Computer Music: Languages and Systems

Academic Year 2023-2024

STEFANO LAVORI OFFICIAL TEAM

Galadini Giuliano, Lenoci Alice, Macrì Carlo, Messina Francisco, Molinari Elena

"Generative music is like gardening. You plant a seed, and then you watch the seed grow into something unpredictable and interesting."

-Brian Eno

The proposed application, provides a comprehensive and flexible environment for real-time sound creation and manipulation, offering users a wide range of tools and techniques to explore the possibilities of digital sound synthesis. Based on a hybrid approach that integrates concepts of additive synthesis, digital signal manipulation, and pattern generation, the application allows for a wide range of sonic and musical experiments.

In terms of sound synthesis, the application predominantly employs additive synthesis, specifically designed to simulate the vibration of a string. This method approach is a simplification of modal synthesis, where complex sound waves are approximated by summing multiple sine waves, each representing a harmonic component of the sound. The *SynthDef* defined in SuperCollider calculates the frequencies and amplitudes of these sinusoids based on a fundamental frequency and user-defined parameters such as plucking position. This allows users to create rich and detailed harmonic spectra, enabling the creation of nuanced and dynamic sounds.

Once the sound is generated through additive synthesis, the application includes a real-time granulator for further sound manipulation. Users can adjust the dry/wet parameter and the pitch of the granulated sound, allowing for the creation of complex sounds by combining sinusoidal signals through audio grains extracted from a buffer. The flexibility of the *SynthDef* allows users to creatively modulate and manipulate sounds through the application of different processing blocks.

Additionally, the application offers a wide range of tools for generating and manipulating musical patterns. Using a combination of timed sequences and generative patterns, users can intuitively and dynamically control the trigger, duration, frequency, and other parameters of sounds. This approach enables the creation of complex and varied musical structures, suitable for both live performance and studio music composition.

It is important to understand the philosophy of our instrument: the process of creating rhythms and melodies is purely generative, giving the user the freedom to explore uncharted territories by interacting with a system designed to suggest new directions for the musician.

The extensibility of the application makes it suitable for a wide range of musical contexts, allowing users to explore new frontiers in interactive music composition. By giving the user control over randomness, we have created a system that excels in experimenting with polyrhythms and complex melodies, while also providing the ability to improvise solo or with other musicians.

Additive Synthesis: Simulating String Vibration on SuperCollider

In this application, the string's vibration is simulated by generating an array of frequencies and amplitudes corresponding to the harmonic series of a vibrating string. We begin with the ideal implementation of a string, where each frequency is a multiple of the fundamental frequency of the string. The amplitude of each frequency is influenced by factors such as the plucking position on the string (*p_pose*). These amplitudes are calculated using a formula based on the d'Alambert equation (1) that simulates the energy distribution of a real vibrating string, resulting in a more realistic sound:

$$B_n = \frac{2m^2}{n^2\pi^2(m-1)} \sin\left(\frac{n\pi}{m}\right) \quad (1)$$

where *m* is the plucking position and *n* the harmonic number.

We also considered the condition where the stiffness of the string is relevant. As an effect, each frequency is slightly detuned to simulate the natural imperfections in a real string. The detuning is controlled by a stiffness parameter, which adjusts both a constant detune applied to all harmonic frequencies and a variable part that depends on the harmonic number. This also considers the real theory based on the d'Alambert equation with a fourth derivative component that takes the stiffness into consideration:

$$\mu \frac{\partial^2 y}{\partial t^2} = T \frac{\partial^2 y}{\partial x^2} - ESK^2 \frac{\partial^4 y}{\partial x^4} \quad (2)$$

where μ is the linear mass density, y is the displacement of the string, T is the string tension, E is the Young's modulus, S the cross-sectional area and K the radius of gyration.

As a solution we derived a new formula for calculating each frequency harmonic:

$$f_n = n f_1 \left[1 + \beta + \beta^2 + \frac{n^2 \pi^2}{8} \beta^2\right] \quad (3)$$

$$\beta = \frac{2K}{L} \sqrt{\frac{ES}{T}} = \frac{r^2}{L} \sqrt{\frac{\pi E}{T}} \quad (4)$$

where f_1 is the fundamental frequency, L the string's length, β a parameter representing the effect of stiffness on the harmonic frequencies and r the radius of the string.

The value of beta that gave us a more realistic sound was 0,011.

This additive synthesis approach was used because much simpler to implement compared to real implementation of modal synthesis, even though it may increase CPU usage. To reduce the stress of the CPU we can reduce the number of harmonics that our Synth calculates.

Pattern Generation on SuperCollider

The pattern generation process in the application is a crucial component that shapes the rhythmic and melodic structure of the sound. This process involves creating and manipulating rhythmic patterns, note sequences, and modulation values.

