, “Particle swarm optimization”, in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.
- [121] A. Horner, N.-M. Cheung, and J. Beauchamp, “Genetic algorithm optimization of additive synthesis envelope breakpoints and group synthesis parameters”, 1995.
- [122] C.-W. Wun, A. B. Horner, and L. Ayers, “Perceptual wavetable matching for synthesis of musical instrument tones”, in *International Conference on Mathematics and Computing*, 2001.

- [123] J. McDermott, N. J. Griffith, and M. O’Neill, “Toward user-directed evolution of sound synthesis parameters”, in *Applications of Evolutionary Computing: EvoWorkshops Proceedings*, 2005, pp. 517–526.
- [124] B. Bozkurt and K. A. Yüksel, “Parallel evolutionary optimization of digital sound synthesis parameters”, in *Applications of Evolutionary Computation*. Springer Berlin Heidelberg, 2011, pp. 194–203.
- [125] S. Wilson, D. Cottle, and N. Collins, *The SuperCollider Book*. The MIT Press, 2011.
- [126] M. Macret, “Automatic tuning of the op-1 synthesizer using a multi-objective genetic algorithm”, Simon Fraser University, MSc Thesis, 2013.
- [127] Op-1 portable synthesizer. Teenage Engineering. (Accessed February 2024). [Online]. Available: <https://teenage.engineering/products/op-1/original>
- [128] M. Macret and P. Pasquier, “Automatic design of sound synthesizers as pure data patches using coevolutionary mixed-typed cartesian genetic programming”, in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’14, 2014, pp. 309–316. [Online]. Available: <https://doi.org/10.1145/2576768.2598303>
- [129] M. Puckette, “Pure data”, in *Proceedings of the International Computer Music Conference*, 1997.
- [130] Z. Ye, W. Xue, X. Tan, Q. Liu, and Y. Guo, “NAS-FM: Neural architecture search for tunable and interpretable sound synthesis based on frequency modulation”, in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, 8 2023, pp. 5869–5877, aI and Arts. [Online]. Available: <https://doi.org/10.24963/ijcai.2023/651>
- [131] M. J. Yee-King, L. Fedden, and M. d’Inverno, “Automatic programming of VST sound synthesizers using deep networks and other

- techniques”, *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 150–159, 2018. [Online]. Available: <https://doi.org/10.1109/TETCI.2017.2783885>
- [132] H. Renney, B. Gaster, and T. J. Mitchell, “Survival of the synthesis—gpu accelerating evolutionary sound matching”, *Concurrency and Computation: Practice and Experience*, vol. 34, no. 10, 2022.
- [133] K. Itoyama and H. Okuno, “Parameter estimation of virtual musical instrument synthesizers”, in *International Computer Music Conference*, 2014. [Online]. Available: <http://hdl.handle.net/2027/spo.bbp2372.2014.217>
- [134] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, “Flow synthesizer: Universal audio synthesizer control with normalizing flows”, *Applied Sciences*, vol. 10, no. 1, 2020.
- [135] O. Barkan, D. Tsiris, O. Katz, and N. Koenigstein, “Inversynth: Deep estimation of synthesizer parameter configurations from audio signals”, *IEEE/ACM Trans. Audio, Speech, Lang. Process. (TASLP)*, vol. 27, no. 12, pp. 2385–2396, 2019.
- [136] Z. Chen, Y. Jing, S. Yuan, Y. Xu, J. Wu, and H. Zhao, “Sound2Synth: Interpreting sound via FM synthesizer parameters estimation”, in *International Joint Conferences on Artificial Intelligence*, 2022, pp. 4921–4928. [Online]. Available: <https://doi.org/10.24963/ijcai.2022/682>
- [137] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [138] H. Terasawa, M. Slaney, and J. Berger, “Perceptual distance in timbre space”, in *Proceedings of the International Conference on Auditory Display (ICAD)*, 2005.
- [139] M. Müller, “Music synchronization”, in *Fundamentals of Music Processing: Using Python and Jupyter Notebooks*. Springer International

