Audio Engineering Society

Convention Paper 10184

Presented at the 146th Convention
2019 March 20 – 23, Dublin, Ireland

Real-time synthesis of sound effects caused by the interaction between two solids

Pedro Sánchez[1] and Joshua D. Reiss[1]

[1]*Queen Mary University of London*

Correspondence should be addressed to Joshua D. Reiss (joshua.reiss@qmul.ac.uk)

ABSTRACT

We present the implementation of two sound effect synthesis engines that work in a web environment. These are physically driven models that recreate the sonic behaviour of friction and impact interactions. The models are integrated into an online project aimed at providing users with browser-based sound effect synthesis tools that can be controlled in real time. This is achieved thanks to a physical modelling approach and existing web tools like the Web Audio API. A modular architecture was followed, making the code versatile and easy to reuse, which encourages the development of higher-level models based on the existing ones, as well as similar models based on the same principles. The final implementations present satisfactory performance results despite some minor issues.

1 Introduction

Research advances have started to make the synthesis of sound effects a viable alternative to Foley and sample libraries. In the last couple of decades, physical modeling engines have become relevant in the study of sound [1]. This type of synthesis provides natural dynamics and relevant controls that help: studying aspects like expressiveness in human-driven acoustic phenomena [2] (which includes music); building auditory based interactive systems, in which real-time, precise feedback is required [3], and studying sound perception from a more ecological perspective that deviates from classical, signal-processing-based studies [4]. On the other hand, books like *Designing sound* [5] and tools like Pd[1] or Max/MSP[2], are encouraging sound designers to explore and interpret the mechanics of sound generation, and build their own sound effect synthesizers. This leads to the development of computationally efficient synthesis engines, that follow an approach to sound generation more suitable for the creation of sound effects than the ones used by traditional synthesizers.

An important part of research in sound synthesis nowadays consists on creating sound effect synthesis models that work in real time. These models would allow sound designers to work by directly controlling the synthesizer to make it fit the visuals, rather than trying to find a previously recorded sound or recording it from real sources. This eliminates the need of expensive equipment and sound libraries, as well as the process of searching through thousands of individual recordings. Moreover, such engines can be adapted for videogames so that they can be controlled by the game interactions (just like in any other interactive application). This allows using a few synthesis models to create thousands of sounds that interact in real time

[1]Pd website: puredata.info/
[2]Max/MSP website: cycling74.com/

with the user, instead of using thousands of samples that take large amounts of storage and must be modified to provide effective interaction. Implementations of this can already be seen in Rockstar's Grand Theft Auto videogame[3] and the FXive website[4].

This work involves development of a synthesis model of sound effects produced by the interaction between solids. The intention is to create a synthesis engine that works in real time, responds to live user input and provides a realistic sound. To achieve this, the implementation will follow the design of the Sound Design Toolkit (SDT) [6], a library of physics-based models that share some of the same requirements as this project. At the same time, the target is to achieve a convincing rather than accurate sound reproduction. Consequently, the design of the model will use simplification of the physics mechanisms in order to: develop more efficient algorithms, and obtain "cartoonification", i.e., an exaggerated effect that might end giving a more convincing result (despite being inaccurate). Ultimately, the aim of this work is to include the model in FXive [7], a website storing over 40 procedural sound effect models. Consequently, to be able to use it as a web tool and fit the standards of the mentioned website, the model will be developed in JavaScript, making use of the Web Audio API, intended to facilitate the creation of audio applications for the web. Therefore, this project intends to contribute in this area of research, to provide realistic, precise and useful tools for sound designers, that will help them focus on the creative aspects of their crafts (i.e., composing soundscapes and effects) rather than getting lost on the technical apects.

This paper is organised as follows. Section 2 describes the frameworks used for constructing the real-time web-based synthesis of solid-solid interaction sounds. Section 3 goes into detail on the modeling method, consolidating and formalising the approach from [1, 6]. First, modal synthesis is presented. Then, solid-solid interaction is described. This consists of formalising the sound production mechanism of a modal resonator, and then showing how this is implemented for both friction and impact interactions. Section 4 shows how the theory of section 3 is constructed into a synthesis model and implemented in real-time with the web audio API. Section 5 then provides results, including comparison against related implementations especially in regards to control parameters, and performance in emulating interaction scenarios from recorded samples, namely a creaking door and rubbing a wine glass. Finally, section 6 summarises and discusses the work.

[3] www.youtube.com/watch?v=L4GuM15QOFE
[4] fxive.com/\#home

2 Frameworks

The models designed throughout this project came from the Sound Design Toolkit and were reimplemented to fit the requirements of FXive. This section introduces both frameworks, in order to provide some familiarity with concepts used throughout the rest of the article.