• Euclidean Rhythm Generation

The core of the rhythmic pattern generation is the Euclidean rhythm algorithm. This algorithm distributes a specified number of beats as evenly as possible across a given number of steps. For example, if you want to distribute 5 beats over 16 steps, the Euclidean algorithm ensures that these beats are spaced as evenly as possible, creating interesting and often complex rhythmic patterns.

The steps parameter defines the total length of the rhythm cycle, while the pulses parameter determines how many of these steps will contain beats. The rotation parameter then shifts the pattern, effectively changing the starting point of the cycle, which adds further variation to the

rhythm. This rotation can be controlled and modified in real-time, allowing dynamic changes to the rhythm during performance.

• Combination with Random Sequences

In addition to the Euclidean rhythm, the application integrates randomness to further enhance the variability and unpredictability of the patterns. A random sequence is generated and updated continuously, with each element in the sequence being a binary value (0 or 1), determined by a coin flip biased by the probability parameter. This sequence is then combined with the Euclidean rhythm using logical operations such as OR, AND, XOR, and NAND.

These operations determine how the two patterns interact. For instance, using an OR operation means that a step will be active (a beat will occur) if either the Euclidean rhythm or the random sequence has a beat at that step. This combination can create highly intricate and evolving rhythmic patterns that blend deterministic structure with stochastic elements.

• Pitch, velocity and modulation

To create melodies, users interact with the pitch section of the application. Initially, a random note sequence is generated by adjusting a probability distribution. The expected value and the variance of this distribution can be controlled by changing the x and y coordinates on the 2D slider, respectively. Modifying the variance allows the probability distribution to smoothly transition from a narrow Gaussian distribution to a uniform distribution, and eventually to a distribution that generates only extreme values.

The generated notes are then quantized, with users having the option to choose from ten different scales. Modulation and velocity are implemented in a similar way, although without the quantization process. The modulation section specifically affects the plucking position of the string.

For both notes and modulation, users can apply glide. Negative glide values result in a linear glide, while positive values produce an exponential glide.

• Looping

The application supports looping of random sequences, including triggers, melodies, velocities, and modulations. Users can enable or disable looping for different attributes and adjust the lengths of these loops individually. This feature is particularly useful for creating cyclical rhythmic motifs, repetitive melodic phrases, and intricate, long sequences. For example, setting the random trigger loop length to 7 and combining it with a 10-step Euclidean pattern produces a 70-step pattern. The complexity can be further enhanced by introducing loops of different lengths for pitch, velocity, and modulation.

For note and random trigger loops, users can permute the elements of the pattern by turning a knob, which generates new sequences. This capability, combined with the ease of changing loop lengths, is especially powerful for live performances.

In summary, pattern generation in this application combines deterministic algorithms, such as Euclidean rhythms, with stochastic elements like random sequences and Gaussian-distributed

modulation values. These patterns can be dynamically manipulated, looped, rotated, and modulated, offering a sophisticated and versatile framework for generative music creation.

JUCE-Based VST3 Plugin

In this project, we developed two VST3 plugins using the JUCE framework to provide advanced audio processing capabilities. These plugins are designed to offer creative tools for sound manipulation, featuring essential audio effects such as distortion, folding, and flanging.

• Distortion and folder

In the first VST3 plugin developed using the JUCE framework, we've implemented two essential audio effects: distortion and folding. These effects are fundamental in shaping and enhancing audio signals, providing users with creative tools for sound manipulation.

The distortion effect implemented in our plugin introduces harmonic richness and saturation to audio signals. This effect is achieved by non-linearly amplifying the input signal, causing signal clipping and introducing harmonic overtones. By adjusting parameters such as the *distortion amount,* users can control the intensity of the effect, ranging from subtle warmth to aggressive saturation.

The folding effect, also known as wavefolding, is a unique form of distortion that creates complex waveform shapes by folding parts of the signal that go over a certain threshold back onto itself. This effect adds harmonically rich overtones and creates unique timbral characteristics in audio signals. In our plugin, the folding effect is achieved by dynamically manipulating the *pre-gain* on the audio signal, folding it back upon exceeding predefined thresholds. Users can adjust parameters such as the *folding amount* to control the intensity of folding, allowing for versatile sound sculpting possibilities.

• Flanger

In the second plugin, we've implemented the flanger effect to offer users a versatile audio modulation tool for creating dynamic and immersive soundscapes.

The flanger effect is a classic modulation effect characterized by its sweeping, comb-filter-like sound. It works by modulating a delayed version of the input signal and mixing it with the original signal, resulting in a distinct sweeping effect with notches in the frequency spectrum.