- Publishing, 2021, pp. 119–170.
- [140] M. Roth and M. Yee-King, “A comparison of parametric optimization techniques for musical instrument tone matching”, *Journal of the Audio Engineering Society*, no. 8418, May 2011. [Online]. Available: <https://aes2.org/publications/elibrary-page/?id=15885>
- [141] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling”, in *NeurIPS Deep Learning and Representation Learning Workshop*, 2014. [Online]. Available: <https://doi.org/10.48550/arXiv.1412.3555>
- [142] H. Han, V. Lostanlen, and M. Lagrange, “Learning to solve inverse problems for perceptual sound matching”, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 2605–2615, 2024. [Online]. Available: <https://doi.org/10.1109/TASLP.2024.3393738>
- [143] J. Andén, V. Lostanlen, and S. Mallat, “Joint time–frequency scattering”, *IEEE Transactions on Signal Processing*, vol. 67, no. 14, pp. 3704–3718, 2019. [Online]. Available: <https://doi.org/10.1109/TSP.2019.2918992>
- [144] J. Shier, G. Tzanetakis, and K. McNally, “Spiegelib: An automatic synthesizer programming library”, in *Audio Engineering Society Convention 148*, May 2020. [Online]. Available: <http://www.aes.org/e-lib/browse.cfm?elib=20794>
- [145] M. A. Martínez Ramírez, O. Wang, P. Smaragdis, and N. J. Bryan, “Differentiable signal processing with black-box audio effects”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 66–70.
- [146] D. P. Kingma and M. Welling, “Auto-encoding variational bayes”, *International Conference on Learning Representations (ICLR)*, 2014.
- [147] G. Le Vaillant, T. Dutoit, and S. Dekeyser, “Improving synthesizer pro-

- gramming from variational autoencoders latent space”, in *24th International Conference on Digital Audio Effects (DAFx)*, 2021, pp. 276–283.
- [148] D. Faronbi, I. Roman, and J. P. Bello, “Exploring approaches to multi-task automatic synthesizer programming”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023. [Online]. Available: <https://doi.org/10.1109/ICASSP49357.2023.10095540>
- [149] Y. M. Siriwardena, G. Marion, and S. Shamma, “The MirrorNet: Learning audio synthesizer controls inspired by sensorimotor interaction”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 946–950. [Online]. Available: <https://doi.org/10.1109/ICASSP43922.2022.9747358>
- [150] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference”, *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 2617–2680, 2021.
- [151] Synplant 2.0 User Guide. NuEdge Development. (Accessed May 2024). [Online]. Available: <https://soniccharge.com/public/Synplant%20User%20Guide.pdf>
- [152] MicroMusic: Vital Preset Generator. (Accessed May 2024). [Online]. Available: <https://micromusic.tech>
- [153] M. Cherep, N. Singh, and J. Shand, “Creative text-to-audio generation via synthesizer programming”, in *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 21–27 Jul 2024, pp. 8270–8285. [Online]. Available: <https://proceedings.mlr.press/v235/cherep24a.html>
- [154] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in python.” in

- Proceedings of the 14th python in science conference*, 2015, pp. 18–25.
- [155] D. Fitzgerald, “Harmonic/percussive separation using median filtering”, in *International Conference on Digital Audio Effects (DAFx)*, 2010.
- [156] J. Driedger, M. Müller, and S. Disch, “Extending harmonic-percussive separation of audio signals”, in *International Society for Music Information Retrieval (ISMIR) Conference*, 2014, pp. 611–616.
- [157] G. Navarro, “A guided tour to approximate string matching”, *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.
- [158] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, “Understanding disentangling in β -vae”, in *NeurIPS Workshop on Learning Disentangled Representations*, 2017. [Online]. Available: <https://arxiv.org/abs/1804.03599>
- [159] D. J. Rezende and F. Viola, “Taming VAEs”, 2018. [Online]. Available: <https://arxiv.org/abs/1810.00597>
- [160] S. Zhao, J. Song, and S. Ermon, “InfoVAE: Balancing learning and inference in variational autoencoders”, in *The Thirty-Third AAAI Conference on Artificial Intelligence*, 2019, pp. 5885–5892. [Online]. Available: <https://doi.org/10.1609/aaai.v33i01.33015885>
- [161] D. Rezende and S. Mohamed, “Variational inference with normalizing flows”, in *International Conference on Machine Learning*. PMLR, 2015, pp. 1530–1538.
- [162] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, “Improved variational inference with inverse autoregressive flow”, in *Advances in Neural Information Processing Systems*, vol. 29, 2016. [Online]. Available: <https://proceedings.neurips.cc/paper/2016/file/ddeebdeefdb7e7e7a697e1c3e3d8ef54-Paper.pdf>
- [163] G. Papamakarios, T. Pavlakou, and I. Murray, “Masked au-