2.1 Sound Design Toolkit

The Sound Design Toolkit is an open source software package that provides a virtual library of sound effects synth engines [6]. The aim of this toolkit is to develop a set of physics-based models that can be used in research and education in the field of Sound Interaction Design. The developed algorithms are computationally affordable for real-time applications and facilitate the interaction of sound models with physical objects. The implementations aim for auditory perceptual relevance, simplification of the underlying physics, i.e. cartoonification, and natural and expressive parametric temporal control [6].

The Sound Design Toolkit is organised according to a hierarchy that goes from low-level sound events to more complex and compound processes. At the same time, most models follow a modular structure of resonator-interactor-resonator (more in Section 4.2), that represents the interaction between two objects. The friction, impact and rolling models that are presented in this work follow the implementation included in the Sound Design Toolkit, turning them from Max/MSP patches into JavaScript. At the same time, the inner structure of the model (i.e., the resonator-interactor model) has been kept, in order to facilitate the implementation of other low level or friction/impact-derived (e.g., braking) models.

2.2 FXive

FXive is an online hub that stores several sound effect synthesis engines designed to be used from the browser. This project intends to create a framework that allows users to create a wide range of sound effects (impact sounds, harmonic sounds, sound textures,

soundscapes...) from scratch. Apart from the mentioned sound effect models, it also includes audio processing effects and spatialisation functionality, enabling post-processing of the synthesized sounds. The models in this site work in real time and provide high-level controls, contributing to an intuitive and simple manipulation [7].

In order for these engines to work on the web, implementation is done in JavaScript using the functions provided by the Web Audio API [8], the NexusUI API [9] for UI elements and the JSAP [10] plugin standard to encapsulate each model.

3 Theoretical Models

Physical modeling aims at reproducing the physical mechanisms of sound production, using models that describe the mechanical and acoustical behaviour of the sound sources. The power of this kind of synthesis is its ability to model the coupled behaviour of an exciter (i.e., the object that causes vibration) and a resonator (i.e., the body of the instrument or object that responds to the excitation), that recreates the complexity and nuances of real sounds, while keeping a direct correspondence between software processes and physical components.

Generally speaking, physical approaches attempt to model the propagation and resonances of sound by using delays and filters, which usually derive in high computational cost, potential instability and memory use for delay buffers. Consequently, for real time applications, algorithms that are not fully physically based, like modal synthesis, might be considered. This section gives a brief overview of the theoretical principals behind modal synthesis, explaining how it is applied to solid-solid interactions,and specifying the differences between impact and friction interactions.

3.1 Modal Synthesis Theory

Modal synthesis theory assumes that a vibrating object can be described by its normal modes of vibration, each of them causing oscillation at a particular frequency [1]. Consequently, an approach to synthesizing the sound coming from solid objects is modeling their resonant frequencies as a bank of physically based oscillators, that are excited by an external stimulus. This model is physically well motivated for two main reasons: the differential equations for a vibrating system (given appropriate boundary conditions) have a sum of exponentially decaying sinusoids as a solution (which is the behaviour of a bank of mechanical oscillators), and the synthesizer is driven by the contact forces (i.e., it responds to physically based inputs), assuming that the sound producing phenomena are linear. Moreover, a modal resonator bank can be efficiently computed [11].

3.2 Solid-Solid Interactions

3.2.1 Modal Resonators

The building block of the modal resonator is the driven harmonic oscillator, which models the motion of a mass attached to a spring and a damper, and is defined by the equation

$$\ddot{x} + g\dot{x} + \omega_0 x = \frac{F_{ext}}{m}, \tag{1}$$

where x represents the position of the mass or displacement, $\dot{x}$ and $\ddot{x}$, respectively, the first and second derivatives of x with respect to time, k the stiffness of the spring, ω_0 the centre frequency of the oscillation, m the mass and g the damping coefficient. A modal resonator is represented by a bank of N oscillators, that is, a system of N equations like the one described in (1). F_{ext} is the external force on the oscillator, defined as the sum of the force applied by the user and the interaction force f, described in Sections 3.2.2 and 3.2.3. The oscillator displacement x at each discrete time instant n is defined by the difference equation

$$x[n+1] = b_1 F_{osc} - a_1 x[n] - a_2 x[n-1], \tag{2}$$

where F_{osc} is the internal force of the resonator at instant n, obtained as the sum of the external and oscillator's forces. The coefficients b_1, a_1 and a_2 are defined as

$$b_1 = (\frac{T_s e^{-gT_s}}{m\omega_0}) sin(\omega_0 T_s), \tag{3}$$

$$a_1 = -2e^{-gT_s} cos(\omega_0 T_s), \tag{4}$$

$$a_2 = e^{-2gT_s}, \tag{5}$$

where T_s is the sampling rate, and they are obtained by the Impulse Invariance Method applied to the continuous-time function of the oscillator.