This effect is controlled by several key parameters:

The *Wave Type* allows users to select the waveform shape used for modulation, offering options such as sine, square, triangle, or sawtooth. The *Rate* controls the speed of modulation, determining how quickly the flanger sweeps through the audio spectrum, and can reach audible frequencies for further sound manipulation. The *Depth* adjusts the intensity of modulation, controlling the amplitude of the modulated signal. The *Feedback* sets the feedback level of the effect, influencing the amount of signal fed back into the modulation loop. The *Width* determines the range of frequencies affected by the flanger effect. The *Dry/Wet Mix* balances the ratio between the dry (original) and wet (processed) signals, allowing users to blend the effect seamlessly with the original audio. The *Color* adjusts the tonal character of the effect, offering

tonal shaping options to match the desired sonic aesthetic. Finally, the *Stereo Spread* controls the stereo width of the effect, allowing users to adjust the spatial distribution of the modulated signal.

The flanger effect is achieved by modulating a delayed version of the input signal using an LFO (Low-Frequency Oscillator). The LFO generates a periodic waveform, such as a sine, square, or triangle wave, which is then used to modulate the delay time of the delayed signal. The modulated delayed signal is mixed with the original signal, resulting in the characteristic sweeping effect of the flanger.

By integrating these effects into our plugin, we provide users with powerful tools for shaping and enhancing their audio productions.

Graphical User Interface (GUI) Implementation

The graphical user interface (GUI) in the provided code is crucial for facilitating user interaction and control over sound synthesis and processing operations. Implemented using the ControlP5 library in Processing, this GUI comprises various interactive elements such as toggles, knobs, and sliders. The left side and central part of the view contains all the parameters that control the generative synth in SuperCollider, while the right side contains all the controls of the Folder&Distortion and Flanger VST3 plugins.

At the top left of the GUI window, horizontal toggle buttons are prominently displayed, each representing a distinct operational mode. These buttons allow users to seamlessly toggle the loop modes, enhancing flexibility and ease of use.

The GUI features several blocks containing knobs, each dedicated to controlling specific parameters related to sound synthesis and processing. These knobs, created from the ControlP5 library, enable users to finely adjust the parameters of the application.

In addition to knobs, the GUI incorporates slider controls for manipulating two-dimensional parameters. These sliders, provide users with an intuitive way to adjust parameters within a defined range by visually positioning a point on the interface. They control the expected value (x direction) and variance (y direction) of the distribution of notes, velocity, and modulation (plucking position of the string) respectively.

In the middle part of the GUI we can see three different controls which allow the user to stop or resume the generative synthesiser. In addition, the user can control the BPM of the sequence, while in the bottom of the GUI the user can observe the shape of the Euclidean rhythm, which allows for a visual representation of the sequence.

Overall, the GUI enhances user interaction by offering tactile controls for adjusting parameters and exploring various sonic possibilities within the Processing environment. It provides a visually appealing and user-friendly interface for controlling sound synthesis and processing operations.

Communication from Processing to SuperCollider and VSTs via OSC

The application uses the OscP5 library in Processing to manage communication via the OSC (Open Sound Control) protocol. This allows for sending messages from Processing to SuperCollider and the VSTs for real-time control of sound parameters.

In the setup() method, multiple OSC connections are established towards SuperCollider and the VSTs. Processing communicates through the localhost address (127.0.0.1).

The Processing GUI allows users to interact with various controls, such as knobs and sliders. When the user interacts with one of these controls, a callback event is generated. This event creates an OSC message containing the current values of all the controls and sends it to SuperCollider and the VSTs. The other applications receive these messages and update their parameters in real time based on the received values.

Sensor-Controlled Integration with Bela

The integration of external sensors with some elements in the GUI is facilitated by Bela, serving as a crucial intermediary platform. Processing listens for an OSC message received in the *oscEvent(OscMessage theOscMessage*) function, which has been sent by Bela. It acts as a bridge between the external sensors and the Processing environment, translating the sensor data into OSC messages that the GUI can understand.

Bela interfaces with external sensors to gather data, including the XY coordinates from a joystick controller (which can also be pressed to switch between 2D sliders) and the state of a digital tilt sensor (either 0 or 1). Bela then packages this data into OSC messages and transmits them to the Processing environment.

In Processing, the *oscEvent* function receives these OSC messages, extracts the necessary data, and dynamically adjusts the positions and configurations of the Slider2D controls based on the joystick controller's inputs. Concurrently, the digital tilt sensor's state is used to activate or deactivate the stereo amount of the flanger VST created with JUCE.

By leveraging Bela's capabilities, the GUI achieves real-time responsiveness to sensor input. This integration enables users to interact intuitively with the controls of both SuperCollider and JUCE. Thus, Bela plays a pivotal role in enhancing the user experience by providing seamless and precise control over specific parameters in the GUI through sensor input.

References

(1),(2),(3),(4). ***'The physics of musical instruments'*** – Fletcher, Rossing, Springer, 1998