SnakeSynth: New Interactions for Generative Audio Synthesis

Eric Easthope
University of British Columbia
Vancouver, British Columbia, Canada
eric@sequent.audio

ABSTRACT

I present *SnakeSynth*, a web-based lightweight audio synthesizer that combines audio generated by a deep generative model and real-time continuous two-dimensional (2D) input to create and control variable-length generative sounds through 2D interaction gestures. Interaction gestures are touch and mobile-compatible and made with analogies to strummed, bowed, brushed, and plucked musical instrument controls. Point-and-click and drag-and-drop gestures directly control audio playback length and intensity. I show that I can modulate sound length and intensity by interacting with a programmable 2D grid and leveraging the speed and ubiquity of web browser-based audio and hardware acceleration to generate time-varying high-fidelity sounds with real-time interactivity. SnakeSynth adaptively reproduces and interpolates between sounds encountered during model training, notably without long training times, and I briefly discuss possible futures for deep generative models as an interactive paradigm for musical expression.

Author Keywords

audio synthesis, generative adversarial network, interaction, gesture, musical expression, controller, 2D

CCS Concepts

•**Applied computing** → **Sound and music computing;** Performing arts; •**Computing methodologies** → **Neural networks;** •**Human-centered computing** → *Interaction techniques; Interaction paradigms;*

1. INTRODUCTION

Interaction paradigms for deep generative models (DGMs) have remained relatively shallow in contrast to the diversity of interactions that are possible with musical interfaces and most research interest in DGMs still seems to revolve around generation of fixed-size images and audio in correspondence to fixed-size training data [12, 13]. These models often work by learning a low-dimensional set of inputs that resemble the statistics of a training dataset enabling us to generate new samples through a small number of controls significantly smaller than the size of training data. In experimental contexts DGMs seem to be capable of generating novel outputs and controllably interpolating between training data features [6]. Recent developments including the success of WaveNet [10] and GANsynth [2] have revealed possibilities for how DGMs might be developed to be more expressive in terms of their outputs and this has consolidated some interest in using DGMs as tools for musical expression. Possibly the largest unified effort to do this might be Magenta (`https://research.google/teams/brain/magenta/`) at Google Research which leverages DGMs as part of a larger effort to create music synthetically using machine learning (ML) models.

Yet many projects featured for DGM-based music production and performance still suffer from common structural limitations in DGMs and how they function. Auto-regressive models [5] like WaveNet [10] inherently rely on sequential and often somewhat random updates to inputs to produce appreciable changes in outputs. In performance contexts changes in sound in response to new inputs then need to computed in real time or otherwise delayed. Setting aside the challenges of computing DGM outputs in real-time this breaks down an essential auditory feedback loop between a performer and their instrument(s). Responses on the part of the performer in response to an auto-regressive DGM then have to be anticipated as inputs sequentially and must randomly evolve towards more refined outputs.

This runs counter to how we think about musical instruments and related interfaces. While we would not expect a plucked string to resonate the same way every time the same note is played, we do expect to hear the same note and for it to resonate when we play it. Particularly there is some expectation in performance being informed by anticipation about where and how to sound the instrument in a one-to-one way. This one-to-one-ness also ensures that instruments play the same way today as they do tomorrow. Perhaps regressive DGMs and randomness alone cannot produce usable and less so re-usable ML-based digital music tools.

Luckily not all deep generative models are regressive and some are capable of producing inputs and outputs in a one-to-one way. Generative Adversarial Networks (GANs) [4] and variational auto-encoders [9] amongst other DGMs require only a single forward pass ("single-pass") from input to output making them better candidates for musical interfaces by enabling performers to learn relationships between how they play digital instruments and what will be sounded when they play them. Technically speaking `(input, output)` pairs can be established and anticipated during performance. To this end the development of audio-based GANs like those by Donahue et al. [1] and Engel

Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). Copyright remains with the author(s).

NIME'23, 31 May–2 June, 2023, Mexico City, Mexico.

et al. [2] have shown particular potential to generate novel sounds and musical forms.

In broader performance contexts DGM-based instruments should also exhibit compatible playing dynamics with respect to player expectation. The application of more energy to the instrument, for example by "strumming" or "bowing" with greater intensity, should correspondingly produce more albeit possibly cacophonous sound. Moving or scanning to selectively "pluck" strings should not produce unwanted sound. Continuous "bowed" sounds should correspond to continuous movements, particularly mechanical "driving" and resonance. Reversing the direction of movement should *reverse* the sound in some way; on a string this might correspond to differences in "down-picked" and "up-picked" sounds. These are difficult to express with DGMs and even neural networks broadly speaking due to the fixed length of their outputs and so there is an opportunity here for new designs. The key problem is making DGMs expressive in ways beyond their capacity to yield different outputs.