The vibration of the surface at a specific point is calculated as a weighted sum of the displacements calculated in (2) for each oscillator in the resonator. The equation of the displacement of the surface of the resonator at a point j is

$$x_j = \sum_{l=1}^{N} t_{lj} x_l, \tag{6}$$

where x_l represents the displacement of oscillator l and t_{lj} the weight applied on oscillator l for surface point j [1, p. 140]. This displacement x_j is sent to the audio output to generate the sound.

3.2.2 Friction interaction equations

Different approaches have been taken to describe the behaviour of friction interactions [12, 13]. For this implementation, an elasto-plastic model proposed in Dupont et al. [14] has been followed. This model describes the interaction as a single state system that represents average bristle behaviour, i.e., the tangential force affects the average deflection, and bristles slip when a certain threshold is reached (see Figure 1).

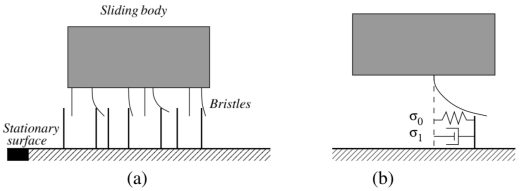


Fig. 1: Bristle interpretation of friction (a) and the averaged bristle behaviour LuGre model (b)[1, p. 152]

The elasto-plastic modelling approach can be summarised by the pair of equations

$$\dot{z}(v,z) = v[1 - \alpha(z,v)\frac{z}{z_{ss}(v)}], \tag{7}$$

$$f(z,\dot{z},v,\omega) = \sigma_0 z + \sigma_1 \dot{z} + \sigma_2 v + \sigma_3 \omega. \tag{8}$$

Equation (7) describes the behaviour of z, which can be interpreted as the mean bristle displacement (and therefore, $\dot{z}$ as the mean bristle velocity). Variable v is the relative velocity between the interacting surfaces at a specific point and is defined as

$$v = \sum_{m=1}^{N^{(r)}} t_{mj}^{(r)} \dot{x}_m^{(r)} - \sum_{l=1}^{N^{(b)}} t_{li}^{(b)} \dot{x}_l^{(b)}. \tag{9}$$

This is derived from the relative displacement described in (6), with b and r identifying each of the resonators, and m and l each of the oscillators in the resonators. Functions α and z_{ss} can be parametrized in various ways, and in this case the ones followed by [15] are used. Equation (8) defines the friction force f, which is the result of summing three components: $\sigma_0 z$, which is an elastic term; $\sigma_1 \dot{z}$, that is an internal dissipation term; the viscosity term $\sigma_2 v$, and a fourth component $\sigma_3 \omega$, introduced by [15], that models surface roughness by introducing noise to the signal. The term $\omega(t)$ is a pseudo-random function, modeled as fractal noise, which is noise with a power spectrum $W(\omega) \sim \omega^{\beta}$.

Therefore, friction force f can be obtained by calculating $\dot{z}$ in (7) and then using it in (8). Both equations use v, which is obtained via (9), that uses the results from (2).

3.2.3 Impact interaction equations

For the impact engine, a model that takes into account complex, hysteretic behaviour has been followed. This model [16], defines the interaction force f as

$$f = \begin{cases} kx^{\gamma}(1+\mu v) & x > 0 \\ 0 & x \leq 0 \end{cases}, \tag{10}$$

with λ being the surface's damping weight, k being the stiffness, $\mu = \lambda/k$ being a mathematically convenient term and γ being a parameter dependent on the geometry (see [17] for a detailed analysis). Variable x is the relative displacement between resonators and is defined as

$$x = \sum_{m=1}^{N^{(r)}} t_{mj}^{(r)} x_m^{(r)} - \sum_{l=1}^{N^{(b)}} t_{li}^{(b)} x_l^{(b)}. \tag{11}$$

This term is derived from (6). On the other hand, v is the relative velocity defined in (9). As in the case of friction, force f is added to the user input force in (1) and (2). With those equations, the displacement of the resonator at a specific point of its surface is calculated.

