Augmented Granular Synthesis Method for GAN Latent Space with Redundancy Parameter

Koray Tahiroğlu
Department of Art and Media
Aalto University School of Arts, Design and Architecture
FI 00076 AALTO Finland
koray.tahiroglu@aalto.fi

Miranda Kastemaa
Department of Art and Media
Aalto University School of Arts, Design and Architecture
FI 00076 AALTO Finland
miranda.kastemaa@aalto.fi

Abstract

In this paper we introduce an augmented granular sound synthesis method for a GAN latent space exploration in audio domain. We use the AI-terity musical instrument for sound generating events in which the neural network (NN) parameters are optimised and then the features are used as a basis to generate new sounds. The exploration of a latent space is realised by creating a latent space through the original features of the training data set and finding the corresponding audio feature of the vector points in this space. Our proposed sound synthesis method can achieve multiple audio generation and sound synthesising events simultaneously without interrupting the playback grains. To do that we introduce redundancy parameter that schedules additional buffer slots divided from a large buffer slot, allowing multiple latent space vector points to be used in granular synthesis, in GPU real-time. Our implementation demonstrates that augmented buffer schedule slots can be used as a feature for a sound synthesis method to explore GAN-latent sound synthesis of granular-musical events with multiple generated audio samples without interrupting the granular musical features of the synthesis method.

1 Introduction

Generative adversarial networks (GANs) have been extensively investigated to model the probability distributions in the fields of computer graphics (such as image synthesis, image to image translation) [1], audio generation (such as sound synthesis) [2] as well as in the domains of text generation [3]. Its application to a deep learning model makes it possible to transform semantic features of data into a latent space, representing the "embeddings" of the training data. In audio sample generation and sound synthesis domain, a data set is usually prepared from a collection of audio files to create a GAN latent space, which is used to map the high-dimensional encoded representation to the generated audio sample. Since the generated audio samples are based on raw audio feature representations in the latent space, synthesised sounds usually need further exploration for musically interesting outcome.

In the work we present here, we focus on a new development of our AI-terity musical instrument, which comprises a GAN deep learning model and generates audio samples for real-time audio synthesis. With the aim of addressing limitations of previous implementation of the instrument AI-terity, discussed in earlier work [4], we developed and implemented an augmented granular synthesis method to synthesise the generated audio samples. In previous implementations of the instruments as well, we used granular synthesis with the representation of features both in the musical timbres and the temporal signature of the audio samples that are synthesised through GAN latent space. However generating multiple audio samples, synthesising them simultaneously with multiple granular features and without interrupting the playback grains while interpolating between multiple points in the latent space, in GPU real-time, resulted in a detrimental synthesis problem [1]. Different rates at which grains are produced and consumed require more controlled scheduling of the events in granular synthesis. It is because audio synthesis algorithms rely on being given access to the complete stream of audio events. This implies that the granular synthesis algorithm must be run after each sample of audio to ensure the synchronisation with the source. The quality of the output may depend on the granular system's ability to process the input at a rate at which the audio source is presented. This is particularly important in the context of our work in which the input and output of multiple audio events have irregular time schedules. For example, if the input of the system failed to produce audio for a certain time, a clicking-type effect may be heard when the output tries to produce a synthesised sample that is absent in the input. Specifically, not being able to synchronise these granular synthesis system-events may cause overlap or gaps in the output signal. If audio events are not time aligned, they may play out of order and cause undesirable results.

In this paper, we describe our solution that relies on the use of two concurrent scheduling threads in synthesis events, where one thread is handling the audio source (generating the audio samples) and the other thread generates the audio grains with the selected feature set as well as the time intervals they represent. The audio samples and their time intervals are stored in buffers and operate synchronously in parallel. In this way we can keep the generated audio samples in buffers in a continuous manner, which are processed only once, after which they are stored and discarded. The advantage of this solution is the minimal control we need over both scheduling threads and explore multiple points simultaneously in GAN latent space without interrupting the granular musical features of the synthesis method.