Part of this is a matter of producing and controlling continuous variable-length sounds with DGMs. Discrete trigger-based controls for musical DGMs resembling MIDI inputs are common but offer little to no control over the length of output audio. This puts the burden of controlling audio length on the underlying DGM(s) yet previous work has done little to address the generation of variable-length audio with DGMs despite the apparent utility of producing variable-length sounds in music contexts. We can concatenate sounds to produce longer streams of audio but results are often cacophonous (see algorithmic music from Dadabots, `https://dadabots.com`). Thinking in terms of a 2D image-based DGM architecture this is equivalent to generating variable-width images by joining images sequentially and is a temporary fix at best, making the problem of generating variable-length audio with DGMs an interesting gap in current work and a means to explore DGM expressivity in creative settings as a performance tool.

SnakeSynth (Figure 1) is an ML and web-based music performance tool and interactive controller that bridges the gap from discrete trigger-based DGM controls to continuous variable-length controls to enable new forms of musical expression and performance dynamics with DGMs. Deriving its name and interactive paradigm from the "Snake" video game genre, SnakeSynth uses real-time 2D point-and-click and drag-and-drop gestures to directly control DGM audio playback length to generate variable-length audio in creative contexts. Interactions with a programmable 2D coordinate grid determine audio length relieving DGMs of un-needed and even extraneous design constraints and giving more control to performers as the primary creative agent. By foregoing concatenation-based approaches and modelling the variability of audio length as an external interactive control over what is otherwise a fixed-length DGM, SnakeSynth enables "strummed," "bowed," "brushed," and "plucked" playing gestures by triggering different fixed-length DGM-generated sounds and blending them through interaction to form longer variable-length sounds.

2. DESIGN

2.1 Model

2.1.1 Generative Adversarial Network

I set up a GAN made of two networks, a *generator* and a *discriminator*, configured as adversaries such that the generator network learns to generate "fake" but convincingly real outputs that "fool" classifications by the discriminator. As they train on new samples the generator improves its weights by back-propagating its losses to produce more "realistic" outputs that resemble the statistics of training data. In turn the discriminator improves its weights to discern fake samples from dataset samples. Any generator losses are theoretical gains to the discriminator and vice versa so both networks improve with training.

Particularly I use a modified Deep Convolutional GAN (DCGAN) architecture but I reduce the DCGAN generator [11] to only three convolutional layers and remove the batch normalization layer following the fully-connected (dense) layer. DCGANs have the advantage of using local convolutional layers in place of exclusively dense layers [4] significantly reducing the total number of trainable parameters and with it reducing total training time. Together the generator consists of a dense layer and three filter layers each containing a convolutional and batch normalization layer activated with a leaky rectified linear unit (leaky ReLU) function. Batch normalization layers regularize training samples to increase training stability [11, 7] and the final convolutional layer is activated with a tanh function amounting to a generator with roughly one million trainable parameters (approximately 280 parameters per pixel).

I also use a Convolutional Neural Network (CNN)-based discriminator but only two convolutional layers, no dropout layers, and no batch normalization. Again this significantly reduces the total number of trainable parameters leaving two convolutional layers activated with leaky ReLU functions and a dense layer with no activation function. Without activation the dense layer outputs values outside of $[-1, 1]$ and the discriminator classifies directly from dense layer *logits*.

Unlike the DCGAN authors I do not change network initialization weights before training and I use *reshape* and *flatten* layers to transform square images to and from layers expecting flat inputs. Both networks are summarized in Figure 2 for a 2D latent space to correspond with a 2D cursor or touch-based input control space.

2.1.2 Dataset: 2D Spectral Images

This DCGAN is trained on square 64x64 pixel images made from Mel-scaled spectral coefficients for a small collection of human voice samples, only one of many possible training sets, in line with observations by Engel et al. [2] that image-based GANs are capable of producing high-fidelity audio from limited spectral information and are significantly faster to train. Mel-scaled coefficients are also an effective compression of spectral information in 2D that simultaneously accounts for human audio perception which I use to significantly reduce overall GAN model dimensionality and correspondingly the memory requirements for training compared to 1D time series-based models like WaveGAN [1].

Sounds in SnakeSynth are made by inverting generated 2D spectral images to 1D time series through Griffin-Lim inversion. Each sound is windowed using a cosine window or similar to remove edge audio artifacts and as I choose the number of sounds in the SnakeSynth interaction grid this inversion is automated as part of post-processing. Faster inversions could be realized in real-time interactive settings.

2.1.3 GAN Training

Generator outputs are initialized as Gaussian noise and I train the DCGAN generator and discriminator in lockstep: the generator first to produce new outputs and the discrim-

