# WHITE BOX SEARCH OVER AUDIO SYNTHESIZER PARAMETERS

**Yuting Yang**[1] **Zeyu Jin**[2] **Connelly Barnes**[2] **Adam Finkelstein** [1]
[1] Princeton University [2] Adobe Research
[1]{yutingy, af}@princeton.edu, [2]{zejin, cobarnes}@adobe.com

## ABSTRACT

Synthesizer parameter inference searches for a set of patch connections and parameters to generate audio that best matches a given target sound. Such optimization tasks benefit from access to accurate gradients. However, typical audio synths incorporate components with discontinuities – such as sawtooth or square waveforms, or a categorical search over discrete parameters like a choice among such waveforms – that thwart conventional automatic differentiation (AD). AD libraries in frameworks like TensorFlow and PyTorch typically ignore discontinuities, providing incorrect gradients at such locations. Thus, SOTA parameter inference methods avoid differentiating the synth directly, and resort to workarounds such as genetic search or neural proxies. Instead, we adapt and extend recent computer graphics methods for differentiable rendering to directly differentiate the synth as a white box program, and thereby optimize its parameters using gradient descent. We evaluate our framework using a generic FM synth with ADSR, noise, and IIR filters, adapting its parameters to match a variety of target audio clips. Our method outperforms baselines in both quantitative and qualitative evaluations.

## 1. INTRODUCTION

Synthesizers provide musicians and sound designers with flexibility for exploring sound with various audio characteristics. However, the versatility of synths also poses challenges in terms of control, because manually searching over numerous parameters to seek a particular type of sound requires expertise, time, and effort. Synth parameter inference addresses these challenges by automating this search process to find parameters that best match a given target sound. Given a synth $f$ with parameters $\boldsymbol{\theta}$ and a target $T$, the search seeks the optimal parameters $\boldsymbol{\theta}^*$ to minimize some loss $L$ between the synth output and the target.

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\, L(f(\boldsymbol{\theta}), T) \tag{1}$$

If the synth $f$ can be expressed as a white box program, a straightforward solution to Equation 1 would differentiate $L$ wrt the parameters $\boldsymbol{\theta}$, and then minimize $L$ by gradient descent. However, in practice, typical synthesizers $f$ contain discontinuous oscillators, like square or sawtooth waveforms, and discrete categorical parameters, such as choosing different waveforms and modules, that thwart traditional automatic differentiation (AD).

Researchers have developed several workarounds to avoid directly differentiating $f$. For example, genetic algorithms [1, 2] approximately solve Equation 1 at the expense of greater computation and potential artifacts from failures near local minima. Alternatively, Equation 1 may be approximated as black box models using deep learning: either the synthesizer can be approximated via a differentiable neural proxy [3,4], or the entire argmin mapping can be approximated by a parameter prediction network [5, 6]. Similarly, the parameter space can be mapped to a VAE latent space for direct control [7]. However, the flexibility of deep learning approaches is constrained, as data collection and training are typically limited to specific synthesizers with fixed parameter choices, making it impractical to directly apply trained models to arbitrary synthesizers.

Graphics researchers developed recent methods to approximate the gradient for discontinuous white box image generation processes [8–10]. These generally integrate over the discontinuous function $f$, and approximate the gradient for the integral $\hat{f}$. This paper builds on A$\delta$ [10], which replaces the traditional calculus rules for AD to directly enable backpropagation on arbitrary discontinuous programs. Our method relies on the key observation that the discontinuous function will eventually be band-limited and sampled at some rate (e.g. 48kHz). Each sample represents an integration over a time interval that *may contain* a discontinuity. However, the band-limited function $\hat{f}$ is continuous so differentiation rules can be developed for it.

Our optimization framework differentiates a pre-filtered white box synth, and solves Equation 1 via gradient descent. We adapt and extend the math in A$\delta$ [10] to differentiate discontinuous and discrete synth components, and also introduce heuristic methods for better convergence. We evaluate on a FM synthesizer and our approach finds parameters that better match the target than baselines qualitatively and quantitatively. Moreover, our framework allows musicians to incorporate domain expertise to flexibly modify and fine-tune synth modules. Because our white box approach does not incur training overhead, our framework can be flexibly applied to arbitrary synth programs.