2 Related Work

High-dimensional nature of a GAN latent space in audio domain affords a wide range of possibilities for exploring its sound world and interfacing to musically interesting compositions and performances. Exploring the capabilities of musical latent space, not only as a generative model, but rather as a design space for the creation of novel synthesised instruments has been of interest to many scholars [5] [6] [7]. Roberts et. al. [5] built the MusicVAE Sequencer, which is interfaced to the 2-bar melody loops and drum beat that are randomly sampled from the latent space of Music VAR [8]. The MusicVAE Sequencer allows users to explore the possibilities of the latent space through the creation of variations of a 2-bar loop. It generates sequences of note and drum lines that can be arranged in patterns interpolated through MusicVAE's latent space. The latent parameters of the sequencer are mapped onto an input space, which consists of user-defined four corner points and 2-D palette control-surface. The authors suggest that exploring the sequentiality of the generated music though latent space morphing of musical sequences will encourage users to generate variations in musical palette, rather than just blending melodies or drum loops. NSynth [9] instrument provides latent space exploration to discover new timbres that exist between pre-existing instruments. The latent space is based on instrument samples generated by NSynth with timbre characteristics and the note values. Following the four corner points interpolation in 2-D input interface, NSynth instrument allows musicians to explore the latent space of timbres and creating new sounds to be used in any kind of DAW, whether it's with hardware or software synthesisers.

DrumGAN explores latent space in similar manner by interpolating between the points and performs percussive sound synthesis aiming to provide more musically meaningful control over the output [10]. While the main control features appears on the conditioning model of the generative system, DrumGAN generates realistic drum patterns using latent space. Furthermore, it is possible to extend the drum pattern generation capability to different conditions by varying the latent space of the control system. The drum pattern generation mechanism allows high-level control on the timbre characteristics of the sounds. From a slightly different approach and at the same time following the conditional audio synthesis method, Kumar et. al [11] introduces MelGAN, a non-autoregressive feed-forward convolutional architecture that reconstructs raw waveform from latent space and translates music on any domain to a target domain trained in the system. It provides fast and

[1]The following URL link to audio example demonstrates a clicking-type effect that appeared using the first version of the granular synthesis model in instrument AI-terity - `https://tinyurl.com/mwrrhyuv`

efficient computation with decent audio quality. While the model is well-suited to the music-specific task, to our knowledge, MelGAN has not yet been applied to music domain in ways and means of exploring musically interesting opportunities. With the availability of providing high-fidelity waveform as input to the GAN training, MelGAN is a promising model for further development of algorithms and frameworks that can work efficiently for various tasks such as noise reduction, re-creation, and editing.

GANSynth is one of the well know GAN model applied to music generation and similar to other GAN architectures in audio domain, generates the audio clips from a single vector point in latent space [2]. GANSynth uses a progressive GAN architecture to incrementally up-sample with convolution from a single vector to the full sound and allows using instantaneous frequencies to align phases and synthesising high-fidelity audio. The generated audio could be used in synthesising musically interesting and realistic music pieces. Some recent related work demonstrates the features of musical instruments that are interfaced with GANSynth model as standalone music applications or plugins for existing audio frameworks [2].

The GAN models presented in this section demonstrate computationally advanced opportunities to reconstruct audio samples in various forms of music composition. It is expected that a musical composition constructed through a particular sound synthesis technique exhibits musically interesting properties and characteristics from that of a sequence of audio samples [6] [12][13]. In order to achieve such exploration, we propose to add additional constraints on sound synthesis process that may be employed in conjunction with the audio samples generated through the vector points in the latent space and various sound synthesis parameters (e.g., the duration, waveform, envelope, index position, and density of the grains) that are specified to generate musical composition. In particular, our augmented sound synthesis method operates on the microsound time-scale that is divided into grains of milliseconds (ms) where each grain contains a number of microsound components. Although these features have been available in various granular synthesis algorithm, our approach uses "redundancy parameter” to schedule the duration of the sound events.