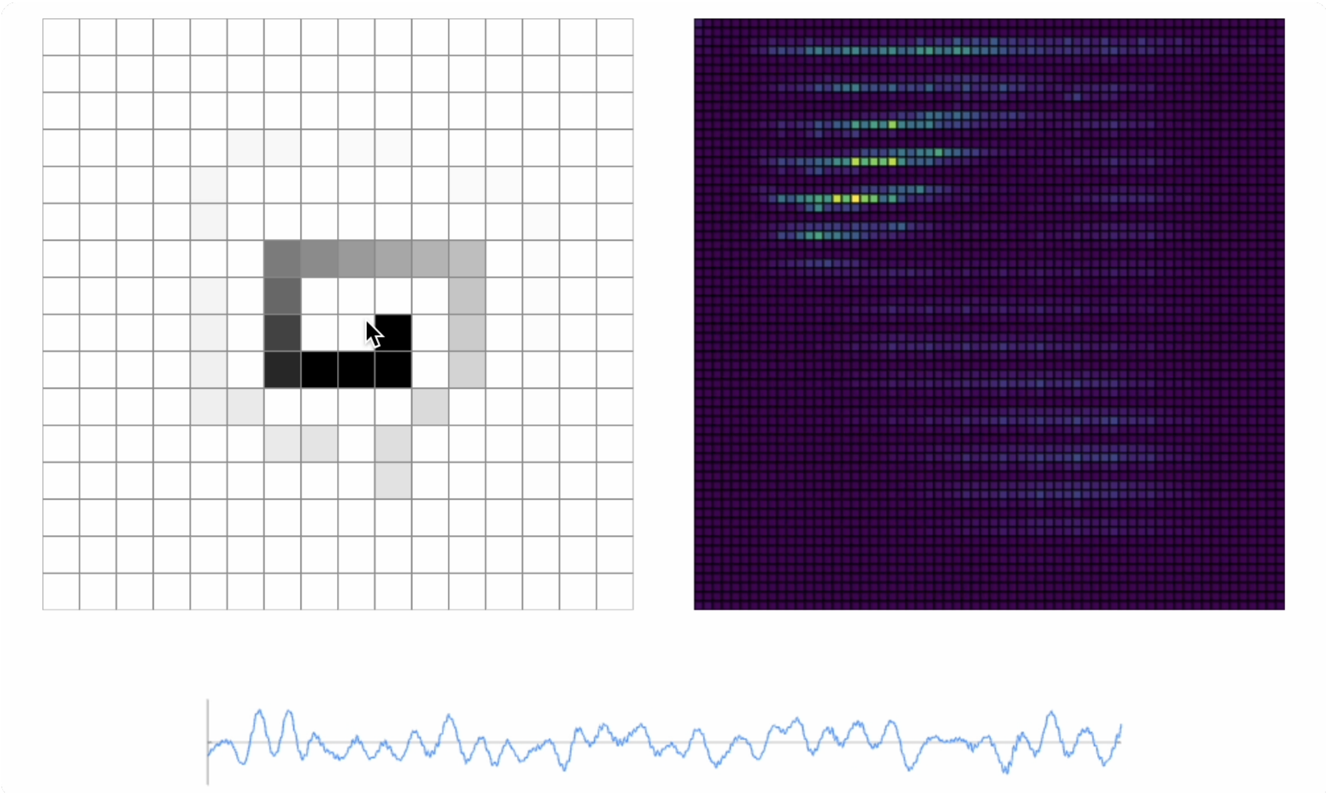

Figure 1: The cursor-based *SnakeSynth* web interface. Interactions with the grid (left) are also touch-compatible supporting tap-to-click and touch-and-drag gestures in correspondence to point-and-click and drag-and-drop cursor gestures. Mel spectra (right) and time series (bottom) for individual sound samples are displayed and updated in real time during playback.

inator second to classify generator outputs against "real" samples of training data. This training strategy is equivalent to a zero-sum competitive game between two players (or networks in this case) where generator losses amount to discriminator gains and vice versa. Goodfellow et al. [4] represents this with the objective function

$$L(G, D) = \mathbb{E}_{x\sim X(x)}[\ln D(x)] + \mathbb{E}_{z\sim Z(z)}[\ln(1 - D(G(z))]$$

for a generator G and discriminator D with the expectation value $\mathbb{E}_{x\sim X(x)}$ that real samples (x) are from the training data distribution X and expectation value $\mathbb{E}_{z\sim Z(z)}$ that fake samples (z) are from a random Gaussian (normal) distribution Z. The generator and discriminator minimize and maximize the objective function respectively to incur a training "loss" that we back-propagate to update their weights.

Training data is shuffled and separated into batch sizes of one so that every image is seen during training and I train for 300 epochs using the same objective function defined by Goodfellow et al. [4]. Increasing the batch size reduces training time and I find that a small 64x64 pixel DCGAN model with a 2D generator latent space trains hundreds of epochs within minutes on a standard MacBook Pro (M1, 2020). Re-training on new samples takes only minutes longer and trained models and sounds can be stored and loaded for later use without further training.

Because the generator inputs are 2D we can directly access and visualize the space of possible generator outputs by passing 2D quantile values as coordinate inputs to the generator latent space. This enables us to produce samples from most of the generator output distribution using only a 2D and particularly *finite* grid-based interactive controller. To do this we compute quantile values up to 95th percentile outcomes from the inverse cumulative distribution function for a 2D Gaussian distribution with zero mean ($\mu = 0$) and unit variance ($\sigma^2 = 1$). Images generated from these quantile values can then be plotted to produce a visualization of nearly all possible generator outputs from a restriction of the entire generator output distribution to arbitrary precision (Figure 3, right) helping us to consider controller designs for underlying generator statistics.

