NUSynth: BROWSER RNBO SYNTHESIZER WITH GENERATIVE PARAMETER CONTROL

Jakob Philippe
Northeastern University
Boston, MA
philippe.j@northeastern.edu

ABSTRACT

NUSynth, developed using RNBO, is a polyphonic synthesizer accessible via a browser-based interface. This synthesizer integrates three oscillators, ADSR envelope, FM synthesis, filters, and LFOs, complemented by effects like tremolo, stereo-delay, reverb, and distortion. A distinctive feature is its generative algorithm that modulates parameters such as oscillator and effect mix levels, filter cutoffs, and more, enriching the sonic texture with evolving timbres. The implementation explores the synergy between traditional synthesis and modern web technologies, presenting challenges and solutions in deploying an audio tool online. This project exemplifies the potential of RNBO for innovative audio software development.

1. INTRODUCTION

Released by Cycling ’74 in 2022, RNBO extends the power of Max by facilitating the export of created patches to various platforms, including JavaScript (JS), C++, Raspberry Pi, and more [1].

This project exploits the versatility of RNBO to transform a polyphonic synthesizer patch, originally developed in Max, into a dynamic web application. It features a comprehensive browser-based graphical interface designed to manipulate the synthesizer's parameters. This interface enhances user interaction by providing real-time visual feedback of the created sound through an oscilloscope and spectrogram.

Users can interact with the synthesizer through a uniquely mapped virtual keyboard using the center and top rows of their computer keyboard or via a MIDI device connected through USB. These two input sources ensure that the synthesizer is accessible to casual visitors and well-equipped musicians alike. Moreover, the incorporation of a generative algorithm, which employs a simplified version of Brownian motion known as a random walk, adds an autonomous sonic dimension. This algorithm was specifically designed to focus not on generating melody or rhythm—as is common in many AI music applications—but on evolving the timbre and texture of the synthesizer's sound itself. This allows users to explore and discover new sonic landscapes without manually adjusting parameters, offering a unique blend of manual play and automated sound generation. As AI and music continue to intersect, this project provides a novel approach by allowing the sound of the synthesizer to be generatively altered while users play their desired notes or melodies through MIDI. The continuous evolution of sound introduced by this algorithm can be customized through user-controlled settings, allowing musicians to either subtly enhance their sounds or radically transform them, catering to a wide range of sound design preferences.

The core of the web-based synthesizer is powered by JS code exported from RNBO. Available at https://nusynth.com, the synthesizer hosts three oscillators—offering sine, square, triangle waves, or noise—ADSR envelope control, FM synthesis, two filters, and an LFO. Effects like tremolo, stereo delay, reverb, and distortion are also implemented to allow the user to enrich the sound quality and texture.

This project not only demonstrates the potential of RNBO in extending Max's reach into web-based applications but also provides a versatile and accessible tool for musicians and sound designers to explore and create music directly from their browsers. By combining traditional sound synthesis with innovative web technology, the project sets a new benchmark in musical interaction, accessible to a global audience without the need for specialized hardware or software. The inclusion of a generative algorithm also showcases just how much Max patch creations can be built upon now that they can be exported to various programming environments.

2. METHODOLOGY

Building NUSynth involved three distinct phases. (1) The polyphonic synthesizer patch was designed within the RNBO patching environment. This included building all the individual patches that would connect to create the final synthesizer. (2) The completed RNBO patch was exported to JS, and the front-end synthesizer interface was implemented. This interface enabled parameter control, audio visualization, and both virtual and MIDI keyboard input. (3) The generative algorithm was implemented in JS.

2.1 RNBO Patch Development

The development of the NUSynth patch was done in the Max software by Cycling ‘74. An additional toolchain, RNBO, was required to extend Max's capabilities and was purchased separately from the base software. RNBO introduces elements to Max that were used in the creation of

NUSynth, particularly the rnbo~ object, and platform export features.

2.1.1 Essential Synth Components

The rnbo~ object functions as a container within which the entire synthesizer is built, like a p (sub patch) object, encompassing all elements that are later exported from Max to JS. Inside the rnbo~ object, the various synthesizer components were implemented in sub-patches. The rnbo~ object automatically exports all defined parameters from its first inlet, allowing the patch creator to test its functionality by exporting it. An additional inlet was added, notein, to allow inputting MIDI notes to control the synthesizer.