3 Augmented Sound Synthesis

To explore our multidimensional latent space in real time, we use the sounds synthesised by GANSpaceSynth[3] as input for granular synthesis [14]. We implemented the GANSpaceSynth, a novel hybrid GAN architecture, to organise GAN latent space using a dimensionality reduction technique, which gives an opportunity to specify particular audio features to be present or absent in the generated audio samples. For the granular synthesis part, we chose the mill granular synthesiser external object for Pure Data (Pd) written by Olli Erik Keskinen[4]. We take advantage of GANSpaceSynth's batch synthesis to synthesise multiple sounds in one go. Using the default `batch_size` of 8, we form a small sphere around the current position in latent space and spread 8 points evenly on the sphere's surface, resulting in 8 distinct latent vectors. The idea is that we synthesise multiple small variations of the sound at the latent space position. GANSpaceSynth thus produces a batch of 8 audio samples, which are played by 8 corresponding instances of the mill granular synthesiser.

In initial experiments with the mill external, we heard harsh clicking artifacts when playing while moving in the latent space. In the instrument AI-terity, GAN synthesis and grain playback are interleaved, and in order to make the instrument responsive we synthesise with GANSpaceSynth as frequently as the GPU hardware allows [5]. With our setup, we are able to synthesise about once every 60 ms. Meanwhile, the grains played by mill range in length from 0 ms to 600 ms, depending on sensor input. This meant that the contents of mill's audio buffer were often updated in the middle of playback, as shown in Figure 1, causing discontinuities in the waveform which manifested as the clicking artifacts.

This is a readers-writers problem [15], a class of concurrency problems well studied in computer science. In this case, we have a single writer (GANSpaceSynth) and multiple readers (mill grains).

[2]`https://magenta.tensorflow.org/studio`

[3]`https://github.com/SopiMlab/GANSpaceSynth`

[4]`https://github.com/ollierik/`

[5]For the current development of the sound synthesis, we built a mini-PC with an external GPU. We used an Intel Core i7-10170U CPU @ 1.1GHz and NVIDIA GTX 1080 Ti GPU connected via Thunderbolt 3, in which we used Ubuntu Linux 20.04.1 LTS.

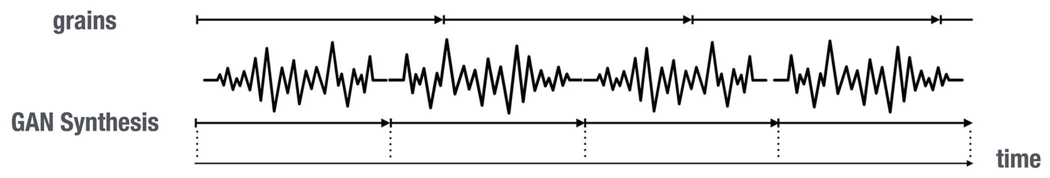

Figure 1: The original audio buffer structure in mill was interrupting the playback grains.

Common solutions to such problems include mutexes [15] and readers-writer locks [16], however our real-time audio context poses some challenges for lock-based solutions. If readers lock the audio buffer, the writer may never get a chance to update the buffer, because grains can play continually and even overlap with each other. On the other hand, if the writer locks the buffer, readers are prevented from reading it for the duration of the lock, meaning the playback of grains would have to be interrupted. Furthermore, GANSpaceSynth and mill are written in different programming languages (Python and C, respectively), making it difficult to apply tools from existing concurrency libraries. Aiming for simplicity of implementation, we looked for a synchronisation mechanism relying only on Pd messages, not requiring any deeper integration between the two components.

Our solution is to use more buffers, so that GANSpaceSynth can always write synthesised audio into a buffer that is not being played back. To do that, we introduce a redundancy parameter as a new implementation to our GANSpaceSynth Pd object. Redundancy parameter is a nonnegative integer that specifies how many additional buffers will be used. The number of buffers to allocate becomes $\texttt{batch\_size} \cdot (1 + \texttt{redundancy})$. In practice, we use a single large buffer divided into this number of slots. For example, given a redundancy of 3 and GANSpaceSynth defaults for audio length (64000 samples) and batch size (8), we allocate a buffer of $64000 \cdot 8 \cdot (1 + 3) = 2048000$ samples. Slot 0 then consists of samples 0–63999, slot 1 of samples 64000–127999, and so on.