- toregressive flow for density estimation”, in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf>
- [164] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using Real NVP”, *International Conference on Learning Representations*, 2017.
- [165] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models”, in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, 2020, pp. 6840–6851. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf
- [166] S. Ioffe and C. Szegedy, “Batch normalization: accelerating deep network training by reducing internal covariate shift”, in *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, 2015, p. 448–456.
- [167] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework”, *International Conference on Learning Representations*, 2017.
- [168] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library”, in *Advances in Neural Information Processing Systems*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

- [169] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *3rd International Conference for Learning Representations (ICLR)*, 2015.
- [170] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, “Ladder variational autoencoders”, in *Advances in Neural Information Processing Systems*, 2016. [Online]. Available: <https://proceedings.neurips.cc/paper/2016/file/6ae07dcb33ec3b7c814df797cbda0f87-Paper.pdf>
- [171] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators”, *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0893608089900208>
- [172] S. M. Ali Eslami *et al.*, “Neural scene representation and rendering”, *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018. [Online]. Available: <https://science.sciencemag.org/content/360/6394/1204>
- [173] P. Masri, A. Bateman, and N. Canagarajah, “A review of time–frequency representations, with application to sound/music analysis–resynthesis”, *Organised Sound*, vol. 2, no. 3, p. 193–205, 1997.
- [174] E. Tellman, L. Haken, and B. Holloway, “Timbre morphing of sounds with unequal numbers of features”, *Journal of the Audio Engineering Society*, vol. 43, no. 9, pp. 678–689, 1995.
- [175] K. Fitz, L. Haken, S. Lefvert, C. Champion, and M. O’Donnell, “Cell-Utes and Flutter-Tongued Cats: Sound Morphing Using Loris and the Reassigned Bandwidth-Enhanced Model”, *Computer Music Journal*, vol. 27, no. 3, pp. 44–65, 09 2003.
- [176] S. Kazazis, P. Depalle, and S. McAdams, “Sound morphing by audio descriptors and parameter interpolation”, in *International Conference on Digital Audio Effects*, 2016, pp. 145–151.

- [177] D. O’Shaughnessy, “Linear predictive coding”, *IEEE Potentials*, vol. 7, no. 1, pp. 29–32, 1988.
- [178] T. Brookes and D. Williams, “Perceptually-motivated audio morphing: Brightness”, in *Audio Engineering Society Convention 122*, May 2007. [Online]. Available: <https://www.aes.org/e-lib/browse.cfm?elib=14020>
- [179] D. Williams and T. Brookes, “Perceptually-motivated audio morphing: Softness”, in *Audio Engineering Society Convention 126*, May 2009. [Online]. Available: <https://www.aes.org/e-lib/browse.cfm?elib=14974>
- [180] T. Brookes and D. Williams, “Perceptually-motivated audio morphing: Warmth”, in *Audio Engineering Society Convention 128*, May 2010. [Online]. Available: <https://www.aes.org/e-lib/browse.cfm?elib=15316>
- [181] X. Rodet and D. Schwarz, “Spectral envelopes and additive + residual analysis/synthesis”, in *Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music*, J. W. Beauchamp, Ed. Springer New York, 2007, pp. 175–227. [Online]. Available: https://doi.org/10.1007/978-0-387-32576-7_5
- [182] C. Scaletti, “Computer music languages, Kyma, and the future”, *Computer Music Journal*, vol. 26, no. 4, pp. 69–82, 2002.
- [183] M. Caetano, “Morphing musical instrument sounds with the sinusoidal model in the sound morphing toolbox”, in *Perception, Representations, Image, Sound, Music*. Springer International Publishing, 2021.
- [184] M. Caetano and X. Rodet, “Sound morphing by feature interpolation”, in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011.
- [185] AudioSculpt 3.0 User Manual. IRCAM. (Accessed September 2024). [Online]. Available: <https://support.ircam.fr/docs/AudioSculpt/3.0/co/AudioSculptguideWeb.html>