## 2. RELATED WORK

Researchers have explored a variety of techniques to automatically search for optimal synthesizer parameters with-

out having to explicitly differentiate the synthesizer. Genetic algorithm (GA) approaches [1, 2] mutate and cross variants to search over the entire program space for arbitrary synthesizers, but suffer from excessive computation and difficulty in accurately converging to local minima without the guidance of the gradient. On the other hand, deep learning models can be used to directly predict the synthesizer parameters [5, 6, 11]. However, they heavily rely on the annotated datasets of synthesizer presets, therefore cannot be flexibly generalized to *any* synthesizer. Similarly, each trained model can only be used for one particular synthesizer patching, greatly limiting the flexibility of the method. Unlike learning methods, our approach does not rely on a dataset, and can flexibly differentiate *any* white-box program, supporting finetuning and parameter transfer between synthesizer patches. Our gradient-based process also converges more robustly than GA.

Alternatively, synthesizers can be defined by differentiable functions, therefore allowing optimal parameters to be learned through gradient descent. For example, neural audio synthesis methods use black-box neural networks to generate audio samples [12, 13]. The neural proxies can be combined with continuous synthesizer components as well, such as DDSP methods that incorporate digital signal processing modules [4, 14], and DWTS methods with learnable wavetables [15]. However, because these methods use continuous proxies, they usually do not match the exact parameterization of complicated discontinuous synthesizers, therefore cannot be flexibly used to control conventional synthesizers. Moreover, the neural modules introduce nontrivial inference overhead and are less efficient than synthesizers. Unlike the differentiable neural proxies, our method directly differentiates a white box program that can be any desired synthesizer. Therefore, it optimizes semantically meaningful parameters.

We leverage recent ideas from differentiable rendering in computer graphics. Researchers developed compiler frameworks to systematically differentiate arbitrary discontinuous programs [8, 10], and application-specific solutions to efficiently differentiate specific types of discontinuities in the rendering pipeline [9, 16]. Our method differentiates synthesizer discontinuities by combining these two approaches: we adapt the gradient rules from A$\delta$ [10] for use with discontinuous audio waveforms, and introduce a specialized gradient rule for discrete categorical choices.

## 3. METHOD

This section describes our optimization pipeline for synth parameter inference. Section 3.1 introduces our approach to differentiating a synth. Section 3.2 considers loss function options. Finally Section 3.3 discusses how to explore the multi-modality and avoid local minima.

### 3.1 Approximating the Gradient

We introduce a customized gradient, which includes differentiating at discontinuities, avoiding plateaus with zero gradient, and efficiently differentiating IIR filters. We first introduce A$\delta$'s [10] gradient rule for differentiating discontinuities and discuss its usage in audio synthesizers in Section 3.1.1, followed by our novel synthesizer-specific gradient rules in Sections 3.1.2 - 3.1.4.

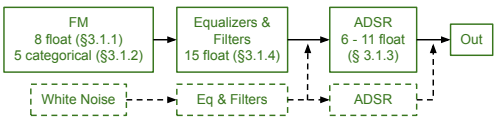


**Figure 1**. Summary for our FM synthesizer and how they are differentiated. Dashed boxes and arrows are optional components whose connection is decided per target.

#### *3.1.1 Differentiating Discontinuous Waveforms*

We view discontinuities as compositions of the Heaviside step function $H$, which evaluates to 0 on the one side of a discontinuity, and 1 on the other side. The discontinuity can be differentiated using the gradient rules from A$\delta$ [10]. The key idea is to approximate the gradient as if the discontinuous function is first convolved with a 1D box filter $\phi(t)$ along the time dimension $t$. As an example, if $H$ is controlled by a continuous function $c$, we can differentiate the convolution of $H(c(t,\theta))$ with $\phi(t)$ by applying the Dirac delta's scaling property at the discontinuity $t_d$.

$$\frac{\partial}{\partial\theta}\int H(c(t',\theta))\phi(t-t')dt' = \int \delta(c)\frac{dc}{d\theta}\phi(t-t')dt'$$
$$= \int \frac{\delta(t'-t_d)\frac{dc}{d\theta}}{|\frac{dc}{dt}|}\phi(t-t')dt' = \phi(t-t_d)\frac{\frac{dc}{d\theta}}{|\frac{dc}{dt}|}\,|_{t_d}$$

