Web Integration of a granular synthesizer using RNBO in Max/MSP

Project Report

Sarunas Kilius

Aalborg University

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.

Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY

STUDENT REPORT

Title:
Web Integration of a granular synthesizer using RNBO in Max/MSP

Project Period:
Spring Semester 2023

Participant(s):
Sarunas Kilius

Supervisor(s):
Daniel Overholt

Copies: 1

Page Numbers: 27

Date of Completion:
May 25, 2023

Abstract:

Recent advancements in web technologies continuously makes it more popular to build web-based alternatives to local applications. Similarly, audio software is increasingly more prevalent as a website and is able to run at native speeds due to developments like WebAssembly. This paper explores how granular synthesis can be implemented on a website using a Max/MSP compatible tool RNBO. The pipeline from a graphical RNBO code to a functional website with granular sound synthesis is described. The pipeline includes analysis of suitable granular synthesis parameters; working with HTML, CSS and JavaScript to build a website; processing audio inside JavaScript to control audio input, output and synthesizer parameters; customising audio samples and updating the user interface. In order to evaluate the software, user testing was performed on usability of the website, performance and functionality of the synthesizer.

Contents

Chapter 1

Introduction

The theory of granular synthesis dates back to mid 20th century where a physicist Dennis Gabor proposed that granular approach could be used to describe any sound [10]. In 1971, Iannis Xenakis was able to analyse the composition for grains of sound, where he suggested an implementation of Gabor's model as an analog synthesizer[15]. A few years later a Swedish composer and researcher Curtis Roads implemented granular synthesis with the use of digital computers. The definition for granular synthesis has been established as a process of separating an audio sample into small segments, grains, and altering them individually to create new soundscapes. The digitisation of music production allowed for granular synthesis techniques to become more accessible and recognised as a means of music production or sound design. It remains a unique method for time stretching, pitch shifting, grain position modulation or spatialization of audio samples and is widely used in both popular and experimental music.

While there are numerous standalone software applications and VST plug-ins that perform granular synthesis and processing with an easily understandable user interface, examples of such devices implemented on a website are limited to a handful. In comparison to web based software, native applications and plug-ins are more time and energy efficient and effective for professional music producers and sound designers with a more advanced workflow, however, it requires access to specific software as well as compatibility with the digital environment they are being used in.

Websites on the other hand are accessible by anyone with an internet connection and do not require additional installation or updates. Web hosted audio software does not rely on powerful local hardware and the processing is performed remotely on a server, offering the almost the same experience regardless of device specifications. Web integration of existing audio software could also provide a playground to easily test the capabilities of the application before purchasing and installing it

locally. Web based sound processing usually requires more extensive knowledge of programming in several coding languages to be able to integrate the software in a website the server for hosting the website has to be powerful enough to handle complex audio processing. However, with machine learning being implemented in JavaScript libraries and servers becoming more powerful, real-time audio processing on the micro scale is more easily achievable on web based platforms. New frameworks allow developers and music producers to implement their software as a website without needing an extensive programming knowledge.

1.1 Audio Processing in Web

Audio web integration can be implemented via a few different approaches. The most straightforward one is using HTML audio tag and controlling the sound file with JavaScript. This however offers limited audio processing capabilities and control over the playback. On the other hand, JavaScript tools like Web Audio API are able to perform advanced audio manipulation in real-time and provide features like mixing, routing, filtering or effects. As part of Web Audio API, the AudioWorklet interface allows to process the sound on o a separate thread on the web, and therefore perform with very low latency. Newer developments push the capabilities even further. Web Assembly is a binary instruction format that is able to run code written in C++, Rust and other languages once it is compiled in Web Assembly. Web Assembly offers code execution at native speed and is able to accelerate computationally intensive audio processing tasks. Web Assembly can be used in parallel with Web Audio API or AudioWorklet to increase performance and handle complex audio processing tasks.

Various different coding environments make use of these audio processing approaches and have possibilities to export the code for web deployment.

1.1.1 Faust

Faust is a programming language designed for real-time audio signal processing and synthesis[9]. The programs compiled in Faust are comparable to handwritten C/C++ while simultaneously being simpler and more intuitive to learn. Faust combines functional programming with algebraic block diagrams to compose functions and is able to translate the code into various different languages and therefore provide the possibility to compile for targets like audio plug-ins, standalone applications, mobile or web applications. Web integration is implemented by allowing to export Faust audio code as JavaScript and WebAssembly modules.

1.1.2 Max/MSP and RNBO

Max/MSP is a visual programming language made for music and multimedia development[14]. It's graphical interface is used to build audio, video and interactive systems by connecting pre-built objects. The coding is done in a graphical interface, where the user is able to select object from the object library and add object attributes and messages. RNBO is a Max compatible library that allows to export Max-like patches into several different targets like Rasberry Pi, Web, VST[4]. RNBO allows to compile the code as Web Assembly and this way integrate it into a website using JavaScript. HTML and JavaScript frameworks can be included to introduce advanced user interface or visual elements into the website.

1.1.3 Pure Data and WebPD

Pure Data is an open-source programming language for audio and multimedia development[8]. It shares a similar graphical interface to Max/MSP and provides a visual and interactive coding experience. A collaborative project called WebPd introduces a possibility to export a Pure Data patch into JavaScript suitable for web integration[12]. WebPd uses Web Audio API for audio processing on web, which in comparison to Web Assembly is less computationally efficient and optimised for advanced sound processing.

1.1.4 Gibber

Gibber[11] is a live coding environment specifically designed for audio and visual performances. The environment offers a variety of different sound synthesis techniques, audio effects and sequencing options. Gibber is build on JavaScript, primarily based around the audioLib.js library and uses no additional syntax, therefore can easily be implemented in websites to create real-time audiovisual processing.