- [186] M. Caetano and X. Rodet, “A source-filter model for musical instrument sound transformation”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 137–140.
- [187] —, “Musical instrument sound morphing guided by perceptually motivated features”, *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 21, no. 8, pp. 1666–1675, 2013.
- [188] H. Kawahara, H. Banno, T. Irino, and P. Zolfaghari, “Algorithm amalgam: morphing waveform based methods, sinusoidal models and STRAIGHT”, in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2004.
- [189] W. Ahmad, H. Hacihabiboglu, and A. M. Kondo, “Morphing of transient sounds based on shift-invariant discrete wavelet transform and singular value decomposition”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2009, pp. 297–300.
- [190] S. Siddiq, “Real-time morphing of impact sounds”, in *Audio Engineering Society Convention 139*, Oct 2015.
- [191] —, “Morphing of granular sounds”, in *International Conference on Digital Audio Effects*, 2015.
- [192] A. Primavera, F. Piazza, and J. D. Reiss, “Audio morphing for percussive hybrid sound generation”, in *Audio Engineering Society Conference: 45th International Conference*, Mar 2012.
- [193] M. Slaney, M. Covell, and B. Lassiter, “Automatic audio morphing”, in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1996.
- [194] A. Olivero, P. Depalle, B. Torr sani, and R. Kronland-Martinet, “Sound morphing strategies based on alterations of time-frequency representations by gabor multipliers”, in *Audio Engineering Society Conference: 45th International Conference*, Mar 2012.

- [195] G. Roma, O. Green, and P. A. Tremblay, “Audio morphing using matrix decomposition and optimal transport”, in *International Conference on Digital Audio Effects (DAFx)*, 2020.
- [196] J. W. Kim, R. Bittner, A. Kumar, and J. P. Bello, “Neural music synthesis for flexible timbre control”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 176–180.
- [197] C. Gupta, P. Kamath, Y. Wei, Z. Li, S. Nanayakkara, and L. Wyse, “Towards controllable audio texture morphing”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [198] Y. Zou, J. Liu, and W. Jiang, “Non-parallel and many-to-one musical timbre morphing using ddsp-autoencoder and spectral feature interpolation”, in *International Conference on Culture-oriented Science and Technology (ICCST)*, 2021, pp. 144–148.
- [199] D. Berthelot, C. Raffel, A. Roy, and I. Goodfellow, “Understanding and improving interpolation in autoencoders via an adversarial regularizer”, in *International Conference on Learning Representations (ICLR)*, 2019.
- [200] T. Ezzat, E. Meyers, J. Glass, and T. Poggio, “Morphing spectral envelopes using audio flow”, in *Proc. Interspeech*, 2005, pp. 2545–2548.
- [201] M. Caetano and N. Osaka, “A formal evaluation framework for sound morphing”, in *Proceedings of the International Computer Music Conference*, 2012.
- [202] M. Caetano, C. Saitis, and K. Siedenburg, “Audio content descriptors of timbre”, in *Timbre: Acoustics, Perception, and Cognition*, K. Siedenburg, C. Saitis, S. McAdams, A. N. Popper, and R. R. Fay, Eds. Springer International Publishing, 2019, pp. 297–333.
- [203] G. Peeters, B. Giordano, P. Susini, and N. Misdariis, “The timbre toolbox: Extracting audio descriptors from musical signals”, in *Journal of the Acoustical Society of America*, vol. 130, no. 5, 2011.

