Modal Resonators for Real-Time Synthesis

Pi Xiaojie
Sean Turland

Abstract

This project develops a modal synthesis plugin for experimental sound design. Using a contact microphone to capture vibrations as an excitation signal, the plugin processes the energy through a bank of resonators simulating physical models like strings, plates, and tubes. Users can control parameters like structure and brightness, allowing for a wide range of sound timbres. Initial results demonstrate that the plugin effectively generates diverse acoustic-like sounds, making it suitable for creative audio applications.

1 Introduction

Physical models of acoustical behaviour for use as a real-time synthesis technique exists in many forms. Through techniques including Karplus-Strong [9] and Digital Waveguides [8], efficient computation of acoustic qualities can be attempted, with varying degrees of accuracy to the underlying mathematical description of the acoustic system.

In this project, we aim to create a sound synthesizer which can approximate and interpolate between physical behaviour, for use in sound design or sound art. Although any sound source can be used as audio input, we choose to focus our design on its use with a contact microphone, which captures small vibrations of an object. Inspired by the Elements eurorack module by Mutable Instruments, our plugin is based on modal synthesis. Under this paradigm, instruments are decomposed into an *exciter*, the energy input into the system, and a resonator, which amplifies the energy at specific resonant frequencies to simulate various resonating objects such as strings, tubes, and plates. This sound can synthesized efficiently by feeding an excitation signal into a parallel bank of resonators [2]. The input signal is first processed through a low-pass filter and then fed into this resonator, with parameters like structure, brightness, and damping affecting the resulting sound. Adjusting these settings can create a wide range of timbres, from sharp percussive tones to rich, sustained harmonics.

As an illustrative aside, we refer to the piece "*I am sitting in a room*" by Alvin Lucier [5], which demonstrates the use of an initial speech excitation used to resonate the natural modal frequencies, upon repeated feedback into the system. The aim of this project is to create a sonically comparable, controllable instrument with this technique in mind, albeit with different geometric resonator models.

2 Background

Objects (stiff strings, plates, metal rabbits [6]) have natural resonant frequencies, based on the geometry and material of its construction, called modes. These modes represent solutions to the standing wave equation for those specific initial and boundary conditions [4].

For an acoustic model with one degree of freedom, harmonic frequencies can be modelled as proportional to a fundamental mode. Each mode, k, can be characterised by the parameters

$$(f_k, A_k, R_k) \tag{1}$$

which represent the frequency, amplitude, and loss factor respectively. Adopting physically-inspired modal-dependant relationships between these parameters can lead to strikingly complex timbral results.

For an ideal string under linear assumptions, the frequency-modal relationship is directly proportional,

$$f_k \propto k \tag{2}$$
$$f_k = k f_0 \ . \tag{3}$$

Although this conserves a regular harmonic pitch relationship, it doesn't account for the dispersive properties of a physical wave. The resonator in this paper is instead based upon the stiff spring model [7], which models solutions to the linear wave equation with an additional stiffness term

$$\epsilon \ddot{y} = K y'' - \kappa y'''' \tag{4}$$

to account for the stretching of the string as it bends. With a greater bending angle of string at higher frequencies, the frequency-dependant resolving force is more pronounced. This introduces a perceptually interesting inharmonicity, which obeys the proportionality relationship

$$f_k \propto k\sqrt{1 + Bk^2} \tag{5}$$

and allows for timbral variations based on a variable stretch factor, B, that provides a closer sonic approximation to the true physical behaviour [1]. Due to real-time constraints, a fast approximation of the square root

$$f_k \propto k[k + (k-1)B] \tag{6}$$

is utilised. In modal synthesis, these frequencies are modelled by individual bandpass filters in parallel, each centred on a particular modal frequency.

The modal decay factor, R_k, represents how stable the energy vibrating at mode k in the particular system is, dictated by a longer decay time and

resonance at that frequency. In modal synthesis, this is represented by each bandpass filter's Q, which controls the amount of feedback (or resonance). Overall, energy at higher frequencies dissipates faster than low frequencies, but there are also geometry-specific dependencies which act as crucial auditory signatures in our perception of timbre. By controlling both an overall decay parameter and a frequency-dependant damping parameter, while coupling individual decay factor relationships to the modal geometry, synthesis-like control over the resonator's timbre is made possible.