GANSpaceSynth maintains a list of free slot numbers into which audio data can safely be written. Figure 2 demonstrates the buffer slot structure in GANSpaceSynth. Initially, all slots are free. When GANSpaceSynth is about to start synthesizing, it checks to see whether `batch_size` free slots are available. If not, the synthesis is skipped and retried later. (Ideally, we want to prevent this from happening.) If free slots are found, synthesis begins. Once done, the synthesized audio is written into the free slots and they are marked as reserved. For each of these slots, a message is sent to the corresponding mill instance notifying it to start playing any new grains from the slot. Our modified version of mill keeps track of the slot that each grain is playing. Any grains that are playing when the new slot message is received continue to play their assigned slots to the end. Once the number of playing grains for a previous slot reaches zero, a message is sent to notify GANSpaceSynth that the slot can be marked as free again.

Given a minimum grain interval, a maximum grain duration and a maximum delay in message delivery and processing, it is possible to calculate a redundancy value sufficient to prevent skipped synthesis batches from happening. In practice, we experiment and settle on 31 for our setup. Our changes to mill introduce a race condition that occasionally miscounts slot grain counts when running with multiple threads, causing slots to not be freed properly and effectively "leaking' them. However, we find that single-threaded performance is more than sufficient, and thus sidestep the issue by disabling multi-threading.

4 Micro-Sound time-scale parameters

We have multiple independent mill instances playing grains from their own buffer slots. Each bend-sensor on the instrument AI-terity control surface produces a linear value between 0 and 1 depending on the amount of bending, and we map each of these values to a set of granular synthesis parameters for the corresponding mill instance. Several of our mappings are non-linear, and for these we use exponential curve-ranges, where a curve of 0 is linear, >0 is positively curved and <0 is negatively curved. Grain density ranges from 0.08 to 0.02 with a curve of 0.5 (lower density values produce more frequent grains) and an added random variation of 0.6. Grain duration ranges from 0 seconds to

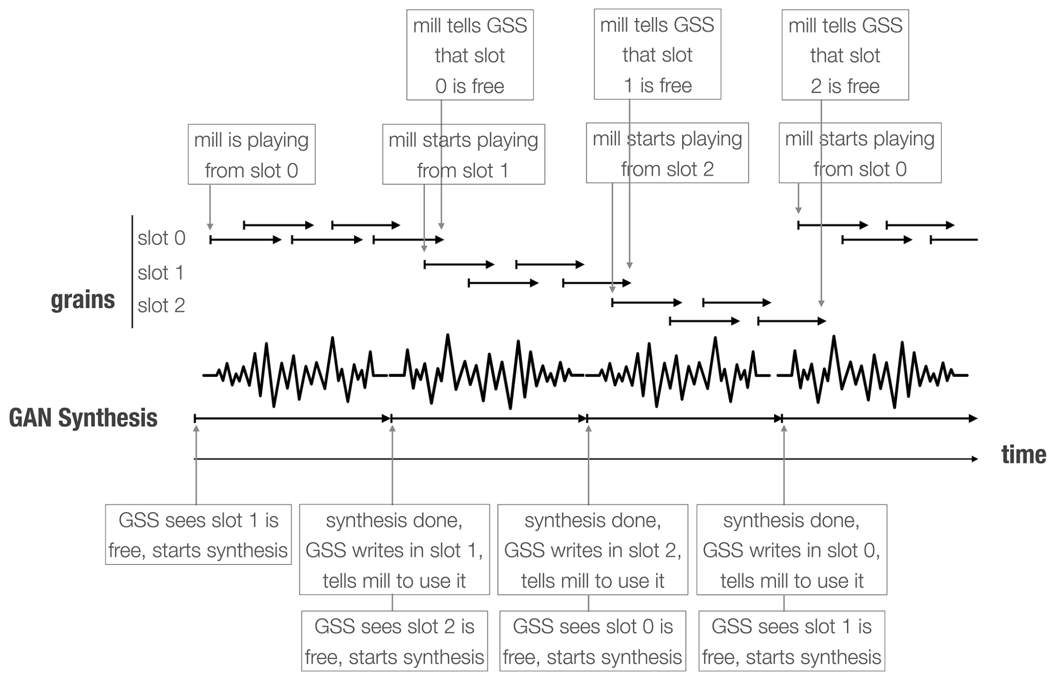
mill tells GSS that slot 0 is free
mill tells GSS that slot 1 is free
mill tells GSS that slot 2 is free
mill is playing from slot 0
mill starts playing from slot 1
mill starts playing from slot 2
mill starts playing from slot 0
slot 0
slot 1
slot 2
grains
GAN Synthesis
time
GSS sees slot 1 is free, starts synthesis
synthesis done, GSS writes in slot 1, tells mill to use it
synthesis done, GSS writes in slot 2, tells mill to use it
synthesis done, GSS writes in slot 0, tells mill to use it
GSS sees slot 2 is free, starts synthesis
GSS sees slot 0 is free, starts synthesis
GSS sees slot 1 is free, starts synthesis