1.1.5 Web Audio Modules 2.0

Web Audio Modules (WAMs) [1] is an open source framework aimed at creating a Web Audio plugin standard. WAMs provide a way to develop and deploy audio plugins that can be used in web based audio software by utilising Web Audio API, AudioWorklet and WebAseembly and enabling real-time audio processing on web. WAMs support implementation of plugins written in JavaScript, C/C++, FAUST or CSound, and therefore are compatible with most major audio development frameworks. Integrations like FAUST, JSPather, Sequancer.Party or Amped Studio allow to quickly build custom plugins, collaborate with other producers or include WAMs in a web-based DAW.

1.2 State of the Art

1.2.1 Grain Train

GrainTrain is a web based granular synthesiser created for real-time performance on both desktop and mobile platforms [2]. The main selling point of the software is its unique user interface, where the user is able to draw waveform paths and control grains using a multi-touch interface. The paper describes analysis of 20 different granular synthesisers in terms of their user interfaces and found three main interaction paradigms: parameter control, keyboard performance, and waveform scrubbing. GrainTrain focuses on custom waveforms, where the user is able to draw in a curve of a custom shape and length and the audio sample will be stretched to facilitate the drawn-in curve. This allows for a creative and interactive way to manipulate the audio samples by scrubbing through the paths using the touchscreen. This technique along with the multi-touch technology creates possibilities for unique performance techniques such as: superimposition of multiple waveforms, discontinuous scrubbing, ergonomic short-touch interactions or intra-mixing. The system was designed using WebAudio API, the WebGL library Three.js, HTML5 and CSS. While it was designed as a web application, internet connection is only required while the application is loading. The user is able to use locally stored audio samples without the need to upload it to any server.

1.2.2 FaceArp

Face Arp is a web based synthesiser created by Maciek Odrowaz utilising head tracking to control the parameters of the synth [3]. It uses a machine learning solution called MediaPipe Face Mesh, which takes a single 2D camera input and estimates the 3D surface of a face. Head pan, tilt and roll as well as the position of the mouth is calculated and used to control pan, note probability, cutoff frequency and reverb of the synthesiser respectively. The actual synthesiser is created using Max 8 and RNBO to make it usable as a Web application and the Face Mesh solution is similarly integrated using its native Javascript API.

1.2.3 HTML5 Granular Synthesiser

Created by Ehsan Ziya and Chris Smith, Multi-Touch Granular Sampler 1.0 is a web based synthesiser that uses preloaded samples to perform granular effects [13]. The synthesiser works by scrubbing through the audio file with a cursor and adjusting the parameters like attack, release, density, spread, pan and transposition. While there is an option to upload custom sound files, it does not allow for recording and exporting the performance. The device has a visual element showing the waveform of the uploaded audio and the position of the grain when scrubbing through the

sample. The user interface and audio processing is created using Bootstrap 3, jQuery Knob, Web Audio API and Procssing.js libraries.

1.2.4 Patatap

Patatap is an interactive audio-visual playground with audio reactive animations triggered by keyboard buttons or screen taps [7]. On a desktop version, every key is assigned a sound and an animation corresponding to the sound, while the mobile app has this information assigned to different locations on the screen. The instrument is also able to take midi input from external software or hardware to be used for the audio visual performances. The creators Jono Brandel and Lullatone aim to explore Visual Music and how sonic stimuli can influence visual stimuli and vice-versa. The sounds and visuals were created by a team of music composers, sound designers and animators. Unfortunately, the technologies used to create the interface and process the sounds and visuals on the web were not disclosed.

1.2.5 Learning Synths

Learning synths is an interactive website created by Ableton for learning the workings of a synthesiser [5]. The website showcases a subtractive synthesiser with several oscillator options, and envelope, LFO, pitch and filter parameters. The synthesiser is built as a Max patch and is exported as a web application with the help of RNBO API. RNBO uses Web Assembly, a machine language optimised for the browser, and Audio Worklets, sound processing technology to be able to smoothly play the synthesiser on a website. RNBO is able to automatically convert the Max patch into a Web Assembly code and run it as a website.

1.3 Target Group

A web based granular synthesizer has potential to appeal to a wide user base. The synthesizer being hosted on a website would mean it is easily accessible regardless of hardware or operating system, while at the same time needing no installation or special equipment to use. Musicians and sound designers can make use of the synthesizer to quickly create sound effects, atmospheric textures or rhythmic elements when not having access to their usual audio software applications. Beginners and people interested to learn more about sound synthesis or explore the granular manipulation of sounds can access the site and explore the capabilities of granular synthesis.

1.4 Final Problem Statement

Based on technological research, a problem statement was formulated.

How can a web based granular tool be implemented to explore granular synthesis in real time?

1.5 Requirements

The technological analysis allowed to determine the requirements for the software. The requirements are split into two parts. Functional requirements dictate the requirements that are integral part of the design and operation of the software. Non-functional requirements represent the ones that are not critical and are more about improving the already functioning software.

1.5.1 Functional Design Requirement

- The software should utilise the techniques of granular synthesis as a basis for sound processing.
- The software should be available as a website for easy access by anyone.
- The software should provide several audio samples for the user to choose from and perform granular processing.

1.5.2 Non-functional Design Requirements

- The software should have an easy to use and visually pleasing user interface.
- The software should have a visual element corresponding to the sound or synth parameters.
- The software should have an engaging way to interact and control the synthesizer.

Chapter 2

Design

Taking into account the design requirements mentioned in the previous chapter, some considerations regarding the software were made. Several iterations of the software were developed and presented for feedback to university peers involved in music production, this way gathering information needed in order to improve the software's functionality and presentation. First, general decisions about the development process were made and the first prototype was created.

2.1 Granular synthesis

As the synthesizer is expected to process sound using granular synthesis technique, a closer look into different parameters was needed. Generally, the main parameters involved are grain size, grain position, grain envelope, grain density and pitch.