This can be approximated with two samples corresponding to two ends of the box kernel $\phi$, denoted as $t^+$ and $t^-$. The box kernel $\phi(t-t_d)$ either evaluates to 0 or $\frac{1}{t^+-t^-}$, depending on whether $H(c(t,\theta))$ evaluates to the same or different values at $t^+$ and $t^-$. Because $c$ is continuous, $dc/d\theta$ can be computed with AD, and its evaluation on either $t^+$ or $t^-$ approximates that of $t_d$ because Lipschitz continuous functions are locally bounded. Finally, $dc/dt$ is approximated by finite difference: $\frac{c(t^+,\theta)-c(t^-,\theta)}{t^+-t^-}$. Because the audio signal already samples at a regular interval along the time dimension (e.g. 48kHz), we conveniently set the support of the box kernel to straddle the current sample and its neighbor. While the mathematical correctness in [10] is derived assuming a single discontinuity in the neighborhood, empirically the approximated gradient also works well for signals with sparse multi-discontinuities, such as when both the carrier and FM modulation waves are square. However, if the sampling rate is too low and causes aliasing, the A$\delta$ rule is unable to correctly approximate the gradient as if the signal was antialiased.

Discontinuous waves such as square and sawtooth can be constructed as periodic compositions of $H$. These discontinuities are differentiated using the gradient rules introduced in A$\delta$ [10] that are analogous to the equation above, where $\theta$ might e.g., be the frequency of a square wave. The gradient for the synthesizer parameters are obtained by differentiating the loss term (Section 3.2) using A$\delta$ gradient rules, which reduce to traditional AD for continuous parameters, combined with our

customized gradients (Sections 3.1.2 - 3.1.4). This approach is more accurate than differentiating a discontinuity naively smoothed with arbitrary linear or sigmoid transitions, especially when discontinuities are composited – for example, the composition of discontinuous modulation and carrier signals in an FM synthesizer.

#### *3.1.2 Differentiating Discrete Categorical Choices*

Section 3.1 discusses a simple scenario where the discontinuity can be sampled along the time dimension. However, the challenge remains for the discrete categorical choices, because for fixed parameterization, the corresponding discontinuity $H$ evaluates to a constant for any time $t$, therefore the discontinuity cannot be easily sampled.

This section proposes a stochastic approach to differentiate the discrete parameters. We define a categorical node $g$ as taking input from a discrete parameter $x$ with potential choices $A$, $B$, ..., and outputs to a floating point value:

$$g(x;\boldsymbol{\theta}) = \begin{cases} g_A(\boldsymbol{\theta}) & \text{if } x == A \\ g_B(\boldsymbol{\theta}) & \text{if } x == B, \\ \dots \end{cases} \tag{2}$$

$g_A, g_B$ are floating point functions associated with choices $A, B$ respectively, such as sine or square wave equations.

Our stochastic approach views the discrete parameter $x$ as a discrete random variable $\mathcal{X}$ with different samples $X$ at different time steps. Therefore $g(\mathcal{X};\boldsymbol{\theta})$ is a random variable as well. Throughout this section, we will use lowercase (e.g. $x$) for the synth parameters that need to be optimized, calligraphic (e.g. $\mathcal{X}$) for its corresponding random variables, and regular uppercase (e.g. $X$) for sampled values from the random variable. Note when $\mathcal{X}$ has close to zero variance, it consistently samples the same choice for every time step, therefore $X$ can be viewed as a constant identical to $x$. We further model $g(\mathcal{X};\boldsymbol{\theta})$ similarly to an argmax operator, where each potential choice $A$, $B$, ... is associated with a "score" random variable, and the output of $g$ corresponds to the choice with the highest "score". Specifically, the "score" for choice $A$ is modeled as $\mathcal{Y}_A = \mu_A + \sigma_A \cdot \mathcal{U}$, where $\mu_A, \sigma_A$ are the mean and standard deviation, and $\mathcal{U}$ is a uniform random variable with zero mean and unit variance. For any two neighboring samples with disagreeing categorical choices $A$ and $B$, we view the inconsistency as a discontinuous branching conditioned on whether the sampled "score" $Y$ for choice $A$ is greater than $B$ or not: $g = \text{select}(Y_A > Y_B, g_A, g_B)$. By forming the discontinuity this way, the gradient wrt $\mu_{A/B}, \sigma_{A/B}$ can be easily computed with the A$\delta$ gradient rules on the time domain. At convergence, the variance to every "score" variable should be reduced to a small value such that the categorical choice is sampled consistently.

The stochastic gradient rule works best when there is a high correlation between the functions associated with each choice $g_A, g_B$, etc. Intuitively, this allows $g_A, g_B$, ... to form a smaller convex hull for the sampled output $g(X;\boldsymbol{\theta})$, therefore reducing the variance of the gradient estimation. Therefore when differentiating categorical waveform choices, we align the phase of the wave functions such that their correlation is maximized.