2.2 Interactions

SnakeSynth affords a number of different interaction types naturally through interactions with a two-dimensional $N \times N$ coordinate grid (Figure 3, left):

1. Click (or touch) gestures produce fixed-length audio (resembles "plucking").
2. Linear or near-linear gestures produce variable-length audio (resembles "strumming") (Figure 5). Gesture distance determines sound length.
3. Suddenly changing movement direction creates sudden audio changes and corresponding audio attack (resembles a "finite bow") (Figure 6).
4. Continuous gestures create continuous audio (resembles an "infinite bow") (Figure 7). Particularly circular or near-circular gestures produce continuous *rhythmic* audio.
5. Chaotic gestures with many directional changes to linear and/or circular movements create cacophonous audio (resembles "brushing") (Figure 8).

Interactions 2-5 are shown in Appendix A.

```
Model: "Generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 65536)             131072

 leaky_re_lu (LeakyReLU)     (None, 65536)             0

 reshape (Reshape)           (None, 16, 16, 256)       0

 conv2d_transpose (Conv2DTr  (None, 16, 16, 128)       819200
 anspose)

 batch_normalization (Batch  (None, 16, 16, 128)       512
 Normalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 conv2d_transpose_1 (Conv2D  (None, 32, 32, 64)        204800
 Transpose)

 batch_normalization_1 (Bat  (None, 32, 32, 64)        256
 chNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 64)        0

 conv2d_transpose_2 (Conv2D  (None, 64, 64, 1)         1600
 Transpose)

 activation (Activation)     (None, 64, 64, 1)         0

=================================================================
Total params: 1157440 (4.42 MB)
Trainable params: 1157056 (4.41 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

```
Model: "Discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        1664

 leaky_re_lu (LeakyReLU)     (None, 32, 32, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       204928

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 1)                 32769

=================================================================
Total params: 239361 (935.00 KB)
Trainable params: 239361 (935.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2: A simplified Deep Convolutional GAN (DCGAN) [11] network architecture used to generate SnakeSynth sounds. The generator inputs two values and outputs 64x64 pixel images activated with a $\tanh$ function layer. The discriminator inputs these 64x64 pixel images and outputs one value to classify "real" dataset samples versus "fake" generator samples. Without an activation function the discriminator outputs *logit* values outside of $[-1, 1]$.

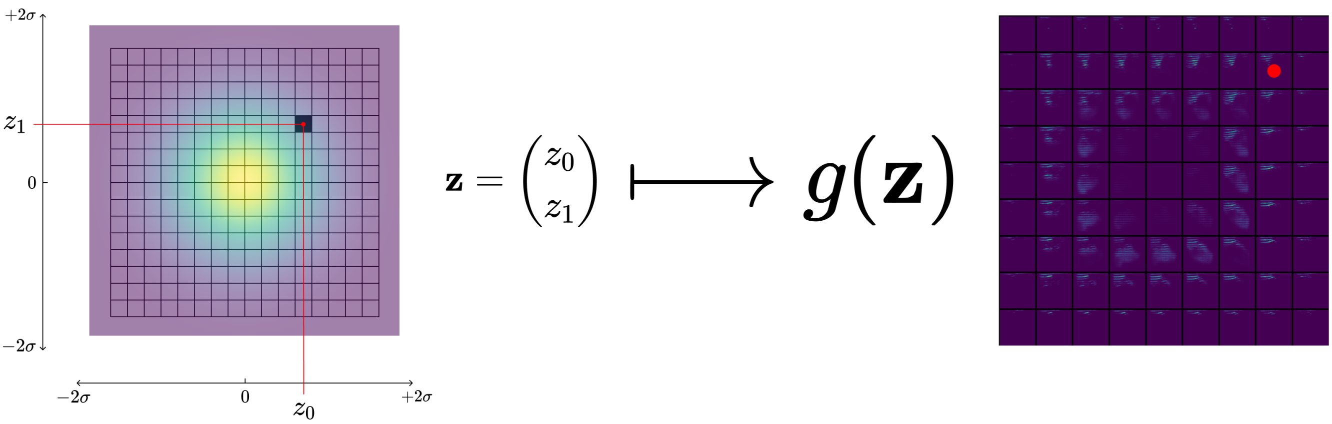

Figure 3: SnakeSynth generates spectral images from maps of quantile values sampled (in red) from two standard deviations of a Gaussian (normal) distribution in an arbitrary 2D coordinate grid. Grid interactions sample outputs from a restriction of the entire possible generator output distribution up to 95th percentile outcomes.