- **Grain size** determines the length of each grain, i.e. short audio fragment in the sample. Smaller grains will generate shorter and more transient sounds, while larger grains will result in more sustained sounds.
- **Grain position** determines the starting point of each grain and its modulation results in timbral and rhythmic changes of the sound.
- **Grain density** is the grain frequency and determines how textured and dense or sparse and spaced-out the sound is.
- **Grain envelope** dictates the shape of the volume envelope for each grain. These can typically be linear, exponential or logarithmic and ultimately controls how sharp or smooth eachi grain is.
- **Pitch** of the grain can control tonal characteristics of the sound and be used to create melodic and harmonic sounds. Pitch is determined by the playback speed of the sound.

These characteristics were kept in mind when developing the synthesizer and its control interface.

2.2 Coding Environment

Max/MSP coding environment was chosen for the development of the synthesizer due to its graphical and beginner friendly interface as well as good documentation regarding granular synthesis. Max uses node infrastructure to build patches by utilising pre-existing objects, messages and parameters. Every object is thoroughly documented and examples of their uses are available within the software. RNBO library integration allows for easy export to multiple targets, including web deployment. While the coding for RNBO is being done in a completely separate instance from the Max main building environment, the objects, messages and parameters are very similar for both architectures. Web export target in RNBO translates the graphical code into WebAssembly, which is then included in a website using JavaScript. RNBO provides a website template, with pre-written HTML, CSS and JavaScript files, which are ready to be deployed as a website. However, the template website is visually unattractive and computationally inefficient as it uses the standard HTML UI and contains extra elements in order to facilitate any patch. The files will have to be edited to eliminate

2.3 Audio Samples

In order to perform audio processing, several sound samples had to be included. It was decided to choose a variety of audio samples differing in tonal, rhythmic and timbral characteristics. Royalty free sounds were chosen from MusicRadar online library. The selected sounds would be used to demonstrate the workings of the synthesizer, and how it is applicable for different sound design purposes.

2.4 User Interface

In order to make the website visually appealing, an additional library was chosen. The web template for RNBO provides a general HTML5 user interface, however, it is visually outdated and not engaging. It was decided to update the user interface with an additional CSS library to make it more engaging and visually interesting.

Chapter 3

Implementation

3.1 Max and RNBO

The first part in developing the synthesizer was creating the patch in Max coding environment using the RNBO architecture. The coding was done using the software's graphical interface - by loading pre-built objects, adding variables and specifying parameters.

The first iteration of the synthesizer was a simple max patch, which allows to input a frequency and outputs the signal of that frequency.

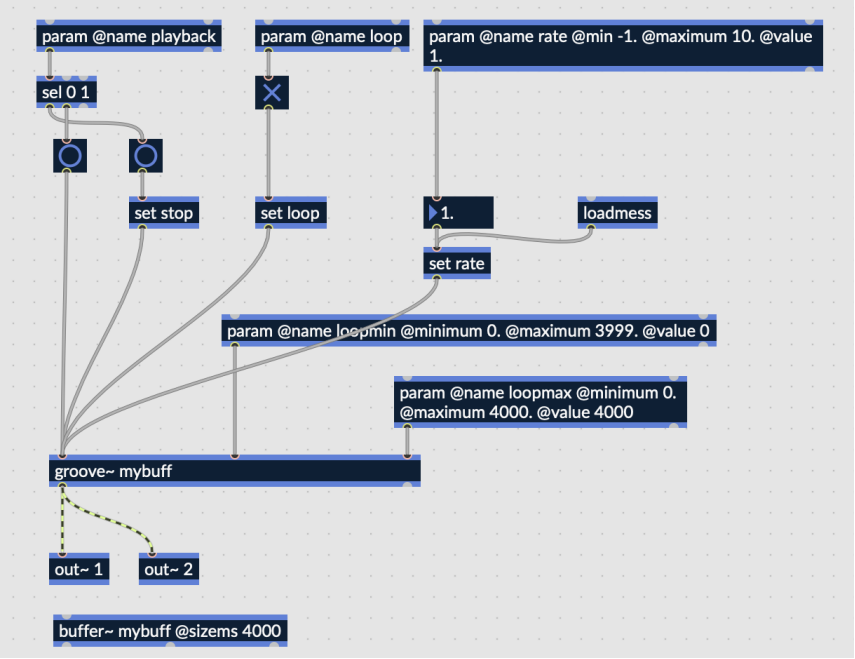

Figure 3.1: First iteration of the synthesizer in RNBO

The first iteration of the synthesizer was created using the groove~ object, which is able to play, loop and perform pitch transformations on an audio sample stored in buffer~ object. As seen in the figure 3.1, the groove~ object references *mybuff* which is the name of the buffer~ object defined below. First parameter of the synthesizer is its playback and it is created using toggle buttons to variate between play and stop. Other parameters like looping, rate, the start of the loop and the end of the loop were defined. The name, the minimum and maximum values and the initial starting value were defined inside the RNBO environment and these values showed up on the website once the patch is exported and included in the code. The input audio sample was linked to the buffer element in JavaScript and the stereo audio output is ensured using the out~ 1 and 2 objects.