#### *3.1.3 Avoiding Zero Gradient in Plateaus*

Many synthesizer parameters have constraints on their values, such as the period for ADSR stages should be nonnegative, and the filters' cutoff frequencies should be within a range to avoid singularities. A typical strategy for optimizing these constrained parameters in an unconstrained problem is to clamp the parameters: taking the min and max against their upper and lower bounds. However, clamping introduces another challenge for optimization: once the parameter clamped, the gradient wrt the parameter becomes zero across an entire "out of bounds" plateau in the loss function. For example, $\frac{\partial \max(\theta,0)}{\partial \theta} = 0$ whenever $\theta < 0$.

We propose a heuristic workaround that avoids constrained parameters getting stuck at out-of-bound values, via a customized gradient for the min (or max) operator $f$:

$$\begin{aligned} f &= \min(\theta, C) \\ \frac{\partial L}{\partial \theta} &= \text{select}(\theta < C, \frac{dL}{df}, \max(\frac{dL}{df}, 0)) \end{aligned} \tag{3}$$

Here the min operator compares with constant $C$, and we assume the gradient wrt $f$ is already computed as $dL/df$. Note only the blue term in Equation 3 is different from traditional AD. The gradient for the max operator is similar to Equation 3, but $<$ and max are replaced by $>$ and min respectively. Note this is only a heuristic workaround for reverse-mode AD, and can not be used for forward-mode because it computes $dL/df$ before differentiating $f$. Intuitively, our customized gradient will push the out-of-bound $\theta$ back to its valid range whenever the gradient wrt $f$ wishes to bring the clamped value back to valid. We only apply this workaround when constraining parameter values against a constant, and generic min/max comparisons between two non-constants are still differentiated by AD.

#### *3.1.4 Efficient IIR Filter Back-propagation*

Infinite impulse response (IIR) filters are widely used in synths to flexibly control the timbre. However, differentiating the IIR filter introduces performance challenges because each output value at a certain time step recurrently depends on every input/output value in previous steps, and naively unrolling the gradient in the time domain is computationally expensive. We, therefore, avoid the complex dependency in the time domain by applying the filter in the frequency domain similar to [3]. During optimization, we only differentiate the multiplication between the unfiltered spectrogram and the frequency response of the filters. Because most popular filters (e.g. Biquad, Butterworth) used in synthesizers already have closed-form solutions for their frequency responses, requiring a frequency domain proxy does not restrict the expressiveness of this approach.

### 3.2 Loss Function

Unlike supervised deep-learning methods that could rely on losses in the parameter space at the cost of collecting the preset dataset, our optimization pipeline can only rely

on spectral and time domain losses. However, finding the ideal loss that is consistent with human perception is challenging for several reasons. Firstly, standard losses such as L2 on the (log mel) spectrogram only work well when distances between two signals are smaller than just noticeable difference (JND); but this is rarely the case during our optimization, as we start with random initial guesses, and the synthesizer may never even approach JND to an out-of-domain target. Furthermore, although deep perceptual metrics have been developed for speech signals (e.g., [17]), they do not generalize well to music synths.

We propose a heuristic combination of several different losses to approximate the perceptual similarity. The intuition is that the gradient to the majority of the losses should agree with human perception even if a few of them are noisy. In addition to standard losses, we also include the 1D Wasserstein distance [18] along the frequency dimension because of its wide applicability in matching distributions. Our final optimization loss is a weighted combination of the Wasserstein distance, L2, log mel L2, and a deep feature distance from the wav2clip model [19]. The weights are chosen such that each component has a relatively equal contribution. For the losses that work on a spectrogram (L2, log mel L2, and Wasserstein), we use three different window sizes (512, 1024, 2048) with 75% overlap between windows. The deep feature loss also uses the same window and hop sizes, but for efficiency, we stochastically evaluate the model using one of the window sizes per iteration. Additionally, because the deep feature model takes time domain signal as input, we need to apply inverse STFT to the spectrogram because of the frequency domain IIR approximation described in Section 3.1.4.

### 3.3 Identifying Perceptually Similar Results

Gradient-based optimizations may converge to a variety of local minima with different perceptual similarities to the target. Our framework runs multiple random restarts to avoid getting stuck at local minima. However, we are not aware of a quantitative metric that reliably characterizes the perceptual similarity for synthesizers [20, 21]. While we use our weighted loss in Section 3.2 to provide a gradient for the optimization, its absolute value does not precisely correspond to perceptual similarity: perceptually dissimilar results sometimes have lower loss than similar results. Thus, manual selection is needed to choose the best results. We also implement early termination to avoid wasting compute at local minima, and also a mechanism to identify good quality results after convergence.