The modal amplitude, A_k, is also dependant on the geometry of the resonant system, and the initial conditions. Any prior experience of plucking a guitar string at different location will motivate that the position of excitation of a resonant system effects the distribution of energy to the range of possible modes. To simulate this effect, the initial amplitude of each mode can be approximated by the sinusoidal relationship

$$A_k = sin(\pi k x) \,, \tag{7}$$

where $x \in (0,1)$ denotes the position along the string. This creates a variable control of the frequency content and a relationship between modes, much-like plucking a guitar string close to the bridge will excite high frequency modes to a greater extent than at the neck.

3 Implementation

Algorithm 1 Resonator::ComputeFilters()

```
Interpolates stiffness and q based on pre-calculated lookup tables.
Reduces the range of brightness to prevent clipping.
for i = 0 to min(kMaxModes, resolution) do
    Calculates the partial frequency.
    Sets the filter properties (frequency and Q(quality factor)).
    Adjusts the stretch factor.
end for
return number of modes
```

The variable `stiffness` is interpolated from the parameter `structure` based on a pre-calculated lookup table, which stores the resonator structures in Resources header files, ranging from plates to strings, bars/tubes, and bells/bowls.

The stretch factor represents the inharmonicity [3], which controls the spacing between harmonic partials. When the frequencies of the modes are in harmonic ratios, the sound is perceived as very musical and strongly pitched. However, when the modes are not aligned with integer ratios of the fundamental frequency, the sound takes on a metallic quality with an ambiguous pitch. In the `ComputeFilter` method, the stretch factor is increased by the modes, meaning the frequency spacing between partials widens as more modes are added. This simulates the behavior of stiff, inharmonic resonators like bells, allowing for a range of timbres from tonal to more dissonant sounds.

In total, this method computes the number of modes based on the parameter settings and sets the properties of each band-pass filter.

Algorithm 2 Resonator::process()

```
Compute the number of active modes.
Ramp the position.
Limit the number of output channels to 2.
for n = 0 to numSamples - 1 do   ▷ Loop through each sample in the input buffer
    Initialize a Cosine Oscillator for amplitude modulation
    Get the input sample, scaled by 0.125
    for i = 0 to numModes - 1 do   ▷ Process through each mode and apply filters
        Process odd mode and add to output channel 0
        Process even mode and add to output channel 1
    end for
end for
```

The `Resonator::process` method first computes the number of active resonator modes (or partials) by calling `ComputeFilters`. This determines how many band-pass filters will be applied to the input signal. The parameter `position` is ramped, preventing abrupt changes and resulting in smoother sound modulation.

The method then sets the number of channels to 2 for simplification, corresponding to a stereo output (left and right). It processes each sample in the buffer using a loop where a `CosineOscillator` is initialized and started. This oscillator modulates the amplitude of each resonator mode, with its parameters updated for each sample based on the interpolated position value.

For each sample, the input from the first channel is scaled down by 0.125 to manage signal levels and prevent clipping. The method then processes each mode using a state-variable filter (`svf[i]`). The filter output is modulated by the cosine oscillator's amplitude. The results for odd and even modes are accumulated separately into the left and right output channels, respectively, creating a stereo effect which would be difficult to replicate in an acoustic resonator. Then, the processed audio is written to the output buffer.

As shown in Figure 1, in `MainProcessor::processBlock`, we first pass the input buffer through a low-pass filter, which serves as the excitation signal for the resonator. After applying the output gain, we obtain the output signal.

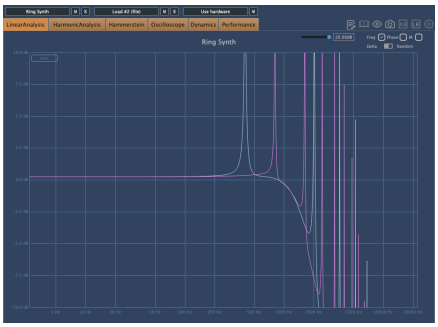
Figure 1: flow chart of the main process block

4 Discussion

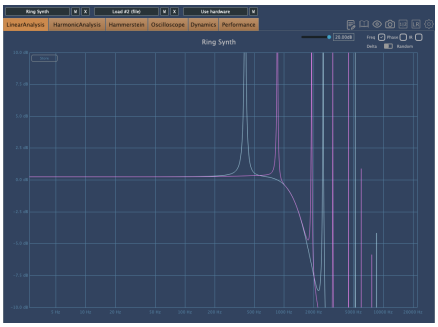
We analyzed how changing the `structure` parameter affects the output spectrum of the resonator. Figure 2 shows the spectral responses at four different values of `structure`, while keeping `brightness = 0.5`, `damping = 0.5`, and `position = 0.25`. As the `structure` value increases from 0.25 to 0.95, we observe a shift in the distribution of modal frequencies and their relative amplitudes.