- [204] A. Pearce, S. Safavi, T. Brookes, R. Mason, W. Wang, and M. Plumbley, “Timbral characterisation tools for semantically annotating non-musical content”, in *AudioCommons project, Deliverable 5.8*, January 2019.
- [205] M. Petrovich, M. J. Black, and G. Varol, “Action-conditioned 3D human motion synthesis with transformer VAE”, in *International Conference on Computer Vision (ICCV)*, 2021.
- [206] P. Esling, A. Chemla-Romeu-Santos, and A. Bitton, “Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics”, *International Conference on Digital Audio Effects (DAFx)*, pp. 369–376, 2018.
- [207] A. Oring, Z. Yakhini, and Y. Hel-Or, “Autoencoder image interpolation by shaping the latent space”, in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 8281–8290. [Online]. Available: <https://proceedings.mlr.press/v139/oring21a.html>
- [208] S. M. Bernsee and D. Gökdag, “Methods for extending frequency transforms to resolve features in the spatio-temporal domain”, Patent EP 3 271 736 B1, 2016.
- [209] R. Huber and B. Kollmeier, “PEMO-Q—A new method for objective audio quality assessment using a model of auditory perception”, *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 6, pp. 1902–1911, 2006.
- [210] B. Evans, *FM synthesis and morphing in Csound: from percussion to brass*. MIT Press, 2000, p. 198–206.
- [211] Native Instruments, “FM8 Manual”, https://www.native-instruments.com/fileadmin/ni_media/producer/fm8/downloads/FM8-book_EN_ebook.pdf, 2006, accessed November 2023.
- [212] M. Blaauw and J. Bonada, “Modeling and transforming speech using variational autoencoders”, in *Proc. Interspeech*, 2016, pp. 1770–1774.

- [213] S. N. Levine and J. O. Smith III, “Spectral envelopes and additive + residual analysis/synthesis”, in *Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music*, J. W. Beauchamp, Ed. Springer New York, 2007, pp. 145–174. [Online]. Available: https://doi.org/10.1007/978-0-387-32576-7_4
- [214] P. Lippe and E. Gavves, “Categorical normalizing flows via continuous transformations”, in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: <https://openreview.net/forum?id=-GLNZeVDuik>
- [215] Alexey Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale”, in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: <https://iclr.cc/virtual/2021/oral/3458>
- [216] Y. Gong, Y.-A. Chung, and J. Glass, “AST: Audio spectrogram transformer”, in *Interspeech*, 2021.
- [217] J. Born and M. Manica, “Regression transformer: Concurrent conditional generation and regression by blending numerical and textual tokens”, in *ICLR 2022 Workshop for Machine Learning for Drug Discovery*, 2022.
- [218] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions”, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
- [219] J. Jiang, G. G. Xia, D. B. Carlton, C. N. Anderson, and R. H. Miyakawa, “Transformer VAE: A hierarchical model for structure-aware and interpretable music representation learning”, in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2020.
- [220] S. Raschka, *Build a Large Language Model (From Scratch)*. Manning,

2024.

- [221] A. Fan, M. Lewis, and Y. Dauphin, “Hierarchical neural story generation”, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 889–898. [Online]. Available: <https://aclanthology.org/P18-1082>
- [222] A. Vijayakumar, M. Cogswell, R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra, “Diverse beam search for improved description of complex scenes”, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/12340>
- [223] C. J. Steinmetz, J. Pons, S. Pascual, and J. Serrà, “Automatic multi-track mixing with a differentiable mixing console of neural audio effects”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 71–75.
- [224] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, “PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications”, in *3rd International Conference for Learning Representations (ICLR)*, 2017.
- [225] G. Le Vaillant and T. Dutoit, “Synthesizer preset interpolation using transformer auto-encoders”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [226] A. Pati and A. Lerch, “Is disentanglement enough? on latent representations for controllable music generation”, in *Proceedings of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*, 2021, pp. 517–524.
- [227] —, “Attribute-based regularization of latent spaces for variational auto-encoders”, *Neural Comput. Appl.*, vol. 33, no. 9, pp. 4429–4444, 2021.