4 Model Design and Implementation

4.1 Requirements

The main intention of this work is to develop an interactive tool that can be used as an instrument and is efficient enough to run from the browser. To meet this goal, the model must:

- Produce a convincing sound effect
- Be developed in JavaScript, in order to use it as a web tool
- Produce sound in real time
- Respond to live user input, producing the sound as feedback

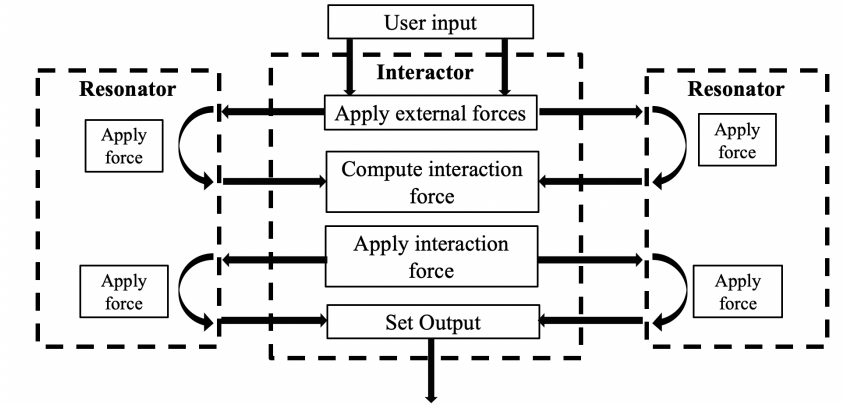


Fig. 2: Block diagram of the `Interactor-Resonator` architecture

The first requisite addresses the need of creating a model that sounds subjectively close enough to the real sound effect, without being a recreation. Therefore, simplification, exaggeration and "cartoonification" are encouraged. The second requirement is a purely technical one. The goal of the third and fourth requisites is to achieve an interactive application that can be controlled seamlessly.

4.2 Resonator/Interactor architecture

An object-oriented approach was taken in the implementation of the SDT, facilitating the development of complex synthesis engines using simpler models as building blocks and allowing reuse of code in different models. In order to support modularity and set the basis for a full Web implementation of the Sound Design Toolkit, it was considered necessary to keep this structure.

Therefore, the basic building blocks for these engines are the `Resonator` objects. They hold the data about the resonating bodies, such as the physical properties of the material, the contact points with other resonators, and the bank of damped oscillators. The functions related to this object are mainly oriented to set, retrieve or calculate these values.

Once the `Resonator` objects are defined, interactions between them can be created with `Interactor` objects. These objects act as an interface: they store the `Resonators` involved in the interaction, the contact points, the energy of the interaction and the state variables that are related to the specific type of interaction (i.e., friction, impact, etc.). The main function executes the following actions, also shown in Figure 2:

1. Apply external forces (user input) to each of the resonators (equation (1))
2. Calculate the force caused by the interaction (i.e., the non-linearities of the model)
3. Apply the calculated force to each of the resonators
4. Calculate the displacement and set it as an audio output

Therefore, steps 1 and 3 compute equation (1) on both of the `Resonator` objects. The second step calculates the force by computing 7 and 8 for friction, and (10) for impact. (9) and (11) are also required in this step. The last step uses (6) to calculate the displacement of one of the `Resonator` objects at a specific point and set it the audio output.

Finally, the sound is produced when the user manipulates two parameters in the friction model, and one parameter in the impact model. For the friction sound, the user controls the external forces, tangential and normal, that are applied by one object onto the other. In the impact model, every time the user strikes, the displacement of the hitting object is set to zero and the velocity to a specified value. It is important to highlight that these are not the only parameters that might be shown to the user, though they are the ones responsible for sound generation.

4.3 Implementation

4.3.1 Javascript Implementation

The models released with the Sound Design Toolkit consisted of a collection of Max/MSP objects coded in

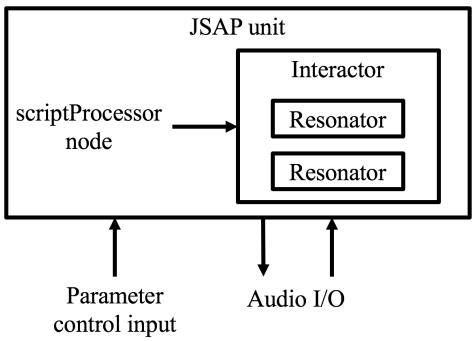


Fig. 3: Block diagram of the script architecture

C, intended to be used in patch design, and they had to be translated to JavaScript.