Figure 2: New buffer-slot structure in AI-terity with GANSpaceSynth (GSS).

0.6 seconds with a curve of -0.1 and a random variation of 0.1. Position of the grains ranges from 0 (start of the buffer slot) to 0.6 (60%, or 2.4 seconds into a 4-second buffer slot) with a curve of 3 and a random variation correspondingly varying linearly from 0.3 to 0.01. Velocity (mill's term for grain playback rate) does not vary with the bend value, but rather each sensor has its own constant velocity tuned to a double harmonic scale (intervals H-3H-H-W-H-3H-H). The root pitch is not specified and instead depends on the tonality (if any) of the audio synthesised by GANSpaceSynth. Finally, panning varies randomly from 100% left to 100% right to create a wide stereo cloud of grains.

5 Reordering Buffers in Sound Synthesis

Our solution to apply redundancy parameter for reordering buffers and adding additional buffers to avoid playback interruption in sound synthesis, allocates memory for an audio sample just for temporary storage. If we're sure that the buffer doesn't overlap with the sound we are synthesising, we can use that buffer slot. Otherwise, we're using dynamic memory, and we need to free the buffer slot later. To apply the redundancy parameter for reordering buffers, we are creating temporary buffers that we use just to keep the same buffers from rewriting. Then we are looping through the buffer indices until we find the buffer slots that we want to change its status for rewriting. At the beginning of the function we are assigning the buffer to a specific index of slot 0. Finally, we change the positions of buffers and check for deadlocks.

Some other methods for reordering of the contents of a sample buffer in order to lock an audio-buffer playback into a fixed loop is used. For example, the order of playback of samples may be rearranged in order to recreate a sequence of samples played, such that they are played in the order of a time sequence, thereby locking the playing of the sample buffer [17]. The above solution, however, has the similar problem we faced earlier for simultaneously generated multiple audio samples that the playing back cannot be started instantly when the reordering of the sample-buffer starts. As another method in streaming audio packets in a network for locking the sample-buffer playback, the order of playback of samples may be rearranged so as to play a loop in delayed buffer, thereby locking the playing of the sample buffer [18]. The delayed buffer option provides an alternative method by delaying the playback in a number of audio buffers. According to this method, however, when the playing of the sample buffer is unlocked, the loop continues in the past state, so that the playing of a number of sample buffers can not be started instantly. Further, according to the authors, when the sample buffer is played in delayed, the playing order of the buffer slots is not necessarily kept in the order of their chronological occurrence, which is not desirable for our sound synthesis method.

Bascou and Pottier [19] presented a need of a flexible synthesis environment with a precise control of granular representation of sounds. With a view to provide sound synthesis to support generation of high density complex grains, the authors proposed the "GMU" system for the creation of accumulated sonic grains, each with their temporal and spectral variabilities. The GMU was realised to control sounds synthesis with “buffer based enhanced grain generator”. The buffers represent one of the most common frameworks for granular synthesis implementations, it relies on storing variety types of waveforms and allows generating sounds with transformations on temporal and spatial features of voices is sound sources.