- [228] F. Locatello, M. Tschannen, S. Bauer, G. Rätsch, B. Schölkopf, and O. Bachem, “Disentangling factors of variations using few labels”, in *International Conference on Learning Representations (ICLR)*, 2020.
- [229] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin, “Cyclical annealing schedule: A simple approach to mitigating kl vanishing”, *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.
- [230] L. N. Smith, “Cyclical learning rates for training neural networks”, in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 464–472.
- [231] T. White, “Sampling generative networks”, 2016, arXiv:1609.04468v3.
- [232] G. Villée and G. Le Vaillant, “Music making mobile application using geometry”, in *Proceedings of the International Computer Music Conference*, 2018, pp. 276–283. [Online]. Available: <https://quod.lib.umich.edu/i/icmc/bbp2372.2018.054/1>
- [233] G. Le Vaillant, T. Dutoit, and R. Giot, “Analytic vs. holistic approaches for the live search of sound presets using graphical interpolation”, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, July 2020, pp. 227–232.
- [234] M. J. Yee-King, “The evolving drum machine”, in *Music-AL workshop, European Conference on Artificial Life*, 2007.
- [235] A. Soemer and S. Saito, “Maintenance of auditory-nonverbal information in working memory”, *Psychonomic Bulletin & Review*, vol. 22, no. 6, pp. 1777–1783, 2015. [Online]. Available: <https://doi.org/10.3758/s13423-015-0854-z>
- [236] K. J. Pallesen, E. Brattico, C. J. Bailey, A. Korvenoja, J. Koivisto, A. Gjedde, and S. Carlson, “Cognitive control in auditory working memory is enhanced in musicians”, *PloS one*, vol. 5, no. 6, 2010.

-
- [237] S. Kazazis, N. Esterer, P. Depalle, and S. McAdams, “A performance evaluation of the Timbre Toolbox and the MIRtoolbox on calibrated test sounds”, in *International Symposium on Musical Acoustics*, June 2017.
- [238] X. Zheng, Z. Wu, H. Meng, and L. Cai, “Contrastive auto-encoder for phoneme recognition”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 2529–2533.
- [239] A. Radford *et al.*, “Learning transferable visual models from natural language supervision”, in *International Conference on Machine Learning (ICML)*, 2021, pp. 8748–8763.
- [240] B. Elizalde, S. Deshmukh, M. A. Ismail, and H. Wang, “CLAP: Learning audio concepts from natural language supervision”, in *2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [241] B. Dai, Z. Wang, and D. Wipf, “The usual suspects? Reassessing blame for VAE posterior collapse”, in *International Conference on Machine Learning (ICML)*, 2020, pp. 2313–2322.

List of Figures

A	Organization of the chapters in this dissertation.	6
1.1	Acoustic correlates of timbre dimensions	10
1.2	Four widespread commercial synthesizers	12
1.3	Additive synthesis example	13
1.4	Subtractive synthesis example	14
1.5	Simplified Minimoog signals routing	14
1.6	Single-note ADSR amplitude envelope	16
1.7	Tremolo effect using an LFO	17
1.8	Vibrato effect using an LFO	17
1.9	FM synthesis examples	19
1.10	Three DX7 algorithms	21
1.11	Dexed synthesizer plugin	22
1.12	Parameters of a single Dexed FM operator	22
1.13	Sample-based synthesizers	24
1.14	Synthesis engine and UI	25
1.15	Sound synthesizer UI: human action cycle	27
1.16	NSynth auto-encoder model	31

1.17	Vanilla autoencoder model	31
1.18	Spectral autoencoder with output phase reconstruction	34
1.19	Differentiable additive synthesizer model	36
1.20	Differentiable DX7 model	38
2.1	Overview of the sound matching task	46
2.2	Sound matching for a non-differentiable synthesizer	49
2.3	Audio autoencoder model with preset decoding from latent values	53
2.4	Presets gathering and filtering	57
2.5	HPSS decomposition of three audio examples.	59
2.6	Histogram of <i>harmonic</i> , <i>percussive</i> and <i>sfx</i> labels	59
2.7	Histogram of labels based on presets' names	61
2.8	Encoder and decoder distributions for a VAE model.	63
2.9	Visualizations of distributions in a 2D latent space	66
2.10	Example of a flow-based transformation	69
2.11	VAE model with a latent normalizing flow	71
2.12	Computational graph for a forward RealNVP layer	73
2.13	PresetGenVAE block diagram (single-channel spectrograms)	75
2.14	One-hot encoding of synthesis parameters	76
2.15	Precomputations of audio files, spectrograms and presets	79
2.16	Sound matching examples using targets from the test dataset.	84
2.17	Neural representation of a 3D scene using 2D pictures	85
2.18	Multi-channel PresetGenVAE model	86