The main consequence of moving from Max/MSP to JavaScript is audio management. In principle, both Max/MSP and JavaScript's Web Audio API follow the same approach [8]: creating audio chains by using several modules with specific functions. However, this model requires sample-wise audio generation. Consequently, the original code from the Sound Design Toolkit relied on developing the necessary audio generating modules from scratch in C [6]. The alternative offered by the Web Audio API used for this implementation was the `scriptProcessor` node. This is a node type that was included as a way to create customised audio-processing code and easily incorporate it into a Web Audio chain [8]. Therefore, all the described functions were implemented inside one `scriptProcessor` node, that was later connected to an audio output.

The `scriptProcessor` node is a sub-optimal workaround to enable inclusion of user audio scripts and is deprecated. However, at the time of writing this article, it is in full use and its main alternative, the AudioWorklet, is unstable. This led to the decision to use the `scriptProcessor` for the implementation.

Another minor deviation is the noise generator used in the friction model. In [1] and [15], a noise component is added to the friction force by a fractal noise generator. However, this caused stability issues when translated to Javascript, often leading to infinite outputs. Consequently it was decided to use simple white noise from [-1, 1] instead.

4.4 FXive Implementation

The JSAP plugin format was used to allow efficient incorporation of the model into the FXive website. To do this, it was necessary to incorporate a script that holds the plugin variable, called `frictionPlugin.js` for the friction model and `impactPlugin.js` for the impact model. This script contains the `scriptProcessor` node declaration, including its audio processing code, and it communicates with the external elements via audio inputs and outputs (in this case only outputs are necessary) and plugin parameters, used to control audio parameters. Moreover, it contains an object that stores the `Interactor` used by the engine. The `Interactor` contained in this object is the one manipulated in the `scriptProcessor` node's audio processing function, which sends it to the function that runs the model's DSP (see Figure 3).

In order to have control over the model, it was necessary to add GUI elements that respond in real time and are able to modify parameters simultaneously as they are being manipulated. This allows the user to control the synthesis engine seamlessly, as if it was a musical instrument. In the FXive website, this is done using the NexusUI API. This means, in essence, that these UI elements have been designed to face the specific interaction problems that arise when creating software musical instruments (which are mainly related to latency and instantaneous response and feedback). That makes them an appropriate choice to control these type of synthesis engines, that intend to generate realistic sound effects from real-time inputs. In this implementation, the NexusUI elements are added into the main HTML file that contains every script and the layout of the webpage. Also in this script are included the functions that are called when each UI object is modified. The code inside these functions is essentially the JSAP parameter changing function, which sets the value of a specific plugin parameter.

5 Results

After building the models, we needed to ensure they fitted the specifications and performed as their predecessors in the SDT implementation. This was done in two stages for the friction model, and only the first stage was done for the impact model.

Parameter	Description	Performance	Compared analysis
Stiffness	affects the evolution of mode lock-in	Good	Similar
Dissipation	affects the sound bandwidth	Good	Similar
Viscosity	affects the speed of timbre evolution and pitch	Good	Similar
Noisiness	affects the perceived surface roughness	Good	Similar
Dynamic Coeff.	high values reduce the sound bandwidth	Good	Similar
Static Coeff.	affects the smoothness of sound attack	Mediocre	Similar
Stribeck	affects the smoothness of sound attack	Mediocre	Similar
Weight	affects pitch	Not good	Not similar
Size	affects low frequency content	Good	Similar

Table 1: Friction model parameter evaluation results. Descriptions extracted from Rocchesso and Fontana [1].

5.1 Parameter evaluation

The first stage of the evaluation process was comparing the current implementation with the original. This was done by comparing how modifications on each parameter affected the sound in both implementations, and analysing whether they fitted the phenomenological descriptions (see [1] and Tables 1 and 2). A two step process was followed. First, one parameter was tested on the web model by trying different settings (including extreme values). The performance of each parameter was labeled as: good, if the effect of the parameter could be clearly heard and/or seen in the spectrum; mediocre, if the effect could not be clearly noticed, or only noticed at extreme values, and not good, if the effect was not heard or seen, or a different effect resulted. In the second step, the same settings were tried on the original implementation, to check whether they affected the sound in the same way. If the resulting sound was perceived as identical in both implementations, the parameter comparison was marked as similar; if there was a perceptual difference, it was marked as not similar. The resulting sounds on each step were examined using auditory analysis and visual inspection of the spectrograms. This process was repeated on each of the inertial resonator and interaction-specific parameters.

5.1.1 Friction model evaluation

In general, the web model performed similar to the original implementation, as can be seen in Table 1 (the phenomenological description for each parameter is described in Rocchesso and Fontana [1, p. 162]). Nevertheless, there are a couple of malfunctions that must be mentioned.