- **At structure = 0.25**, the graph reveals more tightly grouped harmonics, typical of softer, string-like resonances.
- **At structure = 0.55**, the frequencies start to spread further apart, creating a slightly more metallic timbre.
- **At structure = 0.75**, the harmonic content becomes even more inharmonic, simulating the qualities of bars or tubes, with the inharmonicity becoming more pronounced.
- **At structure = 0.95**, the resonator reaches its most dissonant state, resembling bell-like sounds with widely spaced frequencies.

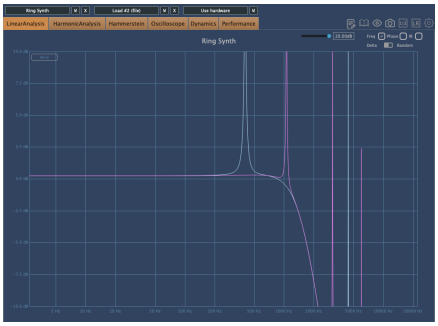
These observations confirm that increasing the `structure` value widens the frequency spacing and shifts the resonator from producing soft, harmonic sounds toward more rigid, inharmonic timbres.



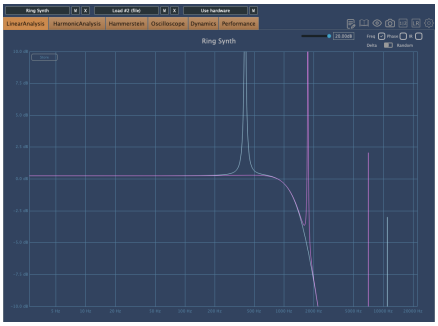
(a) structure = 0.25,



(b) structure = 0.55



(c) structure = 0.75



(d) structure = 0.95

Figure 2: Linear analysis graphs with different `structure` settings

While our implementation is only monophonic, we found the final result is creatively motivating, producing interpolations of acoustic sounds which can be controlled expressively through interesting excitation inputs. The current

implementation demonstrates the potential for a modal synthesis plugin which can be played like an acoustic instrument, and future enhancements may build upon this foundation.

For future work, we would look to implement a customised MIDI handler to allow for detailed expressive control of the frequency and position, where the instrument could be played like a traditional tunable acoustic resonator. In addition to the current parameters, we could also add a frequency modulation block to the resonator, which could create a richer and more complex sound. This would allow for a wider range of expressive timbres and enhance the versatility of the synthesizer.

References

[1] Balázs Bank, Stefano Zambon, and Federico Fontana. A modal-based real-time piano synthesizer. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4):809–821, 2010.

[2] Stefan Bilbao. Modal synthesis, 2004.

[3] Wikipedia contributors. Inharmonicity, 2024.

[4] Joseph Derek Morrison and Jean-Marie Adrien. Mosaic: A framework for modal synthesis. *Computer Music Journal*, 17(1):45–56, 1993.

[5] Alva Noë. *Learning to Look: Dispatches From the Art World*. Oxford University Press, 2021.

[6] Zhimin Ren, Hengchin Yeh, and Ming C. Lin. Example-guided physically based modal sound synthesis. *ACM Trans. Graph.*, 32(1), feb 2013.

[7] Julius O. Smith. *Physical Audio Signal Processing*. `http:`http://ccrma.stanford.edu/ jos/pasp//`ccrma.stanford.edu/~jos/pasp/`. online book, 2010 edition.

[8] Julius O Smith. Physical modeling using digital waveguides. *Computer music journal*, 16(4):74–91, 1992.

[9] Vesa Välimäki, Jyri Huopaniemi, Matti Karjalainen, and Zoltán Jánosy. Physical modeling of plucked string instruments with application to real-time sound synthesis. In *Audio Engineering Society Convention 98*. Audio Engineering Society, 1995.