The first sub-patch implemented was the oscillator. This object contains three tone generators, cycle~, saw~, rect~, and noise~. A selector~ object controlled by a parameter, "waveform," was used to limit the output of the oscillator to the selected waveform. A parameter, "level," was used to control the amplitude of the generated wave. This patch was duplicated a total of three times, and the output of all oscillators was combined using additive synthesis.

The second sub-patch implemented was the FM synth. This object contains a "frequency" param controlling a cycle~ object, and an "index" param multiplied with it. The output of this multiplication is added with the input MIDI frequency, thereby modulating it, and this new signal is run through a cycle~ object. A selector~ object with a param was implemented to turn the FM synth on or off. The output of the FM synth was combined with the outputs of all the oscillator sub-patches via additive synthesis.

The third sub-patch implemented was the filter. This object utilizes the svf~ object, and "type," "frequency," and "q" params to control it. The svf~ object can be of type low-pass, high-pass, bandpass, or notch filter, and a selector~ object was used to route the selected filter type to the output of the object. A "none" option was also included in the "type" param to allow turning the filter on and off. This object was duplicated two times, and the output of the previous objects was run through the two filters in series.

The fourth sub-patch implemented was the ADSR. This object simply used an adsr~ object with four params to control each adsr~ param respectively. The output of filters was multiplied by the output of the adsr~ object thereby attenuating the signal to match the params inputted into the adsr~ object.

The fifth sub-patch implemented was the LFO. This object utilizes a cycle~ object, a "destination" param, and a "frequency" param (0 Hz - 20 Hz). The "destination" param was used to route the LFO to the proper object utilizing a selector~ object, and the "frequency" param controlled the frequency of a cycle~ object which was outputted from a given outlet of the object. This object was duplicated twice.

2.1.2 Effects

The sixth sub-patch implemented was a stereo delay. This object utilized a delay~ object with feedback~ to create a delay with a feedback loop. This process was done twice, with the second delay being delayed extra by a given number of ms controlled by the "stereo-delay" param. The output of each delay was sent out of the left and right outlets respectively, thereby creating a stereo sound from a mono sound source. A "wet" param was introduced to control the mix of the delayed sound with the original.

Three more sub-patches were included in the rnbo~ object, overdrive, reverb, and tremolo. These patches were copied from the RNBO Guitar Pedals package by Manuel Poletti at Cycling '74. The output of the stereo delay was run through each of these patches in series. These patches came prebuilt with parameters to control the sound and mix of the effects. The outputs of the final effect were routed to the left and right outlet of the rnbo~ object.

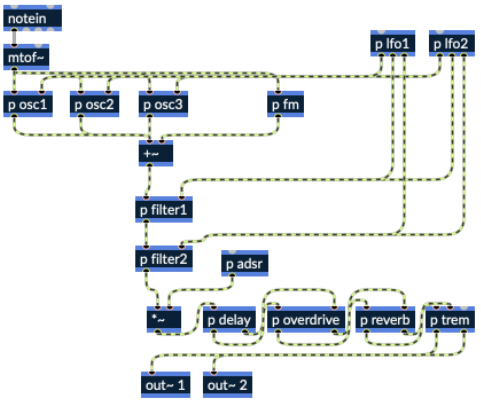

Figure 1. Sub-patches and connection inside the rnbo~ object.

2.1.3 Outside of the rnbo~ Object

The development of the patch did not only occur inside the rnbo~ object. To test the synthesizer implemented in the rnbo~ object, each parameter exported by rnbo~ had to have a dropdown (attrui) object to control it. A kslider and random note generator were also created to test the synthesizer parameters without having to play notes manually.

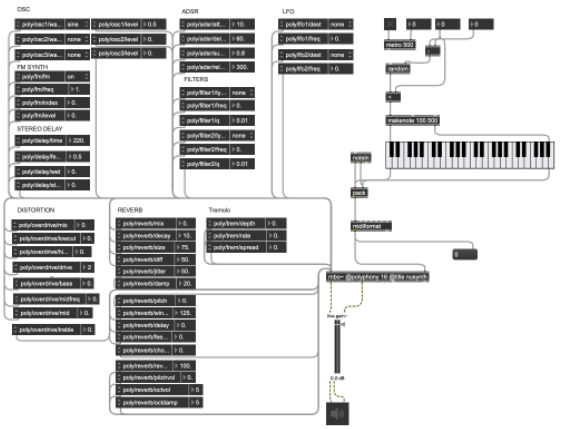

Figure 2. External test environment with exported rnbo~ parameters, kslider, random note generator, and ezdac~.