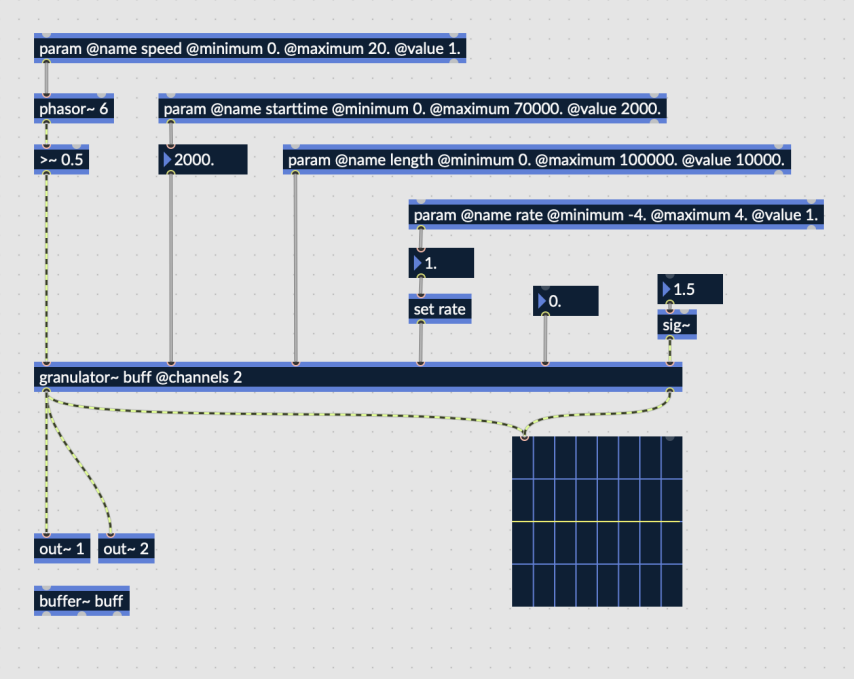

Figure 3.2: Granular synthesizer in RNBO

While the first iteration synthesizer was able to perform similar functions to ones of a granular synthesizer, it did not contain all the functionality of a granular synthesizer and therefore another RNBO patch was created as seen in figure 3.2. While similar in nature, the new patch also included the speed parameter which controls how many grains per second are played, and also used the granulator~ object instead of the groove~ object. This means that object offers audio processing on a granular level, and features like grain overlap and grain envelope are considered. The initial values for speed and rate were chosen as 1 in order to play the

audio sample at its original speed and pitch when the sound is loaded. This allows the user to understand what the sample sounds like originally before performing granular processing.

The graphical RNBO patch was then converted to WebAssembly using the Web Export target option and save as a JSON file locally.

3.2 Web Integration

3.2.1 Generating HTML code

The first step to creating the website was generating the basic structure in HTML. The file index.html would be the home page of the website and included elements like the title of the website, the header text as the name for the synthesizer, any additional text to describe the synthesizer, the audio start and stop button and parameter sliders for the RNBO patch. The HTML file also contained paths to the CSS file for styling the website and JavaScript files for including the RNBO patch and any additional functionality in the website.

```
<!-- Create header -->
<div id="tit" class="w3-padding-32 w3-circle w3-center">
  <h1 id="patcher-title">RNBO Web granulator</h1>
</div>
```

Figure 3.3: HTML code for including the header text

3.2.2 Including the RNBO patch using JavaScript

In order to actually include the audio processing, a JavaScript (JS) application was created. First of all, AudioContext - a Web Audio API interface, where all audio processing is happening - was initiated. AduioContext was responsible not only for sound processing but also outputting audio on a website. Stopping and resuming the AudioContext will do the same to the audio output.

```
// Create AudioContext
const WAContext = window.AudioContext || window.webkitAudioContext;
const context = new WAContext();
```

Figure 3.4: JS code for creating AudioContext

RNBO patch was included in the website by first fetching the patch by specifying the path to the patch and then loading the RNBO script. This allowed to then create the device, which would act as the RNBO patch.

```
// Create the device
let device;
try {
  device = await RNBO.createDevice({ context, patcher });
} catch (err) {
  if (typeof guardrails === "function") {
    guardrails({ error: err });
  } else {
    throw err;
  }
  return;
}
```

Figure 3.5: Create the RNBO device in JS

In order to obtain information from the patch and control the sound using the parameter sliders, the device was accessed in JavaScript. For each of the device's parameters, label, slider and text input was created. The name, minimum, maximum and default values for each parameter were accessed from the device and assigned to the corresponding slider. The device was constantly being updated by including the function to listen to parameter changes and this way perform sound processing in real-time. Some code snippets are included in figures 3.6 and 3.7.

```
// Make each slider reflect its parameter
slider.setAttribute("type", "range");
slider.setAttribute("class", "param-slider");
slider.setAttribute("id", param.id);
slider.setAttribute("name", param.name);
slider.setAttribute("min", param.min);
slider.setAttribute("max", param.max);
```

Figure 3.6: Setting name and value for each slider

```
// Listen to parameter changes from the device
device.parameterChangeEvent.subscribe((param) => {
  if (!isDraggingSlider) uiElements[param.id].slider.value = param.value;
  uiElements[param.id].text.value = param.value.toFixed(1);
});
```

Figure 3.7: Listening to parameter changes and updating the device

3.2.3 Controlling Audio Output

An option to control the sound output was added. In the HTML file, a button element was created and then with the help of JavaScript altered to change its value from 'Start Audio' to 'Stop Audio' once clicked. To connect it to audio output, AudioContext was initiated and stopped as the button was clicked. Figure 3.8 shows how the button element is accessed in JavaScript and how it is value is altered once it is clicked. The code checks the current value of the button and changes 'Start Audio' to 'Stop Audio' and vice versa. Similarly, AudioContext is being resumed it the value of the button is 'Start Audio' and stopped if the value is 'Stop Audio'.

```
// start audio with a button
context.suspend();
document.getElementById("start-button").onclick = (e) => {
  if (document.getElementById("start-button").innerHTML === "Start Audio") {
    context.resume();
    document.getElementById("start-button").innerHTML = "Stop Audio";
  } else {
    context.suspend();
    document.getElementById("start-button").innerHTML = "Start Audio";
  }
```

Figure 3.8: JS code for toggling the value of the button and the status of AudioContext once clicked

3.2.4 Including the Audio Samples

The samples to be used for further processing were downloaded from royalty-free sources and added to the domain. In order to enable sound selection, a drop-down menu was created using HTML. A list of all sound file paths was generated in the same order as it appears on the drop-down menu. Once the value of the drop-down menu changed, the corresponding file path was selected and that audio file was loaded into the device. The sample was connected to the audio buffer object in the RNBO patch by first loading the sample as an ArrayBuffer, then decoding it as an AudioBuffer and connecting it to the device as shown in figure 3.9.