The first issue is that the sound tends to acquire a "tonal" texture, i.e., it sounds pitched. One possible explanation is that the normal force control parameter requires a higher degree of sensitivity. If this parameter is not sensitive enough, the model might not be changing pitch fast enough, and therefore develop an undesired tonal quality.

The second issue is that the inertial weight parameter (weight of the bow) does not work with high values. Again, this might be a problem of parameter sensitivity, caused by differences in the slider curve between the web implementation and the original. However, this is unlikely, because the model still presents differences at extreme values. Unfortunately, this parameter seems to work fine in the impact model, meaning that the issue must have to do with how the parameter relates to the interaction and, therefore, making it more difficult to debug.

5.1.2 Impact model evaluation

A shown in Table 2, the web implementation of the model seems to perform properly on almost every aspect. Nevertheless, the shape parameter presented some problems, since it does not completely behave as in the description, and it does not match the original implementation's behaviour. Finally, as mentioned before, the inertial weight (i.e., "hammer weight" parameter) performs as expected on this model, which means the issue in the friction model is specific of that implementation.

5.2 High level models

The second stage of the evaluation process was using the models to create presets that emulate specific interaction scenarios, and comparing the resulting effects

Parameter	Description	Performance	Compared analysis
Hammer weight	affects duration, brightness and loudness	Good	Similar
Dissipation	affects attack brightness	Good	Similar
Stiffness	affects attack brightness	Good	Similar
Shape	affects attack loudness	Mediocre	Not similar
Strike velocity	affects duration, brightness and loudness	Good	Similar
External force	generates hits	Good	Similar
Inertial size	affects attack loudness	Good	Similar
Modal Size	affects pitch	Good	Similar

Table 2: Impact model parameter evaluation results. Descriptions extracted from Rocchesso and Fontana [1].

with recorded samples using their spectrograms. Moreover, some informal user evaluations, done with three participants with no experience in sound design, were performed in order to obtain some qualitative information.

The tests were run by having each participant identify synthesized sounds and recorded sounds from a reduced number of samples. Each test was run separately (i.e., one participant at a time), in a silent room, with participants listening through headphones. The examiner played each sound once, and asked each participant to identify which one was real and which synthesized. Once they had responded, they were asked which cues had led them to their answers. After that, the examiner told the actual answer and asked for any comments. Samples for this evaluation stage were extracted from Freesound[5]. To select the samples, the target sound had to be recognizable in the recording, and heard with little or no background noise. This stage was only done on the friction model, developing two presets: a creaking door, and rubbing a glass of wine.

5.2.1 Creaking door

For the creaking door sound effect, three presets were designed: two of them were obtained by ear, and the third one was done by trying to match the spectrogram of a real sample. The spectrograms are showed in Figure 4.

The preset was designed trying to recreate the most prominent modes in the original spectrogram. In order to compare them, it should be advisable not to focus too much on the long-term time evolution of the signal, since the intention was not copying the sample, but the source that produced the sample. Consequently, the synthesized version was not controlled trying to imitate the behaviour of the original one.

The main difference is the harmonic content. It can be seen that, in Figure 4a, the signal is richer in harmonics, these present a smoother short-term evolution than in Figure 4b and the main components are less prominent. This could be due to many factors. First of all, the model only uses three modes, making them harmonically simpler than the recording. It might be possible to recreate some of that richness by using modes that modulate each other creating components within the audio range. Another reason why modes in the original spectrogram are a bit more diffuse is the presence of reverb and ambience in the recording, and the recording gear's internal noise.

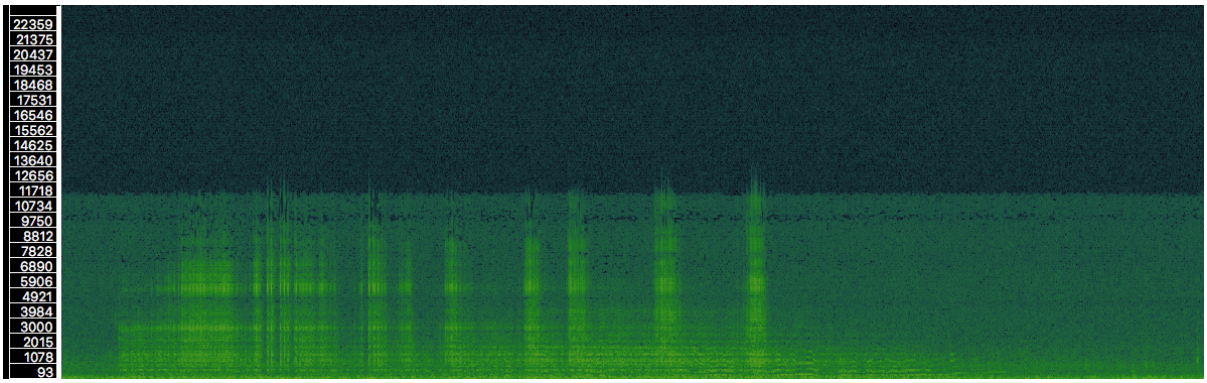
However, over a very short time span, both signals behave similarly: emitting continuous streaks of short impulses. This might be one of the main features that helps identify this sound, meaning that the synthesized model creates a "cartoonification" of the real sound.