Our early termination strategy is a generalization to the intuition that good initializations have a higher probability of good convergence. We generalize the heuristic to arbitrary iterations within the optimization, and terminate the ones with bad results at the end of a sequence of predetermined iterations. Additionally, because the weighted loss in Section 3.2 cannot reliably characterize perceptual similarity, we rely on the Pareto ranking [22] on multiple losses to identify bad results. We terminate optimizations whose Pareto rank on every non-deep-learning loss in Section 3.2 is higher than ceil(0.5max_rank), where max_rank is the maximum Pareto rank for the current population. Our implementation checks for early termination every 100 iterations, starting at iter 200, and we run every optimization until full convergence and simulate the early termination.

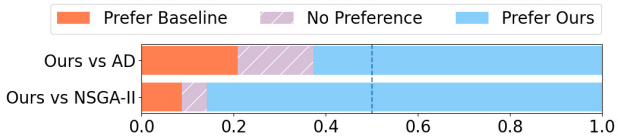


**Figure 2**. MOS listening test preference distribution.

We also note that when the optimization result is already close to the target at convergence, its loss metrics calculated from a larger window size better resemble perceptual similarity. Specifically, large L2 errors are usually bad. We therefore further omit any converged result whose L2 loss on the spectrogram with window size 2048 is 2x higher than the lowest among all results, and finally rank the remaining results based on the weighted sum of Wasserstein, L2, and log mel L2 on the same spectrogram.

## 4. VALIDATION

This section validates our proposed framework by optimizing the parameters of an FM synthesizer to match various audio signals for musical instruments and special sound effects. All the targets are downloaded from the web and are therefore out of domain. We first describe our FM synthesizer in Section 4.1 and evaluation setup in Section 4.2, then compare our method with two baselines through a user study (Section 4.3). Section 4.4 also shows the optimization convergence. Finally, Section 4.5 demonstrates the flexibility of our framework with a case study that modifies the synthesizer modules for better quality result.

### 4.1 Synthesizer Model

We choose an FM synthesizer as in Figure 1 following the recommendation from a synthesizer expert, because it is simple yet expressive enough to approximate most of our target signals. It has one carrier signal modulated by the weighted sum of four different signals. Each signal is parameterized with a categorical choice from the four base waveforms: sin, square, triangle, and sawtooth. Each modulation signal is also parameterized by ratio and index, which controls the frequency and the magnitude of the modulation. The FM signal will further be filtered by three Biquad equalizers (low/high shelf, peak) parameterized by their cutoff frequency, resonance and gain, and a pair of Butterworth low/high pass filters parameterized by their cutoff, bandwidth, and attenuation. After that, the filtered signal is multiplied by an ADSR parameterized with the duration of each stage, overall volume and that of sustain, the starting time of the attack, and optionally the exponential decay of the release as well as the scale, frequency, and phase to an optional AM envelope applied to attack, decay, and sustain. Finally, filtered white noise can be optionally added either by sharing the original ADSR or with a differ-

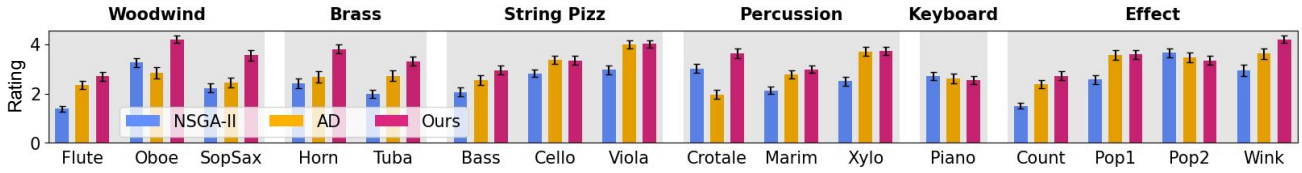


**Figure 3**. MOS listening test ratings (higher is better) for each of 16 target clips, grouped in 6 categories. Error bars correspond to 2SEM (standard error of mean). To save space we shorten names: Marim(ba), Xylo(phone), and Count(down).

ent ADSR. Optional configurations are included based on audio characteristics. For example, sustained sounds such as woodwind and brass uses the AM envelope for ADSR, and shorter sound such as percussion includes a filtered white noise with separate ADSR. The overall model includes 40 (e.g. oboe) - 70 (e.g. crotale) parameters.