3.2.5 UI changes

Finally, the user interface was updated by including a CSS library and changing the padding, color, font and and border characteristics to make the UI a little bit more interesting and cohesive. The initial and the updated user interfaces can be seen in figure 3.10

```js
// create a function to convert the audio sample to a buffer
async function samples(file) {
  const fileResponse = await fetch(file);
  const arrayBuf = await fileResponse.arrayBuffer();
  const audioBuf = await context.decodeAudioData(arrayBuf);
  await device.setDataBuffer(bufferId, audioBuf);
}

samples(tracks[0]);

// Connect the device to the web audio graph
device.node.connect(outputNode);

// get the value from the select menu and change the sound sample
var f = document.getElementById("select");
function onChange() {
  var val = f.value;
  // document.getElementById("soundtxt").innerHTML = val;
  samples(tracks[val - 1]);
}
```

Figure 3.9: JS code showing how as the value of the audio selection changes (*function onChange*), another function (*function samples*) is initiated where knowing the current value of the selection the corresponding audio sample is decoded and loaded into the device.

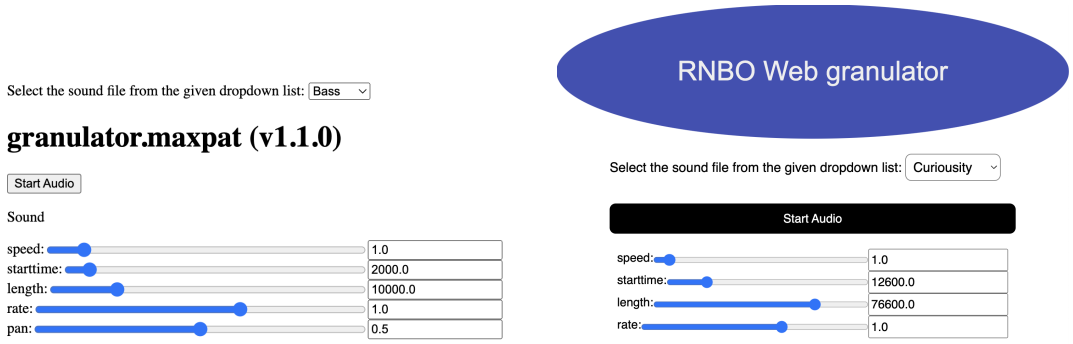

Figure 3.10: Old (left) and updated (right) User Interface

Chapter 4

Testing and Results

4.1 Usability Testing

In order to evaluate the software some user testing was performed. First, a usability study was conducted in order to determine how functional, efficient and user-friendly the software is. The study aimed to identify strengths, weaknesses and potential areas for improvement to help make the software as intuitive and straightforward as possible. The usability study was conducted only with 5 test participants, as suggested by the research from Jakob Nielsen, who claimed that 5 people are optimal enough to determine the usability issues, while keeping the evaluation costs low [6].

4.2 First Iteration

4.2.1 Testing

The survey was created using google forms due to its accessibility, ease of use and ability to generate charts from the results and export them as a spreadsheet. The first part of the survey contained the questions regarding the musical background of the test participant:

- Do you have any experience with music production or sound design?
- Have you used a virtual or a physical synthesizer before?
- If yes, have you used a synthesizer on a website?
- Do you know about granular synthesis and have you had any experience with it before?

The questions helped to understand how familiar the participants were with similar software in order to identify if the usability issues stem from being lack of experience with musical instruments or granular synthesis.

To actually test the usability of the website the following statements regarding user-friendliness, simplicity and ease of use were presented. A 5-point Likert scale from Strongly Disagree to Strongly Agree was presented.

- I think that I would like to use the website frequently.
- I found the synthesizer to be unnecessarily complex.
- I found the synthesizer was easy to use.
- I think that I would need assistance to be able to use this synthesizer.
- I thought the functions in the synthesizer were well integrated.
- I thought there was too much inconsistency in the synthesizer.
- I would imagine that most people would learn how to use the synthesizer very quickly.
- I found the synthesizer very cumbersome / awkward to use.
- I felt confident using the synthesizer.
- I needed to learn a lot of things before I could start using the synthesizer.

4.2.2 Results

The following results were obtained from the user usability questionnaire. The usability statements were presented using the Likert scale and therefore were easily quantified - Strongly Disagree resulting in value of 1 and Strongly Agree in value of 5. The numerical values for answers to statements that were phrased negatively were subtracted from 6. Assigning the numerical values to questions allowed to perform better analysis of the overall usability of the software by summing up the values for each. It is important to note that when talking about positively and negatively rated statements, they are considered from the usability perspective, e.g. a the rating for the negatively phrased statement will be considered positive if the answers were mostly disagreeing with the statement.

The questions which were answered most positively were: *I found the synthesizer to be unnecessarily complex* and *I found the synthesizer was easy to use.*

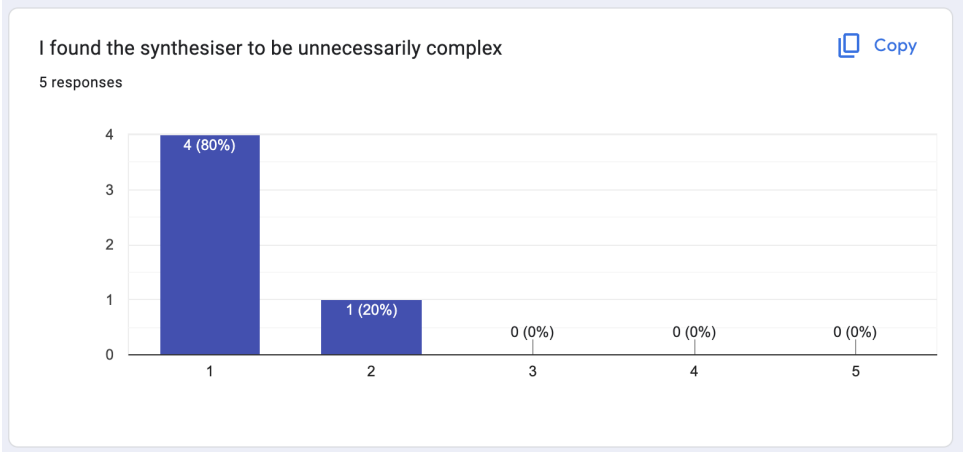

Figure 4.1: Most positively rated statement

In contrast, the questions answered most negatively were *I would like to use the website frequently*, *I needed to learn a lot of things before I could start using the synthesizer* and *I thought the functions in the synthesizer were well integrated*.