In the case of the GMU, a set of pre-calculation of envelopes in buffers is presented with their temporal positions and the corresponding spectral voice profiles. Each of these buffers can be read at synthesis time modified by the user at will to either add a spectral tail or pitchshift the voice on the time axis by a given amount. Here the authors propose the GMU system as an efficient framework that combines the flexibility of the buffer structure in granular synthesis approach with the ability to generate complex sound through a "continuous" sound synthesis.

GMU provides a flexible granular synthesis environment with precise control of individual grains and shares with our modified mill the property that each grain carries its own reference to an audio buffer. However, GMU does not appear to address the problem of synchronising grain playback with continual rewriting of buffer contents. The main difference of the ways buffered used in GMU and in our AI-terity musical instrument is the irregular timing conditions set through jump events in GANSpaceSynth for writing the audio signals into buffers and synthesising them. This enables us to use the buffer slots and move between the slots efficiently. It also makes it possible to apply audio processing to the buffer slots (reading index positions, and changing the density of the grains, etc.) without having any timing conflicts or the gap of audible artefacts between buffer slots. Figure 3 shows the video demonstration of the synthesised sounds using the augmented buffer slots in mill granular synthesiser with the timing conditions. This buffer slot framework specifically allows GANSpaceSynth to generate multiple audio samples, synthesise them and let musicians to move along the orthogonal directions set in the latent space. The granular synthesis parameters (interval, duration, delay) and their relation to the timing parameters chosen for the additional buffers that are required, have a particular influence on the musical outcome with GANSpaceSynth. Furthermore, these features set the idiomatic patterns of its own algorithm for the musicians [20].

6 Conclusions

In this paper we presented an augmented granular sound synthesis method, introducing redundancy parameter for synchronising grain playback with rewriting of buffer contents in granular synthesis. The key feature in the proposed sound synthesis method is in the way it is used for simultaneously generating multiple audio samples and synthesising them without interrupting the playback grains. We have shown that augmenting the mill granular sound synthesis with redundancy parameter in playback events makes the sound synthesis more robust to irregular time signatures that occur between the events of writing generated audio samples to buffer slots and synthesising of granular-musical events. The synchronised playback with continual writing of the buffer can effectively be beneficial for GAN latent space exploration of granular sounds in realtime. The granular sound synthesis methods based on the proposed time-varying buffer slots has demonstrated more efficient sound synthesising compared with methods based on the constant buffer, which enables one to control the multiple sounds playback freely.

The method that is described in this paper is a significant programming/integration effort to address a particular synthesis problem that emerges from the need of our system-structure. One may argue that such a problem will not exist if both heterogeneous systems (GANSpaceSynth - Python and Mills - C compiled for Pure Data) would be patched together and implemented in the same language or perhaps if the GANSpaceSynth was modified to have a production rate matching the consumer one. However, such a solution for customised audio synthesis method with multi-threading buffering strategy is needed, considering the autonomous nature of the AI-terity instrument and its effect on the audio generation/synthesis events. We believe that a system in which one component is compiled in C and the other one in Python would still be desirable, because this allows integrating and optimising