2.3 Synthesis

Instead of directly concatenating audio clips I trigger equal-length clips asynchronously and sum them over time to produce variable-length audio in response to interactions. Each sound is windowed in post-processing to be functionally similar to the overlap-add method. Simulating generated sounds from three equally-spaced interactions with the SnakeSynth grid shows they sum to produce a single variable-length sound and the overlap of their windows shows the amplitude of the combined sound increase with greater sample overlap (Figure 4). This is chosen to produce interaction analogies to mechanical "driving" and resonance (as mentioned before) and other ways to blend overlapping audio could be explored.

3. DISCUSSION

By foregoing concatenation-based approaches and modelling the variability of audio length in terms of interaction we lose the precision of a triggered fixed-size model and we have to choose how to blend sounds in context. Still, what we gain in flexibility in terms of modularity and greater choice over DGMs should not be understated as it keeps the abundance of fixed-length GAN models and ongoing research available to us as design options. Non-generative models of the same dimension would even suffice. Going further this flexibility enables us to create novel controllers for audio DGMs capable of generating variable-length audio. This does not seem to be widely recognized as a design interest and surprisingly I have seen little discussion about it in previous work.

SnakeSynth offers one way around the problem by treating audio length as a parameter of interactive control outside of the generative model. This bridges the gap from fixed-length audio DGMs to controller-driven variable-length DGMs and even to DGM-based music performance by recognizing that asynchronously triggering audio clips over time is congruent to mapping user interactions over time. Given the ubiquity of cursor movements and touch in 2D digital coordinate spaces these seem to be an appropriate starting point for discussion and exploration of user interaction as a means of DGM control and particularly DGM control for musical expression.

Choices on how to map from the SnakeSynth coordinate grid to generator latent space(s) raise interesting questions about both the shape of the DGM latent spaces themselves but also how to construct novel and/or non-trivial maps between them and the SnakeSynth grid. We are not required to use quantile values as inputs either and interactions could be readily extended to any interface that produces at least two values in real time.

This is somewhere in the design of latent space-based control that existing human-computer interface principles like Fitts' law [3] could be applied such as knowledge about distance to target, target size, cognitive load, etc. Similarly the design of real-time sound blending beyond slowed attacks and windowing, especially for asynchronously triggered sounds, deserves further consideration. Semantically some of these design choices would reflect different views of the audio space or context at hand so as to be recognizable and learnable by performers and reproducible in performance settings.

DGM research continues to evolve at quick pace and we are still finding new ways to train high-fidelity GANs quickly enough, for example by progressively adding layers during training [8], that it may soon be feasible to train small GAN models in real time. This would enable SnakeSynth and derived tools to "evolve" new auditory spaces in response to real-time interactions and/or new data. More importantly this removes us from the mindset of fixed-length generative audio models and re-frames digital musical instruments as things capable of evolving to adapt to context and changing in turn how we might think about digital music tools for performance.

4. CONCLUSION

I showed how *SnakeSynth*, demoed as a web-based audio synthesizer, combines DGM audio and real-time continuous 2D input to create and control variable-length generative sounds through several interaction gestures made with analogies to strummed, bowed, brushed, and plucked musical instrument controls. I showed that I can modulate sound length and intensity by interacting with a programmable 2D coordinate grid, demonstrating the real-time potential for derived tools, and I briefly discussed possible futures for DGMs as an interactive paradigm for musical expression.

5. ACKNOWLEDGMENTS

I thank Robert for his guidance and for pointing to the novelty of real-time GAN synthesis in web browsers.

6. ETHICAL STANDARDS

The author is self-funded and reports no conflicts of interest. No living subjects were studied in this work.

7. REFERENCES

[1] C. Donahue, J. McAuley, and M. Puckette. Adversarial Audio Synthesis, Feb. 2019. arXiv:1802.04208 [cs].

[2] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts. GANSynth: Adversarial Neural Audio Synthesis, Apr. 2019. arXiv:1902.08710 [cs, eess, stat].

[3] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat].

[5] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. Deep AutoRegressive Networks, May 2014. arXiv:1310.8499 [cs, stat].

[6] E. Härkönen, A. Hertzmann, J. Lehtinen, and S. Paris. GANSpace: Discovering Interpretable GAN Controls, Dec. 2020. arXiv:2004.02546 [cs].

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32nd Intl. Conf. on Machine Learning*, pages 448–456. PMLR, June 2015.

[8] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation, Feb. 2018. arXiv:1710.10196 [cs, stat].

[9] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes, May 2014. arXiv:1312.6114 [cs, stat].