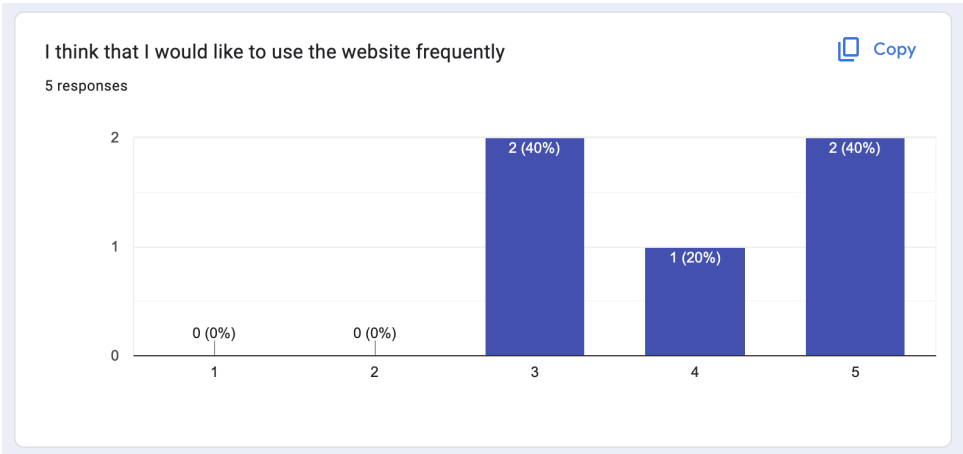

Figure 4.2: Most negatively rated statement

From the available data about musical background, there seemed to be little correlation between familiarity with music production and high ratings for usability. The participant with most negative usability rating had little to know experience with music production and granular synthesis, while the most positive rating came from a participant with almost the same musical background.

At the end of the questionnaire an open ended question to provide feedback was asked and all answers were positively reflected on the usability of the website,

while at the same time acknowledging that the simplicity of the website might be more suitable for beginners.

4.3 Second Iteration

In order to try and increase the usability score for the software, small changes were made to the website and the usability test was performed again. An additional info symbol was added to the website with a short description about granular synthesis and parameters of the synthesizer. To keep the simple and stripped back layout of the website, the information would only show up as the mouse hovers over the info symbol.

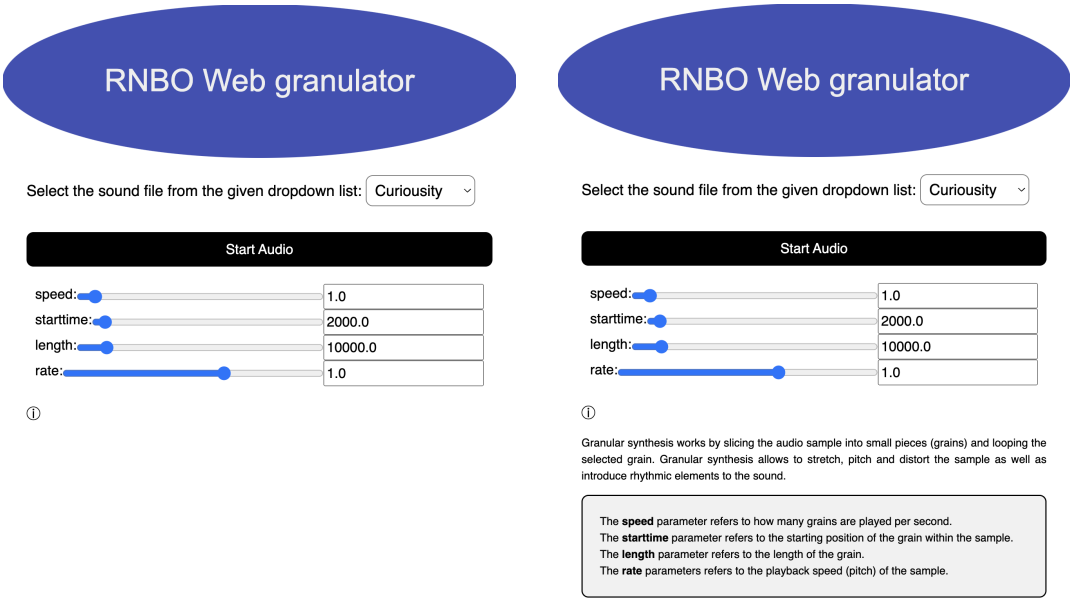

Figure 4.3: Updated website with the information about granular synthesis and parameters

4.3.1 Results

The results from the updated software were overall more positive in all aspects with biggest increase in statements *I think that I would need assistance to be able to use this synthesizer*, *I thought there was too much inconsistency in the synthesizer* and *I would imagine that most people would learn how to use the synthesizer very quickly*.

While the most negatively rated statements remained the same, the most positive statements were *I would imagine that most people would learn how to use the synthesizer very quickly* and *I thought there was too much inconsistency in the synthesizer*.