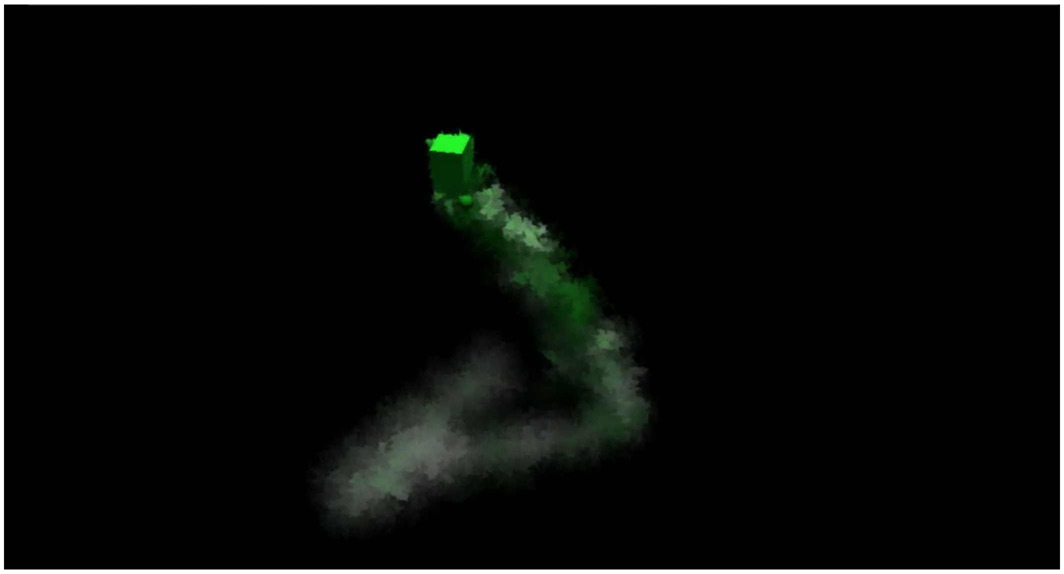

Figure 3: The screen-shoot of the video demonstrates the GAN latent space exploration with the AI-terity instrument, generating mutliple audio samples, synthesising them simultaneously and synchronising grain playback in mill granular synthesis. The video material is available at `https://vimeo.com/558363454`

real-time audio synthesis environment together with various deep learning models, at very little effort and does not introduce conflicts or incompatibilities. At the same time, our goal is to develop a GAN space synthesis engine using Python environment to allow further integration with available sound synthesis tools and to make our solution extensible and modular to be used in other existing synthesis systems. We believe that such a solution would not require the development of new components for a given system to exploit the advantages of the deep leaning models applied to audio synthesis in music context. Our system-structure also implies that the deep learning model synthesis code is not directly used by the musician in many cases, but is exposed to the musician in the form of a "dialog box" as in the form of a GUI function in Pure Data environment that returns input/output synthesis comments. In this way, the musician has to find the best way to build up her own communication with the deep leaning modules, without any need for further "magic" for creating audio synthesis techniques using deep learning algorithms.

The benefits introduced by this augmented sound synthesis method is based on an implementation of redundancy parameter in scheduling additional buffer slots to granular synthesis. This additional buffer slots / large buffer slot combination is then further combined with GANSpaceSynth to allow multiple latent space vector points to be used in granular synthesis, simultaneously, in GPU real-time without any audio artefacts. The result is a system that provides the musician features to explore GAN latent sound synthesis of granular-musical events with expanded range of timbre in multiple generated audio samples without interrupting the granular musical features. These benefits in synthesis method and results have allowed further developments in granular-musical synthesis that may provide a new approach to music creation and composition in granular-musical domain. The instrument AI-terity has been developed further to allow this approach to be further explored in music compositions [21].

There are several open issues in the presented augmented sound synthesis method. An interesting challenge is to improve the quality of generated samples for each given set of input parameters, and this can be an interesting direction to improve on GANSpaceSynth. At the moment, GANSpaceSynth generates 4 seconds audio samples with 16000Hz sampling rate. Furthermore, an alternative approach is needed to explore all kinds of generative deep learning models similar to GANs. For instance, an interesting research direction could be to investigate if a generative model that allows for more control, less computational resources and latent space of audio features with decent audio quality, could become a replacement for the existing generative models applied with GAN architecture. An additional direction for future work would be to investigate the potential of augmenting or reformulating the GAN structure to include more sustainable AI technologies. Finally, we recommend looking into other sound synthesis methods (e.g. wave model generation, generative vocoder, etc.) to compare with our proposed augmented sound synthesis method for a GAN latent space exploration.

Acknowledgement

This work was supported by the Academy of Finland (project 316549) and A!OLE funding. We acknowledge Pierre Chouteau for his work in our project.