2.1.4 Exporting the RNBO Patch

Once essential elements of the synthesizer were completed (all objects minus effects), the focus shifted to exporting the patch to JS to begin development on the browser interface. RNBO provides an intuitive interface for exporting the patch, as seen in Figure 3. The RNBO export process creates a JSON file that contains details about the patch implemented in the rnbo~ object.

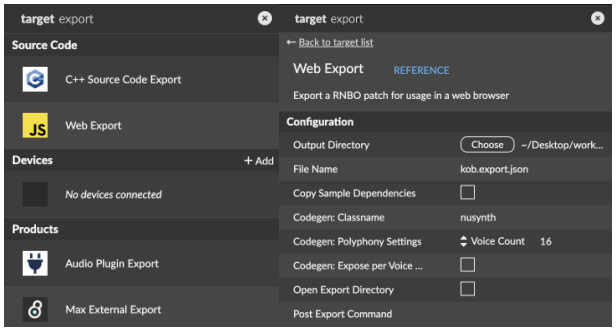

Figure 3. RNBO target export screens.

2.2 NUSynth Web Development

The NUSynth website was developed in Svelte, a concise, and performant JS UI framework. The SvelteKit framework was used to provide build tools and other useful features. Initially, the "RNBO Webpage Template" provided by Cycling '74 was used to get the Web Audio API and the "@rnbo/js" package set-up[1]. This template provided a foundation to test the exported RNBO patch in the browser, by providing basic MIDI input of a few notes by button click and control over all exported parameters from the patch as seen in Figure 4.

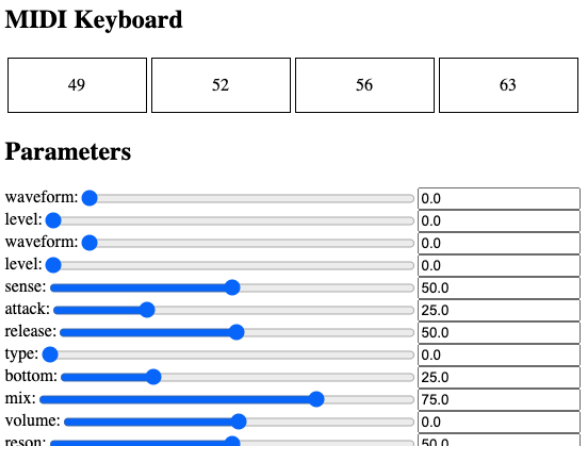

Figure 4. Page created by NUSynth when using RNBO Webpage Template (not all parameters are shown).

Once the parameters and sound were working with the NUSynth patch on the webpage template, front-end development began on the Svelte site. External MIDI input was set up using the "webmidi" package, allowing the user to use a MIDI keyboard to control the synthesizer. Custom code was written to allow the use of the computer keyboard to control the synthesizer. The NexusUI component library was used to create the various buttons, dials, number and text boxes, sliders, and oscillator and spectrogram visualizations. To connect the NexusUI control surfaces and the parameter in the NUSynth RNBO patch, a specific process had to be followed for each parameter:

1. The parameter was fetched using the "@rnbo/js" library via the `device.parametersById.get` function. This returned a JS object representing the parameter in the synth patch.
2. An "on change" event was created for the respective NexusUI control surface to update the patch parameter value whenever the control surface was touched in the UI.

Once this was completed for all the parameters, the main basic front-end functionality of the synthesizer was essentially complete. All parameters in the RNBO patch could be controlled through the browser using MIDI or a computer keyboard. Styling was applied to make the website more appealing and give it its unique look. Styling was done using Tailwind CSS and standalone CSS.

Figure 5. The final version of the NUSynth website.