[10] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior,

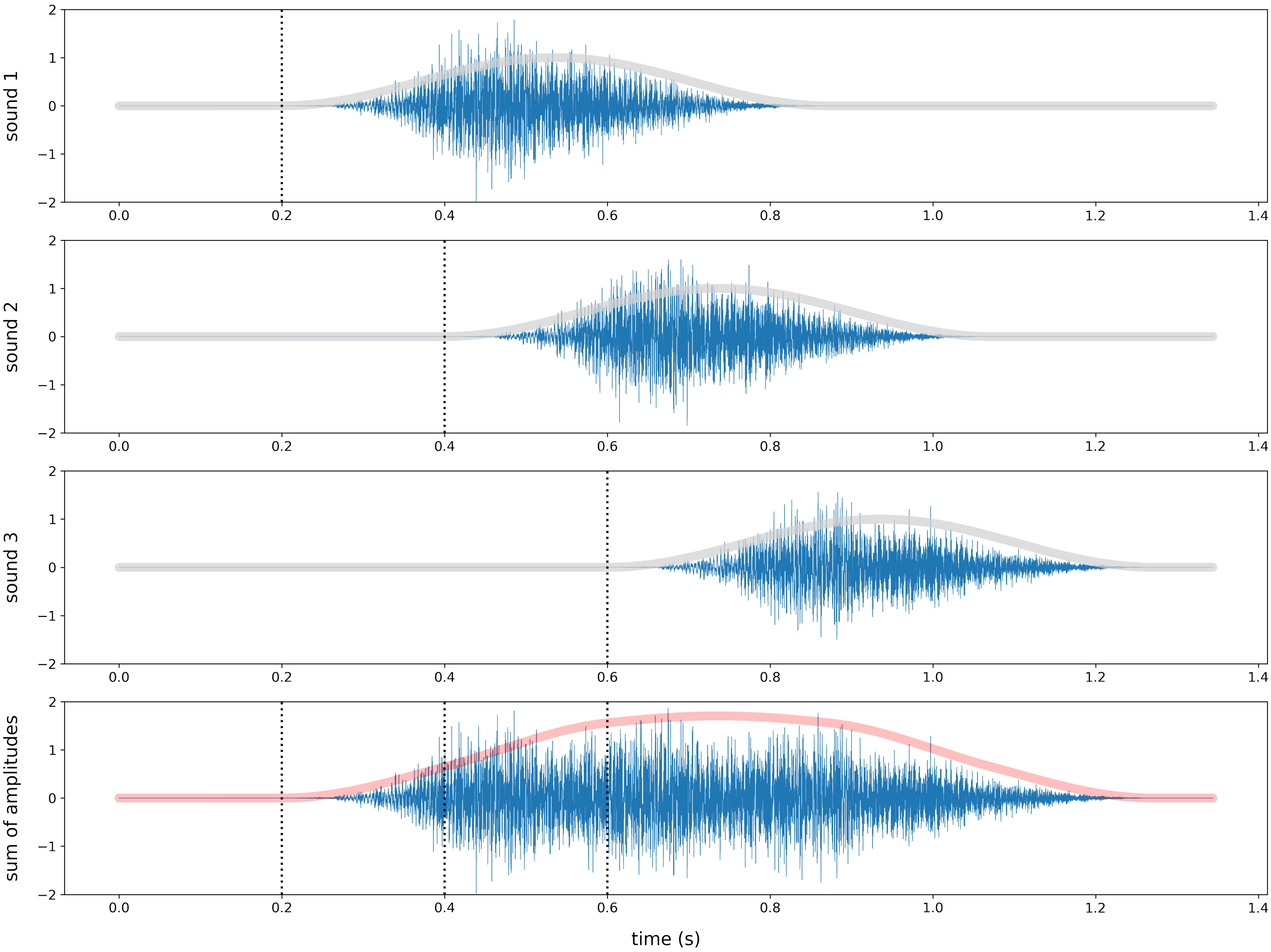

Figure 4: Time-triggered sound samples (first three rows) generated, windowed, and summed over time to produce a single variable-length sound (bottom row). Increasing amplitude by summing overlapping sound windows enables virtual mechanical "driving" through the SnakeSynth controller to produce resonance-like effects. The amplitude of the combined sound increases with greater sample overlap.

and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio, Sept. 2016. arXiv:1609.03499 [cs].

[11] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, Jan. 2016. arXiv:1511.06434 [cs].

[12] S. Shahriar. GAN computers generate arts? A survey on visual arts, music, and literary text generation using generative adversarial network. *Displays*, 73:102237, July 2022.

[13] H. Zhang, L. Xie, and K. Qi. Implement Music Generation with GAN: A Systematic Review. In *2021 Intl. Conf. on Computer Engineering and Application*, pages 352–355. IEEE, June 2021.