We implement the FM synthesizer in PyTorch to leverage its AD framework. The gradient discussed in Section 3.3 is implemented as the cutomized backward pass, and AD is used for the rest of the computation (e.g. ADSR, STFT). Note this could also be generated by a compiler for arbitrary synthesizers similar to A$\delta$ [10].

### 4.2 Evaluation Setup

We compare with two baselines: traditional AD and zeroth order optimization with genetic algorithm NSGA-II. AD baseline uses the same optimization framework described in Section 3, except that the gradient described in Sections 3.1.1 - 3.1.3 is replaced by traditional AD. The zeroth order baseline does not require any gradient, and instead uses the genetic algorithm NSGA-II [23] to search over the parameter space. Because NSGA-II is multi-objective, it directly finds Pareto optimal solutions to the various loss functions in Section 3.2 without having to compute their weighted sum as in gradient-based optimization.

We use 16 different target sounds, including 12 musical instruments and 4 special sound effects listed in Figure 3. For ours and AD, we run the experiment with 100 random restarts for a maximum of 2000 iterations per restart. Note that because of the early termination described in Section 3.3, the actual number of iterations per restart varies. We additionally supply the NSGA-II with a reasonable sample range to the parameters, and run the algorithm with 100 population size and 2000 generations.

### 4.3 MOS Listening Test

As mentioned in Section 3.2, we have no perceptually-accurate loss for comparing synth output to target audio. Therefore, we rely on a Mean Opinion Score (MOS) test to qualitatively compare results for our method and baselines. For each method and target sound, we use the top 4 results based on the Pareto ranking from Section 3.3 for testing, resulting in 12 samples across 3 methods: ours, AD, and NSGA-II. These clips may be heard in our supplemental material.

Workers on Amazon Mechanical Turk (AMT) rate how similar each result is to the target on a scale of 1 (bad) - 5 (identical). They are "master" workers, English-speakers in the US, and are paid $20 per hour. Each worker is asked to rate all 12 samples for two different targets. We further embed four validation tests to filter out careless ratings: two that are intentionally corrupted from the two targets to be worse than any of the 12 samples to be rated, and two that are identical to targets randomly chosen from all 16 targets. Therefore each worker rates $2 \times 12 + 4 = 28$ samples for each assigned task called HIT (Human Intelligence Task). In the end we collected 240 valid HITs where each audio sample gets 30 ratings from 30 different workers.

We compute a preference score for each worker and each instrument: we calculate a mean rating for each method over the 4 rated samples. If Method 1 has a higher score than Method 2, we say that Method 1 is preferred by this worker. Figure 2 shows the preferences among pairs of methods aggregated across all workers. Our method outperforms both baselines by a larger margin, but AD is preferred more than NSGA-II. We compute the p-value for the hypothesis: *our average rating per user per instrument is higher than that of the baseline.* The p-value for the AD baseline is 2e-8, and for the NSGA-II baseline is 3e-61.

We additionally report in Figure 3 the rating for each target. Ours performs best when the FM synth is a good emulation of the underlying instrument , such as for woodwind or brass. AD has similar ratings to ours more frequently than NSGA-II, which is consistent with Figure 2. Note in all cases when baselines have similar or higher ratings than ours, the rating difference is always within the error bar, indicating the preference is not statistically significant. We characterize the cases where ours and baselines have similar ratings into two scenarios. The first one is when the target is less challenging, and can be easily reconstructed by various local minimums, such as Pop1 and Pop2. The second scenario is when the FM synth cannot nicely emulate the instrument, such as for Piano. Therefore none of the methods can converge close enough to the target, resulting in similarly low ratings.

### 4.4 Optimization Convergence

This section discusses the optimization convergence to demonstrate how frequently each method converges in the optimization. Figure 4 demonstrates two representative results: Horn for ours outperforms baselines and Xylophone for ours performs similarly to baseline AD. In both plots, all 100 populations for NSGA-II converge similarly because bad results are removed at the end of each generation. Unlike genetic algorithms, the 100 optimizations for both ours and AD have diverging performances because gradient-descent only explores the local parameter space and may be stuck at a local minimum. The early termina-