2.19	Examples using the single- and multi-note models	87
3.1	Cross-fading and morphing	92
3.2	Overview of an analysis-resynthesis system.	93
3.3	Decomposition into partials and a spectral envelope	96
3.4	Four sequences of interpolated artificial 2D lines.	99
3.5	Rank correlations between the 37 retained timbre descriptors .	104
3.6	Distribution of normalized values for the timbre descriptors . .	105
3.7	Morphing linearity and smoothness of a timbre feature	107
3.8	Four timbre features and associated linearity and smoothness .	109
3.9	Reconstruction accuracy of test sounds	116
4.1	Classic sound morphing techniques vs. preset interpolation for a black-box conventional synthesizer	125
4.2	Basic VAE for synthesizer presets.	126
4.3	Transformer-VAE	128
4.4	Transformer encoder architecture for synthesizer presets	130
4.5	Transformer decoder architecture for synthesizer presets	131
4.6	Architecture of the SPINVAE model and training losses \mathcal{L} . . .	134
4.7	DLM with three components	136
4.8	Two preset interpolation methods (linear and SPINVAE) and SMT sound morphing	144
4.9	Reconstruction accuracy of test sounds	146
4.10	Control interfaces for blending multiple presets or latent codes	149

4.11	Interpolation and extrapolation in a 2D VAE latent space . . .	151
4.12	Interpolation and extrapolation example using SPINVAE. . . .	151
4.13	Modulations of a preset using SPINVAE	152
5.1	Interactive web application used for playing a sequence of sounds	157
5.2	Subjective comparisons between paired interpolations	159
5.3	Subjective evaluations of smoothness and naturalness	160
5.4	PCA on the set of <i>smooth</i> and <i>natural</i> subjective ratings	161
5.5	Correlations between objective interpolation quality metrics and subjective ratings	162
5.6	Interpolation linearity and smoothness vs. latent size L_Z	168
5.7	Average self-attention map for decoder layer 5/6.	174
5.8	Average self-attention map for decoder layer 6/6.	174
5.9	Trajectories of parameters during a preset interpolation	176
5.10	Spectral changes related to the FM algorithm (SPINVAE)	177
5.11	Spectral changes related to the FM algorithm (linear interp.) . .	177
B.1	DX7 FM algorithms	185
B.2	Ableton Operator synthesizer	188
D.1	Histograms of Dexed synthesis parameters (Part 1/3)	196
D.2	Histograms of Dexed synthesis parameters (Part 2/3)	197
D.3	Histograms of Dexed synthesis parameters (Part 3/3)	198
E.1	Rank correlations between all available timbre descriptors	200

E.2	Highly correlated descriptors of timbre.	201
F.1	Linearity of seven morphing techniques (part 1/2).	204
F.2	Linearity of seven morphing techniques (part 2/2).	205
F.3	Smoothness of seven morphing techniques (part 1/2).	206
F.4	Smoothness of seven morphing techniques (part 2/2).	207
F.5	Linearity of two preset interpolation techniques compared to the SMT baseline	208
F.6	Smoothness of two preset interpolation techniques compared to the SMT baseline	209
G.1	Trajectories of parameters during a preset interpolation (1/2) .	212
G.2	Trajectories of parameters during a preset interpolation (2/2) .	213
G.3	Trajectories of parameters during a preset interpolation (1/2) .	214
G.4	Trajectories of parameters during a preset interpolation (2/2) .	215

List of Tables

2.1	Sound matching studies based on genetic algorithms	47
2.2	Data-driven sound matching models (non-generative)	52
2.3	Generative sound matching models	53
2.4	Performance of sound matching models for FM synthesis	55
2.5	Comparison of PresetGenVAE decoder models	81
2.6	Comparison of the single-channel and multi-channel models	86
3.1	Linearity statistics for multiple morphing techniques	113
3.2	Smoothness statistics for multiple morphing techniques.	113
3.3	Computational requirements for analysis and resynthesis	117
4.1	Linearity and smoothness metrics for two preset interpolation techniques and SMT sound morphing.	141
4.2	Computational requirements for analysis and resynthesis	148
5.1	Interpolation performance of variants of the SPINVAE model	165
5.2	Interpolation performance for different probability distributions	169
5.3	Influence of data augmentation techniques	171

B.1 DX7 synthesis parameters	186
--	-----