When presented to users, participants were asked to identify from four sounds: three synthesized and the recording. Every participant considered the sounds coming from the presets configured by ear were the most realistic, while they stated that the sample was fake, and its imitation was the worst sounding of them. Therefore, these observations suggest that the resulting model fulfills its function of providing a simplified yet identifiable sound (i.e., "cartoonification").

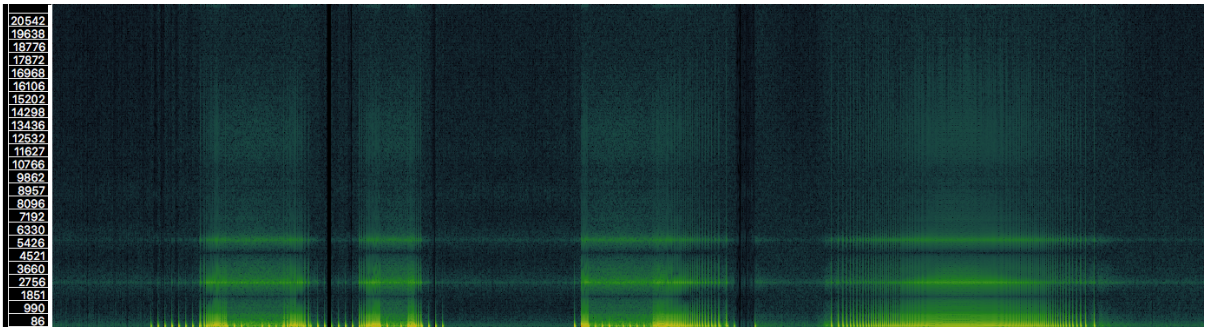
5.2.2 Wine glass rubbing

For this model only one preset was defined, using the imitative process followed with the creaking door. The sound that is pursued is the one obtained by rubbing the rim of a glass of wine with a moist finger. As it can be

[5] freesound.org/



(a) Original spectrogram



(b) Synthesized spectrogram

Fig. 4: Spectrograms of a creaking door recording (12s long) and a synthesized version (15s long)

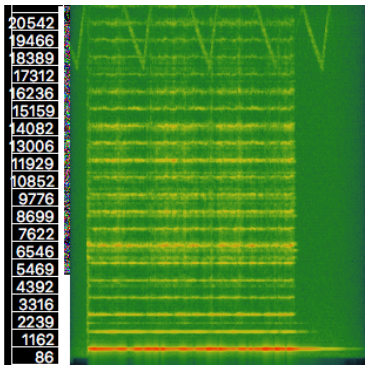
seen in Figure 5, several differences can be appreciated between the signals. Once again, spectral richness is lacking in the modeled sound compared to the real one. This time, this difference is even more significant than in the creaking door example, since harmonics in the recorded sound are clear. Nevertheless, the model still presents some frequency components that do not come directly from the resonator modes. An interesting feature of the recorded sample is the vibrato happening at high frequencies.

When this was shown to users, they stated that none of the two samples sounded more artificial or fake than the other (they said neither of them sounded particularly real), but that the one in Figure 5b seemed like a worse recorded sample, inducing them to think that was the modeled one. That might be due to the lack of high frequency content in the synthesized sound, that made it feel like a low quality recording. Therefore, it could be said that this preset did not perform as well as the creaking door. Consequently, even though stating that a good "cartoonification" has been achieved would be inappropriate, it can be considered a solid base to build a better model.