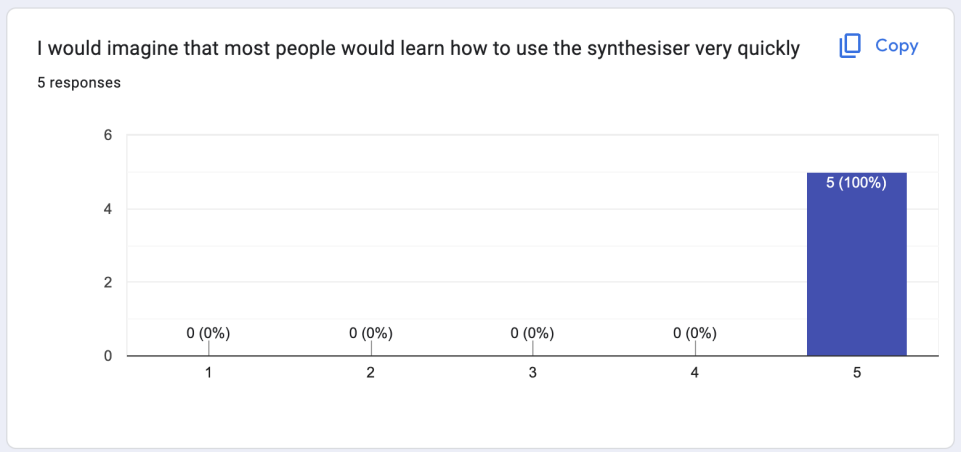

Figure 4.4: Most positively rated statement

Overall, it the addition of informational text in the website resulted in more positive ratings for the software usability, however, the difference was not significant enough to claim the changes were the determining factor.

4.4 Interviews with musicians

As a final evaluation practice, three people involved in music on a professional level were asked to use the website and answer questions regarding not only usability but also the sound quality, synthesizer capabilities, features and real-time performance.

Study participants were presented a similar questionnaire-like system, first asking about musical background and involvement with granular synthesis. All participants expressed that they are familiar and have used granular synthesis enough to understand the standard features and capabilities of a granular synthesizer. The participants' professional musical background and experience with digital synthesizers proved to provide valuable and constructive feedback about the software.

The participants were once again presented with statements and asked to choose between *Strongly Disagree* to *Strongly Agree*. The following statements were presented to the test participants:

- I thought the synthesizer was responsive and and worked well in real time without significant latency.
- I thought the sound was not clear and there was unwanted artefacts and distortion present.

- I thought the synthesizer was able to transition smoothly between the different parameter settings.
- I thought the synthesizer didn't provide adequate parameter control.
- I thought the synthesizer performed well across different parameter combinations.
- I thought the synthesizer's parameters were difficult to handle and control intuitively.
- I thought the user interface was easy to navigate and understand.
- I thought the synthesizer was able to handle different sounds (percussive, melodic, ambient) well.
- I thought the synthesizer was lacking features.
- I would imagine synthesizer being useful for professional sound designers.
- I would imagine the synthesizer being useful for beginners exploring granular synthesis.
- I would consider using the synthesizer for my practice.

Overall the responses from the test participants were positive and the participants most positively rating the user interface and the performance aspect of the synthesizer, claiming that the software worked with little latency or unwanted artefacts. The participants also thought that the synthesizer was able to handle different types of sounds well and performed well across different parameter combinations.

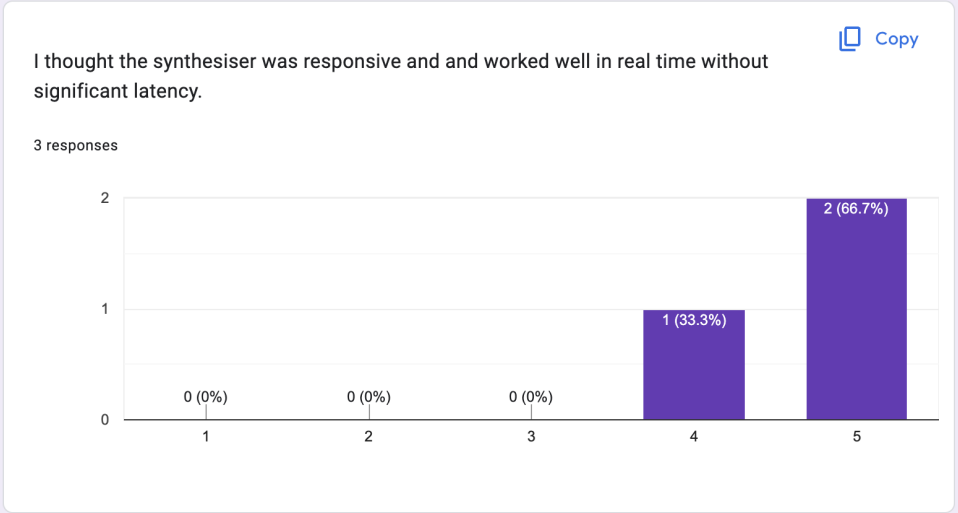

Figure 4.5: Most positively rated statement

On the other hand, the respondents expressed agreement that the software was lacking in features and parameter control. While every participant agreed that the software would be useful for beginners exploring granular synthesis, professional use of the synthesizer seemed less likely.

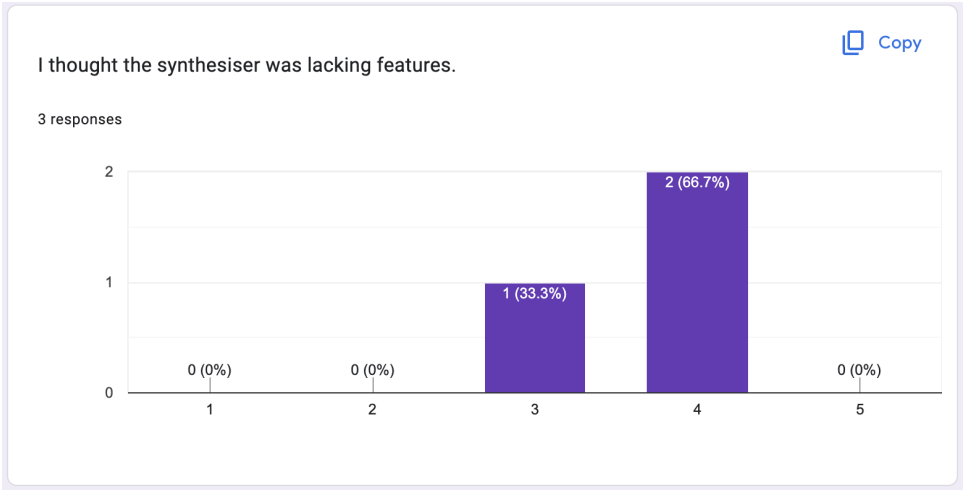

Figure 4.6: Most negatively rated statement

In the window asking to provide additional feedback, all participants responded about the need to include custom samples. Other suggested additions were recording and exporting the performance, adding automation, controlling several parameters at the same time. One of the respondents expressed that additional parameters like grain spread or pan could be added. Overall, the software was seen as something more useful for educational purposes or non-professional music production and something less likely to be used by industry professionals.