APPENDIX

A. INTERACTIONS

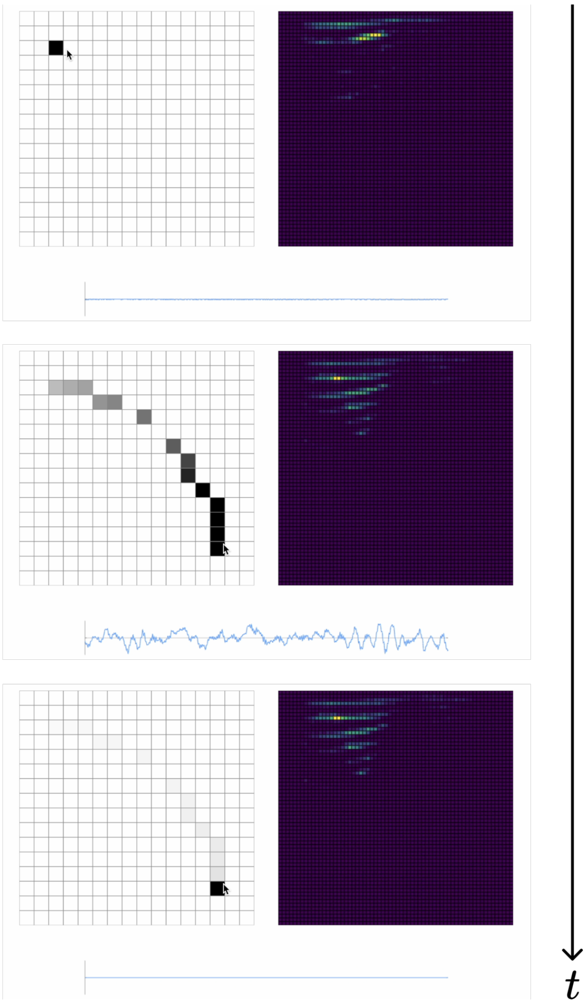

Figure 5: Linear or near-linear gestures produce variable-length audio (resembles "strumming"). Gesture distance determines sound length.

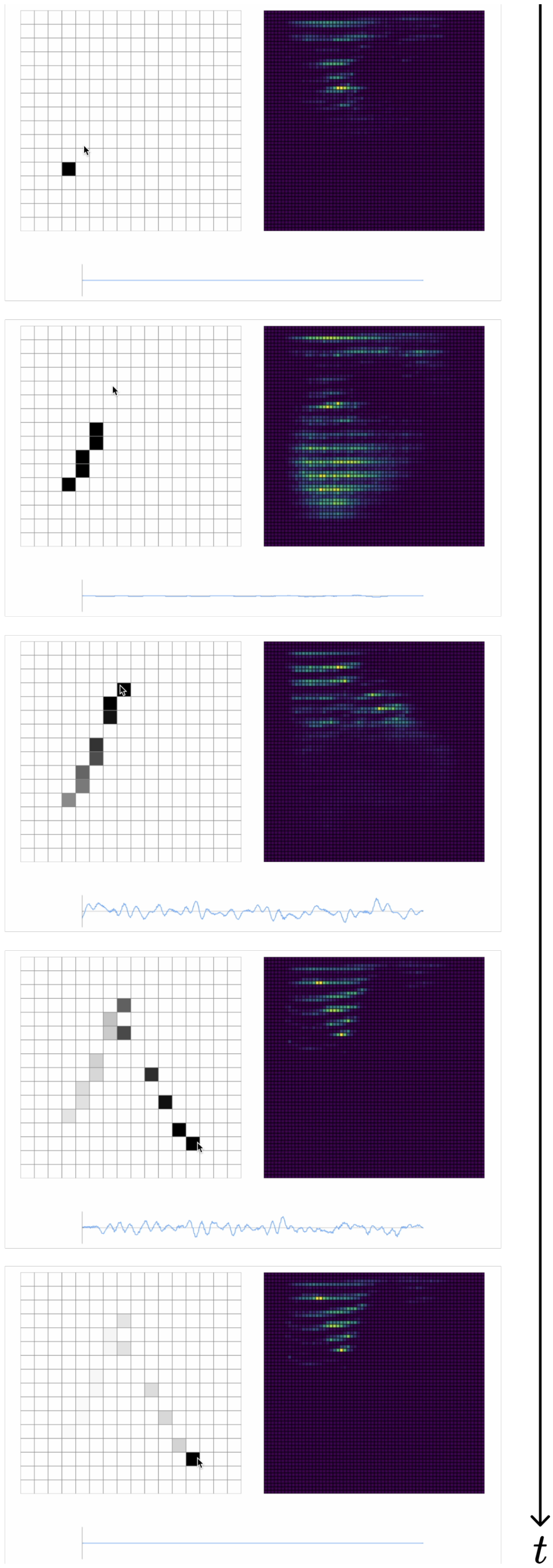

Figure 6: Suddenly changing movement direction creates sudden audio changes and corresponding audio attack (resembles a "finite bow").