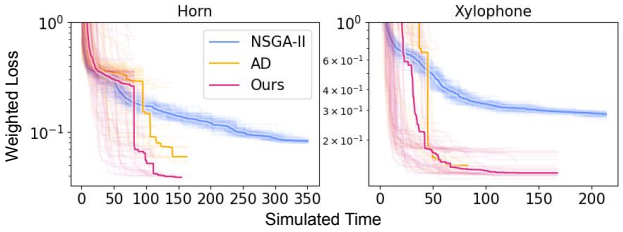


**Figure 4**. Comparing the convergence of ours and baselines for the 100 random restarts of two tasks. The x-axis reports simulated time: the number of function evaluations scaled with the actual runtime for each method. The y-axis reports the weighted loss for the optimization. For ours and AD, each transparent line corresponds to a restart. For NSGA-II, each transparent line plots the loss for the kth population at each generation ($k \in [1, 100]$). The median within all runs at a given time is shown as the solid line.

tion described in Section 3.3 conservatively removes some of the local minimums, but more importantly reduces the number of evaluations toward the end of the optimization because fewer restarts are still active. Typically, the convergence plot is consistent with the listening test result in Figure 3, but with the exception of Oboe, where NSGA-II converges to the lowest error, but its listening test performs worse than ours. But this is simply due to the choice of weights that combine multiple losses into one scalar: NSGA-II converges to lower Wasserstein and higher L2 and log mel L2, thus it is *not* Pareto superior to ours.

### 4.5 Case Study: Modify Synthesizer Modules

This section uses the Xylophone target as a case study to demonstrate that our white box method can be flexibly combined with user expertise to modify the synthesizer components to improve the quality of generated audio.

Similar to other targets, Xylophone is initially approximated by the synth model described in Section 4.1. It uses filtered white noise with independent ADSR to model the strike at the start of the sound. However, the optimization result is not ideal, specifically, the beginning of the audio sounds very different from the target. This can be verified by Figure 5, which compares the spectrogram for the first 0.07s of the sound between the target (a) and the optimization(b): the optimization has a longer attack stage.

We ask a synthesizer expert to identify the potential cause of the inconsistency: instead of using filtered white noise, the beginning of the audio may be better approximated by an impulse with IIR filters. We, therefore, use the following impulse component to replace the original filtered white noise. We first manually calibrate the starting time of the Xylophone within the target audio, and set the impulse at that location. Similar to the white noise, the impulse is also filtered by three Biquad equalizers (low/high shelf and peak) and a pair of low/high-pass Biquad filters. Because the impulse is not static, we have to optimize the IIR parameters in the time domain rather than the frequency domain as in Section 3.1.4. Therefore we avoid using any Butterworth filters mentioned in Section 4.1 for a faster backward pass. Because the original optimization

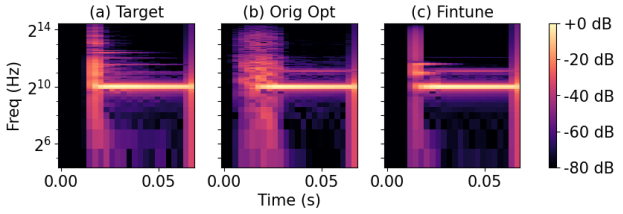


**Figure 5**. Visualizing the spectrogram for the Xylophone target (a), original optimization (b) using filtered white noise described in Section 4.1, and finetune result (c) using an impulse module described in Section 4.5. The spectrogram is computed with window size 512 and hop size 128.

nicely approximates the target except at the beginning, we only compute the loss for the first 2048 samples, and keep all the FM-related parameters fixed to only optimize the newly added IIR parameters, the scale of the impulse, and the original ADSR parameters that are initialized with their previously optimized values. To better characterize the filtered impulse signal, we use smaller spectrogram window sizes: 128, 256, and 512 with 75% overlap. Figure 5(c) shows the spectrogram of the finetune result that indeed better matches the attack stage of the target. Perceptually it also sounds better: please refer to supplemental material.

Note the finetuning process described in this section cannot be supported by deep learning methods without recollecting a new dataset and re-training the model for any change in the synthesizer design. Because our method directly optimizes the white-box programs, we can flexibly add the synthesizer components and reuse any parameters from previous optimizations.

## 5. CONCLUSION AND FUTURE WORK

This paper proposes to find synthesizer parameter settings that best match a given target sound by directly differentiating the white-box synthesizer program. We adapt and extend recent methods from differentiable rendering to differentiate the discontinuous and discrete components of the synthesizer, and design an optimization pipeline to solve the problem through gradient descent. We validate our method through user studies on Mechanical Turk, where our result is preferred over baselines by a large margin. We further demonstrate the benefit of differentiating white-box programs through a case study, where we can flexibly modify and finetune synthesizer components.