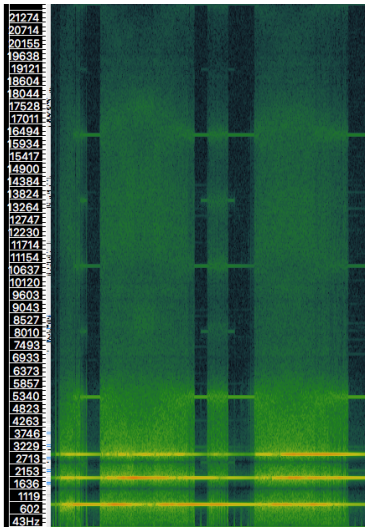
6 Summary

The development process of two sound effect synthesis engines for the web has been presented. These two models, extracted from the Sound Design Toolkit [6], recreate the sound produced by two types of interaction between solids: friction and impact. The models are physically driven, allowing a sound generation process that moves away from more classical signal based-methods. At the same time, it provides real-time control, enabling the user to naturally interact with the interface, that gives instantaneous sound feedback. The destination of the implementations of these two models is the FXive webpage[6,7], where both models have been uploaded. In addition, a rolling model[8], implemented using the impact model, can also be found in the website.

[6]Friction: `fxive.com/app/main-panel/friction.html`
[7]Impact: `fxive.com/app/main-panel/Impact.html`
[8]Rolling: `fxive.com/app/main-panel/rolling.html`



(a) Original spectrogram



(b) Synthesized spectrogram

Fig. 5: Spectrograms of a rubbed glass of wine recording and a synthesized version (both 5s long)

Although the mechanics behind the model have been explained, this paper also focused on creation of a new implementation in a more limited environment. The issues that arose could be overcome, achieving a result that keeps the modular and versatile approach of the original implementation. As a consequence, two different models have been designed, sharing a significant portion of code. Therefore the main objective of building a convincing web implementation of these models was achieved.

References

[1] Rocchesso, D. and Fontana, F., *The sounding object*, Mondo estremo, 2003.

[2] Turchet, L. et al., "What do your footsteps sound like? An investigation on interactive footstep sounds adjustment," *Applied Acoustics*, 111, pp. 77–85, 2016.

[3] Selfridge, R. et al., "Creating Real-Time Aeroacoustic Sound Effects Using Physically Informed Models," *Journal of the Audio Engineering Society*, 2018.

[4] Peng, H. and Reiss, J. D., "Why Can You Hear a Difference between Pouring Hot and Cold Water? An Investigation of Temperature Dependence in Psychoacoustics," in *Audio Engineering Society Convention 145*, 2018.

[5] Farnell, A., *Designing sound*, Mit Press, 2010.

[6] Monache, S. D., Polotti, P., and Rocchesso, D., "A toolkit for explorations in sonic interaction design," in *Proceedings of the 5th audio mostly conference: a conference on interaction with sound*, p. 1, ACM, 2010.

[7] Bahadoran, P. et al., "FXive: A Web Platform for Procedural Sound Synthesis," in *Audio Engineering Society Convention 144*, 2018.

[8] Adenot, P., Wilson, C., and Rogers, C., "Web audio API, W3C working draft," Technical report, Dec 08 2015. Technical Report, W3C, 2015.

[9] Taylor, B. et al., "Simplified Expressive Mobile Development with NexusUI, NexusUp, and NexusDrop." in *NIME*, pp. 257–262, 2014.

[10] Jillings, N. et al., "JSAP: A Plugin Standard for the Web Audio API with Intelligent Functionality," in *Audio Engineering Society Convention 141*, 2016.

[11] Van Den Doel, K., Kry, P. G., and Pai, D. K., "FoleyAutomatic: physically-based sound effects for interactive simulation and animation," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 537–544, ACM, 2001.

[12] De Wit, C. et al., "A new model for control of systems with friction," *IEEE Transactions on automatic control*, 40(3), pp. 419–425, 1995.

[13] Olsson, H. et al., "Friction models and friction compensation," *Eur. J. Control*, 4(3), pp. 176–195, 1998.

[14] Dupont, P. et al., "Single state elastoplastic friction models," *IEEE Transactions on automatic control*, 47(5), pp. 787–792, 2002.

[15] Avanzini, F., Serafin, S., and Rocchesso, D., "Interactive simulation of rigid body interaction with friction-induced sound generation," *IEEE transactions on speech and audio processing*, 13(5), pp. 1073–1081, 2005.

[16] Hunt, K. H. and Crossley, F. R. E., "Coefficient of restitution interpreted as damping in vibroimpact," *Journal of applied mechanics*, 42(2), pp. 440–445, 1975.

[17] Marhefka, D. W. and Orin, D. E., "A compliant contact model with nonlinear damping for simulation of robotic systems," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 29(6), pp. 566–572, 1999.