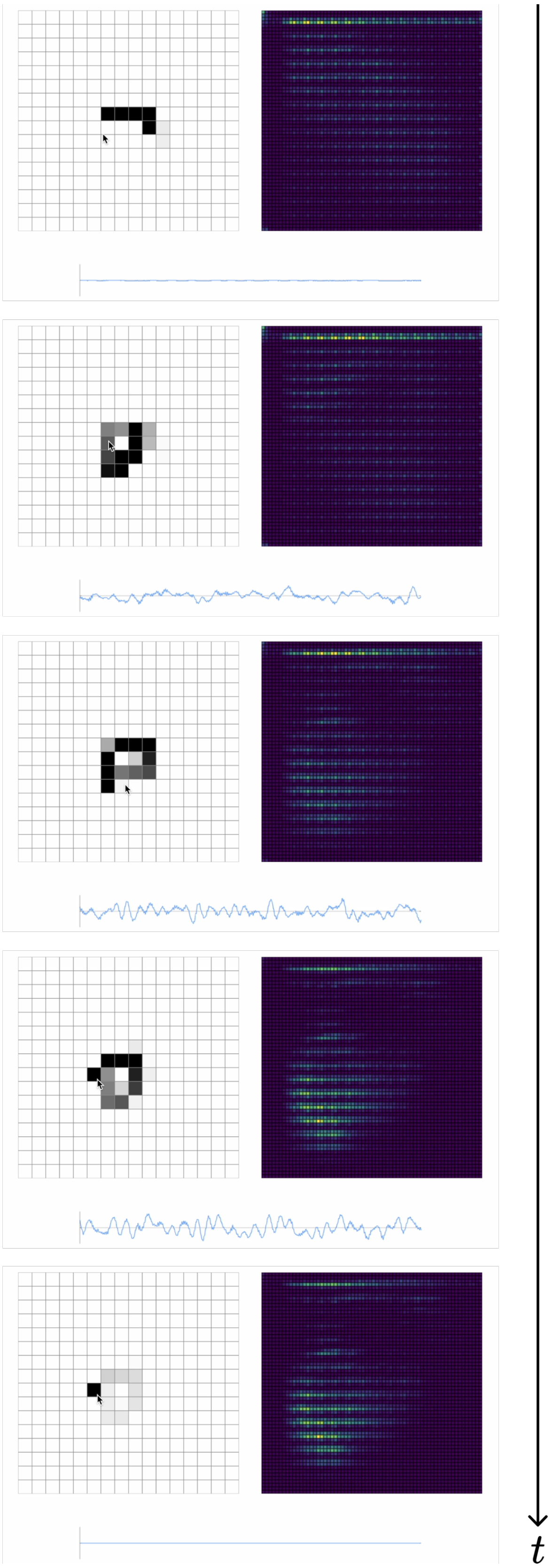

Figure 7: Continuous gestures create continuous audio (resembles an "infinite bow"). Particularly circular or near-circular gestures produce continuous *rhythmic* audio.

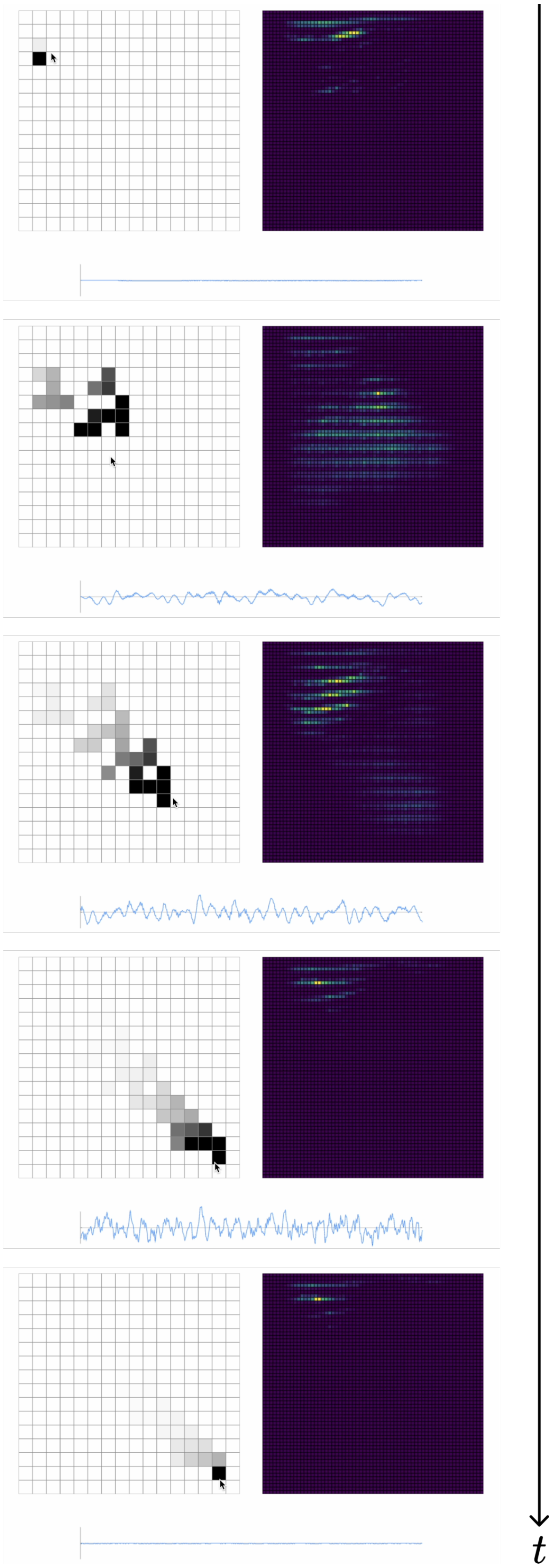

Figure 8: Chaotic gestures with many directional changes to linear and/or circular movements create cacophonous audio (resembles "brushing").