This work suggests several directions for future research. Our framework only searches for synthesizer parameters, and leaves patch connections fixed. Nevertheless, the gradient rules described in Section 3.1 provide a potential solution. It could be easily extended to optimize binary connection decisions, therefore the general patch connection could be optimized if viewed as compositions of binary choices. Additionally, because no perceptually accurate loss exists for music, our framework relies on a combination of various loss terms (Section 3.2) together with a Pareto rank based early termination strategy to improve convergence. Future work on perceptual similarity could simplify and improve our process.

## 6. REFERENCES


[1] 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.

[2] 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.

[3] N. Masuda and D. Saito, "Synthesizer sound matching with differentiable dsp." in *ISMIR*, 2021, pp. 428–434.

[4] F. Caspe, A. McPherson, and M. Sandler, "Ddx7: Differentiable fm synthesis of musical instrument sounds," *arXiv preprint arXiv:2208.06169*, 2022.

[5] G. Le Vaillant, T. Dutoit, and S. Dekeyser, "Improving synthesizer programming from variational autoencoders latent space," in *Proceedings of the 24th International Conference on Digital Audio Effects (DAFx20in21)*, Sep. 2021.

[6] O. Barkan, D. Tsiris, O. Katz, and N. Koenigstein, "Inversynth: Deep estimation of synthesizer parameter configurations from audio signals," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2385–2396, 2019.

[7] P. Esling, N. Masuda, A. Bardet, R. Despres *et al.*, "Universal audio synthesizer control with normalizing flows," *arXiv preprint arXiv:1907.00971*, 2019.

[8] S. Bangaru, J. Michel, K. Mu, G. Bernstein, T.-M. Li, and J. Ragan-Kelley, "Systematically differentiating parametric discontinuities," *ACM Trans. Graph.*, vol. 40, no. 107, pp. 107:1–107:17, 2021.

[9] S. Bangaru, T.-M. Li, and F. Durand, "Unbiased warped-area sampling for differentiable rendering," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 245:1–245:18, 2020.

[10] Y. Yang, C. Barnes, A. Adams, and A. Finkelstein, "A$\delta$: Autodiff for discontinuous programs - applied to shaders," in *SIGGRAPH, to appear*, Aug. 2022.

[11] J. Shier, G. Tzanetakis, and K. McNally, "Spiegelib: An automatic synthesizer programming library," in *Audio Engineering Society Convention 148*. Audio Engineering Society, 2020.

[12] A. v. d. 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," *arXiv preprint arXiv:1609.03499*, 2016.

[13] 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*. PMLR, 2017, pp. 1068–1077.

[14] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "Ddsp: Differentiable digital signal processing," *arXiv preprint arXiv:2001.04643*, 2020.

[15] S. Shan, L. Hantrakul, J. Chen, M. Avent, and D. Trevelyan, "Differentiable wavetable synthesis," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 4598–4602.

[16] T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley, "Differentiable vector graphics rasterization for editing and learning," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 39, no. 6, pp. 193:1–193:15, 2020.

[17] P. Manocha, A. Finkelstein, R. Zhang, N. J. Bryan, G. J. Mysore, and Z. Jin, "A differentiable perceptual audio metric learned from just noticeable differences," in *Interspeech*, Oct. 2020.

[18] Wikipedia contributors, "Wasserstein metric — Wikipedia, the free encyclopedia," 2023, [Online; accessed 15-April-2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Wasserstein_metric&oldid=1147354544

[19] H.-H. Wu, P. Seetharaman, K. Kumar, and J. P. Bello, "Wav2clip: Learning robust audio representations from clip," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

[20] F. Pachet and J.-J. Aucouturier, "Improving timbre similarity: How high is the sky," *Journal of negative results in speech and audio sciences*, vol. 1, no. 1, pp. 1–13, 2004.

[21] K. Siedenburg and D. Müllensiefen, "Modeling timbre similarity of short music clips," *Frontiers in psychology*, vol. 8, p. 639, 2017.

[22] P. Sitthi-Amorn, N. Modly, W. Weimer, and J. Lawrence, "Genetic programming for shader simplification," *ACM Trans. Graph.*, vol. 30, no. 6, p. 1–12, dec 2011. [Online]. Available: https://doi.org/10.1145/2070781.2024186

[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.