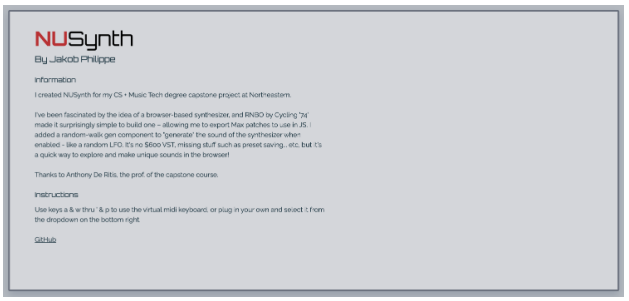

Figure 6. NUSynth information page.

The domain nusynth.com was purchased and a server was set up to host the website, ensuring that it remains accessible to the public into the future. To deploy the site to the server, the SvelteKit app was built and containerized using

[1] See the "RNBO Webpage Template" on GitHub https://github.com/Cycling74/rnbo.example.webpage.

docker and deployed to the server with a Caddy web server reverse proxy.

2.3 Generative Parameter Control

The generative parameter control implemented in NUSynth introduces a dynamic element to sound synthesis by automating the modulation of various synthesizer parameters. This functionality utilizes a random walk algorithm to achieve a continuous, yet controlled, alteration of parameter values, adding a unique dimension to the sonic output of the synthesizer.

2.3.1 Random Walk Algorithm

At the heart of the generative parameter control is the `genParamVal` function, which models a simplified Brownian motion, or random walk. The function is designed to generate a new parameter value based on the previous one, incorporating both a directional drift and random volatility to simulate natural variations. The key components of this function include:

- Drift Direction: Initially set to induce an upward drift in the parameter values, the drift direction can randomly switch based on a predefined probability (`driftSwitchChance`). This introduces unpredictability in the parameter evolution, making the sound generation less deterministic and more organic.
- Drift and Volatility: The parameter values drift at a fixed fraction of the maximum range of the parameter. Additionally, the function adds random volatility around the current value to ensure that the new parameter values are not just linearly incrementing or decrementing, but also reflecting subtle (or large, if volatility is set high) fluctuations.
- Value Clamping and Scaling: To ensure the generated values remain within the allowable range of the synthesizer's parameters (min and max), the output of the random walk is clamped. This is to prevent parameter values from exceeding their logical limits.

2.3.2 Initialization and Continuous Update

The `initGenParam` function leverages the `genParamVal` to set and continuously update the value of a specific parameter. It initiates a perpetual loop that periodically updates the parameter value, simulating a live and autonomous modulation. This process involves:

- Parameter and Dial Association: Each parameter is linked to a corresponding dial on the user interface, allowing real-time visual feedback and user interaction.
- Scheduled Updates: The parameter values are updated at regular, short intervals (`randomTimeout`), maintaining a lively and evolving sound texture.

2.3.3 Targeted Parameter Control

The `startGen` function initializes the generative control by selectively applying it to key parameters such as oscillator waveforms, effects mixes, and more. By focusing on these parameters, the algorithm efficiently contributes to the overall timbre and texture of the output sound without overwhelming the system.

3. DISCUSSION

3.1 Challenges and Limitations

Developing NUSynth posed several challenges, primarily rooted in the integration of RNBO with web technologies. The initial difficulty was producing the first audible output from the synthesizer within the browser environment. This phase had non-obvious bugs that took experimentation to fix, complicating the troubleshooting process. The absence of error messages in the browser about what was going on in the RNBO patch made it challenging to pinpoint the exact issues, requiring a trial-and-error approach to identify and resolve the underlying problems.

Additionally, transitioning from Max to RNBO introduced its own set of hurdles. In Max, the mc object is frequently utilized for managing multiple channels of audio and is present in many forum-posted patches and tutorials. However, RNBO does not support the mc object, which necessitated finding alternative methods to achieve similar functionalities within the RNBO environment. This limitation required a rethinking of how to implement patches.

Styling the browser interface also presented difficulties. Achieving a user-friendly and aesthetically pleasing design involved numerous iterations. The objective was to create an intuitive and engaging interface that could effectively control all the parameters of the NUSynth patch, which presented a challenge for someone averse to design.

Lastly, the generative algorithm posed problems as the quick shifting of parameters caused lots of clicking. These clicks were removed by using line~ objects to create linear ramps instead of instantaneous changes, thereby stopping clicking.