Chapter 5

Discussion and Future Works

5.1 Granular Synthesis

Overall, the software was able to perform granular synthesis well, with little latency or unwanted distortion. The synthesizer provided basic functionality for a granular synthesizer, with features like grain size, grain frequency, grain position and pitch. Additional parameters like grain spread, randomisation and pan could be added in the future to make the synthesizer more attractive to music producers and sound designers. The software could also include an option to import custom audio samples, record an audio sample from the microphone or exported the performance as an audio file. The expressiveness of the synthesizer could be improved by including new methods to interact with the parameters, e.g. hand posture estimation to be able to *play* the instrument or simply an XY pad to control two parameters at the same time.

5.2 Web Integration

The synthesizer was successfully hosted on a website and was able to function with no performance issues. The WebAssembly framework allowed the patch to run smoothly in the browser even as the most extreme parameter settings. However, the pipeline from building the RNBO patch to having a fully functional website is still not suitable for complete beginners and knowledge of coding languages like HTML and JavaScript as well as web audio frameworks like Web Audio API was required. While the RNBO developer community provided a website template, it was difficult to utilise properly without more extensive JS knowledge, especially is changes to the user interface of some aspects of the software were required. Due to the novelty of RNBO web integration features, a lot of the documentation is still not available to be able to quickly deploy custom websites from the RNBO patch.

5.3 UI and Aesthetics

The user interface of the software was positively rated by the test participants due to its simplicity and ease of understanding. With a variety of different JavaScript libraries and frameworks available, the UI could be more improved to look more modern and attractive to the user. While a visually stripped-down and aesthetically minimal website seems to be a popular visual choice by designers, addition of a visual element corresponding to the audio could make it more intuitive and engaging. Several JavaScript libraries and frameworks like p5.js, WebGL, three.js are available for developing 2d and 3d graphics and animations. By accessing the parameters of the RNBO patch and linking them to the graphical elements, a simple audio visualiser could be created.

Chapter 6

Conclusion

This paper explores the possibilities of implementing granular synthesis as a website by using the RNBO architecture in Max/MSP. More superficially, the possibility to export RNBO patches and quickly deploy it as websites is analysed. A pipeline from a RNBO patch to a fully functional website is discussed and the necessary tools and knowledge needed to deploy such website is examined. A graphical RNBO code is exported as WebAssembly and included in a JavaScript code to create a device analogous to the RNBO patch.

The software developed during the course of this project resulted in a granular synthesizer functioning on a website. Overall, the synthesizer was able to run smoothly and produce clear sound with no unnecessary artefacts, clicks or distortion. The synthesizer was responsive to the parameter changes and performed with little latency. The created user interface was simple yet functional and easily understandable. Given the feedback from the test participants, the web integration of the granular synthesizer can be considered successful due to its good performance. However, the current software is considered more suitable for exploration by students or sound enthusiasts rather than a legitimate alternative to existing granular synthesizer used by professional music producers and sound designers. With the addition of features like custom audio samples, expressive parameter control or audio reactive visuals, the software could attract more advanced musicians looking for an engaging way to synthesise sounds.

Bibliography

[1] Michel Buffa et al. "Web Audio Modules 2.0: An Open Web Audio Plugin Standard". In: *Companion Proceedings of the Web Conference 2022*. 2022, pp. 364–369.

[2] Anıl Çamcı. "GrainTrain: A hand-drawn multi-touch interface for granular synthesis". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. 2018, pp. 156–161.

[3] *FaceArp: Web synthesizer with face tracking control*. https://github.com/maceq687/FaceArp. (Accessed on 04/24/2023).

[4] *Learning Synths and RNBO*. URL: https://rnbo.cycling74.com/explore/learning-synths-and-rnbo.

[5] *Learning Synths and RNBO | Cycling '74*. https://rnbo.cycling74.com/explore/learning-synths-and-rnbo. (Accessed on 04/24/2023).

[6] Jakob Nielsen. *Why you only need to test with 5 users*. 2000.

[7] *Patatap - Artistic & Studio Work of Jono*. https://www.jono.fyi/Patatap. (Accessed on 04/24/2023).

[8] *PD community site*. URL: https://puredata.info/.

[9] Shihong Ren et al. "FAUST online IDE: dynamically compile and publish FAUST code as WebAudio Plugins". In: *WAC 2019-5th Web Audio Conference*. 2019.

[10] Curtis Roads. *Microsound*. The MIT Press, 2004.

[11] Charlie Roberts and JoAnn Kuchera-Morin. "Gibber: Live coding audio in the browser". In: *ICMC*. Vol. 11. 2012, p. 6.

[12] Sebpiq. *SEBPIQ/WebPd: Run your pure data patches on the web.* URL: https://github.com/sebpiq/WebPd.

[13] *Web Audio Granular Synthesiser*. http://zya.github.io/granular/. (Accessed on 04/24/2023).

[14] *What is Max?* URL: https://cycling74.com/products/max.

[15] Iannis Xenakis. *Formalized music: thought and mathematics in composition*. 6. Pendragon Press, 1992.

Appendix A

The code for the synthesizer can be found here: RNBO granular synthesizer