References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[2] Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., & Roberts, A. (2019). GANSynth: Adversarial neural audio synthesis. *In Proceedings of the International Conference on Learning Representations*

[3] Fedus, W., Goodfellow, I., & Dai, A. M. (2018). Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736.*

[4] Tahiroğlu, K., Kastemaa, M., & Koli, O. (2020). Al-terity: Non-rigid musical instrument with artificial intelligence applied to real-time audio synthesis *In Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 337–342). Birmingham, UK: Birmingham City University

[5] Roberts, A., Engel, J., Oore, S., & Eck, D. (Eds.). (2018). Learning Latent Representations of Music to Generate Interactive Musical Palettes. *Proceedings of the 2018 ACM Workshop on Intelligent Music Interfaces for Listening and Creation, MILCIUI 2018.*

[6] Tahiroğlu, K., Kastemaa, M., & Koli, O. (2021). AI-terity 2.0: An Autonomous NIME Featuring GANSpaceSynth Deep Learning Model. *In Proceedings of the International Conference on New Interfaces for Musical Expression 2021.* https://doi.org/10.21428/92fbeb44.3d0e9e12

[7] Bryan-Kinns, N., Banar, B., Ford, C., Reed, C. N., Zhang, Y., Colton, S., & Armitage, J. (2021). Exploring XAI for the Arts: Explaining Latent Space in Generative Music. *In eXplainable AI approaches for debugging and diagnosis. Retrieved from https://openreview.net/forum?id=GLhY_0xMLZr*

[8] Roberts, A., Engel, J., & Eck, D. (Eds.). (2017). Hierarchical Variational Autoencoders for Music. *Workshop on Machine Learning for Creativity and Design, NIPS.*

[9] Norouzi, M., Eck, D., & Simonyan, K. (2017). Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In D. Precup & Y. W. Teh (Eds.), *In Proceedings of the 34th International Conference on Machine Learning (Vol. 70, pp. 1068–1077).* PMLR. Retrieved from https://proceedings.mlr.press/v70/engel17a.html

[10] Nistal, J., Lattner, S., & Richard, G. (2020). DrumGAN: Synthesis of drum sounds with timbral feature conditioning using Generative Adversarial Networks. *arXiv preprint arXiv:2008.12073.*

[11] Kumar, K., Kumar, R., de Boissiere, T., Gestin, L., Teoh, W. Z., Sotelo, J., ... & Courville, A. (2019). Melgan: Generative adversarial networks for conditional waveform synthesis. *arXiv preprint arXiv:1910.06711.*

[12] Mouse on Mars (2021). *AAI.* Thrill Jockey.

[13] Iqbal, N. & Heaney, L. (2021). The Whole Earth Chanting. AI and Music S+T+ARTS Festival, Sonar.

[14] Tahiroglu, K., Kastemaa, M., & Koli, O. (2021, July). GANSpaceSynth: A Hybrid Generative Adversarial Network Architecture for Organising the Latent Space using a Dimensionality Reduction for Real-Time Audio Synthesis. *In Proceedings of the 2nd Joint Conference on AI Music Creativity.*

[15] Courtois, P. J., Heymans, F., & Parnas, D. L. (1971). Concurrent Control with “Readers” and “Writers.” *Commun. ACM, 14*(10), 667–668. https://doi.org/10.1145/362759.362813

[16] Krieger, O., Stumm, M., Unrau, R., & Hanna, J. (1993). A fair fast scalable reader-writer lock. *In 1993 International Conference on Parallel Processing-ICPP’93 (Vol. 2, pp. 201–204).*

[17] Lyon, E., Knapp, R. B., & Ouzounian, G. (2014). Compositional and performance mapping in computer chamber music: A case study. *Computer Music Journal, 38*(3), 64-75

[18] A. Carôt, C. Hoene, H. Busse & C. Kuhr, "Results of the Fast-Music Project—Five Contributions to the Domain of Distributed Music," *in IEEE Access, vol. 8, pp. 47925-47951, 2020,* doi: 10.1109/ACCESS.2020.2979362.

[19] Bascou, C., & Pottier, L. (2005, November). Gmu, a flexible granular synthesis environment in max/msp. In Proceedings of the Sound and Music Computing Conference (SMC'05), Salerno, Italy.

[21] McPherson, A., & Tahıroğlu, K. (2020). Idiomatic Patterns and Aesthetic Influence in Computer Music Languages. *Organised Sound, 25*(1), 53-63. doi:10.1017/S1355771819000463

[21] Tahiroğlu, K. (2021). Uncertainty Etude #2. In NIME 2021. https://doi.org/10.21428/92fbeb44.15ad0c07