3.2 Project Iterations

During the developmental phase of NUSynth, several ideas were considered to enhance the interactivity and functionality of the synthesizer in the browser. One of the initial concepts explored was the integration of mixed reality technology, specifically using the Meta Quest headset. The idea was to allow users to manipulate synthesizer parameters through gestures and movements within a virtual reality environment. However, this idea was eventually set aside. While intriguing, it posed significant accessibility barriers, as it required users to own a specific, costly headset to fully experience the synthesizer's capabilities. This requirement would counteract the fundamental goal that became apparent as NUSynth was being built,

which is to provide a universally accessible web-based synth.

Another idea involved the use of camera inputs and AI models to translate user gestures or body movements into synthesizer controls. This approach would harness computer vision technologies to create an interactive user experience. However, upon further research, a project was found with this feature already implemented in an open-source project fork of the "RNBO Webpage Template" discussed in Section 2.2. Since one of the goals of this project was to introduce something new in the realm of browser synthesizers using RNBO, this idea was also set aside.

Ultimately, the selection of generative parameter control for NUSynth marked a significant departure from these earlier concepts, aligning with the overarching objective of broad accessibility and experimentation within the RNBO framework. This decision capitalized on the potential to use JS to automate complex sound modulation processes. By integrating a generative algorithm that dynamically alters synthesizer parameters, NUSynth enhances the user experience by offering an evolving soundscape when enabled, which can be used to help discover new sounds or evolve a performance over time.

3.3 Future Work

Looking ahead, there are several enhancements planned for NUSynth that aim to enrich the user experience and expand technical capabilities. One immediate area of development is the enhancement of the front end to support presets and the ability to save the synthesizer state between page reloads. This feature would allow users to save their settings and return to their sound explorations without starting fresh each time, enhancing the practical usability of NUSynth in musical composition and performance.

Further expansion of the synthesizer patch is also envisioned. By incorporating more features and effects, particularly increasing the destinations available for LFO modulation, the sonic possibilities of NUSynth can be significantly broadened. This expansion would cater to a wider range of sound design ideas.

Another future development involves the implementation of a more sophisticated generative function, possibly leveraging artificial intelligence. The concept of training AI models on synthesizer parameters—rather than traditional focuses like melody or rhythm—presents a novel approach in the realm of generative music. This method could lead to dynamic, AI-driven modulation of sound parameters, creating a continuously evolving auditory landscape that responds intelligently to user interactions or autonomous generative algorithms. These advancements aim to push the boundaries of what web-based music synthesis tools can achieve, making the sophisticated sound design more accessible and stimulating further innovation in the field.

4. CONCLUSION

The development of NUSynth illustrates the practical application of the RNBO framework in creating a browser-based polyphonic synthesizer from a Max patch. This project showcases the integration of traditional sound synthesis components—such as oscillators, envelopes, and filters—with modern web technologies to provide a robust and accessible musical tool. Key features of NUSynth include a comprehensive interface that facilitates real-time sound manipulation and visualization, enhancing the interactive experience for users.

A distinctive aspect of NUSynth is its generative parameter control, which employs a random walk algorithm to autonomously modulate sound parameters. This feature enriches the synthesizer's output with evolving sonic textures, offering users a unique blend of manual and automated sound generation. Despite its capabilities, NUSynth remains a proof of concept that demonstrates the potential of web-based music tools rather than a revolutionary breakthrough in digital music technology.

Challenges encountered during the development included integrating RNBO with JavaScript and the web audio API, as well as ensuring cross-platform functionality. These were addressed through iterative testing and development, highlighting the adaptability of RNBO for web applications.

Future enhancements for NUSynth are planned to include the implementation of user-defined presets and state-saving features to improve usability. Additionally, the expansion of the synthesizer's patch to include more sound modulation options could further broaden its appeal and functionality.

Overall, NUSynth serves as a valuable capstone project that bridges the gap between music technology and web development, reflecting the evolving landscape of digital music tools and their potential for broader accessibility and creative expression.

REFERENCES

[1] P. Kirn, “RNBO (‘rainbow’): Start in Max’s UI, Deliver to Plug-ins, Web browsers, Hardware, any OS,” CDM Create Digital Music, Nov. 01, 2022. https://cdm.link/2022/11/rnbo-max-for-web-hardware-plugin/ (accessed Apr. 22, 2024).